
ASHPY Documentation

Release 0.4.0

ml@zuru.tech

Feb 07, 2020

Contents:

1	Welcome To AshPy!	1
1.1	AshPy	1
2	Write The Docs!	11
2.1	The Whys	11
2.2	Documentation Architecture	12
2.3	Additional Materials	14
3	Getting Started	15
3.1	Datasets	15
3.2	Models	16
3.3	Creating a new Trainer	20
3.4	Complete Examples	20
4	API Reference	31
4.1	ashpy.ashtypes	31
4.2	ashpy.callbacks	31
4.3	ashpy.contexts	71
4.4	ashpy.keras	91
4.5	ashpy.layers	99
4.6	ashpy.losses	105
4.7	ashpy.metrics	151
4.8	ashpy.models	198
4.9	ashpy.modes	309
4.10	ashpy.restorers	310
4.11	ashpy.trainers	326
5	AshPy Internals	363
5.1	Context	363
5.2	Executor	364
6	Dependencies Graph	367
6.1	ashpy.callbacks	367
6.2	ashpy.models	368
6.3	ashpy.trainers	368
6.4	ashpy.restorers	369
6.5	ashpy.layers	369

6.6	ashpy.losses	369
6.7	ashpy.metrics	370
7	About	373
8	Indices and tables	375
	Python Module Index	377
	Index	379

CHAPTER 1

Welcome To AshPy!

Warning: AshPy is still a work in progress and may change substantially before the first proper release. The API is not mature enough to be considered stable, but we'll try to keep breaking changes to a minimum.

1.1 AshPy

AshPy is a TensorFlow 2.1 library for (**distributed**) training, evaluation, model selection, and fast prototyping. It is designed to ease the burden of setting up all the nuances of the architectures built to train complex custom deep learning models.

Quick Example | Features | Set Up | Usage | Dataset Output Format | Test

1.1.1 Quick Example

```
# define a distribution strategy
strategy = tf.distribute.MirroredStrategy()

# work inside the scope of the created strategy
with strategy.scope():

    # get the MNIST dataset
    train, validation = tf.keras.datasets.mnist.load_data()

    # process data if needed
    def process(images, labels):
        data_images = tf.data.Dataset.from_tensor_slices((images)).map(
            lambda x: tf.reshape(x, (28 * 28,))
        )
        data_images = data_images.map(
            lambda x: tf.image.convert_image_dtype(x, tf.float32)
        )
        data_labels = tf.data.Dataset.from_tensor_slices((labels))
        dataset = tf.data.Dataset.zip((data_images, data_labels))
        dataset = dataset.batch(1024 * 1)
        return dataset

    # apply the process function to the data
    train, validation = (
        process(train[0], train[1]),
        process(validation[0], validation[1]),
    )

    # create the model
    model = tf.keras.Sequential(
        [
            tf.keras.layers.Dense(10, activation=tf.nn.sigmoid),
            tf.keras.layers.Dense(10),
        ]
    )

    # define the optimizer
    optimizer = tf.optimizers.Adam(1e-3)

    # the loss is provided by the AshPy library
    loss = ClassifierLoss(tf.losses.SparseCategoricalCrossentropy(from_logits=True))
    logdir = "testlog"
    epochs = 10

    # the metrics are provided by the AshPy library
    # and every metric with model_selection_operator != None performs
    # model selection, saving the best model in a different folder per metric.
    metrics = [
        ClassifierMetric(
            tf.metrics.Accuracy(), model_selection_operator=operator.gt
        ),
        ClassifierMetric(
            tf.metrics.BinaryAccuracy(), model_selection_operator=operator.gt
        ),
    ]

    # define the AshPy trainer
```

(continues on next page)

(continued from previous page)

```

trainer = ClassifierTrainer(
    model, optimizer, loss, epochs, metrics, logdir=logdir
)

# run the training process
trainer(train, validation)

```

1.1.2 Features

AshPy is a library designed to ease the burden of setting up all the nuances of the architectures built to train complex custom deep learning models. It provides both fully convolutional and fully connected models such as:

- autoencoder
- decoder
- encoder

and a fully convolutional:

- unet

Moreover, it provides already prepared trainers for a classifier model and GAN networks. In particular, in regards of the latter, it offers a basic GAN architecture with a Generator-Discriminator structure and an enhanced GAN architecture version made up of a Encoder-Generator-Discriminator structure.

AshPy it is developed around the concepts of *Executor*, *Context*, *Metric*, and *Strategies* that represents its foundations.

Executor An Executor is a class that helps to better generalize a training loop. With an Executor you can construct, for example, a custom loss function and put whatever computation you need inside it. You should define a `call` function inside your class and decorate it with `@Executor.reduce` header. Inside the `call` function you can take advantage of a context.

Context A Context is a useful class in which all the models, metrics, dataset and mode of your network are set. Passing the context around means that you can any time access to all what you need in order to performs any type of computation.

Metric A Metric is a class from which you can inherit to create your custom metric that can automatically keep track of the best performance of the model during training and, automatically save the best one doing what is called the *model selection*.

Strategies If you want to distribute your training across multiple GPUs, there is the `tf.distribute.Strategy` TensorFlow API with which you can distribute your models and training code with minimal code changes. AshPy implements this type of strategies internally and will check everything for you to apply the distribution strategy correctly. All you need to do is as simple as doing the following:

```

strategy = tf.distribute.MirroredStrategy()
with strategy.scope():

    generator = ConvGenerator(
        layer_spec_input_res=(7, 7),
        layer_spec_target_res=(28, 28),
        kernel_size=(5, 5),
        initial_filters=256,
        filters_cap=16,
        channels=1,

```

(continues on next page)

(continued from previous page)

```
)
# rest of the code
# with trainer definition and so on
```

i.e., create the strategy and put the rest of the code inside its scope.

In general AshPy aims to:

- Rapid model prototyping
- Enforcement of best practices & API consistency
- Remove duplicated and boilerplate code
- General usability by new project

NOTE: We invite you to read the full documentation on [the official website](#).

The following README aims to help you understand what you need to do to setup AshPy on your system and, with some examples, what you need to do to setup a complete training of your network. Moreover, it will explain some fundamental modules you need to understand to fully exploit the potential of the library.

1.1.3 Set up

Pip install

```
pip install ashpy
```

Source install

Clone this repo, go inside the downloaded folder and install with:

```
pip install -e .
```

1.1.4 Usage

Let's quickly start with some examples.

Classifier

Let's say we want to train a classifier.

```
import operator
import tensorflow as tf
from ashpy.metrics import ClassifierMetric
from ashpy.trainers.classifier import ClassifierTrainer
from ashpy.losses.classifier import ClassifierLoss

def toy_dataset():
    inputs = tf.expand_dims(tf.range(1, 1000.0), -1)
    labels = tf.expand_dims([1 if tf.equal(tf.math.mod(tf.squeeze(i), 2), 0) else 0,
    ↪ for i in inputs], -1)
    return tf.data.Dataset.from_tensor_slices((inputs, labels)).shuffle(10).batch(2)
```

(continues on next page)

(continued from previous page)

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, activation=tf.nn.sigmoid),
    tf.keras.layers.Dense(2)
])

optimizer = tf.optimizers.Adam(1e-3)
loss = ClassifierLoss(tf.losses.SparseCategoricalCrossentropy(from_logits=True))
logdir = "testlog"
epochs = 2

metrics = [
    ClassifierMetric(tf.metrics.Accuracy(), model_selection_operator=operator.gt),
    ClassifierMetric(tf.metrics.BinaryAccuracy(), model_selection_operator=operator.
    ↪gt),
]

trainer = ClassifierTrainer(model, optimizer, loss, epochs, metrics, logdir=logdir)

train, validation = toy_dataset(), toy_dataset()
trainer(train, validation)
```

Skipping the `toy_dataset()` function that creates a toy dataset, we'll give a look to the code step by step.

So, first of all we define a model and its optimizer. Here, the model is a very simple sequential Keras model defined as:

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, activation=tf.nn.sigmoid),
    tf.keras.layers.Dense(2)
])

optimizer = tf.optimizers.Adam(1e-3)
```

Then we define the loss:

```
loss = ClassifierLoss(tf.losses.SparseCategoricalCrossentropy(from_logits=True))
```

The `ClassifierLoss` loss defined above it is defined using an internal class called “Executor”. The `Executor` is a class that let you define, alongside with a desired loss, the function that you want to use to “evaluate” that loss with all the needed parameters.

This works in conjunction with the following line (we will speak about the “*metrics*” and the other few definition lines in a minute):

```
trainer = ClassifierTrainer(model, optimizer, loss, epochs, metrics, logdir=logdir)
```

where a `ClassifierTrainer` is an object designed to run a specific training procedure adjusted, in this case, for a classifier.

The arguments of this function are the model, the optimizer, the loss, the number of epochs, the metrics and the logdir. We have already seen the definition of the model, the optimizer and of the loss. The definition of epochs, metrics and logdir happens here:

```
logdir = "testlog"
epochs = 2
```

(continues on next page)

(continued from previous page)

```
metrics = [
    ClassifierMetric(tf.metrics.Accuracy(), model_selection_operator=operator.gt),
    ClassifierMetric(
        tf.metrics.BinaryAccuracy(), model_selection_operator=operator.gt),
]
```

What we need to underline here is the definition of the metrics because as you can see they are defined through the use of specific classes: `ClassifierMetric`. As for the `ClassifierTrainer`, the `ClassifierMetric` it is a specified designed class for the Classifier. If you want to create a different metric you should inheriting from the `Metric` class provided by the Ash library. This kind of Metrics are useful because you can indicate a processing function to apply on predictions (e.g., `tf.argmax`) and an operator (e.g., `operator.gt` is the “greater than” operator) if you desire to activate the model selection during the training process based on that particular metric.

Finally, once the datasets has been set, you can start the training procedure calling the trainer object:

```
train, validation = toy_dataset(), toy_dataset()
trainer(train, validation)
```

1.1.5 GAN - Generative Adversarial Network

AshPy is equipped with two types of GAN network architectures:

- A plain GAN network with the classic structure Generator - Discriminator.
- A more elaborated GAN network architecture with the classic Generator - Discriminator structure plus an Encoder model (BiGAN like).

As for the previous classifier training example, let’s see for first a simple example of an entire “toy” code, regarding a simple plain GAN. At the end we will briefly touch upon the differences with the GAN network with the Encoder.

```
import operator
import tensorflow as tf
from ashpy.models.gans import ConvGenerator, ConvDiscriminator
from ashpy.metrics import InceptionScore
from ashpy.losses.gan import DiscriminatorMinMax, GeneratorBCE

generator = ConvGenerator(
    layer_spec_input_res=(7, 7),
    layer_spec_target_res=(28, 28),
    kernel_size=(5, 5),
    initial_filters=32,
    filters_cap=16,
    channels=1,
)

discriminator = ConvDiscriminator(
    layer_spec_input_res=(28, 28),
    layer_spec_target_res=(7, 7),
    kernel_size=(5, 5),
    initial_filters=16,
    filters_cap=32,
    output_shape=1,
)

# Losses
generator_bce = GeneratorBCE()
```

(continues on next page)

(continued from previous page)

```

minmax = DiscriminatorMinMax()

# Real data
batch_size = 2
mnist_x, mnist_y = tf.zeros((100,28,28)), tf.zeros((100,))

# Trainer
epochs = 2
logdir = "testlog/adversarial"

metrics = [
    InceptionScore(
        # Fake inception model
        ConvDiscriminator(
            layer_spec_input_res=(299, 299),
            layer_spec_target_res=(7, 7),
            kernel_size=(5, 5),
            initial_filters=16,
            filters_cap=32,
            output_shape=10,
        ),
        model_selection_operator=operator.gt,
        logdir=logdir,
    )
]

trainer = AdversarialTrainer(
    generator,
    discriminator,
    tf.optimizers.Adam(1e-4),
    tf.optimizers.Adam(1e-4),
    generator_bce,
    minmax,
    epochs,
    metrics,
    logdir,
)

# Dataset
noise_dataset = tf.data.Dataset.from_tensors(0).repeat().map(
    lambda _: tf.random.normal(shape=(100,), dtype=tf.float32, mean=0.0, stddev=1)
).batch(batch_size).prefetch(1)

# take only 2 samples to speed up tests
real_data = tf.data.Dataset.from_tensor_slices(
    (tf.expand_dims(mnist_x, -1), tf.expand_dims(mnist_y, -1))
).take(batch_size).batch(batch_size).prefetch(1)

# Add noise in the same dataset, just by mapping.
# The return type of the dataset must be: tuple(tuple(a,b), noise)
dataset = real_data.map(lambda x, y: ((x, y), tf.random.normal(shape=(batch_size, 1)
↪100)))

trainer(dataset)

```

First we define the generator and discriminator of the GAN architecture:

```
generator = ConvGenerator(
    layer_spec_input_res=(7, 7),
    layer_spec_target_res=(28, 28),
    kernel_size=(5, 5),
    initial_filters=32,
    filters_cap=16,
    channels=1,
)

discriminator = ConvDiscriminator(
    layer_spec_input_res=(28, 28),
    layer_spec_target_res=(7, 7),
    kernel_size=(5, 5),
    initial_filters=16,
    filters_cap=32,
    output_shape=1,
)
```

and then we define the losses:

```
# Losses
generator_bce = GeneratorBCE()
minmax = DiscriminatorMinMax()
```

where `GeneratorBCE()` and `DiscriminatorMinMax()` are the losses defined inheriting `Executor`. Again, as we have seen in the previous classifier example, you can customize this type (the ones inheriting from the `Executor`) of losses.

The metrics are defined as follow:

```
metrics = [
    InceptionScore(
        # Fake inception model
        ConvDiscriminator(
            layer_spec_input_res=(299, 299),
            layer_spec_target_res=(7, 7),
            kernel_size=(5, 5),
            initial_filters=16,
            filters_cap=32,
            output_shape=10,
        ),
        model_selection_operator=operator.gt,
        logdir=logdir,
    )
]
```

and in particular here we have the `InceptionScore` metric constructed on the fly with the `ConvDiscriminator` class provided by `AshPy`.

Finally, the actual trainer is constructed and then called:

```
trainer = AdversarialTrainer(
    generator,
    discriminator,
    tf.optimizers.Adam(1e-4),
    tf.optimizers.Adam(1e-4),
    generator_bce,
    minmax,
```

(continues on next page)

(continued from previous page)

```

    epochs,
    metrics,
    logdir,
)

```

```

trainer(dataset)

```

The main difference with a GAN architecture with an Encoder is that we would have the encoder loss:

```

encoder_bce = EncoderBCE()

```

an encoder accuracy metric:

```

metrics = [EncodingAccuracy(classifier, model_selection_operator=operator.gt,
↪ logdir=logdir)]

```

and an `EncoderTrainer`:

```

trainer = EncoderTrainer(
    generator,
    discriminator,
    encoder,
    tf.optimizers.Adam(1e-4),
    tf.optimizers.Adam(1e-5),
    tf.optimizers.Adam(1e-6),
    generator_bce,
    minmax,
    encoder_bce,
    epochs,
    metrics=metrics,
    logdir=logdir,
)

```

Note that the `EncoderTrainer` indicates a trainer of a GAN network with an Encoder and not a trainer of an Encoder itself.

1.1.6 Dataset Output Format

In order to standardize the GAN training, AshPy requires the input dataset to be in a common format. In particular, the dataset return type must always be in the format showed below, where the first element of the tuple is the discriminator input, and the second is the generator input.

```

tuple(tuple(a,b), noise)

```

Where `a` is the input sample, `b` is the label/condition (if any, otherwise fill it with 0), and `noise` is the latent vector of input.

To train Pix2Pix-like architecture, that have no `noise` as `ConvGenerator` input, just return the values in the format `(tuple(a,b), b)` since the condition is the generator input.

1.1.7 Test

In order to run the tests (with the doctests), linting and docs generation simply use `tox`.

tox

CHAPTER 2

Write The Docs!

Ash being a project built with a **Documentation Driven** approach means that a solid, automated documentation procedure is a mandatory requirement.

The core components of our systems are:

- Sphinx for the documentation generation
- reStructuredText as the markup language
- Google Style docstrings for in-code documentation
- ‘**vale**’_ and **vale**-styles
- Automatic internal deployment via GitLab Pages CI/CD integration

This document goal is threefold:

1. Explaining the Documentation Architecture, the steps taken to automate it and defending such choices
2. Serve as a future reference for other projects
3. Act as an example for the Guide format and a demo of Sphinx + reST superpowers
4. **Convince you of the need to always be on the lookout for errors even in a perfect** system.

2.1 The Whys

2.1.1 Why Sphinx?

Sphinx is the most used documentation framework for Python, developed for the Standard library itself it’s now adopted by all the most known third party libraries. What makes Sphinx so great is the combination of extensibility via themes, extensions and what not, coupled with a plethora of builtin functionalities that make writing docs a breeze.:

An example from Sphinx Site:

- Output formats: HTML (including Windows HTML Help), LaTeX (for printable PDF versions), ePub, Texinfo, manual pages, plain text

- Extensive cross-references: semantic markup and automatic links for functions, classes, citations, glossary terms and similar pieces of information
- Hierarchical structure: easy definition of a document tree, with automatic links to siblings, parents and children
- Automatic indices: general index as well as a language-specific module indices
- Code handling: automatic highlighting using the Pygments highlighter
- Extensions: automatic testing of code snippets, inclusion of docstrings from Python modules (API docs), and more
- Contributed extensions: more than 50 extensions contributed by users in a second repository; most of them installable from PyPI

2.1.2 Why reST?

More than why reST, the real question is *Why not Markdown?*

While Markdown can be easier and slightly quicker to write, it does not offer the same level of fine grained control, necessary for an effort as complex as technical writing, without sacrificing portability.

Eric Holscher has an aptly named article: [Why You Shouldn't Use "Markdown" for Documentation](#), he is one of the greatest documentation advocate out there. Go and read his articles, they are beautiful.

2.1.3 Why Google Style for Docstrings?

Google Docstrings are to us the best way to organically combine code and documentation. Leveraging Napoleon, a Sphinx extension offering automatic documentation support for both Numpy and Google docstrings style, we can write easy to read docstrings and still be able to use `autodoc` and `autosummary` directives.

2.2 Documentation Architecture

2.2.1 Tutorials, Guides, Complex Examples

Any form of documentation which is not generated from the codebase should go here. Parent/Entry Point reStructuredText file, should be added in `docs/src` and then referenced in `index.rst`

2.2.2 API Reference

API reference contains the full API documentation automatically generated from our codebase. The only manual step required is adding the module you want to document to the `api.rst` located inside `docs/source`.

Automate all the docs!

Classes, Functions, Exceptions: Annotate them normally, they do not require anything else.

Autosummary & submodules with imports: A painful story

Exposing Python objects to their parent module by importing them in its `__init__.py` file, breaks the `autosummary` directives when combining it with the automatic generation of stub files. Currently there's no way of making `autosummary` aware of the imported objects thus if you desire to document that API piece you need to find a workaround.

Example

Suppose we have the following structure:

```
keras/
|---> __init__.py
|
|---> models.py
```

And that these two file contains respectively:

- `__init__.py`

```
from .models import Model

__ALL__ = ["Model"]
```

- `models.py`

```
class Model:
    pass
```

Calling the `autosummary` directive (with the `toctree` option) on `keras` will not generate stub files for `keras . Model` causing it to not show in the Table of Contents of our API reference.

To circumvent this limitation it is ideal to insert some manual labour into the `keras` docstring.

- `__init__.py`

```
"""
Documentation example.

.. rubric:: Classes

.. autosummary:: Classes
   :toctree: _autosummary
   :nosignatures:

   keras.Model

.. rubric:: Submodules

.. autosummary:: keras.models
   :toctree: _autosummary
   :nosignatures:
   :template: autosummary/submodule.rst

   keras.models
"""
from .models import Model
```

(continues on next page)

(continued from previous page)

```
__ALL__ = ["Model"]
```

This way `autosummary` will produce the proper API documentation. The same approach applies also when exposing functions, exceptions, and modules.

Note: used when annotating submodules.

2.2.3 Inheritance Diagrams

Inheritance Diagrams are drawn using `sphinx.ext.inheritance_diagram` and `sphinx.ext.graphviz`.

The `autosummary` template for classes has been modified in order to automatically generate an inheritance diagram just below the title.

An `Inheritances Diagrams` page is manually created in order to showcase all the diagrams in one single page. The page gives a quick overview of the relations between the classes of each module.

2.3 Additional Materials

- [Sphinx](#)
- [Google Python Style](#)
- [Google Developer Documentation Style Guide](#)
- [Napoleon - Example Google Style Python Docstrings](#)
- [Read the Docs Sphinx Theme](#)
- [Write the Docs](#)
- [reStructuredText Primer](#)
- [vale](#)
- [Eric Holscher](#)

3.1 Datasets

AshPy supports `tf.data.Dataset` format.

We highly encourage you to use [Tensorflow Datasets](#) to manage and use your datasets in an handy way.

```
pip install tfds-nightly
```

3.1.1 Classification

In order to create a dataset for classification:

```
import tensorflow_datasets as tfds

from ashpy.trainers import ClassifierTrainer

def extract_fn(example):
    return example["image"], example["label"]

def main():
    ds_train, ds_validation = tfds.load(name="mnist", split=["train", "validation"])

    # build the input pipeline
    ds_train = ds_train.batch(BATCH_SIZE).prefetch(1)
    ds_train = ds_train.map(extract_fn)

    # same for validation
    ...

    # define model, loss, optimizer
    ...
```

(continues on next page)

(continued from previous page)

```
# define the classifier trainer
trainer = ClassifierTrainer(model, optimizer, loss, epochs, metrics,
↪ logdir=logdir)

# train
trainer.train(ds_train, ds_validation)
```

3.1.2 GANs

In order to create a datasets for a (Conditional) GANs:

```
import tensorflow_datasets as tfds

from ashpy.trainers import AdversarialTrainer

def extract_fn(example):
    # the ashpy input must be (real, condition), condition
    return (example["image"], example["label"]), example["label"]

def main():
    ds_train = tfds.load(name="mnist", split="train")

    # build the input pipeline
    ds_train = ds_train.batch(BATCH_SIZE).prefetch(1)
    ds_train = ds_train.map(extract_fn)

    # define models, losses, optimizers
    ...

    # define the adversarial trainer
    trainer = AdversarialTrainer(generator,
        discriminator,
        generator_optimizer,
        discriminator_optimizer,
        generator_loss,
        discriminator_loss,
        epochs,
        metrics,
        logdir,
    )

    # train
    trainer.train(ds_train)
```

3.2 Models

AshPy supports [Keras](#) models as inputs. You can use an AshPy predefined model or you can implement your own model.

3.2.1 Using an AshPy model

```
import tensorflow_datasets as tfds

from ashpy.trainers import ClassifierTrainer
from ashpy.models import UNet

def main():

    # create the dataset and the input pipeline

    # define models, loss, optimizer
    model = UNet(
        input_res,
        min_res,
        kernel_size,
        initial_filters,
        filters_cap,
        channels,
        use_dropout_encoder,
        use_dropout_decoder,
        dropout_prob,
        use_attention,
    )

    # define the classifier trainer
    trainer = AdversarialTrainer(generator,
        discriminator,
        generator_optimizer,
        discriminator_optimizer,
        generator_loss,
        discriminator_loss,
        epochs,
        metrics,
        logdir,
    )

    # train
    trainer.train(ds_train)
```

3.2.2 Creating a Model

It's very easy to create a simple model, since AshPy's models are Keras' models.

```
from ashpy.layers import Attention, InstanceNormalization

def downsample(
    filters,
    apply_normalization=True,
    attention=False,
    activation=tf.keras.layers.LeakyReLU(alpha=0.2),
    size=3,
):
    initializer = tf.random_normal_initializer(0.0, 0.02)

    result = tf.keras.Sequential()
```

(continues on next page)

(continued from previous page)

```

result.add(
    tf.keras.layers.Conv2D(
        filters,
        size,
        strides=2,
        padding="same",
        kernel_initializer=initializer,
        use_bias=not apply_normalization,
    )
)

if apply_normalization:
    result.add(InstanceNormalization())

result.add(activation)

if attention:
    result.add(Attention(filters))

return result

def upsample(
    filters,
    apply_dropout=False,
    apply_normalization=True,
    attention=False,
    activation=tf.keras.layers.ReLU(),
    size=3,
):
    initializer = tf.random_normal_initializer(0.0, 0.02)

    result = tf.keras.Sequential()
    result.add(tf.keras.layers.UpSampling2D(size=(2, 2)))
    result.add(tf.keras.layers.ZeroPadding2D(padding=(1, 1)))

    result.add(
        tf.keras.layers.Conv2D(
            filters,
            size,
            strides=1,
            padding="valid",
            kernel_initializer=initializer,
            use_bias=False,
        )
    )

    if apply_dropout:
        result.add(tf.keras.layers.Dropout(0.5))

    if apply_normalization:
        result.add(Normalizer())

    result.add(activation)

    if attention:
        result.add(Attention(filters))

```

(continues on next page)

(continued from previous page)

```

    return result

def Generator(attention, output_channels=3):
    down_stack = [
        downsample(32, apply_normalization=False), # 256
        downsample(32), # 128
        downsample(64, attention=attention), # 64
        downsample(64), # 32
        downsample(64), # 16
        downsample(128), # 8
        downsample(128), # 4
        downsample(256), # 2
        downsample(512, apply_normalization=False), # 1
    ]

    up_stack = [
        upsample(256, apply_dropout=True), # 2
        upsample(128, apply_dropout=True), # 4
        upsample(128, apply_dropout=True), # 8
        upsample(64), # 16
        upsample(64), # 32
        upsample(64, attention=attention), # 64
        upsample(32), # 128
        upsample(32), # 256
        upsample(32), # 512
    ]

    inputs = tf.keras.layers.Input(shape=[None, None, 1])
    x = inputs

    # Downsampling through the model
    skips = []
    for down in down_stack:
        x = down(x)
        skips.append(x)

    skips = reversed(skips[:-1])

    # Upsampling and establishing the skip connections
    for up, skip in zip(up_stack, skips):
        x = up(x)
        x = tf.keras.layers.Concatenate()([x, skip])

    last = upsample(
        output_channels,
        activation=tf.keras.layers.Activation(tf.nn.tanh),
        apply_normalization=False,
    )

    x = last(x)

    return tf.keras.Model(inputs=inputs, outputs=x)

```

In this way we have created a new model to be used inside AshPy.

3.2.3 Inheriting from `ashpy.models.Conv2DInterface`

The third possibility you have to create a new model is to inherit from the `ashpy.models.convolutional.interfaces.Conv2DInterface`.

This class offers the basic methods to implement in a simple way a new model.

3.3 Creating a new Trainer

AshPy has different generics trainers. Trainers implement the basic training loop together with distribution strategy management and logging. By now the only distribution strategy handled is the `tf.distribute.MirroredStrategy`.

3.4 Complete Examples

3.4.1 Classifier

```
1  # Copyright 2019 Zuru Tech HK Limited. All Rights Reserved.
2  #
3  # Licensed under the Apache License, Version 2.0 (the "License");
4  # you may not use this file except in compliance with the License.
5  # You may obtain a copy of the License at
6  #
7  # http://www.apache.org/licenses/LICENSE-2.0
8  #
9  # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 """Example of Multi-GPU classifier trainer."""
16
17 import operator
18
19 import tensorflow as tf
20
21 from ashpy.losses import ClassifierLoss
22 from ashpy.metrics import ClassifierMetric
23 from ashpy.trainers import ClassifierTrainer
24
25
26 def main():
27     """
28     Train a multi-GPU classifier.
29
30     How to use ash to training_set a classifier, measure the
31     performance and perform model selection.
32     """
33     strategy = tf.distribute.MirroredStrategy()
34     with strategy.scope():
```

(continues on next page)

(continued from previous page)

```

35     training_set, validation_set = tf.keras.datasets.mnist.load_data()
36
37     def process(images, labels):
38         data_images = tf.data.Dataset.from_tensor_slices((images)).map(
39             lambda x: tf.reshape(x, (28 * 28,))
40         )
41         data_images = data_images.map(
42             lambda x: tf.image.convert_image_dtype(x, tf.float32)
43         )
44         data_labels = tf.data.Dataset.from_tensor_slices((labels))
45         dataset = tf.data.Dataset.zip((data_images, data_labels))
46         dataset = dataset.batch(1024 * 1)
47         return dataset
48
49     training_set, validation_set = (
50         process(training_set[0], training_set[1]),
51         process(validation_set[0], validation_set[1]),
52     )
53
54     model = tf.keras.Sequential(
55         [
56             tf.keras.layers.Dense(10, activation=tf.nn.sigmoid),
57             tf.keras.layers.Dense(10),
58         ]
59     )
60     optimizer = tf.optimizers.Adam(1e-3)
61     loss = ClassifierLoss(tf.losses.SparseCategoricalCrossentropy(from_
↪ logits=True))
62     logdir = "testlog"
63     epochs = 10
64
65     metrics = [
66         ClassifierMetric(
67             tf.metrics.Accuracy(), model_selection_operator=operator.gt
68         ),
69         ClassifierMetric(
70             tf.metrics.BinaryAccuracy(), model_selection_operator=operator.gt
71         ),
72     ]
73
74     trainer = ClassifierTrainer(
75         model=model,
76         optimizer=optimizer,
77         loss=loss,
78         epochs=epochs,
79         metrics=metrics,
80         logdir=logdir,
81     )
82     trainer(training_set, validation_set)
83
84
85 if __name__ == "__main__":
86     main()

```

3.4.2 GANs

BiGAN

```

1  # Copyright 2019 Zuru Tech HK Limited. All Rights Reserved.
2  #
3  # Licensed under the Apache License, Version 2.0 (the "License");
4  # you may not use this file except in compliance with the License.
5  # You may obtain a copy of the License at
6  #
7  # http://www.apache.org/licenses/LICENSE-2.0
8  #
9  # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 """Bigan dummy implementation."""
16
17 import operator
18
19 import tensorflow as tf
20 from tensorflow import keras
21
22 from ashy.py.losses import DiscriminatorMinMax, EncoderBCE, GeneratorBCE
23 from ashy.py.metrics import EncodingAccuracy
24 from ashy.py.trainers import EncoderTrainer
25
26
27 def main():
28     """Define the trainer and the models."""
29
30     def real_gen():
31         """Define generator of real values."""
32         for _ in tf.range(100):
33             yield ((10.0,), (0,))
34
35     num_classes = 1
36     latent_dim = 100
37
38     generator = keras.Sequential([keras.layers.Dense(1)])
39
40     left_input = tf.keras.layers.Input(shape=(1,))
41     left = tf.keras.layers.Dense(10, activation=tf.nn.elu)(left_input)
42
43     right_input = tf.keras.layers.Input(shape=(latent_dim,))
44     right = tf.keras.layers.Dense(10, activation=tf.nn.elu)(right_input)
45
46     net = tf.keras.layers.Concatenate()([left, right])
47     out = tf.keras.layers.Dense(1)(net)
48
49     discriminator = tf.keras.Model(inputs=[left_input, right_input], outputs=[out])
50
51     encoder = keras.Sequential([keras.layers.Dense(latent_dim)])
52     generator_bce = GeneratorBCE()
53     encoder_bce = EncoderBCE()
54     minmax = DiscriminatorMinMax()
55

```

(continues on next page)

(continued from previous page)

```

56     epochs = 100
57     logdir = "log/adversarial/encoder"
58
59     # Fake pre-trained classifier
60     classifier = tf.keras.Sequential(
61         [tf.keras.layers.Dense(10), tf.keras.layers.Dense(num_classes)]
62     )
63
64     metrics = [
65         EncodingAccuracy(
66             classifier, model_selection_operator=operator.gt, logdir=logdir
67         )
68     ]
69
70     trainer = EncoderTrainer(
71         generator=generator,
72         discriminator=discriminator,
73         encoder=encoder,
74         generator_optimizer=tf.optimizers.Adam(1e-4),
75         discriminator_optimizer=tf.optimizers.Adam(1e-5),
76         encoder_optimizer=tf.optimizers.Adam(1e-6),
77         generator_loss=generator_bce,
78         discriminator_loss=minmax,
79         encoder_loss=encoder_bce,
80         epochs=epochs,
81         metrics=metrics,
82         logdir=logdir,
83     )
84
85     batch_size = 10
86     discriminator_input = tf.data.Dataset.from_generator(
87         real_gen, (tf.float32, tf.int64), ((1), (1))
88     ).batch(batch_size)
89
90     dataset = discriminator_input.map(
91         lambda x, y: ((x, y), tf.random.normal(shape=(batch_size, latent_dim)))
92     )
93
94     trainer(dataset)
95
96
97 if __name__ == "__main__":
98     main()

```

MNIST

```

1  # Copyright 2019 Zuru Tech HK Limited. All Rights Reserved.
2  #
3  # Licensed under the Apache License, Version 2.0 (the "License");
4  # you may not use this file except in compliance with the License.
5  # You may obtain a copy of the License at
6  #
7  # http://www.apache.org/licenses/LICENSE-2.0
8  #
9  # Unless required by applicable law or agreed to in writing, software

```

(continues on next page)

(continued from previous page)

```

10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 """Adversarial trainer example."""
16
17 import tensorflow as tf
18 from tensorflow import keras # pylint: disable=no-name-in-module
19
20 from ashpy.losses import DiscriminatorMinMax, GeneratorBCE
21 from ashpy.metrics import InceptionScore
22 from ashpy.models.gans import ConvDiscriminator, ConvGenerator
23 from ashpy.trainers import AdversarialTrainer
24
25
26 def main():
27     """Adversarial trainer example."""
28     strategy = tf.distribute.MirroredStrategy()
29     with strategy.scope():
30
31         generator = ConvGenerator(
32             layer_spec_input_res=(7, 7),
33             layer_spec_target_res=(28, 28),
34             kernel_size=(5, 5),
35             initial_filters=256,
36             filters_cap=16,
37             channels=1,
38         )
39
40         discriminator = ConvDiscriminator(
41             layer_spec_input_res=(28, 28),
42             layer_spec_target_res=(7, 7),
43             kernel_size=(5, 5),
44             initial_filters=32,
45             filters_cap=128,
46             output_shape=1,
47         )
48
49         # Losses
50         generator_bce = GeneratorBCE()
51         minmax = DiscriminatorMinMax()
52
53         # Trainer
54         logdir = "log/adversarial"
55
56         # InceptionScore: keep commented until the issues
57         # https://github.com/tensorflow/tensorflow/issues/28599
58         # https://github.com/tensorflow/hub/issues/295
59         # Haven't been solved and merged into tf2
60
61         metrics = [
62             # InceptionScore(
63             #     InceptionScore.get_or_train_inception(
64             #         mnist_dataset,
65             #         "mnist",
66             #         num_classes=10,

```

(continues on next page)

(continued from previous page)

```

67         #         epochs=1,
68         #         fine_tuning=False,
69         #         logdir=logdir,
70         #     ),
71         #     model_selection_operator=operator.gt,
72         #     logdir=logdir,
73         # )
74     ]
75
76     epochs = 50
77     trainer = AdversarialTrainer(
78         generator=generator,
79         discriminator=discriminator,
80         generator_optimizer=tf.optimizers.Adam(1e-4),
81         discriminator_optimizer=tf.optimizers.Adam(1e-4),
82         generator_loss=generator_bce,
83         discriminator_loss=minmax,
84         epochs=epochs,
85         metrics=metrics,
86         logdir=logdir,
87     )
88
89     batch_size = 512
90
91     # Real data
92     mnist_x, mnist_y = keras.datasets.mnist.load_data()[0]
93
94     def iterator():
95         """Define an iterator in order to do not load in memory all the dataset."""
96
97         for image, label in zip(mnist_x, mnist_y):
98             yield tf.image.convert_image_dtype(
99                 tf.expand_dims(image, -1), tf.float32
100             ), tf.expand_dims(label, -1)
101
102     real_data = (
103         tf.data.Dataset.from_generator(
104             iterator, (tf.float32, tf.int64), ((28, 28, 1), (1,))
105         )
106         .batch(batch_size)
107         .prefetch(1)
108     )
109
110     # Add noise in the same dataset, just by mapping.
111     # The return type of the dataset must be: tuple(tuple(a,b), noise)
112     dataset = real_data.map(
113         lambda x, y: ((x, y), tf.random.normal(shape=(batch_size, 100)))
114     )
115
116     trainer(dataset)
117
118 if __name__ == "__main__":
119     main()

```

Facades (Pix2Pix)

```

1  # Copyright 2019 Zuru Tech HK Limited. All Rights Reserved.
2  #
3  # Licensed under the Apache License, Version 2.0 (the "License");
4  # you may not use this file except in compliance with the License.
5  # You may obtain a copy of the License at
6  #
7  # http://www.apache.org/licenses/LICENSE-2.0
8  #
9  # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14
15 """
16 Pix2Pix on Facades Datasets dummy implementation.
17
18 Input Pipeline taken from: https://www.tensorflow.org/beta/tutorials/generative/
19 ↪pix2pix
20 """
21 from pathlib import Path
22
23 import tensorflow as tf
24
25 from ashpy import LogEvalMode
26 from ashpy.losses.gan import (
27     AdversarialLossType,
28     Pix2PixLoss,
29     get_adversarial_loss_discriminator,
30 )
31 from ashpy.models.convolutional.discriminators import PatchDiscriminator
32 from ashpy.models.convolutional.unet import FUnet
33 from ashpy.trainers.gan import AdversarialTrainer
34
35 _URL = "https://people.eecs.berkeley.edu/~tinghuiz/projects/pix2pix/datasets/facades.
36 ↪tar.gz"
37
38 PATH_TO_ZIP = tf.keras.utils.get_file("facades.tar.gz", origin=_URL, extract=True)
39 PATH = Path(PATH_TO_ZIP).parent / "facades"
40
41 BUFFER_SIZE = 100
42 BATCH_SIZE = 1
43 IMG_WIDTH = 256
44 IMG_HEIGHT = 256
45
46 def load(image_file):
47     """Load the image from file path."""
48     image = tf.io.read_file(image_file)
49     image = tf.image.decode_jpeg(image)
50
51     width = tf.shape(image)[1]
52
53     width = width // 2
54     real_image = image[:, :width, :]

```

(continues on next page)

(continued from previous page)

```

54     input_image = image[:, width:, :]
55
56     input_image = tf.cast(input_image, tf.float32)
57     real_image = tf.cast(real_image, tf.float32)
58
59     return input_image, real_image
60
61
62 def resize(input_image, real_image, height, width):
63     """Resize input_image and real_image to height x width."""
64     input_image = tf.image.resize(
65         input_image, [height, width], method=tf.image.ResizeMethod.NEAREST_NEIGHBOR
66     )
67     real_image = tf.image.resize(
68         real_image, [height, width], method=tf.image.ResizeMethod.NEAREST_NEIGHBOR
69     )
70
71     return input_image, real_image
72
73
74 def random_crop(input_image, real_image):
75     """Random crop both input_image and real_image."""
76     stacked_image = tf.stack([input_image, real_image], axis=0)
77     cropped_image = tf.image.random_crop(
78         stacked_image, size=[2, IMG_HEIGHT, IMG_WIDTH, 3]
79     )
80
81     return cropped_image[0], cropped_image[1]
82
83
84 def normalize(input_image, real_image):
85     """Normalize images in [-1, 1]."""
86     input_image = (input_image / 127.5) - 1
87     real_image = (real_image / 127.5) - 1
88
89     return input_image, real_image
90
91
92 def load_image_train(image_file):
93     """Load and process the image_file to be ready for the training."""
94     input_image, real_image = load(image_file)
95     input_image, real_image = random_jitter(input_image, real_image)
96     input_image, real_image = normalize(input_image, real_image)
97
98     return input_image, real_image
99
100
101 @tf.function
102 def random_jitter(input_image, real_image):
103     """Apply random jitter to both input_image and real_image."""
104     # resizing to 286 x 286 x 3
105     input_image, real_image = resize(input_image, real_image, 286, 286)
106
107     # randomly cropping to 256 x 256 x 3
108     input_image, real_image = random_crop(input_image, real_image)
109
110     if tf.random.uniform(()) > 0.5:

```

(continues on next page)

(continued from previous page)

```

111     # random mirroring
112     input_image = tf.image.flip_left_right(input_image)
113     real_image = tf.image.flip_left_right(real_image)
114
115     return input_image, real_image
116
117
118 def main(
119     kernel_size=5,
120     learning_rate_d=2e-4,
121     learning_rate_g=2e-4,
122     g_input_res=IMG_WIDTH,
123     g_min_res=1,
124     g_initial_filters=64,
125     g_filters_cap=512,
126     use_dropout_encoder=False,
127     use_dropout_decoder=True,
128     d_target_res=32,
129     d_initial_filters=64,
130     d_filters_cap=512,
131     use_dropout_discriminator=False,
132     dataset_name="facades",
133     resolution=256,
134     epochs=100_000,
135     dropout_prob=0.3,
136     l1_loss_weight=100,
137     gan_loss_weight=1,
138     use_attention_d=False,
139     use_attention_g=False,
140     channels=3,
141     gan_loss_type=AdversarialLossType.LSGAN,
142 ):
143     """Define Trainer and models."""
144     generator = FUNet(
145         input_res=g_input_res,
146         min_res=g_min_res,
147         kernel_size=kernel_size,
148         initial_filters=g_initial_filters,
149         filters_cap=g_filters_cap,
150         channels=channels, # color_to_label_tensor.shape[0],
151         use_dropout_encoder=use_dropout_encoder,
152         use_dropout_decoder=use_dropout_decoder,
153         dropout_prob=dropout_prob,
154         use_attention=use_attention_g,
155     )
156     discriminator = PatchDiscriminator(
157         input_res=resolution,
158         min_res=d_target_res,
159         initial_filters=d_initial_filters,
160         kernel_size=kernel_size,
161         filters_cap=d_filters_cap,
162         use_dropout=use_dropout_discriminator,
163         dropout_prob=dropout_prob,
164         use_attention=use_attention_d,
165     )
166
167     discriminator_loss = get_adversarial_loss_discriminator(gan_loss_type)()

```

(continues on next page)

(continued from previous page)

```

168 generator_loss = Pix2PixLoss(
169     l1_loss_weight=l1_loss_weight,
170     adversarial_loss_weight=gan_loss_weight,
171     adversarial_loss_type=gan_loss_type,
172 )
173
174 metrics = []
175 logdir = Path("log") / dataset_name / "run2"
176
177 if not logdir.exists():
178     logdir.mkdir(parents=True)
179
180 trainer = AdversarialTrainer(
181     generator=generator,
182     discriminator=discriminator,
183     generator_optimizer=tf.optimizers.Adam(learning_rate_g, beta_1=0.5),
184     discriminator_optimizer=tf.optimizers.Adam(learning_rate_d, beta_1=0.5),
185     generator_loss=generator_loss,
186     discriminator_loss=discriminator_loss,
187     epochs=epochs,
188     metrics=metrics,
189     logdir=logdir,
190     log_eval_mode=LogEvalMode.TEST,
191 )
192
193 train_dataset = tf.data.Dataset.list_files(PATH + "train/*.jpg")
194 train_dataset = train_dataset.shuffle(BUFFER_SIZE)
195 train_dataset = train_dataset.map(load_image_train)
196 train_dataset = train_dataset.batch(BATCH_SIZE)
197
198 train_dataset = train_dataset.map(lambda x, y: ((y, x), x))
199
200 trainer(
201     # generator_input,
202     train_dataset
203 )
204
205
206 if __name__ == "__main__":
207     main()

```


<i>ashpy.ashtypes</i>	Custom Type-Aliases.
<i>ashpy.callbacks</i>	Callbacks in order to gain control over the training loop.
<i>ashpy.contexts</i>	Contexts help gaining an easier control over the model selection and testing process of the models.
<i>ashpy.keras</i>	Custom extensions of standard Keras components.
<i>ashpy.layers</i>	Collection of layers.
<i>ashpy.losses</i>	Collection of Losses.
<i>ashpy.metrics</i>	Collection of Metrics.
<i>ashpy.models</i>	Collection of Models.
<i>ashpy.modes</i>	Various modalities used to configure certain ash behaviours.
<i>ashpy.restorers</i>	Restorers allow for easy restoration of tracked objects from <code>tf.train.Checkpoint</code> .
<i>ashpy.trainers</i>	Trainers help reducing boilerplate code by bootstrapping models training.

4.1 ashpy.ashtypes

Custom Type-Aliases. Mostly here for brevity in documentation.

```
TScalar = typing.Union[int, float]
```

```
TWeight = typing.Union[TScalar, tf.Tensor, typing.Callable[... , TScalar]]
```

4.2 ashpy.callbacks

Callbacks in order to gain control over the training loop.

A callback is a set of functions to be called at given stages of the training procedure. You can use callbacks to implement logging, measure custom metrics or get insight about the training procedure. You can pass a list of callbacks

(derived from `ashpy.callbacks.callback.Callback`) (as the keyword argument `callbacks`) to the `.call()` method of the Trainer. The relevant methods of the callbacks will then be called at each stage of the training.

Order:

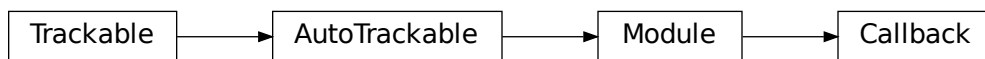
The basic class is `ashpy.callbacks.callback.Callback`. All possible events as listed as Enum inside `ashpy.callbacks.events.Event`.

Classes

<code>callback.Callback</code>	Callback definition.
<code>counter_callback.CounterCallback</code>	Count events of a specific type.
<code>classifier.LogClassifierCallback</code>	Callback used for logging Classifier images to Tensorboard.
<code>events.Event</code>	Define all possible events.
<code>gan.LogImageGANCallback</code>	Callback used for logging GANs images to Tensorboard.
<code>gan.LogImageGANEncoderCallback</code>	Callback used for logging GANs images to Tensorboard.
<code>save_callback.SaveCallback</code>	Save Callback implementation.
<code>save_callback.SaveFormat</code>	Save Format enum.
<code>save_callback.SaveSubFormat</code>	Save Sub-Format enum.

4.2.1 Callback

Inheritance Diagram



```

class ashpy.callbacks.callback.Callback(name)
    Bases: tensorflow.python.module.module.Module
  
```

Callback definition.

Every callback must extend from this class. This class defines the basic events. Every event takes as input the context in order to use the objects defined. Inheritance from `tf.Module` is required since callbacks have a state

Order: .. code-block:

```

--on_train_start

----on_epoch_start

-----on_batch_start
  
```

(continues on next page)

(continued from previous page)

```

-----on_batch_end

----on_epoch_end

--on_train_end

on_exception - if an Exception was raised

on_event - Called when an event is triggered

```

Methods

<code>__init__(name)</code>	Initialize the Callback.
<code>on_batch_end(context)</code>	Handle on_batch_end event.
<code>on_batch_start(context)</code>	Handle on_batch_start event.
<code>on_epoch_end(context)</code>	Handle on_epoch_end event.
<code>on_epoch_start(context)</code>	Handle on_epoch_start event.
<code>on_event(event, context)</code>	Handle the on_event event.
<code>on_exception(context)</code>	Handle on_exception event.
<code>on_train_end(context)</code>	Handle on_train_end event.
<code>on_train_start(context)</code>	Handle on_train_start event.

Attributes

<code>name</code>	Return the name of the callback.
<code>name_scope</code>	Returns a <code>tf.name_scope</code> instance for this class.
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>variables</code>	Sequence of variables owned by this module and it's submodules.

`__init__(name)`

Initialize the Callback.

Parameters `name` (*str*) – Callback name.

Warning: When using multiple callbacks with the same trainer make sure they have different ids.

Return type `None`

name

Return the name of the callback.

on_batch_end (*context*)

Handle on_batch_end event.

Method called at the end of a batch.

Parameters `context` (*ashpy.contexts.context.Context*) – training context

Return type None

on_batch_start (*context*)

Handle on_batch_start event.

Method called at the beginning of a batch.

Parameters `context` (*ashpy.contexts.context.Context*) – training context

Return type None

on_epoch_end (*context*)

Handle on_epoch_end event.

Method called at the end of an epoch.

Parameters `context` (*ashpy.contexts.context.Context*) – training context

Return type None

on_epoch_start (*context*)

Handle on_epoch_start event.

Method called at the beginning of an epoch.

Parameters `context` (*ashpy.contexts.context.Context*) – training context

Return type None

on_event (*event, context*)

Handle the on_event event.

Method called when an event is triggered.

Parameters

- **event** (*ashpy.callbacks.events.Event*) – triggered event
- **context** (*ashpy.contexts.context.Context*) – training context

Return type None

on_exception (*context*)

Handle on_exception event.

Method called when an exception is raised.

Parameters `context` (*ashpy.contexts.context.Context*) – training context

Return type None

on_train_end (*context*)

Handle on_train_end event.

Method called at the end of the training loop.

Parameters `context` (*ashpy.contexts.context.Context*) – training context

Return type None

on_train_start (*context*)

Handle on_train_start event.

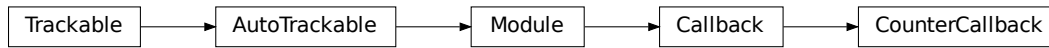
Method called at the beginning of the training loop.

Parameters `context` (*ashpy.contexts.context.Context*) – training context

Return type None

4.2.2 CounterCallback

Inheritance Diagram



class `ashpy.callbacks.counter_callback.CounterCallback` (*event*, *fn*, *name*, *event_freq=1*)

Bases: `ashpy.callbacks.callback.Callback`

Count events of a specific type. Calls *fn* passing the context every *event_freq*.

Useful for logging or for measuring performance. If you want to implement a callback defining a certain behaviour every *n_events* you can just inherit from `CounterCallback`.

Methods

<code>__init__</code> (<i>event</i> , <i>fn</i> , <i>name</i> [, <i>event_freq</i>])	Initialize the <code>CounterCallback</code> .
<code>on_event</code> (<i>event</i> , <i>context</i>)	Count events and calls <i>fn</i> .

Attributes

<code>name</code>	Return the name of the callback.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>variables</code>	Sequence of variables owned by this module and it's submodules.

`__init__` (*event*, *fn*, *name*, *event_freq=1*)
Initialize the `CounterCallback`.

Parameters

- **event** (`ashpy.events.Event`) – event to count.
- **fn** (`Callable`) – function to call every *event_freq* events.
- **event_freq** (*int*) – event frequency.
- **name** (*str*) – name of the Callback.

Raises `ValueError` – if *event_freq* is not valid.

Return type None

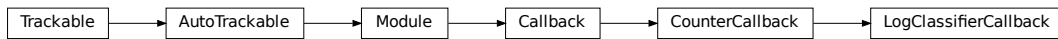
on_event (*event, context*)
Count events and calls fn.

Parameters

- **event** (*ashpy.callbacks.events.Event*) – current event.
- **context** (*ashpy.contexts.context.Context*) – current context.

4.2.3 LogClassifierCallback

Inheritance Diagram



```

class ashpy.callbacks.classifier.LogClassifierCallback (event=<Event.ON_EPOCH_END:
    'ON_EPOCH_END'>,
    name='log_classifier_callback',
    event_freq=1)
  
```

Bases: *ashpy.callbacks.counter_callback.CounterCallback*

Callback used for logging Classifier images to Tensorboard.

Logs the Input Image and true label.

Methods

<code>__init__</code> ([event, name, event_freq])	Initialize the LogClassifierCallback.
---	---------------------------------------

Attributes

<code>name</code>	Return the name of the callback.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>variables</code>	Sequence of variables owned by this module and it's submodules.

```

__init__ (event=<Event.ON_EPOCH_END: 'ON_EPOCH_END'>, name='log_classifier_callback',
    event_freq=1)
    Initialize the LogClassifierCallback.
  
```

Parameters

- **event** (*Event*) – event to consider
- **event_freq** (*int*) – frequency of logging

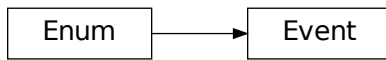
static **_log_fn** (*context*)
Log output of the image and label to Tensorboard.

Parameters **context** (*ClassifierContext*) – current context

Return type *None*

4.2.4 Event

Inheritance Diagram



class `ashpy.callbacks.events.Event`

Bases: `enum.Enum`

Define all possible events.

Attributes

<code>ON_BATCH_END</code>	On Batch End Event
<code>ON_BATCH_START</code>	On Batch Start Event
<code>ON_EPOCH_END</code>	On Epoch End Event
<code>ON_EPOCH_START</code>	On Epoch Start Event
<code>ON_EXCEPTION</code>	On Exception Event
<code>ON_TRAIN_END</code>	On Train End Event
<code>ON_TRAIN_START</code>	On Train Start Event

ON_BATCH_END = 'ON_BATCH_END'

On Batch End Event

ON_BATCH_START = 'ON_BATCH_START'

On Batch Start Event

ON_EPOCH_END = 'ON_EPOCH_END'

On Epoch End Event

ON_EPOCH_START = 'ON_EPOCH_START'

On Epoch Start Event

ON_EXCEPTION = 'ON_EXCEPTION'

On Exception Event

ON_TRAIN_END = 'ON_TRAIN_END'

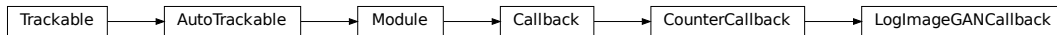
On Train End Event

ON_TRAIN_START = 'ON_TRAIN_START'

On Train Start Event

4.2.5 LogImageGANCallback

Inheritance Diagram



```

class ashpy.callbacks.gan.LogImageGANCallback(event=<Event.ON_EPOCH_END:
                                             'ON_EPOCH_END'>,
                                             name='log_image_gan_callback',
                                             event_freq=1)

```

Bases: *ashpy.callbacks.counter_callback.CounterCallback*

Callback used for logging GANs images to Tensorboard.

Logs the Generator output. Logs G(z).

Examples

```

import shutil
import operator
from pathlib import Path

generator = models.gans.ConvGenerator(
    layer_spec_input_res=(7, 7),
    layer_spec_target_res=(28, 28),
    kernel_size=(5, 5),
    initial_filters=32,
    filters_cap=16,
    channels=1,
)

discriminator = models.gans.ConvDiscriminator(
    layer_spec_input_res=(28, 28),
    layer_spec_target_res=(7, 7),
    kernel_size=(5, 5),
    initial_filters=16,
    filters_cap=32,
    output_shape=1,
)

# Losses

```

(continues on next page)

(continued from previous page)

```

generator_bce = losses.gan.GeneratorBCE()
minmax = losses.gan.DiscriminatorMinMax()

# Real data
batch_size = 2
mnist_x, mnist_y = tf.zeros((100,28,28)), tf.zeros((100,))

# Trainer
epochs = 2
logdir = Path("testlog/callbacks")
callbacks = [callbacks.LogImageGANCallback()]
trainer = trainers.gan.AdversarialTrainer(
    generator=generator,
    discriminator=discriminator,
    generator_optimizer=tf.optimizers.Adam(1e-4),
    discriminator_optimizer=tf.optimizers.Adam(1e-4),
    generator_loss=generator_bce,
    discriminator_loss=minmax,
    epochs=epochs,
    callbacks=callbacks,
    logdir=logdir,
)

# take only 2 samples to speed up tests
real_data = (
    tf.data.Dataset.from_tensor_slices(
        (tf.expand_dims(mnist_x, -1), tf.expand_dims(mnist_y, -1))).take(batch_size)
    .batch(batch_size)
    .prefetch(1)
)

# Add noise in the same dataset, just by mapping.
# The return type of the dataset must be: tuple(tuple(a,b), noise)
dataset = real_data.map(
    lambda x, y: ((x, y), tf.random.normal(shape=(batch_size, 100)))
)

trainer(dataset)
shutil.rmtree(logdir)

assert not logdir.exists()

trainer._global_step.assign_add(500)

```

```

Initializing checkpoint.
Starting epoch 1.
[1] Saved checkpoint: testlog/callbacks/ckpts/ckpt-1
Epoch 1 completed.
Starting epoch 2.
[2] Saved checkpoint: testlog/callbacks/ckpts/ckpt-2
Epoch 2 completed.
Training finished after 2 epochs.

```

Methods

<code>__init__</code> ([event, name, event_freq])	Initialize the LogImageCallbackGAN.
---	-------------------------------------

Attributes

<code>name</code>	Return the name of the callback.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>variables</code>	Sequence of variables owned by this module and it's submodules.

`__init__` (event=<Event.ON_EPOCH_END:
 name='log_image_gan_callback', event_freq=1)
 Initialize the LogImageCallbackGAN.

Parameters

- **event** (*ashpy.callbacks.events.Event*) – event to consider.
- **event_freq** (*int*) – frequency of logging.
- **name** (*str*) – name of the callback.

Return type None

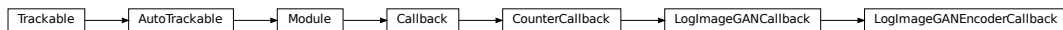
`_log_fn` (context)
 Log output of the generator to Tensorboard.

Parameters **context** (*ashpy.contexts.gan.GANContext*) – current context.

Return type None

4.2.6 LogImageGANEncoderCallback

Inheritance Diagram



```

class ashpy.callbacks.gan.LogImageGANEncoderCallback (event=<Event.ON_EPOCH_END:
                                                    'ON_EPOCH_END'>,
                                                    name='log_image_gan_encoder_callback',
                                                    event_freq=1)

```

Bases: *ashpy.callbacks.gan.LogImageGANCallback*

Callback used for logging GANs images to Tensorboard.

Logs the Generator output evaluated in the encoder output. Logs $G(E(x))$.

Examples

```
import shutil
import operator

def real_gen():
    label = 0
    for _ in tf.range(100):
        yield ((10.0,), (label,))

latent_dim = 100

generator = tf.keras.Sequential([tf.keras.layers.Dense(1)])

left_input = tf.keras.layers.Input(shape=(1,))
left = tf.keras.layers.Dense(10, activation=tf.nn.elu)(left_input)

right_input = tf.keras.layers.Input(shape=(latent_dim,))
right = tf.keras.layers.Dense(10, activation=tf.nn.elu)(right_input)

net = tf.keras.layers.Concatenate()([left, right])
out = tf.keras.layers.Dense(1)(net)

discriminator = tf.keras.Model(inputs=[left_input, right_input], outputs=[out])

encoder = tf.keras.Sequential([tf.keras.layers.Dense(latent_dim)])

# Losses
generator_bce = losses.gan.GeneratorBCE()
encoder_bce = losses.gan.EncoderBCE()
minmax = losses.gan.DiscriminatorMinMax()

epochs = 2

callbacks = [callbacks.LogImageGANEncoderCallback()]

logdir = "testlog/callbacks_encoder"

trainer = trainers.gan.EncoderTrainer(
    generator=generator,
    discriminator=discriminator,
    encoder=encoder,
    discriminator_optimizer=tf.optimizers.Adam(1e-4),
    generator_optimizer=tf.optimizers.Adam(1e-5),
    encoder_optimizer=tf.optimizers.Adam(1e-6),
    generator_loss=generator_bce,
    discriminator_loss=minmax,
    encoder_loss=encoder_bce,
    epochs=epochs,
    callbacks=callbacks,
    logdir=logdir,
)

batch_size = 10
discriminator_input = tf.data.Dataset.from_generator(
    real_gen, (tf.float32, tf.int64), ((1), (1))
).batch(batch_size)
```

(continues on next page)

(continued from previous page)

```
dataset = discriminator_input.map(
    lambda x, y: ((x, y), tf.random.normal(shape=(batch_size, latent_dim)))
)

trainer(dataset)

shutil.rmtree(logdir)
```

```
Initializing checkpoint.
Starting epoch 1.
[10] Saved checkpoint: testlog/callbacks_encoder/ckpts/ckpt-1
Epoch 1 completed.
Starting epoch 2.
[20] Saved checkpoint: testlog/callbacks_encoder/ckpts/ckpt-2
Epoch 2 completed.
Training finished after 2 epochs.
```

Methods

<code>__init__</code> ([event, name, event_freq])	Initialize the LogImageGANEncoderCallback.
---	--

Attributes

<code>name</code>	Return the name of the callback.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>variables</code>	Sequence of variables owned by this module and it's submodules.

```
__init__(event=<Event.ON_EPOCH_END: 'ON_EPOCH_END'>,
         name='log_image_gan_encoder_callback', event_freq=1)
Initialize the LogImageGANEncoderCallback.
```

Parameters

- **event** (*ashpy.callbacks.events.Event*) – event to consider.
- **event_freq** (*int*) – frequency of logging.
- **name** (*str*) – name of the callback.

Return type

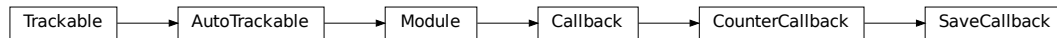
None

```
_log_fn(context)
Log output of the generator to Tensorboard.
Logs G(E(x)).
```

Parameters **context** (*ashpy.contexts.gan.GanEncoderContext*) – current context.

4.2.7 SaveCallback

Inheritance Diagram



```

class ashpy.callbacks.save_callback.SaveCallback (save_dir,          models,
                                                  event=<Event.ON_EPOCH_END:
                                                  'ON_EPOCH_END'>,
                                                  event_freq=1,      max_to_keep=1,
                                                  save_format=<SaveFormat.MODEL\WEIGHTS:
                                                  3>, save_sub_format=<SaveSubFormat.TF:
                                                  'tf'>,              verbose=0,
                                                  name='SaveCallback')

```

Bases: *ashpy.callbacks.counter_callback.CounterCallback*

Save Callback implementation.

Examples

```

import shutil
import operator
import os

generator = models.gans.ConvGenerator(
    layer_spec_input_res=(7, 7),
    layer_spec_target_res=(28, 28),
    kernel_size=(5, 5),
    initial_filters=32,
    filters_cap=16,
    channels=1,
)

discriminator = models.gans.ConvDiscriminator(
    layer_spec_input_res=(28, 28),
    layer_spec_target_res=(7, 7),
    kernel_size=(5, 5),
    initial_filters=16,
    filters_cap=32,
    output_shape=1,
)

models = [generator, discriminator]

save_callback = callbacks.SaveCallback(save_dir="testlog/savedir",
                                       models=models,
                                       save_format=callbacks.SaveFormat.WEIGHTS,
                                       save_sub_format=callbacks.SaveSubFormat.TF)

```

(continues on next page)

(continued from previous page)

```
# initialize trainer passing the save_callback
```

Methods

<code>__init__(save_dir, models[, event, ...])</code>	Build a Save Callback.
<code>save_weights_fn(context)</code>	Save weights and clean up if needed.

Attributes

<code>name</code>	Return the name of the callback.
<code>name_scope</code>	Returns a <code>tf.name_scope</code> instance for this class.
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>variables</code>	Sequence of variables owned by this module and it's submodules.

```
__init__(save_dir, models, event=<Event.ON_EPOCH_END: 'ON_EPOCH_END'>,
         event_freq=1, max_to_keep=1, save_format=<SaveFormat.MODEL|WEIGHTS: 3>,
         save_sub_format=<SaveSubFormat.TF: 'tf'>, verbose=0, name='SaveCallback')
```

Build a Save Callback.

Save Callbacks are used to save the model on events. You can specify two different save formats: weights and model. At the same time you can specify two different save sub-formats: tf or h5. You will find the model saved in the `save_dir` under the directory weights or model.

Parameters

- **save_dir** (*str*) – directory in which to save the weights or the model.
- **models** (`List[tf.keras.models.Model]`) – list of models to save.
- **event** (`ashpy.callbacks.events.Event`) – events on which to trigger the saving operation.
- **event_freq** (*int*) – frequency of saving operation.
- **name** (*str*) – name of the callback.
- **verbose** (*int*) – verbosity of the callback (0 or 1).
- **max_to_keep** (*int*) – maximum files to keep. If `max_to_keep == 1` only the most recent file is kept. In general `max_to_keep` files are kept.
- **save_format** (`ashpy.callbacks.save_callback.SaveFormat`) – weights or model.
- **save_sub_format** (`ashpy.callbacks.save_callback.SaveSubFormat`) – sub-format of the saving (tf or h5).

```
_cleanup()
```

Cleanup stuff.

```
_save_weights_fn(step)
```

Save weights.

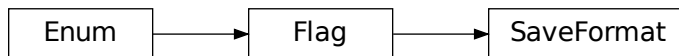
Parameters **step** (*int*) – current step.

save_weights_fn (*context*)

Save weights and clean up if needed.

4.2.8 SaveFormat

Inheritance Diagram



class `ashpy.callbacks.save_callback.SaveFormat`

Bases: `enum.Flag`

Save Format enum.

Attributes

<i>MODEL</i>	Model format (weights and architecture), saved using <i>model.save()</i>
<i>WEIGHTS</i>	Weights format, saved using <i>model.save_weights()</i>

MODEL = 2

Model format (weights and architecture), saved using *model.save()*

WEIGHTS = 1

Weights format, saved using *model.save_weights()*

name ()

Name of the format.

Return type `str`

save (*model*, *save_dir*, *save_sub_format*=<SaveSubFormat.TF: 'tf'>)

Save the model using the correct format and sub-format.

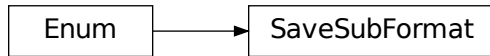
Parameters

- **model** (`tf.keras.models.Model`) – model to Save.
- **save_dir** (`pathlib.Path`) – path of the file in which to save the model.
- **save_sub_format** (`ashpy.callbacks.save_callback.SaveSubFormat`) – sub-format of the save operation.

Return type `None`

4.2.9 SaveSubFormat

Inheritance Diagram



class `ashpy.callbacks.save_callback.SaveSubFormat`

Bases: `enum.Enum`

Save Sub-Format enum.

Attributes

<i>H5</i>	H5 Format
<i>TF</i>	TensorFlow format

H5 = 'h5'
H5 Format

TF = 'tf'
TensorFlow format

Modules

<i>callback</i>	Callback definition.
<i>classifier</i>	Classifier callbacks.
<i>counter_callback</i>	Counter Callback implementation.
<i>events</i>	Event definition as Enum.
<i>gan</i>	GAN callbacks.
<i>save_callback</i>	Save weights callback.

4.2.10 callback

Callback definition.

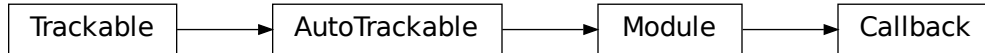
Classes

Callback

Callback definition.

Callback

Inheritance Diagram



class `ashpy.callbacks.callback.Callback` (*name*)
 Bases: `tensorflow.python.module.module.Module`

Callback definition.

Every callback must extend from this class. This class defines the basic events. Every event takes as input the context in order to use the objects defined. Inheritance from `tf.Module` is required since callbacks have a state

Order: .. code-block:

```

--on_train_start

----on_epoch_start

-----on_batch_start

-----on_batch_end

----on_epoch_end

--on_train_end

on_exception - if an Exception was raised

on_event - Called when an event is triggered
  
```

Methods

<code>__init__(name)</code>	Initialize the Callback.
<code>on_batch_end(context)</code>	Handle on_batch_end event.
<code>on_batch_start(context)</code>	Handle on_batch_start event.
<code>on_epoch_end(context)</code>	Handle on_epoch_end event.
<code>on_epoch_start(context)</code>	Handle on_epoch_start event.
<code>on_event(event, context)</code>	Handle the on_event event.
<code>on_exception(context)</code>	Handle on_exception event.

Continued on next page

Table 20 – continued from previous page

<code>on_train_end(context)</code>	Handle on_train_end event.
<code>on_train_start(context)</code>	Handle on_train_start event.

Attributes

<code>name</code>	Return the name of the callback.
<code>name_scope</code>	Returns a <code>tf.name_scope</code> instance for this class.
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>variables</code>	Sequence of variables owned by this module and it's submodules.

`__init__(name)`
Initialize the Callback.

Parameters `name` (*str*) – Callback name.

Warning: When using multiple callbacks with the same trainer make sure they have different ids.

Return type `None`

name
Return the name of the callback.

on_batch_end (*context*)
Handle on_batch_end event.
Method called at the end of a batch.

Parameters `context` (*ashpy.contexts.context.Context*) – training context

Return type `None`

on_batch_start (*context*)
Handle on_batch_start event.
Method called at the beginning of a batch.

Parameters `context` (*ashpy.contexts.context.Context*) – training context

Return type `None`

on_epoch_end (*context*)
Handle on_epoch_end event.
Method called at the end of an epoch.

Parameters `context` (*ashpy.contexts.context.Context*) – training context

Return type `None`

on_epoch_start (*context*)
Handle on_epoch_start event.
Method called at the beginning of an epoch.

Parameters `context` (*ashpy.contexts.context.Context*) – training context

Return type None

on_event (*event, context*)

Handle the on_event event.

Method called when an event is triggered.

Parameters

- **event** (*ashpy.callbacks.events.Event*) – triggered event
- **context** (*ashpy.contexts.context.Context*) – training context

Return type None

on_exception (*context*)

Handle on_exception event.

Method called when an exception is raised.

Parameters **context** (*ashpy.contexts.context.Context*) – training context

Return type None

on_train_end (*context*)

Handle on_train_end event.

Method called at the end of the training loop.

Parameters **context** (*ashpy.contexts.context.Context*) – training context

Return type None

on_train_start (*context*)

Handle on_train_start event.

Method called at the beginning of the training loop.

Parameters **context** (*ashpy.contexts.context.Context*) – training context

Return type None

class `ashpy.callbacks.callback.Callback` (*name*)

Bases: `tensorflow.python.module.module.Module`

Callback definition.

Every callback must extend from this class. This class defines the basic events. Every event takes as input the context in order to use the objects defined. Inheritance from `tf.Module` is required since callbacks have a state

Order: .. code-block:

```
--on_train_start

----on_epoch_start

-----on_batch_start

-----on_batch_end

----on_epoch_end

--on_train_end
```

(continues on next page)

(continued from previous page)

```
on_exception - if an Exception was raised
on_event - Called when an event is triggered
```

__init__ (*name*)

Initialize the Callback.

Parameters **name** (*str*) – Callback name.

Warning: When using multiple callbacks with the same trainer make sure they have different ids.

Return type None

name

Return the name of the callback.

on_batch_end (*context*)

Handle on_batch_end event.

Method called at the end of a batch.

Parameters **context** (*ashpy.contexts.context.Context*) – training context

Return type None

on_batch_start (*context*)

Handle on_batch_start event.

Method called at the beginning of a batch.

Parameters **context** (*ashpy.contexts.context.Context*) – training context

Return type None

on_epoch_end (*context*)

Handle on_epoch_end event.

Method called at the end of an epoch.

Parameters **context** (*ashpy.contexts.context.Context*) – training context

Return type None

on_epoch_start (*context*)

Handle on_epoch_start event.

Method called at the beginning of an epoch.

Parameters **context** (*ashpy.contexts.context.Context*) – training context

Return type None

on_event (*event, context*)

Handle the on_event event.

