ASHPY Documentation

Release 0.3.0

ml@zuru.tech

Jan 29, 2020

Contents:

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,
↪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 fist 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 thee 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 decs 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 Pythony 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

CHAPTER 3

Getting Started

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

```python
# Copyright 2019 Zuru Tech HK Limited. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""Example of Multi-GPU classifier trainer."""

import operator

import tensorflow as tf

from ashpy.losses import ClassifierLoss
from ashpy.metrics import ClassifierMetric
from ashpy.trainers import ClassifierTrainer


def main():
    """
    Train a multi-GPU classifier.

    How to use ash to training_set a classifier, measure the
    performance and perform model selection.
    """
    strategy = tf.distribute.MirroredStrategy()
    with strategy.scope():
```

(continues on next page)

(continued from previous page)

```
        training_set, validation_set = tf.keras.datasets.mnist.load_data()

        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

        training_set, validation_set = (
            process(training_set[0], training_set[1]),
            process(validation_set[0], validation_set[1]),
        )

        model = tf.keras.Sequential(
            [
                tf.keras.layers.Dense(10, activation=tf.nn.sigmoid),
                tf.keras.layers.Dense(10),
            ]
        )
        optimizer = tf.optimizers.Adam(1e-3)
        loss = ClassifierLoss(tf.losses.SparseCategoricalCrossentropy(from_
↪logits=True))
        logdir = "testlog"
        epochs = 10

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

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


if __name__ == "__main__":
    main()
```

3.4.2 GANs

BiGAN

```python
# Copyright 2019 Zuru Tech HK Limited. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""Bigan dummy implementation."""

import operator

import tensorflow as tf
from tensorflow import keras

from ashpy.losses import DiscriminatorMinMax, EncoderBCE, GeneratorBCE
from ashpy.metrics import EncodingAccuracy
from ashpy.trainers import EncoderTrainer


def main():
    """Define the trainer and the models."""

    def real_gen():
        """Define generator of real values."""
        for _ in tf.range(100):
            yield ((10.0,), (0,))

    num_classes = 1
    latent_dim = 100

    generator = keras.Sequential([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 = keras.Sequential([keras.layers.Dense(latent_dim)])
    generator_bce = GeneratorBCE()
    encoder_bce = EncoderBCE()
    minmax = DiscriminatorMinMax()

```

(continues on next page)

(continued from previous page)

```
    epochs = 100
    logdir = "log/adversarial/encoder"

    # Fake pre-trained classifier
    classifier = tf.keras.Sequential(
        [tf.keras.layers.Dense(10), tf.keras.layers.Dense(num_classes)]
    )

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

    trainer = EncoderTrainer(
        generator=generator,
        discriminator=discriminator,
        encoder=encoder,
        generator_optimizer=tf.optimizers.Adam(1e-4),
        discriminator_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)


if __name__ == "__main__":
    main()
```

MNIST

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

(continues on next page)

(continued from previous page)

```
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""Adversarial trainer example."""

import tensorflow as tf
from tensorflow import keras  # pylint: disable=no-name-in-module

from ashpy.losses import DiscriminatorMinMax, GeneratorBCE
from ashpy.metrics import InceptionScore
from ashpy.models.gans import ConvDiscriminator, ConvGenerator
from ashpy.trainers import AdversarialTrainer


def main():
    """Adversarial trainer example."""
    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,
        )

        discriminator = ConvDiscriminator(
            layer_spec_input_res=(28, 28),
            layer_spec_target_res=(7, 7),
            kernel_size=(5, 5),
            initial_filters=32,
            filters_cap=128,
            output_shape=1,
        )

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

        # Trainer
        logdir = "log/adversarial"

        # InceptionScore: keep commented until the issues
        # https://github.com/tensorflow/tensorflow/issues/28599
        # https://github.com/tensorflow/hub/issues/295
        # Haven't been solved and merged into tf2

        metrics = [
            # InceptionScore(
            #    InceptionScore.get_or_train_inception(
            #        mnist_dataset,
            #        "mnist",
            #        num_classes=10,
```

(continues on next page)

(continued from previous page)

```
            #         epochs=1,
            #         fine_tuning=False,
            #         logdir=logdir,
            #     ),
            #     model_selection_operator=operator.gt,
            #     logdir=logdir,
            # )
        ]

        epochs = 50
        trainer = 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,
        )

        batch_size = 512

        # Real data
        mnist_x, mnist_y = keras.datasets.mnist.load_data()[0]

        def iterator():
            """Define an iterator in order to do not load in memory all the dataset."
↪"
            for image, label in zip(mnist_x, mnist_y):
                yield tf.image.convert_image_dtype(
                    tf.expand_dims(image, -1), tf.float32
                ), tf.expand_dims(label, -1)

        real_data = (
            tf.data.Dataset.from_generator(
                iterator, (tf.float32, tf.int64), ((28, 28, 1), (1,))
            )
            .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)


if __name__ == "__main__":
    main()
```

Facades (Pix2Pix)

```python
# Copyright 2019 Zuru Tech HK Limited. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""
Pix2Pix on Facades Datasets dummy implementation.

Input Pipeline taken from: https://www.tensorflow.org/beta/tutorials/generative/
↪pix2pix
"""
import os

import tensorflow as tf

from ashpy import LogEvalMode
from ashpy.losses.gan import (
    AdversarialLossType,
    Pix2PixLoss,
    get_adversarial_loss_discriminator,
)
from ashpy.models.convolutional.discriminators import PatchDiscriminator
from ashpy.models.convolutional.unet import FUNet
from ashpy.trainers.gan import AdversarialTrainer

_URL = "https://people.eecs.berkeley.edu/~tinghuiz/projects/pix2pix/datasets/facades.
↪tar.gz"

PATH_TO_ZIP = tf.keras.utils.get_file("facades.tar.gz", origin=_URL, extract=True)
PATH = os.path.join(os.path.dirname(PATH_TO_ZIP), "facades/")

BUFFER_SIZE = 100
BATCH_SIZE = 1
IMG_WIDTH = 256
IMG_HEIGHT = 256


def load(image_file):
    """Load the image from file path."""
    image = tf.io.read_file(image_file)
    image = tf.image.decode_jpeg(image)

    width = tf.shape(image)[1]

    width = width // 2
    real_image = image[:, :width, :]
```

(continues on next page)

(continued from previous page)

```
    input_image = image[:, width:, :]

    input_image = tf.cast(input_image, tf.float32)
    real_image = tf.cast(real_image, tf.float32)

    return input_image, real_image


def resize(input_image, real_image, height, width):
    """Resize input_image and real_image to height x width."""
    input_image = tf.image.resize(
        input_image, [height, width], method=tf.image.ResizeMethod.NEAREST_NEIGHBOR
    )
    real_image = tf.image.resize(
        real_image, [height, width], method=tf.image.ResizeMethod.NEAREST_NEIGHBOR
    )

    return input_image, real_image


def random_crop(input_image, real_image):
    """Random crop both input_image and real_image."""
    stacked_image = tf.stack([input_image, real_image], axis=0)
    cropped_image = tf.image.random_crop(
        stacked_image, size=[2, IMG_HEIGHT, IMG_WIDTH, 3]
    )

    return cropped_image[0], cropped_image[1]


def normalize(input_image, real_image):
    """Normalize images in [-1, 1]."""
    input_image = (input_image / 127.5) - 1
    real_image = (real_image / 127.5) - 1

    return input_image, real_image


def load_image_train(image_file):
    """Load and process the image_file to be ready for the training."""
    input_image, real_image = load(image_file)
    input_image, real_image = random_jitter(input_image, real_image)
    input_image, real_image = normalize(input_image, real_image)

    return input_image, real_image


@tf.function
def random_jitter(input_image, real_image):
    """Apply random jitter to both input_image and real_image."""
    # resizing to 286 x 286 x 3
    input_image, real_image = resize(input_image, real_image, 286, 286)

    # randomly cropping to 256 x 256 x 3
    input_image, real_image = random_crop(input_image, real_image)

    if tf.random.uniform(()) > 0.5:
```