Method called when an event is triggered.

Parameters

- **event** (*ashpy.callbacks.events.Event*) – triggered event
- **context** (*ashpy.contexts.context.Context*) – training context

Return type None

on_exception (*context*)

Handle on_exception event.

Method called when an exception is raised.

Parameters **context** (*ashpy.contexts.context.Context*) – training context

Return type None

on_train_end (*context*)

Handle on_train_end event.

Method called at the end of the training loop.

Parameters **context** (*ashpy.contexts.context.Context*) – training context

Return type None

on_train_start (*context*)

Handle on_train_start event.

Method called at the beginning of the training loop.

Parameters **context** (*ashpy.contexts.context.Context*) – training context

Return type None

4.2.11 classifier

Classifier callbacks.

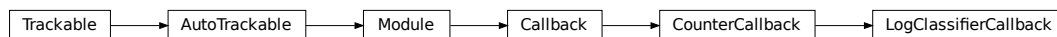
Classes

LogClassifierCallback

Callback used for logging Classifier images to Tensorboard.

LogClassifierCallback

Inheritance Diagram



```

class ashpy.callbacks.classifier.LogClassifierCallback (event=<Event.ON_EPOCH_END:
    'ON_EPOCH_END'>,
    name='log_classifier_callback',
    event_freq=1)
    
```

Bases: *ashpy.callbacks.counter_callback.CounterCallback*

Callback used for logging Classifier images to Tensorboard.

Logs the Input Image and true label.

Methods

<code>__init__</code> ([event, name, event_freq])	Initialize the LogClassifierCallback.
---	---------------------------------------

Attributes

<code>name</code>	Return the name of the callback.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>variables</code>	Sequence of variables owned by this module and it's submodules.

```
__init__ (event=<Event.ON_EPOCH_END: 'ON_EPOCH_END'>, name='log_classifier_callback',
          event_freq=1)
Initialize the LogClassifierCallback.
```

Parameters

- **event** (*Event*) – event to consider
- **event_freq** (*int*) – frequency of logging

```
static _log_fn (context)
Log output of the image and label to Tensorboard.
```

Parameters `context` (*ClassifierContext*) – current context

Return type None

```
class ashpy.callbacks.classifier.LogClassifierCallback (event=<Event.ON_EPOCH_END:
                                                         'ON_EPOCH_END'>,
                                                         name='log_classifier_callback',
                                                         event_freq=1)
```

Bases: *ashpy.callbacks.counter_callback.CounterCallback*

Callback used for logging Classifier images to Tensorboard.

Logs the Input Image and true label.

```
__init__ (event=<Event.ON_EPOCH_END: 'ON_EPOCH_END'>, name='log_classifier_callback',
          event_freq=1)
Initialize the LogClassifierCallback.
```

Parameters

- **event** (*Event*) – event to consider
- **event_freq** (*int*) – frequency of logging

```
static _log_fn (context)
Log output of the image and label to Tensorboard.
```

Parameters `context` (*ClassifierContext*) – current context

Return type None

4.2.12 counter_callback

Counter Callback implementation.

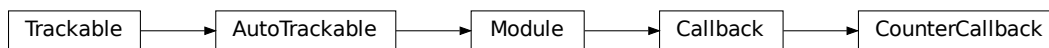
Callback that count events and calls the passed fn evert event_freq.

Classes

<i>CounterCallback</i>	Count events of a specific type.
------------------------	----------------------------------

CounterCallback

Inheritance Diagram



```
class ashpy.callbacks.counter_callback.CounterCallback (event, fn, name,
                                                    event_freq=1)
```

Bases: *ashpy.callbacks.callback.Callback*

Count events of a specific type. Calls fn passing the context every event_freq.

Useful for logging or for measuring performance. If you want to implement a callback defining a certain behaviour every n_events you can just inherit from CounterCallback.

Methods

<code>__init__</code> (event, fn, name[, event_freq])	Initialize the CounterCallback.
<code>on_event</code> (event, context)	Count events and calls fn.

Attributes

<code>name</code>	Return the name of the callback.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>variables</code>	Sequence of variables owned by this module and it's submodules.

```
__init__ (event, fn, name, event_freq=1)
    Initialize the CounterCallback.
```

Parameters

- **event** (`ashpy.events.Event`) – event to count.
- **fn** (Callable) – function to call every `event_freq` events.
- **event_freq** (`int`) – event frequency.
- **name** (`str`) – name of the Callback.

Raises `ValueError` – if `event_freq` is not valid.

Return type `None`

on_event (`event, context`)
Count events and calls fn.

Parameters

- **event** (`ashpy.callbacks.events.Event`) – current event.
- **context** (`ashpy.contexts.context.Context`) – current context.

class `ashpy.callbacks.counter_callback.CounterCallback` (`event`, `fn`, `name`,
`event_freq=1`)

Bases: `ashpy.callbacks.callback.Callback`

Count events of a specific type. Calls fn passing the context every `event_freq`.

Useful for logging or for measuring performance. If you want to implement a callback defining a certain behaviour every `n_events` you can just inherit from `CounterCallback`.

__init__ (`event, fn, name, event_freq=1`)
Initialize the `CounterCallback`.

Parameters

- **event** (`ashpy.events.Event`) – event to count.
- **fn** (Callable) – function to call every `event_freq` events.
- **event_freq** (`int`) – event frequency.
- **name** (`str`) – name of the Callback.

Raises `ValueError` – if `event_freq` is not valid.

Return type `None`

on_event (`event, context`)
Count events and calls fn.

Parameters

- **event** (`ashpy.callbacks.events.Event`) – current event.
- **context** (`ashpy.contexts.context.Context`) – current context.

4.2.13 events

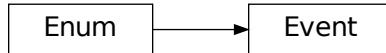
Event definition as Enum.

Classes

*Event*Define all possible events.

Event

Inheritance Diagram



class `ashpy.callbacks.events.Event`Bases: `enum.Enum`

Define all possible events.

Attributes

<code>ON_BATCH_END</code>	On Batch End Event
<code>ON_BATCH_START</code>	On Batch Start Event
<code>ON_EPOCH_END</code>	On Epoch End Event
<code>ON_EPOCH_START</code>	On Epoch Start Event
<code>ON_EXCEPTION</code>	On Exception Event
<code>ON_TRAIN_END</code>	On Train End Event
<code>ON_TRAIN_START</code>	On Train Start Event

`ON_BATCH_END = 'ON_BATCH_END'`
On Batch End Event

`ON_BATCH_START = 'ON_BATCH_START'`
On Batch Start Event

`ON_EPOCH_END = 'ON_EPOCH_END'`
On Epoch End Event

`ON_EPOCH_START = 'ON_EPOCH_START'`
On Epoch Start Event

`ON_EXCEPTION = 'ON_EXCEPTION'`
On Exception Event

`ON_TRAIN_END = 'ON_TRAIN_END'`
On Train End Event

`ON_TRAIN_START = 'ON_TRAIN_START'`
On Train Start Event

class `ashpy.callbacks.events.Event`Bases: `enum.Enum`

Define all possible events.

```
ON_BATCH_END = 'ON_BATCH_END'
    On Batch End Event

ON_BATCH_START = 'ON_BATCH_START'
    On Batch Start Event

ON_EPOCH_END = 'ON_EPOCH_END'
    On Epoch End Event

ON_EPOCH_START = 'ON_EPOCH_START'
    On Epoch Start Event

ON_EXCEPTION = 'ON_EXCEPTION'
    On Exception Event

ON_TRAIN_END = 'ON_TRAIN_END'
    On Train End Event

ON_TRAIN_START = 'ON_TRAIN_START'
    On Train Start Event
```

4.2.14 gan

GAN callbacks.

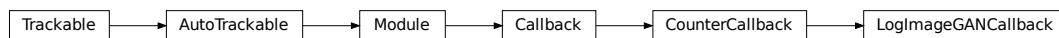
LogImageGANCallback: Log output of the generator when evaluated in its inputs. LogImageGANEncoderCallback: Log output of the generator when evaluated in the encoder

Classes

<i>LogImageGANCallback</i>	Callback used for logging GANs images to Tensorboard.
<i>LogImageGANEncoderCallback</i>	Callback used for logging GANs images to Tensorboard.

LogImageGANCallback

Inheritance Diagram



```
class ashpy.callbacks.gan.LogImageGANCallback(event=<Event.ON_EPOCH_END:
    'ON_EPOCH_END'>,
    name='log_image_gan_callback',
    event_freq=1)
```

Bases: *ashpy.callbacks.counter_callback.CounterCallback*

Callback used for logging GANs images to Tensorboard.

Logs the Generator output. Logs G(z).

Examples

```
import shutil
import operator
from pathlib import Path

generator = models.gans.ConvGenerator(
    layer_spec_input_res=(7, 7),
    layer_spec_target_res=(28, 28),
    kernel_size=(5, 5),
    initial_filters=32,
    filters_cap=16,
    channels=1,
)

discriminator = models.gans.ConvDiscriminator(
    layer_spec_input_res=(28, 28),
    layer_spec_target_res=(7, 7),
    kernel_size=(5, 5),
    initial_filters=16,
    filters_cap=32,
    output_shape=1,
)

# Losses
generator_bce = losses.gan.GeneratorBCE()
minmax = losses.gan.DiscriminatorMinMax()

# Real data
batch_size = 2
mnist_x, mnist_y = tf.zeros((100, 28, 28)), tf.zeros((100,))

# Trainer
epochs = 2
logdir = Path("testlog/callbacks")
callbacks = [callbacks.LogImageGANCallback()]
trainer = trainers.gan.AdversarialTrainer(
    generator=generator,
    discriminator=discriminator,
    generator_optimizer=tf.optimizers.Adam(1e-4),
    discriminator_optimizer=tf.optimizers.Adam(1e-4),
    generator_loss=generator_bce,
    discriminator_loss=minmax,
    epochs=epochs,
    callbacks=callbacks,
    logdir=logdir,
)

# take only 2 samples to speed up tests
real_data = (
    tf.data.Dataset.from_tensor_slices(
        (tf.expand_dims(mnist_x, -1), tf.expand_dims(mnist_y, -1))).take(batch_size)
    .batch(batch_size)
```

(continues on next page)

(continued from previous page)

```
.prefetch(1)
)

# Add noise in the same dataset, just by mapping.
# The return type of the dataset must be: tuple(tuple(a,b), noise)
dataset = real_data.map(
    lambda x, y: ((x, y), tf.random.normal(shape=(batch_size, 100)))
)

trainer(dataset)
shutil.rmtree(logdir)

assert not logdir.exists()

trainer._global_step.assign_add(500)
```

```
Initializing checkpoint.
Starting epoch 1.
[1] Saved checkpoint: testlog/callbacks/ckpts/ckpt-1
Epoch 1 completed.
Starting epoch 2.
[2] Saved checkpoint: testlog/callbacks/ckpts/ckpt-2
Epoch 2 completed.
Training finished after 2 epochs.
```

Methods

<code>__init__</code> ([event, name, event_freq])	Initialize the LogImageCallbackGAN.
---	-------------------------------------

Attributes

<code>name</code>	Return the name of the callback.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>variables</code>	Sequence of variables owned by this module and it's submodules.

```
__init__(event=<Event.ON_EPOCH_END: 'ON_EPOCH_END'>,
         name='log_image_gan_callback', event_freq=1)
Initialize the LogImageCallbackGAN.
```

Parameters

- **event** (*ashpy.callbacks.events.Event*) – event to consider.
- **event_freq** (*int*) – frequency of logging.
- **name** (*str*) – name of the callback.

Return type None

`_log_fn(context)`

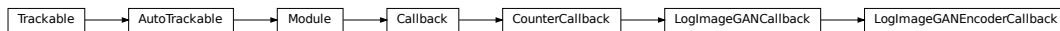
Log output of the generator to Tensorboard.

Parameters `context` (*ashpy.contexts.gan.GANContext*) – current context.

Return type None

LogImageGANEncoderCallback

Inheritance Diagram



```

class ashpy.callbacks.gan.LogImageGANEncoderCallback(event=<Event.ON_EPOCH_END:
    'ON_EPOCH_END'>,
    name='log_image_gan_encoder_callback',
    event_freq=1)

```

Bases: *ashpy.callbacks.gan.LogImageGANCallback*

Callback used for logging GANs images to Tensorboard.

Logs the Generator output evaluated in the encoder output. Logs $G(E(x))$.

Examples

```

import shutil
import operator

def real_gen():
    label = 0
    for _ in tf.range(100):
        yield ((10.0,), (label,))

latent_dim = 100

generator = tf.keras.Sequential([tf.keras.layers.Dense(1)])

left_input = tf.keras.layers.Input(shape=(1,))
left = tf.keras.layers.Dense(10, activation=tf.nn.elu)(left_input)

right_input = tf.keras.layers.Input(shape=(latent_dim,))
right = tf.keras.layers.Dense(10, activation=tf.nn.elu)(right_input)

net = tf.keras.layers.Concatenate()([left, right])
out = tf.keras.layers.Dense(1)(net)

discriminator = tf.keras.Model(inputs=[left_input, right_input], outputs=[out])

encoder = tf.keras.Sequential([tf.keras.layers.Dense(latent_dim)])

```

(continues on next page)

(continued from previous page)

```
# Losses
generator_bce = losses.gan.GeneratorBCE()
encoder_bce = losses.gan.EncoderBCE()
minmax = losses.gan.DiscriminatorMinMax()

epochs = 2

callbacks = [callbacks.LogImageGANEncoderCallback()]

logdir = "testlog/callbacks_encoder"

trainer = trainers.gan.EncoderTrainer(
    generator=generator,
    discriminator=discriminator,
    encoder=encoder,
    discriminator_optimizer=tf.optimizers.Adam(1e-4),
    generator_optimizer=tf.optimizers.Adam(1e-5),
    encoder_optimizer=tf.optimizers.Adam(1e-6),
    generator_loss=generator_bce,
    discriminator_loss=minmax,
    encoder_loss=encoder_bce,
    epochs=epochs,
    callbacks=callbacks,
    logdir=logdir,
)

batch_size = 10
discriminator_input = tf.data.Dataset.from_generator(
    real_gen, (tf.float32, tf.int64), ((1), (1))
).batch(batch_size)

dataset = discriminator_input.map(
    lambda x, y: ((x, y), tf.random.normal(shape=(batch_size, latent_dim)))
)

trainer(dataset)

shutil.rmtree(logdir)
```

```
Initializing checkpoint.
Starting epoch 1.
[10] Saved checkpoint: testlog/callbacks_encoder/ckpts/ckpt-1
Epoch 1 completed.
Starting epoch 2.
[20] Saved checkpoint: testlog/callbacks_encoder/ckpts/ckpt-2
Epoch 2 completed.
Training finished after 2 epochs.
```

Methods

<code>__init__</code> ([event, name, event_freq])	Initialize the LogImageGANEncoderCallback.
---	--

Attributes

<code>name</code>	Return the name of the callback.
<code>name_scope</code>	Returns a <code>tf.name_scope</code> instance for this class.
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>variables</code>	Sequence of variables owned by this module and it's submodules.

`__init__` (*event*=<`Event.ON_EPOCH_END`: `'ON_EPOCH_END'>`,
name='log_image_gan_encoder_callback', *event_freq*=1)
 Initialize the LogImageGANEncoderCallback.

Parameters

- **event** (`ashpy.callbacks.events.Event`) – event to consider.
- **event_freq** (`int`) – frequency of logging.
- **name** (`str`) – name of the callback.

Return type None

`_log_fn` (*context*)
 Log output of the generator to Tensorboard.
 Logs G(E(x)).

Parameters **context** (`ashpy.contexts.gan.GanEncoderContext`) – current context.

class `ashpy.callbacks.gan.LogImageGANCallback` (*event*=<`Event.ON_EPOCH_END`:
`'ON_EPOCH_END'>`,
name='log_image_gan_callback',
event_freq=1)

Bases: `ashpy.callbacks.counter_callback.CounterCallback`

Callback used for logging GANs images to Tensorboard.

Logs the Generator output. Logs G(z).

Examples

```
import shutil
import operator
from pathlib import Path

generator = models.gans.ConvGenerator(
    layer_spec_input_res=(7, 7),
    layer_spec_target_res=(28, 28),
    kernel_size=(5, 5),
    initial_filters=32,
    filters_cap=16,
    channels=1,
)

discriminator = models.gans.ConvDiscriminator(
```

(continues on next page)

(continued from previous page)

```

        layer_spec_input_res=(28, 28),
        layer_spec_target_res=(7, 7),
        kernel_size=(5, 5),
        initial_filters=16,
        filters_cap=32,
        output_shape=1,
    )

    # Losses
    generator_bce = losses.gan.GeneratorBCE()
    minmax = losses.gan.DiscriminatorMinMax()

    # Real data
    batch_size = 2
    mnist_x, mnist_y = tf.zeros((100,28,28)), tf.zeros((100,))

    # Trainer
    epochs = 2
    logdir = Path("testlog/callbacks")
    callbacks = [callbacks.LogImageGANCallback()]
    trainer = trainers.gan.AdversarialTrainer(
        generator=generator,
        discriminator=discriminator,
        generator_optimizer=tf.optimizers.Adam(1e-4),
        discriminator_optimizer=tf.optimizers.Adam(1e-4),
        generator_loss=generator_bce,
        discriminator_loss=minmax,
        epochs=epochs,
        callbacks=callbacks,
        logdir=logdir,
    )

    # take only 2 samples to speed up tests
    real_data = (
        tf.data.Dataset.from_tensor_slices(
            (tf.expand_dims(mnist_x, -1), tf.expand_dims(mnist_y, -1))).take(batch_size)
        .batch(batch_size)
        .prefetch(1)
    )

    # Add noise in the same dataset, just by mapping.
    # The return type of the dataset must be: tuple(tuple(a,b), noise)
    dataset = real_data.map(
        lambda x, y: ((x, y), tf.random.normal(shape=(batch_size, 100)))
    )

    trainer(dataset)
    shutil.rmtree(logdir)

    assert not logdir.exists()

    trainer._global_step.assign_add(500)

```

```

Initializing checkpoint.
Starting epoch 1.
[1] Saved checkpoint: testlog/callbacks/ckpts/ckpt-1

```

(continues on next page)

(continued from previous page)

```
Epoch 1 completed.
Starting epoch 2.
[2] Saved checkpoint: testlog/callbacks/ckpts/ckpt-2
Epoch 2 completed.
Training finished after 2 epochs.
```

```
__init__(event=<Event.ON_EPOCH_END: 'ON_EPOCH_END'>,
         name='log_image_gan_callback', event_freq=1)
    Initialize the LogImageCallbackGAN.
```

Parameters

- **event** (*ashpy.callbacks.events.Event*) – event to consider.
- **event_freq** (*int*) – frequency of logging.
- **name** (*str*) – name of the callback.

Return type None

```
_log_fn(context)
    Log output of the generator to Tensorboard.
```

Parameters **context** (*ashpy.contexts.gan.GANContext*) – current context.

Return type None

```
class ashpy.callbacks.gan.LogImageGANEncoderCallback(event=<Event.ON_EPOCH_END:
                                                    'ON_EPOCH_END'>,
                                                    name='log_image_gan_encoder_callback',
                                                    event_freq=1)
```

Bases: *ashpy.callbacks.gan.LogImageGANCallback*

Callback used for logging GANs images to Tensorboard.

Logs the Generator output evaluated in the encoder output. Logs $G(E(x))$.

Examples

```
import shutil
import operator

def real_gen():
    label = 0
    for _ in tf.range(100):
        yield ((10.0,), (label,))

latent_dim = 100

generator = tf.keras.Sequential([tf.keras.layers.Dense(1)])

left_input = tf.keras.layers.Input(shape=(1,))
left = tf.keras.layers.Dense(10, activation=tf.nn.elu)(left_input)

right_input = tf.keras.layers.Input(shape=(latent_dim,))
right = tf.keras.layers.Dense(10, activation=tf.nn.elu)(right_input)

net = tf.keras.layers.Concatenate()([left, right])
out = tf.keras.layers.Dense(1)(net)
```

(continues on next page)

(continued from previous page)

```
discriminator = tf.keras.Model(inputs=[left_input, right_input], outputs=[out])

encoder = tf.keras.Sequential([tf.keras.layers.Dense(latent_dim)])

# Losses
generator_bce = losses.gan.GeneratorBCE()
encoder_bce = losses.gan.EncoderBCE()
minmax = losses.gan.DiscriminatorMinMax()

epochs = 2

callbacks = [callbacks.LogImageGANEncoderCallback()]

logdir = "testlog/callbacks_encoder"

trainer = trainers.gan.EncoderTrainer(
    generator=generator,
    discriminator=discriminator,
    encoder=encoder,
    discriminator_optimizer=tf.optimizers.Adam(1e-4),
    generator_optimizer=tf.optimizers.Adam(1e-5),
    encoder_optimizer=tf.optimizers.Adam(1e-6),
    generator_loss=generator_bce,
    discriminator_loss=minmax,
    encoder_loss=encoder_bce,
    epochs=epochs,
    callbacks=callbacks,
    logdir=logdir,
)

batch_size = 10
discriminator_input = tf.data.Dataset.from_generator(
    real_gen, (tf.float32, tf.int64), ((1), (1))
).batch(batch_size)

dataset = discriminator_input.map(
    lambda x, y: ((x, y), tf.random.normal(shape=(batch_size, latent_dim)))
)

trainer(dataset)

shutil.rmtree(logdir)
```

```
Initializing checkpoint.
Starting epoch 1.
[10] Saved checkpoint: testlog/callbacks_encoder/ckpts/ckpt-1
Epoch 1 completed.
Starting epoch 2.
[20] Saved checkpoint: testlog/callbacks_encoder/ckpts/ckpt-2
Epoch 2 completed.
Training finished after 2 epochs.
```

```
__init__(event=<Event.ON_EPOCH_END: 'ON_EPOCH_END'>,
        name='log_image_gan_encoder_callback', event_freq=1)
Initialize the LogImageGANEncoderCallback.
```

Parameters

- **event** (*ashpy.callbacks.events.Event*) – event to consider.
- **event_freq** (*int*) – frequency of logging.
- **name** (*str*) – name of the callback.

Return type None

_log_fn (*context*)

Log output of the generator to Tensorboard.

Logs $G(E(x))$.

Parameters **context** (*ashpy.contexts.gan.GanEncoderContext*) – current context.

4.2.15 save_callback

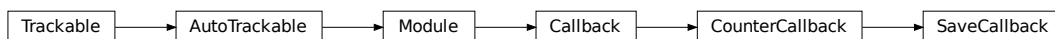
Save weights callback.

Classes

<i>SaveCallback</i>	Save Callback implementation.
<i>SaveFormat</i>	Save Format enum.
<i>SaveSubFormat</i>	Save Sub-Format enum.

SaveCallback

Inheritance Diagram



```

class ashpy.callbacks.save_callback.SaveCallback (save_dir, models,
                                                    event=<Event.ON_EPOCH_END:
'ON_EPOCH_END'>,
                                                    event_freq=1, max_to_keep=1,
                                                    save_format=<SaveFormat.MODEL\WEIGHTS:
3>, save_sub_format=<SaveSubFormat.TF:
'tf'>, verbose=0,
                                                    name='SaveCallback')
Bases: ashpy.callbacks.counter_callback.CounterCallback
Save Callback implementation.

```

Examples

```
import shutil
import operator
import os

generator = models.gans.ConvGenerator(
    layer_spec_input_res=(7, 7),
    layer_spec_target_res=(28, 28),
    kernel_size=(5, 5),
    initial_filters=32,
    filters_cap=16,
    channels=1,
)

discriminator = models.gans.ConvDiscriminator(
    layer_spec_input_res=(28, 28),
    layer_spec_target_res=(7, 7),
    kernel_size=(5, 5),
    initial_filters=16,
    filters_cap=32,
    output_shape=1,
)

models = [generator, discriminator]

save_callback = callbacks.SaveCallback(save_dir="testlog/savedir",
                                       models=models,
                                       save_format=callbacks.SaveFormat.WEIGHTS,
                                       save_sub_format=callbacks.SaveSubFormat.TF)

# initialize trainer passing the save_callback
```

Methods

<code>__init__(save_dir, models[, event, ...])</code>	Build a Save Callback.
<code>save_weights_fn(context)</code>	Save weights and clean up if needed.

Attributes

<code>name</code>	Return the name of the callback.
<code>name_scope</code>	Returns a <code>tf.name_scope</code> instance for this class.
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>variables</code>	Sequence of variables owned by this module and it's submodules.

```
__init__(save_dir, models, event=<Event.ON_EPOCH_END: 'ON_EPOCH_END'>,
         event_freq=1, max_to_keep=1, save_format=<SaveFormat.MODEL|WEIGHTS: 3>,
         save_sub_format=<SaveSubFormat.TF: 'tf'>, verbose=0, name='SaveCallback')
Build a Save Callback.
```

Save Callbacks are used to save the model on events. You can specify two different save formats: weights and model. At the same time you can specify two different save sub-formats: tf or h5. You will find the model saved in the `save_dir` under the directory weights or model.

Parameters

- **save_dir** (*str*) – directory in which to save the weights or the model.
- **models** (`List[tf.keras.models.Model]`) – list of models to save.
- **event** (`ashpy.callbacks.events.Event`) – events on which to trigger the saving operation.
- **event_freq** (*int*) – frequency of saving operation.
- **name** (*str*) – name of the callback.
- **verbose** (*int*) – verbosity of the callback (0 or 1).
- **max_to_keep** (*int*) – maximum files to keep. If `max_to_keep == 1` only the most recent file is kept. In general `max_to_keep` files are kept.
- **save_format** (`ashpy.callbacks.save_callback.SaveFormat`) – weights or model.
- **save_sub_format** (`ashpy.callbacks.save_callback.SaveSubFormat`) – sub-format of the saving (tf or h5).

_cleanup ()
Cleanup stuff.

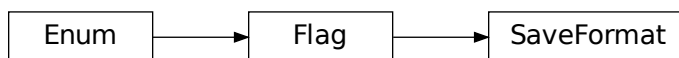
_save_weights_fn (*step*)
Save weights.

Parameters **step** (*int*) – current step.

save_weights_fn (*context*)
Save weights and clean up if needed.

SaveFormat

Inheritance Diagram



```

class ashpy.callbacks.save_callback.SaveFormat
    Bases: enum.Flag
    Save Format enum.
  
```

Attributes

<i>MODEL</i>	Model format (weights and architecture), saved using <i>model.save()</i>
<i>WEIGHTS</i>	Weights format, saved using <i>model.save_weights()</i>

MODEL = 2

Model format (weights and architecture), saved using *model.save()*

WEIGHTS = 1

Weights format, saved using *model.save_weights()*

name()

Name of the format.

Return type `str`

save (*model*, *save_dir*, *save_sub_format*=<*SaveSubFormat.TF*: 'tf'>)

Save the model using the correct format and sub-format.

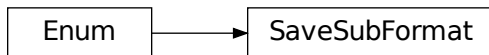
Parameters

- **model** (`tf.keras.models.Model`) – model to Save.
- **save_dir** (`pathlib.Path`) – path of the file in which to save the model.
- **save_sub_format** (`ashpy.callbacks.save_callback.SaveSubFormat`) – sub-format of the save operation.

Return type `None`

SaveSubFormat

Inheritance Diagram



class `ashpy.callbacks.save_callback.SaveSubFormat`

Bases: `enum.Enum`

Save Sub-Format enum.

Attributes

<i>H5</i>	H5 Format
<i>TF</i>	TensorFlow format

H5 = 'h5'
H5 Format

TF = 'tf'
TensorFlow format

```
class ashpy.callbacks.save_callback.SaveCallback(save_dir, models,
                                                event=<Event.ON_EPOCH_END:
                                                'ON_EPOCH_END'>,
                                                event_freq=1, max_to_keep=1,
                                                save_format=<SaveFormat.MODEL\WEIGHTS:
                                                3>, save_sub_format=<SaveSubFormat.TF:
                                                'tf'>, verbose=0,
                                                name='SaveCallback')
```

Bases: *ashpy.callbacks.counter_callback.CounterCallback*

Save Callback implementation.

Examples

```
import shutil
import operator
import os

generator = models.gans.ConvGenerator(
    layer_spec_input_res=(7, 7),
    layer_spec_target_res=(28, 28),
    kernel_size=(5, 5),
    initial_filters=32,
    filters_cap=16,
    channels=1,
)

discriminator = models.gans.ConvDiscriminator(
    layer_spec_input_res=(28, 28),
    layer_spec_target_res=(7, 7),
    kernel_size=(5, 5),
    initial_filters=16,
    filters_cap=32,
    output_shape=1,
)

models = [generator, discriminator]

save_callback = callbacks.SaveCallback(save_dir="testlog/savedir",
                                       models=models,
                                       save_format=callbacks.SaveFormat.WEIGHTS,
                                       save_sub_format=callbacks.SaveSubFormat.TF)

# initialize trainer passing the save_callback
```

```
__init__(save_dir, models, event=<Event.ON_EPOCH_END: 'ON_EPOCH_END'>,
         event_freq=1, max_to_keep=1, save_format=<SaveFormat.MODEL\WEIGHTS: 3>,
         save_sub_format=<SaveSubFormat.TF: 'tf'>, verbose=0, name='SaveCallback')
```

Build a Save Callback.

Save Callbacks are used to save the model on events. You can specify two different save formats: weights and model. At the same time you can specify two different save sub-formats: tf or h5. You will find the

model saved in the `save_dir` under the directory `weights` or `model`.

Parameters

- **save_dir** (*str*) – directory in which to save the weights or the model.
- **models** (`List[tf.keras.models.Model]`) – list of models to save.
- **event** (*ashpy.callbacks.events.Event*) – events on which to trigger the saving operation.
- **event_freq** (*int*) – frequency of saving operation.
- **name** (*str*) – name of the callback.
- **verbose** (*int*) – verbosity of the callback (0 or 1).
- **max_to_keep** (*int*) – maximum files to keep. If `max_to_keep == 1` only the most recent file is kept. In general `max_to_keep` files are kept.
- **save_format** (*ashpy.callbacks.save_callback.SaveFormat*) – weights or model.
- **save_sub_format** (*ashpy.callbacks.save_callback.SaveSubFormat*) – sub-format of the saving (tf or h5).

_cleanup ()

Cleanup stuff.

_save_weights_fn (*step*)

Save weights.

Parameters **step** (*int*) – current step.

save_weights_fn (*context*)

Save weights and clean up if needed.

class `ashpy.callbacks.save_callback.SaveFormat`

Bases: `enum.Flag`

Save Format enum.

MODEL = 2

Model format (weights and architecture), saved using `model.save()`

WEIGHTS = 1

Weights format, saved using `model.save_weights()`

name ()

Name of the format.

Return type *str*

save (*model*, *save_dir*, *save_sub_format*=<`SaveSubFormat.TF`: 'tf'>)

Save the model using the correct format and sub-format.

Parameters

- **model** (`tf.keras.models.Model`) – model to Save.
- **save_dir** (`pathlib.Path`) – path of the file in which to save the model.
- **save_sub_format** (*ashpy.callbacks.save_callback.SaveSubFormat*) – sub-format of the save operation.

Return type `None`

```
class ashpy.callbacks.save_callback.SaveSubFormat
    Bases: enum.Enum
    Save Sub-Format enum.

    H5 = 'h5'
        H5 Format

    TF = 'tf'
        TensorFlow format
```

4.3 ashpy.contexts

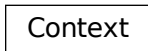
Contexts help gaining an easier control over the model selection and testing process of the models.

Classes

<code>context.Context</code>	ashpy.contexts.Context provide an interface for all contexts.
<code>classifier.ClassifierContext</code>	ashpy.ClassifierContext provide the standard functions to test a classifier.
<code>gan.GANContext</code>	<code>ashpy.contexts.gan.GANContext</code> measure the specified metrics on the GAN.
<code>gan.GANEncoderContext</code>	<code>ashpy.contexts.gan.GANEncoderContext</code> measure the specified metrics on the GAN.

4.3.1 Context

Inheritance Diagram



```
class ashpy.contexts.context.Context (metrics=None, dataset=None,
                                     log_eval_mode=<LogEvalMode.TEST: 1>,
                                     global_step=<tf.Variable 'global_step:0' shape=()
                                     dtype=int64, numpy=0>, checkpoint=None)
```

Bases: `object`

ashpy.contexts.Context provide an interface for all contexts.

Methods

<code>__init__</code> ([metrics, dataset, log_eval_mode, ...])	Initialize the Context.
--	-------------------------

Attributes

<code>current_batch</code>	Return the current batch.
<code>dataset</code>	Retrieve the dataset.
<code>exception</code>	Return the exception.
<code>global_step</code>	Retrieve the global_step.
<code>log_eval_mode</code>	Retrieve model(s) mode.
<code>metrics</code>	Retrieve the metrics.

`__init__`(metrics=None, dataset=None, log_eval_mode=<LogEvalMode.TEST: 1>, global_step=<tf.Variable 'global_step:0' shape=() dtype=int64, numpy=0>, checkpoint=None)
Initialize the Context.

Parameters

- **metrics** (tuple of (`ashpy.metrics.metric.Metric`)) – List of `ashpy.metrics.metric.Metric` objects.
- **dataset** (`tf.data.Dataset`) – The dataset to use, that contains everything needed to use the model in this context.
- **log_eval_mode** (`ashpy.modes.LogEvalMode`) – Models' mode to use when evaluating and logging.
- **global_step** (`tf.Variable`) – Keeps track of the training steps.
- **checkpoint** (`tf.train.Checkpoint`) – Checkpoint to use to keep track of models status.

Return type None

`current_batch`

Return the current batch.

Return type `Optional[Tensor]`

`dataset`

Retrieve the dataset.

Return type `DatasetV2`

Returns `tf.data.Dataset` the current dataset

`exception`

Return the exception.

Return type `Optional[Exception]`

`global_step`

Retrieve the global_step.

Return type `Variable`

Returns `tf.Variable`.

`log_eval_mode`

Retrieve model(s) mode.

Return type *LogEvalMode*

Returns *ashpy.modes.LogEvalMode*.

metrics

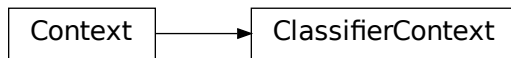
Retrieve the metrics.

Return type *Tuple[Metric]*

Returns *tuple of (ashpy.metrics.metric.Metric)*.

4.3.2 ClassifierContext

Inheritance Diagram



```

class ashpy.contexts.classifier.ClassifierContext (classifier_model=None, loss=None,
                                                    dataset=None, metrics=None,
                                                    log_eval_mode=<LogEvalMode.TEST:
1>, global_step=<tf.Variable
'global_step:0' shape=()
dtype=int64, numpy=0>, check-
point=None)
  
```

Bases: *ashpy.contexts.context.Context*

ashpy.ClassifierContext provide the standard functions to test a classifier.

Methods

<code>__init__</code> ([classifier_model, loss, dataset, ...])	Instantiate the <i>ashpy.contexts.classifier.ClassifierContext</i> context.
--	---

Attributes

<code>classifier_model</code>	Retrieve the Model Object.
<code>current_batch</code>	Return the current batch.
<code>dataset</code>	Retrieve the dataset.
<code>exception</code>	Return the exception.
<code>global_step</code>	Retrieve the global_step.
<code>log_eval_mode</code>	Retrieve model(s) mode.
<code>loss</code>	Retrieve the loss value.
<code>metrics</code>	Retrieve the metrics.

Continued on next page

Table 44 – continued from previous page

<code>training_set</code>	Return the training set.
<code>validation_set</code>	Return the validation set.

`__init__` (*classifier_model=None*, *loss=None*, *dataset=None*, *metrics=None*,
log_eval_mode=<LogEvalMode.TEST: 1>, *global_step=<tf.Variable 'global_step:0'*
shape=() dtype=int64, numpy=0>, *checkpoint=None*)
 Instantiate the `ashpy.contexts.classifier.ClassifierContext` context.

Parameters

- **classifier_model** (`tf.keras.Model`) – A `tf.keras.Model` model.
- **loss** (`ashpy.losses.executor.Executor`) – Loss function, format `f(y_true, y_pred)`.
- **dataset** (`tf.data.Dataset`) – The test dataset.
- **metrics** (list of [`ashpy.metrics.metric.Metric`]) – List of `ashpy.metrics.metric.Metric` with which to measure training and validation data performances.
- **log_eval_mode** (`ashpy.modes.LogEvalMode`) – Models' mode to use when evaluating and logging.
- **global_step** (`tf.Variable`) – `tf.Variable` that keeps track of the training steps.
- **checkpoint** (`tf.train.Checkpoint`) – checkpoint to use to keep track of models status.

Return type `None`

classifier_model

Retrieve the Model Object.

Return type `Model`

Returns `tf.keras.Model` – The classifier model.

loss

Retrieve the loss value.

Return type `Optional[Executor]`

training_set

Return the training set.

Return type `DatasetV2`

Returns `tf.data.Dataset` – The training set.

validation_set

Return the validation set.

Return type `Optional[DatasetV2]`

Returns `tf.data.Dataset` – The validation set.

4.3.3 GANContext

Inheritance Diagram



```

class ashpy.contexts.gan.GANContext (dataset=None, generator_model=None, dis-
    criminator_model=None, generator_loss=None,
    discriminator_loss=None, metrics=None,
    log_eval_mode=<LogEvalMode.TRAIN: 2>,
    global_step=<tf.Variable 'global_step:0' shape=()
    dtype=int64, numpy=0>, checkpoint=None)
    
```

Bases: `ashpy.contexts.context.Context`

`ashpy.contexts.gan.GANContext` measure the specified metrics on the GAN.

Methods

<code>__init__</code> ([dataset, generator_model, ...])	Initialize the Context.
---	-------------------------

Attributes

<code>current_batch</code>	Return the current batch.
<code>dataset</code>	Retrieve the dataset.
<code>discriminator_loss</code>	Retrieve the discriminator loss.
<code>discriminator_model</code>	Retrieve the discriminator model.
<code>exception</code>	Return the exception.
<code>fake_samples</code>	Retrieve the fake samples, i.e.
<code>generator_inputs</code>	Retrieve the generator inputs.
<code>generator_loss</code>	Retrieve the generator loss.
<code>generator_model</code>	Retrieve the generator model.
<code>global_step</code>	Retrieve the global_step.
<code>log_eval_mode</code>	Retrieve model(s) mode.
<code>metrics</code>	Retrieve the metrics.

```

__init__(dataset=None, generator_model=None, discriminator_model=None, generator_loss=None,
    discriminator_loss=None, metrics=None, log_eval_mode=<LogEvalMode.TRAIN: 2>,
    global_step=<tf.Variable 'global_step:0' shape=() dtype=int64, numpy=0>, check-
    point=None)
    
```

Initialize the Context.

Parameters

- **dataset** (`tf.data.Dataset`) – Dataset of tuples. [0] true dataset, [1] generator input dataset.

- **generator_model** (`tf.keras.Model`) – The generator.
- **discriminator_model** (`tf.keras.Model`) – The discriminator.
- **generator_loss** (`ashpy.losses.Executor()`) – The generator loss.
- **discriminator_loss** (`ashpy.losses.Executor()`) – The discriminator loss.
- **metrics** (list of [`ashpy.metrics.metric.Metric`]) – All the metrics to be used to evaluate the model.
- **log_eval_mode** (`ashpy.modes.LogEvalMode`) – Models' mode to use when evaluating and logging.
- **global_step** (`tf.Variable`) – *tf.Variable* that keeps track of the training steps.
- **checkpoint** (`tf.train.Checkpoint`) – checkpoint to use to keep track of models status.

Return type `None`

discriminator_loss

Retrieve the discriminator loss.

Return type `Optional[Executor]`

discriminator_model

Retrieve the discriminator model.

Return type `Model`

Returns `tf.keras.Model`.

fake_samples

Retrieve the fake samples, i.e. output of the generator.

Return type `Optional[Tensor]`

generator_inputs

Retrieve the generator inputs.

Return type `Optional[Tensor]`

generator_loss

Retrieve the generator loss.

Return type `Optional[Executor]`

generator_model

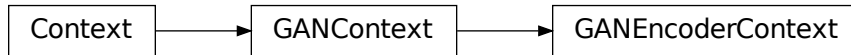
Retrieve the generator model.

Return type `Model`

Returns `tf.keras.Model`.

4.3.4 GANEncoderContext

Inheritance Diagram



```

class ashpy.contexts.gan.GANEncoderContext (dataset=None, generator_model=None,
                                             discriminator_model=None, en-
                                             coder_model=None, generator_loss=None,
                                             discriminator_loss=None, en-
                                             coder_loss=None, metrics=None,
                                             log_eval_mode=<LogEvalMode.TRAIN:
                                             2>, global_step=<tf.Variable 'global_step:0'
                                             shape=() dtype=int64, numpy=0>, check-
                                             point=None)
    
```

Bases: *ashpy.contexts.gan.GANContext*

ashpy.contexts.gan.GANEncoderContext measure the specified metrics on the GAN.

Methods

<code>__init__</code> ([dataset, generator_model, ...])	Initialize the Context.
---	-------------------------

Attributes

<code>current_batch</code>	Return the current batch.
<code>dataset</code>	Retrieve the dataset.
<code>discriminator_loss</code>	Retrieve the discriminator loss.
<code>discriminator_model</code>	Retrieve the discriminator model.
<code>encoder_inputs</code>	Retrieve the inputs of the encoder.
<code>encoder_loss</code>	Retrieve the encoder loss.
<code>encoder_model</code>	Retrieve the encoder model.
<code>exception</code>	Return the exception.
<code>fake_samples</code>	Retrieve the fake samples, i.e.
<code>generator_inputs</code>	Retrieve the generator inputs.
<code>generator_loss</code>	Retrieve the generator loss.
<code>generator_model</code>	Retrieve the generator model.
<code>generator_of_encoder</code>	Retrieve the images generated from the encoder output.
<code>global_step</code>	Retrieve the global_step.
<code>log_eval_mode</code>	Retrieve model(s) mode.
<code>metrics</code>	Retrieve the metrics.

```
__init__(dataset=None, generator_model=None, discriminator_model=None, encoder_model=None, generator_loss=None, discriminator_loss=None, encoder_loss=None, metrics=None, log_eval_mode=<LogEvalMode.TRAIN: 2>, global_step=<tf.Variable 'global_step:0' shape=() dtype=int64, numpy=0>, checkpoint=None)
```

Initialize the Context.

Parameters

- **dataset** (`tf.data.Dataset`) – Dataset of tuples. [0] true dataset, [1] generator input dataset.
- **generator_model** (`tf.keras.Model`) – The generator.
- **discriminator_model** (`tf.keras.Model`) – The discriminator.
- **encoder_model** (`tf.keras.Model`) – The encoder.
- **generator_loss** (`ashpy.losses.Executor()`) – The generator loss.
- **discriminator_loss** (`ashpy.losses.Executor()`) – The discriminator loss.
- **encoder_loss** (`ashpy.losses.Executor()`) – The encoder loss.
- **metrics** (list of [`ashpy.metrics.metric.Metric`]) – All the metrics to be used to evaluate the model.
- **log_eval_mode** (`ashpy.modes.LogEvalMode`) – Models' mode to use when evaluating and logging.
- **global_step** (`tf.Variable`) – `tf.Variable` that keeps track of the training steps.
- **checkpoint** (`tf.train.Checkpoint`) – checkpoint to use to keep track of models status.

Return type `None`

encoder_inputs

Retrieve the inputs of the encoder.

Return type `Tensor`

encoder_loss

Retrieve the encoder loss.

Return type `Optional[Executor]`

encoder_model

Retrieve the encoder model.

Return type `Model`

Returns `tf.keras.Model`.

generator_of_encoder

Retrieve the images generated from the encoder output.

Return type `Tensor`

Modules

<i>context</i>	Primitive Context Interface.
<i>classifier</i>	Classifier Context.
<i>gan</i>	GANContext measures the specified metrics on the GAN.

4.3.5 context

Primitive Context Interface.

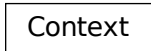
Contexts are checkpointable (subclassed from `tf.train.Checkpoint`) collections of variable encapsulated in a Python Class as a way to seamlessly handle information transfer.

Classes

<i>Context</i>	<code>ashpy.contexts.Context</code> provide an interface for all contexts.
----------------	--

Context

Inheritance Diagram



```

class ashpy.contexts.context.Context (metrics=None, dataset=None,
log_eval_mode=<LogEvalMode.TEST: 1>,
global_step=<tf.Variable 'global_step:0' shape=()
dtype=int64, numpy=0>, checkpoint=None)
  
```

Bases: `object`

`ashpy.contexts.Context` provide an interface for all contexts.

Methods

<code>__init__</code> ([metrics, dataset, log_eval_mode, ...])	Initialize the Context.
--	-------------------------

Attributes

<i>current_batch</i>	Return the current batch.
<i>dataset</i>	Retrieve the dataset.
<i>exception</i>	Return the exception.

Continued on next page

Table 52 – continued from previous page

<code>global_step</code>	Retrieve the <code>global_step</code> .
<code>log_eval_mode</code>	Retrieve model(s) mode.
<code>metrics</code>	Retrieve the metrics.

`__init__` (*metrics=None*, *dataset=None*, *log_eval_mode=<LogEvalMode.TEST: 1>*, *global_step=<tf.Variable 'global_step:0' shape=() dtype=int64, numpy=0>*, *checkpoint=None*)
Initialize the Context.

Parameters

- **metrics** (tuple of (*ashpy.metrics.metric.Metric*)) – List of *ashpy.metrics.metric.Metric* objects.
- **dataset** (*tf.data.Dataset*) – The dataset to use, that contains everything needed to use the model in this context.
- **log_eval_mode** (*ashpy.modes.LogEvalMode*) – Models' mode to use when evaluating and logging.
- **global_step** (*tf.Variable*) – Keeps track of the training steps.
- **checkpoint** (*tf.train.Checkpoint*) – Checkpoint to use to keep track of models status.

Return type *None*

current_batch

Return the current batch.

Return type *Optional[Tensor]*

dataset

Retrieve the dataset.

Return type *DatasetV2*

Returns *tf.data.Dataset* the current dataset

exception

Return the exception.

Return type *Optional[Exception]*

global_step

Retrieve the `global_step`.

Return type *Variable*

Returns *tf.Variable*.

log_eval_mode

Retrieve model(s) mode.

Return type *LogEvalMode*

Returns *ashpy.modes.LogEvalMode*.

metrics

Retrieve the metrics.

Return type *Tuple[Metric]*

Returns tuple of (*ashpy.metrics.metric.Metric*).

```
class ashpy.contexts.context.Context (metrics=None, dataset=None,
                                     log_eval_mode=<LogEvalMode.TEST: 1>,
                                     global_step=<tf.Variable 'global_step:0' shape=()
                                     dtype=int64, numpy=0>, checkpoint=None)
```

Bases: `object`

`ashpy.contexts.Context` provide an interface for all contexts.

```
__init__ (metrics=None, dataset=None, log_eval_mode=<LogEvalMode.TEST: 1>,
          global_step=<tf.Variable 'global_step:0' shape=() dtype=int64, numpy=0>, check-
          point=None)
Initialize the Context.
```

Parameters

- **metrics** (tuple of (`ashpy.metrics.metric.Metric`)) – List of `ashpy.metrics.metric.Metric` objects.
- **dataset** (`tf.data.Dataset`) – The dataset to use, that contains everything needed to use the model in this context.
- **log_eval_mode** (`ashpy.modes.LogEvalMode`) – Models' mode to use when evaluating and logging.
- **global_step** (`tf.Variable`) – Keeps track of the training steps.
- **checkpoint** (`tf.train.Checkpoint`) – Checkpoint to use to keep track of models status.

Return type `None`

current_batch

Return the current batch.

Return type `Optional[Tensor]`

dataset

Retrieve the dataset.

Return type `DatasetV2`

Returns `tf.data.Dataset` the current dataset

exception

Return the exception.

Return type `Optional[Exception]`

global_step

Retrieve the `global_step`.

Return type `Variable`

Returns `tf.Variable`.

log_eval_mode

Retrieve model(s) mode.

Return type `LogEvalMode`

Returns `ashpy.modes.LogEvalMode`.

metrics

Retrieve the metrics.

Return type `Tuple[Metric]`

Returns tuple of (*ashpy.metrics.metric.Metric*).

4.3.6 classifier

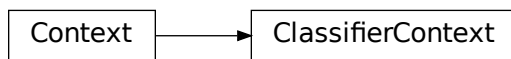
Classifier Context.

Classes

<i>ClassifierContext</i>	<i>ashpy.ClassifierContext</i> provide the standard functions to test a classifier.
--------------------------	---

ClassifierContext

Inheritance Diagram



```

class ashpy.contexts.classifier.ClassifierContext (classifier_model=None, loss=None,
                                                    dataset=None, metrics=None,
                                                    log_eval_mode=<LogEvalMode.TEST:
1>, global_step=<tf.Variable
'global_step:0' shape=()
dtype=int64, numpy=0>, check-
point=None)
  
```

Bases: *ashpy.contexts.context.Context*

ashpy.ClassifierContext provide the standard functions to test a classifier.

Methods

<code>__init__([classifier_model, loss, dataset, ...])</code>	Instantiate the <i>ashpy.contexts.classifier.ClassifierContext</i> context.
---	---

Attributes

<i>classifier_model</i>	Retrieve the Model Object.
<i>current_batch</i>	Return the current batch.
<i>dataset</i>	Retrieve the dataset.
<i>exception</i>	Return the exception.
<i>global_step</i>	Retrieve the <i>global_step</i> .

Continued on next page

Table 55 – continued from previous page

<code>log_eval_mode</code>	Retrieve model(s) mode.
<code>loss</code>	Retrieve the loss value.
<code>metrics</code>	Retrieve the metrics.
<code>training_set</code>	Return the training set.
<code>validation_set</code>	Return the validation set.

`__init__` (*classifier_model=None*, *loss=None*, *dataset=None*, *metrics=None*,
log_eval_mode=<LogEvalMode.TEST: 1>, *global_step=<tf.Variable 'global_step:0'*
shape=() dtype=int64, numpy=0>, *checkpoint=None*)
 Instantiate the `ashpy.contexts.classifier.ClassifierContext` context.

Parameters

- **classifier_model** (`tf.keras.Model`) – A `tf.keras.Model` model.
- **loss** (`ashpy.losses.executor.Executor`) – Loss function, format `f(y_true, y_pred)`.
- **dataset** (`tf.data.Dataset`) – The test dataset.
- **metrics** (list of [`ashpy.metrics.metric.Metric`]) – List of `ashpy.metrics.metric.Metric` with which to measure training and validation data performances.
- **log_eval_mode** (`ashpy.modes.LogEvalMode`) – Models' mode to use when evaluating and logging.
- **global_step** (`tf.Variable`) – `tf.Variable` that keeps track of the training steps.
- **checkpoint** (`tf.train.Checkpoint`) – checkpoint to use to keep track of models status.

Return type `None`

classifier_model

Retrieve the Model Object.

Return type `Model`

Returns `tf.keras.Model` – The classifier model.

loss

Retrieve the loss value.

Return type `Optional[Executor]`

training_set

Return the training set.

Return type `DatasetV2`

Returns `tf.data.Dataset` – The training set.

validation_set

Return the validation set.

Return type `Optional[DatasetV2]`

Returns `tf.data.Dataset` – The validation set.

```
class ashpy.contexts.classifier.ClassifierContext (classifier_model=None, loss=None,
                                                dataset=None, metrics=None,
                                                log_eval_mode=<LogEvalMode.TEST:
1>, global_step=<tf.Variable
'global_step:0' shape=()
dtype=int64, numpy=0>, check-
point=None)
```

Bases: `ashpy.contexts.context.Context`

`ashpy.ClassifierContext` provide the standard functions to test a classifier.

```
__init__ (classifier_model=None, loss=None, dataset=None, metrics=None,
log_eval_mode=<LogEvalMode.TEST: 1>, global_step=<tf.Variable 'global_step:0'
shape=() dtype=int64, numpy=0>, checkpoint=None)
```

Instantiate the `ashpy.contexts.classifier.ClassifierContext` context.

Parameters

- **classifier_model** (`tf.keras.Model`) – A `tf.keras.Model` model.
- **loss** (`ashpy.losses.executor.Executor`) – Loss function, format `f(y_true, y_pred)`.
- **dataset** (`tf.data.Dataset`) – The test dataset.
- **metrics** (list of [`ashpy.metrics.metric.Metric`]) – List of `ashpy.metrics.metric.Metric` with which to measure training and validation data performances.
- **log_eval_mode** (`ashpy.modes.LogEvalMode`) – Models' mode to use when evaluating and logging.
- **global_step** (`tf.Variable`) – `tf.Variable` that keeps track of the training steps.
- **checkpoint** (`tf.train.Checkpoint`) – checkpoint to use to keep track of models status.

Return type `None`

classifier_model

Retrieve the Model Object.

Return type `Model`

Returns `tf.keras.Model` – The classifier model.

loss

Retrieve the loss value.

Return type `Optional[Executor]`

training_set

Return the training set.

Return type `DatasetV2`

Returns `tf.data.Dataset` – The training set.

validation_set

Return the validation set.

Return type `Optional[DatasetV2]`

Returns `tf.data.Dataset` – The validation set.

4.3.7 gan

GANContext measures the specified metrics on the GAN.

Classes

<i>GANContext</i>	<i>ashpy.contexts.gan.GANContext</i> measure the specified metrics on the GAN.
<i>GANEncoderContext</i>	<i>ashpy.contexts.gan.GANEncoderContext</i> measure the specified metrics on the GAN.

GANContext

Inheritance Diagram



```

class ashpy.contexts.gan.GANContext (dataset=None, generator_model=None, dis-
    criminator_model=None, generator_loss=None,
    discriminator_loss=None, metrics=None,
    log_eval_mode=<LogEvalMode.TRAIN: 2>,
    global_step=<tf.Variable 'global_step:0' shape=()
    dtype=int64, numpy=0>, checkpoint=None)

Bases: ashpy.contexts.context.Context
ashpy.contexts.gan.GANContext measure the specified metrics on the GAN.
    
```

Methods

<i>__init__</i> ([dataset, generator_model, ...])	Initialize the Context.
---	-------------------------

Attributes

<i>current_batch</i>	Return the current batch.
<i>dataset</i>	Retrieve the dataset.
<i>discriminator_loss</i>	Retrieve the discriminator loss.
<i>discriminator_model</i>	Retrieve the discriminator model.
<i>exception</i>	Return the exception.
<i>fake_samples</i>	Retrieve the fake samples, i.e.
<i>generator_inputs</i>	Retrieve the generator inputs.

Continued on next page

Table 58 – continued from previous page

<code>generator_loss</code>	Retrieve the generator loss.
<code>generator_model</code>	Retrieve the generator model.
<code>global_step</code>	Retrieve the <code>global_step</code> .
<code>log_eval_mode</code>	Retrieve model(s) mode.
<code>metrics</code>	Retrieve the metrics.

`__init__` (*dataset=None, generator_model=None, discriminator_model=None, generator_loss=None, discriminator_loss=None, metrics=None, log_eval_mode=<LogEvalMode.TRAIN: 2>, global_step=<tf.Variable 'global_step:0' shape=() dtype=int64, numpy=0>, checkpoint=None*)

Initialize the Context.

Parameters

- **dataset** (`tf.data.Dataset`) – Dataset of tuples. [0] true dataset, [1] generator input dataset.
- **generator_model** (`tf.keras.Model`) – The generator.
- **discriminator_model** (`tf.keras.Model`) – The discriminator.
- **generator_loss** (`ashpy.losses.Executor()`) – The generator loss.
- **discriminator_loss** (`ashpy.losses.Executor()`) – The discriminator loss.
- **metrics** (list of [`ashpy.metrics.metric.Metric`]) – All the metrics to be used to evaluate the model.
- **log_eval_mode** (`ashpy.modes.LogEvalMode`) – Models' mode to use when evaluating and logging.
- **global_step** (`tf.Variable`) – `tf.Variable` that keeps track of the training steps.
- **checkpoint** (`tf.train.Checkpoint`) – checkpoint to use to keep track of models status.

Return type `None`

discriminator_loss

Retrieve the discriminator loss.

Return type `Optional[Executor]`

discriminator_model

Retrieve the discriminator model.

Return type `Model`

Returns `tf.keras.Model`.

fake_samples

Retrieve the fake samples, i.e. output of the generator.

Return type `Optional[Tensor]`

generator_inputs

Retrieve the generator inputs.

Return type `Optional[Tensor]`

generator_loss

Retrieve the generator loss.

Return type `Optional[Executor]`

generator_model

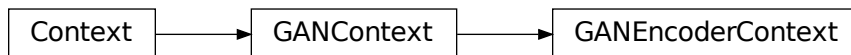
Retrieve the generator model.

Return type `Model`

Returns `tf.keras.Model`.

GANEncoderContext

Inheritance Diagram



```

class ashpy.contexts.gan.GANEncoderContext (dataset=None, generator_model=None,
                                             discriminator_model=None, encoder_model=None,
                                             generator_loss=None, discriminator_loss=None,
                                             encoder_loss=None, metrics=None,
                                             log_eval_mode=<LogEvalMode.TRAIN: 2>,
                                             global_step=<tf.Variable 'global_step:0'
                                             shape=() dtype=int64, numpy=0>,
                                             checkpoint=None)
    
```

Bases: `ashpy.contexts.gan.GANContext`

`ashpy.contexts.gan.GANEncoderContext` measure the specified metrics on the GAN.

Methods

<code>__init__</code> ([dataset, generator_model, ...])	Initialize the Context.
---	-------------------------

Attributes

<code>current_batch</code>	Return the current batch.
<code>dataset</code>	Retrieve the dataset.
<code>discriminator_loss</code>	Retrieve the discriminator loss.
<code>discriminator_model</code>	Retrieve the discriminator model.
<code>encoder_inputs</code>	Retrieve the inputs of the encoder.
<code>encoder_loss</code>	Retrieve the encoder loss.
<code>encoder_model</code>	Retrieve the encoder model.
<code>exception</code>	Return the exception.
<code>fake_samples</code>	Retrieve the fake samples, i.e.
<code>generator_inputs</code>	Retrieve the generator inputs.

Continued on next page

Table 60 – continued from previous page

<code>generator_loss</code>	Retrieve the generator loss.
<code>generator_model</code>	Retrieve the generator model.
<code>generator_of_encoder</code>	Retrieve the images generated from the encoder output.
<code>global_step</code>	Retrieve the <code>global_step</code> .
<code>log_eval_mode</code>	Retrieve model(s) mode.
<code>metrics</code>	Retrieve the metrics.

`__init__` (*dataset=None, generator_model=None, discriminator_model=None, encoder_model=None, generator_loss=None, discriminator_loss=None, encoder_loss=None, metrics=None, log_eval_mode=<LogEvalMode.TRAIN: 2>, global_step=<tf.Variable 'global_step:0' shape=() dtype=int64, numpy=0>, checkpoint=None*)
Initialize the Context.

Parameters

- **dataset** (`tf.data.Dataset`) – Dataset of tuples. [0] true dataset, [1] generator input dataset.
- **generator_model** (`tf.keras.Model`) – The generator.
- **discriminator_model** (`tf.keras.Model`) – The discriminator.
- **encoder_model** (`tf.keras.Model`) – The encoder.
- **generator_loss** (`ashpy.losses.Executor()`) – The generator loss.
- **discriminator_loss** (`ashpy.losses.Executor()`) – The discriminator loss.
- **encoder_loss** (`ashpy.losses.Executor()`) – The encoder loss.
- **metrics** (list of [`ashpy.metrics.metric.Metric`]) – All the metrics to be used to evaluate the model.
- **log_eval_mode** (`ashpy.modes.LogEvalMode`) – Models' mode to use when evaluating and logging.
- **global_step** (`tf.Variable`) – `tf.Variable` that keeps track of the training steps.
- **checkpoint** (`tf.train.Checkpoint`) – checkpoint to use to keep track of models status.

Return type `None`

`encoder_inputs`

Retrieve the inputs of the encoder.

Return type `Tensor`

`encoder_loss`

Retrieve the encoder loss.

Return type Optional[`Executor`]

`encoder_model`

Retrieve the encoder model.

Return type `Model`

Returns `tf.keras.Model`.

`generator_of_encoder`

Retrieve the images generated from the encoder output.

Return type Tensor

```
class ashpy.contexts.gan.GANContext (dataset=None, generator_model=None, dis-
    criminator_model=None, generator_loss=None,
    discriminator_loss=None, metrics=None,
    log_eval_mode=<LogEvalMode.TRAIN: 2>,
    global_step=<tf.Variable 'global_step:0' shape=()
    dtype=int64, numpy=0>, checkpoint=None)
```

Bases: `ashpy.contexts.context.Context`

`ashpy.contexts.gan.GANContext` measure the specified metrics on the GAN.

```
__init__ (dataset=None, generator_model=None, discriminator_model=None, generator_loss=None,
    discriminator_loss=None, metrics=None, log_eval_mode=<LogEvalMode.TRAIN: 2>,
    global_step=<tf.Variable 'global_step:0' shape=() dtype=int64, numpy=0>, check-
    point=None)
```

Initialize the Context.

Parameters

- **dataset** (`tf.data.Dataset`) – Dataset of tuples. [0] true dataset, [1] generator input dataset.
- **generator_model** (`tf.keras.Model`) – The generator.
- **discriminator_model** (`tf.keras.Model`) – The discriminator.
- **generator_loss** (`ashpy.losses.Executor()`) – The generator loss.
- **discriminator_loss** (`ashpy.losses.Executor()`) – The discriminator loss.
- **metrics** (list of [`ashpy.metrics.metric.Metric`]) – All the metrics to be used to evaluate the model.
- **log_eval_mode** (`ashpy.modes.LogEvalMode`) – Models' mode to use when evaluating and logging.
- **global_step** (`tf.Variable`) – `tf.Variable` that keeps track of the training steps.
- **checkpoint** (`tf.train.Checkpoint`) – checkpoint to use to keep track of models status.

Return type None

discriminator_loss

Retrieve the discriminator loss.

Return type Optional[`Executor`]

discriminator_model

Retrieve the discriminator model.

Return type Model

Returns `tf.keras.Model`.

fake_samples

Retrieve the fake samples, i.e. output of the generator.

Return type Optional[`Tensor`]

generator_inputs

Retrieve the generator inputs.

Return type Optional[`Tensor`]

generator_loss

Retrieve the generator loss.

Return type Optional[*Executor*]

generator_model

Retrieve the generator model.

Return type Model

Returns *tf.keras.Model*.

```
class ashpy.contexts.gan.GANEncoderContext (dataset=None, generator_model=None,
                                           discriminator_model=None, en-
                                           coder_model=None, generator_loss=None,
                                           discriminator_loss=None, en-
                                           coder_loss=None, metrics=None,
                                           log_eval_mode=<LogEvalMode.TRAIN:
                                           2>, global_step=<tf.Variable 'global_step:0'
                                           shape=() dtype=int64, numpy=0>, check-
                                           point=None)
```

Bases: *ashpy.contexts.gan.GANContext*

ashpy.contexts.gan.GANEncoderContext measure the specified metrics on the GAN.

```
__init__ (dataset=None, generator_model=None, discriminator_model=None, en-
           coder_model=None, generator_loss=None, discriminator_loss=None, encoder_loss=None,
           metrics=None, log_eval_mode=<LogEvalMode.TRAIN: 2>, global_step=<tf.Variable
           'global_step:0' shape=() dtype=int64, numpy=0>, checkpoint=None)
Initialize the Context.
```

Parameters

- **dataset** (*tf.data.Dataset*) – Dataset of tuples. [0] true dataset, [1] generator input dataset.
- **generator_model** (*tf.keras.Model*) – The generator.
- **discriminator_model** (*tf.keras.Model*) – The discriminator.
- **encoder_model** (*tf.keras.Model*) – The encoder.
- **generator_loss** (*ashpy.losses.Executor()*) – The generator loss.
- **discriminator_loss** (*ashpy.losses.Executor()*) – The discriminator loss.
- **encoder_loss** (*ashpy.losses.Executor()*) – The encoder loss.
- **metrics** (list of [*ashpy.metrics.metric.Metric*]) – All the metrics to be used to evaluate the model.
- **log_eval_mode** (*ashpy.modes.LogEvalMode*) – Models' mode to use when evaluating and logging.
- **global_step** (*tf.Variable*) – *tf.Variable* that keeps track of the training steps.
- **checkpoint** (*tf.train.Checkpoint*) – checkpoint to use to keep track of models status.

Return type None

encoder_inputs

Retrieve the inputs of the encoder.

Return type Tensor

encoder_loss

Retrieve the encoder loss.

Return type Optional[*Executor*]

encoder_model

Retrieve the encoder model.

Return type Model

Returns `tf.keras.Model`.

generator_of_encoder

Retrieve the images generated from the encoder output.

Return type Tensor

4.4 ashpy.keras

Custom extensions of standard Keras components.

Modules

<i>losses</i>	Custom Keras losses, used by the AshPy executors.
---------------	---

4.4.1 losses

Custom Keras losses, used by the AshPy executors.

Classes

<i>DHingeLoss</i>	Discriminator Hinge Loss as Keras Metric.
<i>DLeastSquare</i>	Discriminator Least Square Loss as <code>tf.keras.losses.Loss</code> .
<i>DMinMax</i>	Implementation of MinMax Discriminator loss as <code>tf.keras.losses.Loss</code> .
<i>GHingeLoss</i>	Generator Hinge Loss as Keras Metric.
<i>L1</i>	L1 Loss implementation as <code>tf.keras.losses.Loss</code> .

DHingeLoss

Inheritance Diagram



class ashpy.keras.losses.DHingeLoss

Bases: tensorflow.python.keras.losses.Loss

Discriminator Hinge Loss as Keras Metric.

See Geometric GAN¹ for more details.

The Discriminator Hinge loss is the hinge version of the adversarial loss. The Hinge loss is defined as:

$$L_{\text{hinge}} = \max(0, 1 - ty)$$

where y is the Discriminator output and t is the target class (+1 or -1 in the case of binary classification).

For the case of GANs:

$$L_{D_{\text{hinge}}} = -\mathbb{E}_{(x,y) \sim p_{\text{data}}}[\min(0, -1 + D(x, y))] - \mathbb{E}_{x \sim p_x, y \sim p_{\text{data}}}[\min(0, -1 - D(G(z), y))]$$

Methods

<code>__init__()</code>	Initialize the Loss.
<code>call(d_real, d_fake)</code>	Compute the hinge loss.

Attributes

<code>reduction</code>	Return the current <i>reduction</i> for this type of loss.
------------------------	--

`__init__()`

Initialize the Loss.

Return type None

`call(d_real, d_fake)`

Compute the hinge loss.

Return type Tensor

reduction

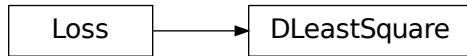
Return the current *reduction* for this type of loss.

Return type ReductionV2

¹ Geometric GAN <https://arxiv.org/abs/1705.02894>

DLeastSquare

Inheritance Diagram



class `ashpy.keras.losses.DLeastSquare`

Bases: `tensorflow.python.keras.losses.Loss`

Discriminator Least Square Loss as `tf.keras.losses.Loss`.

Methods

<code>__init__()</code>	Least square Loss for Discriminator.
<code>call(d_real, d_fake)</code>	Compute the Least Square Loss.

Attributes

<code>reduction</code>	Return the reduction type for this loss.
------------------------	--

`__init__()`

Least square Loss for Discriminator.

Reference: Least Squares Generative Adversarial Networks¹.

Basically the Mean Squared Error between the discriminator output when evaluated in fake samples and 0 and the discriminator output when evaluated in real samples and 1: For the unconditioned case this is:

$$L_D = \frac{1}{2} E[(D(x) - 1)^2 + (0 - D(G(z)))^2]$$

where x are real samples and z is the latent vector.

For the conditioned case this is:

$$L_D = \frac{1}{2} E[(D(x, c) - 1)^2 + (0 - D(G(c), c))^2]$$

where c is the condition and x are real samples.

Return type `None`

call (`d_real`, `d_fake`)

Compute the Least Square Loss.

Parameters

¹ Least Squares Generative Adversarial Networks <https://arxiv.org/abs/1611.04076>

- **d_real** (`tf.Tensor`) – Discriminator evaluated in real samples.
- **d_fake** (`tf.Tensor`) – Discriminator evaluated in fake samples.

Return type `Tensor`

Returns `tf.Tensor` – Loss.

reduction

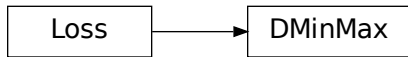
Return the reduction type for this loss.

Return type `ReductionV2`

Returns `tf.keras.losses.Reduction` – Reduction.

DMinMax

Inheritance Diagram



class `ashpy.keras.losses.DMinMax` (*from_logits=True, label_smoothing=0.0*)

Bases: `tensorflow.python.keras.losses.Loss`

Implementation of MinMax Discriminator loss as `tf.keras.losses.Loss`.

$$L_D = -\frac{1}{2}E[\log(D(x)) + \log(1 - D(G(z)))]$$

Methods

<code>__init__</code> ([<i>from_logits</i> , <i>label_smoothing</i>])	Initialize the loss.
<code>call</code> (<i>d_real</i> , <i>d_fake</i>)	Compute the MinMax Loss.

Attributes

<code>reduction</code>	Return the reduction type of this loss.
------------------------	---

`__init__` (*from_logits=True, label_smoothing=0.0*)

Initialize the loss.

Return type `None`

`call` (*d_real*, *d_fake*)

Compute the MinMax Loss.

Play the DiscriminatorMinMax game between the discriminator computed in real and the discriminator

compute with fake inputs.

Parameters

- **d_real** (`tf.Tensor`) – Real data.
- **d_fake** (`tf.Tensor`) – Fake (generated) data.

Return type `Tensor`

Returns `tf.Tensor` – Output Tensor.

reduction

Return the reduction type of this loss.

Return type `ReductionV2`

Returns `:py:classes:'tf.keras.losses.Reduction'` – Reduction.

GHingeLoss

Inheritance Diagram



class `ashpy.keras.losses.GHingeLoss`

Bases: `tensorflow.python.keras.losses.Loss`

Generator Hinge Loss as Keras Metric.

See Geometric GAN¹ for more details. The Generator Hinge loss is the hinge version of the adversarial loss. The Hinge loss is defined as:

$$L_{\text{hinge}} = \max(0, 1 - ty)$$

where y is the Discriminator output and t is the target class (+1 or -1 in the case of binary classification). The target class of the generated images is +1.

For the case of GANs

$$L_{G_{\text{hinge}}} = -\mathbb{E}_{(x \sim p_x, y \sim p_{data})} [\min(0, -1 + D(G(x), y))]$$

This can be simply approximated as:

$$L_{G_{\text{hinge}}} = -\mathbb{E}_{(x \sim p_x, y \sim p_{data})} [D(G(x), y)]$$

Methods

¹ Geometric GAN <https://arxiv.org/abs/1705.02894>

<code>__init__()</code>	Initialize the Loss.
<code>call(d_real, d_fake)</code>	Compute the hinge loss.