(continues on next page)

(continued from previous page)

```
        # random mirroring
        input_image = tf.image.flip_left_right(input_image)
        real_image = tf.image.flip_left_right(real_image)

    return input_image, real_image


def main(
    kernel_size=5,
    learning_rate_d=2e-4,
    learning_rate_g=2e-4,
    g_input_res=IMG_WIDTH,
    g_min_res=1,
    g_initial_filters=64,
    g_filters_cap=512,
    use_dropout_encoder=False,
    use_dropout_decoder=True,
    d_target_res=32,
    d_initial_filters=64,
    d_filters_cap=512,
    use_dropout_discriminator=False,
    dataset_name="facades",
    resolution=256,
    epochs=100_000,
    dropout_prob=0.3,
    l1_loss_weight=100,
    gan_loss_weight=1,
    use_attention_d=False,
    use_attention_g=False,
    channels=3,
    gan_loss_type=AdversarialLossType.LSGAN,
):
    """Define Trainer and models."""
    generator = FUNet(
        input_res=g_input_res,
        min_res=g_min_res,
        kernel_size=kernel_size,
        initial_filters=g_initial_filters,
        filters_cap=g_filters_cap,
        channels=channels,  # color_to_label_tensor.shape[0],
        use_dropout_encoder=use_dropout_encoder,
        use_dropout_decoder=use_dropout_decoder,
        dropout_prob=dropout_prob,
        use_attention=use_attention_g,
    )
    discriminator = PatchDiscriminator(
        input_res=resolution,
        min_res=d_target_res,
        initial_filters=d_initial_filters,
        kernel_size=kernel_size,
        filters_cap=d_filters_cap,
        use_dropout=use_dropout_discriminator,
        dropout_prob=dropout_prob,
        use_attention=use_attention_d,
    )

    discriminator_loss = get_adversarial_loss_discriminator(gan_loss_type)()
```

(continues on next page)

(continued from previous page)

```
    generator_loss = Pix2PixLoss(
        l1_loss_weight=l1_loss_weight,
        adversarial_loss_weight=gan_loss_weight,
        adversarial_loss_type=gan_loss_type,
    )

    metrics = []
    logdir = f'{"log"}/{dataset_name}/run2'

    if not os.path.exists(logdir):
        os.makedirs(logdir)

    trainer = AdversarialTrainer(
        generator=generator,
        discriminator=discriminator,
        generator_optimizer=tf.optimizers.Adam(learning_rate_g, beta_1=0.5),
        discriminator_optimizer=tf.optimizers.Adam(learning_rate_d, beta_1=0.5),
        generator_loss=generator_loss,
        discriminator_loss=discriminator_loss,
        epochs=epochs,
        metrics=metrics,
        logdir=logdir,
        log_eval_mode=LogEvalMode.TEST,
    )

    train_dataset = tf.data.Dataset.list_files(PATH + "train/*.jpg")
    train_dataset = train_dataset.shuffle(BUFFER_SIZE)
    train_dataset = train_dataset.map(load_image_train)
    train_dataset = train_dataset.batch(BATCH_SIZE)

    train_dataset = train_dataset.map(lambda x, y: ((y, x), x))

    trainer(
        # generator_input,
        train_dataset
    )


if __name__ == "__main__":
    main()
```

CHAPTER 4

API Reference

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

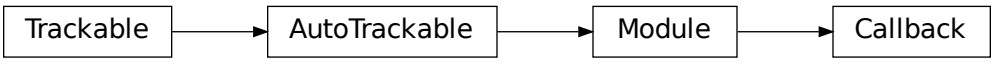
The basic class is *ashpy.callbacks.callback.Callback* . All possible events as listed as Enum inside *ashpy.callbacks.events.Event* .

Classes

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

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

Attributes

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

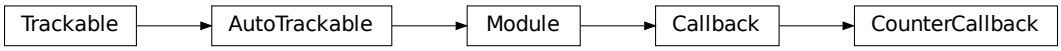
Method called at the beginning of the training loop.

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

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

Attributes

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

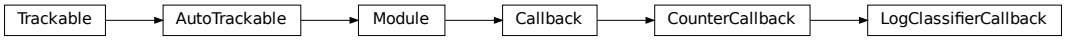
on_event(*event*, *context*)
Count events and calls fn.

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

__init__([event, name, event_freq])	Initialize the LogClassifierCallback.

Attributes

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

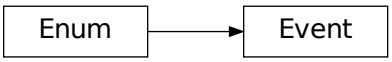
static _log_fn(*context*)
Log output of the image and label to Tensorboard.

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

ON_BATCH_END	On Batch End Event
ON_BATCH_START	On Batch Start Event
ON_EPOCH_END	On Epoch End Event
ON_EPOCH_START	On Epoch Start Event
ON_EXCEPTION	On Exception Event
ON_TRAIN_END	On Train End Event
ON_TRAIN_START	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

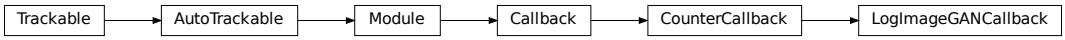
ON_EXCEPTION = 'ON_EXCEPTION'
On Exception Event

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
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,
)

# 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 = "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 os.path.exists(logdir)

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

`__init__`([event, name, event_freq])	Initialize the LogImageCallbackGAN.

Attributes

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

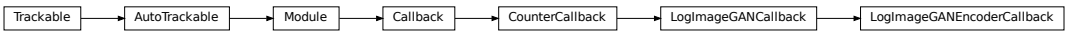
Log output of the generator to Tensorboard.

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

Return type None

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

__init__([event, name, event_freq])	Initialize the LogImageGANEncoderCallback.

Attributes

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

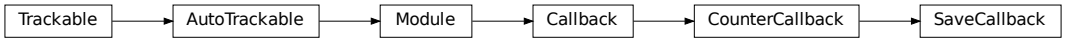
Log output of the generator to Tensorboard.

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

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

Attributes

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

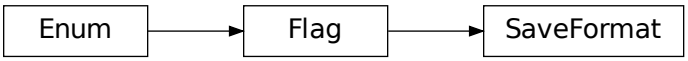
Save weights.

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

MODEL	Model format (weights and architecture), saved using *model.save()*
WEIGHTS	Weights format, saved using *model.save_weights()*

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.

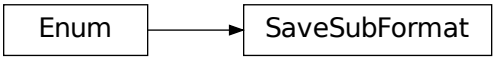
Parameters

- **model** (tf.keras.models.Model) – model to Save.
- **save_dir** (*str*) – 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

H5	H5 Format
TF	TensorFlow format

H5 = 'h5'

H5 Format

TF = 'tf'

TensorFlow format

Modules

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

4.2.10 callback

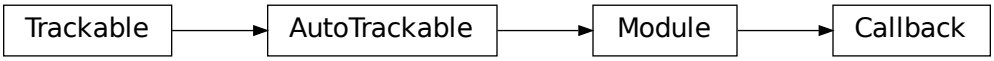
Callback definition.

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

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

Continued on next page

Table 20 – continued from previous page

on_train_end(context)	Handle on_train_end event.
on_train_start(context)	Handle on_train_start event.

Attributes

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

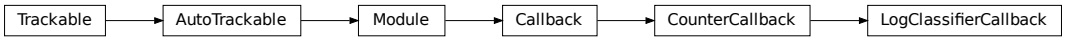
Classifier callbacks.

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

`__init__`([event, name, event_freq])	Initialize the LogClassifierCallback.

Attributes

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

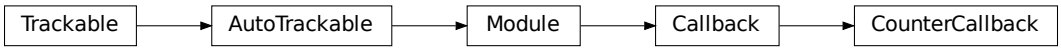
Callback that count events and calls the passed fn evert event_freq.

Classes

CounterCallback	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

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

Attributes

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

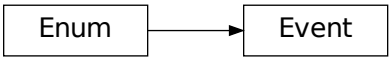
Event definition as Enum.

Classes

Event	Define all possible events.

Event

Inheritance Diagram



---

class ashpy.callbacks.events.**Event**

Bases: enum.Enum

Define all possible events.

Attributes

ON_BATCH_END	On Batch End Event
ON_BATCH_START	On Batch Start Event
ON_EPOCH_END	On Epoch End Event
ON_EPOCH_START	On Epoch Start Event
ON_EXCEPTION	On Exception Event
ON_TRAIN_END	On Train End Event
ON_TRAIN_START	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.

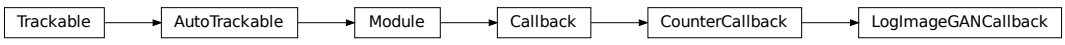
LogImageGANCallback: Log output of the generator when evaluated in its inputs. LogImageGANEncoderCallback: Log output of the generator when evaluated in the encoder

Classes

LogImageGANCallback	Callback used for logging GANs images to Tensorboard.
LogImageGANEncoderCallback	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
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,
)

# 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/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 os.path.exists(logdir)

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

__init__([event, name, event_freq])	Initialize the LogImageCallbackGAN.

Attributes

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

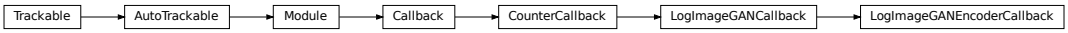
_log_fn(*context*)
Log output of the generator to Tensorboard.

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

__init__([event, name, event_freq])	Initialize the LogImageGANEncoderCallback.

Attributes

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

(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 = "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 os.path.exists(logdir)

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

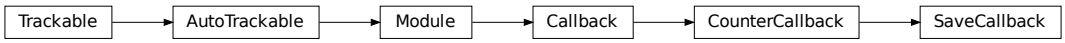
Save weights callback.

Classes

SaveCallback	Save Callback implementation.
SaveFormat	Save Format enum.
SaveSubFormat	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

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

Attributes

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

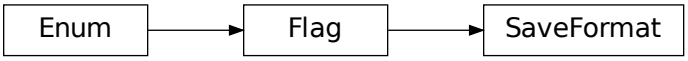
_save_weights_fn(*step*)
Save weights.

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

MODEL	Model format (weights and architecture), saved using *model.save()*
WEIGHTS	Weights format, saved using *model.save_weights()*

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.

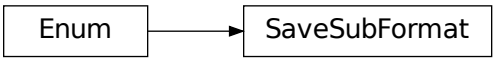
Parameters

- **model** (tf.keras.models.Model) – model to Save.
- **save_dir** (*str*) – 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

H5	H5 Format
TF	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** (*str*) – 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

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

__init__([metrics, dataset, log_eval_mode, ...])	Initialize the Context.

Attributes

current_batch	Return the current batch.
dataset	Retrieve the dataset.
exception	Return the exception.
global_step	Retrieve the global_step.
log_eval_mode	Retrieve model(s) mode.
metrics	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** (list 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*.

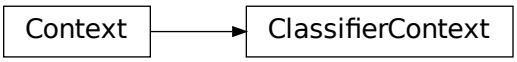
metrics
Retrieve the metrics.

Return type List[*Metric*]

Returns list 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>*, *checkpoint=None*)

Bases: *ashpy.contexts.context.Context*

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

Methods

__init__([classifier_model, loss, dataset, ...])	Instantiate the *ashpy.contexts.classifier.ClassifierContext* context.

Attributes

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

Continued on next page

Table 44 – continued from previous page

training_set	Return the training set.
validation_set	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

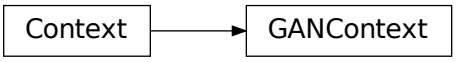
Return the validation set.

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*, *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*)

Bases: *ashpy.contexts.context.Context*

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

Methods

__init__([dataset, generator_model, ...])	Initialize the Context.

Attributes

current_batch	Return the current batch.
dataset	Retrieve the dataset.
discriminator_loss	Retrieve the discriminator loss.
discriminator_model	Retrieve the discriminator model.
exception	Return the exception.
fake_samples	Retrieve the fake samples, i.e.
generator_inputs	Retrieve the generator inputs.
generator_loss	Retrieve the generator loss.
generator_model	Retrieve the generator model.
global_step	Retrieve the global_step.
log_eval_mode	Retrieve model(s) mode.
metrics	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*) – Modes' 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*]

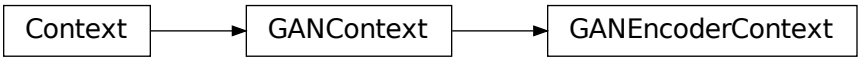
generator_model
Retrieve the generator model.

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*, *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

__init__([dataset, generator_model, ...])	Initialize the Context.

Attributes

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

context	Primitive Context Interface.
classifier	Classifier Context.
gan	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

Context	`ashpy.contexts.Context` 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

__init__([metrics, dataset, log_eval_mode, ...])	Initialize the Context.

Attributes

current_batch	Return the current batch.
dataset	Retrieve the dataset.
exception	Return the exception.

Continued on next page

Table 52 – continued from previous page

global_step	Retrieve the global_step.
log_eval_mode	Retrieve model(s) mode.
metrics	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** (list 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 List[*Metric*]

Returns list 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>*, *checkpoint=None*)

Initialize the Context.

Parameters

- **metrics** (list 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 List[*Metric*]

Returns `list` of [*ashpy.metrics.metric.Metric*].

4.3.6 classifier

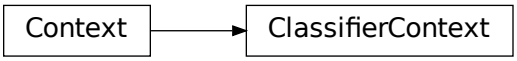
Classifier Context.