Attributes

<code>reduction</code>	Return the current <i>reduction</i> for this type of loss.
------------------------	--

`__init__()`
Initialize the Loss.

Return type None

`call(d_real, d_fake)`
Compute the hinge loss.

Return type Tensor

`reduction`
Return the current *reduction* for this type of loss.

Return type ReductionV2

L1

Inheritance Diagram



```

class ashpy.keras.losses.L1
    Bases: tensorflow.python.keras.losses.Loss
    L1 Loss implementation as tf.keras.losses.Loss.

```

Methods

<code>__init__()</code>	Initialize the Loss.
<code>call(x, y)</code>	Compute the mean of the l1 between x and y.

Attributes

<code>reduction</code>	Return the current <i>reduction</i> for this type of loss.
------------------------	--

`__init__()`

Initialize the Loss.

Return type None

`call(x, y)`

Compute the mean of the l1 between x and y.

Return type Tensor

reduction

Return the current *reduction* for this type of loss.

Return type ReductionV2

class `ashpy.keras.losses.DHingeLoss`

Bases: `tensorflow.python.keras.losses.Loss`

Discriminator Hinge Loss as Keras Metric.

See Geometric GAN [1]_ for more details.

The Discriminator Hinge loss is the hinge version of the adversarial loss. The Hinge loss is defined as:

$$L_{\text{hinge}} = \max(0, 1 - ty)$$

where y is the Discriminator output and t is the target class (+1 or -1 in the case of binary classification).

For the case of GANs:

$$L_{D_{\text{hinge}}} = -\mathbb{E}_{(x,y) \sim p_{data}}[\min(0, -1 + D(x, y))] - \mathbb{E}_{x \sim p_x, y \sim p_{data}}[\min(0, -1 - D(G(z), y))]$$

`__init__()`

Initialize the Loss.

Return type None

`call(d_real, d_fake)`

Compute the hinge loss.

Return type Tensor

reduction

Return the current *reduction* for this type of loss.

Return type ReductionV2

class `ashpy.keras.losses.DLeastSquare`

Bases: `tensorflow.python.keras.losses.Loss`

Discriminator Least Square Loss as `tf.keras.losses.Loss`.

`__init__()`

Least square Loss for Discriminator.

Reference: Least Squares Generative Adversarial Networks [1]_.

Basically the Mean Squared Error between the discriminator output when evaluated in fake samples and 0 and the discriminator output when evaluated in real samples and 1: For the unconditioned case this is:

$$L_D = \frac{1}{2} E[(D(x) - 1)^2 + (0 - D(G(z)))^2]$$

where x are real samples and z is the latent vector.

For the conditioned case this is:

$$L_D = \frac{1}{2} E[(D(x, c) - 1)^2 + (0 - D(G(c), c)^2]$$

where c is the condition and x are real samples.

Return type None

call (*d_real*, *d_fake*)

Compute the Least Square Loss.

Parameters

- **d_real** (`tf.Tensor`) – Discriminator evaluated in real samples.
- **d_fake** (`tf.Tensor`) – Discriminator evaluated in fake samples.

Return type Tensor

Returns `tf.Tensor` – Loss.

reduction

Return the reduction type for this loss.

Return type ReductionV2

Returns `tf.keras.losses.Reduction` – Reduction.

class `ashpy.keras.losses.DMinMax` (*from_logits=True*, *label_smoothing=0.0*)

Bases: `tensorflow.python.keras.losses.Loss`

Implementation of MinMax Discriminator loss as `tf.keras.losses.Loss`.

$$L_D = -\frac{1}{2} E[\log(D(x)) + \log(1 - D(G(z)))]$$

__init__ (*from_logits=True*, *label_smoothing=0.0*)

Initialize the loss.

Return type None

call (*d_real*, *d_fake*)

Compute the MinMax Loss.

Play the DiscriminatorMinMax game between the discriminator computed in real and the discriminator compute with fake inputs.

Parameters

- **d_real** (`tf.Tensor`) – Real data.
- **d_fake** (`tf.Tensor`) – Fake (generated) data.

Return type Tensor

Returns `tf.Tensor` – Output Tensor.

reduction

Return the reduction type of this loss.

Return type ReductionV2

Returns **:py:classes:‘tf.keras.losses.Reduction‘** – Reduction.

class ashpy.keras.losses.GHingeLoss

Bases: tensorflow.python.keras.losses.Loss

Generator Hinge Loss as Keras Metric.

See Geometric GAN [1] for more details. The Generator Hinge loss is the hinge version of the adversarial loss. The Hinge loss is defined as:

$$L_{\text{hinge}} = \max(0, 1 - ty)$$

where y is the Discriminator output and t is the target class (+1 or -1 in the case of binary classification). The target class of the generated images is +1.

For the case of GANs

$$L_{G_{\text{hinge}}} = -\mathbb{E}_{(x \sim p_x, y \sim p_{data})} [\min(0, -1 + D(G(x), y))]$$

This can be simply approximated as:

$$L_{G_{\text{hinge}}} = -\mathbb{E}_{(x \sim p_x, y \sim p_{data})} [D(G(x), y)]$$

__init__()

Initialize the Loss.

Return type None

call(d_real, d_fake)

Compute the hinge loss.

Return type Tensor

reduction

Return the current *reduction* for this type of loss.

Return type ReductionV2

class ashpy.keras.losses.L1

Bases: tensorflow.python.keras.losses.Loss

L1 Loss implementation as tf.keras.losses.Loss.

__init__()

Initialize the Loss.

Return type None

call(x, y)

Compute the mean of the l1 between x and y.

Return type Tensor

reduction

Return the current *reduction* for this type of loss.

Return type ReductionV2

4.5 ashpy.layers

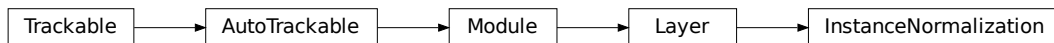
Collection of layers.

Layers

<code>instance_normalization.</code>	Instance Normalization Layer (used by Pix2Pix [1]_ and
<code>InstanceNormalization</code>	Pix2PixHD [2]_).
<code>attention.Attention</code>	Attention Layer from Self-Attention GAN [1]_.

4.5.1 InstanceNormalization

Inheritance Diagram



```

class ashpy.layers.instance_normalization.InstanceNormalization(eps=1e-06,
                                                                beta_initializer='zeros',
                                                                gamma_initializer='ones')
  
```

Bases: tensorflow.python.keras.engine.base_layer.Layer

Instance Normalization Layer (used by Pix2Pix¹ and Pix2PixHD²).

Basically it's a normalization done at instance level. The implementation follows the basic implementation of the Batch Normalization Layer.

- Direct Usage:

```

x = tf.ones((1, 10, 10, 64))

# instantiate attention layer as model.
normalization = InstanceNormalization()

# evaluate passing x.
output = normalization(x)

# the output shape is.
# the same as the input shape.
print(output.shape)
  
```

- Inside a Model:

```

def MyModel():
    inputs = tf.keras.layers.Input(shape=[None, None, 64])
    normalization = InstanceNormalization()
    return tf.keras.Model(inputs=inputs,
        ↪ outputs=normalization(inputs))

x = tf.ones((1, 10, 10, 64))
  
```

(continues on next page)

¹ Image-to-Image Translation with Conditional Adversarial Networks <https://arxiv.org/abs/1611.07004>

² High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs <https://arxiv.org/abs/1711.11585>

(continued from previous page)

```
model = MyModel()
output = model(x)

print(output.shape)
```

```
(1, 10, 10, 64)
```

Methods

<code>__init__([eps, beta_initializer, ...])</code>	Initialize the layer.
<code>build(input_shape)</code>	Assemble the layer.
<code>call(inputs[, training])</code>	Perform the computation.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

```
__init__(eps=1e-06, beta_initializer='zeros', gamma_initializer='ones')
    Initialize the layer.
```

Parameters

- **eps** (*float*) – Variance_epsilon used by batch_norm layer.
- **beta_initializer** (*str*) – Initializer for the beta variable.
- **gamma_initializer** (*str*) – Initializer for the gamma variable.

Return type None

build (*input_shape*)

Assemble the layer.

Parameters **input_shape** (*tuple* of (*int*)) – Specifies the shape of the input accepted by the layer.

Return type None

call (*inputs, training=False*)

Perform the computation.

Parameters

- **inputs** (*tf.Tensor*) – Inputs for the computation.
- **training** (*bool*) – Controls for training or evaluation mode.

Return type Tensor

Returns *tf.Tensor* – Output Tensor.

4.5.2 Attention

Inheritance Diagram



class ashpy.layers.attention.**Attention** (*filters*)

Bases: tensorflow.python.keras.engine.training.Model

Attention Layer from Self-Attention GAN¹.

First we extract features from the previous layer:

$$f(x) = W_f x$$

$$g(x) = W_g x$$

$$h(x) = W_h x$$

¹ Self-Attention Generative Adversarial Networks <https://arxiv.org/abs/1805.08318>

Then we calculate the importance matrix:

$$\beta_{j,i} = \frac{\exp(s_{i,j})}{\sum_{i=1}^N \exp(s_{ij})}$$

$\beta_{j,i}$ indicates the extent to which the model attends to the i^{th} location when synthethizing the j^{th} region.

Then we calculate the output of the attention layer $(o_1, \dots, o_N) \in \mathbb{R}^{C \times N}$:

$$o_j = \sum_{i=1}^N \beta_{j,i} h(x_i)$$

Finally we combine the (scaled) attention and the input to get the final output of the layer:

$$y_i = \gamma o_i + x_i$$

where γ is initialized as 0.

Examples

- Direct Usage:

```
x = tf.ones((1, 10, 10, 64))

# instantiate attention layer as model
attention = Attention(64)

# evaluate passing x
output = attention(x)

# the output shape is
# the same as the input shape
print(output.shape)
```

- Inside a Model:

```
def MyModel():
    inputs = tf.keras.layers.Input(shape=[None, None, 64])
    attention = Attention(64)
    return tf.keras.Model(inputs=inputs, outputs=attention(inputs))

x = tf.ones((1, 10, 10, 64))
model = MyModel()
output = model(x)

print(output.shape)
```

```
(1, 10, 10, 64)
```

Methods

<code>__init__(filters)</code>	Build the Attention Layer.
<code>call(inputs[, training])</code>	Perform the computation.

Attributes

activity_regularizer	Optional regularizer function for the output of this layer.
dtype	
dynamic	
inbound_nodes	Deprecated, do NOT use! Only for compatibility with external Keras.
input	Retrieves the input tensor(s) of a layer.
input_mask	Retrieves the input mask tensor(s) of a layer.
input_shape	Retrieves the input shape(s) of a layer.
input_spec	Gets the network's input specs.
layers	
losses	Losses which are associated with this <i>Layer</i> .
metrics	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
metrics_names	Returns the model's display labels for all outputs.
name	Returns the name of this module as passed or determined in the ctor.
name_scope	Returns a <i>tf.name_scope</i> instance for this class.
non_trainable_variables	
non_trainable_weights	
outbound_nodes	Deprecated, do NOT use! Only for compatibility with external Keras.
output	Retrieves the output tensor(s) of a layer.
output_mask	Retrieves the output mask tensor(s) of a layer.
output_shape	Retrieves the output shape(s) of a layer.
run_eagerly	Settable attribute indicating whether the model should run eagerly.
sample_weights	
state_updates	Returns the <i>updates</i> from all layers that are stateful.
stateful	
submodules	Sequence of all sub-modules.
trainable	
trainable_variables	Sequence of variables owned by this module and it's submodules.
trainable_weights	
updates	
variables	Returns the list of all layer variables/weights.
weights	Returns the list of all layer variables/weights.

__init__ (*filters*)

Build the Attention Layer.

Parameters **filters** (*int*) – Number of filters of the input tensor. It should be preferably a multiple of 8.

Return type None

call (*inputs*, *training=False*)

Perform the computation.

Parameters

- **inputs** (`tf.Tensor`) – Inputs for the computation.
- **training** (`bool`) – Controls for training or evaluation mode.

Return type `Tensor`

Returns `tf.Tensor` – Output Tensor.

4.6 ashpy.losses

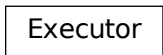
Collection of Losses.

Executor

<code>executor.Executor</code>	Carry a function and the way of executing it.
<code>executor.SumExecutor</code>	The sum executor.

4.6.1 Executor

Inheritance Diagram



class `ashpy.losses.executor.Executor` (`fn=None`)

Bases: `object`

Carry a function and the way of executing it. Given a context.

Methods

<code>__init__</code> (<code>[fn]</code>)	Initialize the Executor.
<code>call</code> (<code>context, **kwargs</code>)	Execute the function, using the information provided by the context.
<code>reduce_loss</code> (<code>call_fn</code>)	Create a Decorator to reduce Losses.

Attributes

<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.

Continued on next page

Table 80 – continued from previous page

<i>weight</i>	Return the loss weight.
__init__ (<i>fn=None</i>)	Initialize the Executor.
Parameters <i>fn</i> (<code>tf.keras.losses.Loss</code>)	– A Keras Loss to execute.
Return type	<code>None</code>
Returns	<code>None</code>
call (<i>context, **kwargs</i>)	Execute the function, using the information provided by the context.
Parameters <i>context</i> (<code>ashpy.contexts.Context</code>)	– The function execution Context.
Return type	<code>Tensor</code>
Returns	<code>tf.Tensor</code> – Output Tensor.
fn	Return the Keras loss function to execute.
Return type	<code>Loss</code>
Returns	<code>tf.keras.losses.Loss</code> – Keras Loss.
global_batch_size	Global batch size comprises the batch size for each cpu.
	Calculated as <code>batch_size_for_replica*replica_numbers</code> .
Return type	<code>int</code>
Returns	<code>int</code> – Global Batch size value.
static reduce_loss (<i>call_fn</i>)	Create a Decorator to reduce Losses. Used to simplify things.
	Apply a <code>reduce_sum</code> operation to the loss and divide the result by the batch size.
Parameters <i>call_fn</i> (<code>typing.Callable</code>)	– The executor call method.
Return type	<code>Callable</code>
Returns	<code>typing.Callable</code> – The decorated function.
weight	Return the loss weight.
	This weight is multiplied by the loss value. This is useful when working with multiples losses.
Return type	<code>Callable[...float]</code>
Returns	<code>typing.Callable</code> – Callable returning the weight (<code>float</code>).

4.6.2 SumExecutor

Inheritance Diagram



class `ashpy.losses.executor.SumExecutor` (*executors*)

Bases: `ashpy.losses.executor.Executor`

The sum executor. Executes the call of each fn and weights the losses.

Each Executor gets called (thus reducing its carried function), the results are then summed together.

Methods

<code>__init__(executors)</code>	Initialize the SumExecutor.
<code>call(*args, **kwargs)</code>	Evaluate and sum together the Executors.

Attributes

<code>executors</code>	Return the List of Executors.
<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__` (*executors*)

Initialize the SumExecutor.

Parameters `executors` (`list` of [`ashpy.executors.Executor`]) – Array of `ashpy.executors.Executor` to sum evaluate and sum together.

Return type `None`

Returns `None`

`call` (**args, **kwargs*)

Evaluate and sum together the Executors.

Return type `Tensor`

Returns `:py:classes:'tf.Tensor'` – Output Tensor.

`executors`

Return the List of Executors.

Return type `List[Executor]`

`global_batch_size`

Global batch size comprises the batch size for each cpu.

Calculated as `batch_size_for_replica*replica_numbers`.

Return type `int`

Returns `int` – Global Batch size value.

Classifier

`classifier.ClassifierLoss`

Classifier Loss Executor using the classifier model, instantiated with a fn.

4.6.3 ClassifierLoss

Inheritance Diagram



class `ashpy.losses.classifier.ClassifierLoss` (*fn*)

Bases: `ashpy.losses.executor.Executor`

Classifier Loss Executor using the classifier model, instantiated with a fn.

Methods

<code>__init__(fn)</code>	Initialize <code>ClassifierLoss</code> .
<code>call(*args, **kwargs)</code>	

Attributes

<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

__init__ (*fn*)
Initialize `ClassifierLoss`.

Parameters **fn** (`tf.keras.losses.Loss`) – Classification Loss function, should take as input labels and prediction.

Return type `None`

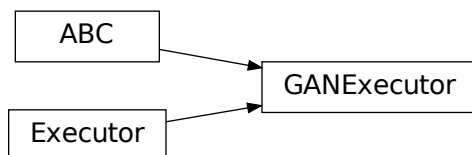
Returns None

GAN

<code>gan.GANExecutor</code>	Executor for GANs.
<code>gan.AdversarialLossType</code>	Enumeration for Adversarial Losses.
<code>gan.GeneratorAdversarialLoss</code>	Base class for the adversarial loss of the generator.
<code>gan.DiscriminatorAdversarialLoss</code>	Base class for the adversarial loss of the discriminator.
<code>gan.GeneratorBCE</code>	The Binary CrossEntropy computed among the generator and the 1 label.
<code>gan.GeneratorLSGAN</code>	Least Square GAN Loss for generator.
<code>gan.GeneratorL1</code>	L1 loss between the generator output and the target.
<code>gan.GeneratorHingeLoss</code>	Hinge loss for the Generator.
<code>gan.FeatureMatchingLoss</code>	Conditional GAN Feature matching loss.
<code>gan.CategoricalCrossEntropy</code>	Categorical Cross Entropy between generator output and target.
<code>gan.Pix2PixLoss</code>	Pix2Pix Loss.
<code>gan.Pix2PixLossSemantic</code>	Semantic Pix2Pix Loss.
<code>gan.EncoderBCE</code>	The Binary Cross Entropy computed among the encoder and the 0 label.
<code>gan.DiscriminatorMinMax</code>	The min-max game played by the discriminator.
<code>gan.DiscriminatorLSGAN</code>	Least square Loss for discriminator.
<code>gan.DiscriminatorHingeLoss</code>	Hinge loss for the Discriminator.
<code>gan.get_adversarial_loss_discriminator</code>	Return the correct loss for the Discriminator.
<code>gan.get_adversarial_loss_generator</code>	Return the correct loss for the Generator.

4.6.4 GANExecutor

Inheritance Diagram



```

class ashpy.losses.gan.GANExecutor (fn=None)
    Bases: ashpy.losses.executor.Executor, abc.ABC

    Executor for GANs.

    Implements the basic functions needed by the GAN losses.
  
```

Methods

<code>call(context, **kwargs)</code>	Execute the function, using the information provided by the context.
<code>get_discriminator_inputs(context, ...)</code>	Return the discriminator inputs.

Attributes

<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

call (*context*, ***kwargs*)

Execute the function, using the information provided by the context.

Parameters **context** (`ashpy.contexts.Context`) – The function execution Context.

Returns `tf.Tensor` – Output Tensor.

static get_discriminator_inputs (*context*, *fake_or_real*, *condition*, *training*)

Return the discriminator inputs. If needed it uses the encoder.

The current implementation uses the number of inputs to determine whether the discriminator is conditioned or not.

Parameters

- **context** (`ashpy.contexts.gan.GANContext`) – Context for GAN models.
- **fake_or_real** (`tf.Tensor`) – Discriminator input tensor, it can be fake (generated) or real.
- **condition** (`tf.Tensor`) – Discriminator condition (it can also be generator noise).
- **training** (`bool`) – whether is training phase or not

Return type `Union[Tensor, List[Tensor]]`

Returns The discriminator inputs.

4.6.5 AdversarialLossType

Inheritance Diagram



class ashpy.losses.gan.AdversarialLossType

Bases: `enum.Enum`

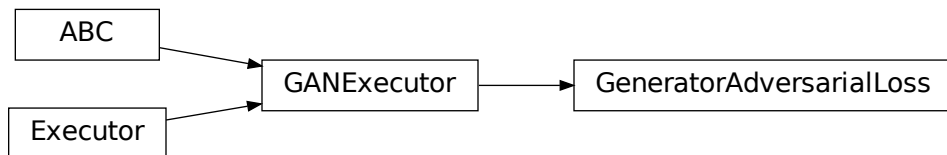
Enumeration for Adversarial Losses. Implemented: GAN and LSGAN.

Attributes

GAN
HINGE_LOSS
LSGAN

4.6.6 GeneratorAdversarialLoss

Inheritance Diagram



class ashpy.losses.gan.GeneratorAdversarialLoss (*loss_fn=None*)

Bases: `ashpy.losses.gan.GANExecutor`

Base class for the adversarial loss of the generator.

Methods

<code>__init__</code> ([<i>loss_fn</i>])	Initialize the Executor.
<code>call</code> (*args, **kwargs)	

Attributes

<i>fn</i>	Return the Keras loss function to execute.
<i>global_batch_size</i>	Global batch size comprises the batch size for each cpu.
<i>weight</i>	Return the loss weight.

`__init__` (*loss_fn=None*)
Initialize the Executor.

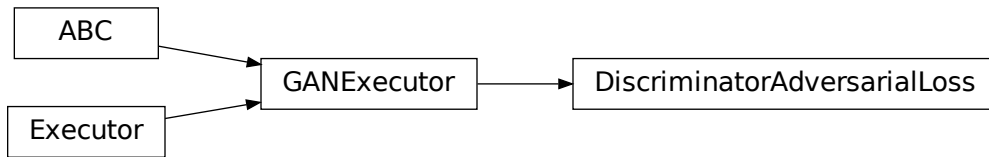
Parameters *loss_fn* (`tf.keras.losses.Loss`) – Keras Loss function to call passing

(tf.ones_like(d_fake_i), d_fake_i).

Return type None

4.6.7 DiscriminatorAdversarialLoss

Inheritance Diagram



class ashpy.losses.gan.DiscriminatorAdversarialLoss (*loss_fn=None*)

Bases: *ashpy.losses.gan.GANExecutor*

Base class for the adversarial loss of the discriminator.

Methods

<code>__init__([loss_fn])</code>	Initialize the Executor.
<code>call(*args, **kwargs)</code>	

Attributes

<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__ (loss_fn=None)`

Initialize the Executor.

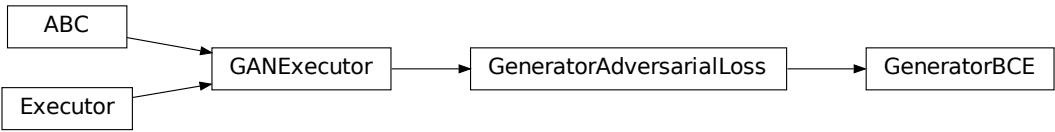
Parameters

- **loss_fn** (*tf.keras.losses.Loss*) – Loss function call passing
- **d_fake** (*(d_real,)*) –

Return type None

4.6.8 GeneratorBCE

Inheritance Diagram



class `ashpy.losses.gan.GeneratorBCE` (*from_logits=True*)
Bases: `ashpy.losses.gan.GeneratorAdversarialLoss`
The Binary CrossEntropy computed among the generator and the 1 label.

$$L_G = E[\log(D(G(z)))]$$

Methods

<code>__init__</code> ([from_logits])	Initialize the BCE Loss for the Generator.
---------------------------------------	--

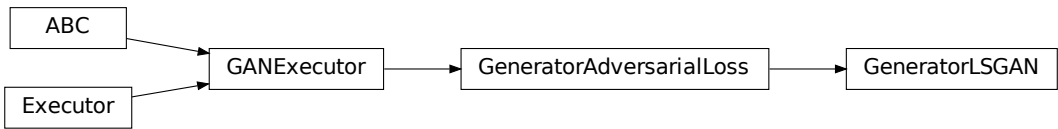
Attributes

<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__` (*from_logits=True*)
Initialize the BCE Loss for the Generator.
Return type None

4.6.9 GeneratorLSGAN

Inheritance Diagram



class `ashpy.losses.gan.GeneratorLSGAN`
Bases: `ashpy.losses.gan.GeneratorAdversarialLoss`
Least Square GAN Loss for generator.
Reference: <https://arxiv.org/abs/1611.04076>

Note: Basically the Mean Squared Error between the discriminator output when evaluated in fake and 1.

$$L_G = \frac{1}{2} E[(1 - D(G(z)))^2]$$

Methods

<code>__init__()</code>	Initialize the Least Square Loss for the Generator.
-------------------------	---

Attributes

<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__()`
Initialize the Least Square Loss for the Generator.
Return type `None`

4.6.10 GeneratorL1

Inheritance Diagram



class `ashpy.losses.gan.GeneratorL1`
Bases: `ashpy.losses.gan.GANExecutor`
L1 loss between the generator output and the target.

$$L_G = E||x - G(z)||_1$$

Where x is the target and G(z) is generated image.

Methods

<code>__init__()</code>	Initialize the Executor.
<code>call(*args, **kwargs)</code>	

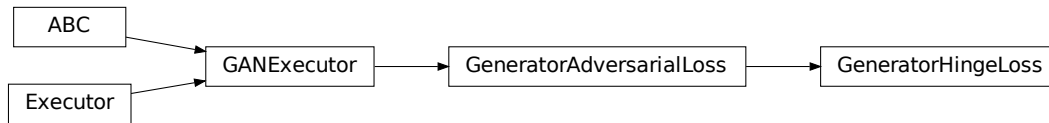
Attributes

<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__()`
Initialize the Executor.
Return type None

4.6.11 GeneratorHingeLoss

Inheritance Diagram



class `ashpy.losses.gan.GeneratorHingeLoss`

Bases: `ashpy.losses.gan.GeneratorAdversarialLoss`

Hinge loss for the Generator.

See Geometric GAN¹ for more details.

Methods

<code>__init__()</code>	Initialize the Least Square Loss for the Generator.
-------------------------	---

Attributes

<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

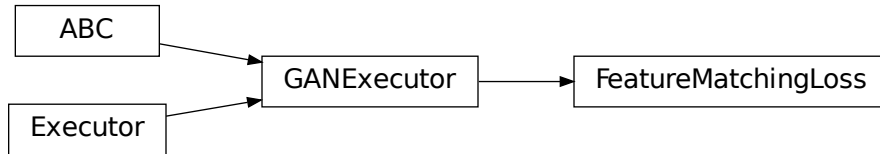
`__init__()`
Initialize the Least Square Loss for the Generator.

Return type None

4.6.12 FeatureMatchingLoss

¹ Geometric GAN <https://arxiv.org/abs/1705.02894>

Inheritance Diagram



class `ashpy.losses.gan.FeatureMatchingLoss`

Bases: `ashpy.losses.gan.GANExecutor`

Conditional GAN Feature matching loss.

The loss is computed for each example and it's the L1 (MAE) of the feature difference. Implementation of pix2pix HD: <https://github.com/NVIDIA/pix2pixHD>

$$FM = \sum_{i=0}^N \frac{1}{M_i} \|D_i(x, c) - D_i(G(c), c)\|_1$$

Where:

- D_i is the i -th layer of the discriminator
- N is the total number of layer of the discriminator
- M_i is the number of components for the i -th layer
- x is the target image
- c is the condition
- $G(c)$ is the generated image from the condition c
- $\| \cdot \|_1$ stands for norm 1.

This is for a single example: basically for each layer of the discriminator we compute the absolute error between the layer evaluated in real examples and in fake examples. Then we average along the batch. In the case where D_i is a multidimensional tensor we simply calculate the mean over the axis 1,2,3.

Methods

<code>__init__()</code>	Initialize the Executor.
<code>call(*args, **kwargs)</code>	

Attributes

<code>fn</code>	Return the Keras loss function to execute.
-----------------	--

Continued on next page

Table 103 – continued from previous page

<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__()`
Initialize the Executor.

Return type None

4.6.13 CategoricalCrossEntropy

Inheritance Diagram



class `ashpy.losses.gan.CategoricalCrossEntropy`

Bases: `ashpy.losses.executor.Executor`

Categorical Cross Entropy between generator output and target.

Useful when the output of the generator is a distribution over classes.

..note:: The target must be represented in one hot notation.

Methods

<code>__init__()</code>	Initialize the Categorical Cross Entropy Executor.
<code>call(*args, **kwargs)</code>	

Attributes

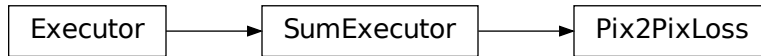
<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__()`
Initialize the Categorical Cross Entropy Executor.

Return type None

4.6.14 Pix2PixLoss

Inheritance Diagram



```

class ashpy.losses.gan.Pix2PixLoss (l1_loss_weight=100.0,      adversarial_loss_weight=1.0,
                                   feature_matching_weight=10.0,      adversar-
                                   ial_loss_type=<AdversarialLossType.GAN: 1>,
                                   use_feature_matching_loss=False)
  
```

Bases: *ashpy.losses.executor.SumExecutor*

Pix2Pix Loss.

Weighted sum of *ashpy.losses.gan.GeneratorL1*, *ashpy.losses.gan.AdversarialLossG* and *ashpy.losses.gan.FeatureMatchingLoss*.

Used by Pix2Pix [1] and Pix2PixHD [2]

Methods

<code>__init__</code> ([l1_loss_weight, ...])	Initialize the loss.
---	----------------------

Attributes

<code>executors</code>	Return the List of Executors.
<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

```

__init__ (l1_loss_weight=100.0, adversarial_loss_weight=1.0, feature_matching_weight=10.0, ad-
         versarial_loss_type=<AdversarialLossType.GAN: 1>, use_feature_matching_loss=False)
  Initialize the loss.
  
```

Weighted sum of *ashpy.losses.gan.GeneratorL1*, *ashpy.losses.gan.AdversarialLossG* and *ashpy.losses.gan.FeatureMatchingLoss*.

Parameters

- **l1_loss_weight** (*ashpy.ashtypes.TWeight*) – Weight of L1 loss.
- **adversarial_loss_weight** (*ashpy.ashtypes.TWeight*) – Weight of adversarial loss.
- **feature_matching_weight** (*ashpy.ashtypes.TWeight*) – Weight of the fea-

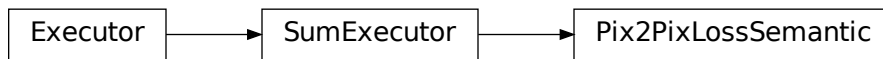
ture matching loss.

- **adversarial_loss_type** (*ashpy.losses.gan.AdversarialLossType*) – Adversarial loss type (*ashpy.losses.gan.AdversarialLossType.GAN* or *ashpy.losses.gan.AdversarialLossType.LSGAN*).
- **use_feature_matching_loss** (*bool*) – if True use also uses *ashpy.losses.gan.FeatureMatchingLoss*.

Return type *None*

4.6.15 Pix2PixLossSemantic

Inheritance Diagram



```

class ashpy.losses.gan.Pix2PixLossSemantic (cross_entropy_weight=100.0,          ad-
                                          versarial_loss_weight=1.0,          fea-
                                          ture_matching_weight=10.0,          adversar-
                                          ial_loss_type=<AdversarialLossType.GAN:
                                          1>, use_feature_matching_loss=False)
  
```

Bases: *ashpy.losses.executor.SumExecutor*

Semantic Pix2Pix Loss.

Weighted sum of *ashpy.losses.gan.CategoricalCrossEntropy*, *ashpy.losses.gan.AdversarialLossG* and *ashpy.losses.gan.FeatureMatchingLoss*.

Methods

<code>__init__</code> ([<i>cross_entropy_weight</i> , ...])	Initialize the Executor.
--	--------------------------

Attributes

<code>executors</code>	Return the List of Executors.
<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

```
__init__(cross_entropy_weight=100.0, adversarial_loss_weight=1.0, feature_matching_weight=10.0, adversarial_loss_type=<AdversarialLossType.GAN: 1>, use_feature_matching_loss=False)
```

Initialize the Executor.

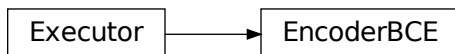
Weighted sum of `ashpy.losses.gan.CategoricalCrossEntropy`, `ashpy.losses.gan.AdversarialLossG` and `ashpy.losses.gan.FeatureMatchingLoss`

Parameters

- **cross_entropy_weight** (`ashpy.ashtypes.TWeight`) – Weight of the categorical cross entropy loss.
- **adversarial_loss_weight** (`ashpy.ashtypes.TWeight`) – Weight of the adversarial loss.
- **feature_matching_weight** (`ashpy.ashtypes.TWeight`) – Weight of the feature matching loss.
- **adversarial_loss_type** (`ashpy.losses.gan.AdversarialLossType`) – type of adversarial loss, see `ashpy.losses.gan.AdversarialLossType`
- **use_feature_matching_loss** (`bool`) – whether to use feature matching loss or not

4.6.16 EncoderBCE

Inheritance Diagram



```
class ashpy.losses.gan.EncoderBCE (from_logits=True)
```

Bases: `ashpy.losses.executor.Executor`

The Binary Cross Entropy computed among the encoder and the 0 label.

Methods

<code>__init__</code> ([from_logits])	Initialize the Executor.
<code>call</code> (*args, **kwargs)	

Attributes

<code>fn</code>	Return the Keras loss function to execute.
-----------------	--

Continued on next page

Table 111 – continued from previous page

<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__` (*from_logits=True*)

Initialize the Executor.

Return type None

4.6.17 DiscriminatorMinMax

Inheritance Diagram



class `ashpy.losses.gan.DiscriminatorMinMax` (*from_logits=True, label_smoothing=0.0*)

Bases: `ashpy.losses.gan.DiscriminatorAdversarialLoss`

The min-max game played by the discriminator.

$$L_D = -\frac{1}{2}E[\log(D(x)) + \log(1 - D(G(z)))]$$

Methods

<code>__init__</code> ([<i>from_logits</i> , <i>label_smoothing</i>])	Initialize Loss.
---	------------------

Attributes

<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__` (*from_logits=True, label_smoothing=0.0*)

Initialize Loss.

4.6.18 DiscriminatorLSGAN

Inheritance Diagram



class `ashpy.losses.gan.DiscriminatorLSGAN`

Bases: `ashpy.losses.gan.DiscriminatorAdversarialLoss`

Least square Loss for discriminator.

Reference: Least Squares Generative Adversarial Networks¹ .

Basically the Mean Squared Error between the discriminator output when evaluated in fake samples and 0 and the discriminator output when evaluated in real samples and 1: For the unconditioned case this is:

$$L_D = \frac{1}{2} E[(D(x) - 1)^2 + (0 - D(G(z)))^2]$$

where x are real samples and z is the latent vector.

For the conditioned case this is:

$$L_D = \frac{1}{2} E[(D(x, c) - 1)^2 + (0 - D(G(c), c))^2]$$

where c is the condition and x are real samples.

Methods

<code>__init__()</code>	Initialize loss.
-------------------------	------------------

Attributes

<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

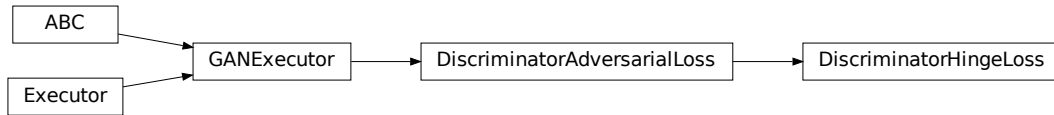
`__init__()`
Initialize loss.

Return type None

¹ Least Squares Generative Adversarial Networks <https://arxiv.org/abs/1611.04076>

4.6.19 DiscriminatorHingeLoss

Inheritance Diagram



class `ashpy.losses.gan.DiscriminatorHingeLoss`

Bases: `ashpy.losses.gan.DiscriminatorAdversarialLoss`

Hinge loss for the Discriminator.

See Geometric GAN¹ for more details.

Methods

<code>__init__()</code>	Initialize the Least Square Loss for the Generator.
-------------------------	---

Attributes

<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__()`
Initialize the Least Square Loss for the Generator.

Return type `None`

4.6.20 `ashpy.losses.gan.get_adversarial_loss_discriminator`

`ashpy.losses.gan.get_adversarial_loss_discriminator` (*adversarial_loss_type*=<`AdversarialLossType.GAN`: 1>)

Return the correct loss for the Discriminator.

Parameters `adversarial_loss_type` (`ashpy.losses.gan.AdversarialLossType`) – Type of loss (`ashpy.losses.gan.AdversarialLossType.GAN` or `ashpy.losses.gan.AdversarialLossType.LSGAN`)

Return type `Type[Executor]`

¹ Geometric GAN <https://arxiv.org/abs/1705.02894>

Returns The correct (*ashpy.losses.executor.Executor*) (to be instantiated).

4.6.21 ashpy.losses.gan.get_adversarial_loss_generator

ashpy.losses.gan.get_adversarial_loss_generator (*adversarial_loss_type*=<*AdversarialLossType.GAN*:
I>)

Return the correct loss for the Generator.

Parameters *adversarial_loss_type* (*ashpy.losses.AdversarialLossType*) –
Type of loss (*ashpy.losses.AdversarialLossType.GAN* or *ashpy.losses.*
AdversarialLossType.LSGAN).

Return type *Type[Executor]*

Returns The correct (*ashpy.losses.executor.Executor*) (to be instantiated).

Modules

<i>classifier</i>	The classification losses.
<i>executor</i>	The Executor.
<i>gan</i>	GAN losses.

4.6.22 classifier

The classification losses.

Classes

<i>ClassifierLoss</i>	Classifier Loss Executor using the classifier model, instantiated with a fn.
-----------------------	--

ClassifierLoss

Inheritance Diagram



class *ashpy.losses.classifier.ClassifierLoss* (*fn*)

Bases: *ashpy.losses.executor.Executor*

Classifier Loss Executor using the classifier model, instantiated with a fn.

Methods

<code>__init__(fn)</code>	Initialize <i>ClassifierLoss</i> .
<code>call(*args, **kwargs)</code>	

Attributes

<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__(fn)`
Initialize *ClassifierLoss*.

Parameters `fn` (`tf.keras.losses.Loss`) – Classification Loss function, should take as input labels and prediction.

Return type `None`

Returns `None`

class `ashpy.losses.classifier.ClassifierLoss(fn)`

Bases: *ashpy.losses.executor.Executor*

Classifier Loss Executor using the classifier model, instantiated with a fn.

`__init__(fn)`
Initialize *ClassifierLoss*.

Parameters `fn` (`tf.keras.losses.Loss`) – Classification Loss function, should take as input labels and prediction.

Return type `None`

Returns `None`

4.6.23 executor

The Executor.

An object that, given an `ashpy.contexts.Context`, carries a function and the way of executing it.

Classes

<i>Executor</i>	Carry a function and the way of executing it.
<i>SumExecutor</i>	The sum executor.

Executor

Inheritance Diagram

Executor

class ashpy.losses.executor.**Executor** (*fn=None*)

Bases: `object`

Carry a function and the way of executing it. Given a context.

Methods

<code>__init__</code> (<i>fn</i>)	Initialize the Executor.
<code>call</code> (<i>context</i> , <i>**kwargs</i>)	Execute the function, using the information provided by the context.
<code>reduce_loss</code> (<i>call_fn</i>)	Create a Decorator to reduce Losses.

Attributes

<i>fn</i>	Return the Keras loss function to execute.
<i>global_batch_size</i>	Global batch size comprises the batch size for each cpu.
<i>weight</i>	Return the loss weight.

`__init__` (*fn=None*)

Initialize the Executor.

Parameters *fn* (`tf.keras.losses.Loss`) – A Keras Loss to execute.

Return type `None`

Returns `None`

`call` (*context*, ***kwargs*)

Execute the function, using the information provided by the context.

Parameters *context* (`ashpy.contexts.Context`) – The function execution Context.

Return type `Tensor`

Returns `tf.Tensor` – Output Tensor.

fn

Return the Keras loss function to execute.

Return type `Loss`

Returns `tf.keras.losses.Loss` – Keras Loss.

global_batch_size

Global batch size comprises the batch size for each cpu.

Calculated as `batch_size_for_replica*replica_numbers`.

Return type `int`

Returns `int` – Global Batch size value.

static reduce_loss (*call_fn*)

Create a Decorator to reduce Losses. Used to simplify things.

Apply a `reduce sum` operation to the loss and divide the result by the batch size.

Parameters `call_fn` (`typing.Callable`) – The executor call method.

Return type `Callable`

Returns `typing.Callable` – The decorated function.

weight

Return the loss weight.

This weight is multiplied by the loss value. This is useful when working with multiples losses.

Return type `Callable[...float]`

Returns `typing.Callable` – Callable returning the weight (`float`).

SumExecutor

Inheritance Diagram



class `ashpy.losses.executor.SumExecutor` (*executors*)

Bases: `ashpy.losses.executor.Executor`

The sum executor. Executes the call of each fn and weights the losses.

Each Executor gets called (thus reducing its carried function), the results are then summed together.

Methods

<code>__init__</code> (executors)	Initialize the SumExecutor.
<code>call</code> (*args, **kwargs)	Evaluate and sum together the Executors.

Attributes

<code>executors</code>	Return the List of Executors.
<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__` (*executors*)

Initialize the SumExecutor.

Parameters `executors` (`list` of [`ashpy.executors.Executor`]) – Array of `ashpy.executors.Executor` to sum evaluate and sum together.

Return type `None`

Returns `None`

`call` (**args, **kwargs*)

Evaluate and sum together the Executors.

Return type `Tensor`

Returns `:py:classes:'tf.Tensor'` – Output Tensor.

`executors`

Return the List of Executors.

Return type `List[Executor]`

`global_batch_size`

Global batch size comprises the batch size for each cpu.

Calculated as `batch_size_for_replica*replica_numbers`.

Return type `int`

Returns `int` – Global Batch size value.

class `ashpy.losses.executor.Executor` (*fn=None*)

Bases: `object`

Carry a function and the way of executing it. Given a context.

`__init__` (*fn=None*)

Initialize the Executor.

Parameters `fn` (`tf.keras.losses.Loss`) – A Keras Loss to execute.

Return type `None`

Returns `None`

`call` (*context, **kwargs*)

Execute the function, using the information provided by the context.

Parameters `context` (`ashpy.contexts.Context`) – The function execution Context.

Return type `Tensor`

Returns `tf.Tensor` – Output Tensor.

`fn`

Return the Keras loss function to execute.

Return type `Loss`

Returns `tf.keras.losses.Loss` – Keras Loss.

global_batch_size

Global batch size comprises the batch size for each cpu.

Calculated as `batch_size_for_replica*replica_numbers`.

Return type `int`

Returns `int` – Global Batch size value.

static reduce_loss (*call_fn*)

Create a Decorator to reduce Losses. Used to simplify things.

Apply a `reduce` sum operation to the loss and divide the result by the batch size.

Parameters `call_fn` (`typing.Callable`) – The executor call method.

Return type `Callable`

Returns `typing.Callable` – The decorated function.

weight

Return the loss weight.

This weight is multiplied by the loss value. This is useful when working with multiples losses.

Return type `Callable[..., float]`

Returns `typing.Callable` – Callable returning the weight (`float`).

class `ashpy.losses.executor.SumExecutor` (*executors*)

Bases: `ashpy.losses.executor.Executor`

The sum executor. Executes the call of each fn and weights the losses.

Each Executor gets called (thus reducing its carried function), the results are then summed together.

__init__ (*executors*)

Initialize the SumExecutor.

Parameters `executors` (`list` of [`ashpy.executors.Executor`]) – Array of `ashpy.executors.Executor` to sum evaluate and sum together.

Return type `None`

Returns `None`

call (**args, **kwargs*)

Evaluate and sum together the Executors.

Return type `Tensor`

Returns **:py:classes:tf.Tensor** – Output Tensor.

executors

Return the List of Executors.

Return type `List[Executor]`

global_batch_size

Global batch size comprises the batch size for each cpu.

Calculated as `batch_size_for_replica*replica_numbers`.

Return type `int`

Returns `int` – Global Batch size value.

4.6.24 gan

GAN losses.

Functions

<code>get_adversarial_loss_discriminator</code>	Return the correct loss for the Discriminator.
<code>get_adversarial_loss_generator</code>	Return the correct loss for the Generator.

`ashpy.losses.gan.get_adversarial_loss_discriminator`

`ashpy.losses.gan.get_adversarial_loss_discriminator` (*adversarial_loss_type*=<*AdversarialLossType.GAN*:
1>)

Return the correct loss for the Discriminator.

Parameters *adversarial_loss_type* (*ashpy.losses.gan.AdversarialLossType*) – Type of loss (*ashpy.losses.gan.AdversarialLossType.GAN* or *ashpy.losses.gan.AdversarialLossType.LSGAN*)

Return type *Type[Executor]*

Returns The correct (*ashpy.losses.executor.Executor*) (to be instantiated).

`ashpy.losses.gan.get_adversarial_loss_generator`

`ashpy.losses.gan.get_adversarial_loss_generator` (*adversarial_loss_type*=<*AdversarialLossType.GAN*:
1>)

Return the correct loss for the Generator.

Parameters *adversarial_loss_type* (*ashpy.losses.AdversarialLossType*) – Type of loss (*ashpy.losses.AdversarialLossType.GAN* or *ashpy.losses.AdversarialLossType.LSGAN*).

Return type *Type[Executor]*

Returns The correct (*ashpy.losses.executor.Executor*) (to be instantiated).

Classes

<i>AdversarialLossType</i>	Enumeration for Adversarial Losses.
<i>CategoricalCrossEntropy</i>	Categorical Cross Entropy between generator output and target.
<i>DiscriminatorAdversarialLoss</i>	Base class for the adversarial loss of the discriminator.
<i>DiscriminatorHingeLoss</i>	Hinge loss for the Discriminator.
<i>DiscriminatorLSGAN</i>	Least square Loss for discriminator.
<i>DiscriminatorMinMax</i>	The min-max game played by the discriminator.
<i>EncoderBCE</i>	The Binary Cross Entropy computed among the encoder and the 0 label.
<i>FeatureMatchingLoss</i>	Conditional GAN Feature matching loss.
<i>GANExecutor</i>	Executor for GANs.
<i>GeneratorAdversarialLoss</i>	Base class for the adversarial loss of the generator.

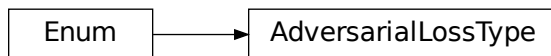
Continued on next page

Table 128 – continued from previous page

<i>GeneratorBCE</i>	The Binary CrossEntropy computed among the generator and the 1 label.
<i>GeneratorHingeLoss</i>	Hinge loss for the Generator.
<i>GeneratorL1</i>	L1 loss between the generator output and the target.
<i>GeneratorLSGAN</i>	Least Square GAN Loss for generator.
<i>Pix2PixLoss</i>	Pix2Pix Loss.
<i>Pix2PixLossSemantic</i>	Semantic Pix2Pix Loss.

AdversarialLossType

Inheritance Diagram



class ashpy.losses.gan.**AdversarialLossType**

Bases: `enum.Enum`

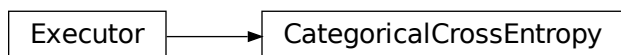
Enumeration for Adversarial Losses. Implemented: GAN and LSGAN.

Attributes

GAN
HINGE_LOSS
LSGAN

CategoricalCrossEntropy

Inheritance Diagram



class ashpy.losses.gan.**CategoricalCrossEntropy**

Bases: `ashpy.losses.executor.Executor`

Categorical Cross Entropy between generator output and target.

Useful when the output of the generator is a distribution over classes.

..note:: The target must be represented in one hot notation.

Methods

<code>__init__()</code>	Initialize the Categorical Cross Entropy Executor.
<code>call(*args, **kwargs)</code>	

Attributes

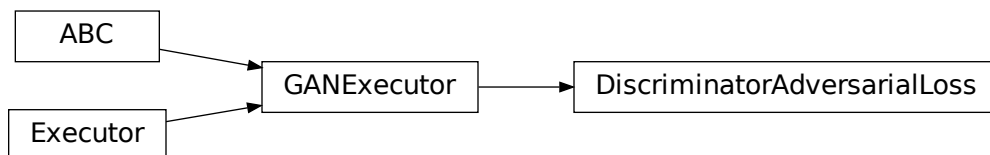
<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__()`
Initialize the Categorical Cross Entropy Executor.

Return type None

DiscriminatorAdversarialLoss

Inheritance Diagram



```
class ashpy.losses.gan.DiscriminatorAdversarialLoss (loss_fn=None)
```

Bases: `ashpy.losses.gan.GANExecutor`

Base class for the adversarial loss of the discriminator.

Methods

<code>__init__([loss_fn])</code>	Initialize the Executor.
<code>call(*args, **kwargs)</code>	

Attributes

<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__(loss_fn=None)`
Initialize the Executor.

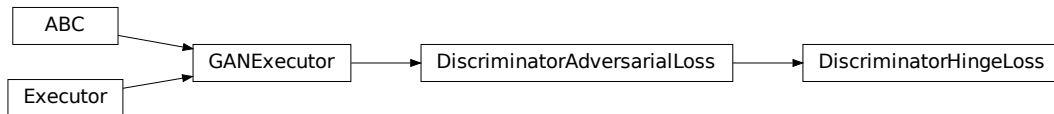
Parameters

- `loss_fn` (`tf.keras.losses.Loss`) – Loss function call passing
- `d_fake` (`(d_real,)`) –

Return type `None`

DiscriminatorHingeLoss

Inheritance Diagram



class `ashpy.losses.gan.DiscriminatorHingeLoss`

Bases: `ashpy.losses.gan.DiscriminatorAdversarialLoss`

Hinge loss for the Discriminator.

See Geometric GAN¹ for more details.

Methods

<code>__init__()</code>	Initialize the Least Square Loss for the Generator.
-------------------------	---

Attributes

<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.

Continued on next page

¹ Geometric GAN <https://arxiv.org/abs/1705.02894>

Table 135 – continued from previous page

weight	Return the loss weight.
--------	-------------------------

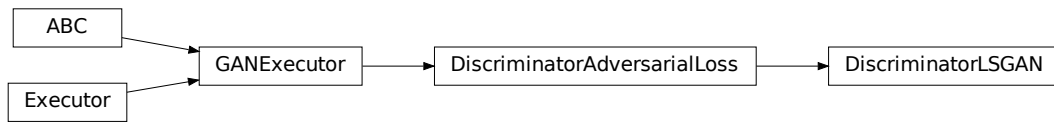
`__init__()`

Initialize the Least Square Loss for the Generator.

Return type None

DiscriminatorLSGAN

Inheritance Diagram



class `ashpy.losses.gan.DiscriminatorLSGAN`

Bases: `ashpy.losses.gan.DiscriminatorAdversarialLoss`

Least square Loss for discriminator.

Reference: Least Squares Generative Adversarial Networks¹.

Basically the Mean Squared Error between the discriminator output when evaluated in fake samples and 0 and the discriminator output when evaluated in real samples and 1: For the unconditioned case this is:

$$L_D = \frac{1}{2} E[(D(x) - 1)^2 + (0 - D(G(z)))^2]$$

where x are real samples and z is the latent vector.

For the conditioned case this is:

$$L_D = \frac{1}{2} E[(D(x, c) - 1)^2 + (0 - D(G(c), c))^2]$$

where c is the condition and x are real samples.

Methods

<code>__init__()</code>	Initialize loss.
-------------------------	------------------

Attributes

¹ Least Squares Generative Adversarial Networks <https://arxiv.org/abs/1611.04076>

<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

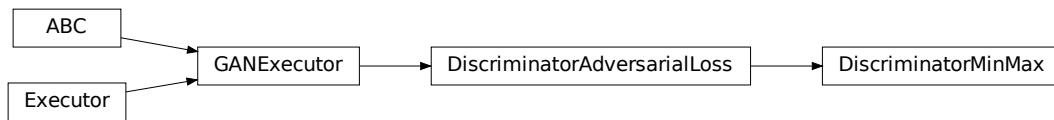
`__init__()`

Initialize loss.

Return type None

DiscriminatorMinMax

Inheritance Diagram



class `ashpy.losses.gan.DiscriminatorMinMax` (*from_logits=True, label_smoothing=0.0*)

Bases: `ashpy.losses.gan.DiscriminatorAdversarialLoss`

The min-max game played by the discriminator.

$$L_D = -\frac{1}{2}E[\log(D(x)) + \log(1 - D(G(z)))]$$

Methods

<code>__init__([from_logits, label_smoothing])</code>	Initialize Loss.
---	------------------

Attributes

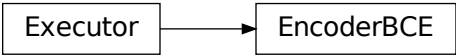
<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__` (*from_logits=True, label_smoothing=0.0*)

Initialize Loss.

EncoderBCE

Inheritance Diagram



class `ashpy.losses.gan.EncoderBCE` (*from_logits=True*)
Bases: `ashpy.losses.executor.Executor`
The Binary Cross Entropy computed among the encoder and the 0 label.

Methods

<code>__init__</code> ([from_logits])	Initialize the Executor.
<code>call</code> (*args, **kwargs)	

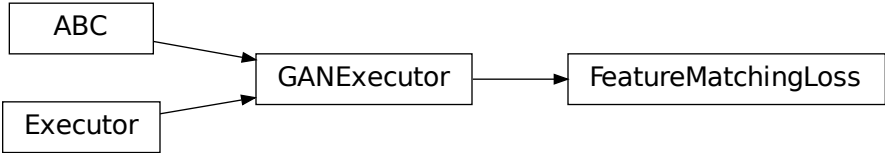
Attributes

<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__` (*from_logits=True*)
Initialize the Executor.
Return type None

FeatureMatchingLoss

Inheritance Diagram



class ashpy.losses.gan.FeatureMatchingLoss

Bases: *ashpy.losses.gan.GANExecutor*

Conditional GAN Feature matching loss.

The loss is computed for each example and it's the L1 (MAE) of the feature difference. Implementation of pix2pix HD: <https://github.com/NVIDIA/pix2pixHD>

$$FM = \sum_{i=0}^N \frac{1}{M_i} ||D_i(x, c) - D_i(G(c), c)||_1$$

Where:

- D_i is the i -th layer of the discriminator
- N is the total number of layer of the discriminator
- M_i is the number of components for the i -th layer
- x is the target image
- c is the condition
- $G(c)$ is the generated image from the condition c
- $|| \cdot ||_1$ stands for norm 1.

This is for a single example: basically for each layer of the discriminator we compute the absolute error between the layer evaluated in real examples and in fake examples. Then we average along the batch. In the case where D_i is a multidimensional tensor we simply calculate the mean over the axis 1,2,3.

Methods

<code>__init__()</code>	Initialize the Executor.
<code>call(*args, **kwargs)</code>	

Attributes

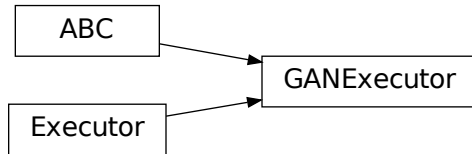
<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__()`
Initialize the Executor.

Return type None

GANExecutor

Inheritance Diagram



class `ashpy.losses.gan.GANExecutor` (*fn=None*)
 Bases: `ashpy.losses.executor.Executor`, `abc.ABC`

Executor for GANs.

Implements the basic functions needed by the GAN losses.

Methods

<code>call(context, **kwargs)</code>	Execute the function, using the information provided by the context.
<code>get_discriminator_inputs(context, ...)</code>	Return the discriminator inputs.

Attributes

<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

call (*context, **kwargs*)

Execute the function, using the information provided by the context.

Parameters **context** (`ashpy.contexts.Context`) – The function execution Context.

Returns `tf.Tensor` – Output Tensor.

static get_discriminator_inputs (*context, fake_or_real, condition, training*)

Return the discriminator inputs. If needed it uses the encoder.

The current implementation uses the number of inputs to determine whether the discriminator is conditioned or not.

Parameters

- **context** (`ashpy.contexts.gan.GANContext`) – Context for GAN models.
- **fake_or_real** (`tf.Tensor`) – Discriminator input tensor, it can be fake (generated) or real.

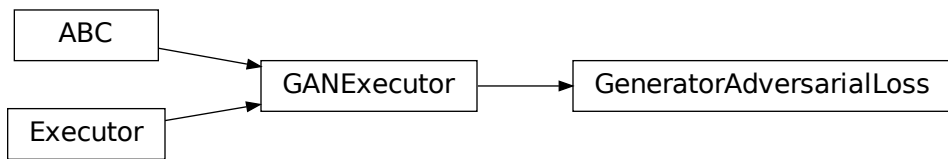
- **condition** (`tf.Tensor`) – Discriminator condition (it can also be generator noise).
- **training** (`bool`) – whether is training phase or not

Return type `Union[Tensor, List[Tensor]]`

Returns The discriminator inputs.

GeneratorAdversarialLoss

Inheritance Diagram



class `ashpy.losses.gan.GeneratorAdversarialLoss` (*loss_fn=None*)

Bases: `ashpy.losses.gan.GANExecutor`

Base class for the adversarial loss of the generator.

Methods

<code>__init__</code> (<i>[loss_fn]</i>)	Initialize the Executor.
<code>call</code> (*args, **kwargs)	

Attributes

<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

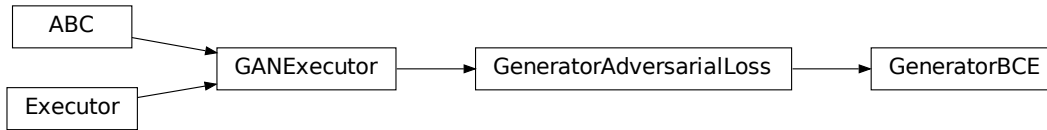
`__init__` (*loss_fn=None*)
Initialize the Executor.

Parameters `loss_fn` (`tf.keras.losses.Loss`) – Keras Loss function to call passing (`tf.ones_like(d_fake_i)`, `d_fake_i`).

Return type `None`

GeneratorBCE

Inheritance Diagram



class `ashpy.losses.gan.GeneratorBCE` (*from_logits=True*)
 Bases: `ashpy.losses.gan.GeneratorAdversarialLoss`
 The Binary CrossEntropy computed among the generator and the 1 label.

$$L_G = E[\log(D(G(z)))]$$

Methods

<code>__init__</code> ([from_logits])	Initialize the BCE Loss for the Generator.
---------------------------------------	--

Attributes

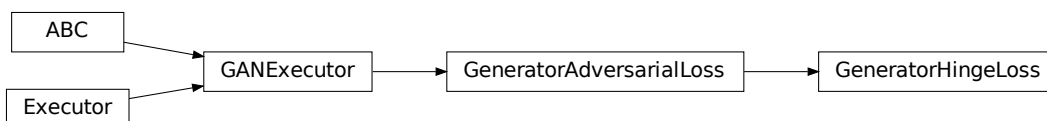
<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__` (*from_logits=True*)
 Initialize the BCE Loss for the Generator.

Return type `None`

GeneratorHingeLoss

Inheritance Diagram



class `ashpy.losses.gan.GeneratorHingeLoss`

Bases: `ashpy.losses.gan.GeneratorAdversarialLoss`

Hinge loss for the Generator.

See Geometric GAN¹ for more details.

Methods

<code>__init__()</code>	Initialize the Least Square Loss for the Generator.
-------------------------	---

Attributes

<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

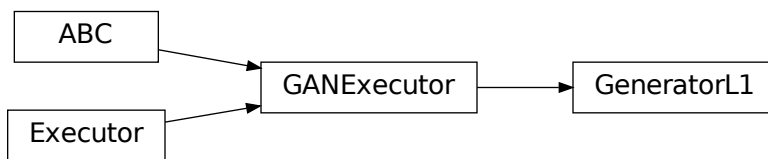
`__init__()`

Initialize the Least Square Loss for the Generator.

Return type None

GeneratorL1

Inheritance Diagram



class `ashpy.losses.gan.GeneratorL1`

Bases: `ashpy.losses.gan.GANExecutor`

L1 loss between the generator output and the target.

$$L_G = E||x - G(z)||_1$$

Where x is the target and G(z) is generated image.

¹ Geometric GAN <https://arxiv.org/abs/1705.02894>

Methods

<code>__init__()</code>	Initialize the Executor.
<code>call(*args, **kwargs)</code>	

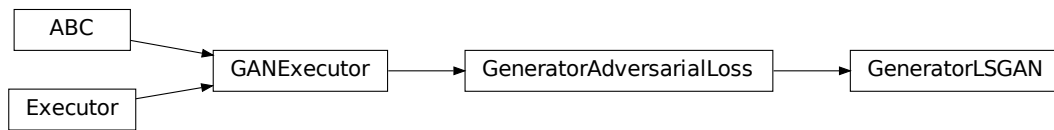
Attributes

<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__()`
Initialize the Executor.
Return type None

GeneratorLSGAN

Inheritance Diagram



class `ashpy.losses.gan.GeneratorLSGAN`
 Bases: `ashpy.losses.gan.GeneratorAdversarialLoss`
 Least Square GAN Loss for generator.
 Reference: <https://arxiv.org/abs/1611.04076>

Note: Basically the Mean Squared Error between the discriminator output when evaluated in fake and 1.

$$L_G = \frac{1}{2} E[(1 - D(G(z)))^2]$$

Methods

<code>__init__()</code>	Initialize the Least Square Loss for the Generator.
-------------------------	---

Attributes

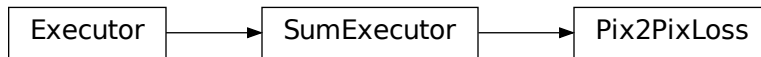
<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__()`
Initialize the Least Square Loss for the Generator.

Return type None

Pix2PixLoss

Inheritance Diagram



```

class ashpy.losses.gan.Pix2PixLoss (l1_loss_weight=100.0,      adversarial_loss_weight=1.0,
                                   feature_matching_weight=10.0,      adversarial_loss_type=<AdversarialLossType.GAN: 1>,
                                   use_feature_matching_loss=False)

```

Bases: `ashpy.losses.executor.SumExecutor`

Pix2Pix Loss.

Weighted sum of `ashpy.losses.gan.GeneratorL1`, `ashpy.losses.gan.AdversarialLossG` and `ashpy.losses.gan.FeatureMatchingLoss`.

Used by Pix2Pix [1] and Pix2PixHD [2]

Methods

<code>__init__([l1_loss_weight, ...])</code>	Initialize the loss.
--	----------------------

Attributes

<code>executors</code>	Return the List of Executors.
<code>fn</code>	Return the Keras loss function to execute.

Continued on next page

Table 157 – continued from previous page

<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__` (*l1_loss_weight=100.0, adversarial_loss_weight=1.0, feature_matching_weight=10.0, adversarial_loss_type=<AdversarialLossType.GAN: 1>, use_feature_matching_loss=False*)
Initialize the loss.

Weighted sum of `ashpy.losses.gan.GeneratorL1`, `ashpy.losses.gan.AdversarialLossG` and `ashpy.losses.gan.FeatureMatchingLoss`.

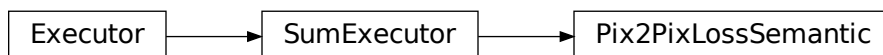
Parameters

- `l1_loss_weight` (`ashpy.ashtypes.TWeight`) – Weight of L1 loss.
- `adversarial_loss_weight` (`ashpy.ashtypes.TWeight`) – Weight of adversarial loss.
- `feature_matching_weight` (`ashpy.ashtypes.TWeight`) – Weight of the feature matching loss.
- `adversarial_loss_type` (`ashpy.losses.gan.AdversarialLossType`) – Adversarial loss type (`ashpy.losses.gan.AdversarialLossType.GAN` or `ashpy.losses.gan.AdversarialLossType.LSGAN`).
- `use_feature_matching_loss` (*bool*) – if True use also uses `ashpy.losses.gan.FeatureMatchingLoss`.

Return type `None`

Pix2PixLossSemantic

Inheritance Diagram



```

class ashpy.losses.gan.Pix2PixLossSemantic (cross_entropy_weight=100.0,          ad-
                                             adversarial_loss_weight=1.0,          fea-
                                             feature_matching_weight=10.0,          adversar-
                                             ial_loss_type=<AdversarialLossType.GAN:
                                             1>, use_feature_matching_loss=False)
    
```

Bases: `ashpy.losses.executor.SumExecutor`

Semantic Pix2Pix Loss.

Weighted sum of `ashpy.losses.gan.CategoricalCrossEntropy`, `ashpy.losses.gan.AdversarialLossG` and `ashpy.losses.gan.FeatureMatchingLoss`.

Methods

<code>__init__</code> (<code>cross_entropy_weight</code> , ...)	Initialize the Executor.
--	--------------------------

Attributes

<code>executors</code>	Return the List of Executors.
<code>fn</code>	Return the Keras loss function to execute.
<code>global_batch_size</code>	Global batch size comprises the batch size for each cpu.
<code>weight</code>	Return the loss weight.

`__init__`(`cross_entropy_weight=100.0`, `adversarial_loss_weight=1.0`, `feature_matching_weight=10.0`, `adversarial_loss_type=<AdversarialLossType.GAN: 1>`, `use_feature_matching_loss=False`)
Initialize the Executor.

Weighted sum of `ashpy.losses.gan.CategoricalCrossEntropy`, `ashpy.losses.gan.AdversarialLossG` and `ashpy.losses.gan.FeatureMatchingLoss`

Parameters

- **`cross_entropy_weight`** (`ashpy.ashtypes.TWeight`) – Weight of the categorical cross entropy loss.
- **`adversarial_loss_weight`** (`ashpy.ashtypes.TWeight`) – Weight of the adversarial loss.
- **`feature_matching_weight`** (`ashpy.ashtypes.TWeight`) – Weight of the feature matching loss.
- **`adversarial_loss_type`** (`ashpy.losses.gan.AdversarialLossType`) – type of adversarial loss, see `ashpy.losses.gan.AdversarialLossType`
- **`use_feature_matching_loss`** (`bool`) – whether to use feature matching loss or not

class `ashpy.losses.gan.AdversarialLossType`

Bases: `enum.Enum`

Enumeration for Adversarial Losses. Implemented: GAN and LSGAN.

class `ashpy.losses.gan.CategoricalCrossEntropy`

Bases: `ashpy.losses.executor.Executor`

Categorical Cross Entropy between generator output and target.

Useful when the output of the generator is a distribution over classes.

..note:: The target must be represented in one hot notation.

`__init__`()

Initialize the Categorical Cross Entropy Executor.

Return type `None`

class `ashpy.losses.gan.DiscriminatorAdversarialLoss`(`loss_fn=None`)

Bases: `ashpy.losses.gan.GANExecutor`

Base class for the adversarial loss of the discriminator.

`__init__(loss_fn=None)`
Initialize the Executor.

Parameters

- `loss_fn` (`tf.keras.losses.Loss`) – Loss function call passing
- `d_fake` (`(d_real,)`) –

Return type None

class `ashpy.losses.gan.DiscriminatorHingeLoss`
Bases: `ashpy.losses.gan.DiscriminatorAdversarialLoss`

Hinge loss for the Discriminator.

See Geometric GAN [1]_ for more details.

`__init__()`
Initialize the Least Square Loss for the Generator.

Return type None

class `ashpy.losses.gan.DiscriminatorLSGAN`
Bases: `ashpy.losses.gan.DiscriminatorAdversarialLoss`

Least square Loss for discriminator.

Reference: Least Squares Generative Adversarial Networks [1]_.

Basically the Mean Squared Error between the discriminator output when evaluated in fake samples and 0 and the discriminator output when evaluated in real samples and 1: For the unconditioned case this is:

$$L_D = \frac{1}{2} E[(D(x) - 1)^2 + (0 - D(G(z)))^2]$$

where x are real samples and z is the latent vector.

For the conditioned case this is:

$$L_D = \frac{1}{2} E[(D(x, c) - 1)^2 + (0 - D(G(c), c))^2]$$

where c is the condition and x are real samples.

`__init__()`
Initialize loss.

Return type None

class `ashpy.losses.gan.DiscriminatorMinMax` (`from_logits=True, label_smoothing=0.0`)
Bases: `ashpy.losses.gan.DiscriminatorAdversarialLoss`

The min-max game played by the discriminator.

$$L_D = -\frac{1}{2} E[\log(D(x)) + \log(1 - D(G(z)))]$$

`__init__(from_logits=True, label_smoothing=0.0)`
Initialize Loss.

class `ashpy.losses.gan.EncoderBCE` (`from_logits=True`)
Bases: `ashpy.losses.executor.Executor`

The Binary Cross Entropy computed among the encoder and the 0 label.

__init__ (*from_logits=True*)
Initialize the Executor.

Return type None

class `ashpy.losses.gan.FeatureMatchingLoss`

Bases: `ashpy.losses.gan.GANExecutor`

Conditional GAN Feature matching loss.

The loss is computed for each example and it's the L1 (MAE) of the feature difference. Implementation of pix2pix HD: <https://github.com/NVIDIA/pix2pixHD>

$$FM = \sum_{i=0}^N \frac{1}{M_i} ||D_i(x, c) - D_i(G(c), c)||_1$$

Where:

- D_i is the i -th layer of the discriminator
- N is the total number of layer of the discriminator
- M_i is the number of components for the i -th layer
- x is the target image
- c is the condition
- $G(c)$ is the generated image from the condition c
- $|| \cdot ||_1$ stands for norm 1.

This is for a single example: basically for each layer of the discriminator we compute the absolute error between the layer evaluated in real examples and in fake examples. Then we average along the batch. In the case where D_i is a multidimensional tensor we simply calculate the mean over the axis 1,2,3.

__init__ ()
Initialize the Executor.

Return type None

class `ashpy.losses.gan.GANExecutor` (*fn=None*)

Bases: `ashpy.losses.executor.Executor`, `abc.ABC`

Executor for GANs.

Implements the basic functions needed by the GAN losses.

call (*context, **kwargs*)

Execute the function, using the information provided by the context.

Parameters **context** (`ashpy.contexts.Context`) – The function execution Context.

Returns `tf.Tensor` – Output Tensor.

static **get_discriminator_inputs** (*context, fake_or_real, condition, training*)

Return the discriminator inputs. If needed it uses the encoder.

The current implementation uses the number of inputs to determine whether the discriminator is conditioned or not.

Parameters

- **context** (`ashpy.contexts.gan.GANContext`) – Context for GAN models.
- **fake_or_real** (`tf.Tensor`) – Discriminator input tensor, it can be fake (generated) or real.

- **condition** (`tf.Tensor`) – Discriminator condition (it can also be generator noise).
- **training** (`bool`) – whether is training phase or not

Return type `Union[Tensor, List[Tensor]]`

Returns The discriminator inputs.

class `ashpy.losses.gan.GeneratorAdversarialLoss` (*loss_fn=None*)

Bases: `ashpy.losses.gan.GANExecutor`

Base class for the adversarial loss of the generator.

__init__ (*loss_fn=None*)

Initialize the Executor.

Parameters **loss_fn** (`tf.keras.losses.Loss`) – Keras Loss function to call passing (`tf.ones_like(d_fake_i), d_fake_i`).

Return type `None`

class `ashpy.losses.gan.GeneratorBCE` (*from_logits=True*)

Bases: `ashpy.losses.gan.GeneratorAdversarialLoss`

The Binary CrossEntropy computed among the generator and the 1 label.

$$L_G = E[\log(D(G(z)))]$$

__init__ (*from_logits=True*)

Initialize the BCE Loss for the Generator.