Classes

ClassifierContext	`ashpy.ClassifierContext` 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>*, *checkpoint=None*)

Bases: *ashpy.contexts.context.Context*

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

Methods

__init__([classifier_model, loss, dataset, ...])	Instantiate the *ashpy.contexts.classifier.ClassifierContext* context.

Attributes

classifier_model	Retrieve the Model Object.
`current_batch`	Return the current batch.
`dataset`	Retrieve the dataset.
`exception`	Return the exception.
`global_step`	Retrieve the global_step.

Continued on next page

Table 55 – continued from previous page

log_eval_mode	Retrieve model(s) mode.
loss	Retrieve the loss value.
metrics	Retrieve the metrics.
training_set	Return the training set.
validation_set	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>*, *checkpoint=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

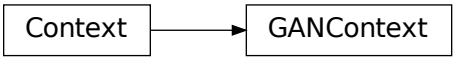
GANContext measures the specified metrics on the GAN.

Classes

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

GANContext

Inheritance Diagram



---

class ashpy.contexts.gan.**GANContext**(*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*)

Bases: *ashpy.contexts.context.Context*

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

Methods

__init__([dataset, generator_model, ...])	Initialize the Context.

Attributes

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

Continued on next page

Table 58 – continued from previous page

generator_loss	Retrieve the generator loss.
generator_model	Retrieve the generator model.
global_step	Retrieve the global_step.
log_eval_mode	Retrieve model(s) mode.
metrics	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*) – Modes' 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*]

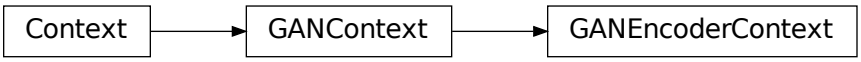
generator_model
Retrieve the generator model.

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

__init__([dataset, generator_model, ...])	Initialize the Context.

Attributes

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

Continued on next page

Table 60 – continued from previous page

`generator_loss`	Retrieve the generator loss.
`generator_model`	Retrieve the generator model.
`generator_of_encoder`	Retrieve the images generated from the encoder output.
`global_step`	Retrieve the global_step.
`log_eval_mode`	Retrieve model(s) mode.
`metrics`	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*, *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*)

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>*, *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`.

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.

__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*) – Modes' 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

losses	Custom Keras losses, used by the AshPy executors.

4.4.1 losses

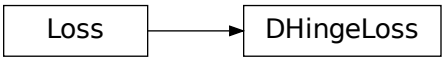
Custom Keras losses, used by the AshPy executors.

Classes

DHingeLoss	Discriminator Hinge Loss as Keras Metric.
DLeastSquare	Discriminator Least Square Loss as `tf.keras.losses.Loss`.
DMinMax	Implementation of MinMax Discriminator loss as `tf.keras.losses.Loss`.
GHingeLoss	Generator Hinge Loss as Keras Metric.
L1	L1 Loss implementation as `tf.keras.losses.Loss`.

DHingeLoss

Inheritance Diagram



---

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:

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

Methods

__init__()	Initialize the Loss.
call(d_real, d_fake)	Compute the hinge loss.

Attributes

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

__init__()
Initialize the Loss.

Return type None

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

Return type Tensor

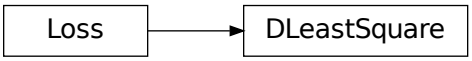
reduction
Return the current *reduction* for this type of loss.

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

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

Attributes

reduction	Return the reduction type for this 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

[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.

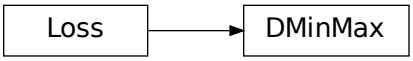
reduction
Return the reduction type for this loss.

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

__init__([from_logits, label_smoothing])	Initialize the loss.
call(d_real, d_fake)	Compute the MinMax Loss.

Attributes

reduction	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

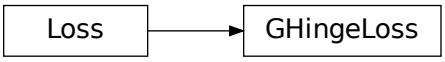
Return the reduction type of this loss.

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 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

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

This can be simply approximated as:

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

Methods

[1] Geometric GAN https://arxiv.org/abs/1705.02894

__init__()	Initialize the Loss.
call(d_real, d_fake)	Compute the hinge loss.

Attributes

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

__init__()
Initialize the Loss.

Return type None

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

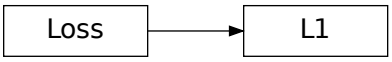
Return type Tensor

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

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

Attributes

reduction	Return the current *reduction* 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

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

This can be simply approximated as:

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

__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

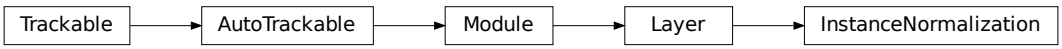
Collection of layers.

Layers

instance_normalization.InstanceNormalization	Instance Normalization Layer (used by Pix2Pix [1]_ and Pix2PixHD [2]_).
attention.Attention	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 Pix2Pix[1] and Pix2PixHD[2]).

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

__init__([eps, beta_initializer, ...])	Initialize the layer.
build(input_shape)	Assemble the layer.
call(inputs[, training])	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	
losses	Losses which are associated with this *Layer*.
metrics	
name	Returns the name of this module as passed or determined in the ctor.
name_scope	Returns a *tf.name_scope* 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.
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__(*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

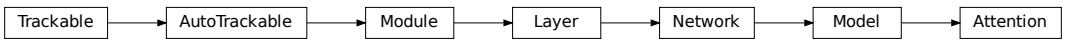
- **inputs** (tf.Tensor) – Inputs for the computation.
- **training** (*bool*) – Controls for training or evaluation mode.

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 GAN[1].

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 $(o_1, ..., o_N) \in \mathbb{R}^{C \times N}$:

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

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

__init__(filters)	Build the Attention Layer.
call(inputs[, training])	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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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

executor.Executor	Carry a function and the way of executing it.
executor.SumExecutor	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

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

Attributes

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

Continued on next page

Table 80 – continued from previous page

weight	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.

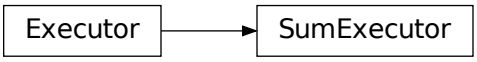
This weight is multiplied by the loss value. This is useful when working with multiples losses.

Return type Callable[..., float]

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

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

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

Attributes

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

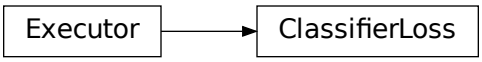
Returns int – Global Batch size value.

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

__init__(fn)	Initialize *ClassifierLoss*.
call(*args, **kwargs)	

Attributes

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

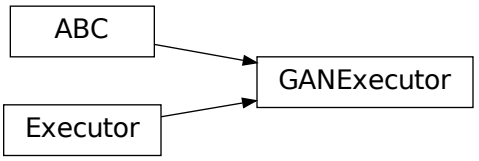
Returns None

GAN

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

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

Attributes

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

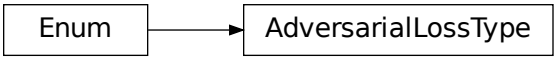
- **context** (*ashpy.contexts.gan.GANContext*) – Context for GAN models.
- **fake_or_real** (tf.Tensor) – Discriminator input tensor, it can be fake (generated) or real.
- **condition** (tf.Tensor) – Discriminator condition (it can also be generator noise).
- **training** (bool) – whether is training 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

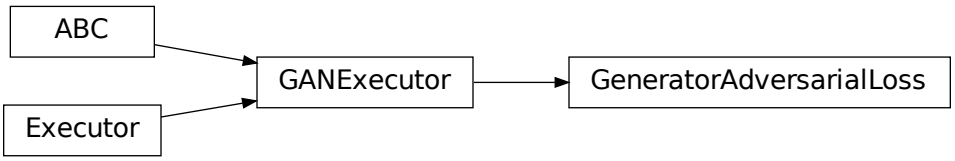
Enumeration for Adversarial Losses. Implemented: GAN and LSGAN.

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

__init__([loss_fn])	Initialize the Executor.
call(*args, **kwargs)	

Attributes

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

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

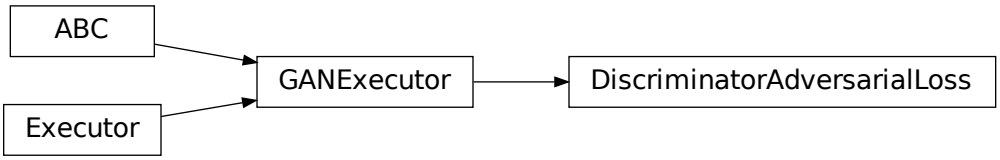
Parameters **loss_fn** (tf.keras.losses.Loss) – Keras Loss function to call passing

(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

__init__([loss_fn])	Initialize the Executor.
call(*args, **kwargs)	

Attributes

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

__init__(*loss_fn=None*)

Initialize the Executor.

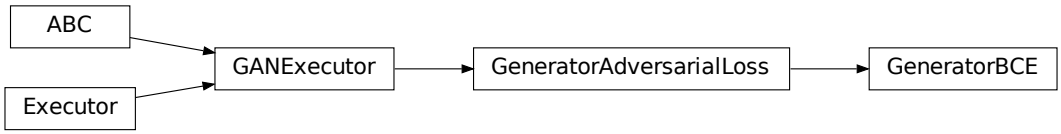
Parameters

- **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

__init__([from_logits])	Initialize the BCE Loss for the Generator.

Attributes

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

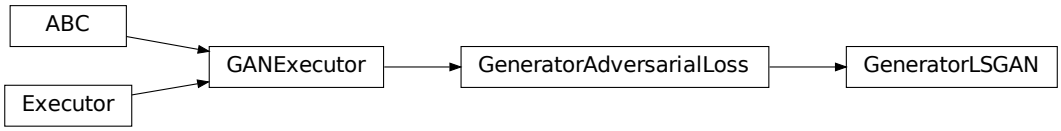
__init__(*from_logits=True*)

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

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

Attributes

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

__init__()	Initialize the Executor.
call(*args, **kwargs)	

Attributes

fn	Return the Keras loss function to execute.
global_batch_size	Global batch size comprises the batch size for each cpu.
weight	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 GAN[1] for more details.

Methods

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

Attributes

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

__init__()

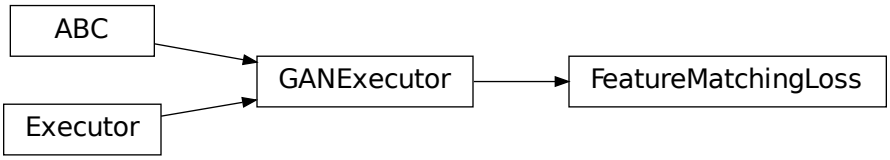
Initialize the Least Square Loss for the Generator.

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

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

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
- || ||_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

__init__()	Initialize the Executor.
call(*args, **kwargs)	

Attributes

fn	Return the Keras loss function to execute.

Continued on next page

Table 103 – continued from previous page

global_batch_size	Global batch size comprises the batch size for each cpu.
weight	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

__init__()	Initialize the Categorical Cross Entropy Executor.
call(*args, **kwargs)	

Attributes

fn	Return the Keras loss function to execute.
global_batch_size	Global batch size comprises the batch size for each cpu.
weight	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*, *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

__init__([l1_loss_weight, ...])	Initialize the loss.

Attributes

executors	Return the List of Executors.
fn	Return the Keras loss function to execute.
global_batch_size	Global batch size comprises the batch size for each cpu.
weight	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 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*, *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*.

Methods

__init__([cross_entropy_weight, ...])	Initialize the Executor.

Attributes

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

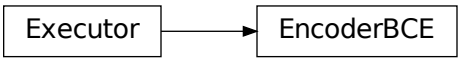
Weighted sum of *ashpy.losses.gan.CategoricalCrossEntropy*, ashpy.losses.gan.AdversarialLossG and *ashpy.losses.gan.FeatureMatchingLoss*

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

__init__([from_logits])	Initialize the Executor.
call(*args, **kwargs)	

Attributes

fn	Return the Keras loss function to execute.

Continued on next page

Table 111 – continued from previous page

`global_batch_size`	Global batch size comprises the batch size for each cpu.
`weight`	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

__init__([from_logits, label_smoothing])	Initialize Loss.

Attributes

`fn`	Return the Keras loss function to execute.
`global_batch_size`	Global batch size comprises the batch size for each cpu.
`weight`	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 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.

Methods

__init__()	Initialize loss.

Attributes

fn	Return the Keras loss function to execute.
global_batch_size	Global batch size comprises the batch size for each cpu.
weight	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 GAN[1] for more details.

Methods

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

Attributes

fn	Return the Keras loss function to execute.
global_batch_size	Global batch size comprises the batch size for each cpu.
weight	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 fot 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: 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).

Modules

classifier	The classification losses.
executor	The Executor.
gan	GAN losses.

4.6.22 classifier

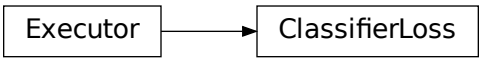
The classification losses.

Classes

ClassifierLoss	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

__init__(fn)	Initialize *ClassifierLoss*.
call(*args, **kwargs)	

Attributes

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

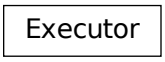
An object that, given an ashpy.contexts.Context, carries a function and the way of executing it.

Classes

Executor	Carry a function and the way of executing it.
SumExecutor	The sum executor.

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

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

Attributes

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

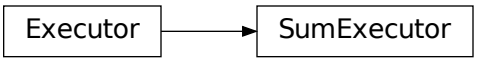
This weight is multiplied by the loss value. This is useful when working with multiples losses.

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

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

Attributes

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

get_adversarial_loss_discriminator	Return the correct loss fot the Discriminator.
get_adversarial_loss_generator	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 fot 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

AdversarialLossType	Enumeration for Adversarial Losses.
CategoricalCrossEntropy	Categorical Cross Entropy between generator output and target.
DiscriminatorAdversarialLoss	Base class for the adversarial loss of the discriminator.
DiscriminatorHingeLoss	Hinge loss for the Discriminator.
DiscriminatorLSGAN	Least square Loss for discriminator.
DiscriminatorMinMax	The min-max game played by the discriminator.
EncoderBCE	The Binary Cross Entropy computed among the encoder and the 0 label.
FeatureMatchingLoss	Conditional GAN Feature matching loss.
GANExecutor	Executor for GANs.
GeneratorAdversarialLoss	Base class for the adversarial loss of the generator.