Return type `None`

class `ashpy.losses.gan.GeneratorHingeLoss`

Bases: `ashpy.losses.gan.GeneratorAdversarialLoss`

Hinge loss for the Generator.

See Geometric GAN [1] for more details.

__init__ ()

Initialize the Least Square Loss for the Generator.

Return type `None`

class `ashpy.losses.gan.GeneratorL1`

Bases: `ashpy.losses.gan.GANExecutor`

L1 loss between the generator output and the target.

$$L_G = E[||x - G(z)||_1]$$

Where x is the target and G(z) is generated image.

__init__ ()

Initialize the Executor.

Return type `None`

class `ashpy.losses.gan.GeneratorLSGAN`

Bases: `ashpy.losses.gan.GeneratorAdversarialLoss`

Least Square GAN Loss for generator.

Reference: <https://arxiv.org/abs/1611.04076>

Note: Basically the Mean Squared Error between the discriminator output when evaluated in fake and 1.

$$L_G = \frac{1}{2} E[(1 - D(G(z)))^2]$$

__init__ ()

Initialize the Least Square Loss for the Generator.

Return type None

```
class ashpy.losses.gan.Pix2PixLoss (l1_loss_weight=100.0,      adversarial_loss_weight=1.0,
                                   feature_matching_weight=10.0,      adversarial_loss_type=<AdversarialLossType.GAN: 1>,
                                   use_feature_matching_loss=False)
```

Bases: *ashpy.losses.executor.SumExecutor*

Pix2Pix Loss.

Weighted sum of *ashpy.losses.gan.GeneratorL1*, *ashpy.losses.gan.AdversarialLossG* and *ashpy.losses.gan.FeatureMatchingLoss*.

Used by Pix2Pix [1] and Pix2PixHD [2]

__init__ (l1_loss_weight=100.0, adversarial_loss_weight=1.0, feature_matching_weight=10.0, adversarial_loss_type=<AdversarialLossType.GAN: 1>, use_feature_matching_loss=False)
Initialize the loss.

Weighted sum of *ashpy.losses.gan.GeneratorL1*, *ashpy.losses.gan.AdversarialLossG* and *ashpy.losses.gan.FeatureMatchingLoss*.

Parameters

- **l1_loss_weight** (*ashpy.ashtypes.TWeight*) – Weight of L1 loss.
- **adversarial_loss_weight** (*ashpy.ashtypes.TWeight*) – Weight of adversarial loss.
- **feature_matching_weight** (*ashpy.ashtypes.TWeight*) – Weight of the feature matching loss.
- **adversarial_loss_type** (*ashpy.losses.gan.AdversarialLossType*) – Adversarial loss type (*ashpy.losses.gan.AdversarialLossType.GAN* or *ashpy.losses.gan.AdversarialLossType.LSGAN*).
- **use_feature_matching_loss** (*bool*) – if True use also uses *ashpy.losses.gan.FeatureMatchingLoss*.

Return type None

```
class ashpy.losses.gan.Pix2PixLossSemantic (cross_entropy_weight=100.0,      adversarial_loss_weight=1.0,      feature_matching_weight=10.0,      adversarial_loss_type=<AdversarialLossType.GAN: 1>, use_feature_matching_loss=False)
```

Bases: *ashpy.losses.executor.SumExecutor*

Semantic Pix2Pix Loss.

Weighted sum of `ashpy.losses.gan.CategoricalCrossEntropy`, `ashpy.losses.gan.AdversarialLossG` and `ashpy.losses.gan.FeatureMatchingLoss`.

```
__init__(cross_entropy_weight=100.0, adversarial_loss_weight=1.0, feature_matching_weight=10.0, adversarial_loss_type=<AdversarialLossType.GAN: 1>, use_feature_matching_loss=False)
```

Initialize the Executor.

Weighted sum of `ashpy.losses.gan.CategoricalCrossEntropy`, `ashpy.losses.gan.AdversarialLossG` and `ashpy.losses.gan.FeatureMatchingLoss`

Parameters

- **cross_entropy_weight** (`ashpy.ashtypes.TWeight`) – Weight of the categorical cross entropy loss.
- **adversarial_loss_weight** (`ashpy.ashtypes.TWeight`) – Weight of the adversarial loss.
- **feature_matching_weight** (`ashpy.ashtypes.TWeight`) – Weight of the feature matching loss.
- **adversarial_loss_type** (`ashpy.losses.gan.AdversarialLossType`) – type of adversarial loss, see `ashpy.losses.gan.AdversarialLossType`
- **use_feature_matching_loss** (`bool`) – whether to use feature matching loss or not

```
ashpy.losses.gan.get_adversarial_loss_discriminator(adversarial_loss_type=<AdversarialLossType.GAN: 1>)
```

Return the correct loss for the Discriminator.

Parameters **adversarial_loss_type** (`ashpy.losses.gan.AdversarialLossType`) – Type of loss (`ashpy.losses.gan.AdversarialLossType.GAN` or `ashpy.losses.gan.AdversarialLossType.LSGAN`)

Return type `Type[Executor]`

Returns The correct (`ashpy.losses.executor.Executor`) (to be instantiated).

```
ashpy.losses.gan.get_adversarial_loss_generator(adversarial_loss_type=<AdversarialLossType.GAN: 1>)
```

Return the correct loss for the Generator.

Parameters **adversarial_loss_type** (`ashpy.losses.AdversarialLossType`) – Type of loss (`ashpy.losses.AdversarialLossType.GAN` or `ashpy.losses.AdversarialLossType.LSGAN`).

Return type `Type[Executor]`

Returns The correct (`ashpy.losses.executor.Executor`) (to be instantiated).

4.7 ashpy.metrics

Collection of Metrics.

Metric

metric.Metric

Metric is the abstract class that every ash Metric must implement.

4.7.1 Metric

Inheritance Diagram



```

class ashpy.metrics.metric.Metric(name, metric, model_selection_operator=None,
                                  logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checko
  Bases: abc.ABC
  
```

Metric is the abstract class that every ash Metric must implement.

AshPy Metrics wrap and extend Keras Metrics.

Methods

<i>__init__</i> (name, metric[, ...])	Initialize the Metric object.
<i>json_read</i> (filename)	Read a JSON file.
<i>json_write</i> (filename, what_to_write)	Write inside the specified JSON file the mean and stddev.
<i>log</i> (step)	Log the metric.
<i>model_selection</i> (checkpoint, global_step)	Perform model selection.
<i>reset_states</i> ()	Reset the state of the metric.
<i>result</i> ()	Get the result of the metric.
<i>update_state</i> (context)	Update the internal state of the metric, using the information from the context object.

Attributes

<i>best_folder</i>	Retrieve the folder used to save the best model when doing model selection.
<i>best_model_sel_file</i>	Retrieve the path to JSON file containing the measured performance of the best model.
<i>logdir</i>	Retrieve the log directory.
<i>metric</i>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<i>model_selection_operator</i>	Retrieve the operator used for model selection.
<i>name</i>	Retrieve the metric name.

Continued on next page

Table 162 – continued from previous page

<i>sanitized_name</i>	all / are _.
-----------------------	--------------

__init__ (*name, metric, model_selection_operator=None, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_bui*
Initialize the Metric object.

Parameters

- **name** (*str*) – Name of the metric.
- **metric** (*tf.keras.metrics.Metric*) – The Keras metric to use.
- **model_selection_operator** (*typing.Callable*) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any *typing.Callable* behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an *model_selection_operator* is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.

Return type *None*

best_folder

Retrieve the folder used to save the best model when doing model selection.

Return type *Path*

best_model_sel_file

Retrieve the path to JSON file containing the measured performance of the best model.

Return type *Path*

static json_read (*filename*)

Read a JSON file.

Parameters **filename** (*str*) – The path to the JSON file to read.

Return type *Dict[str, Any]*

Returns *typing.Dict* – Dictionary containing the content of the JSON file.

static json_write (*filename, what_to_write*)

Write inside the specified JSON file the mean and stddev.

Parameters

- **filename** (*str*) – Path to the JSON file to write.
- **what_to_write** (*dict*) – A dictionary containing what to write.

Return type *None*

log (*step*)

Log the metric.

Parameters **step** (*int*) – global step of training

Return type *None*

logdir

Retrieve the log directory.

Return type *Path*

metric

Retrieve the `tf.keras.metrics.Metric` object.

Return type `Metric`

model_selection (*checkpoint*, *global_step*)

Perform model selection.

Parameters

- **checkpoint** (`tf.train.Checkpoint`) – Checkpoint object that contains the model status.
- **global_step** (`tf.Variable`) – current training step

Return type `Optional[Path]`

model_selection_operator

Retrieve the operator used for model selection.

Return type `Optional[Callable]`

name

Retrieve the metric name.

Return type `str`

reset_states ()

Reset the state of the metric.

Return type `None`

result ()

Get the result of the metric.

Returns `numpy.ndarray` – The current value of the metric.

sanitized_name

all / are _.

This is done since adding a prefix to a metric name with a / allows for TensorBoard automatic grouping. When we are not working with TB we want to replace all / with _.

Type Retrieve the sanitized name

Return type `str`

update_state (*context*)

Update the internal state of the metric, using the information from the context object.

Parameters **context** (`ashpy.contexts.Context`) – An AshPy Context holding all the information the Metric needs.

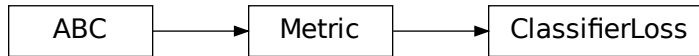
Return type `None`

Classifier

<code>classifier.ClassifierLoss</code>	A handy way to measure the classification loss.
<code>classifier.ClassifierMetric</code>	Wrap a metric using <i>argmax</i> to extract predictions out of a classifier's output.

4.7.2 ClassifierLoss

Inheritance Diagram



```

class ashpy.metrics.classifier.ClassifierLoss(name='loss',
                                              model_selection_operator=None,
                                              logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_bui
    
```

Bases: `ashpy.metrics.metric.Metric`

A handy way to measure the classification loss.

Methods

<code>__init__([name, model_selection_operator, ...])</code>	Initialize the Metric.
<code>update_state(context)</code>	Update the internal state of the metric, using the information from the context object.

Attributes

<code>best_folder</code>	Retrieve the folder used to save the best model when doing model selection.
<code>best_model_sel_file</code>	Retrieve the path to JSON file containing the measured performance of the best model.
<code>logdir</code>	Retrieve the log directory.
<code>metric</code>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<code>model_selection_operator</code>	Retrieve the operator used for model selection.
<code>name</code>	Retrieve the metric name.
<code>sanitized_name</code>	all / are _.

```

__init__(name='loss', model_selection_operator=None, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_bui
Initialize the Metric.
    
```

Parameters

- **name** (`str`) – Name of the metric.
- **model_selection_operator** (`typing.Callable`) – The operation that will be used when `model_selection` is triggered to compare the metrics, used by the `update_state`. Any `typing.Callable` behaving like an `operator` is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.

Return type *None*

update_state (*context*)

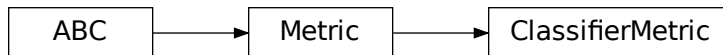
Update the internal state of the metric, using the information from the context object.

Parameters **context** (`ashpy.contexts.ClassifierContext`) – An AshPy Context holding all the information the Metric needs.

Return type *None*

4.7.3 ClassifierMetric

Inheritance Diagram



```

class ashpy.metrics.classifier.ClassifierMetric(metric,
                                                model_selection_operator=None,
                                                logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_
                                                processing_predictions=None))
  
```

Bases: *ashpy.metrics.metric.Metric*

Wrap a metric using *argmax* to extract predictions out of a classifier's output.

Methods

<code>__init__(metric[, model_selection_operator, ...])</code>	Initialize the Metric.
<code>update_state(context)</code>	Update the internal state of the metric, using the information from the context object.

Attributes

<code>best_folder</code>	Retrieve the folder used to save the best model when doing model selection.
<code>best_model_sel_file</code>	Retrieve the path to JSON file containing the measured performance of the best model.
<code>logdir</code>	Retrieve the log directory.

Continued on next page

Table 167 – continued from previous page

<code>metric</code>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<code>model_selection_operator</code>	Retrieve the operator used for model selection.
<code>name</code>	Retrieve the metric name.
<code>sanitized_name</code>	all / are _.

`__init__` (*metric*, *model_selection_operator*=None, *logdir*=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/conda/envs/ashpy-0.4.0/lib/python3.7/site-packages/ashpy/metrics'), *processing_predictions*=None)
Initialize the Metric.

Parameters

- **metric** (`tf.keras.metrics.Metric`) – The Keras Metric to use with the classifier (e.g.: Accuracy()).
- **model_selection_operator** (`typing.Callable`) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any `typing.Callable` behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an *model_selection_operator* is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.
- **processing_predictions** (`typing.Dict`) – A *dict* in the form of `{"fn": tf.argmax, "kwargs": {"axis": -1}}` with a function “fn” to be used for predictions processing purposes and its “kwargs” as its keyword-arguments. Defaults to `{"fn": tf.argmax, "kwargs": {"axis": -1}}`.

Return type None

update_state (*context*)

Update the internal state of the metric, using the information from the context object.

Parameters **context** (`ashpy.contexts.ClassifierContext`) – An AshPy Context holding all the information the Metric needs.

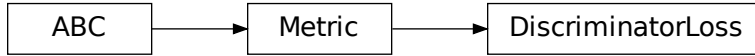
Return type None

GAN

<code>gan.DiscriminatorLoss</code>	The Discriminator loss value.
<code>gan.GeneratorLoss</code>	Generator loss value.
<code>gan.EncoderLoss</code>	Encoder Loss value.
<code>gan.InceptionScore</code>	Inception Score Metric.
<code>gan.EncodingAccuracy</code>	Generator and Encoder accuracy performance.
<code>sliced_wasserstein_metric</code> . <code>SlicedWassersteinDistance</code>	Sliced Wasserstein Distance.
<code>ssim_multiscale.SSIM_Multiscale</code>	Multiscale Structural Similarity.

4.7.4 DiscriminatorLoss

Inheritance Diagram



```

class ashpy.metrics.gan.DiscriminatorLoss (name='d_loss',
                                           model_selection_operator=None,
                                           logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/a...

Bases: ashpy.metrics.metric.Metric

The Discriminator loss value.
  
```

Methods

<code>__init__</code> ([name, model_selection_operator, ...])	Initialize the Metric.
<code>update_state</code> (context)	Update the internal state of the metric, using the information from the context object.

Attributes

<code>best_folder</code>	Retrieve the folder used to save the best model when doing model selection.
<code>best_model_sel_file</code>	Retrieve the path to JSON file containing the measured performance of the best model.
<code>logdir</code>	Retrieve the log directory.
<code>metric</code>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<code>model_selection_operator</code>	Retrieve the operator used for model selection.
<code>name</code>	Retrieve the metric name.
<code>sanitized_name</code>	all / are _.

```

__init__ (name='d_loss', model_selection_operator=None, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_b...
Initialize the Metric.
  
```

Parameters

- **name** (*str*) – Name of the metric.
- **model_selection_operator** (*typing.Callable*) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any *typing.Callable* behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.

Return type `None`

update_state (*context*)

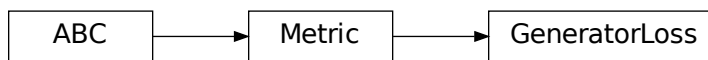
Update the internal state of the metric, using the information from the context object.

Parameters **context** (*ashpy.contexts.gan.GANContext*) – An AshPy Context Object that carries all the information the Metric needs.

Return type `None`

4.7.5 GeneratorLoss

Inheritance Diagram



class `ashpy.metrics.gan.GeneratorLoss` (*name='g_loss', model_selection_operator=None, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/c*

Bases: `ashpy.metrics.metric.Metric`

Generator loss value.

Methods

<code>__init__</code> (<i>[name, model_selection_operator, ...]</i>)	Initialize the Metric.
<code>update_state</code> (<i>context</i>)	Update the internal state of the metric, using the information from the context object.

Attributes

<code>best_folder</code>	Retrieve the folder used to save the best model when doing model selection.
<code>best_model_sel_file</code>	Retrieve the path to JSON file containing the measured performance of the best model.
<code>logdir</code>	Retrieve the log directory.
<code>metric</code>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<code>model_selection_operator</code>	Retrieve the operator used for model selection.
<code>name</code>	Retrieve the metric name.
<code>sanitized_name</code>	all / are _.

`__init__` (*name*=`'g_loss'`, *model_selection_operator*=`None`, *logdir*=`PosixPath('/home/docs/checkouts/readthedocs.org/user_b`
Initialize the Metric.

Parameters

- **name** (*str*) – Name of the metric.
- **model_selection_operator** (*typing.Callable*) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any *typing.Callable* behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.

update_state (*context*)

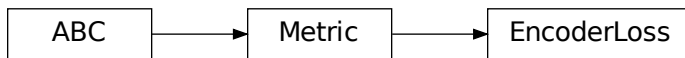
Update the internal state of the metric, using the information from the context object.

Parameters **context** (*ashpy.contexts.GANContext*) – An AshPy Context Object that carries all the information the Metric needs.

Return type *None*

4.7.6 EncoderLoss

Inheritance Diagram



class `ashpy.metrics.gan.EncoderLoss` (*name*=`'e_loss'`, *model_selection_operator*=`None`,
logdir=`PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/che`

Bases: `ashpy.metrics.metric.Metric`

Encoder Loss value.

Methods

<code>__init__</code> (<i>[name, model_selection_operator, ...]</i>)	Initialize the Metric.
<code>update_state</code> (<i>context</i>)	Update the internal state of the metric, using the information from the context object.

Attributes

<code>best_folder</code>	Retrieve the folder used to save the best model when doing model selection.
<code>best_model_sel_file</code>	Retrieve the path to JSON file containing the measured performance of the best model.
<code>logdir</code>	Retrieve the log directory.
<code>metric</code>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<code>model_selection_operator</code>	Retrieve the operator used for model selection.
<code>name</code>	Retrieve the metric name.
<code>sanitized_name</code>	all / are _.

`__init__` (*name*='e_loss', *model_selection_operator*=None, *logdir*=PosixPath('/home/docs/checkouts/readthedocs.org/user_b...
Initialize the Metric.

Parameters

- **name** (*str*) – Name of the metric.
- **model_selection_operator** (*typing.Callable*) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any *typing.Callable* behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.

Return type None

update_state (*context*)

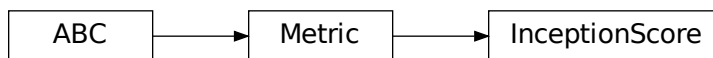
Update the internal state of the metric, using the information from the context object.

Parameters **context** (*ashpy.contexts.gan.GANEncoderContext*) – An AshPy Context Object that carries all the information the Metric needs.

Return type None

4.7.7 InceptionScore

Inheritance Diagram



```

class ashpy.metrics.gan.InceptionScore (inception,
                                         name='inception_score',
                                         model_selection_operator=<built-in function gt>,
                                         logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy...

Bases: ashpy.metrics.metric.Metric
  
```

Inception Score Metric.

This class is an implementation of the Inception Score technique for evaluating a GAN.

See Improved Techniques for Training GANs¹.

Methods

<code>__init__(inception[, name, ...])</code>	Initialize the Metric.
<code>get_or_train_inception(dataset, name, ...[, ...])</code>	Restore or train (and save) the Inception model.
<code>inception_score(images)</code>	Compute the Inception Score.
<code>update_state(context)</code>	Update the internal state of the metric, using the information from the context object.

Attributes

<code>best_folder</code>	Retrieve the folder used to save the best model when doing model selection.
<code>best_model_sel_file</code>	Retrieve the path to JSON file containing the measured performance of the best model.
<code>logdir</code>	Retrieve the log directory.
<code>metric</code>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<code>model_selection_operator</code>	Retrieve the operator used for model selection.
<code>name</code>	Retrieve the metric name.
<code>sanitized_name</code>	all / are _.

```
__init__(inception, name='inception_score', model_selection_operator=<built-in function gt>,
        logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log'))
```

Initialize the Metric.

Parameters

- **inception** (`tf.keras.Model`) – Keras Inception model.
- **name** (`str`) – Name of the metric.
- **model_selection_operator** (`typing.Callable`) – The operation that will be used when `model_selection` is triggered to compare the metrics, used by the `update_state`. Any `typing.Callable` behaving like an `operator` is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (`str`) – Path to the log dir, defaults to a `log` folder in the current directory.

```
static get_or_train_inception(dataset, name, num_classes, epochs, fine_tuning=False,
                             loss_fn=<tensorflow.python.keras.losses.SparseCategoricalCrossentropy
                             object>, optimizer=<tensorflow.python.keras.optimizer_v2.adam.Adam
                             object>, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy'))
```

Restore or train (and save) the Inception model.

Parameters

¹ Improved Techniques for Training GANs <https://arxiv.org/abs/1606.03498>

- **dataset** (`tf.data.Dataset`) – Dataset to re-train Inception Model on.
- **name** (`str`) – Name of this new Inception Model, used for saving it.
- **num_classes** (`int`) – Number of classes to use for classification.
- **epochs** (`int`) – Epochs to train the Inception model for.
- **fine_tuning** (`bool`) – Controls whether the model will be fine-tuned or used as is.
- **loss_fn** (`tf.keras.losses.Loss`) – Keras Loss for the model.
- **optimizer** (`tf.keras.optimizers.Optimizer`) – Keras optimizer for the model.
- **logdir** (`str`) – Path to the log dir, defaults to a *log* folder in the current directory.

Return type `Model`

Returns `tf.keras.Model` – The Inception Model.

inception_score (`images`)

Compute the Inception Score.

Parameters **images** (`list` of `[numpy.ndarray]`) – A list of ndarray of generated images of 299x299 of size.

Return type `Tensor`

Returns `tuple` of (`numpy.ndarray`, `numpy.ndarray`) – Mean and STD.

update_state (`context`)

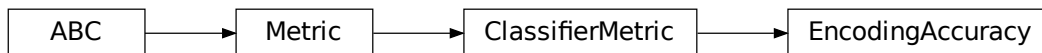
Update the internal state of the metric, using the information from the context object.

Parameters **context** (`ashpy.contexts.ClassifierContext`) – An AshPy Context holding all the information the Metric needs.

Return type `None`

4.7.8 EncodingAccuracy

Inheritance Diagram



```

class ashpy.metrics.gan.EncodingAccuracy(classifier, name='encoding_accuracy',
                                         model_selection_operator=None,
                                         logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/builds/0.4.0/docs'))

```

Bases: `ashpy.metrics.classifier.ClassifierMetric`

Generator and Encoder accuracy performance.

Measure the Generator and Encoder performance together, by classifying: $G(E(x))$, y using a pre-trained classifier (on the dataset of x).

Methods

<code>__init__(classifier[, name, ...])</code>	Measure the Generator and Encoder performance together.
<code>update_state(context)</code>	Update the internal state of the metric, using the information from the context object.

Attributes

<code>best_folder</code>	Retrieve the folder used to save the best model when doing model selection.
<code>best_model_sel_file</code>	Retrieve the path to JSON file containing the measured performance of the best model.
<code>logdir</code>	Retrieve the log directory.
<code>metric</code>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<code>model_selection_operator</code>	Retrieve the operator used for model selection.
<code>name</code>	Retrieve the metric name.
<code>sanitized_name</code>	all / are _.

`__init__(classifier, name='encoding_accuracy', model_selection_operator=None, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log'))`
 Measure the Generator and Encoder performance together.

This is done by classifying: $G(E(x))$, y using a pre-trained classifier (on the dataset of x).

Parameters

- **classifier** (`tf.keras.Model`) – Keras Model to use as a Classifier to measure the accuracy. Generally assumed to be the Inception Model.
- **name** (`str`) – Name of the metric.
- **model_selection_operator** (`typing.Callable`) – The operation that will be used when `model_selection` is triggered to compare the metrics, used by the `update_state`. Any `typing.Callable` behaving like an `operator` is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (`str`) – Path to the log dir, defaults to a `log` folder in the current directory.

Return type `None`

update_state (`context`)

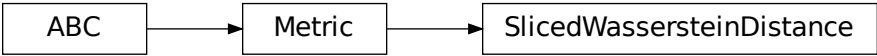
Update the internal state of the metric, using the information from the context object.

Parameters **context** (`ashpy.contexts.GANEncoderContext`) – An AshPy Context Object that carries all the information the Metric needs.

Return type `None`

4.7.9 SlicedWassersteinDistance

Inheritance Diagram



```
class ashpy.metrics.sliced_wasserstein_metric.SlicedWassersteinDistance (name='SWD',
                                                                           model_selection_operator=
                                                                           in
                                                                           func-
                                                                           tion
                                                                           lt>,
                                                                           logdir=PosixPath('/home/d
                                                                           res-
                                                                           o-
                                                                           lu-
                                                                           tion=128,
                                                                           res-
                                                                           o-
                                                                           lu-
                                                                           tion_min=16,
                                                                           patches_per_image=64,
                                                                           patch_size=7,
                                                                           ran-
                                                                           dom_sampling_count=1,
                                                                           ran-
                                                                           dom_projection_dim=147,
                                                                           use_svd=False)
```

Bases: *ashpy.metrics.metric.Metric*
Sliced Wasserstein Distance.
Used as metric in Progressive Growing of GANs¹.

Methods

<code>__init__</code> (name, model_selection_operator, ...)	Initialize the Metric.
<code>log</code> (step)	Log the SWD mean and each sub-metric.
<code>model_selection</code> (checkpoint, global_step)	Perform model selection for each sub-metric.
<code>reset_states</code> ()	Reset the state of the metric and the state of each child metric.
<code>update_state</code> (context)	Update the internal state of the metric, using the information from the context object.

¹ Progressive Growing of GANs <https://arxiv.org/abs/1710.10196>

Attributes

<code>best_folder</code>	Retrieve the folder used to save the best model when doing model selection.
<code>best_model_sel_file</code>	Retrieve the path to JSON file containing the measured performance of the best model.
<code>logdir</code>	Retrieve the log directory.
<code>metric</code>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<code>model_selection_operator</code>	Retrieve the operator used for model selection.
<code>name</code>	Retrieve the metric name.
<code>sanitized_name</code>	all / are _.

```
__init__(name='SWD', model_selection_operator=<built-in function lt>,
        logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log'),
        resolution=128, resolution_min=16, patches_per_image=64, patch_size=7, random_sampling_count=1, random_projection_dim=147, use_svd=False)
Initialize the Metric.
```

Parameters

- **name** (*str*) – Name of the metric.
- **model_selection_operator** (*typing.Callable*) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any *typing.Callable* behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.
- **resolution** (*int*) – Image Resolution, defaults to 128
- **resolution_min** (*int*) – Min Resolution achieved by the metric
- **patches_per_image** (*int*) – Number of patches to extract per image per Laplacian level.
- **patch_size** (*int*) – Width of a square patch.
- **random_sampling_count** (*int*) – Number of random projections to average.
- **random_projection_dim** (*int*) – Dimension of the random projection space.
- **use_svd** (*bool*) – experimental method to compute a more accurate distance.

Return type None

log (*step*)
Log the SWD mean and each sub-metric.

logdir
Retrieve the log directory.

Return type *str*

model_selection (*checkpoint, global_step*)
Perform model selection for each sub-metric.

Return type None

reset_states()

Reset the state of the metric and the state of each child metric.

Return type `None`

update_state(context)

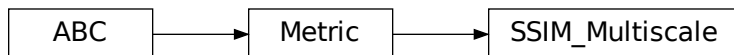
Update the internal state of the metric, using the information from the context object.

Parameters **context** (*ashpy.contexts.gan.GANContext*) – An AshPy Context Object that carries all the information the Metric needs.

Return type `None`

4.7.10 SSIM_Multiscale

Inheritance Diagram



```

class ashpy.metrics.ssim_multiscale.SSIM_Multiscale(name='SSIM_Multiscale',
                                                    model_selection_operator=<built-in
                                                    function lt>,
                                                    logdir=PosixPath('/home/docs/checkouts/readthedocs.org
                                                    max_val=2.0,
                                                    power_factors=None, filter
                                                    ter_size=11, filter_sigma=1.5,
                                                    k1=0.01, k2=0.03)

```

Bases: *ashpy.metrics.metric.Metric*

Multiscale Structural Similarity.

See Multiscale structural similarity for image quality assessment¹

Methods

<code>__init__</code> (name, model_selection_operator, ...)	Initialize the Metric.
<code>split_batch</code> (batch)	Split a batch along axis 0 into two tensors having the same size.
<code>update_state</code> (context)	Update the internal state of the metric, using the information from the context object.

Attributes

¹ Multiscale structural similarity for image quality assessment <https://ieeexplore.ieee.org/document/1292216>

<code>best_folder</code>	Retrieve the folder used to save the best model when doing model selection.
<code>best_model_sel_file</code>	Retrieve the path to JSON file containing the measured performance of the best model.
<code>logdir</code>	Retrieve the log directory.
<code>metric</code>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<code>model_selection_operator</code>	Retrieve the operator used for model selection.
<code>name</code>	Retrieve the metric name.
<code>sanitized_name</code>	all / are _.

```
__init__(name='SSIM_Multiscale', model_selection_operator=<built-in function lt>,
        logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log'),
        max_val=2.0, power_factors=None, filter_size=11, filter_sigma=1.5, k1=0.01, k2=0.03)
Initialize the Metric.
```

Parameters

- **name** (*str*) – Name of the metric.
- **model_selection_operator** (*typing.Callable*) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any *typing.Callable* behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.
- **max_val** (*float*) – The dynamic range of the images (i.e., the difference between the maximum the and minimum) (see www.tensorflow.org/versions/r2.0/api_docs/python/tf/image/ssim_multiscale)
- **power_factors** (*List[float]*) – Iterable of weights for each of the scales. The number of scales used is the length of the list. Index 0 is the unscaled resolution's weight and each increasing scale corresponds to the image being downsampled by 2. Defaults to (0.0448, 0.2856, 0.3001, 0.2363, 0.1333), which are the values obtained in the original paper.
- **filter_size** (*int*) – Default value 11 (size of gaussian filter).
- **filter_sigma** (*float*) – Default value 1.5 (width of gaussian filter).
- **k1** (*float*) – Default value 0.01.
- **k2** (*float*) – Default value 0.03 (SSIM is less sensitivity to K2 for lower values, so it would be better if we take the values in range of $0 < K2 < 0.4$).

Return type `None`

static split_batch (*batch*)

Split a batch along axis 0 into two tensors having the same size.

Parameters **batch** (*tf.Tensor*) – A batch of images.

Return type `Tuple[Tensor, Tensor]`

Returns (`Tuple[tf.Tensor, tf.Tensor]`) The batch split in two tensors.

Raises `ValueError` – if the batch has size 1.

update_state (*context*)

Update the internal state of the metric, using the information from the context object.

Parameters **context** (*ashpy.contexts.gan.GANContext*) – An AshPy Context Object that carries all the information the Metric needs.

Return type *None*

Modules

<i>classifier</i>	The classification metrics.
<i>gan</i>	GAN metrics.
<i>metric</i>	Metric is the abstract class that every ash metric must implement.
<i>sliced_wasserstein_metric</i>	Sliced Wasserstein Distance metric.
<i>ssim_multiscale</i>	Multiscale Structural Similarity metric.

4.7.11 classifier

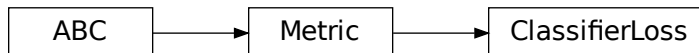
The classification metrics.

Classes

<i>ClassifierLoss</i>	A handy way to measure the classification loss.
<i>ClassifierMetric</i>	Wrap a metric using <i>argmax</i> to extract predictions out of a classifier's output.

ClassifierLoss

Inheritance Diagram



```

class ashpy.metrics.classifier.ClassifierLoss (name='loss',
                                                model_selection_operator=None,
                                                logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_bu

    Bases: ashpy.metrics.metric.Metric

    A handy way to measure the classification loss.
  
```

Methods

<code>__init__</code> (<code>name</code> , <code>model_selection_operator</code> , ...)	Initialize the Metric.
<code>update_state</code> (<code>context</code>)	Update the internal state of the metric, using the information from the context object.

Attributes

<code>best_folder</code>	Retrieve the folder used to save the best model when doing model selection.
<code>best_model_sel_file</code>	Retrieve the path to JSON file containing the measured performance of the best model.
<code>logdir</code>	Retrieve the log directory.
<code>metric</code>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<code>model_selection_operator</code>	Retrieve the operator used for model selection.
<code>name</code>	Retrieve the metric name.
<code>sanitized_name</code>	all / are _.

`__init__` (`name`=`'loss'`, `model_selection_operator`=`None`, `logdir`=`PosixPath('/home/docs/checkouts/readthedocs.org/user_bui`
Initialize the Metric.

Parameters

- **name** (`str`) – Name of the metric.
- **model_selection_operator** (`typing.Callable`) – The operation that will be used when `model_selection` is triggered to compare the metrics, used by the `update_state`. Any `typing.Callable` behaving like an `operator` is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (`str`) – Path to the log dir, defaults to a `log` folder in the current directory.

Return type `None`

update_state (`context`)

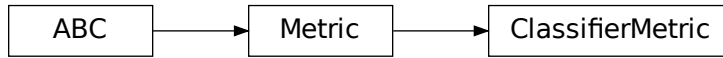
Update the internal state of the metric, using the information from the context object.

Parameters **context** (`ashpy.contexts.ClassifierContext`) – An AshPy Context holding all the information the Metric needs.

Return type `None`

ClassifierMetric

Inheritance Diagram



```

class ashpy.metrics.classifier.ClassifierMetric(metric,
                                                model_selection_operator=None,
                                                logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_
                                                processing_predictions=None)
    
```

Bases: `ashpy.metrics.metric.Metric`

Wrap a metric using *argmax* to extract predictions out of a classifier's output.

Methods

<code>__init__(metric[, model_selection_operator, ...])</code>	Initialize the Metric.
<code>update_state(context)</code>	Update the internal state of the metric, using the information from the context object.

Attributes

<code>best_folder</code>	Retrieve the folder used to save the best model when doing model selection.
<code>best_model_sel_file</code>	Retrieve the path to JSON file containing the measured performance of the best model.
<code>logdir</code>	Retrieve the log directory.
<code>metric</code>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<code>model_selection_operator</code>	Retrieve the operator used for model selection.
<code>name</code>	Retrieve the metric name.
<code>sanitized_name</code>	all / are _.

```

__init__(metric, model_selection_operator=None, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/builds/ashpy-0.4.0'),
         processing_predictions=None)
    Initialize the Metric.
    
```

Parameters

- **metric** (`tf.keras.metrics.Metric`) – The Keras Metric to use with the classifier (e.g.: `Accuracy()`).
- **model_selection_operator** (`typing.Callable`) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any `typing.Callable` behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an *model_selection_operator* is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.
- **processing_predictions** (*typing.Dict*) – A *dict* in the form of {"fn": *tf.argmax*, "kwargs": {"axis": -1}} with a function "fn" to be used for predictions processing purposes and its "kwargs" as its keyword-arguments. Defaults to {"fn": *tf.argmax*, "kwargs": {"axis": -1}}.

Return type *None*

update_state (*context*)

Update the internal state of the metric, using the information from the context object.

Parameters **context** (*ashpy.contexts.ClassifierContext*) – An AshPy Context holding all the information the Metric needs.

Return type *None*

```
class ashpy.metrics.classifier.ClassifierLoss (name='loss',
                                              model_selection_operator=None,
                                              logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_bui
```

Bases: *ashpy.metrics.metric.Metric*

A handy way to measure the classification loss.

```
__init__ (name='loss', model_selection_operator=None, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_bui
Initialize the Metric.
```

Parameters

- **name** (*str*) – Name of the metric.
- **model_selection_operator** (*typing.Callable*) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any *typing.Callable* behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.

Return type *None*

update_state (*context*)

Update the internal state of the metric, using the information from the context object.

Parameters **context** (*ashpy.contexts.ClassifierContext*) – An AshPy Context holding all the information the Metric needs.

Return type *None*

```
class ashpy.metrics.classifier.ClassifierMetric (metric,
                                              model_selection_operator=None,
                                              logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_
                                              processing_predictions=None)
```

Bases: *ashpy.metrics.metric.Metric*

Wrap a metric using *argmax* to extract predictions out of a classifier's output.

`__init__` (*metric*, *model_selection_operator*=None, *logdir*=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/ashpy'), *processing_predictions*=None)
Initialize the Metric.

Parameters

- **metric** (`tf.keras.metrics.Metric`) – The Keras Metric to use with the classifier (e.g.: `Accuracy()`).
- **model_selection_operator** (`typing.Callable`) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any `typing.Callable` behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an *model_selection_operator* is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.
- **processing_predictions** (`typing.Dict`) – A *dict* in the form of `{"fn": tf.argmax, "kwargs": {"axis": -1}}` with a function “*fn*” to be used for predictions processing purposes and its “*kwargs*” as its keyword-arguments. Defaults to `{"fn": tf.argmax, "kwargs": {"axis": -1}}`.

Return type None

update_state (*context*)

Update the internal state of the metric, using the information from the context object.

Parameters **context** (`ashpy.contexts.ClassifierContext`) – An AshPy Context holding all the information the Metric needs.

Return type None

4.7.12 gan

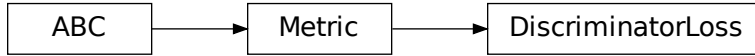
GAN metrics.

Classes

<i>DiscriminatorLoss</i>	The Discriminator loss value.
<i>EncoderLoss</i>	Encoder Loss value.
<i>EncodingAccuracy</i>	Generator and Encoder accuracy performance.
<i>GeneratorLoss</i>	Generator loss value.
<i>InceptionScore</i>	Inception Score Metric.

DiscriminatorLoss

Inheritance Diagram



```

class ashpy.metrics.gan.DiscriminatorLoss (name='d_loss',
                                           model_selection_operator=None,
                                           logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/a...

Bases: ashpy.metrics.metric.Metric

The Discriminator loss value.
  
```

Methods

<code>__init__</code> ([name, model_selection_operator, ...])	Initialize the Metric.
<code>update_state</code> (context)	Update the internal state of the metric, using the information from the context object.

Attributes

<code>best_folder</code>	Retrieve the folder used to save the best model when doing model selection.
<code>best_model_sel_file</code>	Retrieve the path to JSON file containing the measured performance of the best model.
<code>logdir</code>	Retrieve the log directory.
<code>metric</code>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<code>model_selection_operator</code>	Retrieve the operator used for model selection.
<code>name</code>	Retrieve the metric name.
<code>sanitized_name</code>	all / are _.

```

__init__ (name='d_loss', model_selection_operator=None, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_b...
Initialize the Metric.
  
```

Parameters

- **name** (*str*) – Name of the metric.
- **model_selection_operator** (*typing.Callable*) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any *typing.Callable* behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.

Return type `None`

update_state (*context*)

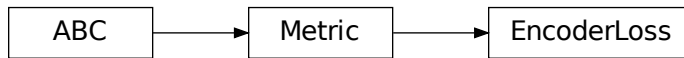
Update the internal state of the metric, using the information from the context object.

Parameters **context** (*ashpy.contexts.gan.GANContext*) – An AshPy Context Object that carries all the information the Metric needs.

Return type `None`

EncoderLoss

Inheritance Diagram



class `ashpy.metrics.gan.EncoderLoss` (*name='e_loss', model_selection_operator=None, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/cheo*

Bases: `ashpy.metrics.metric.Metric`

Encoder Loss value.

Methods

<code>__init__</code> ([<i>name</i> , <i>model_selection_operator</i> , ...])	Initialize the Metric.
<code>update_state</code> (<i>context</i>)	Update the internal state of the metric, using the information from the context object.

Attributes

<code>best_folder</code>	Retrieve the folder used to save the best model when doing model selection.
<code>best_model_sel_file</code>	Retrieve the path to JSON file containing the measured performance of the best model.
<code>logdir</code>	Retrieve the log directory.
<code>metric</code>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<code>model_selection_operator</code>	Retrieve the operator used for model selection.
<code>name</code>	Retrieve the metric name.
<code>sanitized_name</code>	all / are _.

`__init__` (*name*='e_loss', *model_selection_operator*=None, *logdir*=PosixPath('/home/docs/checkouts/readthedocs.org/user_b...'))
Initialize the Metric.

Parameters

- **name** (*str*) – Name of the metric.
- **model_selection_operator** (*typing.Callable*) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any *typing.Callable* behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.

Return type None

update_state (*context*)

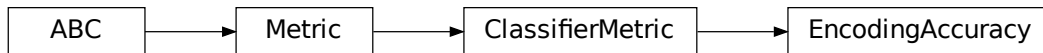
Update the internal state of the metric, using the information from the context object.

Parameters **context** (*ashpy.contexts.gan.GANEncoderContext*) – An AshPy Context Object that carries all the information the Metric needs.

Return type None

EncodingAccuracy

Inheritance Diagram



class `ashpy.metrics.gan.EncodingAccuracy` (*classifier*, *name*='encoding_accuracy', *model_selection_operator*=None, *logdir*=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/builds/ashpy-0.4.0/docs'))

Bases: `ashpy.metrics.classifier.ClassifierMetric`

Generator and Encoder accuracy performance.

Measure the Generator and Encoder performance together, by classifying: $G(E(x))$, y using a pre-trained classifier (on the dataset of x).

Methods

<code>__init__</code> (<i>classifier</i> [, <i>name</i> , ...])	Measure the Generator and Encoder performance together.
<code>update_state</code> (<i>context</i>)	Update the internal state of the metric, using the information from the context object.

Attributes

<code>best_folder</code>	Retrieve the folder used to save the best model when doing model selection.
<code>best_model_sel_file</code>	Retrieve the path to JSON file containing the measured performance of the best model.
<code>logdir</code>	Retrieve the log directory.
<code>metric</code>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<code>model_selection_operator</code>	Retrieve the operator used for model selection.
<code>name</code>	Retrieve the metric name.
<code>sanitized_name</code>	all / are _.

`__init__` (*classifier*, *name*='encoding_accuracy', *model_selection_operator*=None, *logdir*=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log'))
Measure the Generator and Encoder performance together.

This is done by classifying: $G(E(x))$, y using a pre-trained classifier (on the dataset of x).

Parameters

- **classifier** (`tf.keras.Model`) – Keras Model to use as a Classifier to measure the accuracy. Generally assumed to be the Inception Model.
- **name** (*str*) – Name of the metric.
- **model_selection_operator** (`typing.Callable`) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any `typing.Callable` behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.

Return type None

update_state (*context*)

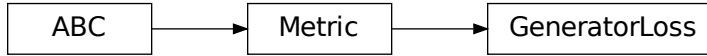
Update the internal state of the metric, using the information from the context object.

Parameters **context** (`ashpy.contexts.GANEncoderContext`) – An AshPy Context Object that carries all the information the Metric needs.

Return type None

GeneratorLoss

Inheritance Diagram



```

class ashpy.metrics.gan.GeneratorLoss (name='g_loss',      model_selection_operator=None,
                                         logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/c

```

Bases: `ashpy.metrics.metric.Metric`

Generator loss value.

Methods

<code>__init__</code> (name, model_selection_operator, ...)	Initialize the Metric.
<code>update_state</code> (context)	Update the internal state of the metric, using the information from the context object.

Attributes

<code>best_folder</code>	Retrieve the folder used to save the best model when doing model selection.
<code>best_model_sel_file</code>	Retrieve the path to JSON file containing the measured performance of the best model.
<code>logdir</code>	Retrieve the log directory.
<code>metric</code>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<code>model_selection_operator</code>	Retrieve the operator used for model selection.
<code>name</code>	Retrieve the metric name.
<code>sanitized_name</code>	all / are _.

```

__init__ (name='g_loss', model_selection_operator=None, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_b
Initialize the Metric.

```

Parameters

- **name** (*str*) – Name of the metric.
- **model_selection_operator** (`typing.Callable`) – The operation that will be used when `model_selection` is triggered to compare the metrics, used by the `update_state`. Any `typing.Callable` behaving like an `operator` is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a `log` folder in the current directory.

update_state (*context*)

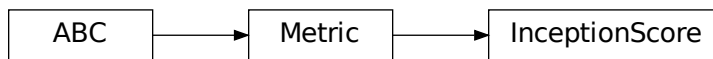
Update the internal state of the metric, using the information from the context object.

Parameters **context** (`ashpy.contexts.GANContext`) – An AshPy Context Object that carries all the information the Metric needs.

Return type `None`

InceptionScore

Inheritance Diagram



```

class ashpy.metrics.gan.InceptionScore (inception, name='inception_score',
                                         model_selection_operator=<built-in function gt>,
                                         logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/
  
```

Bases: `ashpy.metrics.metric.Metric`

Inception Score Metric.

This class is an implementation of the Inception Score technique for evaluating a GAN.

See Improved Techniques for Training GANs¹.

Methods

<code>__init__(inception[, name, ...])</code>	Initialize the Metric.
<code>get_or_train_inception(dataset, name, ...[,...])</code>	Restore or train (and save) the Inception model.
<code>inception_score(images)</code>	Compute the Inception Score.
<code>update_state(context)</code>	Update the internal state of the metric, using the information from the context object.

Attributes

<code>best_folder</code>	Retrieve the folder used to save the best model when doing model selection.
<code>best_model_sel_file</code>	Retrieve the path to JSON file containing the measured performance of the best model.
<code>logdir</code>	Retrieve the log directory.

Continued on next page

¹ Improved Techniques for Training GANs <https://arxiv.org/abs/1606.03498>

Table 199 – continued from previous page

<code>metric</code>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<code>model_selection_operator</code>	Retrieve the operator used for model selection.
<code>name</code>	Retrieve the metric name.
<code>sanitized_name</code>	all / are _.

`__init__` (*inception*, *name*='inception_score', *model_selection_operator*=<built-in function gt>, *logdir*=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log'))
Initialize the Metric.

Parameters

- **inception** (`tf.keras.Model`) – Keras Inception model.
- **name** (`str`) – Name of the metric.
- **model_selection_operator** (`typing.Callable`) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any `typing.Callable` behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (`str`) – Path to the log dir, defaults to a *log* folder in the current directory.

static get_or_train_inception (*dataset*, *name*, *num_classes*, *epochs*, *fine_tuning*=False, *loss_fn*=<tensorflow.python.keras.losses.SparseCategoricalCrossentropy object>, *optimizer*=<tensorflow.python.keras.optimizer_v2.adam.Adam object>, *logdir*=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log'))
Restore or train (and save) the Inception model.

Parameters

- **dataset** (`tf.data.Dataset`) – Dataset to re-train Inception Model on.
- **name** (`str`) – Name of this new Inception Model, used for saving it.
- **num_classes** (`int`) – Number of classes to use for classification.
- **epochs** (`int`) – Epochs to train the Inception model for.
- **fine_tuning** (`bool`) – Controls whether the model will be fine-tuned or used as is.
- **loss_fn** (`tf.keras.losses.Loss`) – Keras Loss for the model.
- **optimizer** (`tf.keras.optimizers.Optimizer`) – Keras optimizer for the model.
- **logdir** (`str`) – Path to the log dir, defaults to a *log* folder in the current directory.

Return type `Model`

Returns `tf.keras.Model` – The Inception Model.

inception_score (*images*)
Compute the Inception Score.

Parameters **images** (*list* of [`numpy.ndarray`]) – A list of ndarray of generated images of 299x299 of size.

Return type `Tensor`

Returns *tuple* of (`numpy.ndarray`, `numpy.ndarray`) – Mean and STD.

update_state (*context*)

Update the internal state of the metric, using the information from the context object.

Parameters **context** (`ashpy.contexts.ClassifierContext`) – An AshPy Context holding all the information the Metric needs.

Return type `None`

```
class ashpy.metrics.gan.DiscriminatorLoss (name='d_loss',
                                         model_selection_operator=None,
                                         logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/a
```

Bases: `ashpy.metrics.metric.Metric`

The Discriminator loss value.

```
__init__ (name='d_loss', model_selection_operator=None, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_b
Initialize the Metric.
```

Parameters

- **name** (*str*) – Name of the metric.
- **model_selection_operator** (`typing.Callable`) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any `typing.Callable` behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.

Return type `None`

update_state (*context*)

Update the internal state of the metric, using the information from the context object.

Parameters **context** (`ashpy.contexts.gan.GANContext`) – An AshPy Context Object that carries all the information the Metric needs.

Return type `None`

```
class ashpy.metrics.gan.EncoderLoss (name='e_loss',          model_selection_operator=None,
                                     logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/che
```

Bases: `ashpy.metrics.metric.Metric`

Encoder Loss value.

```
__init__ (name='e_loss', model_selection_operator=None, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_b
Initialize the Metric.
```

Parameters

- **name** (*str*) – Name of the metric.
- **model_selection_operator** (`typing.Callable`) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any `typing.Callable` behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.

Return type `None`

update_state (*context*)

Update the internal state of the metric, using the information from the context object.

Parameters **context** (*ashpy.contexts.gan.GANEncoderContext*) – An AshPy Context Object that carries all the information the Metric needs.

Return type *None*

class *ashpy.metrics.gan.EncodingAccuracy* (*classifier*, *name='encoding_accuracy'*, *model_selection_operator=None*, *logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log')*)

Bases: *ashpy.metrics.classifier.ClassifierMetric*

Generator and Encoder accuracy performance.

Measure the Generator and Encoder performance together, by classifying: $G(E(x))$, y using a pre-trained classifier (on the dataset of x).

__init__ (*classifier*, *name='encoding_accuracy'*, *model_selection_operator=None*, *logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log')*)

Measure the Generator and Encoder performance together.

This is done by classifying: $G(E(x))$, y using a pre-trained classifier (on the dataset of x).

Parameters

- **classifier** (*tf.keras.Model*) – Keras Model to use as a Classifier to measure the accuracy. Generally assumed to be the Inception Model.
- **name** (*str*) – Name of the metric.
- **model_selection_operator** (*typing.Callable*) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any *typing.Callable* behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.

Return type *None*

update_state (*context*)

Update the internal state of the metric, using the information from the context object.

Parameters **context** (*ashpy.contexts.GANEncoderContext*) – An AshPy Context Object that carries all the information the Metric needs.

Return type *None*

class *ashpy.metrics.gan.GeneratorLoss* (*name='g_loss'*, *model_selection_operator=None*, *logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log')*)

Bases: *ashpy.metrics.metric.Metric*

Generator loss value.

__init__ (*name='g_loss'*, *model_selection_operator=None*, *logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log')*)

Initialize the Metric.

Parameters

- **name** (*str*) – Name of the metric.

- **model_selection_operator** (`typing.Callable`) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any `typing.Callable` behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (`str`) – Path to the log dir, defaults to a *log* folder in the current directory.

update_state (`context`)

Update the internal state of the metric, using the information from the context object.

Parameters **context** (`ashpy.contexts.GANContext`) – An AshPy Context Object that carries all the information the Metric needs.

Return type `None`

```
class ashpy.metrics.gan.InceptionScore (inception, name='inception_score',
                                         model_selection_operator=<built-in function gt>,
                                         logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/'))
```

Bases: `ashpy.metrics.metric.Metric`

Inception Score Metric.

This class is an implementation of the Inception Score technique for evaluating a GAN.

See Improved Techniques for Training GANs¹.

```
__init__ (inception, name='inception_score', model_selection_operator=<built-in function gt>,
          logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log'))
```

Initialize the Metric.

Parameters

- **inception** (`tf.keras.Model`) – Keras Inception model.
- **name** (`str`) – Name of the metric.
- **model_selection_operator** (`typing.Callable`) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any `typing.Callable` behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (`str`) – Path to the log dir, defaults to a *log* folder in the current directory.

```
static get_or_train_inception (dataset, name, num_classes, epochs, fine_tuning=False,
                              loss_fn=<tensorflow.python.keras.losses.SparseCategoricalCrossentropy
                              object>, optimizer=<tensorflow.python.keras.optimizer_y2.adam.Adam
                              object>, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/'))
```

Restore or train (and save) the Inception model.

Parameters

- **dataset** (`tf.data.Dataset`) – Dataset to re-train Inception Model on.
- **name** (`str`) – Name of this new Inception Model, used for saving it.
- **num_classes** (`int`) – Number of classes to use for classification.
- **epochs** (`int`) – Epochs to train the Inception model for.

¹ Improved Techniques for Training GANs <https://arxiv.org/abs/1606.03498>

- **fine_tuning** (*bool*) – Controls wether the model will be fine-tuned or used as is.
- **loss_fn** (`tf.keras.losses.Loss`) – Keras Loss for the model.
- **optimizer** (`tf.keras.optimizers.Optimizer`) – Keras optimizer for the model.
- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.

Return type `Model`

Returns `tf.keras.Model` – The Inception Model.

inception_score (*images*)

Compute the Inception Score.

Parameters **images** (*list* of [`numpy.ndarray`]) – A list of ndarray of generated images of 299x299 of size.

Return type `Tensor`

Returns *tuple* of (`numpy.ndarray`, `numpy.ndarray`) – Mean and STD.

update_state (*context*)

Update the internal state of the metric, using the information from the context object.

Parameters **context** (`ashpy.contexts.ClassifierContext`) – An AshPy Context holding all the information the Metric needs.

Return type `None`

4.7.13 metric

Metric is the abstract class that every ash metric must implement.

Classes

<i>Metric</i>	Metric is the abstract class that every ash Metric must implement.
---------------	--

Metric

Inheritance Diagram



```

class ashpy.metrics.metric.Metric(name, metric, model_selection_operator=None,
    logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checko
    Bases: abc.ABC
  
```

Metric is the abstract class that every ash Metric must implement.

AshPy Metrics wrap and extend Keras Metrics.

Methods

<code>__init__(name, metric[, ...])</code>	Initialize the Metric object.
<code>json_read(filename)</code>	Read a JSON file.
<code>json_write(filename, what_to_write)</code>	Write inside the specified JSON file the mean and stddev.
<code>log(step)</code>	Log the metric.
<code>model_selection(checkpoint, global_step)</code>	Perform model selection.
<code>reset_states()</code>	Reset the state of the metric.
<code>result()</code>	Get the result of the metric.
<code>update_state(context)</code>	Update the internal state of the metric, using the information from the context object.

Attributes

<code>best_folder</code>	Retrieve the folder used to save the best model when doing model selection.
<code>best_model_sel_file</code>	Retrieve the path to JSON file containing the measured performance of the best model.
<code>logdir</code>	Retrieve the log directory.
<code>metric</code>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<code>model_selection_operator</code>	Retrieve the operator used for model selection.
<code>name</code>	Retrieve the metric name.
<code>sanitized_name</code>	all / are _.

`__init__(name, metric, model_selection_operator=None, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_bui`
Initialize the Metric object.

Parameters

- **name** (*str*) – Name of the metric.
- **metric** (`tf.keras.metrics.Metric`) – The Keras metric to use.
- **model_selection_operator** (`typing.Callable`) – The operation that will be used when `model_selection` is triggered to compare the metrics, used by the `update_state`. Any `typing.Callable` behaving like an `operator` is accepted.

Note: Model selection is done ONLY if an `model_selection_operator` is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a `log` folder in the current directory.

Return type `None`

best_folder

Retrieve the folder used to save the best model when doing model selection.

Return type `Path`

best_model_sel_file

Retrieve the path to JSON file containing the measured performance of the best model.

Return type `Path`

static json_read (*filename*)

Read a JSON file.

Parameters **filename** (*str*) – The path to the JSON file to read.

Return type `Dict[str, Any]`

Returns `typing.Dict` – Dictionary containing the content of the JSON file.

static json_write (*filename, what_to_write*)

Write inside the specified JSON file the mean and stddev.

Parameters

- **filename** (*str*) – Path to the JSON file to write.
- **what_to_write** (*dict*) – A dictionary containing what to write.

Return type `None`

log (*step*)

Log the metric.

Parameters **step** (*int*) – global step of training

Return type `None`

logdir

Retrieve the log directory.

Return type `Path`

metric

Retrieve the `tf.keras.metrics.Metric` object.

Return type `Metric`

model_selection (*checkpoint, global_step*)

Perform model selection.

Parameters

- **checkpoint** (`tf.train.Checkpoint`) – Checkpoint object that contains the model status.
- **global_step** (`tf.Variable`) – current training step

Return type `Optional[Path]`

model_selection_operator

Retrieve the operator used for model selection.

Return type `Optional[Callable]`

name

Retrieve the metric name.

Return type `str`

reset_states ()

Reset the state of the metric.

Return type `None`

result ()

Get the result of the metric.

Returns `numpy.ndarray` – The current value of the metric.

sanitized_name

all / are _.

This is done since adding a prefix to a metric name with a / allows for TensorBoard automatic grouping. When we are not working with TB we want to replace all / with _.

Type Retrieve the sanitized name

Return type `str`

update_state (*context*)

Update the internal state of the metric, using the information from the context object.

Parameters **context** (`ashpy.contexts.Context`) – An AshPy Context holding all the information the Metric needs.

Return type `None`

class `ashpy.metrics.metric.Metric` (*name*, *metric*, *model_selection_operator=None*, *logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/ashpy')*)

Bases: `abc.ABC`

Metric is the abstract class that every ash Metric must implement.

AshPy Metrics wrap and extend Keras Metrics.

__init__ (*name*, *metric*, *model_selection_operator=None*, *logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/ashpy')*)
Initialize the Metric object.

Parameters

- **name** (`str`) – Name of the metric.
- **metric** (`tf.keras.metrics.Metric`) – The Keras metric to use.
- **model_selection_operator** (`typing.Callable`) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any `typing.Callable` behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an *model_selection_operator* is specified here.

- **logdir** (`str`) – Path to the log dir, defaults to a *log* folder in the current directory.

Return type `None`

best_folder

Retrieve the folder used to save the best model when doing model selection.

Return type `Path`

best_model_sel_file

Retrieve the path to JSON file containing the measured performance of the best model.

Return type `Path`

static json_read (*filename*)

Read a JSON file.

Parameters **filename** (`str`) – The path to the JSON file to read.

Return type `Dict[str, Any]`

Returns `typing.Dict` – Dictionary containing the content of the JSON file.

static `json_write(filename, what_to_write)`

Write inside the specified JSON file the mean and stddev.

Parameters

- **filename** (`str`) – Path to the JSON file to write.
- **what_to_write** (`dict`) – A dictionary containing what to write.

Return type `None`

log (`step`)

Log the metric.

Parameters **step** (`int`) – global step of training

Return type `None`

logdir

Retrieve the log directory.

Return type `Path`

metric

Retrieve the `tf.keras.metrics.Metric` object.

Return type `Metric`

model_selection (`checkpoint, global_step`)

Perform model selection.

Parameters

- **checkpoint** (`tf.train.Checkpoint`) – Checkpoint object that contains the model status.
- **global_step** (`tf.Variable`) – current training step

Return type `Optional[Path]`

model_selection_operator

Retrieve the operator used for model selection.

Return type `Optional[Callable]`

name

Retrieve the metric name.

Return type `str`

reset_states ()

Reset the state of the metric.

Return type `None`

result ()

Get the result of the metric.

Returns `numpy.ndarray` – The current value of the metric.

sanitized_name

all / are _.

This is done since adding a prefix to a metric name with a / allows for TensorBoard automatic grouping. When we are not working with TB we want to replace all / with _.

Type Retrieve the sanitized name

Return type `str`

update_state (*context*)

Update the internal state of the metric, using the information from the context object.

Parameters **context** (`ashpy.contexts.Context`) – An AshPy Context holding all the information the Metric needs.

Return type `None`

4.7.14 sliced_wasserstein_metric

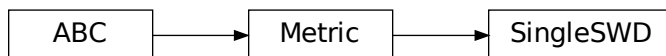
Sliced Wasserstein Distance metric.

Classes

<i>SingleSWD</i>	SlicedWassersteinDistance for a certain level of the pyramid.
<i>SlicedWassersteinDistance</i>	Sliced Wasserstein Distance.

SingleSWD

Inheritance Diagram



```

class ashpy.metrics.sliced_wasserstein_metric.SingleSWD (model_selection_operator=<built-
in function lt>,
logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_
level_of_pyramid=0,
real_or_fake='fake')

```

Bases: `ashpy.metrics.metric.Metric`

SlicedWassersteinDistance for a certain level of the pyramid.

Methods

<code>__init__</code> ([<i>model_selection_operator</i> , <i>logdir</i> , ...])	Initialize the Metric.
--	------------------------

Continued on next page

Table 204 – continued from previous page

<code>update_state(context, score)</code>	Update the internal state of the metric, using the information from the context object.
---	---

Attributes

<code>best_folder</code>	Retrieve the folder used to save the best model when doing model selection.
<code>best_model_sel_file</code>	Retrieve the path to JSON file containing the measured performance of the best model.
<code>logdir</code>	Retrieve the log directory.
<code>metric</code>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<code>model_selection_operator</code>	Retrieve the operator used for model selection.
<code>name</code>	Retrieve the metric name.
<code>sanitized_name</code>	all / are _.

`__init__` (*model_selection_operator=<built-in function lt>*, *logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_b...level_of_pyramid=0, real_or_fake='fake')*)
Initialize the Metric.

Parameters

- **model_selection_operator** (*typing.Callable*) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any *typing.Callable* behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.
- **level_of_pyramid** (*int*) – Level of the pyramid related to this metric.
- **real_or_fake** (*str*) – string identifying this metric (real or fake distance).

Return type `None`

update_state (*context, score*)

Update the internal state of the metric, using the information from the context object.

Parameters **context** (*ashpy.contexts.gan.GANContext*) – An AshPy Context Object that carries all the information the Metric needs.

Return type `None`

SlicedWassersteinDistance

Inheritance Diagram



```

class ashpy.metrics.sliced_wasserstein_metric.SlicedWassersteinDistance (name='SWD',
                                                                    model_selection_operator=
                                                                    in
                                                                    func-
                                                                    tion
                                                                    lt>,
                                                                    logdir=PosixPath('/home/d
                                                                    res-
                                                                    o-
                                                                    lu-
                                                                    tion=128,
                                                                    res-
                                                                    o-
                                                                    lu-
                                                                    tion_min=16,
                                                                    patches_per_image=64,
                                                                    patch_size=7,
                                                                    ran-
                                                                    dom_sampling_count=1,
                                                                    ran-
                                                                    dom_projection_dim=147,
                                                                    use_svd=False)
  
```

Bases: `ashpy.metrics.metric.Metric`

Sliced Wasserstein Distance.

Used as metric in Progressive Growing of GANs¹.

Methods

<code>__init__</code> (name, model_selection_operator, ...)	Initialize the Metric.
<code>log</code> (step)	Log the SWD mean and each sub-metric.
<code>model_selection</code> (checkpoint, global_step)	Perform model selection for each sub-metric.
<code>reset_states</code> ()	Reset the state of the metric and the state of each child metric.
<code>update_state</code> (context)	Update the internal state of the metric, using the information from the context object.

¹ Progressive Growing of GANs <https://arxiv.org/abs/1710.10196>

Attributes

<code>best_folder</code>	Retrieve the folder used to save the best model when doing model selection.
<code>best_model_sel_file</code>	Retrieve the path to JSON file containing the measured performance of the best model.
<code>logdir</code>	Retrieve the log directory.
<code>metric</code>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<code>model_selection_operator</code>	Retrieve the operator used for model selection.
<code>name</code>	Retrieve the metric name.
<code>sanitized_name</code>	all / are _.

```
__init__(name='SWD', model_selection_operator=<built-in function lt>,
        logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log'),
        resolution=128, resolution_min=16, patches_per_image=64, patch_size=7, random_sampling_count=1, random_projection_dim=147, use_svd=False)
Initialize the Metric.
```

Parameters

- **name** (*str*) – Name of the metric.
- **model_selection_operator** (*typing.Callable*) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any *typing.Callable* behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.
- **resolution** (*int*) – Image Resolution, defaults to 128
- **resolution_min** (*int*) – Min Resolution achieved by the metric
- **patches_per_image** (*int*) – Number of patches to extract per image per Laplacian level.
- **patch_size** (*int*) – Width of a square patch.
- **random_sampling_count** (*int*) – Number of random projections to average.
- **random_projection_dim** (*int*) – Dimension of the random projection space.
- **use_svd** (*bool*) – experimental method to compute a more accurate distance.

Return type None

log (*step*)
Log the SWD mean and each sub-metric.

logdir
Retrieve the log directory.

Return type *str*

model_selection (*checkpoint, global_step*)
Perform model selection for each sub-metric.

Return type None

reset_states()

Reset the state of the metric and the state of each child metric.

Return type `None`

update_state(context)

Update the internal state of the metric, using the information from the context object.

Parameters **context** (`ashpy.contexts.gan.GANContext`) – An AshPy Context Object that carries all the information the Metric needs.

Return type `None`

```
class ashpy.metrics.sliced_wasserstein_metric.SingleSWD(model_selection_operator=<built-in function lt>,
logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_b
level_of_pyramid=0,
real_or_fake='fake')
```

Bases: `ashpy.metrics.metric.Metric`

SlicedWassersteinDistance for a certain level of the pyramid.

```
__init__(model_selection_operator=<built-in function lt>, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_b
level_of_pyramid=0, real_or_fake='fake')
Initialize the Metric.
```

Parameters

- **model_selection_operator** (`typing.Callable`) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any `typing.Callable` behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (`str`) – Path to the log dir, defaults to a *log* folder in the current directory.
- **level_of_pyramid** (`int`) – Level of the pyramid related to this metric.
- **real_or_fake** (`str`) – string identifying this metric (real or fake distance).

Return type `None`

update_state(context, score)

Update the internal state of the metric, using the information from the context object.

Parameters **context** (`ashpy.contexts.gan.GANContext`) – An AshPy Context Object that carries all the information the Metric needs.

Return type `None`

```
class ashpy.metrics.sliced_wasserstein_metric.SlicedWassersteinDistance (name='SWD',
                                                                    model_selection_operator=
                                                                    in
                                                                    func-
                                                                    tion
                                                                    lt>,
                                                                    logdir=PosixPath('/home/d
                                                                    res-
                                                                    o-
                                                                    lu-
                                                                    tion=128,
                                                                    res-
                                                                    o-
                                                                    lu-
                                                                    tion_min=16,
                                                                    patches_per_image=64,
                                                                    patch_size=7,
                                                                    ran-
                                                                    dom_sampling_count=1,
                                                                    ran-
                                                                    dom_projection_dim=147,
                                                                    use_svd=False)
```

Bases: `ashpy.metrics.metric.Metric`

Sliced Wasserstein Distance.

Used as metric in Progressive Growing of GANs¹.

```
__init__ (name='SWD',          model_selection_operator=<built-in          function          lt>,
          logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log'),
          resolution=128, resolution_min=16, patches_per_image=64, patch_size=7, ran-
          dom_sampling_count=1, random_projection_dim=147, use_svd=False)
Initialize the Metric.
```

Parameters

- **name** (*str*) – Name of the metric.
- **model_selection_operator** (`typing.Callable`) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any `typing.Callable` behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.
- **resolution** (*int*) – Image Resolution, defaults to 128
- **resolution_min** (*int*) – Min Resolution achieved by the metric
- **patches_per_image** (*int*) – Number of patches to extract per image per Laplacian level.
- **patch_size** (*int*) – Width of a square patch.
- **random_sampling_count** (*int*) – Number of random projections to average.
- **random_projection_dim** (*int*) – Dimension of the random projection space.

¹ Progressive Growing of GANs <https://arxiv.org/abs/1710.10196>

- **use_svd** (*bool*) – experimental method to compute a more accurate distance.

Return type `None`

log (*step*)

Log the SWD mean and each sub-metric.

logdir

Retrieve the log directory.

Return type `str`

model_selection (*checkpoint, global_step*)

Perform model selection for each sub-metric.

Return type `None`

reset_states ()

Reset the state of the metric and the state of each child metric.

Return type `None`

update_state (*context*)

Update the internal state of the metric, using the information from the context object.

Parameters **context** (*ashpy.contexts.gan.GANContext*) – An AshPy Context Object that carries all the information the Metric needs.

Return type `None`

4.7.15 ssim_multiscale

Multiscale Structural Similarity metric.

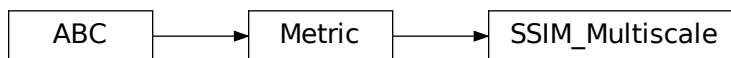
Classes

SSIM_Multiscale

Multiscale Structural Similarity.

SSIM_Multiscale

Inheritance Diagram



```
class ashpy.metrics.ssim_multiscale.SSIM_Multiscale(name='SSIM_Multiscale',
                                                    model_selection_operator=<built-in
                                                    function lt>,
                                                    logdir=PosixPath('/home/docs/checkouts/readthedocs.org'),
                                                    max_val=2.0,
                                                    power_factors=None, filter_size=11,
                                                    filter_sigma=1.5,
                                                    k1=0.01, k2=0.03)
```

Bases: *ashpy.metrics.metric.Metric*

Multiscale Structural Similarity.

See Multiscale structural similarity for image quality assessment¹

Methods

<code>__init__</code> ([name, model_selection_operator, ...])	Initialize the Metric.
<code>split_batch</code> (batch)	Split a batch along axis 0 into two tensors having the same size.
<code>update_state</code> (context)	Update the internal state of the metric, using the information from the context object.

Attributes

<code>best_folder</code>	Retrieve the folder used to save the best model when doing model selection.
<code>best_model_sel_file</code>	Retrieve the path to JSON file containing the measured performance of the best model.
<code>logdir</code>	Retrieve the log directory.
<code>metric</code>	Retrieve the <code>tf.keras.metrics.Metric</code> object.
<code>model_selection_operator</code>	Retrieve the operator used for model selection.
<code>name</code>	Retrieve the metric name.
<code>sanitized_name</code>	all / are _.

```
__init__(name='SSIM_Multiscale', model_selection_operator=<built-in function lt>,
         logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log'),
         max_val=2.0, power_factors=None, filter_size=11, filter_sigma=1.5, k1=0.01, k2=0.03)
Initialize the Metric.
```

Parameters

- **name** (*str*) – Name of the metric.
- **model_selection_operator** (*typing.Callable*) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any *typing.Callable* behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (*str*) – Path to the log dir, defaults to a *log* folder in the current directory.

¹ Multiscale structural similarity for image quality assessment <https://ieeexplore.ieee.org/document/1292216>

- **max_val** (*float*) – The dynamic range of the images (i.e., the difference between the maximum and minimum) (see www.tensorflow.org/versions/r2.0/api_docs/python/tf/image/ssim_multiscale)
- **power_factors** (*List[float]*) – Iterable of weights for each of the scales. The number of scales used is the length of the list. Index 0 is the unscaled resolution's weight and each increasing scale corresponds to the image being downsampled by 2. Defaults to (0.0448, 0.2856, 0.3001, 0.2363, 0.1333), which are the values obtained in the original paper.
- **filter_size** (*int*) – Default value 11 (size of gaussian filter).
- **filter_sigma** (*float*) – Default value 1.5 (width of gaussian filter).
- **k1** (*float*) – Default value 0.01.
- **k2** (*float*) – Default value 0.03 (SSIM is less sensitivity to K2 for lower values, so it would be better if we take the values in range of $0 < K2 < 0.4$).