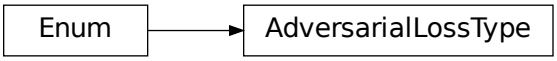
Continued on next page

Table 128 – continued from previous page

GeneratorBCE	The Binary CrossEntropy computed among the generator and the 1 label.
GeneratorHingeLoss	Hinge loss for the Generator.
GeneratorL1	L1 loss between the generator output and the target.
GeneratorLSGAN	Least Square GAN Loss for generator.
Pix2PixLoss	Pix2Pix Loss.
Pix2PixLossSemantic	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

__init__()	Initialize the Categorical Cross Entropy Executor.
call(*args, **kwargs)	

Attributes

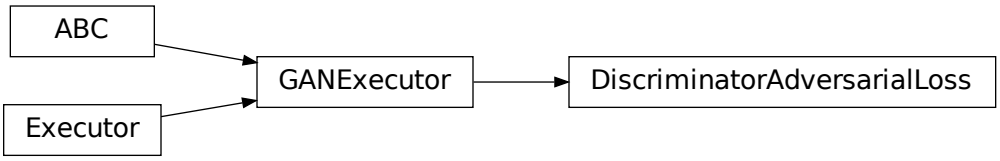
fn	Return the Keras loss function to execute.
global_batch_size	Global batch size comprises the batch size for each cpu.
weight	Return the loss weight.

__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

__init__([loss_fn])	Initialize the Executor.
call(*args, **kwargs)	

Attributes

`fn`	Return the Keras loss function to execute.
`global_batch_size`	Global batch size comprises the batch size for each cpu.
`weight`	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 GAN[1] for more details.

Methods

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

Attributes

`fn`	Return the Keras loss function to execute.
`global_batch_size`	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 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.

Methods

__init__()	Initialize loss.

Attributes

[1] Least Squares Generative Adversarial Networks https://arxiv.org/abs/1611.04076

`fn`	Return the Keras loss function to execute.
`global_batch_size`	Global batch size comprises the batch size for each cpu.
`weight`	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

__init__([from_logits, label_smoothing])	Initialize Loss.

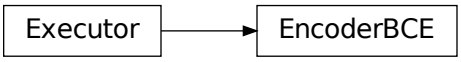
Attributes

`fn`	Return the Keras loss function to execute.
`global_batch_size`	Global batch size comprises the batch size for each cpu.
`weight`	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

__init__([from_logits])	Initialize the Executor.
call(*args, **kwargs)	

Attributes

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

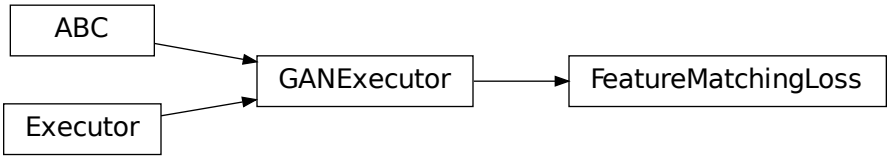
__init__(*from_logits=True*)

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

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

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
- || ||_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

__init__()	Initialize the Executor.
call(*args, **kwargs)	

Attributes

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

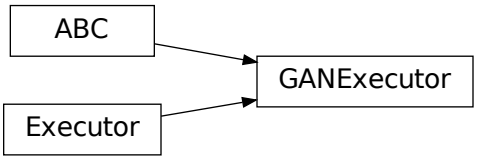
__init__()

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

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

Attributes

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

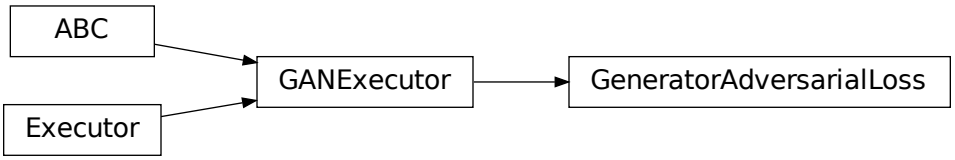
- **condition** (tf.Tensor) – Discriminator condition (it can also be generator noise).
- **training** (bool) – whether is training phase or not

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

__init__([loss_fn])	Initialize the Executor.
call(*args, **kwargs)	

Attributes

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

__init__(*loss_fn=None*)

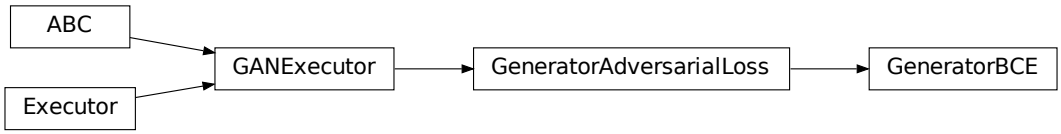
Initialize the Executor.

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

__init__([from_logits])	Initialize the BCE Loss for the Generator.

Attributes

fn	Return the Keras loss function to execute.
global_batch_size	Global batch size comprises the batch size for each cpu.
weight	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 GAN[1] for more details.

Methods

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

Attributes

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

__init__()	Initialize the Executor.
call(*args, **kwargs)	

Attributes

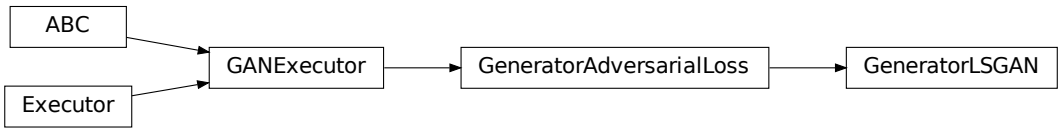
fn	Return the Keras loss function to execute.
global_batch_size	Global batch size comprises the batch size for each cpu.
weight	Return the loss weight.

__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

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

Attributes

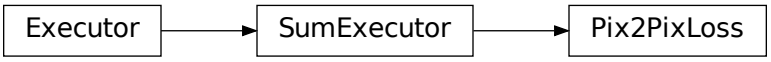
fn	Return the Keras loss function to execute.
global_batch_size	Global batch size comprises the batch size for each cpu.
weight	Return the loss weight.

__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

__init__([l1_loss_weight, ...])	Initialize the loss.

Attributes

executors	Return the List of Executors.
fn	Return the Keras loss function to execute.

Continued on next page

Table 157 – continued from previous page

`global_batch_size`	Global batch size comprises the batch size for each cpu.
`weight`	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*, *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*.

Methods

__init__([cross_entropy_weight, ...])	Initialize the Executor.

Attributes

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

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

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
- || ||_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 fot 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

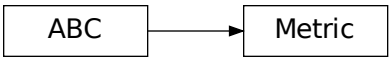
Collection of Metrics.

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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/*

Bases: abc.ABC

Metric is the abstract class that every ash Metric must implement.

AshPy Metrics wrap and extend Keras Metrics.

Methods

__init__(name, metric[, ...])	Initialize the Metric object.
json_read(filename)	Read a JSON file.
json_write(filename, what_to_write)	Write inside the specified JSON file the mean and stddev.
log(step)	Log the metric.
model_selection(checkpoint, global_step)	Perform model selection.
reset_states()	Reset the state of the metric.
result()	Get the result of the metric.
update_state(context)	Update the internal state of the metric, using the information from the context object.