Return type `None`

static split_batch (*batch*)

Split a batch along axis 0 into two tensors having the same size.

Parameters **batch** (*tf.Tensor*) – A batch of images.

Return type `Tuple[Tensor, Tensor]`

Returns (`Tuple[tf.Tensor, tf.Tensor]`) The batch split in two tensors.

Raises `ValueError` – if the batch has size 1.

update_state (*context*)

Update the internal state of the metric, using the information from the context object.

Parameters **context** (*ashpy.contexts.gan.GANContext*) – An AshPy Context Object that carries all the information the Metric needs.

Return type `None`

```
class ashpy.metrics.ssim_multiscale.SSIM_Multiscale (name='SSIM_Multiscale',
                                                    model_selection_operator=<built-in
                                                    function lt>,
                                                    logdir=PosixPath('/home/docs/checkouts/readthedocs.org
                                                    max_val=2.0,
                                                    power_factors=None,      fil-
                                                    ter_size=11,  filter_sigma=1.5,
                                                    k1=0.01, k2=0.03)
```

Bases: *ashpy.metrics.metric.Metric*

Multiscale Structural Similarity.

See Multiscale structural similarity for image quality assessment¹

```
__init__ (name='SSIM_Multiscale',      model_selection_operator=<built-in      function      lt>,
          logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log'),
          max_val=2.0, power_factors=None, filter_size=11, filter_sigma=1.5, k1=0.01, k2=0.03)
Initialize the Metric.
```

Parameters

- **name** (*str*) – Name of the metric.

¹ Multiscale structural similarity for image quality assessment <https://ieeexplore.ieee.org/document/1292216>

- **model_selection_operator** (`typing.Callable`) – The operation that will be used when *model_selection* is triggered to compare the metrics, used by the *update_state*. Any `typing.Callable` behaving like an *operator* is accepted.

Note: Model selection is done ONLY if an operator is specified here.

- **logdir** (`str`) – Path to the log dir, defaults to a *log* folder in the current directory.
- **max_val** (`float`) – The dynamic range of the images (i.e., the difference between the maximum the and minimum) (see www.tensorflow.org/versions/r2.0/api_docs/python/tf/image/ssim_multiscale)
- **power_factors** (`List[float]`) – Iterable of weights for each of the scales. The number of scales used is the length of the list. Index 0 is the unscaled resolution's weight and each increasing scale corresponds to the image being downsampled by 2. Defaults to (0.0448, 0.2856, 0.3001, 0.2363, 0.1333), which are the values obtained in the original paper.
- **filter_size** (`int`) – Default value 11 (size of gaussian filter).
- **filter_sigma** (`float`) – Default value 1.5 (width of gaussian filter).
- **k1** (`float`) – Default value 0.01.
- **k2** (`float`) – Default value 0.03 (SSIM is less sensitivity to K2 for lower values, so it would be better if we take the values in range of $0 < K2 < 0.4$).

Return type `None`

static split_batch (`batch`)

Split a batch along axis 0 into two tensors having the same size.

Parameters **batch** (`tf.Tensor`) – A batch of images.

Return type `Tuple[Tensor, Tensor]`

Returns (`Tuple[tf.Tensor, tf.Tensor]`) The batch split in two tensors.

Raises `ValueError` – if the batch has size 1.

update_state (`context`)

Update the internal state of the metric, using the information from the context object.

Parameters **context** (`ashpy.contexts.gan.GANContext`) – An AshPy Context Object that carries all the information the Metric needs.

Return type `None`

4.8 ashpy.models

Collection of Models.

Architectures

<code>convolutional</code>	Collection of Convolutional Models constructors.
<code>fc</code>	Collection of Fully Connected Models constructors.

4.8.1 convolutional

Collection of Convolutional Models constructors.

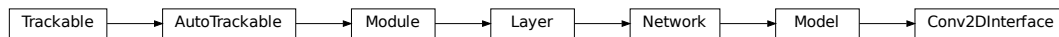
Interfaces

`interfaces.Conv2DInterface`

Primitive Interface to be used by all *ashpy.models*.

Conv2DInterface

Inheritance Diagram



class `ashpy.models.convolutional.interfaces.Conv2DInterface`

Bases: `tensorflow.python.keras.engine.training.Model`

Primitive Interface to be used by all *ashpy.models*.

Methods

<code>__init__()</code>	Primitive Interface to be used by all <i>ashpy.models</i> .
<code>call(inputs[, training, return_features])</code>	Execute the model on input data.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.

Continued on next page

Table 214 – continued from previous page

<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

`__init__()`

Primitive Interface to be used by all *ashpy.models*.

Declares the *self.model_layers* list.

static `_get_layer_spec` (*initial_filters*, *filters_cap*, *input_res*, *target_res*)

Compose the *layer_spec*, the building block of a convolutional model.

The *layer_spec* is an iterator. Every element returned is the number of filters to learn for the current layer. The generated sequence of filters starts from *initial_filters* and halve/double the number of filters depending on the *input_res* and *target_res*. If *input_res* > *target_res* the number of filters increases, else it decreases. The progression is always a power of 2.

Parameters

- **initial_filters** (*int*) – Depth of the first convolutional layer.
- **filters_cap** (*int*) – Maximum number of filters per layer.
- **input_res** (*tuple* of (*int*, *int*)) – Input resolution.
- **target_res** (*tuple* of (*int*, *int*)) – Output resolution.

Yields *int* – Number of filters to use for the conv layer.

Examples

```
# Encoder
class T(Conv2DInterface):
    pass
```

(continues on next page)

(continued from previous page)

```
spec = T._get_layer_spec(
    initial_filters=16,
    filters_cap=128,
    input_res=(512, 256),
    target_res=(32, 16)
)
print([s for s in spec])

spec = T._get_layer_spec(
    initial_filters=16,
    filters_cap=128,
    input_res=(28, 28),
    target_res=(7, 7)
)
print([s for s in spec])

# Decoder
spec = T._get_layer_spec(
    initial_filters=128,
    filters_cap=16,
    input_res=(32, 16),
    target_res=(512, 256)
)
print([s for s in spec])

spec = T._get_layer_spec(
    initial_filters=128,
    filters_cap=16,
    input_res=(7, 7),
    target_res=(28, 28)
)
print([s for s in spec])
```

```
[32, 64, 128, 128]
[32, 64]
[64, 32, 16, 16]
[64, 32]
```

Notes

This is useful since it enables us to dynamically redefine models sharing an underlying architecture but with different resolutions.

call (*inputs*, *training=True*, *return_features=False*)

Execute the model on input data.

Parameters

- **inputs** (*tf.Tensor*) – Input tensor(s).
- **training** (*bool*) – Training flag.
- **return_features** (*bool*) – If True returns the features.

Returns *tf.Tensor* – The model output.

Decoders

<code>decoders.Decoder</code>	Primitive Model for all decoder (i.e., transpose convolution) based architecture.
<code>decoders.FCNNDecoder</code>	Fully Convolutional Decoder.

Decoder

Inheritance Diagram



```
class ashpy.models.convolutional.decoders.Decoder(layer_spec_input_res,
                                                  layer_spec_target_res, kernel_size, initial_filters, filters_cap,
                                                  channels, use_dropout=True, dropout_prob=0.3,
                                                  non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>)
```

Bases: `ashpy.models.convolutional.interfaces.Conv2DInterface`

Primitive Model for all decoder (i.e., transpose convolution) based architecture.

Notes

Default to DCGAN Generator architecture.

Examples

- Direct Usage:

```
dummy_generator = Decoder(
    layer_spec_input_res=(8, 8),
    layer_spec_target_res=(64, 64),
    kernel_size=(5, 5),
    initial_filters=1024,
    filters_cap=16,
    channels=3,
)
```

- Subclassing

```
class DummyGenerator(Decoder):
    def call(self, input, training=True):
        print("Dummy Generator!")
```

(continues on next page)

(continued from previous page)

```

        return input

dummy_generator = DummyGenerator(
    layer_spec_input_res=(8, 8),
    layer_spec_target_res=(32, 32),
    kernel_size=(5, 5),
    initial_filters=1024,
    filters_cap=16,
    channels=3,
)
dummy_generator(tf.random.normal((1, 100)))
    
```

Dummy Generator!

Methods

<code>__init__(layer_spec_input_res, ...[, ...])</code>	Instantiate the <i>Decoder</i> .
---	----------------------------------

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.

Continued on next page

Table 217 – continued from previous page

<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

`__init__`(*layer_spec_input_res*, *layer_spec_target_res*, *kernel_size*, *initial_filters*, *filters_cap*, *channels*, *use_dropout=True*, *dropout_prob=0.3*, *non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>*)
 Instantiate the *Decoder*.

Model Assembly: 1. `_add_initial_block()`: Ingest the `tf.keras.Model` inputs and prepare them for `_add_building_block()`.

2. `_add_building_block()`: Core of the model, the layers specified here get added to the `tf.keras.Model` multiple times consuming the hyperparameters generated in the `_get_layer_spec()`.

3. `_add_final_block()`: Final block of our `tf.keras.Model`, take the model after `_add_building_block()` and prepare them for the final output.

Parameters

- **layer_spec_input_res** (tuple of (int, int)) – Shape of the `_get_layer_spec()` input tensors.
- **layer_spec_target_res** – (tuple of (int, int)): Shape of tensor desired as output of `_get_layer_spec()`.
- **kernel_size** (tuple of (int, int)) – Kernel used by the convolution layers.
- **initial_filters** (int) – Numbers of filters at the end of the first block.
- **filters_cap** (int) – Cap filters to a set amount, in the case of Decoder is a floor value AKA the minimum amount of filters.
- **channels** (int) – Channels of the output images (1 for Grayscale, 3 for RGB).

Returns `None`

Raises `ValueError` – If `filters_cap > initial_filters`.

`_add_building_block`(*filters*)

Construct the core of the `tf.keras.Model`.

The layers specified here get added to the `tf.keras.Model` multiple times consuming the hyperparameters generated in the `_get_layer_spec()`.

Parameters **filters** (int) – Number of filters to use for this iteration of the Building Block.

`_add_final_block`(*channels*)

Prepare results of `_add_building_block()` for the final output.

Parameters **channels** (int) – Channels of the output images (1 for Grayscale, 3 for RGB).

`_add_initial_block`(*initial_filters*, *input_res*)

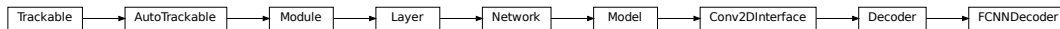
Ingest the `tf.keras.Model` inputs and prepare them for `_add_building_block()`.

Parameters

- **initial_filters** (*int*) – Numbers of filters to used as a base value.
- **input_res** (*tuple* of (*int*, *int*)) – Shape of the `_get_layer_spec()` input tensors.

FCNNDecoder

Inheritance Diagram



```

class ashpy.models.convolutional.decoders.FCNNDecoder(layer_spec_input_res,
                                                         layer_spec_target_res,
                                                         kernel_size,    initial_filters,
                                                         filters_cap,      channels,
                                                         use_dropout=True,
                                                         dropout_prob=0.3,
                                                         non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>)

```

Bases: *ashpy.models.convolutional.decoders.Decoder*

Fully Convolutional Decoder. Expected input is a feature map.

Examples

- **Direct Usage:**

```

dummy_generator = FCNNDecoder(
    layer_spec_input_res=(8, 8),
    layer_spec_target_res=(64, 64),
    kernel_size=(5, 5),
    initial_filters=1024,
    filters_cap=16,
    channels=3,
)

print(dummy_generator(tf.zeros((1, 1, 1, 100))).shape)

```

```
(1, 64, 64, 3)
```

Methods

<code>__init__(layer_spec_input_res, ...[, ...])</code>	Build a Fully Convolutional Decoder.
---	--------------------------------------

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

```
__init__(layer_spec_input_res, layer_spec_target_res, kernel_size, initial_filters, filters_cap,
         channels, use_dropout=True, dropout_prob=0.3, non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>)
```

Build a Fully Convolutional Decoder.

```
_add_initial_block(initial_filters, input_res)
```

Ingest the `tf.keras.Model` inputs and prepare them for `_add_building_block()`.

Parameters

- **initial_filters** (*int*) – Numbers of filters to used as a base value.
- **input_res** (*tuple of (int, int)*) – Shape of the `_get_layer_spec()` input ten-

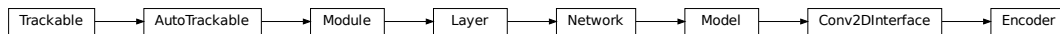
SORS.

Encoders

<code>encoders.Encoder</code>	Primitive Model for all encoder (i.e., convolution) based architecture.
<code>encoders.FCNNEncoder</code>	Fully Convolutional Encoder.

Encoder

Inheritance Diagram



```

class ashpy.models.convolutional.encoders.Encoder(layer_spec_input_res,
                                                  layer_spec_target_res, kernel_size,
                                                  initial_filters, filters_cap, out-
                                                  put_shape, use_dropout=True,
                                                  dropout_prob=0.3,
                                                  non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>)

```

Bases: `ashpy.models.convolutional.interfaces.Conv2DInterface`

Primitive Model for all encoder (i.e., convolution) based architecture.

Notes

Default to DCGAN Discriminator architecture.

Examples

- Direct Usage:

```

dummy_generator = Encoder(
    layer_spec_input_res=(64, 64),
    layer_spec_target_res=(8, 8),
    kernel_size=5,
    initial_filters=4,
    filters_cap=128,
    output_shape=1,
)

```

- Subclassing

```
class DummyDiscriminator(Encoder):
    def call(self, inputs, training=True):
        print("Dummy Discriminator!")
        # build the model using
        # self._layers and inputs
        return inputs

dummy_discriminator = DummyDiscriminator(
    layer_spec_input_res=(64, 64),
    layer_spec_target_res=(8, 8),
    kernel_size=5,
    initial_filters=16,
    filters_cap=128,
    output_shape=1,
)
dummy_discriminator(tf.zeros((1, 28, 28, 3)))
```

```
Dummy Discriminator!
```

Methods

<code>__init__(layer_spec_input_res, ...[, ...])</code>	Instantiate the Decoder.
---	--------------------------

Attributes

activity_regularizer	Optional regularizer function for the output of this layer.
dtype	
dynamic	
inbound_nodes	Deprecated, do NOT use! Only for compatibility with external Keras.
input	Retrieves the input tensor(s) of a layer.
input_mask	Retrieves the input mask tensor(s) of a layer.
input_shape	Retrieves the input shape(s) of a layer.
input_spec	Gets the network's input specs.
layers	
losses	Losses which are associated with this <i>Layer</i> .
metrics	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
metrics_names	Returns the model's display labels for all outputs.
name	Returns the name of this module as passed or determined in the ctor.
name_scope	Returns a <i>tf.name_scope</i> instance for this class.
non_trainable_variables	
non_trainable_weights	
outbound_nodes	Deprecated, do NOT use! Only for compatibility with external Keras.
output	Retrieves the output tensor(s) of a layer.
output_mask	Retrieves the output mask tensor(s) of a layer.
output_shape	Retrieves the output shape(s) of a layer.

Continued on next page

Table 222 – continued from previous page

<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

`__init__`(*layer_spec_input_res*, *layer_spec_target_res*, *kernel_size*, *initial_filters*, *filters_cap*, *output_shape*, *use_dropout=True*, *dropout_prob=0.3*, *non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>*)
 Instantiate the Decoder.

Parameters

- **`layer_spec_input_res`** (tuple of (int, int)) – Shape of the input tensors.
- **`layer_spec_target_res`** (Union[int, Tuple[int, int]]) – (tuple of (int, int)): Shape of tensor desired as output of `_get_layer_spec()`.
- **`kernel_size`** (int) – Kernel used by the convolution layers.
- **`initial_filters`** (int) – Numbers of filters to used as a base value.
- **`filters_cap`** (int) – Cap filters to a set amount, in the case of an Encoder is a ceil value AKA the max amount of filters.
- **`output_shape`** (int) – Amount of units of the last `tf.keras.layers.Dense`.

Returns None

Raises `ValueError` – If `filters_cap < initial_filters`

`_add_building_block`(*filters*)

Construct the core of the `tf.keras.Model`.

The layers specified here get added to the `tf.keras.Model` multiple times consuming the hyper-parameters generated in the `_get_layer_spec()`.

Parameters **`filters`** (int) – Number of filters to use for this iteration of the Building Block.

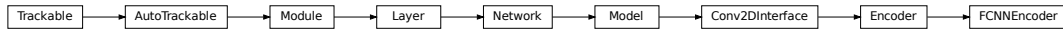
`_add_final_block`(*output_shape*)

Prepare the results of `_add_building_block()` for the final output.

Parameters **`output_shape`** (int) – Amount of units of the last `tf.keras.layers.Dense`

FCNNEncoder

Inheritance Diagram



```
class ashpy.models.convolutional.encoders.FCNNEncoder (layer_spec_input_res,  
                                                    layer_spec_target_res,  
                                                    kernel_size,    initial_filters,  
                                                    filters_cap,    encoding_dimension)
```

Bases: *ashpy.models.convolutional.encoders.Encoder*

Fully Convolutional Encoder.

Output a 1x1encoding_size vector. The output neurons are linear.

Examples

- Direct Usage:

```
dummy_generator = FCNNEncoder(  
    layer_spec_input_res=(64, 64),  
    layer_spec_target_res=(8, 8),  
    kernel_size=5,  
    initial_filters=4,  
    filters_cap=128,  
    encoding_dimension=100,  
)  
print(dummy_generator(tf.zeros((1, 64, 64, 3))).shape)
```

```
(1, 1, 1, 100)
```

Methods

<code>__init__(layer_spec_input_res, ...)</code>	Instantiate the FCNNDecoder.
--	------------------------------

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.

Continued on next page

Table 224 – continued from previous page

<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

`__init__`(*layer_spec_input_res*, *layer_spec_target_res*, *kernel_size*, *initial_filters*, *filters_cap*, *encoding_dimension*)

Instantiate the FCNNDecoder.

Parameters

- **`layer_spec_input_res`** (tuple of (int, int)) – Shape of the input tensors.
- **`layer_spec_target_res`** – (tuple of (int, int)): Shape of tensor desired as output of `_get_layer_spec()`.
- **`kernel_size`** (int) – Kernel used by the convolution layers.
- **`initial_filters`** (int) – Numbers of filters to used as a base value.
- **`filters_cap`** (int) – Cap filters to a set amount, in the case of an Encoder is a ceil value AKA the max amount of filters.
- **`encoding_dimension`** (int) – encoding dimension.

Returns `None`

Raises `ValueError` – If *filters_cap* < *initial_filters*

`_add_final_block(output_shape)`

Prepare the results of `_add_building_block()` for the final output.

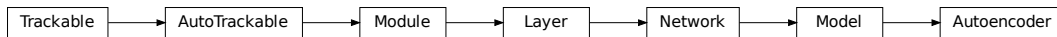
Parameters `output_shape` (*int*) – Amount of units of the last `tf.keras.layers.Dense`

Autoencoders

<code>autoencoders.Autoencoder</code>	Primitive Model for all convolutional autoencoders.
<code>autoencoders.FCNNAutoencoder</code>	Primitive Model for all fully convolutional autoencoders.

Autoencoder

Inheritance Diagram



```

class ashpy.models.convolutional.autoencoders.Autoencoder(layer_spec_input_res,
                                                         layer_spec_target_res,
                                                         kernel_size,      initial_filters, filters_cap,
                                                         encoding_dimension,
                                                         channels)

```

Bases: `tensorflow.python.keras.engine.training.Model`

Primitive Model for all convolutional autoencoders.

Examples

- Direct Usage:

```

autoencoder = Autoencoder(
    layer_spec_input_res=(64, 64),
    layer_spec_target_res=(8, 8),
    kernel_size=5,
    initial_filters=32,
    filters_cap=128,
    encoding_dimension=100,
    channels=3,
)

encoding, reconstruction = autoencoder(tf.zeros((1, 64, 64, 3)))
print(encoding.shape)
print(reconstruction.shape)

```


Methods

<code>__init__(layer_spec_input_res, ...)</code>	Instantiate the BaseAutoEncoder.
<code>call(inputs[, training])</code>	Execute the model on input data.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

`__init__` (*layer_spec_input_res*, *layer_spec_target_res*, *kernel_size*, *initial_filters*, *filters_cap*, *encoding_dimension*, *channels*)
 Instantiate the BaseAutoEncoder.

Parameters

- **layer_spec_input_res** (*tuple* of (*int*, *int*)) – Shape of the input tensors.
- **layer_spec_target_res** – (*tuple* of (*int*, *int*)): Shape of tensor desired as output of `_get_layer_spec()`.
- **kernel_size** (*int*) – Kernel used by the convolution layers.
- **initial_filters** (*int*) – Numbers of filters to used as a base value.
- **filters_cap** (*int*) – Cap filters to a set amount, in the case of an Encoder is a ceil value AKA the max amount of filters.
- **encoding_dimension** (*int*) – encoding dimension.
- **channels** (*int*) – Number of channels for the reconstructed image.

Returns `None`

call (*inputs*, *training=True*)
 Execute the model on input data.

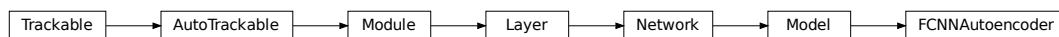
Parameters

- **inputs** (*tf.Tensor*) – Input tensors.
- **training** (*bool*) – Training flag.

Returns (*encoding*, *reconstruction*) – Pair of tensors.

FCNNAutoencoder

Inheritance Diagram



```

class ashpy.models.convolutional.autoencoders.FCNNAutoencoder(layer_spec_input_res,
                                                                layer_spec_target_res,
                                                                kernel_size, ini-
                                                                tial_filters, fil-
                                                                ters_cap, encod-
                                                                ing_dimension,
                                                                channels)
  
```

Bases: `tensorflow.python.keras.engine.training.Model`

Primitive Model for all fully convolutional autoencoders.

Examples

- **Direct Usage:**

```
autoencoder = FCNNAutoencoder(
    layer_spec_input_res=(64, 64),
    layer_spec_target_res=(8, 8),
    kernel_size=5,
    initial_filters=32,
    filters_cap=128,
    encoding_dimension=100,
    channels=3,
)

encoding, reconstruction = autoencoder(tf.zeros((1, 64, 64, 3)))
print(encoding.shape)
print(reconstruction.shape)
```

Methods

<code>__init__(layer_spec_input_res, ...)</code>	Instantiate the FCNNBaseAutoEncoder.
<code>call(inputs[, training])</code>	Execute the model on input data.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.

Continued on next page

Table 229 – continued from previous page

<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

`__init__` (*layer_spec_input_res*, *layer_spec_target_res*, *kernel_size*, *initial_filters*, *filters_cap*, *encoding_dimension*, *channels*)

Instantiate the FCNNBaseAutoEncoder.

Parameters

- **`layer_spec_input_res`** (tuple of (*int*, *int*)) – Shape of the input tensors.
- **`layer_spec_target_res`** – (tuple of (*int*, *int*)): Shape of tensor desired as output of `_get_layer_spec()`.
- **`kernel_size`** (*int*) – Kernel used by the convolution layers.
- **`initial_filters`** (*int*) – Numbers of filters to used as a base value.
- **`filters_cap`** (*int*) – Cap filters to a set amount, in the case of an Encoder is a ceil value AKA the max amount of filters.
- **`encoding_dimension`** (*int*) – encoding dimension.
- **`channels`** (*int*) – Number of channels for the reconstructed image.

Returns `None`

`call` (*inputs*, *training=True*)

Execute the model on input data.

Parameters

- **`inputs`** (*tf.Tensor*) – Input tensors.
- **`training`** (*bool*) – Training flag.

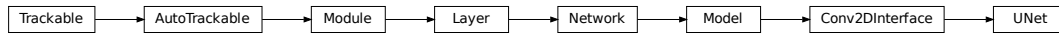
Returns (*encoding*, *reconstruction*) – Pair of tensors.

UNet

<code>unet.UNet</code>	UNet Architecture.
<code>unet.SUNet</code>	Semantic UNet.
<code>unet.FUNet</code>	Functional UNET Implementation.

UNet

Inheritance Diagram



```

class ashpy.models.convolutional.unet.UNet(input_res,      min_res,      kernel_size,
                                           initial_filters, filters_cap, channels,
                                           use_dropout_encoder=True,
                                           use_dropout_decoder=True,
                                           dropout_prob=0.3,
                                           encoder_non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>,
                                           decoder_non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.ReLU'>,
                                           normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceNormalization'>,
                                           last_activation=<function tanh>,
                                           use_attention=False)

```

Bases: *ashpy.models.convolutional.interfaces.Conv2DInterface*

UNet Architecture.

Architecture similar to the one found in “Image-to-Image Translation with Conditional Adversarial Nets”¹.

Originally proposed in “U-Net: Convolutional Networks for Biomedical Image Segmentation”².

Examples

- Direct Usage:

```

x = tf.ones((1, 512, 512, 3))
u_net = UNet(input_res = 512,
              min_res=4,
              kernel_size=4,
              initial_filters=64,
              filters_cap=512,
              channels=3)

y = u_net(x)
print(y.shape)
print(len(u_net.trainable_variables)>0)

```

```

(1, 512, 512, 3)
True

```

¹ Image-to-Image Translation with Conditional Adversarial Nets - <https://arxiv.org/abs/1611.07004>

² U-Net: Convolutional Networks for Biomedical Image Segmentation - <https://arxiv.org/abs/1505.04597>

Methods

<code>__init__(input_res, min_res, kernel_size, ...)</code>	Initialize the UNet.
<code>call(inputs[, training])</code>	Forward pass of the UNet model.
<code>get_decoder_block(filters[, use_bn, ...])</code>	Return a block to be used in the decoder part of the UNET.
<code>get_encoder_block(filters[, use_bn, ...])</code>	Return a block to be used in the encoder part of the UNET.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

```
__init__(input_res, min_res, kernel_size, initial_filters, filters_cap, chan-
nels, use_dropout_encoder=True, use_dropout_decoder=True,
dropout_prob=0.3, encoder_non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>, de-
coder_non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.ReLU'>,
normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceNormalization'>,
last_activation=<function tanh>, use_attention=False)
```

Initialize the UNet.

Parameters

- **input_res** (`int`) – input resolution.
- **min_res** (`int`) – minimum resolution reached after decode.
- **kernel_size** (`int`) – kernel size used in the network.
- **initial_filters** (`int`) – number of filter of the initial convolution.
- **filters_cap** (`int`) – maximum number of filters.
- **channels** (`int`) – number of output channels.
- **use_dropout_encoder** (`bool`) – whether to use dropout in the encoder module.
- **use_dropout_decoder** (`bool`) – whether to use dropout in the decoder module.
- **dropout_prob** (`float`) – probability of dropout.
- **encoder_non_linearity** (`Type[Layer]`) – non linearity of encoder.
- **decoder_non_linearity** (`Type[Layer]`) – non linearity of decoder.
- **last_activation** (`<module 'tensorflow_core.keras.activations' from '/home/docs/checkouts/readthedocs.org/user_builds/ashpy/envs/v0.4.0/lib/python3.7/site-packages/tensorflow_core/python/keras/api/_v2/keras/activations/__init__.py'>`) – last activation function, tanh or softmax (for semantic images).
- **use_attention** (`bool`) – whether to use attention.

call (*inputs*, *training=False*)

Forward pass of the UNet model.

get_decoder_block (*filters*, *use_bn=True*, *use_dropout=False*, *use_attention=False*)

Return a block to be used in the decoder part of the UNET.

Parameters

- **filters** – number of filters
- **use_bn** – whether to use batch normalization
- **use_dropout** – whether to use dropout
- **use_attention** – whether to use attention

Returns A block to be used in the decoder part

get_encoder_block (*filters*, *use_bn=True*, *use_attention=False*)

Return a block to be used in the encoder part of the UNET.

Parameters

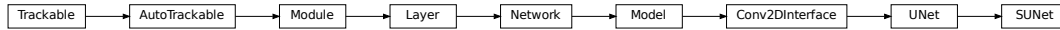
- **filters** – number of filters.
- **use_bn** – whether to use batch normalization.

- **use_attention** – whether to use attention.

Returns A block to be used in the encoder part.

SUNet

Inheritance Diagram



```

class ashpy.models.convolutional.unet.SUNet(input_res, min_res, kernel_size,
                                             initial_filters, filters_cap, chan-
                                             nels, use_dropout_encoder=True,
                                             use_dropout_decoder=True,
                                             dropout_prob=0.3, en-
                                             coder_non_linearity=<class 'tensor-
                                             flow.python.keras.layers.advanced_activations.LeakyReLU'>,
                                             decoder_non_linearity=<class 'tensor-
                                             flow.python.keras.layers.advanced_activations.ReLU'>,
                                             use_attention=False)

```

Bases: *ashpy.models.convolutional.unet.UNet*

Semantic UNet.

Methods

<code>__init__(input_res, min_res, kernel_size, ...)</code>	Build the Semantic UNet model.
---	--------------------------------

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.

Continued on next page

Table 234 – continued from previous page

<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

```

__init__(input_res,      min_res,      kernel_size,      initial_filters,      filters_cap,      chan-
        nels,          use_dropout_encoder=True,          use_dropout_decoder=True,
        dropout_prob=0.3,          encoder_non_linearity=<class      'tensor-
        flow.python.keras.layers.advanced_activations.LeakyReLU'>,          de-
        coder_non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.ReLU'>,
        use_attention=False)
    Build the Semantic UNet model.
    
```

ashpy.models.convolutional.unet.FUNet

```

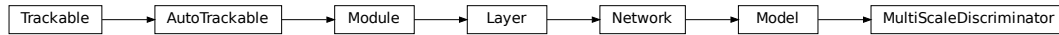
ashpy.models.convolutional.unet.FUNet(input_res,      min_res,      kernel_size,      ini-
        tial_filters,      filters_cap,      channels,      in-
        put_channels=3,          use_dropout_encoder=True,
        use_dropout_decoder=True,      dropout_prob=0.3,
        encoder_non_linearity=<class      'tensor-
        flow.python.keras.layers.advanced_activations.LeakyReLU'>,
        decoder_non_linearity=<class      'tensor-
        flow.python.keras.layers.advanced_activations.ReLU'>,
        last_activation=<function      tanh>,
        use_attention=False)
    Functional UNET Implementation.
    
```

Discriminators

<code>discriminators.MultiScaleDiscriminator</code>	Multi-Scale discriminator.
<code>discriminators.PatchDiscriminator</code>	Pix2Pix discriminator.

MultiScaleDiscriminator

Inheritance Diagram



```
class ashpy.models.convolutional.discriminators.MultiScaleDiscriminator(input_res,
min_res,
ker-
nel_size,
ini-
tial_filters,
fil-
ters_cap,
use_dropout=True,
dropout_prob=0.3,
non_linearity=<class
'ten-
sor-
flow.python.keras.layers.ad
nor-
mal-
iza-
tion_layer=<class
'ashpy.layers.instance_norm
use_attention=False,
n_discriminators=1)
```

Bases: tensorflow.python.keras.engine.training.Model

Multi-Scale discriminator.

This discriminator architecture is composed by multiple discriminators working at different scales. Each discriminator is a `ashpy.models.convolutional.discriminators.PatchDiscriminator`.

Examples

```
x = tf.ones((1, 256, 256, 3))

# instantiate the PathDiscriminator
multiScaleDiscriminator = MultiScaleDiscriminator(input_res=256,
min_res=16,
kernel_size=5,
initial_filters=64,
```

(continues on next page)

(continued from previous page)

```

filters_cap=512,
n_discriminators=3
)

# evaluate passing x
outputs = multiScaleDiscriminator(x)

# the output shape is
# the same as the input shape
print(len(outputs))
for output in outputs:
    print(output.shape)

```

```

3
(1, 12, 12, 1)
(1, 12, 12, 1)
(1, 12, 12, 1)

```

Methods

<code>__init__(input_res, min_res, kernel_size, ...)</code>	Multi Scale Discriminator.
<code>build_discriminator(input_res)</code>	Build a single discriminator using parameters defined in this object.
<code>call(inputs[, training, return_features])</code>	Forward pass of the Multi Scale Discriminator.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.

Continued on next page

Table 237 – continued from previous page

<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

```
__init__(input_res, min_res, kernel_size, initial_filters, filters_cap,
         use_dropout=True, dropout_prob=0.3, non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>,
         normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceNormalization'>, use_attention=False,
         n_discriminators=1)
```

Multi Scale Discriminator.

Different generator for different scales of the input image.

Used by Pix2PixHD¹.

Parameters

- **input_res** (*int*) – input resolution
- **min_res** (*int*) – minimum resolution reached by the discriminators
- **kernel_size** (*int*) – kernel size of discriminators
- **initial_filters** (*int*) – number of initial filters in the first layer of the discriminators
- **filters_cap** (*int*) – maximum number of filters in the discriminators
- **use_dropout** (*bool*) – whether to use dropout
- **dropout_prob** (*float*) – probability of dropout
- **non_linearity** (*tf.keras.layers.Layer*) – non linearity used in discriminators
- **normalization_layer** (*tf.keras.layers.Layer*) – normalization used by the discriminators
- **use_attention** (*bool*) – whether to use attention
- **n_discriminators** (*int*) – Number of discriminators

build_discriminator (*input_res*)

Build a single discriminator using parameters defined in this object.

Parameters **input_res** – input resolution of the discriminator.

¹ High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs <https://arxiv.org/abs/1711.11585>

Return type *Encoder*

Returns A Discriminator (PatchDiscriminator).

call (*inputs, training=True, return_features=False*)
Forward pass of the Multi Scale Discriminator.

Parameters

- **inputs** (*tf.Tensor*) – input tensor.
- **training** (*bool*) – whether is training or not.
- **return_features** (*bool*) – whether to return features or not.

Return type *Union[List[Tensor], Tuple[List[Tensor], List[Tensor]]]*

Returns

(*tf.Tensor*) –

A List of Tensors containing the value of D_i for each input.

(*tf.Tensor*): A List of features for each discriminator if *return_features*.

PatchDiscriminator

Inheritance Diagram



```

class ashpy.models.convolutional.discriminators.PatchDiscriminator(input_res,
                                                                    min_res,
                                                                    ker-
                                                                    nel_size,
                                                                    ini-
                                                                    tial_filters,
                                                                    fil-
                                                                    ters_cap,
                                                                    use_dropout=True,
                                                                    dropout_prob=0.3,
                                                                    non_linearity=<class
                                                                    'tensor-
                                                                    flow.python.keras.layers.advanced_
                                                                    nor-
                                                                    maliza-
                                                                    tion_layer=<class
                                                                    'ashpy.layers.instance_normalizati
                                                                    use_attention=False)
    
```

Bases: *ashpy.models.convolutional.encoders.Encoder*

Pix2Pix discriminator.

The last layer is an image in which each pixels is the probability of being fake or real.

Examples

```
x = tf.ones((1, 64, 64, 3))

# instantiate the PatchDiscriminator
patchDiscriminator = PatchDiscriminator(input_res=64,
                                       min_res=16,
                                       kernel_size=5,
                                       initial_filters=64,
                                       filters_cap=512,
                                       )

# evaluate passing x
output = patchDiscriminator(x)

# the output shape is the same as the input shape
print(output.shape)
```

```
(1, 12, 12, 1)
```

Methods

<code>__init__(input_res, min_res, kernel_size, ...)</code>	Patch Discriminator used by pix2pix.
<code>call(inputs[, training, return_features])</code>	Forward pass of the PatchDiscriminator.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.

Continued on next page

Table 239 – continued from previous page

<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

```
__init__(input_res, min_res, kernel_size, initial_filters, filters_cap,
         use_dropout=True, dropout_prob=0.3, non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>,
         normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceNormalization'>, use_attention=False)
```

Patch Discriminator used by pix2pix.

When `min_res=1` this is the same as a standard fully convolutional discriminator.

Parameters

- **`input_res`** (*int*) – Input Resolution.
- **`min_res`** (*int*) – Minimum Resolution reached by the discriminator.
- **`kernel_size`** (*int*) – Kernel Size used in Conv Layer.
- **`initial_filters`** (*int*) – number of filters in the first convolutional layer.
- **`filters_cap`** (*int*) – Maximum number of filters.
- **`use_dropout`** (*bool*) – whether to use dropout.
- **`dropout_prob`** (*float*) – probability of dropout.
- **`non_linearity`** (*tf.keras.layers.Layer*) – non linearity used in the model.
- **`normalization_layer`** (*tf.keras.layers.Layer*) – normalization layer used in the model.
- **`use_attention`** (*bool*) – whether to use attention.

```
_add_building_block(filters, use_bn=False)
```

Construct the core of the `tf.keras.Model`.

The layers specified here get added to the `tf.keras.Model` multiple times consuming the hyper-parameters generated in the `_get_layer_spec()`.

Parameters **`filters`** (*int*) – Number of filters to use for this iteration of the Building Block.

```
_add_final_block(output_shape)
```

Prepare the results of `_add_building_block()` for the final output.

Parameters **`output_shape`** (*int*) – Amount of units of the last `tf.keras.layers.Dense`

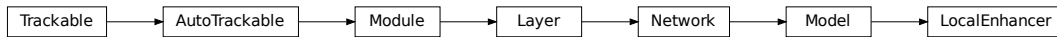
call (*inputs, training=False, return_features=False*)
Forward pass of the PatchDiscriminator.

Pix2PixHD

<code>pix2pixhd.LocalEnhancer</code>	Local Enhancer module of the Pix2PixHD architecture.
<code>pix2pixhd.GlobalGenerator</code>	Global Generator from pix2pixHD paper.

LocalEnhancer

Inheritance Diagram



```

class ashpy.models.convolutional.pix2pixhd.LocalEnhancer(input_res=512,
                                                         min_res=64,      ini-
                                                         tial_filters=64,    fil-
                                                         ters_cap=512,      chan-
                                                         nels=3,           normaliza-
                                                         tion_layer=<class
                                                         'ashpy.layers.instance_normalization.InstanceNormaliza-
                                                         tion'>,
                                                         non_linearity=<class
                                                         'tensorflow.layers.advanced_activations.relu'>,
                                                         num_resnet_blocks_global=9,
                                                         num_resnet_blocks_local=3,
                                                         kernel_size_resnet=3,
                                                         kernel_size_front_back=7,
                                                         num_internal_resnet_blocks=2)

```

Bases: tensorflow.python.keras.engine.training.Model

Local Enhancer module of the Pix2PixHD architecture.

Example

```

# instantiate the model
model = LocalEnhancer()

# call the model passing inputs
inputs = tf.ones((1, 512, 512, 3))
output = model(inputs)

```

(continues on next page)

(continued from previous page)

```
# the output shape is
# the same as the input shape
print(output.shape)
```

```
(1, 512, 512, 3)
```

Methods

<code>__init__([input_res, min_res, ...])</code>	Build the LocalEnhancer module of the Pix2PixHD architecture.
<code>call(inputs[, training])</code>	Call the LocalEnhancer model.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	

Continued on next page

Table 242 – continued from previous page

<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

```
__init__(input_res=512, min_res=64, initial_filters=64, filters_cap=512, channels=3, normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceNormalization'>, non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.ReLU'>, num_resnet_blocks_global=9, num_resnet_blocks_local=3, kernel_size_resnet=3, kernel_size_front_back=7, num_internal_resnet_blocks=2)
```

Build the LocalEnhancer module of the Pix2PixHD architecture.

See High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs² for more details.

Parameters

- **`input_res`** (*int*) – input resolution.
- **`min_res`** (*int*) – minimum resolution reached by the global generator.
- **`initial_filters`** (*int*) – number of initial filters.
- **`filters_cap`** (*int*) – maximum number of filters.
- **`channels`** (*int*) – number of channels.
- **`normalization_layer`** (*tf.keras.layers.Layer*) – layer of normalization
- **Instance Normalization or BatchNormalization or LayerNormalization** (*(e.g.)*) –
- **`non_linearity`** (*tf.keras.layers.Layer*) – non linearity used in Pix2Pix HD.
- **`num_resnet_blocks_global`** (*int*) – number of residual blocks used in the global generator.
- **`num_resnet_blocks_local`** (*int*) – number of residual blocks used in the local generator.
- **`kernel_size_resnet`** (*int*) – kernel size used in resnets.
- **`kernel_size_front_back`** (*int*) – kernel size used for the front and back convolution.
- **`num_internal_resnet_blocks`** (*int*) – number of internal blocks of the resnet.

`call` (*inputs*, *training=False*)

Call the LocalEnhancer model.

Parameters

- **`inputs`** (*tf.Tensor*) – Input Tensors.
- **`training`** (*bool*) – Whether it is training phase or not.

Returns

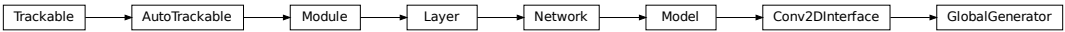
(*tf.Tensor*) –

² High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs <https://arxiv.org/abs/1711.11585>

Image of size (input_res, input_res, channels) as specified in the init call.

GlobalGenerator

Inheritance Diagram



```
class ashpy.models.convolutional.pix2pixhd.GlobalGenerator (input_res=512,
                                                            min_res=64,      ini-
                                                            tial_filters=64,
                                                            filters_cap=512,
                                                            channels=3,  normal-
                                                            ization_layer=<class
                                                            'ashpy.layers.instance_normalization.Instance
                                                            non_linearity=<class
                                                            'tensor-
                                                            flow.python.keras.layers.advanced_activations.
                                                            num_resnet_blocks=9,
                                                            kernel_size_resnet=3,
                                                            ker-
                                                            nel_size_front_back=7,
                                                            num_internal_resnet_blocks=2)
```

Bases: *ashpy.models.convolutional.interfaces.Conv2DInterface*

Global Generator from pix2pixHD paper.

- G1^F: Convolutional frontend (downsampling)
- G1^R: ResNet Block
- G1^B: Convolutional backend (upsampling)

Methods

<code>__init__</code> ([input_res, min_res, ...])	Global Generator from Pix2PixHD.
<code>call</code> (inputs[, training])	Call of the Pix2Pix HD model.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	

Continued on next page

Table 244 – continued from previous page

<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

```
__init__(input_res=512, min_res=64, initial_filters=64, filters_cap=512, channels=3, normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceNormalization'>, non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.ReLU'>, num_resnet_blocks=9, kernel_size_resnet=3, kernel_size_front_back=7, num_internal_resnet_blocks=2)
Global Generator from Pix2PixHD.
```

Parameters

- **input_res** (*int*) – Input Resolution.
- **min_res** (*int*) – Minimum resolution reached by the downsampling.
- **initial_filters** (*int*) – number of initial filters.
- **filters_cap** (*int*) – maximum number of filters.
- **channels** (*int*) – output channels.
- **normalization_layer** (*tf.keras.layers.Layer*) – normalization layer used by the global generator, can be Instance Norm, Layer Norm, Batch Norm.

- **non_linearity** (`tf.keras.layers.Layer`) – non linearity used in the global generator.
- **num_resnet_blocks** (`int`) – number of resnet blocks.
- **kernel_size_resnet** (`int`) – kernel size used in resnets conv layers.
- **kernel_size_front_back** (`int`) – kernel size used by the convolutional frontend and backend.
- **num_internal_resnet_blocks** (`int`) – number of blocks used by internal resnet.

call (*inputs*, *training=True*)

Call of the Pix2Pix HD model.

Parameters

- **inputs** – input tensor(s).
- **training** – If True training phase.

Returns `Tuple` – Generated images.

Modules

<i>autoencoders</i>	Collection of Fully Convolutional Autoencoders.
<i>discriminators</i>	Convolutional Discriminators.
<i>decoders</i>	Collection of Decoders (i.e., GANs' Generators) models.
<i>encoders</i>	Collection of Encoders (i.e., GANs' Discriminators) models.
<i>interfaces</i>	Primitive Convolutional interfaces.
<i>unet</i>	UNET implementations.
<i>pix2pixhd</i>	Pix2Pix HD Implementation.

autoencoders

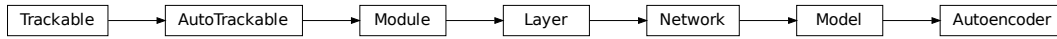
Collection of Fully Convolutional Autoencoders.

Classes

<i>Autoencoder</i>	Primitive Model for all convolutional autoencoders.
<i>FCNNAutoencoder</i>	Primitive Model for all fully convolutional autoencoders.

Autoencoder

Inheritance Diagram



```

class ashpy.models.convolutional.autoencoders.Autoencoder(layer_spec_input_res,
                                                            layer_spec_target_res,
                                                            kernel_size,          ini-
                                                            tial_filters, filters_cap,
                                                            encoding_dimension,
                                                            channels)

```

Bases: tensorflow.python.keras.engine.training.Model

Primitive Model for all convolutional autoencoders.

Examples

- Direct Usage:

```

autoencoder = Autoencoder(
    layer_spec_input_res=(64, 64),
    layer_spec_target_res=(8, 8),
    kernel_size=5,
    initial_filters=32,
    filters_cap=128,
    encoding_dimension=100,
    channels=3,
)

encoding, reconstruction = autoencoder(tf.zeros((1, 64, 64, 3)))
print (encoding.shape)
print (reconstruction.shape)

```

Methods

<code>__init__(layer_spec_input_res, ...)</code>	Instantiate the BaseAutoEncoder.
<code>call(inputs[, training])</code>	Execute the model on input data.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	

Continued on next page

Table 248 – continued from previous page

dynamic	
inbound_nodes	Deprecated, do NOT use! Only for compatibility with external Keras.
input	Retrieves the input tensor(s) of a layer.
input_mask	Retrieves the input mask tensor(s) of a layer.
input_shape	Retrieves the input shape(s) of a layer.
input_spec	Gets the network's input specs.
layers	
losses	Losses which are associated with this <i>Layer</i> .
metrics	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
metrics_names	Returns the model's display labels for all outputs.
name	Returns the name of this module as passed or determined in the ctor.
name_scope	Returns a <i>tf.name_scope</i> instance for this class.
non_trainable_variables	
non_trainable_weights	
outbound_nodes	Deprecated, do NOT use! Only for compatibility with external Keras.
output	Retrieves the output tensor(s) of a layer.
output_mask	Retrieves the output mask tensor(s) of a layer.
output_shape	Retrieves the output shape(s) of a layer.
run_eagerly	Settable attribute indicating whether the model should run eagerly.
sample_weights	
state_updates	Returns the <i>updates</i> from all layers that are stateful.
stateful	
submodules	Sequence of all sub-modules.
trainable	
trainable_variables	Sequence of variables owned by this module and it's submodules.
trainable_weights	
updates	
variables	Returns the list of all layer variables/weights.
weights	Returns the list of all layer variables/weights.

__init__ (*layer_spec_input_res*, *layer_spec_target_res*, *kernel_size*, *initial_filters*, *filters_cap*, *encoding_dimension*, *channels*)

Instantiate the BaseAutoEncoder.

Parameters

- **layer_spec_input_res** (tuple of (int, int)) – Shape of the input tensors.
- **layer_spec_target_res** – (tuple of (int, int)): Shape of tensor desired as output of `_get_layer_spec()`.
- **kernel_size** (int) – Kernel used by the convolution layers.
- **initial_filters** (int) – Numbers of filters to used as a base value.
- **filters_cap** (int) – Cap filters to a set amount, in the case of an Encoder is a ceil value AKA the max amount of filters.
- **encoding_dimension** (int) – encoding dimension.

- **channels** (*int*) – Number of channels for the reconstructed image.

Returns *None*

call (*inputs, training=True*)

Execute the model on input data.

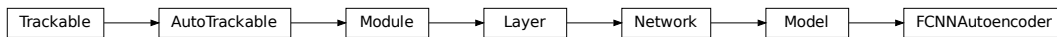
Parameters

- **inputs** (*tf.Tensor*) – Input tensors.
- **training** (*bool*) – Training flag.

Returns (*encoding, reconstruction*) – Pair of tensors.

FCNNAutoencoder

Inheritance Diagram



```

class ashpy.models.convolutional.autoencoders.FCNNAutoencoder(layer_spec_input_res,
                                                                layer_spec_target_res,
                                                                kernel_size, ini-
                                                                tial_filters, fil-
                                                                ters_cap, encod-
                                                                ing_dimension,
                                                                channels)
  
```

Bases: tensorflow.python.keras.engine.training.Model

Primitive Model for all fully convolutional autoencoders.

Examples

- **Direct Usage:**

```

autoencoder = FCNNAutoencoder(
    layer_spec_input_res=(64, 64),
    layer_spec_target_res=(8, 8),
    kernel_size=5,
    initial_filters=32,
    filters_cap=128,
    encoding_dimension=100,
    channels=3,
)

encoding, reconstruction = autoencoder(tf.zeros((1, 64, 64, 3)))
print (encoding.shape)
print (reconstruction.shape)
  
```


Methods

<code>__init__(layer_spec_input_res, ...)</code>	Instantiate the <code>FCNNBaseAutoEncoder</code> .
<code>call(inputs[, training])</code>	Execute the model on input data.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

`__init__` (*layer_spec_input_res*, *layer_spec_target_res*, *kernel_size*, *initial_filters*, *filters_cap*, *encoding_dimension*, *channels*)
 Instantiate the FCNNBaseAutoEncoder.

Parameters

- **layer_spec_input_res** (tuple of (int, int)) – Shape of the input tensors.
- **layer_spec_target_res** – (tuple of (int, int)): Shape of tensor desired as output of `_get_layer_spec()`.
- **kernel_size** (int) – Kernel used by the convolution layers.
- **initial_filters** (int) – Numbers of filters to used as a base value.
- **filters_cap** (int) – Cap filters to a set amount, in the case of an Encoder is a ceil value AKA the max amount of filters.
- **encoding_dimension** (int) – encoding dimension.
- **channels** (int) – Number of channels for the reconstructed image.

Returns None

call (*inputs*, *training=True*)
 Execute the model on input data.

Parameters

- **inputs** (tf.Tensor) – Input tensors.
- **training** (bool) – Training flag.

Returns (*encoding*, *reconstruction*) – Pair of tensors.

class ashpy.models.convolutional.autoencoders.**Autoencoder** (*layer_spec_input_res*, *layer_spec_target_res*, *kernel_size*, *initial_filters*, *filters_cap*, *encoding_dimension*, *channels*)

Bases: tensorflow.python.keras.engine.training.Model

Primitive Model for all convolutional autoencoders.

Examples

- Direct Usage:

```
autoencoder = Autoencoder(
    layer_spec_input_res=(64, 64),
    layer_spec_target_res=(8, 8),
    kernel_size=5,
    initial_filters=32,
    filters_cap=128,
    encoding_dimension=100,
    channels=3,
)

encoding, reconstruction = autoencoder(tf.zeros((1, 64, 64, 3)))
print(encoding.shape)
print(reconstruction.shape)
```

```
__init__(layer_spec_input_res, layer_spec_target_res, kernel_size, initial_filters, filters_cap, encoding_dimension, channels)
```

Instantiate the BaseAutoEncoder.

Parameters

- **layer_spec_input_res** (tuple of (int, int)) – Shape of the input tensors.
- **layer_spec_target_res** – (tuple of (int, int)): Shape of tensor desired as output of `_get_layer_spec()`.
- **kernel_size** (int) – Kernel used by the convolution layers.
- **initial_filters** (int) – Numbers of filters to used as a base value.
- **filters_cap** (int) – Cap filters to a set amount, in the case of an Encoder is a ceil value AKA the max amount of filters.
- **encoding_dimension** (int) – encoding dimension.
- **channels** (int) – Number of channels for the reconstructed image.

Returns None

```
call(inputs, training=True)
```

Execute the model on input data.

Parameters

- **inputs** (tf.Tensor) – Input tensors.
- **training** (bool) – Training flag.

Returns (encoding, reconstruction) – Pair of tensors.

```
class ashpy.models.convolutional.autoencoders.FCNNAutoencoder(layer_spec_input_res,
layer_spec_target_res,
kernel_size, initial_filters, filters_cap, encoding_dimension,
channels)
```

Bases: tensorflow.python.keras.engine.training.Model

Primitive Model for all fully convolutional autoencoders.

Examples

• Direct Usage:

```
autoencoder = FCNNAutoencoder(
    layer_spec_input_res=(64, 64),
    layer_spec_target_res=(8, 8),
    kernel_size=5,
    initial_filters=32,
    filters_cap=128,
    encoding_dimension=100,
    channels=3,
)
```

(continues on next page)

(continued from previous page)

```
encoding, reconstruction = autoencoder(tf.zeros((1, 64, 64, 3)))
print(encoding.shape)
print(reconstruction.shape)
```

__init__ (*layer_spec_input_res*, *layer_spec_target_res*, *kernel_size*, *initial_filters*, *filters_cap*, *encoding_dimension*, *channels*)

Instantiate the FCNNBaseAutoEncoder.

Parameters

- **layer_spec_input_res** (tuple of (int, int)) – Shape of the input tensors.
- **layer_spec_target_res** – (tuple of (int, int)): Shape of tensor desired as output of `_get_layer_spec()`.
- **kernel_size** (int) – Kernel used by the convolution layers.
- **initial_filters** (int) – Numbers of filters to used as a base value.
- **filters_cap** (int) – Cap filters to a set amount, in the case of an Encoder is a ceil value AKA the max amount of filters.
- **encoding_dimension** (int) – encoding dimension.
- **channels** (int) – Number of channels for the reconstructed image.

Returns None

call (*inputs*, *training=True*)

Execute the model on input data.

Parameters

- **inputs** (tf.Tensor) – Input tensors.
- **training** (bool) – Training flag.

Returns (*encoding*, *reconstruction*) – Pair of tensors.

discriminators

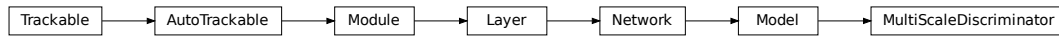
Convolutional Discriminators.

Classes

<i>MultiScaleDiscriminator</i>	Multi-Scale discriminator.
<i>PatchDiscriminator</i>	Pix2Pix discriminator.

MultiScaleDiscriminator

Inheritance Diagram



```

class ashpy.models.convolutional.discriminators.MultiScaleDiscriminator(input_res,
                                                                    min_res,
                                                                    ker-
                                                                    nel_size,
                                                                    ini-
                                                                    tial_filters,
                                                                    fil-
                                                                    ters_cap,
                                                                    use_dropout=True,
                                                                    dropout_prob=0.3,
                                                                    non_linearity=<class
                                                                    'ten-
                                                                    sor-
                                                                    flow.python.keras.layers.ad-
                                                                    nor-
                                                                    mal-
                                                                    iza-
                                                                    tion_layer=<class
                                                                    'ashpy.layers.instance_norm
                                                                    use_attention=False,
                                                                    n_discriminators=1)

```

Bases: tensorflow.python.keras.engine.training.Model

Multi-Scale discriminator.

This discriminator architecture is composed by multiple discriminators working at different scales. Each discriminator is a [ashpy.models.convolutional.discriminators.PatchDiscriminator](#).

Examples

```

x = tf.ones((1, 256, 256, 3))

# instantiate the PatchDiscriminator
multiScaleDiscriminator = MultiScaleDiscriminator(input_res=256,
                                                    min_res=16,
                                                    kernel_size=5,
                                                    initial_filters=64,
                                                    filters_cap=512,
                                                    n_discriminators=3
                                                    )

# evaluate passing x
outputs = multiScaleDiscriminator(x)

```

(continues on next page)

(continued from previous page)

```
# the output shape is
# the same as the input shape
print(len(outputs))
for output in outputs:
    print(output.shape)
```

```
3
(1, 12, 12, 1)
(1, 12, 12, 1)
(1, 12, 12, 1)
```

Methods

<code>__init__(input_res, min_res, kernel_size, ...)</code>	Multi Scale Discriminator.
<code>build_discriminator(input_res)</code>	Build a single discriminator using parameters defined in this object.
<code>call(inputs[, training, return_features])</code>	Forward pass of the Multi Scale Discriminator.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.

Continued on next page

Table 253 – continued from previous page

<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

```
__init__(input_res, min_res, kernel_size, initial_filters, filters_cap,
         use_dropout=True, dropout_prob=0.3, non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>,
         normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceNormalization'>, use_attention=False,
         n_discriminators=1)
```

Multi Scale Discriminator.

Different generator for different scales of the input image.

Used by Pix2PixHD¹.

Parameters

- **input_res** (*int*) – input resolution
- **min_res** (*int*) – minimum resolution reached by the discriminators
- **kernel_size** (*int*) – kernel size of discriminators
- **initial_filters** (*int*) – number of initial filters in the first layer of the discriminators
- **filters_cap** (*int*) – maximum number of filters in the discriminators
- **use_dropout** (*bool*) – whether to use dropout
- **dropout_prob** (*float*) – probability of dropout
- **non_linearity** (*tf.keras.layers.Layer*) – non linearity used in discriminators
- **normalization_layer** (*tf.keras.layers.Layer*) – normalization used by the discriminators
- **use_attention** (*bool*) – whether to use attention
- **n_discriminators** (*int*) – Number of discriminators

build_discriminator (*input_res*)

Build a single discriminator using parameters defined in this object.

Parameters **input_res** – input resolution of the discriminator.

Return type *Encoder*

Returns A Discriminator (PatchDiscriminator).

call (*inputs, training=True, return_features=False*)

Forward pass of the Multi Scale Discriminator.

¹ High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs <https://arxiv.org/abs/1711.11585>

Parameters

- **inputs** (`tf.Tensor`) – input tensor.
- **training** (`bool`) – whether is training or not.
- **return_features** (`bool`) – whether to return features or not.

Return type `Union[List[Tensor], Tuple[List[Tensor], List[Tensor]]]`

Returns

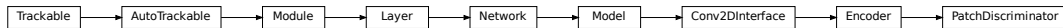
(`tf.Tensor`) –

A List of Tensors containing the value of D_i for each input.

(`tf.Tensor`): A List of features for each discriminator if *return_features*.

PatchDiscriminator

Inheritance Diagram



```

class ashpy.models.convolutional.discriminators.PatchDiscriminator(input_res,
                                                                    min_res,
                                                                    ker-
                                                                    nel_size,
                                                                    ini-
                                                                    tial_filters,
                                                                    fil-
                                                                    ters_cap,
                                                                    use_dropout=True,
                                                                    dropout_prob=0.3,
                                                                    non_linearity=<class
                                                                    'tensor-
                                                                    flow.python.keras.layers.advanced_
                                                                    nor-
                                                                    maliza-
                                                                    tion_layer=<class
                                                                    'ashpy.layers.instance_normalizati
                                                                    use_attention=False)
  
```

Bases: `ashpy.models.convolutional.encoders.Encoder`

Pix2Pix discriminator.

The last layer is an image in which each pixels is the probability of being fake or real.

Examples

```
x = tf.ones((1, 64, 64, 3))

# instantiate the PatchDiscriminator
patchDiscriminator = PatchDiscriminator(input_res=64,
                                       min_res=16,
                                       kernel_size=5,
                                       initial_filters=64,
                                       filters_cap=512,
                                       )

# evaluate passing x
output = patchDiscriminator(x)

# the output shape is the same as the input shape
print(output.shape)
```

```
(1, 12, 12, 1)
```

Methods

<code>__init__(input_res, min_res, kernel_size, ...)</code>	Patch Discriminator used by pix2pix.
<code>call(inputs[, training, return_features])</code>	Forward pass of the PatchDiscriminator.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.

Continued on next page

Table 255 – continued from previous page

<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

```
__init__(input_res, min_res, kernel_size, initial_filters, filters_cap,
         use_dropout=True, dropout_prob=0.3, non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>,
         normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceNormalization'>, use_attention=False)
```

Patch Discriminator used by pix2pix.

When `min_res=1` this is the same as a standard fully convolutional discriminator.

Parameters

- **`input_res`** (*int*) – Input Resolution.
- **`min_res`** (*int*) – Minimum Resolution reached by the discriminator.
- **`kernel_size`** (*int*) – Kernel Size used in Conv Layer.
- **`initial_filters`** (*int*) – number of filters in the first convolutional layer.
- **`filters_cap`** (*int*) – Maximum number of filters.
- **`use_dropout`** (*bool*) – whether to use dropout.
- **`dropout_prob`** (*float*) – probability of dropout.
- **`non_linearity`** (*tf.keras.layers.Layer*) – non linearity used in the model.
- **`normalization_layer`** (*tf.keras.layers.Layer*) – normalization layer used in the model.
- **`use_attention`** (*bool*) – whether to use attention.

```
_add_building_block(filters, use_bn=False)
```

Construct the core of the `tf.keras.Model`.

The layers specified here get added to the `tf.keras.Model` multiple times consuming the hyper-parameters generated in the `_get_layer_spec()`.

Parameters **`filters`** (*int*) – Number of filters to use for this iteration of the Building Block.

```
_add_final_block(output_shape)
```

Prepare the results of `_add_building_block()` for the final output.

Parameters **`output_shape`** (*int*) – Amount of units of the last `tf.keras.layers.Dense`

call (*inputs*, *training=False*, *return_features=False*)
 Forward pass of the PatchDiscriminator.

class `ashpy.models.convolutional.discriminators.MultiScaleDiscriminator` (*input_res*,
min_res,
kernel_size,
initial_filters,
filters_cap,
use_dropout=True,
dropout_prob=0.3,
non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.InstanceNormalization'>,
use_attention=False,
n_discriminators=1)

Bases: `tensorflow.python.keras.engine.training.Model`

Multi-Scale discriminator.

This discriminator architecture is composed by multiple discriminators working at different scales. Each discriminator is a `ashpy.models.convolutional.discriminators.PatchDiscriminator`.

Examples

```
x = tf.ones((1, 256, 256, 3))

# instantiate the PatchDiscriminator
multiScaleDiscriminator = MultiScaleDiscriminator(input_res=256,
                                                  min_res=16,
                                                  kernel_size=5,
                                                  initial_filters=64,
                                                  filters_cap=512,
                                                  n_discriminators=3
                                                  )

# evaluate passing x
outputs = multiScaleDiscriminator(x)

# the output shape is
# the same as the input shape
print(len(outputs))
for output in outputs:
    print(output.shape)
```

```
3
(1, 12, 12, 1)
```

(continues on next page)

(continued from previous page)

```
(1, 12, 12, 1)
(1, 12, 12, 1)
```

```
__init__(input_res, min_res, kernel_size, initial_filters, filters_cap,
         use_dropout=True, dropout_prob=0.3, non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>, normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceNormalization'>, use_attention=False, n_discriminators=1)
```

Multi Scale Discriminator.

Different generator for different scales of the input image.

Used by Pix2PixHD¹.

Parameters

- **input_res** (*int*) – input resolution
- **min_res** (*int*) – minimum resolution reached by the discriminators
- **kernel_size** (*int*) – kernel size of discriminators
- **initial_filters** (*int*) – number of initial filters in the first layer of the discriminators
- **filters_cap** (*int*) – maximum number of filters in the discriminators
- **use_dropout** (*bool*) – whether to use dropout
- **dropout_prob** (*float*) – probability of dropout
- **non_linearity** (*tf.keras.layers.Layer*) – non linearity used in discriminators
- **normalization_layer** (*tf.keras.layers.Layer*) – normalization used by the discriminators
- **use_attention** (*bool*) – whether to use attention
- **n_discriminators** (*int*) – Number of discriminators

build_discriminator (*input_res*)

Build a single discriminator using parameters defined in this object.

Parameters **input_res** – input resolution of the discriminator.

Return type *Encoder*

Returns A Discriminator (PatchDiscriminator).

call (*inputs, training=True, return_features=False*)

Forward pass of the Multi Scale Discriminator.

Parameters

- **inputs** (*tf.Tensor*) – input tensor.
- **training** (*bool*) – whether is training or not.
- **return_features** (*bool*) – whether to return features or not.

Return type *Union[List[Tensor], Tuple[List[Tensor], List[Tensor]]]*

¹ High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs <https://arxiv.org/abs/1711.11585>

Returns

(`[tf.Tensor]`) –

A List of Tensors containing the value of `D_i` for each input.

(`[tf.Tensor]`): A List of features for each discriminator if `return_features`.

```
class ashpy.models.convolutional.discriminators.PatchDiscriminator(input_res,
                                                                    min_res,
                                                                    ker-
                                                                    nel_size,
                                                                    ini-
                                                                    tial_filters,
                                                                    fil-
                                                                    ters_cap,
                                                                    use_dropout=True,
                                                                    dropout_prob=0.3,
                                                                    non_linearity=<class
                                                                    'tensor-
                                                                    flow.python.keras.layers.advanced_
                                                                    nor-
                                                                    maliza-
                                                                    tion_layer=<class
                                                                    'ashpy.layers.instance_normalizati
                                                                    use_attention=False)
```

Bases: `ashpy.models.convolutional.encoders.Encoder`

Pix2Pix discriminator.

The last layer is an image in which each pixels is the probability of being fake or real.

Examples

```
x = tf.ones((1, 64, 64, 3))

# instantiate the PathDiscriminator
patchDiscriminator = PatchDiscriminator(input_res=64,
                                        min_res=16,
                                        kernel_size=5,
                                        initial_filters=64,
                                        filters_cap=512,
                                        )

# evaluate passing x
output = patchDiscriminator(x)

# the output shape is the same as the input shape
print(output.shape)
```

```
(1, 12, 12, 1)
```

```
__init__(input_res, min_res, kernel_size, initial_filters, filters_cap,
         use_dropout=True, dropout_prob=0.3, non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>, normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceNormalization'>, use_attention=False)
Patch Discriminator used by pix2pix.
```

When `min_res=1` this is the same as a standard fully convolutional discriminator.

Parameters

- **input_res** (*int*) – Input Resolution.
- **min_res** (*int*) – Minimum Resolution reached by the discriminator.
- **kernel_size** (*int*) – Kernel Size used in Conv Layer.
- **initial_filters** (*int*) – number of filters in the first convolutional layer.
- **filters_cap** (*int*) – Maximum number of filters.
- **use_dropout** (*bool*) – whether to use dropout.
- **dropout_prob** (*float*) – probability of dropout.
- **non_linearity** (`tf.keras.layers.Layer`) – non linearity used in the model.
- **normalization_layer** (`tf.keras.layers.Layer`) – normalization layer used in the model.
- **use_attention** (*bool*) – whether to use attention.

_add_building_block (*filters*, *use_bn=False*)

Construct the core of the `tf.keras.Model`.

The layers specified here get added to the `tf.keras.Model` multiple times consuming the hyper-parameters generated in the `_get_layer_spec()`.

Parameters **filters** (*int*) – Number of filters to use for this iteration of the Building Block.

_add_final_block (*output_shape*)

Prepare the results of `_add_building_block()` for the final output.

Parameters **output_shape** (*int*) – Amount of units of the last `tf.keras.layers.Dense`

call (*inputs*, *training=False*, *return_features=False*)

Forward pass of the PatchDiscriminator.

decoders

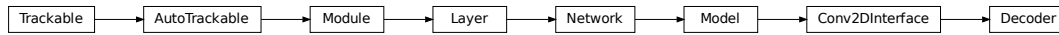
Collection of Decoders (i.e., GANs' Generators) models.

Classes

<i>Decoder</i>	Primitive Model for all decoder (i.e., transpose convolution) based architecture.
<i>FCNNDecoder</i>	Fully Convolutional Decoder.

Decoder

Inheritance Diagram



```

class ashpy.models.convolutional.decoders.Decoder(layer_spec_input_res,
                                                  layer_spec_target_res, kernel_size, initial_filters, filters_cap,
                                                  channels, use_dropout=True, dropout_prob=0.3,
                                                  non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>)
    
```

Bases: `ashpy.models.convolutional.interfaces.Conv2DInterface`

Primitive Model for all decoder (i.e., transpose convolution) based architecture.

Notes

Default to DCGAN Generator architecture.

Examples

- Direct Usage:

```

dummy_generator = Decoder(
    layer_spec_input_res=(8, 8),
    layer_spec_target_res=(64, 64),
    kernel_size=(5, 5),
    initial_filters=1024,
    filters_cap=16,
    channels=3,
)
    
```

- Subclassing

```

class DummyGenerator(Decoder):
    def call(self, input, training=True):
        print("Dummy Generator!")
        return input

dummy_generator = DummyGenerator(
    layer_spec_input_res=(8, 8),
    layer_spec_target_res=(32, 32),
    kernel_size=(5, 5),
    initial_filters=1024,
    filters_cap=16,
    channels=3,
)
dummy_generator(tf.random.normal((1, 100)))
    
```

Dummy Generator!

Methods

<code>__init__(layer_spec_input_res, ...[, ...])</code>	Instantiate the <i>Decoder</i> .
---	----------------------------------

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.


```
__init__(layer_spec_input_res, layer_spec_target_res, kernel_size, initial_filters, filters_cap,
         channels, use_dropout=True, dropout_prob=0.3, non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>)
```

Instantiate the `Decoder`.

Model Assembly: 1. `_add_initial_block()`: Ingest the `tf.keras.Model` inputs and prepare them for `_add_building_block()`.

2. `_add_building_block()`: Core of the model, the layers specified here get added to the `tf.keras.Model` multiple times consuming the hyperparameters generated in the `_get_layer_spec()`.

3. `_add_final_block()`: Final block of our `tf.keras.Model`, take the model after `_add_building_block()` and prepare them for the for the final output.

Parameters

- **layer_spec_input_res** (tuple of (int, int)) – Shape of the `_get_layer_spec()` input tensors.
- **layer_spec_target_res** – (tuple of (int, int)): Shape of tensor desired as output of `_get_layer_spec()`.
- **kernel_size** (tuple of (int, int)) – Kernel used by the convolution layers.
- **initial_filters** (int) – Numbers of filters at the end of the first block.
- **filters_cap** (int) – Cap filters to a set amount, in the case of Decoder is a floor value AKA the minimum amount of filters.
- **channels** (int) – Channels of the output images (1 for Grayscale, 3 for RGB).

Returns `None`

Raises `ValueError` – If `filters_cap > initial_filters`.

```
_add_building_block(filters)
```

Construct the core of the `tf.keras.Model`.

The layers specified here get added to the `tf.keras.Model` multiple times consuming the hyperparameters generated in the `_get_layer_spec()`.

Parameters **filters** (int) – Number of filters to use for this iteration of the Building Block.

```
_add_final_block(channels)
```

Prepare results of `_add_building_block()` for the for the final output.

Parameters **channels** (int) – Channels of the output images (1 for Grayscale, 3 for RGB).

```
_add_initial_block(initial_filters, input_res)
```

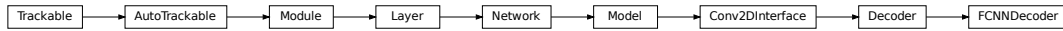
Ingest the `tf.keras.Model` inputs and prepare them for `_add_building_block()`.

Parameters

- **initial_filters** (int) – Numbers of filters to used as a base value.
- **input_res** (tuple of (int, int)) – Shape of the `_get_layer_spec()` input tensors.

FCNNDecoder

Inheritance Diagram



```

class ashpy.models.convolutional.decoders.FCNNDecoder(layer_spec_input_res,
                                                    layer_spec_target_res,
                                                    kernel_size,    initial_filters,
                                                    filters_cap,      channels,
                                                    use_dropout=True,
                                                    dropout_prob=0.3,
                                                    non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>)

```

Bases: *ashpy.models.convolutional.decoders.Decoder*

Fully Convolutional Decoder. Expected input is a feature map.

Examples

• Direct Usage:

```

dummy_generator = FCNNDecoder(
    layer_spec_input_res=(8, 8),
    layer_spec_target_res=(64, 64),
    kernel_size=(5, 5),
    initial_filters=1024,
    filters_cap=16,
    channels=3,
)

print(dummy_generator(tf.zeros((1, 1, 1, 100))).shape)

```

```
(1, 64, 64, 3)
```

Methods

<code>__init__(layer_spec_input_res, ...[, ...])</code>	Build a Fully Convolutional Decoder.
---	--------------------------------------

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	

Continued on next page

Table 260 – continued from previous page

<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

```
__init__(layer_spec_input_res, layer_spec_target_res, kernel_size, initial_filters, filters_cap,
         channels, use_dropout=True, dropout_prob=0.3, non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>)
```

Build a Fully Convolutional Decoder.

```
_add_initial_block(initial_filters, input_res)
```

Ingest the `tf.keras.Model` inputs and prepare them for `_add_building_block()`.

Parameters

- **initial_filters** (*int*) – Numbers of filters to used as a base value.
- **input_res** (tuple of (*int*, *int*)) – Shape of the `_get_layer_spec()` input tensors.

```
class ashpy.models.convolutional.decoders.Decoder(layer_spec_input_res,
                                                  layer_spec_target_res,      ker-
                                                  nel_size, initial_filters, filters_cap,
                                                  channels,      use_dropout=True,
                                                  dropout_prob=0.3,
                                                  non_linearity=<class      'tensor-
flow.python.keras.layers.advanced_activations.LeakyReLU'>)
```

Bases: [ashpy.models.convolutional.interfaces.Conv2DInterface](#)

Primitive Model for all decoder (i.e., transpose convolution) based architecture.

Notes

Default to DCGAN Generator architecture.

Examples

- Direct Usage:

```
dummy_generator = Decoder(
    layer_spec_input_res=(8, 8),
    layer_spec_target_res=(64, 64),
    kernel_size=(5, 5),
    initial_filters=1024,
    filters_cap=16,
    channels=3,
)
```

- Subclassing

```
class DummyGenerator(Decoder):
    def call(self, input, training=True):
        print("Dummy Generator!")
        return input

dummy_generator = DummyGenerator(
    layer_spec_input_res=(8, 8),
    layer_spec_target_res=(32, 32),
    kernel_size=(5, 5),
    initial_filters=1024,
    filters_cap=16,
    channels=3,
)
dummy_generator(tf.random.normal((1, 100)))
```

```
Dummy Generator!
```

```
__init__(layer_spec_input_res, layer_spec_target_res, kernel_size, initial_filters, filters_cap,
          channels, use_dropout=True, dropout_prob=0.3, non_linearity=<class 'tensor-
flow.python.keras.layers.advanced_activations.LeakyReLU'>)
```

Instantiate the [Decoder](#).

Model Assembly: 1. [_add_initial_block\(\)](#): Ingest the `tf.keras.Model` inputs and prepare them for [_add_building_block\(\)](#).

2. `_add_building_block()`: Core of the model, the layers specified here get added to the `tf.keras.Model` multiple times consuming the hyperparameters generated in the `_get_layer_spec()`.
3. `_add_final_block()`: Final block of our `tf.keras.Model`, take the model after `_add_building_block()` and prepare them for the for the final output.

Parameters

- **layer_spec_input_res** (tuple of (int, int)) – Shape of the `_get_layer_spec()` input tensors.
- **layer_spec_target_res** – (tuple of (int, int)): Shape of tensor desired as output of `_get_layer_spec()`.
- **kernel_size** (tuple of (int, int)) – Kernel used by the convolution layers.
- **initial_filters** (int) – Numbers of filters at the end of the first block.
- **filters_cap** (int) – Cap filters to a set amount, in the case of Decoder is a floor value AKA the minimum amount of filters.
- **channels** (int) – Channels of the output images (1 for Grayscale, 3 for RGB).

Returns `None`

Raises `ValueError` – If `filters_cap > initial_filters`.

`_add_building_block(filters)`

Construct the core of the `tf.keras.Model`.

The layers specified here get added to the `tf.keras.Model` multiple times consuming the hyperparameters generated in the `_get_layer_spec()`.

Parameters **filters** (int) – Number of filters to use for this iteration of the Building Block.

`_add_final_block(channels)`

Prepare results of `_add_building_block()` for the for the final output.

Parameters **channels** (int) – Channels of the output images (1 for Grayscale, 3 for RGB).

`_add_initial_block(initial_filters, input_res)`

Ingest the `tf.keras.Model` inputs and prepare them for `_add_building_block()`.

Parameters

- **initial_filters** (int) – Numbers of filters to used as a base value.
- **input_res** (tuple of (int, int)) – Shape of the `_get_layer_spec()` input tensors.

```
class ashpy.models.convolutional.decoders.FCNNDecoder(layer_spec_input_res,
                                                    layer_spec_target_res,
                                                    kernel_size,    initial_filters,
                                                    filters_cap,      channels,
                                                    use_dropout=True,
                                                    dropout_prob=0.3,
                                                    non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>)
```

Bases: `ashpy.models.convolutional.decoders.Decoder`

Fully Convolutional Decoder. Expected input is a feature map.

Examples

- **Direct Usage:**

```
dummy_generator = FCNNDecoder(
    layer_spec_input_res=(8, 8),
    layer_spec_target_res=(64, 64),
    kernel_size=(5, 5),
    initial_filters=1024,
    filters_cap=16,
    channels=3,
)

print(dummy_generator(tf.zeros((1, 1, 1, 100))).shape)
```

```
(1, 64, 64, 3)
```

__init__(*layer_spec_input_res*, *layer_spec_target_res*, *kernel_size*, *initial_filters*, *filters_cap*, *channels*, *use_dropout=True*, *dropout_prob=0.3*, *non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>*)

Build a Fully Convolutional Decoder.

_add_initial_block(*initial_filters*, *input_res*)

Ingest the `tf.keras.Model` inputs and prepare them for `_add_building_block()`.

Parameters

- **initial_filters** (*int*) – Numbers of filters to used as a base value.
- **input_res** (*tuple* of (*int*, *int*)) – Shape of the `_get_layer_spec()` input tensors.

encoders

Collection of Encoders (i.e., GANs' Discriminators) models.

Classes

<i>Encoder</i>	Primitive Model for all encoder (i.e., convolution) based architecture.
<i>FCNNEncoder</i>	Fully Convolutional Encoder.

Encoder

Inheritance Diagram



```
class ashpy.models.convolutional.encoders.Encoder(layer_spec_input_res,
                                                layer_spec_target_res, kernel_size,
                                                initial_filters, filters_cap, out-
                                                put_shape, use_dropout=True,
                                                dropout_prob=0.3,
                                                non_linearity=<class      'tensor-
                                                flow.python.keras.layers.advanced_activations.LeakyReLU'>
```

Bases: `ashpy.models.convolutional.interfaces.Conv2DInterface`

Primitive Model for all encoder (i.e., convolution) based architecture.

Notes

Default to DCGAN Discriminator architecture.

Examples

- Direct Usage:

```
dummy_generator = Encoder(
    layer_spec_input_res=(64, 64),
    layer_spec_target_res=(8, 8),
    kernel_size=5,
    initial_filters=4,
    filters_cap=128,
    output_shape=1,
)
```

- Subclassing

```
class DummyDiscriminator(Encoder):
    def call(self, inputs, training=True):
        print("Dummy Discriminator!")
        # build the model using
        # self._layers and inputs
        return inputs

dummy_discriminator = DummyDiscriminator(
    layer_spec_input_res=(64, 64),
    layer_spec_target_res=(8, 8),
    kernel_size=5,
    initial_filters=16,
    filters_cap=128,
    output_shape=1,
)
dummy_discriminator(tf.zeros((1,28,28,3)))
```

```
Dummy Discriminator!
```

Methods

<code>__init__(layer_spec_input_res, ...[, ...])</code>	Instantiate the Decoder.
---	--------------------------

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

```
__init__(layer_spec_input_res, layer_spec_target_res, kernel_size, initial_filters, filters_cap,
         output_shape, use_dropout=True, dropout_prob=0.3, non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>)
```

Instantiate the Decoder.

Parameters

- **layer_spec_input_res** (tuple of (int, int)) – Shape of the input tensors.
- **layer_spec_target_res** (Union[int, Tuple[int, int]]) – (tuple of (int, int)): Shape of tensor desired as output of `_get_layer_spec()`.
- **kernel_size** (int) – Kernel used by the convolution layers.

- **initial_filters** (*int*) – Numbers of filters to used as a base value.
- **filters_cap** (*int*) – Cap filters to a set amount, in the case of an Encoder is a ceil value AKA the max amount of filters.
- **output_shape** (*int*) – Amount of units of the last `tf.keras.layers.Dense`.

Returns `None`

Raises `ValueError` – If `filters_cap < initial_filters`

`_add_building_block` (*filters*)

Construct the core of the `tf.keras.Model`.

The layers specified here get added to the `tf.keras.Model` multiple times consuming the hyper-parameters generated in the `_get_layer_spec()`.

Parameters **filters** (*int*) – Number of filters to use for this iteration of the Building Block.

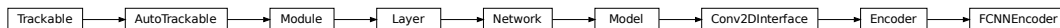
`_add_final_block` (*output_shape*)

Prepare the results of `_add_building_block()` for the final output.

Parameters **output_shape** (*int*) – Amount of units of the last `tf.keras.layers.Dense`

FCNNEncoder

Inheritance Diagram



```

class ashpy.models.convolutional.encoders.FCNNEncoder(layer_spec_input_res,
                                                       layer_spec_target_res,
                                                       kernel_size,    initial_filters,
                                                       filters_cap,      encod-
                                                       ing_dimension)

```

Bases: `ashpy.models.convolutional.encoders.Encoder`

Fully Convolutional Encoder.

Output a `1x1encoding_size` vector. The output neurons are linear.

Examples

- Direct Usage:

```

dummy_generator = FCNNEncoder(
    layer_spec_input_res=(64, 64),
    layer_spec_target_res=(8, 8),
    kernel_size=5,
    initial_filters=4,

```

(continues on next page)

(continued from previous page)

```

        filters_cap=128,
        encoding_dimension=100,
    )
    print(dummy_generator(tf.zeros((1, 64, 64, 3))).shape)

```

```
(1, 1, 1, 100)
```

Methods

<code>__init__(layer_spec_input_res, ...)</code>	Instantiate the FCNNDecoder.
--	------------------------------

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	

Continued on next page

Table 265 – continued from previous page

updates	
variables	Returns the list of all layer variables/weights.
weights	Returns the list of all layer variables/weights.

__init__ (*layer_spec_input_res*, *layer_spec_target_res*, *kernel_size*, *initial_filters*, *filters_cap*, *encoding_dimension*)

Instantiate the FCNNDecoder.

Parameters

- **layer_spec_input_res** (tuple of (int, int)) – Shape of the input tensors.
- **layer_spec_target_res** – (tuple of (int, int)): Shape of tensor desired as output of `_get_layer_spec()`.
- **kernel_size** (int) – Kernel used by the convolution layers.
- **initial_filters** (int) – Numbers of filters to used as a base value.
- **filters_cap** (int) – Cap filters to a set amount, in the case of an Encoder is a ceil value AKA the max amount of filters.
- **encoding_dimension** (int) – encoding dimension.

Returns None

Raises `ValueError` – If `filters_cap < initial_filters`

_add_final_block (*output_shape*)

Prepare the results of `_add_building_block()` for the final output.

Parameters **output_shape** (int) – Amount of units of the last `tf.keras.layers.Dense`

```
class ashpy.models.convolutional.encoders.Encoder (layer_spec_input_res,
                                                    layer_spec_target_res, kernel_size,
                                                    initial_filters, filters_cap, output_shape,
                                                    use_dropout=True, dropout_prob=0.3,
                                                    non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>)
```

Bases: `ashpy.models.convolutional.interfaces.Conv2DInterface`

Primitive Model for all encoder (i.e., convolution) based architecture.

Notes

Default to DCGAN Discriminator architecture.

Examples

- Direct Usage:

```
dummy_generator = Encoder(
    layer_spec_input_res=(64, 64),
    layer_spec_target_res=(8, 8),
    kernel_size=5,
    initial_filters=4,
```

(continues on next page)

(continued from previous page)

```
filters_cap=128,
output_shape=1,
)
```

- Subclassing

```
class DummyDiscriminator(Encoder):
    def call(self, inputs, training=True):
        print("Dummy Discriminator!")
        # build the model using
        # self._layers and inputs
        return inputs

dummy_discriminator = DummyDiscriminator(
    layer_spec_input_res=(64, 64),
    layer_spec_target_res=(8, 8),
    kernel_size=5,
    initial_filters=16,
    filters_cap=128,
    output_shape=1,
)
dummy_discriminator(tf.zeros((1, 28, 28, 3)))
```

```
Dummy Discriminator!
```

`__init__`(*layer_spec_input_res*, *layer_spec_target_res*, *kernel_size*, *initial_filters*, *filters_cap*, *output_shape*, *use_dropout=True*, *dropout_prob=0.3*, *non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>*)
 Instantiate the Decoder.

Parameters

- **layer_spec_input_res** (tuple of (int, int)) – Shape of the input tensors.
- **layer_spec_target_res** (Union[int, Tuple[int, int]]) – (tuple of (int, int)): Shape of tensor desired as output of `_get_layer_spec()`.
- **kernel_size** (int) – Kernel used by the convolution layers.
- **initial_filters** (int) – Numbers of filters to used as a base value.
- **filters_cap** (int) – Cap filters to a set amount, in the case of an Encoder is a ceil value AKA the max amount of filters.
- **output_shape** (int) – Amount of units of the last `tf.keras.layers.Dense`.

Returns None

Raises `ValueError` – If *filters_cap* < *initial_filters*

`_add_building_block`(*filters*)

Construct the core of the `tf.keras.Model`.

The layers specified here get added to the `tf.keras.Model` multiple times consuming the hyper-parameters generated in the `_get_layer_spec()`.

Parameters **filters** (int) – Number of filters to use for this iteration of the Building Block.

`_add_final_block`(*output_shape*)

Prepare the results of `_add_building_block()` for the final output.

Parameters **output_shape** (*int*) – Amount of units of the last `tf.keras.layers.Dense`

```
class ashpy.models.convolutional.encoders.FCNNEncoder(layer_spec_input_res,
                                                       layer_spec_target_res,
                                                       kernel_size,      initial_filters,
                                                       filters_cap,         encod-
                                                       ing_dimension)
```

Bases: `ashpy.models.convolutional.encoders.Encoder`

Fully Convolutional Encoder.

Output a 1x1encoding_size vector. The output neurons are linear.

Examples

- Direct Usage:

```
dummy_generator = FCNNEncoder(
    layer_spec_input_res=(64, 64),
    layer_spec_target_res=(8, 8),
    kernel_size=5,
    initial_filters=4,
    filters_cap=128,
    encoding_dimension=100,
)
print(dummy_generator(tf.zeros((1, 64, 64, 3))).shape)
```

```
(1, 1, 1, 100)
```

__init__ (*layer_spec_input_res*, *layer_spec_target_res*, *kernel_size*, *initial_filters*, *filters_cap*, *encoding_dimension*)

Instantiate the FCNNDecoder.

Parameters

- **layer_spec_input_res** (*tuple* of (*int*, *int*)) – Shape of the input tensors.
- **layer_spec_target_res** – (*tuple* of (*int*, *int*)): Shape of tensor desired as output of `_get_layer_spec()`.
- **kernel_size** (*int*) – Kernel used by the convolution layers.
- **initial_filters** (*int*) – Numbers of filters to used as a base value.
- **filters_cap** (*int*) – Cap filters to a set amount, in the case of an Encoder is a ceil value AKA the max amount of filters.
- **encoding_dimension** (*int*) – encoding dimension.

Returns `None`

Raises `ValueError` – If `filters_cap < initial_filters`

_add_final_block (*output_shape*)

Prepare the results of `_add_building_block()` for the final output.

Parameters **output_shape** (*int*) – Amount of units of the last `tf.keras.layers.Dense`

interfaces

Primitive Convolutional interfaces.

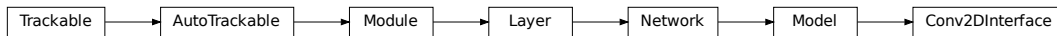
Classes

Conv2DInterface

Primitive Interface to be used by all *ashpy.models*.

Conv2DInterface

Inheritance Diagram



class `ashpy.models.convolutional.interfaces.Conv2DInterface`

Bases: `tensorflow.python.keras.engine.training.Model`

Primitive Interface to be used by all *ashpy.models*.

Methods

<code>__init__()</code>	Primitive Interface to be used by all <i>ashpy.models</i> .
<code>call(inputs[, training, return_features])</code>	Execute the model on input data.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.

Continued on next page

Table 268 – continued from previous page

<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

`__init__()`

Primitive Interface to be used by all *ashpy.models*.

Declares the *self.model_layers* list.

static `_get_layer_spec` (*initial_filters*, *filters_cap*, *input_res*, *target_res*)

Compose the *layer_spec*, the building block of a convolutional model.

The *layer_spec* is an iterator. Every element returned is the number of filters to learn for the current layer. The generated sequence of filters starts from *initial_filters* and halve/double the number of filters depending on the *input_res* and *target_res*. If *input_res* > *target_res* the number of filters increases, else it decreases. The progression is always a power of 2.

Parameters

- **initial_filters** (*int*) – Depth of the first convolutional layer.
- **filters_cap** (*int*) – Maximum number of filters per layer.
- **input_res** (*tuple* of (*int*, *int*)) – Input resolution.
- **target_res** (*tuple* of (*int*, *int*)) – Output resolution.

Yields *int* – Number of filters to use for the conv layer.

Examples

```
# Encoder
class T(Conv2DInterface):
    pass
```

(continues on next page)

(continued from previous page)

```

spec = T._get_layer_spec(
    initial_filters=16,
    filters_cap=128,
    input_res=(512, 256),
    target_res=(32, 16)
)
print([s for s in spec])

spec = T._get_layer_spec(
    initial_filters=16,
    filters_cap=128,
    input_res=(28, 28),
    target_res=(7, 7)
)
print([s for s in spec])

# Decoder
spec = T._get_layer_spec(
    initial_filters=128,
    filters_cap=16,
    input_res=(32, 16),
    target_res=(512, 256)
)
print([s for s in spec])

spec = T._get_layer_spec(
    initial_filters=128,
    filters_cap=16,
    input_res=(7, 7),
    target_res=(28, 28)
)
print([s for s in spec])

```

```

[32, 64, 128, 128]
[32, 64]
[64, 32, 16, 16]
[64, 32]

```

Notes

This is useful since it enables us to dynamically redefine models sharing an underlying architecture but with different resolutions.

call (*inputs*, *training=True*, *return_features=False*)

Execute the model on input data.

Parameters

- **inputs** (*tf.Tensor*) – Input tensor(s).
- **training** (*bool*) – Training flag.
- **return_features** (*bool*) – If True returns the features.

Returns *tf.Tensor* – The model output.

class `ashpy.models.convolutional.interfaces.Conv2DInterface`

Bases: `tensorflow.python.keras.engine.training.Model`

Primitive Interface to be used by all *ashpy.models*.

__init__()

Primitive Interface to be used by all *ashpy.models*.

Declares the *self.model_layers* list.

static **_get_layer_spec**(*initial_filters, filters_cap, input_res, target_res*)

Compose the *layer_spec*, the building block of a convolutional model.

The *layer_spec* is an iterator. Every element returned is the number of filters to learn for the current layer. The generated sequence of filters starts from *initial_filters* and halve/double the number of filters depending on the *input_res* and *target_res*. If *input_res* > *target_res* the number of filters increases, else it decreases. The progression is always a power of 2.

Parameters

- **initial_filters** (*int*) – Depth of the first convolutional layer.
- **filters_cap** (*int*) – Maximum number of filters per layer.
- **input_res** (*tuple* of (*int*, *int*)) – Input resolution.
- **target_res** (*tuple* of (*int*, *int*)) – Output resolution.

Yields *int* – Number of filters to use for the conv layer.

Examples

```
# Encoder
class T(Conv2DInterface):
    pass
spec = T._get_layer_spec(
    initial_filters=16,
    filters_cap=128,
    input_res=(512, 256),
    target_res=(32, 16)
)
print([s for s in spec])

spec = T._get_layer_spec(
    initial_filters=16,
    filters_cap=128,
    input_res=(28, 28),
    target_res=(7, 7)
)
print([s for s in spec])

# Decoder
spec = T._get_layer_spec(
    initial_filters=128,
    filters_cap=16,
    input_res=(32, 16),
    target_res=(512, 256)
)
print([s for s in spec])

spec = T._get_layer_spec(
    initial_filters=128,
    filters_cap=16,
```

(continues on next page)

(continued from previous page)

```

    input_res=(7, 7),
    target_res=(28, 28)
)
print([s for s in spec])

```

```

[32, 64, 128, 128]
[32, 64]
[64, 32, 16, 16]
[64, 32]

```

Notes

This is useful since it enables us to dynamically redefine models sharing an underlying architecture but with different resolutions.

call (*inputs*, *training=True*, *return_features=False*)

Execute the model on input data.

Parameters

- **inputs** (*tf.Tensor*) – Input tensor(s).
- **training** (*bool*) – Training flag.
- **return_features** (*bool*) – If True returns the features.

Returns *tf.Tensor* – The model output.

unet

UNET implementations.

Functions

FUNet

Functional UNET Implementation.

ashpy.models.convolutional.unet.FUNet

```

ashpy.models.convolutional.unet.FUNet(input_res,      min_res,      kernel_size,      ini-
                                     tial_filters,    filters_cap,    channels,      in-
                                     put_channels=3,    use_dropout_encoder=True,
                                     use_dropout_decoder=True, dropout_prob=0.3,
                                     encoder_non_linearity=<class      'tensor-
                                     flow.python.keras.layers.advanced_activations.LeakyReLU'>,
                                     decoder_non_linearity=<class      'tensor-
                                     flow.python.keras.layers.advanced_activations.ReLU'>,
                                     last_activation=<function      tanh>,
                                     use_attention=False)

```

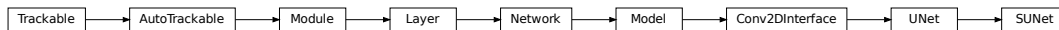
Functional UNET Implementation.

Classes

<i>SUNet</i>	Semantic UNet.
<i>UNet</i>	UNet Architecture.

SUNet

Inheritance Diagram



```

class ashpy.models.convolutional.unet.SUNet(input_res,      min_res,      kernel_size,
                                             initial_filters,  filters_cap,  chan-
                                             nels,            use_dropout_encoder=True,
                                             use_dropout_decoder=True,
                                             dropout_prob=0.3,
                                             coder_non_linearity=<class 'tensor-
flow.python.keras.layers.advanced_activations.LeakyReLU'>,
                                             decoder_non_linearity=<class 'tensor-
flow.python.keras.layers.advanced_activations.ReLU'>,
                                             use_attention=False)

```

Bases: *ashpy.models.convolutional.unet.UNet*

Semantic UNet.

Methods

<code>__init__(input_res, min_res, kernel_size, ...)</code>	Build the Semantic UNet model.
---	--------------------------------

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	

Continued on next page

Table 272 – continued from previous page

<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

```
__init__(input_res, min_res, kernel_size, initial_filters, filters_cap, chan-
        nels, use_dropout_encoder=True, use_dropout_decoder=True,
        dropout_prob=0.3, encoder_non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>, de-
        coder_non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.ReLU'>,
        use_attention=False)
    Build the Semantic UNet model.
```

UNet

Inheritance Diagram



```
class ashpy.models.convolutional.unet.UNet(input_res, min_res, kernel_size,
initial_filters, filters_cap, channels,
use_dropout_encoder=True,
use_dropout_decoder=True,
dropout_prob=0.3, encoder_non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>,
decoder_non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.ReLU'>,
normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceNormalization'>,
last_activation=<function tanh>,
use_attention=False)
```

Bases: *ashpy.models.convolutional.interfaces.Conv2DInterface*

UNet Architecture.

Architecture similar to the one found in “Image-to-Image Translation with Conditional Adversarial Nets”¹.

Originally proposed in “U-Net: Convolutional Networks for Biomedical Image Segmentation”².

Examples

- Direct Usage:

```
x = tf.ones((1, 512, 512, 3))
u_net = UNet(input_res = 512,
min_res=4,
kernel_size=4,
initial_filters=64,
filters_cap=512,
channels=3)
y = u_net(x)
print(y.shape)
print(len(u_net.trainable_variables)>0)
```

```
(1, 512, 512, 3)
True
```

Methods

<code>__init__(input_res, min_res, kernel_size, ...)</code>	Initialize the UNet.
<code>call(inputs[, training])</code>	Forward pass of the UNet model.
<code>get_decoder_block(filters[, use_bn, ...])</code>	Return a block to be used in the decoder part of the UNET.
<code>get_encoder_block(filters[, use_bn, ...])</code>	Return a block to be used in the encoder part of the UNET.

Attributes

¹ Image-to-Image Translation with Conditional Adversarial Nets - <https://arxiv.org/abs/1611.07004>

² U-Net: Convolutional Networks for Biomedical Image Segmentation - <https://arxiv.org/abs/1505.04597>

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

```

__init__(input_res,      min_res,      kernel_size,      initial_filters,      filters_cap,      chan-
        nels,            use_dropout_encoder=True,            use_dropout_decoder=True,
        dropout_prob=0.3,            encoder_non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.LeakyReLU'>,
        decoder_non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.ReLU'>,
        normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceNormalization'>,
        last_activation=<function tanh>, use_attention=False)
Initialize the UNet.
    
```

Parameters

- **input_res** (`int`) – input resolution.
- **min_res** (`int`) – minimum resolution reached after decode.

- **kernel_size** (`int`) – kernel size used in the network.
- **initial_filters** (`int`) – number of filter of the initial convolution.
- **filters_cap** (`int`) – maximum number of filters.
- **channels** (`int`) – number of output channels.
- **use_dropout_encoder** (`bool`) – whether to use dropout in the encoder module.
- **use_dropout_decoder** (`bool`) – whether to use dropout in the decoder module.
- **dropout_prob** (`float`) – probability of dropout.
- **encoder_non_linearity** (`Type[Layer]`) – non linearity of encoder.
- **decoder_non_linearity** (`Type[Layer]`) – non linearity of decoder.
- **last_activation** (`<module 'tensorflow_core.keras.activations' from '/home/docs/checkouts/readthedocs.org/user_builds/ashpy/envs/v0.4.0/lib/python3.7/site-packages/tensorflow_core/python/keras/api/_v2/keras/activations/__init__.py'>`) – last activation function, tanh or softmax (for semantic images).
- **use_attention** (`bool`) – whether to use attention.

call (*inputs*, *training=False*)

Forward pass of the UNet model.

get_decoder_block (*filters*, *use_bn=True*, *use_dropout=False*, *use_attention=False*)

Return a block to be used in the decoder part of the UNET.

Parameters

- **filters** – number of filters
- **use_bn** – whether to use batch normalization
- **use_dropout** – whether to use dropout
- **use_attention** – whether to use attention

Returns A block to be used in the decoder part

get_encoder_block (*filters*, *use_bn=True*, *use_attention=False*)

Return a block to be used in the encoder part of the UNET.

Parameters

- **filters** – number of filters.
- **use_bn** – whether to use batch normalization.
- **use_attention** – whether to use attention.

Returns A block to be used in the encoder part.

```
ashpy.models.convolutional.unet.FUNet (input_res,      min_res,      kernel_size,      ini-
                                     tial_filters,    filters_cap,    channels,    in-
                                     put_channels=3,    use_dropout_encoder=True,
                                     use_dropout_decoder=True,    dropout_prob=0.3,
                                     encoder_non_linearity=<class    'tensor-
                                     flow.python.keras.layers.advanced_activations.LeakyReLU'>,
                                     decoder_non_linearity=<class    'tensor-
                                     flow.python.keras.layers.advanced_activations.ReLU'>,
                                     last_activation=<function    tanh>,
                                     use_attention=False)
```

Functional UNET Implementation.

```
class ashpy.models.convolutional.unet.SUNet(input_res,      min_res,      kernel_size,
                                           initial_filters,  filters_cap,  chan-
                                           nels,            use_dropout_encoder=True,
                                           use_dropout_decoder=True,
                                           dropout_prob=0.3,
                                           encoder_non_linearity=<class      'tensor-
flow.python.keras.layers.advanced_activations.LeakyReLU'>,
                                           decoder_non_linearity=<class      'tensor-
flow.python.keras.layers.advanced_activations.ReLU'>,
                                           use_attention=False)
```

Bases: *ashpy.models.convolutional.unet.UNet*

Semantic UNet.

```
__init__(input_res,      min_res,      kernel_size,      initial_filters,      filters_cap,      chan-
          nels,          use_dropout_encoder=True,          use_dropout_decoder=True,
          dropout_prob=0.3,          encoder_non_linearity=<class      'tensor-
flow.python.keras.layers.advanced_activations.LeakyReLU'>,          de-
          coder_non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.ReLU'>,
          use_attention=False)
```

Build the Semantic UNet model.

```
class ashpy.models.convolutional.unet.UNet(input_res,      min_res,      kernel_size,
                                           initial_filters,  filters_cap,  chan-
                                           nels,            use_dropout_encoder=True,
                                           use_dropout_decoder=True,
                                           dropout_prob=0.3,
                                           encoder_non_linearity=<class      'tensor-
flow.python.keras.layers.advanced_activations.LeakyReLU'>,
                                           decoder_non_linearity=<class      'tensor-
flow.python.keras.layers.advanced_activations.ReLU'>,
                                           normalization_layer=<class
'ashpy.layers.instance_normalization.InstanceNormalization'>,
                                           last_activation=<function      tanh>,
                                           use_attention=False)
```

Bases: *ashpy.models.convolutional.interfaces.Conv2DInterface*

UNet Architecture.

Architecture similar to the one found in “Image-to-Image Translation with Conditional Adversarial Nets”¹.

Originally proposed in “U-Net: Convolutional Networks for Biomedical Image Segmentation”².

Examples

- Direct Usage:

```
x = tf.ones((1, 512, 512, 3))
u_net = UNet(input_res = 512,
             min_res=4,
             kernel_size=4,
             initial_filters=64,
             filters_cap=512,
```

(continues on next page)

¹ Image-to-Image Translation with Conditional Adversarial Nets - <https://arxiv.org/abs/1611.07004>

² U-Net: Convolutional Networks for Biomedical Image Segmentation - <https://arxiv.org/abs/1505.04597>

(continued from previous page)

```

        channels=3)
y = u_net(x)
print(y.shape)
print(len(u_net.trainable_variables)>0)

```

```

(1, 512, 512, 3)
True

```

```

__init__(input_res, min_res, kernel_size, initial_filters, filters_cap, chan-
nels, use_dropout_encoder=True, use_dropout_decoder=True,
dropout_prob=0.3, encoder_non_linearity=<class 'tensor-
flow.python.keras.layers.advanced_activations.LeakyReLU'>, de-
coder_non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.ReLU'>,
normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceNormalization'>,
last_activation=<function tanh>, use_attention=False)
Initialize the UNet.

```

Parameters

- **input_res** (`int`) – input resolution.
- **min_res** (`int`) – minimum resolution reached after decode.
- **kernel_size** (`int`) – kernel size used in the network.
- **initial_filters** (`int`) – number of filter of the initial convolution.
- **filters_cap** (`int`) – maximum number of filters.
- **channels** (`int`) – number of output channels.
- **use_dropout_encoder** (`bool`) – whether to use dropout in the encoder module.
- **use_dropout_decoder** (`bool`) – whether to use dropout in the decoder module.
- **dropout_prob** (`float`) – probability of dropout.
- **encoder_non_linearity** (`Type[Layer]`) – non linearity of encoder.
- **decoder_non_linearity** (`Type[Layer]`) – non linearity of decoder.
- **last_activation** (`<module 'tensorflow_core.keras.activations' from '/home/docs/checkouts/readthedocs.org/user_builds/ashpy/envs/v0.4.0/lib/python3.7/site-packages/tensorflow_core/python/keras/api/_v2/keras/activations/__init__.py'>`) – last activation function, tanh or softmax (for semantic images).
- **use_attention** (`bool`) – whether to use attention.

```

call(inputs, training=False)
Forward pass of the UNet model.

```

```

get_decoder_block(filters, use_bn=True, use_dropout=False, use_attention=False)
Return a block to be used in the decoder part of the UNET.

```

Parameters

- **filters** – number of filters
- **use_bn** – whether to use batch normalization
- **use_dropout** – whether to use dropout

- **use_attention** – whether to use attention

Returns A block to be used in the decoder part

get_encoder_block (*filters, use_bn=True, use_attention=False*)

Return a block to be used in the encoder part of the UNET.

Parameters

- **filters** – number of filters.
- **use_bn** – whether to use batch normalization.
- **use_attention** – whether to use attention.

Returns A block to be used in the encoder part.

pix2pixhd

Pix2Pix HD Implementation.

See: “High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs”¹

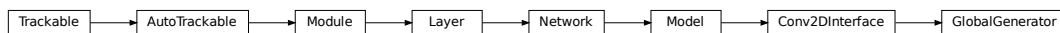
Global Generator + Local Enhancer

Classes

<i>GlobalGenerator</i>	Global Generator from pix2pixHD paper.
<i>LocalEnhancer</i>	Local Enhancer module of the Pix2PixHD architecture.
<i>ResNetBlock</i>	ResNet Blocks.

GlobalGenerator

Inheritance Diagram



¹ High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs: <https://arxiv.org/abs/1711.11585>

```

class ashpy.models.convolutional.pix2pixhd.GlobalGenerator (input_res=512,
                                                            min_res=64,      ini-
                                                            tial_filters=64,
                                                            filters_cap=512,
                                                            channels=3, normal-
                                                            ization_layer=<class
                                                            'ashpy.layers.instance_normalization.InstanceNormalizer'>,
                                                            non_linearity=<class
                                                            'tensorflow.python.keras.layers.advanced_activations.ReLU'>,
                                                            num_resnet_blocks=9,
                                                            kernel_size_resnet=3,
                                                            kernel_size_front_back=7,
                                                            num_internal_resnet_blocks=2)
    
```

Bases: [ashpy.models.convolutional.interfaces.Conv2DInterface](#)

Global Generator from pix2pixHD paper.

- G1^F: Convolutional frontend (downsampling)
- G1^R: ResNet Block
- G1^B: Convolutional backend (upsampling)

Methods

<code>__init__</code> ([input_res, min_res, ...])	Global Generator from Pix2PixHD.
<code>call</code> (inputs[, training])	Call of the Pix2Pix HD model.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	

Continued on next page

Table 277 – continued from previous page

outbound_nodes	Deprecated, do NOT use! Only for compatibility with external Keras.
output	Retrieves the output tensor(s) of a layer.
output_mask	Retrieves the output mask tensor(s) of a layer.
output_shape	Retrieves the output shape(s) of a layer.
run_eagerly	Settable attribute indicating whether the model should run eagerly.
sample_weights	
state_updates	Returns the <i>updates</i> from all layers that are stateful.
stateful	
submodules	Sequence of all sub-modules.
trainable	
trainable_variables	Sequence of variables owned by this module and it's submodules.
trainable_weights	
updates	
variables	Returns the list of all layer variables/weights.
weights	Returns the list of all layer variables/weights.

```
__init__(input_res=512, min_res=64, initial_filters=64, filters_cap=512, channels=3, normal-
        ization_layer=<class      'ashpy.layers.instance_normalization.InstanceNormalization'>,
        non_linearity=<class      'tensorflow.python.keras.layers.advanced_activations.ReLU'>,
        num_resnet_blocks=9,      kernel_size_resnet=3,      kernel_size_front_back=7,
        num_internal_resnet_blocks=2)
Global Generator from Pix2PixHD.
```

Parameters

- **input_res** (*int*) – Input Resolution.
- **min_res** (*int*) – Minimum resolution reached by the downsampling.
- **initial_filters** (*int*) – number of initial filters.
- **filters_cap** (*int*) – maximum number of filters.
- **channels** (*int*) – output channels.
- **normalization_layer** (*tf.keras.layers.Layer*) – normalization layer used by the global generator, can be Instance Norm, Layer Norm, Batch Norm.
- **non_linearity** (*tf.keras.layers.Layer*) – non linearity used in the global generator.
- **num_resnet_blocks** (*int*) – number of resnet blocks.
- **kernel_size_resnet** (*int*) – kernel size used in resnets conv layers.
- **kernel_size_front_back** (*int*) – kernel size used by the convolutional frontend and backend.
- **num_internal_resnet_blocks** (*int*) – number of blocks used by internal resnet.

call (*inputs, training=True*)

Call of the Pix2Pix HD model.

Parameters

- **inputs** – input tensor(s).

- **training** – If True training phase.

Returns Tuple – Generated images.

LocalEnhancer

Inheritance Diagram



```

class ashpy.models.convolutional.pix2pixhd.LocalEnhancer (input_res=512,
                                                           min_res=64,      ini-
                                                           tial_filters=64,    fil-
                                                           ters_cap=512,      chan-
                                                           nels=3,            normaliza-
                                                           tion_layer=<class
                                                           'ashpy.layers.instance_normalization.InstanceNormaliza-
                                                           tion'>,
                                                           non_linearity=<class
                                                           'tensorflow.python.keras.layers.advanced_activations.ReLU'>,
                                                           num_resnet_blocks_global=9,
                                                           num_resnet_blocks_local=3,
                                                           kernel_size_resnet=3,
                                                           kernel_size_front_back=7,
                                                           num_internal_resnet_blocks=2)
    
```

Bases: tensorflow.python.keras.engine.training.Model

Local Enhancer module of the Pix2PixHD architecture.

Example

```

# instantiate the model
model = LocalEnhancer()

# call the model passing inputs
inputs = tf.ones((1, 512, 512, 3))
output = model(inputs)

# the output shape is
# the same as the input shape
print(output.shape)
    
```

```
(1, 512, 512, 3)
```

Methods

<code>__init__([input_res, min_res, ...])</code>	Build the LocalEnhancer module of the Pix2PixHD architecture.
<code>call(inputs[, training])</code>	Call the LocalEnhancer model.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

```
__init__(input_res=512, min_res=64, initial_filters=64, filters_cap=512, channels=3, normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceNormalization'>, non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.ReLU'>, num_resnet_blocks_global=9, num_resnet_blocks_local=3, kernel_size_resnet=3, kernel_size_front_back=7, num_internal_resnet_blocks=2)
```

Build the LocalEnhancer module of the Pix2PixHD architecture.

See High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs² for more details.

Parameters

- **input_res** (*int*) – input resolution.
- **min_res** (*int*) – minimum resolution reached by the global generator.
- **initial_filters** (*int*) – number of initial filters.
- **filters_cap** (*int*) – maximum number of filters.
- **channels** (*int*) – number of channels.
- **normalization_layer** (*tf.keras.layers.Layer*) – layer of normalization
- **Instance Normalization or BatchNormalization or LayerNormalization** (*(e.g.)*) –
- **non_linearity** (*tf.keras.layers.Layer*) – non linearity used in Pix2Pix HD.
- **num_resnet_blocks_global** (*int*) – number of residual blocks used in the global generator.
- **num_resnet_blocks_local** (*int*) – number of residual blocks used in the local generator.
- **kernel_size_resnet** (*int*) – kernel size used in resnets.
- **kernel_size_front_back** (*int*) – kernel size used for the front and back convolution.
- **num_internal_resnet_blocks** (*int*) – number of internal blocks of the resnet.

call (*inputs, training=False*)

Call the LocalEnhancer model.

Parameters

- **inputs** (*tf.Tensor*) – Input Tensors.
- **training** (*bool*) – Whether it is training phase or not.

Returns

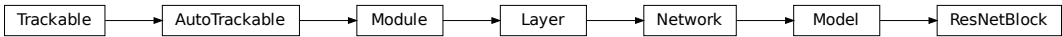
(*tf.Tensor*) –

Image of size (**input_res**, **input_res**, **channels**) as specified in the init call.

ResNetBlock

² High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs <https://arxiv.org/abs/1711.11585>

Inheritance Diagram



```
class ashpy.models.convolutional.pix2pixhd.ResNetBlock(filters, normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceNormalization'>, non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.ReLU'>, kernel_size=3, num_blocks=2)
```

Bases: tensorflow.python.keras.engine.training.Model

ResNet Blocks.

The input filters is the same as the output filters.

Methods

<code>__init__(filters[, normalization_layer, ...])</code>	Build the ResNet block composed by num_blocks.
<code>call(inputs[, training])</code>	Forward pass.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this Layer.
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	

Continued on next page

Table 281 – continued from previous page

<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

`__init__` (*filters*, *normalization_layer*=<class 'ashpy.layers.instance_normalization.InstanceNormalization'>, *non_linearity*=<class 'tensorflow.python.keras.layers.advanced_activations.ReLU'>, *kernel_size*=3, *num_blocks*=2)
Build the ResNet block composed by *num_blocks*.

Each block is composed by

- Conv2D with strides 1 and padding “same”
- Normalization Layer
- Non Linearity

The final result is the output of the ResNet + input.

Parameters

- **filters** (*int*) – initial filters (same as the output filters).
- **normalization_layer** (`tf.keras.layers.Layer`) – layer of normalization used by the residual block.
- **non_linearity** (`tf.keras.layers.Layer`) – non linearity used in the resnet block.
- **kernel_size** (*int*) – kernel size used in the resnet block.
- **num_blocks** (*int*) – number of blocks, each block is composed by conv, normalization and non linearity.

`call` (*inputs*, *training*=*False*)
Forward pass.

Parameters

- **inputs** – input tensor.
- **training** – whether is training or not.

Returns A Tensor of the same shape as the inputs. The input passed through *num_blocks* blocks.

```
class ashpy.models.convolutional.pix2pixhd.GlobalGenerator(input_res=512,
                                                         min_res=64,      ini-
                                                         tial_filters=64,
                                                         filters_cap=512,
                                                         channels=3, normal-
                                                         ization_layer=<class
                                                         'ashpy.layers.instance_normalization.InstanceNormaliza-
                                                         tion'>,
                                                         non_linearity=<class
                                                         'tensorflow.python.keras.layers.advanced_activations.ReLU'>,
                                                         num_resnet_blocks=9,
                                                         kernel_size_resnet=3,
                                                         kernel_size_front_back=7,
                                                         num_internal_resnet_blocks=2)
```

Bases: `ashpy.models.convolutional.interfaces.Conv2DInterface`

Global Generator from pix2pixHD paper.

- G1^F: Convolutional frontend (downsampling)
- G1^R: ResNet Block
- G1^B: Convolutional backend (upsampling)

```
__init__(input_res=512, min_res=64, initial_filters=64, filters_cap=512, channels=3, normal-
          ization_layer=<class 'ashpy.layers.instance_normalization.InstanceNormalization'>,
          non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.ReLU'>,
          num_resnet_blocks=9, kernel_size_resnet=3, kernel_size_front_back=7,
          num_internal_resnet_blocks=2)
```

Global Generator from Pix2PixHD.

Parameters

- **input_res** (*int*) – Input Resolution.
- **min_res** (*int*) – Minimum resolution reached by the downsampling.
- **initial_filters** (*int*) – number of initial filters.
- **filters_cap** (*int*) – maximum number of filters.
- **channels** (*int*) – output channels.
- **normalization_layer** (`tf.keras.layers.Layer`) – normalization layer used by the global generator, can be Instance Norm, Layer Norm, Batch Norm.
- **non_linearity** (`tf.keras.layers.Layer`) – non linearity used in the global generator.
- **num_resnet_blocks** (*int*) – number of resnet blocks.
- **kernel_size_resnet** (*int*) – kernel size used in resnets conv layers.
- **kernel_size_front_back** (*int*) – kernel size used by the convolutional frontend and backend.
- **num_internal_resnet_blocks** (*int*) – number of blocks used by internal resnet.

call (*inputs*, *training=True*)

Call of the Pix2Pix HD model.

Parameters

- **inputs** – input tensor(s).
- **training** – If True training phase.

Returns Tuple – Generated images.

```
class ashpy.models.convolutional.pix2pixhd.LocalEnhancer (input_res=512,
                                                         min_res=64,          ini-
                                                         tial_filters=64,       fil-
                                                         ters_cap=512,         chan-
                                                         nels=3,              normaliza-
                                                         tion_layer=<class
                                                         'ashpy.layers.instance_normalization.InstanceNormaliza-
                                                         tion'>,
                                                         non_linearity=<class
                                                         'tensorflow.python.keras.layers.advanced_activations.ReLU'>,
                                                         num_resnet_blocks_global=9,
                                                         num_resnet_blocks_local=3,
                                                         kernel_size_resnet=3,
                                                         kernel_size_front_back=7,
                                                         num_internal_resnet_blocks=2)
```

Bases: tensorflow.python.keras.engine.training.Model

Local Enhancer module of the Pix2PixHD architecture.

Example

```
# instantiate the model
model = LocalEnhancer()

# call the model passing inputs
inputs = tf.ones((1, 512, 512, 3))
output = model(inputs)

# the output shape is
# the same as the input shape
print(output.shape)
```

```
(1, 512, 512, 3)
```

```
__init__ (input_res=512, min_res=64, initial_filters=64, filters_cap=512, channels=3, normal-
          ization_layer=<class 'ashpy.layers.instance_normalization.InstanceNormalization'>,
          non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.ReLU'>,
          num_resnet_blocks_global=9, num_resnet_blocks_local=3, kernel_size_resnet=3, ker-
          nel_size_front_back=7, num_internal_resnet_blocks=2)
```

Build the LocalEnhancer module of the Pix2PixHD architecture.

See High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs² for more details.

Parameters

- **input_res** (*int*) – input resolution.
- **min_res** (*int*) – minimum resolution reached by the global generator.

² High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs <https://arxiv.org/abs/1711.11585>

- **initial_filters** (*int*) – number of initial filters.
- **filters_cap** (*int*) – maximum number of filters.
- **channels** (*int*) – number of channels.
- **normalization_layer** (`tf.keras.layers.Layer`) – layer of normalization
- **Instance Normalization or BatchNormalization or LayerNormalization** ((*e.g.*) –
- **non_linearity** (`tf.keras.layers.Layer`) – non linearity used in Pix2Pix HD.
- **num_resnet_blocks_global** (*int*) – number of residual blocks used in the global generator.
- **num_resnet_blocks_local** (*int*) – number of residual blocks used in the local generator.
- **kernel_size_resnet** (*int*) – kernel size used in resnets.
- **kernel_size_front_back** (*int*) – kernel size used for the front and back convolution.
- **num_internal_resnet_blocks** (*int*) – number of internal blocks of the resnet.

call (*inputs*, *training=False*)
Call the LocalEnhancer model.

Parameters

- **inputs** (`tf.Tensor`) – Input Tensors.
- **training** (*bool*) – Whether it is training phase or not.

Returns

(`tf.Tensor`) –

Image of size (**input_res**, **input_res**, **channels**) as specified in the init call.

```
class ashpy.models.convolutional.pix2pixhd.ResNetBlock (filters, normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceNormalization'>,  
                                                    non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.ReLU'>,  
                                                    kernel_size=3,  
                                                    num_blocks=2)
```

Bases: `tensorflow.python.keras.engine.training.Model`

ResNet Blocks.

The input filters is the same as the output filters.

__init__ (*filters*, *normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceNormalization'>*,
 non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.ReLU'>, *kernel_size=3*, *num_blocks=2*)
Build the ResNet block composed by num_blocks.

Each block is composed by

- Conv2D with strides 1 and padding “same”
- Normalization Layer
- Non Linearity

The final result is the output of the ResNet + input.

Parameters

- **filters** (*int*) – initial filters (same as the output filters).
- **normalization_layer** (`tf.keras.layers.Layer`) – layer of normalization used by the residual block.
- **non_linearity** (`tf.keras.layers.Layer`) – non linearity used in the resnet block.
- **kernel_size** (*int*) – kernel size used in the resnet block.
- **num_blocks** (*int*) – number of blocks, each block is composed by conv, normalization and non linearity.

call (*inputs*, *training=False*)

Forward pass.

Parameters

- **inputs** – input tensor.
- **training** – whether is training or not.

Returns A Tensor of the same shape as the inputs. The input passed through num_blocks blocks.

4.8.2 fc

Collection of Fully Connected Models constructors.

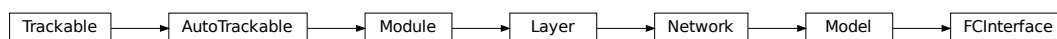
Interfaces

interfaces.FCInterface

Primitive Interface to be used by all *ashpy.models*.

FCInterface

Inheritance Diagram



```
class ashpy.models.fc.interfaces.FCInterface
```

Bases: `tensorflow.python.keras.engine.training.Model`

Primitive Interface to be used by all *ashpy.models*.

Methods

<code>__init__()</code>	Primitive Interface to be used by all <i>ashpy.models</i> .
<code>call(inputs[, training])</code>	Execute the model on input data.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

`__init__()`
 Primitive Interface to be used by all *ashpy.models*.

Declares the `self._layers` list.

Returns `None`

call (*inputs*, *training=True*)
Execute the model on input data.

Parameters

- **inputs** (`tf.Tensor`) – Input tensors.
- **training** (`bool`) – Training flag.

Returns :py_class:`tf.Tensor`

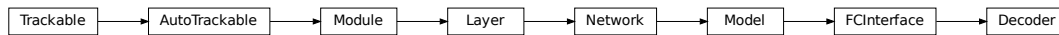
Decoders

`decoders.Decoder`

Primitive Model for all fully connected decoder based architecture.

Decoder

Inheritance Diagram



class `ashpy.models.fc.decoders.Decoder` (*hidden_units*, *output_shape*)

Bases: `ashpy.models.fc.interfaces.FCInterface`

Primitive Model for all fully connected decoder based architecture.

Examples

```
decoder = Decoder(
    hidden_units=[64,128,256],
    output_shape=55)
print(decoder(tf.zeros((1,10))).shape)
```

```
(1, 55)
```

Methods

`__init__`(*hidden_units*, *output_shape*)

Instantiate the `Decoder`.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

`__init__` (*hidden_units*, *output_shape*)
 Instantiate the *Decoder*.

Parameters

- **hidden_units** (*tuple* of *int*) – Number of units per hidden layer.
- **output_shape** (*int*) – Amount of units of the last `tf.keras.layers.Dense`.

Returns `None`

Encoders

encoders.Encoder

Primitive Model for all fully connected encoder based architecture.

Encoder

Inheritance Diagram



class `ashpy.models.fc.encoders.Encoder` (*hidden_units*, *output_shape*)

Bases: *ashpy.models.fc.interfaces.FCInterface*

Primitive Model for all fully connected encoder based architecture.

Examples

```

encoder = Encoder(
    hidden_units=[256, 128, 64],
    output_shape=10)
print(encoder(tf.zeros((1, 55))).shape)
  
```

```
(1, 10)
```

Methods

<code>__init__</code> (<i>hidden_units</i> , <i>output_shape</i>)	Instantiate the Decoder.
---	--------------------------

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.

Continued on next page

Table 290 – continued from previous page

layers	
losses	Losses which are associated with this <i>Layer</i> .
metrics	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
metrics_names	Returns the model's display labels for all outputs.
name	Returns the name of this module as passed or determined in the ctor.
name_scope	Returns a <i>tf.name_scope</i> instance for this class.
non_trainable_variables	
non_trainable_weights	
outbound_nodes	Deprecated, do NOT use! Only for compatibility with external Keras.
output	Retrieves the output tensor(s) of a layer.
output_mask	Retrieves the output mask tensor(s) of a layer.
output_shape	Retrieves the output shape(s) of a layer.
run_eagerly	Settable attribute indicating whether the model should run eagerly.
sample_weights	
state_updates	Returns the <i>updates</i> from all layers that are stateful.
stateful	
submodules	Sequence of all sub-modules.
trainable	
trainable_variables	Sequence of variables owned by this module and it's submodules.
trainable_weights	
updates	
variables	Returns the list of all layer variables/weights.
weights	Returns the list of all layer variables/weights.

__init__ (*hidden_units*, *output_shape*)

Instantiate the Decoder.

Parameters

- **hidden_units** (*tuple* of *int*) – Number of units per hidden layer.
- **output_shape** (*int*) – Amount of units of the last *tf.keras.layers.Dense*.

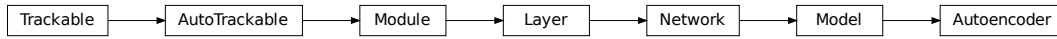
Returns *None*

Autoencoders

<i>autoencoders.Autoencoder</i>	Primitive Model for all fully connected autoencoders.
---------------------------------	---

Autoencoder

Inheritance Diagram



class ashpy.models.fc.autoencoders.**Autoencoder** (*hidden_units, encoding_dimension, output_shape*)

Bases: tensorflow.python.keras.engine.training.Model

Primitive Model for all fully connected autoencoders.

Examples

• Direct Usage:

```

autoencoder = Autoencoder(
    hidden_units=[256, 128, 64],
    encoding_dimension=100,
    output_shape=55
)

encoding, reconstruction = autoencoder(tf.zeros((1, 55)))
print(encoding.shape)
print(reconstruction.shape)
    
```

Methods

<code>__init__(hidden_units, encoding_dimension, ...)</code>	Instantiate the Decoder.
<code>call(inputs[, training])</code>	Execute the model on input data.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.

Continued on next page

Table 293 – continued from previous page

<code>input_spec</code>	Gets the network’s input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model’s metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model’s display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it’s submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

`__init__` (*hidden_units*, *encoding_dimension*, *output_shape*)

Instantiate the Decoder.

Parameters

- **`hidden_units`** (*tuple* of *int*) – Number of units per hidden layer.
- **`encoding_dimension`** (*int*) – encoding dimension.
- **`output_shape`** (*int*) – output shape, usual equal to the input shape.

Returns *None*

`call` (*inputs*, *training=True*)

Execute the model on input data.

Parameters

- **`inputs`** (*tf.Tensor*) – Input tensors.
- **`training`** (*bool*) – Training flag.

Returns (*encoding*, *reconstruction*) – Pair of tensors.

Modules

<i>autoencoders</i>	Collection of Fully Connected Autoencoders.
<i>decoders</i>	Collection of Decoders (i.e., GANs' Discriminators) models.
<i>encoders</i>	Collection of Encoders (i.e., GANs' Discriminators) models.
<i>interfaces</i>	Primitive Fully Connected interfaces.

autoencoders

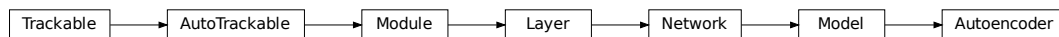
Collection of Fully Connected Autoencoders.

Classes

<i>Autoencoder</i>	Primitive Model for all fully connected autoencoders.
--------------------	---

Autoencoder

Inheritance Diagram



class `ashpy.models.fc.autoencoders.Autoencoder` (*hidden_units*, *encoding_dimension*, *output_shape*)
 Bases: `tensorflow.python.keras.engine.training.Model`
 Primitive Model for all fully connected autoencoders.

Examples

• Direct Usage:

```

autoencoder = Autoencoder(
    hidden_units=[256, 128, 64],
    encoding_dimension=100,
    output_shape=55
)

encoding, reconstruction = autoencoder(tf.zeros((1, 55)))
print(encoding.shape)
print(reconstruction.shape)
    
```



Methods

<code>__init__(hidden_units, encoding_dimension, ...)</code>	Instantiate the Decoder.
<code>call(inputs[, training])</code>	Execute the model on input data.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

`__init__` (*hidden_units*, *encoding_dimension*, *output_shape*)

Instantiate the `Decoder`.

Parameters

- **hidden_units** (*tuple* of *int*) – Number of units per hidden layer.
- **encoding_dimension** (*int*) – encoding dimension.
- **output_shape** (*int*) – output shape, usual equal to the input shape.

Returns `None`

call (*inputs*, *training=True*)

Execute the model on input data.

Parameters

- **inputs** (*tf.Tensor*) – Input tensors.
- **training** (*bool*) – Training flag.

Returns (*encoding*, *reconstruction*) – Pair of tensors.

class `ashpy.models.fc.autoencoders.Autoencoder` (*hidden_units*, *encoding_dimension*, *output_shape*)

Bases: `tensorflow.python.keras.engine.training.Model`

Primitive Model for all fully connected autoencoders.

Examples

• Direct Usage:

```
autoencoder = Autoencoder(
    hidden_units=[256, 128, 64],
    encoding_dimension=100,
    output_shape=55
)

encoding, reconstruction = autoencoder(tf.zeros((1, 55)))
print(encoding.shape)
print(reconstruction.shape)
```

`__init__` (*hidden_units*, *encoding_dimension*, *output_shape*)

Instantiate the `Decoder`.

Parameters

- **hidden_units** (*tuple* of *int*) – Number of units per hidden layer.
- **encoding_dimension** (*int*) – encoding dimension.
- **output_shape** (*int*) – output shape, usual equal to the input shape.

Returns `None`

call (*inputs*, *training=True*)

Execute the model on input data.

Parameters

- **inputs** (`tf.Tensor`) – Input tensors.
 - **training** (`bool`) – Training flag.
- Returns** (*encoding, reconstruction*) – Pair of tensors.

decoders

Collection of Decoders (i.e., GANs' Discriminators) models.

Classes

<i>Decoder</i>	Primitive Model for all fully connected decoder based architecture.
----------------	---

Decoder

Inheritance Diagram



class `ashpy.models.fc.decoders.Decoder` (*hidden_units, output_shape*)

Bases: `ashpy.models.fc.interfaces.FCInterface`

Primitive Model for all fully connected decoder based architecture.

Examples

```
decoder = Decoder(
    hidden_units=[64,128,256],
    output_shape=55)
print(decoder(tf.zeros((1,10))).shape)
```

```
(1, 55)
```

Methods

<code>__init__</code> (<i>hidden_units, output_shape</i>)	Instantiate the <i>Decoder</i> .
---	----------------------------------

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

`__init__` (*hidden_units*, *output_shape*)

Instantiate the *Decoder*.

Parameters

- **hidden_units** (*tuple* of *int*) – Number of units per hidden layer.
- **output_shape** (*int*) – Amount of units of the last `tf.keras.layers.Dense`.

Returns `None`

class `ashpy.models.fc.decoders.Decoder` (*hidden_units*, *output_shape*)

Bases: `ashpy.models.fc.interfaces.FCInterface`

Primitive Model for all fully connected decoder based architecture.

Examples

```
decoder = Decoder(
    hidden_units=[64,128,256],
    output_shape=55)
print(decoder(tf.zeros((1,10))).shape)
```

```
(1, 55)
```

__init__ (*hidden_units*, *output_shape*)
 Instantiate the *Decoder*.

Parameters

- **hidden_units** (*tuple* of *int*) – Number of units per hidden layer.
- **output_shape** (*int*) – Amount of units of the last `tf.keras.layers.Dense`.

Returns *None*

encoders

Collection of Encoders (i.e., GANs' Discriminators) models.

Classes

<i>Encoder</i>	Primitive Model for all fully connected encoder based architecture.
----------------	---

Encoder

Inheritance Diagram



class `ashpy.models.fc.encoders.Encoder` (*hidden_units*, *output_shape*)

Bases: `ashpy.models.fc.interfaces.FCInterface`

Primitive Model for all fully connected encoder based architecture.

Examples

```
encoder = Encoder(
    hidden_units=[256,128,64],
    output_shape=10)
print(encoder(tf.zeros((1,55))).shape)
```

```
(1, 10)
```

Methods

<code>__init__(hidden_units, output_shape)</code>	Instantiate the Decoder.
---	--------------------------

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.
<code>weights</code>	Returns the list of all layer variables/weights.

```
__init__(hidden_units, output_shape)
    Instantiate the Decoder.
```

Parameters

- **hidden_units** (*tuple of int*) – Number of units per hidden layer.
- **output_shape** (*int*) – Amount of units of the last `tf.keras.layers.Dense`.

Returns `None`

class `ashpy.models.fc.encoders.Encoder` (*hidden_units, output_shape*)

Bases: `ashpy.models.fc.interfaces.FCInterface`

Primitive Model for all fully connected encoder based architecture.

Examples

```
encoder = Encoder(
    hidden_units=[256, 128, 64],
    output_shape=10)
print(encoder(tf.zeros((1, 55))).shape)
```

```
(1, 10)
```

__init__ (*hidden_units, output_shape*)

Instantiate the Decoder.

Parameters

- **hidden_units** (*tuple of int*) – Number of units per hidden layer.
- **output_shape** (*int*) – Amount of units of the last `tf.keras.layers.Dense`.

Returns `None`

interfaces

Primitive Fully Connected interfaces.

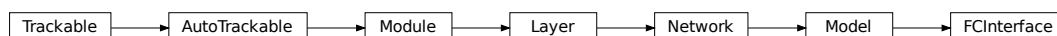
Classes

FCInterface

Primitive Interface to be used by all *ashpy.models*.

FCInterface

Inheritance Diagram



class `ashpy.models.fc.interfaces.FCInterface`

Bases: `tensorflow.python.keras.engine.training.Model`

Primitive Interface to be used by all *ashpy.models*.

Methods

<code>__init__()</code>	Primitive Interface to be used by all <i>ashpy.models</i> .
<code>call(inputs[, training])</code>	Execute the model on input data.

Attributes

<code>activity_regularizer</code>	Optional regularizer function for the output of this layer.
<code>dtype</code>	
<code>dynamic</code>	
<code>inbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>input</code>	Retrieves the input tensor(s) of a layer.
<code>input_mask</code>	Retrieves the input mask tensor(s) of a layer.
<code>input_shape</code>	Retrieves the input shape(s) of a layer.
<code>input_spec</code>	Gets the network's input specs.
<code>layers</code>	
<code>losses</code>	Losses which are associated with this <i>Layer</i> .
<code>metrics</code>	Returns the model's metrics added using <i>compile</i> , <i>add_metric</i> APIs.
<code>metrics_names</code>	Returns the model's display labels for all outputs.
<code>name</code>	Returns the name of this module as passed or determined in the ctor.
<code>name_scope</code>	Returns a <i>tf.name_scope</i> instance for this class.
<code>non_trainable_variables</code>	
<code>non_trainable_weights</code>	
<code>outbound_nodes</code>	Deprecated, do NOT use! Only for compatibility with external Keras.
<code>output</code>	Retrieves the output tensor(s) of a layer.
<code>output_mask</code>	Retrieves the output mask tensor(s) of a layer.
<code>output_shape</code>	Retrieves the output shape(s) of a layer.
<code>run_eagerly</code>	Settable attribute indicating whether the model should run eagerly.
<code>sample_weights</code>	
<code>state_updates</code>	Returns the <i>updates</i> from all layers that are stateful.
<code>stateful</code>	
<code>submodules</code>	Sequence of all sub-modules.
<code>trainable</code>	
<code>trainable_variables</code>	Sequence of variables owned by this module and it's submodules.
<code>trainable_weights</code>	
<code>updates</code>	
<code>variables</code>	Returns the list of all layer variables/weights.

Continued on next page

Table 306 – continued from previous page

<i>weights</i>	Returns the list of all layer variables/weights.
<code>__init__()</code> Primitive Interface to be used by all <i>ashpy.models</i>. Declares the self._layers list. Returns <code>None</code> <code>call(inputs, training=True)</code> Execute the model on input data. Parameters • inputs (<code>tf.Tensor</code>) – Input tensors. • training (<code>bool</code>) – Training flag. Returns <code>:py_class:'tf.Tensor'</code> class <code>ashpy.models.fc.interfaces.FCInterface</code> Bases: <code>tensorflow.python.keras.engine.training.Model</code> Primitive Interface to be used by all <i>ashpy.models</i>. <code>__init__()</code> Primitive Interface to be used by all <i>ashpy.models</i>. Declares the self._layers list. Returns <code>None</code> <code>call(inputs, training=True)</code> Execute the model on input data. Parameters • inputs (<code>tf.Tensor</code>) – Input tensors. • training (<code>bool</code>) – Training flag. Returns <code>:py_class:'tf.Tensor'</code>	
<i>gans</i>	GANs Models.

4.8.3 gans

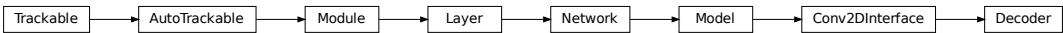
GANs Models.

Generators

<i>ConvGenerator</i>	alias of <i>ashpy.models.convolutional.decoders.Decoder</i>
<i>DenseGenerator</i>	alias of <i>ashpy.models.fc.decoders.Decoder</i>

ConvGenerator

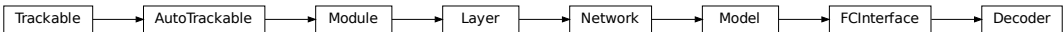
Inheritance Diagram



`ashpy.models.gans.ConvGenerator`
alias of `ashpy.models.convolutional.decoders.Decoder`

DenseGenerator

Inheritance Diagram



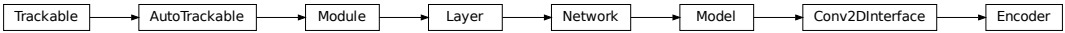
`ashpy.models.gans.DenseGenerator`
alias of `ashpy.models.fc.decoders.Decoder`

Discriminators

<code>ConvDiscriminator</code>	alias of <code>ashpy.models.convolutional.encoders.Encoder</code>
<code>DenseDiscriminator</code>	alias of <code>ashpy.models.fc.encoders.Encoder</code>

ConvDiscriminator

Inheritance Diagram



`ashpy.models.gans.ConvDiscriminator`
alias of `ashpy.models.convolutional.encoders.Encoder`

DenseDiscriminator

Inheritance Diagram



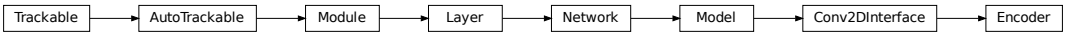
`ashpy.models.gans.DenseDiscriminator`
alias of `ashpy.models.fc.encoders.Encoder`

Encoders

<code>ConvEncoder</code>	alias of <code>ashpy.models.convolutional.encoders.Encoder</code>
<code>DenseEncoder</code>	alias of <code>ashpy.models.fc.encoders.Encoder</code>

ConvEncoder

Inheritance Diagram



`ashpy.models.gans.ConvEncoder`
alias of `ashpy.models.convolutional.encoders.Encoder`

DenseEncoder

Inheritance Diagram



`ashpy.models.gans.DenseEncoder`
alias of `ashpy.models.fc.encoders.Encoder`

4.9 ashpy.modes

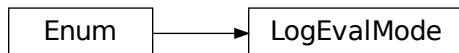
Various modalities used to configure certain ash behaviours.

Classes

<code>LogEvalMode</code>	Mode to use when logging end evaluating a model.
--------------------------	--

4.9.1 LogEvalMode

Inheritance Diagram



class `ashpy.modes.LogEvalMode`

Bases: `enum.Enum`

Mode to use when logging end evaluating a model.

Models often have the same behaviour (or very similar) when evaluated in trainable and non trainable setting.

There are some other models, see `pix2pix`, that instead require to be tested and trained in TRAIN mode (Model with `trainable=True`).

Attributes

TEST
TRAIN

class `ashpy.modes.LogEvalMode`

Bases: `enum.Enum`

Mode to use when logging end evaluating a model.

Models often have the same behaviour (or very similar) when evaluated in trainable and non trainable setting.

There are some other models, see pix2pix, that instead require to be tested and trained in TRAIN mode (Model with trainable=True).

4.10 ashpy.restorers

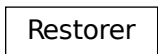
Restorers allow for easy restoration of tracked objects from `tf.train.Checkpoint`.

Classes

<i>Restorer</i>	<i>Restorer</i> provide a way to restore objects from <code>tf.train.Checkpoint</code> .
<i>AdversarialRestorer</i>	Convenience <i>Restorer</i> for ease of use with the AdversairalTrainer.
<i>AdversarialEncoderRestorer</i>	Convenience <i>Restorer</i> for ease of use with the EncoderTrainer.
<i>ClassifierRestorer</i>	Convenience <i>Restorer</i> for ease of use with the ClassifierTrainer.

4.10.1 Restorer

Inheritance Diagram



class `ashpy.restorers.Restorer` (`logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v`
`ckpts_dir='ckpts', expect_partial=True)`

Bases: `object`

Restorer provide a way to restore objects from `tf.train.Checkpoint`.

Can be standalone.

Methods

<code>__init__([logdir, ckpts_dir, expect_partial])</code>	Initialize the Restorer.
<code>get_global_step()</code>	Return the restored global_step.
<code>get_steps_per_epoch()</code>	Return the restored global_step.
<code>restore_callback(callback, callback_ckpt_id)</code>	Return the restored callbacks.
<code>restore_object(placeholder, object_ckpt_id)</code>	Restore a placeholder from a checkpoint using the specified id.

Attributes

<code>checkpoint_map</code>	Get the map of the ids in the checkpoint.
-----------------------------	---

`__init__ (logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log'), ckpts_dir='ckpts', expect_partial=True)`
Initialize the Restorer.

Parameters

- **logdir** (*str*) – Path to the directory with the logs.
- **ckpts_dir** (*str*) – Name of the directory with the checkpoints to restore.
- **expect_partial** (*bool*) – Whether to expect partial restoring or not. Default to true.
For more information see the docs for `tf.train.Checkpoint.restore()`.

Return type `None`

static `_check_model_construction (restored_model)`

Optimistically check that the `model.weights` property returns a non empty-list.

The underlying assumption is that Models created via the sub-classing API, when restored without being properly constructed AKA called on some input, will have empty lists as `layers.weights`.

TODO: add docs for the exception. TODO: add test case for the Sequential without input shape

Return type `bool`

`_restore_checkpoint (checkpoint, partial=True)`

Restore or initialize the persistence layer (checkpoint).

checkpoint_map

Get the map of the ids in the checkpoint.