Attributes

best_folder	Retrieve the folder used to save the best model when doing model selection.
best_model_sel_file	Retrieve the path to JSON file containing the measured performance of the best model.
logdir	Retrieve the log directory.
metric	Retrieve the tf.keras.metrics.Metric object.
model_selection_operator	Retrieve the operator used for model selection.
name	Retrieve the metric name.

Continued on next page

Table 162 – continued from previous page

sanitized_name	all / are _.

__init__(*name*, *metric*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/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 str

best_model_sel_file
Retrieve the path to JSON file containing the measured performance of the best model.

Return type str

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 str

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 `None`

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.

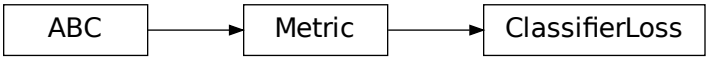
Return type None

Classifier

classifier.ClassifierLoss	A handy way to measure the classification loss.
classifier.ClassifierMetric	Wrap a metric using *argmax* 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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/*

Bases: *ashpy.metrics.metric.Metric*

A handy way to measure the classification loss.

Methods

__init__([name, model_selection_operator, ...])	Initialize the Metric.
update_state(context)	Update the internal state of the metric, using the information from the context object.

Attributes

best_folder	Retrieve the folder used to save the best model when doing model selection.
best_model_sel_file	Retrieve the path to JSON file containing the measured performance of the best model.
logdir	Retrieve the log directory.
metric	Retrieve the tf.keras.metrics.Metric object.
model_selection_operator	Retrieve the operator used for model selection.
name	Retrieve the metric name.
sanitized_name	all / are _.

__init__(*name='loss'*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/c*

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*)

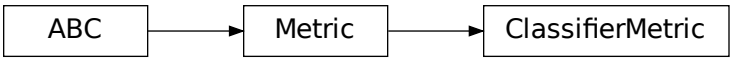
Update the internal state of the metric, using the information from the context object.

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='/home/docs/checkouts/readthedocs.org/user_builds/ash*
processing_predictions=None)

Bases: *ashpy.metrics.metric.Metric*

Wrap a metric using *argmax* to extract predictions out of a classifier's output.

Methods

__init__(metric[, model_selection_operator, ...])	Initialize the Metric.
update_state(context)	Update the internal state of the metric, using the information from the context object.

Attributes

best_folder	Retrieve the folder used to save the best model when doing model selection.
best_model_sel_file	Retrieve the path to JSON file containing the measured performance of the best model.
logdir	Retrieve the log directory.

Continued on next page

Table 167 – continued from previous page

`metric`	Retrieve the `tf.keras.metrics.Metric` object.
`model_selection_operator`	Retrieve the operator used for model selection.
`name`	Retrieve the metric name.
`sanitized_name`	all / are _.

__init__ (*metric*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checko*
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.

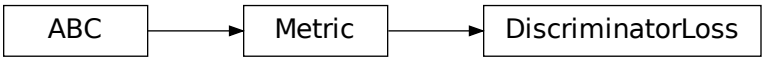
Return type None

GAN

gan.DiscriminatorLoss	The Discriminator loss value.
gan.GeneratorLoss	Generator loss value.
gan.EncoderLoss	Encoder Loss value.
gan.InceptionScore	Inception Score Metric.
gan.EncodingAccuracy	Generator and Encoder accuracy performance.
sliced_wasserstein_metric.SlicedWassersteinDistance	Sliced Wasserstein Distance.
ssim_multiscale.SSIM_Multiscale	Multiscale Structural Similarity.

4.7.4 DiscriminatorLoss

Inheritance Diagram



---

class ashpy.metrics.gan.**DiscriminatorLoss**(*name='d_loss'*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/check*

Bases: *ashpy.metrics.metric.Metric*

The Discriminator loss value.

Methods

__init__([name, model_selection_operator, ...])	Initialize the Metric.
update_state(context)	Update the internal state of the metric, using the information from the context object.

Attributes

best_folder	Retrieve the folder used to save the best model when doing model selection.
best_model_sel_file	Retrieve the path to JSON file containing the measured performance of the best model.
logdir	Retrieve the log directory.
metric	Retrieve the tf.keras.metrics.Metric object.
model_selection_operator	Retrieve the operator used for model selection.
name	Retrieve the metric name.
sanitized_name	all / are _.

__init__(*name='d_loss'*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashp*

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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v*

Bases: *ashpy.metrics.metric.Metric*

Generator loss value.

Methods

__init__([name, model_selection_operator, ...])	Initialize the Metric.
update_state(context)	Update the internal state of the metric, using the information from the context object.

Attributes

best_folder	Retrieve the folder used to save the best model when doing model selection.
best_model_sel_file	Retrieve the path to JSON file containing the measured performance of the best model.
logdir	Retrieve the log directory.
metric	Retrieve the tf.keras.metrics.Metric object.
model_selection_operator	Retrieve the operator used for model selection.
name	Retrieve the metric name.
sanitized_name	all / are _.

__init__ (*name='g_loss'*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashp*
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.

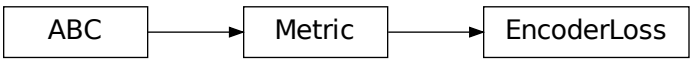
update_state (*context*)
Update the internal state of the metric, using the information from the context object.

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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3*
Bases: *ashpy.metrics.metric.Metric*

Encoder Loss value.

Methods

__init__([name, model_selection_operator, ...])	Initialize the Metric.
update_state(context)	Update the internal state of the metric, using the information from the context object.

Attributes

best_folder	Retrieve the folder used to save the best model when doing model selection.
best_model_sel_file	Retrieve the path to JSON file containing the measured performance of the best model.
logdir	Retrieve the log directory.
metric	Retrieve the tf.keras.metrics.Metric object.
model_selection_operator	Retrieve the operator used for model selection.
name	Retrieve the metric name.
sanitized_name	all / are _.

__init__(*name='e_loss'*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashp*
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

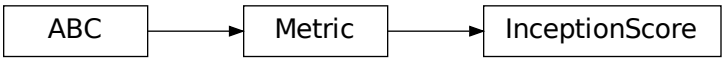
update_state(*context*)
Update the internal state of the metric, using the information from the context object.

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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/*
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 GANs[1].

Methods

__init__(inception[, name, ...])	Initialize the Metric.
get_or_train_inception(dataset, name, ...[, ...])	Restore or train (and save) the Inception model.
inception_score(images)	Compute the Inception Score.
update_state(context)	Update the internal state of the metric, using the information from the context object.

Attributes

best_folder	Retrieve the folder used to save the best model when doing model selection.
best_model_sel_file	Retrieve the path to JSON file containing the measured performance of the best model.
logdir	Retrieve the log directory.
metric	Retrieve the tf.keras.metrics.Metric object.
model_selection_operator	Retrieve the operator used for model selection.
name	Retrieve the metric name.
sanitized_name	all / are _.