Map is a Dict where keys are the *ids* in the checkpoint and the values are the string representation of the types.

Return type `Optional[Dict[str, str]]`

Returns Dict if the map is found, else None.

`get_global_step()`

Return the restored global_step.

Return type `Variable`

`get_steps_per_epoch()`

Return the restored global_step.

Return type `Variable`

restore_callback (*callback, callback_ckpt_id*)

Return the restored callbacks.

Return type `List[Callback]`

restore_object (*placeholder, object_ckpt_id*)

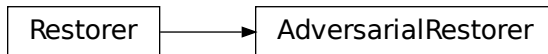
Restore a placeholder from a checkpoint using the specified id.

Warning: When restoring a `tf.keras.Model` object from checkpoint assure that the model has been correctly built and instantiated by firstly calling it on some sample inputs. In the case of a model built with either the Sequential or Functional API an exception will be raised; for a model built with the Chainer API it will fail silently, restoration will be “successful” but no values will actually be restored since there are no valid placeholder as the model has not be built yet.

TODO: Args TODO: Example

4.10.2 AdversarialRestorer

Inheritance Diagram



class `ashpy.restorers.AdversarialRestorer` (*logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/a*
ckpts_dir='ckpts', expect_partial=True))

Bases: `ashpy.restorers.restorer.Restorer`

Convenience `Restorer` for ease of use with the `AdversairalTrainer`.

Methods

<code>restore_discriminator(model)</code>	Restore the Discriminator model.
<code>restore_discriminator_optimizer(optimizer)</code>	Restore the Optimizer used to train the Discriminator model.
<code>restore_generator(model)</code>	Restore the Generator model.
<code>restore_generator_optimizer(optimizer)</code>	Restore the Optimizer used to train the Generator model.

Attributes

<code>checkpoint_map</code>	Get the map of the ids in the checkpoint.
-----------------------------	---

restore_discriminator (*model*)

Restore the Discriminator model.

Parameters *model* (`tf.keras.Model`) – The placeholder model in which values from the checkpoint will be restored.

Return type `Model`

Returns Restored model.

Warning: When restoring a `tf.keras.Model` object from checkpoint assure that the model has been correctly built and instantiated by firstly calling it on some sample inputs. In the case of a model built with either the Sequential or Functional API an exception will be raised; for a model built with the Chainer API it will fail silently, restoration will be “successful” but no values will actually be restored since there are no valid placeholder as the model has not be built yet.

restore_discriminator_optimizer (*optimizer*)

Restore the Optimizer used to train the Discriminator model.

Parameters *model* (`tf.keras.optimizers.Optimizer`) – The placeholder Optimizer in which values from the checkpoint will be restored.

Return type `OptimizerV2`

Returns Restored optimizer.

restore_generator (*model*)

Restore the Generator model.

Parameters *model* (`tf.keras.Model`) – The placeholder model in which values from the checkpoint will be restored.

Return type `Model`

Returns Restored model.

Warning: When restoring a `tf.keras.Model` object from checkpoint assure that the model has been correctly built and instantiated by firstly calling it on some sample inputs. In the case of a model built with either the Sequential or Functional API an exception will be raised; for a model built with the Chainer API it will fail silently, restoration will be “successful” but no values will actually be restored since there are no valid placeholder as the model has not be built yet.

restore_generator_optimizer (*optimizer*)

Restore the Optimizer used to train the Generator model.

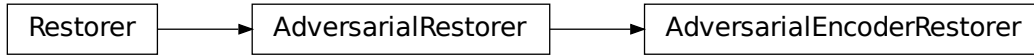
Parameters *model* (`tf.keras.optimizers.Optimizer`) – The placeholder Optimizer in which values from the checkpoint will be restored.

Return type `OptimizerV2`

Returns Restored optimizer.

4.10.3 AdversarialEncoderRestorer

Inheritance Diagram



```

class ashpy.restorers.AdversarialEncoderRestorer (logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_
                                                    ckpts_dir='ckpts',                ex-
                                                    pect_partial=True)

Bases: ashpy.restorers.gan.AdversarialRestorer

Convenience Restorer for ease of use with the EncoderTrainer.
  
```

Methods

<code>restore_encoder(model)</code>	Restore the Encoder model.
<code>restore_encoder_optimizer(optimizer)</code>	Restore the Optimizer used to train the Encoder model.

Attributes

<code>checkpoint_map</code>	Get the map of the ids in the checkpoint.
-----------------------------	---

restore_encoder (*model*)
Restore the Encoder model.

Parameters **model** (`tf.keras.Model`) – The placeholder model in which values from the checkpoint will be restored.

Return type `Model`

Returns Restored model.

Warning: When restoring a `tf.keras.Model` object from checkpoint assure that the model has been correctly built and instantiated by firstly calling it on some sample inputs. In the case of a model built with either the Sequential or Functional API an exception will be raised; for a model built with the Chainer API it will fail silently, restoration will be “successful” but no values will actually be restored since there are no valid placeholder as the model has not be built yet.

restore_encoder_optimizer (*optimizer*)
Restore the Optimizer used to train the Encoder model.

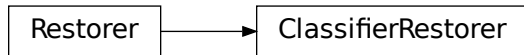
Parameters **model** (`tf.keras.optimizers.Optimizer`) – The placeholder Optimizer in which values from the checkpoint will be restored.

Return type `OptimizerV2`

Returns Restored optimizer.

4.10.4 ClassifierRestorer

Inheritance Diagram



class `ashpy.restorers.ClassifierRestorer` (`logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/0.4.0/docs')`, `ckpts_dir='ckpts'`, `expect_partial=True`)

Bases: `ashpy.restorers.restorer.Restorer`

Convenience `Restorer` for ease of use with the `ClassifierTrainer`.

Methods

<code>restore_model(model)</code>	Restore the Classifier model.
<code>restore_optimizer(optimizer)</code>	Restore the Optimizer used to train the Classifier model.

Attributes

<code>checkpoint_map</code>	Get the map of the ids in the checkpoint.
-----------------------------	---

restore_model (*model*)

Restore the Classifier model.

Parameters **model** (`tf.keras.Model`) – The placeholder model in which values from the checkpoint will be restored.

Return type `Model`

Returns Restored model.

Warning: When restoring a `tf.keras.Model` object from checkpoint assure that the model has been correctly built and instantiated by firstly calling it on some sample inputs. In the case of a model built with either the Sequential or Functional API an exception will be raised; for a model built with the Chainer API it will fail silently, restoration will be “successful” but no values will actually be restored since there are no valid placeholder as the model has not be built yet.

restore_optimizer (*optimizer*)

Restore the Optimizer used to train the Classifier model.

Parameters `model` (`tf.keras.optimizers.Optimizer`) – The placeholder Optimizer in which values from the checkpoint will be restored.

Return type `OptimizerV2`

Returns Restored optimizer.

Modules

<code>restorer</code>	Primitive Restorer, can be used standalone.
<code>classifier</code>	Convenience <code>Restorer</code> to be used with <code>ashpy.trainers.classifier</code> .
<code>gan</code>	Convenience <code>Restorer</code> to be used with <code>ashpy.trainers.gan</code> .

4.10.5 restorer

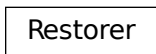
Primitive Restorer, can be used standalone.

Classes

<code>Restorer</code>	<code>Restorer</code> provide a way to restore objects from <code>tf.train.Checkpoint</code> .
-----------------------	--

Restorer

Inheritance Diagram



class `ashpy.restorers.restorer.Restorer` (`logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/ckpts_dir='ckpts', expect_partial=True)`)

Bases: `object`

`Restorer` provide a way to restore objects from `tf.train.Checkpoint`.

Can be standalone.

Methods

<code>__init__([logdir, ckpts_dir, expect_partial])</code>	Initialize the Restorer.
<code>get_global_step()</code>	Return the restored global_step.
<code>get_steps_per_epoch()</code>	Return the restored global_step.
<code>restore_callback(callback, callback_ckpt_id)</code>	Return the restored callbacks.
<code>restore_object(placeholder, object_ckpt_id)</code>	Restore a placeholder from a checkpoint using the specified id.

Attributes

<code>checkpoint_map</code>	Get the map of the ids in the checkpoint.
-----------------------------	---

`__init__ (logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log'), ckpts_dir='ckpts', expect_partial=True)`
Initialize the Restorer.

Parameters

- **logdir** (*str*) – Path to the directory with the logs.
- **ckpts_dir** (*str*) – Name of the directory with the checkpoints to restore.
- **expect_partial** (*bool*) – Whether to expect partial restoring or not. Default to true.
For more information see the docs for `tf.train.Checkpoint.restore()`.

Return type `None`

static `_check_model_construction (restored_model)`

Optimistically check that the model.weights property returns a non empty-list.

The underlying assumption is that Models created via the sub-classing API, when restored without being properly constructed AKA called on some input, will have empty lists as layers.weights.

TODO: add docs for the exception. TODO: add test case for the Sequential without input shape

Return type `bool`

`_restore_checkpoint (checkpoint, partial=True)`

Restore or initialize the persistence layer (checkpoint).

checkpoint_map

Get the map of the ids in the checkpoint.

Map is a Dict where keys are the *ids* in the checkpoint and the values are the string representation of the types.

Return type `Optional[Dict[str, str]]`

Returns Dict if the map is found, else None.

`get_global_step()`

Return the restored global_step.

Return type `Variable`

`get_steps_per_epoch()`

Return the restored global_step.

Return type `Variable`

restore_callback (*callback, callback_ckpt_id*)

Return the restored callbacks.

Return type `List[Callback]`

restore_object (*placeholder, object_ckpt_id*)

Restore a placeholder from a checkpoint using the specified id.

Warning: When restoring a `tf.keras.Model` object from checkpoint assure that the model has been correctly built and instantiated by firstly calling it on some sample inputs. In the case of a model built with either the Sequential or Functional API an exception will be raised; for a model built with the Chainer API it will fail silently, restoration will be “successful” but no values will actually be restored since there are no valid placeholder as the model has not be built yet.

TODO: Args TODO: Example

Exceptions

ModelNotConstructedError

Exception raised while restoring sub-classed Model before having called it on data.

ashpy.restorers.restorer.ModelNotConstructedError

exception `ashpy.restorers.restorer.ModelNotConstructedError`

Bases: `Exception`

Exception raised while restoring sub-classed Model before having called it on data.

Warning: When restoring a `tf.keras.Model` object from checkpoint assure that the model has been correctly built and instantiated by firstly calling it on some sample inputs. In the case of a model built with either the Sequential or Functional API an exception will be raised; for a model built with the Chainer API it will fail silently, restoration will be “successful” but no values will actually be restored since there are no valid placeholder as the model has not be built yet.

exception `ashpy.restorers.restorer.ModelNotConstructedError`

Bases: `Exception`

Exception raised while restoring sub-classed Model before having called it on data.

Warning: When restoring a `tf.keras.Model` object from checkpoint assure that the model has been correctly built and instantiated by firstly calling it on some sample inputs. In the case of a model built with either the Sequential or Functional API an exception will be raised; for a model built with the Chainer API it will fail silently, restoration will be “successful” but no values will actually be restored since there are no valid placeholder as the model has not be built yet.

class `ashpy.restorers.restorer.Restorer` (*logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/0.4.0'), ckpts_dir='ckpts', expect_partial=True*)

Bases: `object`

Restorer provide a way to restore objects from `tf.train.Checkpoint`.

Can be standalone.

__init__ (*logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log')*,
ckpts_dir='ckpts', expect_partial=True)
 Initialize the Restorer.

Parameters

- **logdir** (*str*) – Path to the directory with the logs.
- **ckpts_dir** (*str*) – Name of the directory with the checkpoints to restore.
- **expect_partial** (*bool*) – Whether to expect partial restoring or not. Default to true.
 For more information see the docs for `tf.train.Checkpoint.restore()`.

Return type `None`

static _check_model_construction (*restored_model*)

Optimistically check that the `model.weights` property returns a non empty-list.

The underlying assumption is that Models created via the sub-classing API, when restored without being properly constructed AKA called on some input, will have empty lists as `layers.weights`.

TODO: add docs for the exception. TODO: add test case for the Sequential without input shape

Return type `bool`

_restore_checkpoint (*checkpoint, partial=True*)

Restore or initialize the persistence layer (checkpoint).

checkpoint_map

Get the map of the ids in the checkpoint.

Map is a Dict where keys are the *ids* in the checkpoint and the values are the string representation of the types.

Return type `Optional[Dict[str, str]]`

Returns Dict if the map is found, else None.

get_global_step ()

Return the restored `global_step`.

Return type `Variable`

get_steps_per_epoch ()

Return the restored `global_step`.

Return type `Variable`

restore_callback (*callback, callback_ckpt_id*)

Return the restored callbacks.

Return type `List[Callback]`

restore_object (*placeholder, object_ckpt_id*)

Restore a placeholder from a checkpoint using the specified id.

Warning: When restoring a `tf.keras.Model` object from checkpoint assure that the model has been correctly built and instantiated by firstly calling it on some sample inputs. In the case of a model built with either the Sequential or Functional API an exception will be raised; for a model built with the Chainer API it will fail silently, restoration will be “successful” but no values will actually be restored since there are no valid placeholder as the model has not be built yet.

TODO: Args TODO: Example

4.10.6 classifier

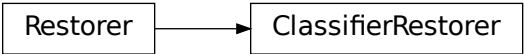
Convenience Restorer to be used with `ashpy.trainers.classifier`.

Classes

<code>ClassifierRestorer</code>	Convenience Restorer for ease of use with the ClassifierTrainer.
---------------------------------	--

ClassifierRestorer

Inheritance Diagram



class `ashpy.restorers.classifier.ClassifierRestorer` (`logdir=PosixPath('/home/docs/checkouts/readthedocs.org')`
`ckpts_dir='ckpts'`, `expect_partial=True`) *ex-*

Bases: `ashpy.restorers.restorer.Restorer`

Convenience Restorer for ease of use with the ClassifierTrainer.

Methods

<code>restore_model(model)</code>	Restore the Classifier model.
<code>restore_optimizer(optimizer)</code>	Restore the Optimizer used to train the Classifier model.

Attributes

<code>checkpoint_map</code>	Get the map of the ids in the checkpoint.
-----------------------------	---

restore_model (*model*)

Restore the Classifier model.

Parameters `model` (`tf.keras.Model`) – The placeholder model in which values from the checkpoint will be restored.

Return type `Model`

Returns Restored model.

Warning: When restoring a `tf.keras.Model` object from checkpoint assure that the model has been correctly built and instantiated by firstly calling it on some sample inputs. In the case of a model built with either the Sequential or Functional API an exception will be raised; for a model built with the Chainer API it will fail silently, restoration will be “successful” but no values will actually be restored since there are no valid placeholder as the model has not be built yet.

restore_optimizer (*optimizer*)

Restore the Optimizer used to train the Classifier model.

Parameters **model** (`tf.keras.optimizers.Optimizer`) – The placeholder Optimizer in which values from the checkpoint will be restored.

Return type `OptimizerV2`

Returns Restored optimizer.

class `ashpy.restorers.classifier.ClassifierRestorer` (`logdir=PosixPath('/home/docs/checkouts/readthedocs.org')`
`ckpts_dir='ckpts',` *ex-*
`pect_partial=True)`

Bases: `ashpy.restorers.restorer.Restorer`

Convenience Restorer for ease of use with the ClassifierTrainer.

restore_model (*model*)

Restore the Classifier model.

Parameters **model** (`tf.keras.Model`) – The placeholder model in which values from the checkpoint will be restored.

Return type `Model`

Returns Restored model.

Warning: When restoring a `tf.keras.Model` object from checkpoint assure that the model has been correctly built and instantiated by firstly calling it on some sample inputs. In the case of a model built with either the Sequential or Functional API an exception will be raised; for a model built with the Chainer API it will fail silently, restoration will be “successful” but no values will actually be restored since there are no valid placeholder as the model has not be built yet.

restore_optimizer (*optimizer*)

Restore the Optimizer used to train the Classifier model.

Parameters **model** (`tf.keras.optimizers.Optimizer`) – The placeholder Optimizer in which values from the checkpoint will be restored.

Return type `OptimizerV2`

Returns Restored optimizer.

4.10.7 gan

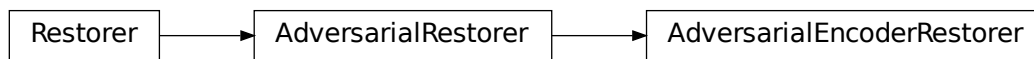
Convenience Restorer to be used with `ashpy.trainers.gan`.

Classes

<i>AdversarialEncoderRestorer</i>	Convenience Restorer for ease of use with the EncoderTrainer.
<i>AdversarialRestorer</i>	Convenience Restorer for ease of use with the AdversairalTrainer.

AdversarialEncoderRestorer

Inheritance Diagram



class `ashpy.restorers.gan.AdversarialEncoderRestorer` (`logdir=PosixPath('/home/docs/checkouts/readthedocs.o`
`ckpts_dir='ckpts',` `ex-`
`pect_partial=True)`

Bases: `ashpy.restorers.gan.AdversarialRestorer`

Convenience Restorer for ease of use with the EncoderTrainer.

Methods

<i>restore_encoder</i> (model)	Restore the Encoder model.
<i>restore_encoder_optimizer</i> (optimizer)	Restore the Optimizer used to train the Encoder model.

Attributes

<i>checkpoint_map</i>	Get the map of the ids in the checkpoint.
-----------------------	---

restore_encoder (*model*)
 Restore the Encoder model.

Parameters **model** (`tf.keras.Model`) – The placeholder model in which values from the checkpoint will be restored.

Return type `Model`

Returns Restored model.

Warning: When restoring a `tf.keras.Model` object from checkpoint assure that the model has been correctly built and instantiated by firstly calling it on some sample inputs. In the case of a model

built with either the Sequential or Functional API an exception will be raised; for a model built with the Chainer API it will fail silently, restoration will be “successful” but no values will actually be restored since there are no valid placeholder as the model has not be built yet.

restore_encoder_optimizer (*optimizer*)

Restore the Optimizer used to train the Encoder model.

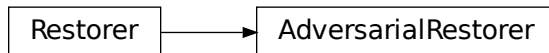
Parameters **model** (`tf.keras.optimizers.Optimizer`) – The placeholder Optimizer in which values from the checkpoint will be restored.

Return type `OptimizerV2`

Returns Restored optimizer.

AdversarialRestorer

Inheritance Diagram



class `ashpy.restorers.gan.AdversarialRestorer` (*logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_bu*
ckpts_dir='ckpts', expect_partial=True))

Bases: `ashpy.restorers.restorer.Restorer`

Convenience Restorer for ease of use with the AdversairalTrainer.

Methods

<code>restore_discriminator(model)</code>	Restore the Discriminator model.
<code>restore_discriminator_optimizer(optimizer)</code>	Restore the Optimizer used to train the Discriminator model.
<code>restore_generator(model)</code>	Restore the Generator model.
<code>restore_generator_optimizer(optimizer)</code>	Restore the Optimizer used to train the Generator model.

Attributes

<code>checkpoint_map</code>	Get the map of the ids in the checkpoint.
-----------------------------	---

restore_discriminator (*model*)

Restore the Discriminator model.

Parameters **model** (`tf.keras.Model`) – The placeholder model in which values from the

checkpoint will be restored.

Return type `Model`

Returns Restored model.

Warning: When restoring a `tf.keras.Model` object from checkpoint assure that the model has been correctly built and instantiated by firstly calling it on some sample inputs. In the case of a model built with either the Sequential or Functional API an exception will be raised; for a model built with the Chainer API it will fail silently, restoration will be “successful” but no values will actually be restored since there are no valid placeholder as the model has not be built yet.

restore_discriminator_optimizer (*optimizer*)

Restore the Optimizer used to train the Discriminator model.

Parameters **model** (`tf.keras.optimizers.Optimizer`) – The placeholder Optimizer in which values from the checkpoint will be restored.

Return type `OptimizerV2`

Returns Restored optimizer.

restore_generator (*model*)

Restore the Generator model.

Parameters **model** (`tf.keras.Model`) – The placeholder model in which values from the checkpoint will be restored.

Return type `Model`

Returns Restored model.

Warning: When restoring a `tf.keras.Model` object from checkpoint assure that the model has been correctly built and instantiated by firstly calling it on some sample inputs. In the case of a model built with either the Sequential or Functional API an exception will be raised; for a model built with the Chainer API it will fail silently, restoration will be “successful” but no values will actually be restored since there are no valid placeholder as the model has not be built yet.

restore_generator_optimizer (*optimizer*)

Restore the Optimizer used to train the Generator model.

Parameters **model** (`tf.keras.optimizers.Optimizer`) – The placeholder Optimizer in which values from the checkpoint will be restored.

Return type `OptimizerV2`

Returns Restored optimizer.

class `ashpy.restorers.gan.AdversarialEncoderRestorer` (*logdir=PosixPath('/home/docs/checkouts/readthedocs.o*
ckpts_dir='ckpts', *ex-*
pect_partial=True)

Bases: `ashpy.restorers.gan.AdversarialRestorer`

Convenience Restorer for ease of use with the EncoderTrainer.

restore_encoder (*model*)

Restore the Encoder model.

Parameters `model` (`tf.keras.Model`) – The placeholder model in which values from the checkpoint will be restored.

Return type `Model`

Returns Restored model.

Warning: When restoring a `tf.keras.Model` object from checkpoint assure that the model has been correctly built and instantiated by firstly calling it on some sample inputs. In the case of a model built with either the Sequential or Functional API an exception will be raised; for a model built with the Chainer API it will fail silently, restoration will be “successful” but no values will actually be restored since there are no valid placeholder as the model has not be built yet.

restore_encoder_optimizer (`optimizer`)

Restore the Optimizer used to train the Encoder model.

Parameters `model` (`tf.keras.optimizers.Optimizer`) – The placeholder Optimizer in which values from the checkpoint will be restored.

Return type `OptimizerV2`

Returns Restored optimizer.

class `ashpy.restorers.gan.AdversarialRestorer` (`logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_bu`
`ckpts_dir='ckpts', expect_partial=True)`)

Bases: `ashpy.restorers.restorer.Restorer`

Convenience Restorer for ease of use with the AdversairalTrainer.

restore_discriminator (`model`)

Restore the Discriminator model.

Parameters `model` (`tf.keras.Model`) – The placeholder model in which values from the checkpoint will be restored.

Return type `Model`

Returns Restored model.

Warning: When restoring a `tf.keras.Model` object from checkpoint assure that the model has been correctly built and instantiated by firstly calling it on some sample inputs. In the case of a model built with either the Sequential or Functional API an exception will be raised; for a model built with the Chainer API it will fail silently, restoration will be “successful” but no values will actually be restored since there are no valid placeholder as the model has not be built yet.

restore_discriminator_optimizer (`optimizer`)

Restore the Optimizer used to train the Discriminator model.

Parameters `model` (`tf.keras.optimizers.Optimizer`) – The placeholder Optimizer in which values from the checkpoint will be restored.

Return type `OptimizerV2`

Returns Restored optimizer.

restore_generator (`model`)

Restore the Generator model.

Parameters `model` (`tf.keras.Model`) – The placeholder model in which values from the checkpoint will be restored.

Return type `Model`

Returns Restored model.

Warning: When restoring a `tf.keras.Model` object from checkpoint assure that the model has been correctly built and instantiated by firstly calling it on some sample inputs. In the case of a model built with either the Sequential or Functional API an exception will be raised; for a model built with the Chainer API it will fail silently, restoration will be “successful” but no values will actually be restored since there are no valid placeholder as the model has not be built yet.

restore_generator_optimizer (`optimizer`)

Restore the Optimizer used to train the Generator model.

Parameters `model` (`tf.keras.optimizers.Optimizer`) – The placeholder Optimizer in which values from the checkpoint will be restored.

Return type `OptimizerV2`

Returns Restored optimizer.

4.11 ashpy.trainers

Trainers help reducing boilerplate code by bootstrapping models training.

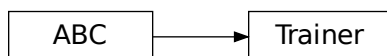
The module contains a primitive Interface and specific trainers that inherits from it.

Classes

<code>Trainer</code>	<code>Trainer</code> provide an interface for all trainers to inherit from.
<code>AdversarialTrainer</code>	Primitive Trainer for GANs subclassed from <code>ashpy.trainers.Trainer</code> .
<code>EncoderTrainer</code>	Primitive Trainer for GANs using an Encoder sub-network.

4.11.1 Trainer

Inheritance Diagram



```
class ashpy.trainers.Trainer(epochs, example_dim, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0'),
                             log_eval_mode=<LogEvalMode.TEST: 1>, global_step=None,
                             metrics=None, callbacks=None)
```

Bases: `abc.ABC`

Trainer provide an interface for all trainers to inherit from.

Methods

<code>__init__(epochs, example_dim[, logdir, ...])</code>	Primitive trainer interface.
<code>call(*args, **kwargs)</code>	Execute the training process.
<code>local_example(example, dims)</code>	Return a local example from a distributed example.
<code>measure_metrics()</code>	Measure the metrics.
<code>model_selection()</code>	Use the metrics to perform model selection.

Attributes

<code>ckpt_id_callbacks</code>	
<code>ckpt_id_global_step</code>	
<code>ckpt_id_steps_per_epoch</code>	
<code>context</code>	Return the training context.

```
__init__(epochs, example_dim, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0'),
          log_eval_mode=<LogEvalMode.TEST: 1>, global_step=None, metrics=None, call-
          backs=None)
```

Primitive trainer interface. Handles model saving and restore.

Parameters

- **epochs** (*int*) – Number of training epochs.
- **example_dim** (*Tuple[int, int]*) – Dimension of an example. In the case of GANs the example has dimension (2,1) since it's composed by a tuple in which the first element is a tuple with 2 components and the second component is a single element. In the case of classifier the example has dimension (1, 1) since it's composed by the example and the label.
- **logdir** (*str*) – Checkpoint and log directory.
- **log_eval_mode** (`py:class:ashpy.modes.LogEvalMode`) – to use when evaluating and logging.
- **global_step** (Optional[`py:class:ashpy.modes.LogEvalMode`]) – tf.Variable that keeps track of the training steps.
- **metrics** (Optional[List | Tuple [ashpy.metrics.Metric]]) – list or tuple of metrics.
- **callbacks** (Optional[List[ashpy.callbacks.Callback]]) – list or tuple of callbacks to handle events.

Return type `None`

```
_build_and_restore_models(dataset)
```

Build and restore a Subclassed model by firstly calling it on some data.

static _check_name_collision (*objects, obj_type*)

Check that all objects have unique name.

_current_epoch ()

Get the current epoch using the (restored) variables.

Return type Tensor

Returns current_epoch (*tf.Tensor*) – the current epoch of training.

_dataset_from_example (*example, dims*)

Get a dataset from a given example.

Return type DatasetV2

Returns The dataset containing only the example.

_generate_checkpoint_map ()

Generate a human readable map of the id and type mapping in the checkpoint.

_log_metrics_and_reset ()

Call for each metric the log and reset_states.

_measure_performance ()

Measure performance on dataset.

_measure_performance_if_needed (*example, measure_performance_freq*)

Measure performance if needed.

Measure performance if self._global_step % measure_performance_freq is 0.

_on_batch_end ()

Handle the end of a training batch.

Return type None

_on_batch_start ()

Handle the start of a training batch.

Return type None

_on_epoch_end ()

Handle the end of the training epoch.

Return type None

_on_epoch_start ()

Handle the start of the training epoch.

Return type None

_on_exception ()

Handle the exception.

Return type None

_on_train_end ()

Handle the end of training.

Return type None

_on_train_start ()

Handle the start of training.

Return type None

`_reduce` (*per_replica_tensor*, *reduce_op*)

Reduce the input tensor in a distributed fashion, using the specified op.

`_restore_or_init` ()

Restore or initialize the persistence layer (checkpoint).

`_save` ()

Save the current checkpointable object status.

`_update_checkpoint` (*ckpt_dict*)

Update the checkpoint with the new checkpoint dictionary.

`_update_global_batch_size` (*dataset*, *executors=None*)

Set the *self._global_batch_size* variable where needed.

Parameters **`dataset`** (`tf.data.Dataset`) – a dataset from which the batch size will be extracted.

:param executors (`Union[List[ashpy.losses.executor.Executor], ashpy.losses.executor.Executor]`) – with the property “global_batch_size”.

`_validate_callbacks` ()

Check if every callback is an `ashpy.callbacks.Callback`.

`_validate_metrics` ()

Check if every metric is an `ashpy.metrics.Metric`.

`call` (**args*, ***kwargs*)

Execute the training process.

`context`

Return the training context.

Return type `Context`

`local_example` (*example*, *dims*)

Return a local example from a distributed example.

Returns A local example from a distributed example.

`measure_metrics` ()

Measure the metrics.

Return type `None`

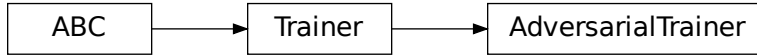
`model_selection` ()

Use the metrics to perform model selection.

Return type `None`

4.11.2 AdversarialTrainer

Inheritance Diagram



class `ashpy.trainers.AdversarialTrainer` (*generator, discriminator, generator_optimizer, discriminator_optimizer, generator_loss, discriminator_loss, epochs, metrics=None, callbacks=None, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/log_eval_mode=<LogEvalMode.TEST: 1>, global_step=None)*)

Bases: `ashpy.trainers.trainer.Trainer`

Primitive Trainer for GANs subclassed from `ashpy.trainers.Trainer`.

Examples

```

import shutil
import operator

generator = models.gans.ConvGenerator(
    layer_spec_input_res=(7, 7),
    layer_spec_target_res=(28, 28),
    kernel_size=(5, 5),
    initial_filters=32,
    filters_cap=16,
    channels=1,
)

discriminator = models.gans.ConvDiscriminator(
    layer_spec_input_res=(28, 28),
    layer_spec_target_res=(7, 7),
    kernel_size=(5, 5),
    initial_filters=16,
    filters_cap=32,
    output_shape=1,
)

# Losses
generator_bce = losses.gan.GeneratorBCE()
minmax = losses.gan.DiscriminatorMinMax()

# Real data
batch_size = 2
mnist_x, mnist_y = tf.zeros((100, 28, 28)), tf.zeros((100,))

# Trainer
epochs = 2
  
```

(continues on next page)

(continued from previous page)

```

logdir = "testlog/adversarial"
metrics = [
    metrics.gan.InceptionScore(
        # Fake inception model
        models.gans.ConvDiscriminator(
            layer_spec_input_res=(299, 299),
            layer_spec_target_res=(7, 7),
            kernel_size=(5, 5),
            initial_filters=16,
            filters_cap=32,
            output_shape=10,
        ),
        model_selection_operator=operator.gt,
    )
]
trainer = trainers.gan.AdversarialTrainer(
    generator=generator,
    discriminator=discriminator,
    generator_optimizer=tf.optimizers.Adam(1e-4),
    discriminator_optimizer=tf.optimizers.Adam(1e-4),
    generator_loss=generator_bce,
    discriminator_loss=minmax,
    epochs=epochs,
    metrics=metrics,
    logdir=logdir,
)

# take only 2 samples to speed up tests
real_data = (
    tf.data.Dataset.from_tensor_slices(
        (tf.expand_dims(mnist_x, -1), tf.expand_dims(mnist_y, -1))).take(batch_size)
    .batch(batch_size)
    .prefetch(1)
)

# Add noise in the same dataset, just by mapping.
# The return type of the dataset must be: tuple(tuple(a,b), noise)
dataset = real_data.map(
    lambda x, y: ((x, y), tf.random.normal(shape=(batch_size, 100)))
)

trainer(dataset)
shutil.rmtree(logdir)

```

```

Initializing checkpoint.
Starting epoch 1.
[1] Saved checkpoint: testlog/adversarial/ckpts/ckpt-1
Epoch 1 completed.
Starting epoch 2.
[2] Saved checkpoint: testlog/adversarial/ckpts/ckpt-2
Epoch 2 completed.
Training finished after 2 epochs.

```

Methods

<code>__init__(generator, discriminator, ...[, ...])</code>	Instantiate a <i>AdversarialTrainer</i> .
<code>call(dataset[, log_freq, ...])</code>	Perform the adversarial training.
<code>train_step(real_xy, g_inputs)</code>	Train step for the AdversarialTrainer.

Attributes

<code>ckpt_id_callbacks</code>	
<code>ckpt_id_discriminator</code>	
<code>ckpt_id_generator</code>	
<code>ckpt_id_global_step</code>	
<code>ckpt_id_optimizer_discriminator</code>	
<code>ckpt_id_optimizer_generator</code>	
<code>ckpt_id_steps_per_epoch</code>	
<code>context</code>	Return the training context.

`__init__(generator, discriminator, generator_optimizer, discriminator_optimizer, generator_loss, discriminator_loss, epochs, metrics=None, callbacks=None, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log'), log_eval_mode=<LogEvalMode.TEST: 1>, global_step=None)`
 Instantiate a *AdversarialTrainer*.

Parameters

- **generator** (`tf.keras.Model`) – A `tf.keras.Model` describing the Generator part of a GAN.
- **discriminator** (`tf.keras.Model`) – A `tf.keras.Model` describing the Discriminator part of a GAN.
- **generator_optimizer** (`tf.optimizers.Optimizer`) – A `tf.optimizers` to use for the Generator.
- **discriminator_optimizer** (`tf.optimizers.Optimizer`) – A `tf.optimizers` to use for the Discriminator.
- **generator_loss** (`ashpy.losses.executor.Executor`) – A ash Executor to compute the loss of the Generator.
- **discriminator_loss** (`ashpy.losses.executor.Executor`) – A ash Executor to compute the loss of the Discriminator.
- **epochs** (`int`) – number of training epochs.
- **metrics** (`Union[Tuple[Metric], List[Metric], None]`) – (List): list of `ashpy.metrics.Metric` to measure on training and validation data.
- **callbacks** (`List`) – list of `ashpy.callbacks.Callback` to measure on training and validation data.
- **logdir** (`Union[Path, str]`) – checkpoint and log directory.
- **log_eval_mode** (`LogEvalMode`) – models' mode to use when evaluating and logging.
- **global_step** (`Optional[tf.Variable]`) – `tf.Variable` that keeps track of the training steps.

Returns `None`

`_build_and_restore_models` (*dataset*)

Build and restore a Subclassed model by firstly calling it on some data.

`_train_step`

Training step with the distribution strategy.

`call` (*dataset, log_freq=10, measure_performance_freq=10*)

Perform the adversarial training.

Parameters

- **`dataset`** (*tf.data.Dataset*) – The adversarial training dataset.
- **`log_freq`** (*int*) – Specifies how many steps to run before logging the losses, e.g. *log_frequency=10* logs every 10 steps of training. Pass *log_frequency<=0* in case you don't want to log.
- **`measure_performance_freq`** (*int*) – Specifies how many steps to run before measuring the performance, e.g. *measure_performance_freq=10* measures performance every 10 steps of training. Pass *measure_performance_freq<=0* in case you don't want to measure performance.

`train_step` (*real_xy, g_inputs*)

Train step for the AdversarialTrainer.

Parameters

- **`real_xy`** – input batch as extracted from the input dataset. (features, label) pair.
- **`g_inputs`** – batch of generator_input as generated from the input dataset.

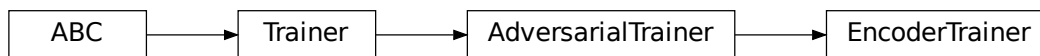
Returns

d_loss, g_loss, fake –

discriminator, generator loss values. fake is the generator output.

4.11.3 EncoderTrainer

Inheritance Diagram



```

class ashpy.trainers.EncoderTrainer(generator, discriminator, encoder, genera-
tor_optimizer, discriminator_optimizer, en-
coder_optimizer, generator_loss, discriminator_loss,
encoder_loss, epochs, metrics=None, callbacks=None,
logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/che
log_eval_mode=<LogEvalMode.TEST: 1>,
global_step=None)
    
```

Bases: `ashpy.trainers.gan.AdversarialTrainer`

Primitive Trainer for GANs using an Encoder sub-network.

The implementation is thought to be used with the BCE losses. To use another loss function consider subclassing the model and overriding the `train_step` method.

Examples

```
from pathlib import Path

import shutil
import operator

def real_gen():
    label = 0
    for _ in tf.range(100):
        yield ((10.0,), (label,))

latent_dim = 100

generator = tf.keras.Sequential([tf.keras.layers.Dense(1)])

left_input = tf.keras.layers.Input(shape=(1,))
left = tf.keras.layers.Dense(10, activation=tf.nn.elu)(left_input)

right_input = tf.keras.layers.Input(shape=(latent_dim,))
right = tf.keras.layers.Dense(10, activation=tf.nn.elu)(right_input)

net = tf.keras.layers.Concatenate()([left, right])
out = tf.keras.layers.Dense(1)(net)

discriminator = tf.keras.Model(inputs=[left_input, right_input], outputs=[out])

encoder = tf.keras.Sequential([tf.keras.layers.Dense(latent_dim)])

# Losses
generator_bce = losses.gan.GeneratorBCE()
encoder_bce = losses.gan.EncoderBCE()
minmax = losses.gan.DiscriminatorMinMax()

epochs = 2

# Fake pre-trained classifier
num_classes = 1
classifier = tf.keras.Sequential(
    [tf.keras.layers.Dense(10), tf.keras.layers.Dense(num_classes)]
)

logdir = Path("testlog") / "adversarial_encoder"

if logdir.exists():
    shutil.rmtree(logdir)

metrics = [metrics.gan.EncodingAccuracy(classifier)]

trainer = trainers.gan.EncoderTrainer(
```

(continues on next page)

(continued from previous page)

```

generator=generator,
discriminator=discriminator,
encoder=encoder,
discriminator_optimizer=tf.optimizers.Adam(1e-4),
generator_optimizer=tf.optimizers.Adam(1e-5),
encoder_optimizer=tf.optimizers.Adam(1e-6),
generator_loss=generator_bce,
discriminator_loss=minmax,
encoder_loss=encoder_bce,
epochs=epochs,
metrics=metrics,
logdir=logdir,
)

batch_size = 10
discriminator_input = tf.data.Dataset.from_generator(
    real_gen, (tf.float32, tf.int64), ((1), (1))
).batch(batch_size)

dataset = discriminator_input.map(
    lambda x, y: ((x, y), tf.random.normal(shape=(batch_size, latent_dim)))
)

trainer(dataset)

shutil.rmtree(logdir)

```

```

Initializing checkpoint.
Starting epoch 1.
[10] Saved checkpoint: testlog/adversarial_encoder/ckpts/ckpt-1
Epoch 1 completed.
Starting epoch 2.
[20] Saved checkpoint: testlog/adversarial_encoder/ckpts/ckpt-2
Epoch 2 completed.
Training finished after 2 epochs.

```

Methods

<code>__init__(generator, discriminator, encoder, ...)</code>	Instantiate a <i>EncoderTrainer</i> .
<code>call(dataset[, log_freq, ...])</code>	Perform the adversarial training.
<code>train_step(real_xy, g_inputs)</code>	Adversarial training step.

Attributes

<code>ckpt_id_callbacks</code>
<code>ckpt_id_discriminator</code>
<code>ckpt_id_encoder</code>
<code>ckpt_id_generator</code>
<code>ckpt_id_global_step</code>
<code>ckpt_id_optimizer_discriminator</code>
<code>ckpt_id_optimizer_encoder</code>

Continued on next page

Table 341 – continued from previous page

ckpt_id_optimizer_generator	
ckpt_id_steps_per_epoch	
context	Return the training context.

__init__(*generator*, *discriminator*, *encoder*, *generator_optimizer*, *discriminator_optimizer*, *encoder_optimizer*, *generator_loss*, *discriminator_loss*, *encoder_loss*, *epochs*, *metrics=None*, *callbacks=None*, *logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log')*, *log_eval_mode=<LogEvalMode.TEST: 1>*, *global_step=None*)
 Instantiate a *EncoderTrainer*.

Parameters

- **generator** (*tf.keras.Model*) – A *tf.keras.Model* describing the Generator part of a GAN.
- **discriminator** (*tf.keras.Model*) – A *tf.keras.Model* describing the Discriminator part of a GAN.
- **encoder** (*tf.keras.Model*) – A *tf.keras.Model* describing the Encoder part of a GAN.
- **generator_optimizer** (*tf.optimizers.Optimizer*) – A *tf.optimizers* to use for the Generator.
- **discriminator_optimizer** (*tf.optimizers.Optimizer*) – A *tf.optimizers* to use for the Discriminator.
- **encoder_optimizer** (*tf.optimizers.Optimizer*) – A *tf.optimizers* to use for the Encoder.
- **generator_loss** (*ashpy.losses.executor.Executor*) – A *ash* Executor to compute the loss of the Generator.
- **discriminator_loss** (*ashpy.losses.executor.Executor*) – A *ash* Executor to compute the loss of the Discriminator.
- **encoder_loss** (*ashpy.losses.executor.Executor*) – A *ash* Executor to compute the loss of the Discriminator.
- **epochs** (*int*) – number of training epochs.
- **metrics** (*Optional[List[Metric]]*) – (List): list of *ashpy.metrics.Metric* to measure on training and validation data.
- **callbacks** (*List*) – List of *ashpy.callbacks.Callback* to call on events
- **logdir** (*Union[Path, str]*) – checkpoint and log directory.
- **log_eval_mode** (*ashpy.modes.LogEvalMode*) – models' mode to use when evaluating and logging.
- **global_step** (*Optional[tf.Variable]*) – *tf.Variable* that keeps track of the training steps.

_build_and_restore_models (*dataset*)

Build and restore a Subclassed model by firstly calling it on some data.

_train_step

Perform the training step using the distribution strategy.

call (*dataset*, *log_freq=10*, *measure_performance_freq=10*)
 Perform the adversarial training.

Parameters

- **dataset** (*tf.data.Dataset*) – The adversarial training dataset.
- **log_freq** (*int*) – Specifies how many steps to run before logging the losses, e.g. *log_frequency=10* logs every 10 steps of training. Pass *log_frequency<=0* in case you don't want to log.
- **measure_performance_freq** (*int*) – Specifies how many steps to run before measuring the performance, e.g. *measure_performance_freq=10* measures performance every 10 steps of training. Pass *measure_performance_freq<=0* in case you don't want to measure performance.

train_step (*real_xy*, *g_inputs*)
 Adversarial training step.

Parameters

- **real_xy** – input batch as extracted from the discriminator input dataset. (features, label) pair
- **g_inputs** – batch of noise as generated by the generator input dataset.

Returns *d_loss*, *g_loss*, *e_loss* – discriminator, generator, encoder loss values.

Modules

<i>trainer</i>	Primitive Trainer Interface.
<i>classifier</i>	Primitive Trainer Interface.
<i>gan</i>	Collection of GANs trainers.

4.11.4 trainer

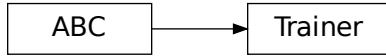
Primitive Trainer Interface.

Classes

<i>Trainer</i>	<i>Trainer</i> provide an interface for all trainers to inherit from.
----------------	---

Trainer

Inheritance Diagram



```

class ashpy.trainers.trainer.Trainer (epochs,                                example_dim,
                                       logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/ch
                                       log_eval_mode=<LogEvalMode.TEST: 1>,
                                       global_step=None, metrics=None, callbacks=None)
  
```

Bases: `abc.ABC`

Trainer provide an interface for all trainers to inherit from.

Methods

<code>__init__(epochs, example_dim[, logdir, ...])</code>	Primitive trainer interface.
<code>call(*args, **kwargs)</code>	Execute the training process.
<code>local_example(example, dims)</code>	Return a local example from a distributed example.
<code>measure_metrics()</code>	Measure the metrics.
<code>model_selection()</code>	Use the metrics to perform model selection.

Attributes

<code>ckpt_id_callbacks</code>	
<code>ckpt_id_global_step</code>	
<code>ckpt_id_steps_per_epoch</code>	
<code>context</code>	Return the training context.

```

__init__(epochs, example_dim, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0
        log_eval_mode=<LogEvalMode.TEST: 1>, global_step=None, metrics=None, call-
        backs=None)
  
```

Primitive trainer interface. Handles model saving and restore.

Parameters

- **epochs** (*int*) – Number of training epochs.
- **example_dim** (*Tuple[int, int]*) – Dimension of an example. In the case of GANs the example has dimension (2,1) since it's composed by a tuple in which the first element is a tuple with 2 components and the second component is a single element. In the case of classifier the example has dimension (1, 1) since it's composed by the example and the label.
- **logdir** (*str*) – Checkpoint and log directory.
- **log_eval_mode** (*py:class:ashpy.modes.LogEvalMode*) – to use when evaluating and

logging.

- **global_step** (Optional[py:class:ashpy.modes.LogEvalMode]) – tf.Variable that keeps track of the training steps.
- **metrics** (Optional[List | Tuple [ashpy.metrics.Metric]]) – list or tuple of metrics.
- **callbacks** (Optional[List[ashpy.callbacks.Callback]]) – list or tuple of callbacks to handle events.

Return type None

__build_and_restore_models (*dataset*)

Build and restore a Subclassed model by firstly calling it on some data.

static __check_name_collision (*objects, obj_type*)

Check that all objects have unique name.

__current_epoch ()

Get the current epoch using the (restored) variables.

Return type Tensor

Returns current_epoch (tf.Tensor) – the current epoch of training.

__dataset_from_example (*example, dims*)

Get a dataset from a given example.

Return type DatasetV2

Returns The dataset containing only the example.

__generate_checkpoint_map ()

Generate a human readable map of the id and type mapping in the checkpoint.

__log_metrics_and_reset ()

Call for each metric the log and reset_states.

__measure_performance ()

Measure performance on dataset.

__measure_performance_if_needed (*example, measure_performance_freq*)

Measure performance if needed.

Measure performance if self._global_step % measure_performance_freq is 0.

__on_batch_end ()

Handle the end of a training batch.

Return type None

__on_batch_start ()

Handle the start of a training batch.

Return type None

__on_epoch_end ()

Handle the end of the training epoch.

Return type None

__on_epoch_start ()

Handle the start of the training epoch.

Return type None

`_on_exception()`

Handle the exception.

Return type `None`

`_on_train_end()`

Handle the end of training.

Return type `None`

`_on_train_start()`

Handle the start of training.

Return type `None`

`_reduce(per_replica_tensor, reduce_op)`

Reduce the input tensor in a distributed fashion, using the specified op.

`_restore_or_init()`

Restore or initialize the persistence layer (checkpoint).

`_save()`

Save the current checkpointable object status.

`_update_checkpoint(ckpt_dict)`

Update the checkpoint with the new checkpoint dictionary.

`_update_global_batch_size(dataset, executors=None)`

Set the `self._global_batch_size` variable where needed.

Parameters **dataset** (`tf.data.Dataset`) – a dataset from which the batch size will be extracted.

:param executors (`Union[List[ashpy.losses.executor.Executor], ashpy.losses.executor.Executor]`) – with the property “global_batch_size”.

`_validate_callbacks()`

Check if every callback is an `ashpy.callbacks.Callback`.

`_validate_metrics()`

Check if every metric is an `ashpy.metrics.Metric`.

`call(*args, **kwargs)`

Execute the training process.

`context`

Return the training context.

Return type `Context`

`local_example(example, dims)`

Return a local example from a distributed example.

Returns A local example from a distributed example.

`measure_metrics()`

Measure the metrics.

Return type `None`

`model_selection()`

Use the metrics to perform model selection.

Return type `None`


```
class ashpy.trainers.trainer.Trainer (epochs, example_dim,
                                       logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/ch
                                       log_eval_mode=<LogEvalMode.TEST: 1>,
                                       global_step=None, metrics=None, callbacks=None)
```

Bases: `abc.ABC`

`Trainer` provide an interface for all trainers to inherit from.

```
__init__ (epochs, example_dim, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0
         log_eval_mode=<LogEvalMode.TEST: 1>, global_step=None, metrics=None, call-
         backs=None)
```

Primitive trainer interface. Handles model saving and restore.

Parameters

- **epochs** (*int*) – Number of training epochs.
- **example_dim** (*Tuple[int, int]*) – Dimension of an example. In the case of GANs the example has dimension (2,1) since it's composed by a tuple in which the first element is a tuple with 2 components and the second component is a single element. In the case of classifier the example has dimension (1, 1) since it's composed by the example and the label.
- **logdir** (*str*) – Checkpoint and log directory.
- **log_eval_mode** (*py:class:ashpy.modes.LogEvalMode*) – to use when evaluating and logging.
- **global_step** (*Optional[py:class:ashpy.modes.LogEvalMode]*) – tf.Variable that keeps track of the training steps.
- **metrics** (*Optional[List | Tuple [ashpy.metrics.Metric]]*) – list or tuple of metrics.
- **callbacks** (*Optional[List[ashpy.callbacks.Callback]]*) – list or tuple of callbacks to handle events.

Return type `None`

```
_build_and_restore_models (dataset)
```

Build and restore a Subclassed model by firstly calling it on some data.

```
static _check_name_collision (objects, obj_type)
```

Check that all objects have unique name.

```
_current_epoch ()
```

Get the current epoch using the (restored) variables.

Return type `Tensor`

Returns `current_epoch (tf.Tensor)` – the current epoch of training.

```
_dataset_from_example (example, dims)
```

Get a dataset from a given example.

Return type `DatasetV2`

Returns The dataset containing only the example.

```
_generate_checkpoint_map ()
```

Generate a human readable map of the id and type mapping in the checkpoint.

```
_log_metrics_and_reset ()
```

Call for each metric the log and reset_states.

`_measure_performance()`

Measure performance on dataset.

`_measure_performance_if_needed(example, measure_performance_freq)`

Measure performance if needed.

Measure performance if `self._global_step % measure_performance_freq` is 0.

`_on_batch_end()`

Handle the end of a training batch.

Return type `None`

`_on_batch_start()`

Handle the start of a training batch.

Return type `None`

`_on_epoch_end()`

Handle the end of the training epoch.

Return type `None`

`_on_epoch_start()`

Handle the start of the training epoch.

Return type `None`

`_on_exception()`

Handle the exception.

Return type `None`

`_on_train_end()`

Handle the end of training.

Return type `None`

`_on_train_start()`

Handle the start of training.

Return type `None`

`_reduce(per_replica_tensor, reduce_op)`

Reduce the input tensor in a distributed fashion, using the specified op.

`_restore_or_init()`

Restore or initialize the persistence layer (checkpoint).

`_save()`

Save the current checkpointable object status.

`_update_checkpoint(ckpt_dict)`

Update the checkpoint with the new checkpoint dictionary.

`_update_global_batch_size(dataset, executors=None)`

Set the `self._global_batch_size` variable where needed.

Parameters `dataset` (`tf.data.Dataset`) – a dataset from which the batch size will be extracted.

:param executors (`Union[List[ashpy.losses.executor.Executor], ashpy.losses.executor.Executor]`) – with the property “`global_batch_size`”.

`_validate_callbacks()`
Check if every callback is an `ashpy.callbacks.Callback`.

`_validate_metrics()`
Check if every metric is an `ashpy.metrics.Metric`.

`call(*args, **kwargs)`
Execute the training process.

`context`
Return the training context.
Return type `Context`

`local_example(example, dims)`
Return a local example from a distributed example.
Returns A local example from a distributed example.

`measure_metrics()`
Measure the metrics.
Return type `None`

`model_selection()`
Use the metrics to perform model selection.
Return type `None`

4.11.5 classifier

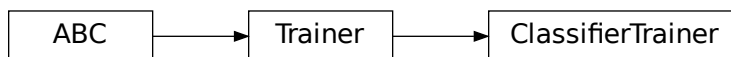
Primitive Trainer Interface.

Classes

<code>ClassifierTrainer</code>	<code>ClassifierTrainer</code> provide the standard training loop for a classifier.
--------------------------------	---

ClassifierTrainer

Inheritance Diagram



```

class ashpy.trainers.classifier.ClassifierTrainer(model, optimizer, loss, epochs,
                                                  metrics=None, callbacks=None,
                                                  logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_
                                                  global_step=None)
  
```

Bases: `ashpy.trainers.trainer.Trainer`

`ClassifierTrainer` provide the standard training loop for a classifier.

Methods

<code>__init__(model, optimizer, loss, epochs[, ...])</code>	Instantiate the <code>ClassifierTrainer</code> trainer.
<code>call(training_set, validation_set[, ...])</code>	Start the training.
<code>train_step(features, labels)</code>	Train step.

Attributes

<code>ckpt_id_callbacks</code>	
<code>ckpt_id_global_step</code>	
<code>ckpt_id_model</code>	
<code>ckpt_id_optimizer</code>	
<code>ckpt_id_steps_per_epoch</code>	
<code>context</code>	Return the training context.

`__init__(model, optimizer, loss, epochs, metrics=None, callbacks=None, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log'), global_step=None)`
Instantiate the `ClassifierTrainer` trainer.

Parameters

- **model** (`tf.keras.Model`) – A `tf.keras.Model` model.
- **optimizer** (`tf.optimizers.Optimizer`) – A `tf.optimizers.Optimizer`.
- **loss** (`ashpy.losses.classifier.ClassifierLoss`) – A loss function built following `ashpy.executors``.
- **epochs** (`int`) – Number of training epochs.
- **metrics** (`Union[Tuple[Metric], List[Metric], None]`) – (Tuple/List): Tuple/List of `ashpy.metrics.metric.Metric` to measure on training and validation data.
- **callbacks** (`List`) – List of `ashpy.callbacks.callback.Callback` to call on events
- **logdir** (`str`) – Checkpoint and log directory.
- **global_step** (Optional[`py:class:tf.Variable`]) – `tf.Variable` that keeps track of the training steps.

Examples

```
def toy_dataset():  
    inputs = tf.expand_dims(tf.range(1, 1000.0), -1)  
    labels = tf.expand_dims(  
        [1 if tf.equal(tf.math.mod(tf.squeeze(i), 2), 0) else 0 for i in_  
↪ inputs], -1
```

(continues on next page)

(continued from previous page)

```

    )
    return tf.data.Dataset.from_tensor_slices((inputs, labels)).shuffle(10).
↳batch(2)

model = tf.keras.Sequential(
    [tf.keras.layers.Dense(10, activation=tf.nn.sigmoid), tf.keras.layers.
↳Dense(2)]
)
optimizer = tf.optimizers.Adam(1e-3)

loss = ClassifierLoss(tf.losses.SparseCategoricalCrossentropy(from_
↳logits=True))
logdir = "testlog"
epochs = 2

if pathlib.Path(logdir).exists():
    shutil.rmtree(logdir)

metrics = [
    ClassifierMetric(tf.metrics.Accuracy()),
    ClassifierMetric(tf.metrics.BinaryAccuracy()),
]

trainer = ClassifierTrainer(model=model,
                            optimizer=optimizer,
                            loss=loss,
                            epochs=epochs,
                            metrics=metrics,
                            logdir=logdir)
train, validation = toy_dataset(), toy_dataset()
trainer(train, validation)

shutil.rmtree(logdir)

```

```

Initializing checkpoint.
Starting epoch 1.
[500] Saved checkpoint: testlog/ckpts/ckpt-1
Epoch 1 completed.
Starting epoch 2.
[1000] Saved checkpoint: testlog/ckpts/ckpt-2
Epoch 2 completed.
Training finished after 2 epochs.

```

build_and_restore_models (*dataset*)

Build and restore a Subclassed model by firstly calling it on some data.

train_step

Perform the training step using the distribution strategy.

call (*training_set*, *validation_set*, *log_freq=10*, *measure_performance_freq=10*)

Start the training.

Parameters

- **training_set** (*tf.data.Dataset*) – Training dataset.
- **validation_set** (*tf.data.Dataset*) – Validation dataset.

- **log_freq** (*int*) – Specifies how many steps to run before logging the losses, e.g. *log_frequency=10* logs every 10 steps of training. Pass *log_frequency<=0* in case you don't want to log.
- **measure_performance_freq** (*int*) – Specifies how many steps to run before measuring the performance, e.g. *measure_performance_freq=10* measures performance every 10 steps of training. Pass *measure_performance_freq<=0* in case you don't want to measure performance.

train_step (*features, labels*)
Train step.

Parameters

- **features** – Input features.
- **labels** – The labels.

Returns Loss value.

```
class ashpy.trainers.classifier.ClassifierTrainer(model, optimizer, loss, epochs,
                                                metrics=None, callbacks=None,
                                                logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log'),
                                                global_step=None)
```

Bases: *ashpy.trainers.trainer.Trainer*

ClassifierTrainer provide the standard training loop for a classifier.

```
__init__(model, optimizer, loss, epochs, metrics=None, callbacks=None,
          logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log'),
          global_step=None)
```

Instantiate the *ClassifierTrainer* trainer.

Parameters

- **model** (*tf.keras.Model*) – A *tf.keras.Model* model.
- **optimizer** (*tf.optimizers.Optimizer*) – A *tf.optimizers.Optimizer*.
- **loss** (*ashpy.losses.classifier.ClassifierLoss*) – A loss function built following *ashpy.executors`*.
- **epochs** (*int*) – Number of training epochs.
- **metrics** (*Union[Tuple[Metric], List[Metric], None]*) – (Tuple/List): Tuple/List of *ashpy.metrics.metric.Metric* to measure on training and validation data.
- **callbacks** (*List*) – List of *ashpy.callbacks.callback.Callback* to call on events
- **logdir** (*str*) – Checkpoint and log directory.
- **global_step** (Optional[*py:class:tf.Variable*]) – *tf.Variable* that keeps track of the training steps.

Examples

```
def toy_dataset():
    inputs = tf.expand_dims(tf.range(1, 1000.0), -1)
    labels = tf.expand_dims(
```

(continues on next page)

(continued from previous page)

```

        [1 if tf.equal(tf.math.mod(tf.squeeze(i), 2), 0) else 0 for i in
↪inputs], -1
    )
    return tf.data.Dataset.from_tensor_slices((inputs, labels)).shuffle(10).
↪batch(2)

model = tf.keras.Sequential(
    [tf.keras.layers.Dense(10, activation=tf.nn.sigmoid), tf.keras.layers.
↪Dense(2)]
)
optimizer = tf.optimizers.Adam(1e-3)

loss = ClassifierLoss(tf.losses.SparseCategoricalCrossentropy(from_
↪logits=True))
logdir = "testlog"
epochs = 2

if pathlib.Path(logdir).exists():
    shutil.rmtree(logdir)

metrics = [
    ClassifierMetric(tf.metrics.Accuracy()),
    ClassifierMetric(tf.metrics.BinaryAccuracy()),
]

trainer = ClassifierTrainer(model=model,
                            optimizer=optimizer,
                            loss=loss,
                            epochs=epochs,
                            metrics=metrics,
                            logdir=logdir)
train, validation = toy_dataset(), toy_dataset()
trainer(train, validation)

shutil.rmtree(logdir)

```

```

Initializing checkpoint.
Starting epoch 1.
[500] Saved checkpoint: testlog/ckpts/ckpt-1
Epoch 1 completed.
Starting epoch 2.
[1000] Saved checkpoint: testlog/ckpts/ckpt-2
Epoch 2 completed.
Training finished after 2 epochs.

```

_build_and_restore_models (*dataset*)

Build and restore a Subclassed model by firstly calling it on some data.

_train_step

Perform the training step using the distribution strategy.

call (*training_set*, *validation_set*, *log_freq=10*, *measure_performance_freq=10*)

Start the training.

Parameters

- **training_set** (*tf.data.Dataset*) – Training dataset.

- **validation_set** (`tf.data.Dataset`) – Validation dataset.
- **log_freq** (`int`) – Specifies how many steps to run before logging the losses, e.g. `log_frequency=10` logs every 10 steps of training. Pass `log_frequency<=0` in case you don't want to log.
- **measure_performance_freq** (`int`) – Specifies how many steps to run before measuring the performance, e.g. `measure_performance_freq=10` measures performance every 10 steps of training. Pass `measure_performance_freq<=0` in case you don't want to measure performance.

train_step (*features, labels*)

Train step.

Parameters

- **features** – Input features.
- **labels** – The labels.

Returns Loss value.

4.11.6 gan

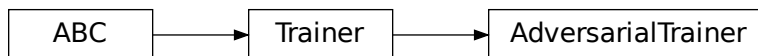
Collection of GANs trainers.

Classes

<code>AdversarialTrainer</code>	Primitive Trainer for GANs subclassed from <code>ashpy.trainers.Trainer</code> .
<code>EncoderTrainer</code>	Primitive Trainer for GANs using an Encoder sub-network.

AdversarialTrainer

Inheritance Diagram



```

class ashpy.trainers.gan.AdversarialTrainer(generator, discriminator, genera-
tor_optimizer, discriminator_optimizer,
generator_loss, discriminator_loss,
epochs, metrics=None, callbacks=None,
logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/
log_eval_mode=<LogEvalMode.TEST: 1>,
global_step=None)
  
```

Bases: `ashpy.trainers.trainer.Trainer`

Primitive Trainer for GANs subclassed from `ashpy.trainers.Trainer`.

Examples

```
import shutil
import operator

generator = models.gans.ConvGenerator(
    layer_spec_input_res=(7, 7),
    layer_spec_target_res=(28, 28),
    kernel_size=(5, 5),
    initial_filters=32,
    filters_cap=16,
    channels=1,
)

discriminator = models.gans.ConvDiscriminator(
    layer_spec_input_res=(28, 28),
    layer_spec_target_res=(7, 7),
    kernel_size=(5, 5),
    initial_filters=16,
    filters_cap=32,
    output_shape=1,
)

# Losses
generator_bce = losses.gan.GeneratorBCE()
minmax = losses.gan.DiscriminatorMinMax()

# Real data
batch_size = 2
mnist_x, mnist_y = tf.zeros((100, 28, 28)), tf.zeros((100,))

# Trainer
epochs = 2
logdir = "testlog/adversarial"
metrics = [
    metrics.gan.InceptionScore(
        # Fake inception model
        models.gans.ConvDiscriminator(
            layer_spec_input_res=(299, 299),
            layer_spec_target_res=(7, 7),
            kernel_size=(5, 5),
            initial_filters=16,
            filters_cap=32,
            output_shape=10,
        ),
        model_selection_operator=operator.gt,
    )
]
trainer = trainers.gan.AdversarialTrainer(
    generator=generator,
    discriminator=discriminator,
    generator_optimizer=tf.optimizers.Adam(1e-4),
    discriminator_optimizer=tf.optimizers.Adam(1e-4),
    generator_loss=generator_bce,
    discriminator_loss=minmax,
```

(continues on next page)

(continued from previous page)

```

    epochs=epochs,
    metrics=metrics,
    logdir=logdir,
)

# take only 2 samples to speed up tests
real_data = (
    tf.data.Dataset.from_tensor_slices(
        (tf.expand_dims(mnist_x, -1), tf.expand_dims(mnist_y, -1))).take(batch_size)
        .batch(batch_size)
        .prefetch(1)
    )

# Add noise in the same dataset, just by mapping.
# The return type of the dataset must be: tuple(tuple(a,b), noise)
dataset = real_data.map(
    lambda x, y: ((x, y), tf.random.normal(shape=(batch_size, 100)))
)

trainer(dataset)
shutil.rmtree(logdir)

```

```

Initializing checkpoint.
Starting epoch 1.
[1] Saved checkpoint: testlog/adversarial/ckpts/ckpt-1
Epoch 1 completed.
Starting epoch 2.
[2] Saved checkpoint: testlog/adversarial/ckpts/ckpt-2
Epoch 2 completed.
Training finished after 2 epochs.

```

Methods

<code>__init__(generator, discriminator, ...[, ...])</code>	Instantiate a <i>AdversarialTrainer</i> .
<code>call(dataset[, log_freq, ...])</code>	Perform the adversarial training.
<code>train_step(real_xy, g_inputs)</code>	Train step for the <i>AdversarialTrainer</i> .

Attributes

<code>ckpt_id_callbacks</code>	
<code>ckpt_id_discriminator</code>	
<code>ckpt_id_generator</code>	
<code>ckpt_id_global_step</code>	
<code>ckpt_id_optimizer_discriminator</code>	
<code>ckpt_id_optimizer_generator</code>	
<code>ckpt_id_steps_per_epoch</code>	
<code>context</code>	Return the training context.

```
__init__(generator, discriminator, generator_optimizer, discriminator_optimizer, gen-
erator_loss, discriminator_loss, epochs, metrics=None, callbacks=None,
logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log'),
log_eval_mode=<LogEvalMode.TEST: 1>, global_step=None)
Instantiate a AdversarialTrainer.
```

Parameters

- **generator** (`tf.keras.Model`) – A `tf.keras.Model` describing the Generator part of a GAN.
- **discriminator** (`tf.keras.Model`) – A `tf.keras.Model` describing the Discriminator part of a GAN.
- **generator_optimizer** (`tf.optimizers.Optimizer`) – A `tf.optimizers` to use for the Generator.
- **discriminator_optimizer** (`tf.optimizers.Optimizer`) – A `tf.optimizers` to use for the Discriminator.
- **generator_loss** (`ashpy.losses.executor.Executor`) – A ash Executor to compute the loss of the Generator.
- **discriminator_loss** (`ashpy.losses.executor.Executor`) – A ash Executor to compute the loss of the Discriminator.
- **epochs** (`int`) – number of training epochs.
- **metrics** (`Union[Tuple[Metric], List[Metric], None]`) – (List): list of `ashpy.metrics.Metric` to measure on training and validation data.
- **callbacks** (`List`) – list of `ashpy.callbacks.Callback` to measure on training and validation data.
- **logdir** (`Union[Path, str]`) – checkpoint and log directory.
- **log_eval_mode** (`LogEvalMode`) – models' mode to use when evaluating and logging.
- **global_step** (`Optional[tf.Variable]`) – `tf.Variable` that keeps track of the training steps.

Returns `None`

`_build_and_restore_models(dataset)`

Build and restore a Subclassed model by firstly calling it on some data.

`_train_step`

Training step with the distribution strategy.

`call(dataset, log_freq=10, measure_performance_freq=10)`

Perform the adversarial training.

Parameters

- **dataset** (`tf.data.Dataset`) – The adversarial training dataset.
- **log_freq** (`int`) – Specifies how many steps to run before logging the losses, e.g. `log_frequency=10` logs every 10 steps of training. Pass `log_frequency<=0` in case you don't want to log.
- **measure_performance_freq** (`int`) – Specifies how many steps to run before measuring the performance, e.g. `measure_performance_freq=10` measures performance every 10 steps of training. Pass `measure_performance_freq<=0` in case you don't want to measure

performance.

train_step (*real_xy*, *g_inputs*)

Train step for the AdversarialTrainer.

Parameters

- **real_xy** – input batch as extracted from the input dataset. (features, label) pair.
- **g_inputs** – batch of generator_input as generated from the input dataset.

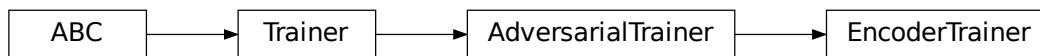
Returns

d_loss, *g_loss*, *fake* –

discriminator, generator loss values. fake is the generator output.

EncoderTrainer

Inheritance Diagram



```

class ashpy.trainers.gan.EncoderTrainer(generator, discriminator, encoder, generator_optimizer, discriminator_optimizer, encoder_optimizer, generator_loss, discriminator_loss, encoder_loss, epochs, metrics=None, callbacks=None, logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/conda/envs/ashpy-0.4.0/lib/python3.7/site-packages/ashpy/trainers/gan'), log_eval_mode=<LogEvalMode.TEST: 1>, global_step=None)
  
```

Bases: *ashpy.trainers.gan.AdversarialTrainer*

Primitive Trainer for GANs using an Encoder sub-network.

The implementation is thought to be used with the BCE losses. To use another loss function consider subclassing the model and overriding the train_step method.

Examples

```

from pathlib import Path

import shutil
import operator

def real_gen():
    label = 0
    for _ in tf.range(100):
        yield ((10.0,), (label,))
  
```

(continues on next page)

(continued from previous page)

```

latent_dim = 100

generator = tf.keras.Sequential([tf.keras.layers.Dense(1)])

left_input = tf.keras.layers.Input(shape=(1,))
left = tf.keras.layers.Dense(10, activation=tf.nn.elu)(left_input)

right_input = tf.keras.layers.Input(shape=(latent_dim,))
right = tf.keras.layers.Dense(10, activation=tf.nn.elu)(right_input)

net = tf.keras.layers.Concatenate()([left, right])
out = tf.keras.layers.Dense(1)(net)

discriminator = tf.keras.Model(inputs=[left_input, right_input], outputs=[out])

encoder = tf.keras.Sequential([tf.keras.layers.Dense(latent_dim)])

# Losses
generator_bce = losses.gan.GeneratorBCE()
encoder_bce = losses.gan.EncoderBCE()
minmax = losses.gan.DiscriminatorMinMax()

epochs = 2

# Fake pre-trained classifier
num_classes = 1
classifier = tf.keras.Sequential(
    [tf.keras.layers.Dense(10), tf.keras.layers.Dense(num_classes)]
)

logdir = Path("testlog") / "adversarial_encoder"

if logdir.exists():
    shutil.rmtree(logdir)

metrics = [metrics.gan.EncodingAccuracy(classifier)]

trainer = trainers.gan.EncoderTrainer(
    generator=generator,
    discriminator=discriminator,
    encoder=encoder,
    discriminator_optimizer=tf.optimizers.Adam(1e-4),
    generator_optimizer=tf.optimizers.Adam(1e-5),
    encoder_optimizer=tf.optimizers.Adam(1e-6),
    generator_loss=generator_bce,
    discriminator_loss=minmax,
    encoder_loss=encoder_bce,
    epochs=epochs,
    metrics=metrics,
    logdir=logdir,
)

batch_size = 10
discriminator_input = tf.data.Dataset.from_generator(
    real_gen, (tf.float32, tf.int64), ((1), (1))
).batch(batch_size)

```

(continues on next page)

(continued from previous page)

```
dataset = discriminator_input.map(
    lambda x, y: ((x, y), tf.random.normal(shape=(batch_size, latent_dim)))
)

trainer(dataset)

shutil.rmtree(logdir)
```

```
Initializing checkpoint.
Starting epoch 1.
[10] Saved checkpoint: testlog/adversarial_encoder/ckpts/ckpt-1
Epoch 1 completed.
Starting epoch 2.
[20] Saved checkpoint: testlog/adversarial_encoder/ckpts/ckpt-2
Epoch 2 completed.
Training finished after 2 epochs.
```

Methods

<code>__init__(generator, discriminator, encoder, ...)</code>	Instantiate a <i>EncoderTrainer</i> .
<code>call(dataset[, log_freq, ...])</code>	Perform the adversarial training.
<code>train_step(real_xy, g_inputs)</code>	Adversarial training step.

Attributes

<code>ckpt_id_callbacks</code>	
<code>ckpt_id_discriminator</code>	
<code>ckpt_id_encoder</code>	
<code>ckpt_id_generator</code>	
<code>ckpt_id_global_step</code>	
<code>ckpt_id_optimizer_discriminator</code>	
<code>ckpt_id_optimizer_encoder</code>	
<code>ckpt_id_optimizer_generator</code>	
<code>ckpt_id_steps_per_epoch</code>	
<code>context</code>	Return the training context.

```
__init__(generator, discriminator, encoder, generator_optimizer, discrim-
inator_optimizer, encoder_optimizer, generator_loss, discrimina-
tor_loss, encoder_loss, epochs, metrics=None, callbacks=None,
logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log'),
log_eval_mode=<LogEvalMode.TEST: 1>, global_step=None)
Instantiate a EncoderTrainer.
```

Parameters

- **generator** (`tf.keras.Model`) – A `tf.keras.Model` describing the Generator part of a GAN.
- **discriminator** (`tf.keras.Model`) – A `tf.keras.Model` describing the Discriminator part of a GAN.

- **encoder** (`tf.keras.Model`) – A `tf.keras.Model` describing the Encoder part of a GAN.
- **generator_optimizer** (`tf.optimizers.Optimizer`) – A `tf.optimizers` to use for the Generator.
- **discriminator_optimizer** (`tf.optimizers.Optimizer`) – A `tf.optimizers` to use for the Discriminator.
- **encoder_optimizer** (`tf.optimizers.Optimizer`) – A `tf.optimizers` to use for the Encoder.
- **generator_loss** (`ashpy.losses.executor.Executor`) – A ash Executor to compute the loss of the Generator.
- **discriminator_loss** (`ashpy.losses.executor.Executor`) – A ash Executor to compute the loss of the Discriminator.
- **encoder_loss** (`ashpy.losses.executor.Executor`) – A ash Executor to compute the loss of the Discriminator.
- **epochs** (`int`) – number of training epochs.
- **metrics** (`Optional[List[Metric]]`) – (List): list of `ashpy.metrics.Metric` to measure on training and validation data.
- **callbacks** (`List`) – List of `ashpy.callbacks.Callback` to call on events
- **logdir** (`Union[Path, str]`) – checkpoint and log directory.
- **log_eval_mode** (`ashpy.modes.LogEvalMode`) – models' mode to use when evaluating and logging.
- **global_step** (`Optional[tf.Variable]`) – `tf.Variable` that keeps track of the training steps.

`_build_and_restore_models` (`dataset`)

Build and restore a Subclassed model by firstly calling it on some data.

`_train_step`

Perform the training step using the distribution strategy.

`call` (`dataset, log_freq=10, measure_performance_freq=10`)

Perform the adversarial training.

Parameters

- **dataset** (`tf.data.Dataset`) – The adversarial training dataset.
- **log_freq** (`int`) – Specifies how many steps to run before logging the losses, e.g. `log_frequency=10` logs every 10 steps of training. Pass `log_frequency<=0` in case you don't want to log.
- **measure_performance_freq** (`int`) – Specifies how many steps to run before measuring the performance, e.g. `measure_performance_freq=10` measures performance every 10 steps of training. Pass `measure_performance_freq<=0` in case you don't want to measure performance.

`train_step` (`real_xy, g_inputs`)

Adversarial training step.

Parameters

- **real_xy** – input batch as extracted from the discriminator input dataset. (features, label) pair

- **g_inputs** – batch of noise as generated by the generator input dataset.

Returns *d_loss, g_loss, e_loss* – discriminator, generator, encoder loss values.

```
class ashpy.trainers.gan.AdversarialTrainer(generator, discriminator, genera-
tor_optimizer, discriminator_optimizer,
generator_loss, discriminator_loss,
epochs, metrics=None, callbacks=None,
logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/
log_eval_mode=<LogEvalMode.TEST: 1>,
global_step=None)
```

Bases: *ashpy.trainers.trainer.Trainer*

Primitive Trainer for GANs subclassed from *ashpy.trainers.Trainer*.

Examples

```
import shutil
import operator

generator = models.gans.ConvGenerator(
    layer_spec_input_res=(7, 7),
    layer_spec_target_res=(28, 28),
    kernel_size=(5, 5),
    initial_filters=32,
    filters_cap=16,
    channels=1,
)

discriminator = models.gans.ConvDiscriminator(
    layer_spec_input_res=(28, 28),
    layer_spec_target_res=(7, 7),
    kernel_size=(5, 5),
    initial_filters=16,
    filters_cap=32,
    output_shape=1,
)

# Losses
generator_bce = losses.gan.GeneratorBCE()
minmax = losses.gan.DiscriminatorMinMax()

# Real data
batch_size = 2
mnist_x, mnist_y = tf.zeros((100,28,28)), tf.zeros((100,))

# Trainer
epochs = 2
logdir = "testlog/adversarial"
metrics = [
    metrics.gan.InceptionScore(
        # Fake inception model
        models.gans.ConvDiscriminator(
            layer_spec_input_res=(299, 299),
            layer_spec_target_res=(7, 7),
            kernel_size=(5, 5),
            initial_filters=16,
```

(continues on next page)

(continued from previous page)

```

        filters_cap=32,
        output_shape=10,
    ),
    model_selection_operator=operator.gt,
)
]
trainer = trainers.gan.AdversarialTrainer(
    generator=generator,
    discriminator=discriminator,
    generator_optimizer=tf.optimizers.Adam(1e-4),
    discriminator_optimizer=tf.optimizers.Adam(1e-4),
    generator_loss=generator_bce,
    discriminator_loss=minmax,
    epochs=epochs,
    metrics=metrics,
    logdir=logdir,
)

# take only 2 samples to speed up tests
real_data = (
    tf.data.Dataset.from_tensor_slices(
        (tf.expand_dims(mnist_x, -1), tf.expand_dims(mnist_y, -1))).take(batch_size)
        .batch(batch_size)
        .prefetch(1)
)

# Add noise in the same dataset, just by mapping.
# The return type of the dataset must be: tuple(tuple(a,b), noise)
dataset = real_data.map(
    lambda x, y: ((x, y), tf.random.normal(shape=(batch_size, 100)))
)

trainer(dataset)
shutil.rmtree(logdir)

```

```

Initializing checkpoint.
Starting epoch 1.
[1] Saved checkpoint: testlog/adversarial/ckpts/ckpt-1
Epoch 1 completed.
Starting epoch 2.
[2] Saved checkpoint: testlog/adversarial/ckpts/ckpt-2
Epoch 2 completed.
Training finished after 2 epochs.

```

`__init__`(*generator*, *discriminator*, *generator_optimizer*, *discriminator_optimizer*, *generator_loss*, *discriminator_loss*, *epochs*, *metrics=None*, *callbacks=None*, *logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log')*, *log_eval_mode=<LogEvalMode.TEST: 1>*, *global_step=None*)

Instantiate a `AdversarialTrainer`.

Parameters

- **generator** (`tf.keras.Model`) – A `tf.keras.Model` describing the Generator part of a GAN.
- **discriminator** (`tf.keras.Model`) – A `tf.keras.Model` describing the Discriminator part of a GAN.

- **generator_optimizer** (`tf.optimizers.Optimizer`) – A `tf.optimizers` to use for the Generator.
- **discriminator_optimizer** (`tf.optimizers.Optimizer`) – A `tf.optimizers` to use for the Discriminator.
- **generator_loss** (`ashpy.losses.executor.Executor`) – A `ash` `Executor` to compute the loss of the Generator.
- **discriminator_loss** (`ashpy.losses.executor.Executor`) – A `ash` `Executor` to compute the loss of the Discriminator.
- **epochs** (`int`) – number of training epochs.
- **metrics** (`Union[Tuple[Metric], List[Metric], None]`) – (List): list of `ashpy.metrics.Metric` to measure on training and validation data.
- **callbacks** (`List`) – list of `ashpy.callbacks.Callback` to measure on training and validation data.
- **logdir** (`Union[Path, str]`) – checkpoint and log directory.
- **log_eval_mode** (`LogEvalMode`) – models' mode to use when evaluating and logging.
- **global_step** (`Optional[tf.Variable]`) – `tf.Variable` that keeps track of the training steps.

Returns `None`

`_build_and_restore_models` (`dataset`)

Build and restore a Subclassed model by firstly calling it on some data.

`_train_step`

Training step with the distribution strategy.

`call` (`dataset`, `log_freq=10`, `measure_performance_freq=10`)

Perform the adversarial training.

Parameters

- **dataset** (`tf.data.Dataset`) – The adversarial training dataset.
- **log_freq** (`int`) – Specifies how many steps to run before logging the losses, e.g. `log_frequency=10` logs every 10 steps of training. Pass `log_frequency<=0` in case you don't want to log.
- **measure_performance_freq** (`int`) – Specifies how many steps to run before measuring the performance, e.g. `measure_performance_freq=10` measures performance every 10 steps of training. Pass `measure_performance_freq<=0` in case you don't want to measure performance.

`train_step` (`real_xy`, `g_inputs`)

Train step for the `AdversarialTrainer`.

Parameters

- **real_xy** – input batch as extracted from the input dataset. (features, label) pair.
- **g_inputs** – batch of generator_input as generated from the input dataset.

Returns

`d_loss`, `g_loss`, `fake` –

discriminator, generator loss values. fake is the generator output.

```
class ashpy.trainers.gan.EncoderTrainer(generator, discriminator, encoder, gener-
                                         ator_optimizer, discriminator_optimizer,
                                         encoder_optimizer, generator_loss,
                                         discriminator_loss, encoder_loss,
                                         epochs, metrics=None, callbacks=None,
                                         logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/
log_eval_mode=<LogEvalMode.TEST: 1>,
                                         global_step=None)
```

Bases: `ashpy.trainers.gan.AdversarialTrainer`

Primitive Trainer for GANs using an Encoder sub-network.

The implementation is thought to be used with the BCE losses. To use another loss function consider subclassing the model and overriding the `train_step` method.

Examples

```
from pathlib import Path

import shutil
import operator

def real_gen():
    label = 0
    for _ in tf.range(100):
        yield ((10.0,), (label,))

latent_dim = 100

generator = tf.keras.Sequential([tf.keras.layers.Dense(1)])

left_input = tf.keras.layers.Input(shape=(1,))
left = tf.keras.layers.Dense(10, activation=tf.nn.elu)(left_input)

right_input = tf.keras.layers.Input(shape=(latent_dim,))
right = tf.keras.layers.Dense(10, activation=tf.nn.elu)(right_input)

net = tf.keras.layers.Concatenate()([left, right])
out = tf.keras.layers.Dense(1)(net)

discriminator = tf.keras.Model(inputs=[left_input, right_input], outputs=[out])

encoder = tf.keras.Sequential([tf.keras.layers.Dense(latent_dim)])

# Losses
generator_bce = losses.gan.GeneratorBCE()
encoder_bce = losses.gan.EncoderBCE()
minmax = losses.gan.DiscriminatorMinMax()

epochs = 2

# Fake pre-trained classifier
num_classes = 1
classifier = tf.keras.Sequential(
    [tf.keras.layers.Dense(10), tf.keras.layers.Dense(num_classes)]
```

(continues on next page)

(continued from previous page)

```

)

logdir = Path("testlog") / "adversarial_encoder"

if logdir.exists():
    shutil.rmtree(logdir)

metrics = [metrics.gan.EncodingAccuracy(classifier)]

trainer = trainers.gan.EncoderTrainer(
    generator=generator,
    discriminator=discriminator,
    encoder=encoder,
    discriminator_optimizer=tf.optimizers.Adam(1e-4),
    generator_optimizer=tf.optimizers.Adam(1e-5),
    encoder_optimizer=tf.optimizers.Adam(1e-6),
    generator_loss=generator_bce,
    discriminator_loss=minmax,
    encoder_loss=encoder_bce,
    epochs=epochs,
    metrics=metrics,
    logdir=logdir,
)

batch_size = 10
discriminator_input = tf.data.Dataset.from_generator(
    real_gen, (tf.float32, tf.int64), ((1), (1))
).batch(batch_size)

dataset = discriminator_input.map(
    lambda x, y: ((x, y), tf.random.normal(shape=(batch_size, latent_dim)))
)

trainer(dataset)

shutil.rmtree(logdir)

```

```

Initializing checkpoint.
Starting epoch 1.
[10] Saved checkpoint: testlog/adversarial_encoder/ckpts/ckpt-1
Epoch 1 completed.
Starting epoch 2.
[20] Saved checkpoint: testlog/adversarial_encoder/ckpts/ckpt-2
Epoch 2 completed.
Training finished after 2 epochs.

```

__init__(*generator*, *discriminator*, *encoder*, *generator_optimizer*, *discriminator_optimizer*, *encoder_optimizer*, *generator_loss*, *discriminator_loss*, *encoder_loss*, *epochs*, *metrics=None*, *callbacks=None*, *logdir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.4.0/docs/source/log')*, *log_eval_mode=<LogEvalMode.TEST: 1>*, *global_step=None*)

Instantiate a [EncoderTrainer](#).

Parameters

- **generator** (`tf.keras.Model`) – A `tf.keras.Model` describing the Generator part of a GAN.

- **discriminator** (`tf.keras.Model`) – A `tf.keras.Model` describing the Discriminator part of a GAN.
- **encoder** (`tf.keras.Model`) – A `tf.keras.Model` describing the Encoder part of a GAN.
- **generator_optimizer** (`tf.optimizers.Optimizer`) – A `tf.optimizers` to use for the Generator.
- **discriminator_optimizer** (`tf.optimizers.Optimizer`) – A `tf.optimizers` to use for the Discriminator.
- **encoder_optimizer** (`tf.optimizers.Optimizer`) – A `tf.optimizers` to use for the Encoder.
- **generator_loss** (`ashpy.losses.executor.Executor`) – A ash Executor to compute the loss of the Generator.
- **discriminator_loss** (`ashpy.losses.executor.Executor`) – A ash Executor to compute the loss of the Discriminator.
- **encoder_loss** (`ashpy.losses.executor.Executor`) – A ash Executor to compute the loss of the Discriminator.
- **epochs** (`int`) – number of training epochs.
- **metrics** (`Optional[List[Metric]]`) – (List): list of `ashpy.metrics.Metric` to measure on training and validation data.
- **callbacks** (`List`) – List of `ashpy.callbacks.Callback` to call on events
- **logdir** (`Union[Path, str]`) – checkpoint and log directory.
- **log_eval_mode** (`ashpy.modes.LogEvalMode`) – models' mode to use when evaluating and logging.
- **global_step** (`Optional[tf.Variable]`) – `tf.Variable` that keeps track of the training steps.

`_build_and_restore_models` (`dataset`)

Build and restore a Subclassed model by firstly calling it on some data.

`_train_step`

Perform the training step using the distribution strategy.

`call` (`dataset`, `log_freq=10`, `measure_performance_freq=10`)

Perform the adversarial training.

Parameters

- **dataset** (`tf.data.Dataset`) – The adversarial training dataset.
- **log_freq** (`int`) – Specifies how many steps to run before logging the losses, e.g. `log_frequency=10` logs every 10 steps of training. Pass `log_frequency<=0` in case you don't want to log.
- **measure_performance_freq** (`int`) – Specifies how many steps to run before measuring the performance, e.g. `measure_performance_freq=10` measures performance every 10 steps of training. Pass `measure_performance_freq<=0` in case you don't want to measure performance.

`train_step` (`real_xy`, `g_inputs`)

Adversarial training step.

Parameters

- **real_xy** – input batch as extracted from the discriminator input dataset. (features, label) pair
- **g_inputs** – batch of noise as generated by the generator input dataset.

Returns *d_loss*, *g_loss*, *e_loss* – discriminator, generator, encoder loss values.

The two main concepts of AshPy internals are *Context* and *Executor*.

5.1 Context

A *Context* is an object that contains all the needed information. Here needed depends on the application. In AshPy the *Context* concept links a generic training loop with the loss function calculation and the model evaluation. A *Context* is a useful class in which all the models, metrics, dataset and mode of your network are set. Passing the context around means that you can any time access to all what you need in order to perform any type of computation.

In AshPy we have (until now) three types of contexts:

- *Classifier Context*
- *GAN Context*
- *GANEncoder Context*

5.1.1 Classifier Context

The *ClassifierContext* is very simple, it contains only:

- `classifier_model`
- `loss`
- `dataset`
- `metrics`
- `log_eval_mode`
- `global_step`
- `ckpt`

In this way the loss function (*Executor*) can use the context in order to get the model and the needed information in order to correctly feed the model.

5.1.2 GAN Context

The basic *GANContext* is composed by:

- dataset
- generator_model
- discriminator_model
- generator_loss
- discriminator_loss
- metrics
- log_eval_mode
- global_step
- ckpt

As we can see we have all information needed to define our training and evaluation loop.

5.1.3 GANEncoder Context

The *GANEncoderContext* extends the *GANContext*, contains all the information of the base class plus:

- Encoder Model
- Encoder Loss

5.2 Executor

The *Executor* is the main concept behind the loss function implementation in AshPy. An Executor is a class that helps in order to better generalize a training loop. With an Executor you can construct, for example, a custom loss function and put every computation you need inside it. You should define a `call` function inside your class and decorate it with `@Executor.reduce` header, if needed.

Inside the `call` function you can take advantage of a context.

Executors can be summed up, subtracted and multiplied by scalars.

An executor takes also care of the distribution strategy by reducing appropriately the loss (see [Tensorflow Guide](#)).

In this example we will see the implementation of the Generator Binary CrossEntropy loss.

The `__init__` method is straightforward, we need only to instantiate `tf.losses.BinaryCrossentropy` object and then we pass it to our parent:

```
class GeneratorBCE(GANExecutor):

    def __init__(self, from_logits=True):
        self.name = "GeneratorBCE"
        # call super passing the BinaryCrossentropy as function
        super().__init__(tf.losses.BinaryCrossentropy(from_logits=from_logits))
```


Then we need to implement the call function respecting the signature:

```
def call(self, context, *, fake, condition, training, **kwargs):

    # we need a function that gives us the correct inputs given the discriminator_
    ↪model
    fake_inputs = self.get_discriminator_inputs(
        context=context, fake_or_real=fake, condition=condition, training=training
    )

    # get the discriminator predictions from the discriminator model
    d_fake = context.discriminator_model(fake_inputs, training=training)

    # get the target prediction for the generator
    value = self._fn(tf.ones_like(d_fake), d_fake)

    # mean everything
    return tf.reduce_mean(value)
```

The function `get_discriminator_inputs()` returns the correct discriminator inputs using the context. The discriminator input can be the output of the generator (unconditioned case) or the output of the generator together with the condition (conditioned case).

The `call()` uses the discriminator model inside the context in order to obtain the output of the discriminator when evaluated in the `fake_inputs`.

After that the `self._fn()` (BinaryCrossentropy) is used to get the value of the loss. This loss is then averaged.

In this way the executor computes correctly the loss function.

This is ok if we do not want use our code in a distribution strategy.

If we want to use our executor in a distribution strategy the only modifications are:

```
@Executor.reduce_loss
def call(self, context, *, fake, condition, training, **kwargs):

    # we need a function that gives us the correct inputs given the discriminator_
    ↪model
    fake_inputs = self.get_discriminator_inputs(
        context=context, fake_or_real=fake, condition=condition, training=training
    )

    # get the discriminator predictions from the discriminator model
    d_fake = context.discriminator_model(fake_inputs, training=training)

    # get the target prediction for the generator
    value = self._fn(tf.ones_like(d_fake), d_fake)

    # mean only over the axis 1
    return tf.reduce_mean(value, axis=1)
```

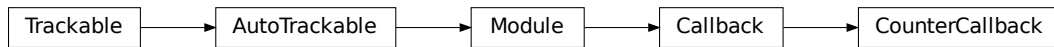
The important things are:

- `Executor.reduce_loss` decoration: uses the `Executor` decorator in order to correctly reduce the loss
- `tf.reduce_mean(value, axis=1)` (last line), we perform only the mean over the axis 1. The output of the `call` function

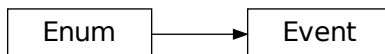
should be a `tf.Tensor` with shape `(N, 1)` or `(N,)`. This is because the decorator performs the mean over the axis 0.

Dependencies Graph

6.1 ashpy.callbacks

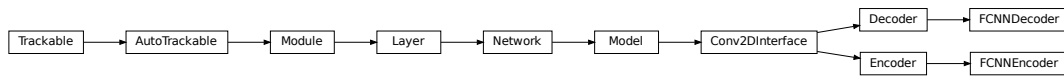


6.1.1 Events



6.2 ashpy.models

6.2.1 Convolutional

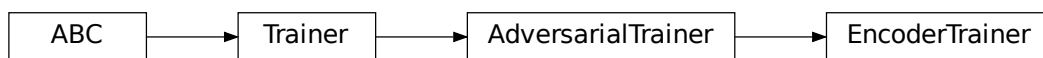


6.2.2 GANs

GANs models are just aliases.

6.3 ashpy.trainers

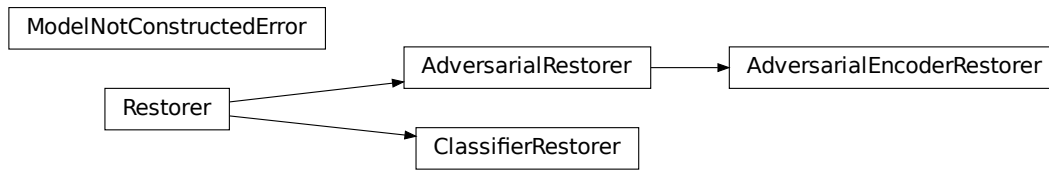
6.3.1 Adversarial



6.3.2 Classifier

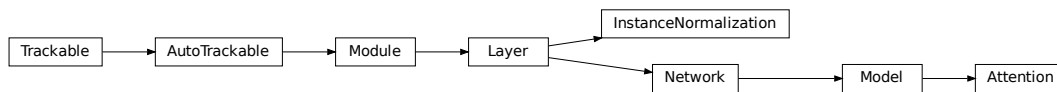


6.4 ashpy.restorers



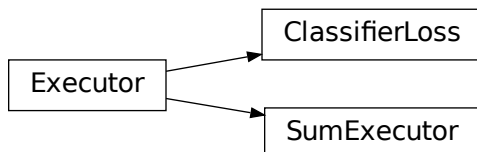
6.5 ashpy.layers

6.5.1 Layers

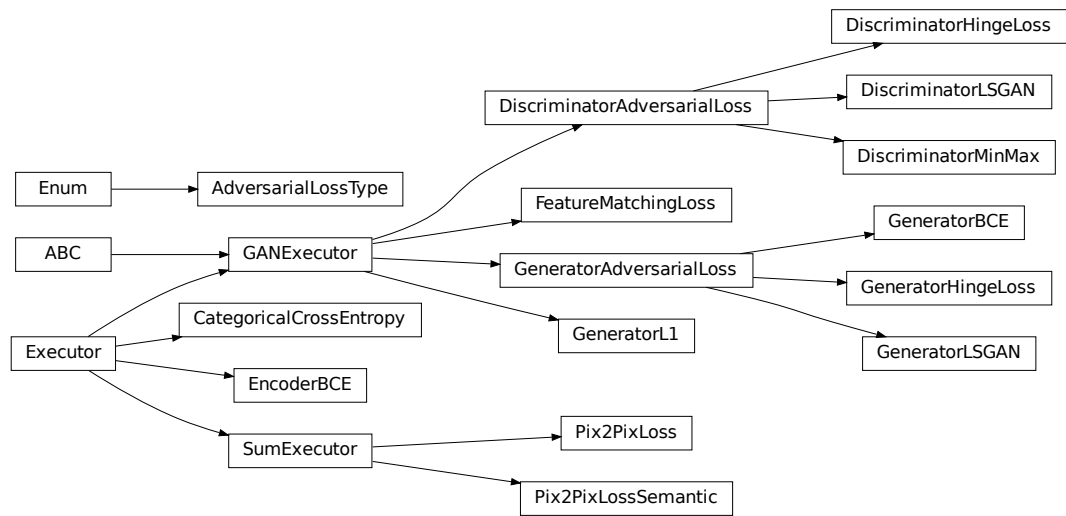


6.6 ashpy.losses

6.6.1 Classifier Losses

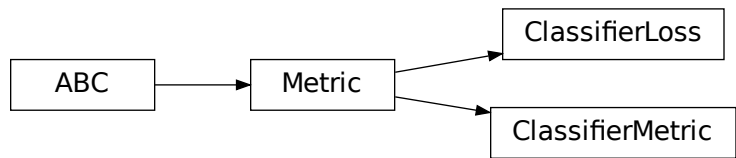


6.6.2 GAN Losses

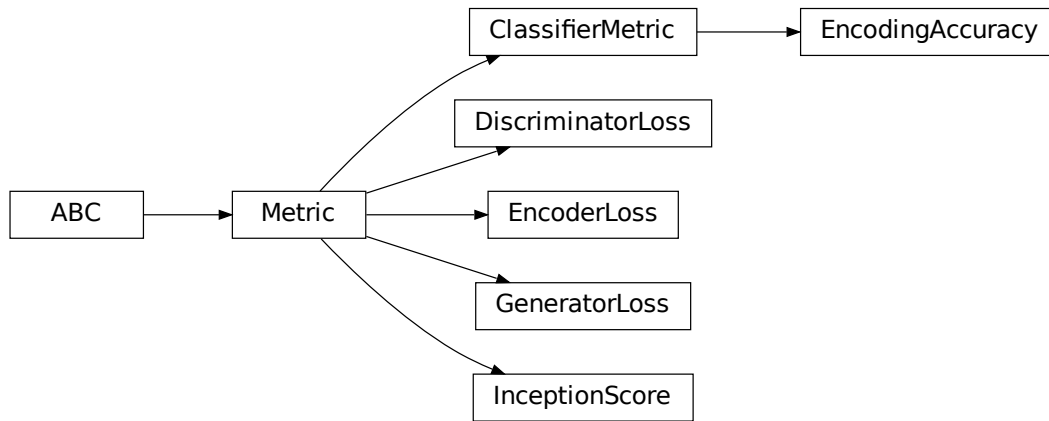


6.7 ashpy.metrics

6.7.1 Classifier Metrics



6.7.2 GAN Metrics



CHAPTER 7

About

AshPy is an open-source project available on [Github](#) under APACHE licence.

The Framework is created and maintained primarily by the ML & CV Lab @ [Zuru Tech](#).

Please contact ml@zuru.tech for doubt, information or suggestions.

CHAPTER 8

Indices and tables

- `genindex`
- `modindex`
- `search`

a

- `ashpy.ashtypes`, 31
- `ashpy.callbacks`, 31
 - `ashpy.callbacks.callback`, 46
 - `ashpy.callbacks.classifier`, 51
 - `ashpy.callbacks.counter_callback`, 53
 - `ashpy.callbacks.events`, 54
 - `ashpy.callbacks.gan`, 56
 - `ashpy.callbacks.save_callback`, 65
- `ashpy.contexts`, 71
 - `ashpy.contexts.classifier`, 82
 - `ashpy.contexts.context`, 79
 - `ashpy.contexts.gan`, 85
- `ashpy.keras`, 91
 - `ashpy.keras.losses`, 91
- `ashpy.layers`, 99
- `ashpy.losses`, 105
 - `ashpy.losses.classifier`, 125
 - `ashpy.losses.executor`, 126
 - `ashpy.losses.gan`, 131
- `ashpy.metrics`, 151
 - `ashpy.metrics.classifier`, 169
 - `ashpy.metrics.gan`, 173
 - `ashpy.metrics.metric`, 184
 - `ashpy.metrics.sliced_wasserstein_metric`, 189
 - `ashpy.metrics.ssim_multiscale`, 195
- `ashpy.models`, 198
 - `ashpy.models.convolutional`, 199
 - `ashpy.models.convolutional.autoencoders`, 233
 - `ashpy.models.convolutional.decoders`, 250
 - `ashpy.models.convolutional.discriminators`, 240
 - `ashpy.models.convolutional.encoders`, 258
 - `ashpy.models.convolutional.interfaces`, 266
 - `ashpy.models.convolutional.pix2pixhd`, 278
 - `ashpy.models.convolutional.unet`, 270
 - `ashpy.models.fc`, 289
 - `ashpy.models.fc.autoencoders`, 297
 - `ashpy.models.fc.decoders`, 300
 - `ashpy.models.fc.encoders`, 302
 - `ashpy.models.fc.interfaces`, 304
 - `ashpy.models.gans`, 306
 - `ashpy.modes`, 309
 - `ashpy.restorers`, 310
 - `ashpy.restorers.classifier`, 320
 - `ashpy.restorers.gan`, 321
 - `ashpy.restorers.restorer`, 316
 - `ashpy.trainers`, 326
 - `ashpy.trainers.classifier`, 343
 - `ashpy.trainers.gan`, 348
 - `ashpy.trainers.trainer`, 337

Symbols

<code>__init__()</code> (<i>ashpy.callbacks.callback.Callback</i> method), 33, 48, 50	<code>__init__()</code> (<i>ashpy.losses.gan.CategoricalCrossEntropy</i> method), 118, 133, 146
<code>__init__()</code> (<i>ashpy.callbacks.classifier.LogClassifierCallback</i> method), 36, 52	<code>__init__()</code> (<i>ashpy.losses.gan.DiscriminatorAdversarialLoss</i> method), 112, 134, 146
<code>__init__()</code> (<i>ashpy.callbacks.counter_callback.CounterCallback</i> method), 35, 53, 54	<code>__init__()</code> (<i>ashpy.losses.gan.DiscriminatorHingeLoss</i> method), 124, 135, 147
<code>__init__()</code> (<i>ashpy.callbacks.gan.LogImageGANCallback</i> method), 40, 58, 63	<code>__init__()</code> (<i>ashpy.losses.gan.DiscriminatorLSGAN</i> method), 123, 136, 147
<code>__init__()</code> (<i>ashpy.callbacks.gan.LogImageGANEncoderCallback</i> method), 42, 61, 64	<code>__init__()</code> (<i>ashpy.losses.gan.DiscriminatorMinMax</i> method), 122, 136, 147
<code>__init__()</code> (<i>ashpy.callbacks.save_callback.SaveCallback</i> method), 44, 66, 69	<code>__init__()</code> (<i>ashpy.losses.gan.EncoderBCE</i> method), 122, 137, 147
<code>__init__()</code> (<i>ashpy.contexts.classifier.ClassifierContext</i> method), 74, 83, 84	<code>__init__()</code> (<i>ashpy.losses.gan.FeatureMatchingLoss</i> method), 118, 138, 148
<code>__init__()</code> (<i>ashpy.contexts.context.Context</i> method), 72, 80, 81	<code>__init__()</code> (<i>ashpy.losses.gan.GeneratorAdversarialLoss</i> method), 111, 140, 149
<code>__init__()</code> (<i>ashpy.contexts.gan.GANContext</i> method), 75, 86, 89	<code>__init__()</code> (<i>ashpy.losses.gan.GeneratorBCE</i> method), 113, 141, 149
<code>__init__()</code> (<i>ashpy.contexts.gan.GANEncoderContext</i> method), 78, 88, 90	<code>__init__()</code> (<i>ashpy.losses.gan.GeneratorHingeLoss</i> method), 116, 142, 149
<code>__init__()</code> (<i>ashpy.keras.losses.DHingeLoss</i> method), 92, 97	<code>__init__()</code> (<i>ashpy.losses.gan.GeneratorL1</i> method), 115, 143, 149
<code>__init__()</code> (<i>ashpy.keras.losses.DLeastSquare</i> method), 93, 97	<code>__init__()</code> (<i>ashpy.losses.gan.GeneratorLSGAN</i> method), 114, 144, 150
<code>__init__()</code> (<i>ashpy.keras.losses.DMinMax</i> method), 94, 98	<code>__init__()</code> (<i>ashpy.losses.gan.Pix2PixLoss</i> method), 119, 145, 150
<code>__init__()</code> (<i>ashpy.keras.losses.GHingeLoss</i> method), 96, 99	<code>__init__()</code> (<i>ashpy.losses.gan.Pix2PixLossSemantic</i> method), 121, 146, 151
<code>__init__()</code> (<i>ashpy.keras.losses.L1</i> method), 97, 99	<code>__init__()</code> (<i>ashpy.metrics.classifier.ClassifierLoss</i> method), 155, 170, 172
<code>__init__()</code> (<i>ashpy.layers.attention.Attention</i> method), 104	<code>__init__()</code> (<i>ashpy.metrics.classifier.ClassifierMetric</i> method), 157, 171, 172
<code>__init__()</code> (<i>ashpy.layers.instance_normalization.InstanceNormalization</i> method), 101	<code>__init__()</code> (<i>ashpy.metrics.gan.DiscriminatorLoss</i> method), 158, 174, 181
<code>__init__()</code> (<i>ashpy.losses.classifier.ClassifierLoss</i> method), 108, 126	<code>__init__()</code> (<i>ashpy.metrics.gan.EncoderLoss</i> method), 161, 176, 181
<code>__init__()</code> (<i>ashpy.losses.executor.Executor</i> method), 106, 127, 129	<code>__init__()</code> (<i>ashpy.metrics.gan.EncodingAccuracy</i> method), 164, 177, 182
<code>__init__()</code> (<i>ashpy.losses.executor.SumExecutor</i> method), 107, 129, 130	<code>__init__()</code> (<i>ashpy.metrics.gan.GeneratorLoss</i> method), 164, 177, 182

method), 160, 178, 182

`__init__()` (ashpy.metrics.gan.InceptionScore method), 162, 180, 183

`__init__()` (ashpy.metrics.metric.Metric method), 153, 185, 187

`__init__()` (ashpy.metrics.sliced_wasserstein_metric.SingleSWD method), 190, 193

`__init__()` (ashpy.metrics.sliced_wasserstein_metric.SlicedWassersteinMetric method), 166, 192, 194

`__init__()` (ashpy.metrics.ssim_multiscale.SSIM_Multiscale method), 168, 196, 197

`__init__()` (ashpy.models.convolutional.autoencoders.Autoencoder method), 214, 235, 239

`__init__()` (ashpy.models.convolutional.autoencoders.FCNNAutoencoder method), 216, 238, 240

`__init__()` (ashpy.models.convolutional.decoders.Decoder method), 204, 253, 256

`__init__()` (ashpy.models.convolutional.decoders.FCNNDDecoder method), 209, 261, 264

`__init__()` (ashpy.models.convolutional.decoders.FCNNDDecoder method), 206, 255, 258

`__init__()` (ashpy.models.convolutional.discriminators.MultiScaleDiscriminator method), 224, 243, 248

`__init__()` (ashpy.models.convolutional.discriminators.PatchDiscriminator method), 227, 246, 249

`__init__()` (ashpy.models.convolutional.encoders.Encoder method), 209, 260, 264

`__init__()` (ashpy.models.convolutional.encoders.FCNNEncoder method), 211, 263, 265

`__init__()` (ashpy.models.convolutional.interfaces.Conv2DInterface method), 200, 267, 269

`__init__()` (ashpy.models.convolutional.pix2pixhd.GlobalGenerator method), 232, 280, 286

`__init__()` (ashpy.models.convolutional.pix2pixhd.LocalEnhancer method), 230, 283, 287

`__init__()` (ashpy.models.convolutional.pix2pixhd.ResNetBlock method), 285, 288

`__init__()` (ashpy.models.convolutional.unet.SUNet method), 221, 272, 276

`__init__()` (ashpy.models.convolutional.unet.UNet method), 219, 274, 277

`__init__()` (ashpy.models.fc.autoencoders.Autoencoder method), 296, 299

`__init__()` (ashpy.models.fc.decoders.Decoder method), 292, 301, 302

`__init__()` (ashpy.models.fc.encoders.Encoder method), 294, 303, 304

`__init__()` (ashpy.models.fc.interfaces.FCInterface method), 290, 306

`__init__()` (ashpy.restorers.Restorer method), 311

`__init__()` (ashpy.restorers.restorer.Restorer method), 317, 319

`__init__()` (ashpy.trainers.AdversarialTrainer method), 332

`__init__()` (ashpy.trainers.EncoderTrainer method), 336

`__init__()` (ashpy.trainers.Trainer method), 327

`__init__()` (ashpy.trainers.classifier.ClassifierTrainer method), 344, 346

`__init__()` (ashpy.trainers.gan.AdversarialTrainer method), 351, 357

`__init__()` (ashpy.trainers.gan.EncoderTrainer method), 354, 360

`__init__()` (ashpy.trainers.trainer.Trainer method), 338, 341

`_add_building_block()` (ashpy.models.convolutional.decoders.Decoder method), 204, 253, 257

`_add_building_block()` (ashpy.models.convolutional.discriminators.PatchDiscriminator method), 227, 246, 250

`_add_building_block()` (ashpy.models.convolutional.encoders.Encoder method), 209, 261, 264

`_add_building_block()` (ashpy.models.convolutional.decoders.Decoder method), 204, 253, 257

`_add_building_block()` (ashpy.models.convolutional.discriminators.PatchDiscriminator method), 227, 246, 250

`_add_building_block()` (ashpy.models.convolutional.encoders.FCNNEncoder method), 211, 263, 265

`_add_building_block()` (ashpy.models.convolutional.decoders.Decoder method), 204, 253, 257

`_add_initial_block()` (ashpy.models.convolutional.decoders.FCNNDDecoder method), 206, 255, 258

`_build_and_restore_models()` (ashpy.trainers.AdversarialTrainer method), 332

`_build_and_restore_models()` (ashpy.trainers.EncoderTrainer method), 336

`_build_and_restore_models()` (ashpy.trainers.Trainer method), 327

`_build_and_restore_models()` (ashpy.trainers.classifier.ClassifierTrainer method), 345, 347

`_build_and_restore_models()` (ashpy.trainers.gan.AdversarialTrainer method), 351, 358

`_build_and_restore_models()` (ashpy.trainers.gan.EncoderTrainer method), 355, 361

`_build_and_restore_models()`

<code>(ashpy.trainers.trainer.Trainer</code>	<code>method),</code>	<code>328</code>	<code>_on_batch_end()</code>	<code>(ashpy.trainers.trainer.Trainer</code>
<code>339, 341</code>			<code>method), 339, 342</code>	
<code>_check_model_construction()</code>	<code>(ashpy.restorers.Restorer</code>	<code>static</code>	<code>_on_batch_start()</code>	<code>(ashpy.trainers.Trainer</code>
<code>311</code>	<code>method),</code>		<code>method), 328</code>	
<code>_check_model_construction()</code>	<code>(ashpy.restorers.restorer.Restorer</code>	<code>static</code>	<code>_on_batch_start()</code>	<code>(ashpy.trainers.trainer.Trainer</code>
<code>(ashpy.restorers.restorer.Restorer</code>	<code>method), 317, 319</code>		<code>method), 339, 342</code>	
<code>_check_name_collision()</code>	<code>(ashpy.trainers.Trainer</code>	<code>static</code>	<code>_on_epoch_end()</code>	<code>(ashpy.trainers.Trainer</code>
<code>(ashpy.trainers.Trainer</code>	<code>static</code>	<code>method), 327</code>	<code>328</code>	<code>method),</code>
<code>_check_name_collision()</code>	<code>(ashpy.trainers.trainer.Trainer</code>	<code>static</code>	<code>_on_epoch_end()</code>	<code>(ashpy.trainers.trainer.Trainer</code>
<code>(ashpy.trainers.trainer.Trainer</code>	<code>static</code>	<code>method),</code>	<code>_on_epoch_start()</code>	<code>(ashpy.trainers.Trainer</code>
<code>339, 341</code>			<code>method), 328</code>	
<code>_cleanup()</code>	<code>(ashpy.callbacks.save_callback.SaveCallback</code>	<code>method), 44, 67, 70</code>	<code>_on_epoch_start()</code>	<code>(ashpy.trainers.trainer.Trainer</code>
<code>(ashpy.callbacks.save_callback.SaveCallback</code>	<code>method), 44, 67, 70</code>		<code>method), 339, 342</code>	
<code>_current_epoch()</code>	<code>(ashpy.trainers.Trainer</code>	<code>method),</code>	<code>_on_exception()</code>	<code>(ashpy.trainers.Trainer</code>
<code>328</code>			<code>328</code>	<code>method),</code>
<code>_current_epoch()</code>	<code>(ashpy.trainers.trainer.Trainer</code>	<code>method), 339, 341</code>	<code>_on_exception()</code>	<code>(ashpy.trainers.trainer.Trainer</code>
<code>(ashpy.trainers.trainer.Trainer</code>	<code>method), 339, 341</code>		<code>_on_train_end()</code>	<code>(ashpy.trainers.Trainer</code>
<code>_dataset_from_example()</code>	<code>(ashpy.trainers.Trainer</code>	<code>method), 328</code>	<code>328</code>	<code>method),</code>
<code>_dataset_from_example()</code>	<code>(ashpy.trainers.trainer.Trainer</code>	<code>method),</code>	<code>_on_train_end()</code>	<code>(ashpy.trainers.trainer.Trainer</code>
<code>(ashpy.trainers.trainer.Trainer</code>	<code>method),</code>		<code>_on_train_end()</code>	<code>(ashpy.trainers.trainer.Trainer</code>
<code>339, 341</code>			<code>_on_train_start()</code>	<code>(ashpy.trainers.Trainer</code>
<code>_generate_checkpoint_map()</code>	<code>(ashpy.trainers.Trainer</code>	<code>method), 328</code>	<code>_on_train_start()</code>	<code>(ashpy.trainers.trainer.Trainer</code>
<code>(ashpy.trainers.Trainer</code>	<code>method), 328</code>		<code>_on_train_start()</code>	<code>(ashpy.trainers.trainer.Trainer</code>
<code>_generate_checkpoint_map()</code>	<code>(ashpy.trainers.trainer.Trainer</code>	<code>method),</code>	<code>_reduce()</code>	<code>(ashpy.trainers.Trainer</code>
<code>(ashpy.trainers.trainer.Trainer</code>	<code>method),</code>		<code>_reduce()</code>	<code>(ashpy.trainers.Trainer</code>
<code>339, 341</code>			<code>328</code>	<code>method), 328</code>
<code>_get_layer_spec()</code>	<code>(ashpy.models.convolutional.interfaces.Conv2DInterface</code>	<code>static</code>	<code>_reduce()</code>	<code>(ashpy.trainers.trainer.Trainer</code>
<code>(ashpy.models.convolutional.interfaces.Conv2DInterface</code>	<code>static</code>	<code>method), 200, 267, 269</code>	<code>340, 342</code>	<code>method),</code>
<code>_log_fn()</code>	<code>(ashpy.callbacks.classifier.LogClassifierCallback</code>	<code>static</code>	<code>_restore_checkpoint()</code>	<code>(ashpy.restorers.Restorer</code>
<code>(ashpy.callbacks.classifier.LogClassifierCallback</code>	<code>static</code>	<code>method), 37, 52</code>	<code>_restore_checkpoint()</code>	<code>(ashpy.restorers.restorer.Restorer</code>
<code>_log_fn()</code>	<code>(ashpy.callbacks.gan.LogImageGANCallback</code>	<code>method), 40, 58, 63</code>	<code>317, 319</code>	<code>method),</code>
<code>_log_fn()</code>	<code>(ashpy.callbacks.gan.LogImageGANCallback</code>	<code>method), 42, 61, 65</code>	<code>_restore_or_init()</code>	<code>(ashpy.trainers.Trainer</code>
<code>_log_fn()</code>	<code>(ashpy.callbacks.gan.LogImageGANCallback</code>	<code>method), 42, 61, 65</code>	<code>_restore_or_init()</code>	<code>(ashpy.trainers.trainer.Trainer</code>
<code>_log_metrics_and_reset()</code>	<code>(ashpy.trainers.Trainer</code>	<code>method), 328</code>	<code>_save()</code>	<code>(ashpy.trainers.Trainer</code>
<code>(ashpy.trainers.Trainer</code>	<code>method), 328</code>		<code>340, 342</code>	<code>method),</code>
<code>_log_metrics_and_reset()</code>	<code>(ashpy.trainers.trainer.Trainer</code>	<code>method),</code>	<code>_save()</code>	<code>(ashpy.trainers.Trainer</code>
<code>(ashpy.trainers.trainer.Trainer</code>	<code>method),</code>		<code>329</code>	<code>method), 340,</code>
<code>339, 341</code>			<code>_save_weights_fn()</code>	<code>(ashpy.callbacks.save_callback.SaveCallback</code>
<code>_measure_performance()</code>	<code>(ashpy.trainers.Trainer</code>	<code>method), 328</code>	<code>(ashpy.callbacks.save_callback.SaveCallback</code>	<code>method), 44, 67, 70</code>
<code>_measure_performance()</code>	<code>(ashpy.trainers.trainer.Trainer</code>	<code>method),</code>	<code>_train_step</code>	<code>(ashpy.trainers.AdversarialTrainer</code>
<code>(ashpy.trainers.trainer.Trainer</code>	<code>method),</code>		<code>_train_step</code>	<code>(ashpy.trainers.EncoderTrainer</code>
<code>339, 341</code>			<code>_train_step</code>	<code>(ashpy.trainers.classifier.ClassifierTrainer</code>
<code>_measure_performance_if_needed()</code>	<code>(ashpy.trainers.Trainer</code>	<code>method), 328</code>	<code>_train_step</code>	<code>(ashpy.trainers.gan.AdversarialTrainer</code>
<code>(ashpy.trainers.Trainer</code>	<code>method),</code>		<code>_train_step</code>	<code>(ashpy.trainers.gan.AdversarialTrainer</code>
<code>_measure_performance_if_needed()</code>	<code>(ashpy.trainers.trainer.Trainer</code>	<code>method),</code>	<code>351, 358</code>	<code>attribute), 351, 358</code>
<code>339, 342</code>				
<code>_on_batch_end()</code>	<code>(ashpy.trainers.Trainer</code>	<code>method),</code>		

`_train_step` (*ashpy.trainers.gan.EncoderTrainer* attribute), 355, 361
`_update_checkpoint()` (*ashpy.trainers.Trainer* method), 329
`_update_checkpoint()` (*ashpy.trainers.trainer.Trainer* method), 340, 342
`_update_global_batch_size()` (*ashpy.trainers.Trainer* method), 329
`_update_global_batch_size()` (*ashpy.trainers.trainer.Trainer* method), 340, 342
`_validate_callbacks()` (*ashpy.trainers.Trainer* method), 329
`_validate_callbacks()` (*ashpy.trainers.trainer.Trainer* method), 340, 342
`_validate_metrics()` (*ashpy.trainers.Trainer* method), 329
`_validate_metrics()` (*ashpy.trainers.trainer.Trainer* method), 340, 343

A

`AdversarialEncoderRestorer` (class in *ashpy.restorers*), 314
`AdversarialEncoderRestorer` (class in *ashpy.restorers.gan*), 322, 324
`AdversarialLossType` (class in *ashpy.losses.gan*), 110, 132, 146
`AdversarialRestorer` (class in *ashpy.restorers*), 312
`AdversarialRestorer` (class in *ashpy.restorers.gan*), 323, 325
`AdversarialTrainer` (class in *ashpy.trainers*), 330
`AdversarialTrainer` (class in *ashpy.trainers.gan*), 348, 356
`ashpy.ashtypes` (module), 31
`ashpy.callbacks` (module), 31
`ashpy.callbacks.callback` (module), 46
`ashpy.callbacks.classifier` (module), 51
`ashpy.callbacks.counter_callback` (module), 53
`ashpy.callbacks.events` (module), 54
`ashpy.callbacks.gan` (module), 56
`ashpy.callbacks.save_callback` (module), 65
`ashpy.contexts` (module), 71
`ashpy.contexts.classifier` (module), 82
`ashpy.contexts.context` (module), 79
`ashpy.contexts.gan` (module), 85
`ashpy.keras` (module), 91
`ashpy.keras.losses` (module), 91
`ashpy.layers` (module), 99
`ashpy.losses` (module), 105

`ashpy.losses.classifier` (module), 125
`ashpy.losses.executor` (module), 126
`ashpy.losses.gan` (module), 131
`ashpy.metrics` (module), 151
`ashpy.metrics.classifier` (module), 169
`ashpy.metrics.gan` (module), 173
`ashpy.metrics.metric` (module), 184
`ashpy.metrics.sliced_wasserstein_metric` (module), 189
`ashpy.metrics.ssim_multiscale` (module), 195
`ashpy.models` (module), 198
`ashpy.models.convolutional` (module), 199
`ashpy.models.convolutional.autoencoders` (module), 233
`ashpy.models.convolutional.decoders` (module), 250
`ashpy.models.convolutional.discriminators` (module), 240
`ashpy.models.convolutional.encoders` (module), 258
`ashpy.models.convolutional.interfaces` (module), 266
`ashpy.models.convolutional.pix2pixhd` (module), 278
`ashpy.models.convolutional.unet` (module), 270
`ashpy.models.fc` (module), 289
`ashpy.models.fc.autoencoders` (module), 297
`ashpy.models.fc.decoders` (module), 300
`ashpy.models.fc.encoders` (module), 302
`ashpy.models.fc.interfaces` (module), 304
`ashpy.models.gans` (module), 306
`ashpy.modes` (module), 309
`ashpy.restorers` (module), 310
`ashpy.restorers.classifier` (module), 320
`ashpy.restorers.gan` (module), 321
`ashpy.restorers.restorer` (module), 316
`ashpy.trainers` (module), 326
`ashpy.trainers.classifier` (module), 343
`ashpy.trainers.gan` (module), 348
`ashpy.trainers.trainer` (module), 337
`Attention` (class in *ashpy.layers.attention*), 102
`Autoencoder` (class in *ashpy.models.convolutional.autoencoders*), 212, 234, 238
`Autoencoder` (class in *ashpy.models.fc.autoencoders*), 295, 297, 299

B

`best_folder` (*ashpy.metrics.metric.Metric* attribute), 153, 185, 187
`best_model_sel_file` (*ashpy.metrics.metric.Metric* attribute), 153,

- 185, 187
- `build()` (*ashpy.layers.instance_normalization.InstanceNormalization* method), 102
- `build_discriminator()` (*ashpy.models.convolutional.discriminators.MultiScaleDiscriminator* method), 224, 243, 248
- ## C
- `call()` (*ashpy.keras.losses.DHingeLoss* method), 92, 97
- `call()` (*ashpy.keras.losses.DLeastSquare* method), 93, 98
- `call()` (*ashpy.keras.losses.DMinMax* method), 94, 98
- `call()` (*ashpy.keras.losses.GHingeLoss* method), 96, 99
- `call()` (*ashpy.keras.losses.L1* method), 97, 99
- `call()` (*ashpy.layers.attention.Attention* method), 104
- `call()` (*ashpy.layers.instance_normalization.InstanceNormalization* method), 102
- `call()` (*ashpy.losses.executor.Executor* method), 106, 127, 129
- `call()` (*ashpy.losses.executor.SumExecutor* method), 107, 129, 130
- `call()` (*ashpy.losses.gan.GANExecutor* method), 110, 139, 148
- `call()` (*ashpy.models.convolutional.autoencoders.Autoencoder* method), 214, 236, 239
- `call()` (*ashpy.models.convolutional.autoencoders.FCNNAutoencoder* method), 216, 238, 240
- `call()` (*ashpy.models.convolutional.discriminators.MultiScaleDiscriminator* method), 225, 243, 248
- `call()` (*ashpy.models.convolutional.discriminators.PatchDiscriminator* method), 227, 246, 250
- `call()` (*ashpy.models.convolutional.interfaces.Conv2DInterface* method), 201, 268, 270
- `call()` (*ashpy.models.convolutional.pix2pixhd.GlobalGenerator* method), 233, 280, 286
- `call()` (*ashpy.models.convolutional.pix2pixhd.LocalEnhancer* method), 230, 283, 288
- `call()` (*ashpy.models.convolutional.pix2pixhd.ResNetBlock* method), 285, 289
- `call()` (*ashpy.models.convolutional.unet.UNet* method), 219, 275, 277
- `call()` (*ashpy.models.fc.autoencoders.Autoencoder* method), 296, 299
- `call()` (*ashpy.models.fc.interfaces.FCInterface* method), 290, 306
- `call()` (*ashpy.trainers.AdversarialTrainer* method), 333
- `call()` (*ashpy.trainers.classifier.ClassifierTrainer* method), 345, 347
- `call()` (*ashpy.trainers.EncoderTrainer* method), 336
- `call()` (*ashpy.trainers.gan.AdversarialTrainer* method), 351, 358
- `call()` (*ashpy.trainers.gan.EncoderTrainer* method), 355, 361
- `call()` (*ashpy.trainers.Trainer* method), 329
- `call()` (*ashpy.trainers.trainer.Trainer* method), 340, 342
- `Callback` (class in *ashpy.callbacks.callback*), 32, 47, 49
- `CategoricalCrossEntropy` (class in *ashpy.losses.gan*), 118, 132, 146
- `checkpoint_map` (*ashpy.restorers.Restorer* attribute), 311
- `checkpoint_map` (*ashpy.restorers.restorer.Restorer* attribute), 317, 319
- `classifier_model` (*ashpy.contexts.classifier.ClassifierContext* attribute), 74, 83, 84
- `ClassifierContext` (class in *ashpy.contexts.classifier*), 73, 82, 83
- `ClassifierLoss` (class in *ashpy.losses.classifier*), 108, 125, 126
- `ClassifierLoss` (class in *ashpy.metrics.classifier*), 155, 169, 172
- `ClassifierMetric` (class in *ashpy.metrics.classifier*), 156, 171, 172
- `ClassifierRestorer` (class in *ashpy.restorers*), 315
- `ClassifierRestorer` (class in *ashpy.restorers.classifier*), 320, 321
- `ClassifierTrainer` (class in *ashpy.trainers.classifier*), 343, 346
- `context` (*ashpy.trainers.Trainer* attribute), 329
- `Context` (class in *ashpy.trainers.trainer.Trainer* attribute), 340, 343
- `Context` (class in *ashpy.contexts.context*), 71, 79, 80
- `Conv2DInterface` (class in *ashpy.models.convolutional.interfaces*), 199, 266, 268
- `ConvDiscriminator` (in module *ashpy.models.gans*), 308
- `ConvEncoder` (in module *ashpy.models.gans*), 309
- `ConvGenerator` (in module *ashpy.models.gans*), 307
- `CounterCallback` (class in *ashpy.callbacks.counter_callback*), 35, 53, 54
- `current_batch` (*ashpy.contexts.context.Context* attribute), 72, 80, 81
- ## D
- `dataset` (*ashpy.contexts.context.Context* attribute), 72, 80, 81
- `Decoder` (class in *ashpy.models.convolutional.decoders*), 202, 251, 255
- `Decoder` (class in *ashpy.models.fc.decoders*), 291, 300, 301
- `DenseDiscriminator` (in module *ashpy.models.gans*), 308

DenseEncoder (in module *ashpy.models.gans*), 309
DenseGenerator (in module *ashpy.models.gans*), 307
DHingeLoss (class in *ashpy.keras.losses*), 92, 97
discriminator_loss (ashpy.contexts.gan.GANContext attribute), 76, 86, 89
discriminator_model (ashpy.contexts.gan.GANContext attribute), 76, 86, 89
DiscriminatorAdversarialLoss (class in *ashpy.losses.gan*), 112, 133, 146
DiscriminatorHingeLoss (class in *ashpy.losses.gan*), 124, 134, 147
DiscriminatorLoss (class in *ashpy.metrics.gan*), 158, 174, 181
DiscriminatorLSGAN (class in *ashpy.losses.gan*), 123, 135, 147
DiscriminatorMinMax (class in *ashpy.losses.gan*), 122, 136, 147
DLeastSquare (class in *ashpy.keras.losses*), 93, 97
DMinMax (class in *ashpy.keras.losses*), 94, 98

E

Encoder (class in *ashpy.models.convolutional.encoders*), 207, 259, 263
Encoder (class in *ashpy.models.fc.encoders*), 293, 302, 304
encoder_inputs (ashpy.contexts.gan.GANEncoderContext attribute), 78, 88, 90
encoder_loss (ashpy.contexts.gan.GANEncoderContext attribute), 78, 88, 90
encoder_model (ashpy.contexts.gan.GANEncoderContext attribute), 78, 88, 91
EncoderBCE (class in *ashpy.losses.gan*), 121, 137, 147
EncoderLoss (class in *ashpy.metrics.gan*), 160, 175, 181
EncoderTrainer (class in *ashpy.trainers*), 333
EncoderTrainer (class in *ashpy.trainers.gan*), 352, 359
EncodingAccuracy (class in *ashpy.metrics.gan*), 163, 176, 182
Event (class in *ashpy.callbacks.events*), 37, 55
exception (ashpy.contexts.context.Context attribute), 72, 80, 81
Executor (class in *ashpy.losses.executor*), 105, 127, 129
executors (ashpy.losses.executor.SumExecutor attribute), 107, 129, 130

F

fake_samples (ashpy.contexts.gan.GANContext attribute), 76, 86, 89
FCInterface (class in *ashpy.models.fc.interfaces*), 289, 304, 306

FCNNAutoencoder (class in *ashpy.models.convolutional.autoencoders*), 214, 236, 239
FCNNDecoder (class in *ashpy.models.convolutional.decoders*), 205, 254, 257
FCNNEncoder (class in *ashpy.models.convolutional.encoders*), 210, 261, 265
FeatureMatchingLoss (class in *ashpy.losses.gan*), 117, 138, 148
fn (ashpy.losses.executor.Executor attribute), 106, 127, 129
FUNet () (in module *ashpy.models.convolutional.unet*), 221, 270, 275

G

GANContext (class in *ashpy.contexts.gan*), 75, 85, 89
GANEncoderContext (class in *ashpy.contexts.gan*), 77, 87, 90
GANExecutor (class in *ashpy.losses.gan*), 109, 139, 148
generator_inputs (ashpy.contexts.gan.GANContext attribute), 76, 86, 89
generator_loss (ashpy.contexts.gan.GANContext attribute), 76, 86, 89
generator_model (ashpy.contexts.gan.GANContext attribute), 76, 86, 90
generator_of_encoder (ashpy.contexts.gan.GANEncoderContext attribute), 78, 88, 91
GeneratorAdversarialLoss (class in *ashpy.losses.gan*), 111, 140, 149
GeneratorBCE (class in *ashpy.losses.gan*), 113, 141, 149
GeneratorHingeLoss (class in *ashpy.losses.gan*), 116, 142, 149
GeneratorL1 (class in *ashpy.losses.gan*), 115, 142, 149
GeneratorLoss (class in *ashpy.metrics.gan*), 159, 178, 182
GeneratorLSGAN (class in *ashpy.losses.gan*), 114, 143, 149
get_adversarial_loss_discriminator () (in module *ashpy.losses.gan*), 124, 131, 151
get_adversarial_loss_generator () (in module *ashpy.losses.gan*), 125, 131, 151
get_decoder_block () (ashpy.models.convolutional.unet.UNet method), 219, 275, 277
get_discriminator_inputs () (ashpy.losses.gan.GANExecutor static method), 110, 139, 148
get_encoder_block ()

(*ashpy.models.convolutional.unet.UNet*
method), 219, 275, 278
get_global_step() (*ashpy.restorers.Restorer*
method), 311
get_global_step()
 (*ashpy.restorers.restorer.Restorer* *method*),
 317, 319
get_or_train_inception()
 (*ashpy.metrics.gan.InceptionScore* *static*
method), 162, 180, 183
get_steps_per_epoch() (*ashpy.restorers.Restorer*
method), 311
get_steps_per_epoch()
 (*ashpy.restorers.restorer.Restorer* *method*),
 317, 319
GHingeLoss (*class in ashpy.keras.losses*), 95, 98
global_batch_size
 (*ashpy.losses.executor.Executor* *attribute*),
 106, 127, 130
global_batch_size
 (*ashpy.losses.executor.SumExecutor* *attribute*),
 107, 129, 130
global_step (*ashpy.contexts.context.Context* *at-*
tribute), 72, 80, 81
GlobalGenerator (*class in*
ashpy.models.convolutional.pix2pixhd), 231,
 278, 285

H

H5 (*ashpy.callbacks.save_callback.SaveSubFormat* *at-*
tribute), 46, 69, 71

I

inception_score()
 (*ashpy.metrics.gan.InceptionScore* *method*),
 163, 180, 184
InceptionScore (*class in ashpy.metrics.gan*), 161,
 179, 183
InstanceNormalization (*class in*
ashpy.layers.instance_normalization), 100

J

json_read() (*ashpy.metrics.metric.Metric* *static*
method), 153, 186, 187
json_write() (*ashpy.metrics.metric.Metric* *static*
method), 153, 186, 188

L

L1 (*class in ashpy.keras.losses*), 96, 99
local_example() (*ashpy.trainers.Trainer* *method*),
 329
local_example() (*ashpy.trainers.trainer.Trainer*
method), 340, 343

LocalEnhancer (*class in*
ashpy.models.convolutional.pix2pixhd), 228,
 281, 287
log() (*ashpy.metrics.metric.Metric* *method*), 153, 186,
 188
log() (*ashpy.metrics.sliced_wasserstein_metric.SlicedWassersteinDistanc*
method), 166, 192, 195
log_eval_mode (*ashpy.contexts.context.Context* *at-*
tribute), 72, 80, 81
LogClassifierCallback (*class in*
ashpy.callbacks.classifier), 36, 51, 52
logdir (*ashpy.metrics.metric.Metric* *attribute*), 153,
 186, 188
logdir (*ashpy.metrics.sliced_wasserstein_metric.SlicedWassersteinDistanc*
attribute), 166, 192, 195
LogEvalMode (*class in ashpy.modes*), 309, 310
LogImageGANCallback (*class in*
ashpy.callbacks.gan), 38, 56, 61
LogImageGANEncoderCallback (*class in*
ashpy.callbacks.gan), 40, 59, 63
loss (*ashpy.contexts.classifier.ClassifierContext* *at-*
tribute), 74, 83, 84

M

measure_metrics() (*ashpy.trainers.Trainer*
method), 329
measure_metrics() (*ashpy.trainers.trainer.Trainer*
method), 340, 343
metric (*ashpy.metrics.metric.Metric* *attribute*), 153,
 186, 188
Metric (*class in ashpy.metrics.metric*), 152, 184, 187
metrics (*ashpy.contexts.context.Context* *attribute*), 73,
 80, 81
MODEL (*ashpy.callbacks.save_callback.SaveFormat* *at-*
tribute), 45, 68, 70
model_selection() (*ashpy.metrics.metric.Metric*
method), 154, 186, 188
model_selection()
 (*ashpy.metrics.sliced_wasserstein_metric.SlicedWassersteinDistanc*
method), 166, 192, 195
model_selection() (*ashpy.trainers.Trainer*
method), 329
model_selection() (*ashpy.trainers.trainer.Trainer*
method), 340, 343
model_selection_operator
 (*ashpy.metrics.metric.Metric* *attribute*), 154,
 186, 188
ModelNotConstructedError, 318
MultiScaleDiscriminator (*class in*
ashpy.models.convolutional.discriminators),
 222, 241, 247
N
name (*ashpy.callbacks.callback.Callback* *attribute*), 33,

48, 50
 name (*ashpy.metrics.metric.Metric* attribute), 154, 186, 188
 name () (*ashpy.callbacks.save_callback.SaveFormat* method), 45, 68, 70

O

ON_BATCH_END (*ashpy.callbacks.events.Event* attribute), 37, 55, 56
 on_batch_end () (*ashpy.callbacks.callback.Callback* method), 33, 48, 50
 ON_BATCH_START (*ashpy.callbacks.events.Event* attribute), 37, 55, 56
 on_batch_start () (*ashpy.callbacks.callback.Callback* method), 34, 48, 50
 ON_EPOCH_END (*ashpy.callbacks.events.Event* attribute), 37, 55, 56
 on_epoch_end () (*ashpy.callbacks.callback.Callback* method), 34, 48, 50
 ON_EPOCH_START (*ashpy.callbacks.events.Event* attribute), 37, 55, 56
 on_epoch_start () (*ashpy.callbacks.callback.Callback* method), 34, 48, 50
 on_event () (*ashpy.callbacks.callback.Callback* method), 34, 49, 50
 on_event () (*ashpy.callbacks.counter_callback.CounterCallback* method), 36, 54
 ON_EXCEPTION (*ashpy.callbacks.events.Event* attribute), 37, 55, 56
 on_exception () (*ashpy.callbacks.callback.Callback* method), 34, 49, 51
 ON_TRAIN_END (*ashpy.callbacks.events.Event* attribute), 37, 55, 56
 on_train_end () (*ashpy.callbacks.callback.Callback* method), 34, 49, 51
 ON_TRAIN_START (*ashpy.callbacks.events.Event* attribute), 38, 55, 56
 on_train_start () (*ashpy.callbacks.callback.Callback* method), 34, 49, 51

P

PatchDiscriminator (class in *ashpy.models.convolutional.discriminators*), 225, 244, 249
 Pix2PixLoss (class in *ashpy.losses.gan*), 119, 144, 150
 Pix2PixLossSemantic (class in *ashpy.losses.gan*), 120, 145, 150

R

reduce_loss () (*ashpy.losses.executor.Executor* static method), 106, 128, 130
 reduction (*ashpy.keras.losses.DHingeLoss* attribute), 92, 97

reduction (*ashpy.keras.losses.DLeastSquare* attribute), 94, 98
 reduction (*ashpy.keras.losses.DMinMax* attribute), 95, 98
 reduction (*ashpy.keras.losses.GHingeLoss* attribute), 96, 99
 reduction (*ashpy.keras.losses.L1* attribute), 97, 99
 reset_states () (*ashpy.metrics.metric.Metric* method), 154, 186, 188
 reset_states () (*ashpy.metrics.sliced_wasserstein_metric.SlicedWassersteinMetric* method), 166, 192, 195
 ResNetBlock (class in *ashpy.models.convolutional.pix2pixhd*), 284, 288
 restore_callback () (*ashpy.restorers.Restorer* method), 311
 restore_callback () (*ashpy.restorers.restorer.Restorer* method), 317, 319
 restore_discriminator () (*ashpy.restorers.AdversarialRestorer* method), 313
 restore_discriminator () (*ashpy.restorers.gan.AdversarialRestorer* method), 323, 325
 restore_discriminator_optimizer () (*ashpy.restorers.AdversarialRestorer* method), 313
 restore_discriminator_optimizer () (*ashpy.restorers.gan.AdversarialRestorer* method), 324, 325
 restore_encoder () (*ashpy.restorers.AdversarialEncoderRestorer* method), 314
 restore_encoder () (*ashpy.restorers.gan.AdversarialEncoderRestorer* method), 322, 324
 restore_encoder_optimizer () (*ashpy.restorers.AdversarialEncoderRestorer* method), 314
 restore_encoder_optimizer () (*ashpy.restorers.gan.AdversarialEncoderRestorer* method), 323, 325
 restore_generator () (*ashpy.restorers.AdversarialRestorer* method), 313
 restore_generator () (*ashpy.restorers.gan.AdversarialRestorer* method), 324, 325
 restore_generator_optimizer () (*ashpy.restorers.AdversarialRestorer* method), 313
 restore_generator_optimizer () (*ashpy.restorers.gan.AdversarialRestorer* method), 324, 325

method), 324, 326
 restore_model() (ashpy.restorers.classifier.ClassifierRestorer method), 320, 321
 restore_model() (ashpy.restorers.ClassifierRestorer method), 315
 restore_object() (ashpy.restorers.Restorer method), 312
 restore_object() (ashpy.restorers.restorer.Restorer method), 318, 319
 restore_optimizer() (ashpy.restorers.classifier.ClassifierRestorer method), 321
 restore_optimizer() (ashpy.restorers.ClassifierRestorer method), 315
 Restorer (class in ashpy.restorers), 310
 Restorer (class in ashpy.restorers.restorer), 316, 318
 result() (ashpy.metrics.metric.Metric method), 154, 186, 188

S

sanitized_name (ashpy.metrics.metric.Metric attribute), 154, 187, 188
 save() (ashpy.callbacks.save_callback.SaveFormat method), 45, 68, 70
 save_weights_fn() (ashpy.callbacks.save_callback.SaveCallback method), 45, 67, 70
 SaveCallback (class in ashpy.callbacks.save_callback), 43, 65, 69
 SaveFormat (class in ashpy.callbacks.save_callback), 45, 67, 70
 SaveSubFormat (class in ashpy.callbacks.save_callback), 46, 68, 70
 SingleSWD (class in ashpy.metrics.sliced_wasserstein_metric), 189, 193
 SlicedWassersteinDistance (class in ashpy.metrics.sliced_wasserstein_metric), 165, 191, 193
 split_batch() (ashpy.metrics.ssim_multiscale.SSIM_Multiscale static method), 168, 197, 198
 SSIM_Multiscale (class in ashpy.metrics.ssim_multiscale), 167, 195, 197
 SumExecutor (class in ashpy.losses.executor), 107, 128, 130
 SUNet (class in ashpy.models.convolutional.unet), 220, 271, 276

T

TF (ashpy.callbacks.save_callback.SaveSubFormat at-tribute), 46, 69, 71
 train_step() (ashpy.trainers.AdversarialTrainer method), 333
 train_step() (ashpy.trainers.classifier.ClassifierTrainer method), 346, 348
 train_step() (ashpy.trainers.EncoderTrainer method), 337
 train_step() (ashpy.trainers.gan.AdversarialTrainer method), 352, 358
 train_step() (ashpy.trainers.gan.EncoderTrainer method), 355, 361
 Trainer (class in ashpy.trainers), 326
 Trainer (class in ashpy.trainers.trainer), 338, 340
 training_set (ashpy.contexts.classifier.ClassifierContext attribute), 74, 83, 84

U

UNet (class in ashpy.models.convolutional.unet), 217, 272, 276
 update_state() (ashpy.metrics.classifier.ClassifierLoss method), 156, 170, 172
 update_state() (ashpy.metrics.classifier.ClassifierMetric method), 157, 172, 173
 update_state() (ashpy.metrics.gan.DiscriminatorLoss method), 159, 175, 181
 update_state() (ashpy.metrics.gan.EncoderLoss method), 161, 176, 181
 update_state() (ashpy.metrics.gan.EncodingAccuracy method), 164, 177, 182
 update_state() (ashpy.metrics.gan.GeneratorLoss method), 160, 178, 183
 update_state() (ashpy.metrics.gan.InceptionScore method), 163, 180, 184
 update_state() (ashpy.metrics.metric.Metric method), 154, 187, 189
 update_state() (ashpy.metrics.sliced_wasserstein_metric.SingleSWD method), 190, 193
 update_state() (ashpy.metrics.sliced_wasserstein_metric.SlicedWassersteinDistance method), 167, 193, 195
 update_state() (ashpy.metrics.ssim_multiscale.SSIM_Multiscale method), 168, 197, 198
 validation_set (ashpy.contexts.classifier.ClassifierContext attribute), 74, 83, 84

W

weight (ashpy.losses.executor.Executor attribute), 106, 128, 130
 WEIGHTS (ashpy.callbacks.save_callback.SaveFormat attribute), 45, 68, 70