__init__(*inception*, *name='inception_score'*, *model_selection_operator=<built-in function gt>*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkout*

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 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

4.7.8 EncodingAccuracy

Inheritance Diagram

---

class ashpy.metrics.gan.**EncodingAccuracy**(*classifier*, *name='encoding_accuracy'*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkou*

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 classified (on the dataset of x).

Methods

__init__(classifier[, name, ...])	Measure the Generator and Encoder performance together.
update_state(context)	Update the internal state of the metric, using the information from the context object.

Attributes

best_folder	Retrieve the folder used to save the best model when doing model selection.
best_model_sel_file	Retrieve the path to JSON file containing the measured performance of the best model.
logdir	Retrieve the log directory.
metric	Retrieve the tf.keras.metrics.Metric object.
model_selection_operator	Retrieve the operator used for model selection.
name	Retrieve the metric name.
sanitized_name	all / are _.

__init__(*classifier*, *name='encoding_accuracy'*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/docs/source/log'*)

Measure the Generator and Encoder performance together.

This is done by classifying: *G(E(x)), y* using a pre-trained classified (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=<function lt>*, *logdir='/home/docs/checko*, *resolution=128*, *resolution_min=16*, *patches_per_image=64*, *patch_size=7*, *random_sampling_count=1*, *random_projection_dim=147*, *use_svd=False*)

Bases: *ashpy.metrics.metric.Metric*

Sliced Wasserstein Distance.

Used as metric in Progressive Growing of GANs[1].

Methods

__init__([name, model_selection_operator, ...])	Initialize the Metric.
log(step)	Log the SWD mean and each sub-metric.
model_selection(checkpoint, global_step)	Perform model selection for each sub-metric.
reset_states()	Reset the state of the metric and the state of each child metric.
update_state(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

best_folder	Retrieve the folder used to save the best model when doing model selection.
best_model_sel_file	Retrieve the path to JSON file containing the measured performance of the best model.
logdir	Retrieve the log directory.
metric	Retrieve the tf.keras.metrics.Metric object.
model_selection_operator	Retrieve the operator used for model selection.
name	Retrieve the metric name.
sanitized_name	all / are _.

__init__(*name='SWD'*, *model_selection_operator=<built-in function lt>*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/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

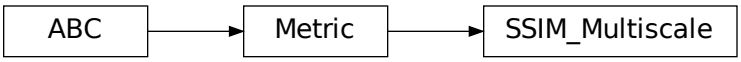
update_state(*context*)
Update the internal state of the metric, using the information from the context object.

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='/home/docs/checkouts/readthedocs.org/user_buil*
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 assessment[1]

Methods

__init__([name, model_selection_operator, ...])	Initialize the Metric.
split_batch(batch)	Split a batch along axis 0 into two tensors having the same size.
update_state(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

best_folder	Retrieve the folder used to save the best model when doing model selection.
best_model_sel_file	Retrieve the path to JSON file containing the measured performance of the best model.
logdir	Retrieve the log directory.
metric	Retrieve the tf.keras.metrics.Metric object.
model_selection_operator	Retrieve the operator used for model selection.
name	Retrieve the metric name.
sanitized_name	all / are _.

__init__(*name='SSIM_Multiscale'*, *model_selection_operator=<built-in function lt>*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/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** (*int*) – 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

classifier	The classification metrics.
gan	GAN metrics.
metric	Metric is the abstract class that every ash metric must implement.
sliced_wasserstein_metric	Sliced Wasserstein Distance metric.
ssim_multiscale	Multiscale Structural Similarity metric.

4.7.11 classifier

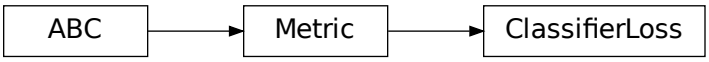
The classification metrics.

Classes

ClassifierLoss	A handy way to measure the classification loss.
ClassifierMetric	Wrap a metric using *argmax* to extract predictions out of a classifier's output.

ClassifierLoss

Inheritance Diagram



class ashpy.metrics.classifier.**ClassifierLoss**(*name='loss'*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/*

Bases: *ashpy.metrics.metric.Metric*

A handy way to measure the classification loss.

Methods

__init__([name, model_selection_operator, ...])	Initialize the Metric.
update_state(context)	Update the internal state of the metric, using the information from the context object.

Attributes

best_folder	Retrieve the folder used to save the best model when doing model selection.
best_model_sel_file	Retrieve the path to JSON file containing the measured performance of the best model.
logdir	Retrieve the log directory.
metric	Retrieve the tf.keras.metrics.Metric object.
model_selection_operator	Retrieve the operator used for model selection.
name	Retrieve the metric name.
sanitized_name	all / are _.

__init__(*name='loss'*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/c*
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

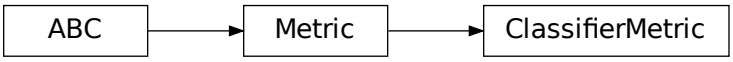
update_state(*context*)
Update the internal state of the metric, using the information from the context object.

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='/home/docs/checkouts/readthedocs.org/user_builds/ash*
processing_predictions=None)

Bases: *ashpy.metrics.metric.Metric*

Wrap a metric using *argmax* to extract predictions out of a classifier's output.

Methods

__init__(metric[, model_selection_operator, ...])	Initialize the Metric.
update_state(context)	Update the internal state of the metric, using the information from the context object.

Attributes

best_folder	Retrieve the folder used to save the best model when doing model selection.
best_model_sel_file	Retrieve the path to JSON file containing the measured performance of the best model.
logdir	Retrieve the log directory.
metric	Retrieve the tf.keras.metrics.Metric object.
model_selection_operator	Retrieve the operator used for model selection.
name	Retrieve the metric name.
sanitized_name	all / are _.

__init__(*metric*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checko*
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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/*

Bases: *ashpy.metrics.metric.Metric*

A handy way to measure the classification loss.

__init__(*name='loss'*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/c*
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='/home/docs/checkouts/readthedocs.org/user_builds/ash* *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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checko*
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

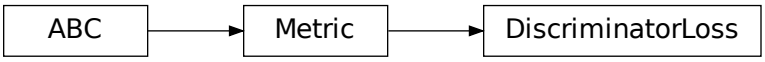
GAN metrics.

Classes

DiscriminatorLoss	The Discriminator loss value.
EncoderLoss	Encoder Loss value.
EncodingAccuracy	Generator and Encoder accuracy performance.
GeneratorLoss	Generator loss value.
InceptionScore	Inception Score Metric.

DiscriminatorLoss

Inheritance Diagram



---

class ashpy.metrics.gan.**DiscriminatorLoss**(*name='d_loss'*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/check*

Bases: *ashpy.metrics.metric.Metric*

The Discriminator loss value.

Methods

__init__([name, model_selection_operator, ...])	Initialize the Metric.
update_state(context)	Update the internal state of the metric, using the information from the context object.

Attributes

best_folder	Retrieve the folder used to save the best model when doing model selection.
best_model_sel_file	Retrieve the path to JSON file containing the measured performance of the best model.
logdir	Retrieve the log directory.
metric	Retrieve the tf.keras.metrics.Metric object.
model_selection_operator	Retrieve the operator used for model selection.
name	Retrieve the metric name.
sanitized_name	all / are _.

__init__(*name='d_loss'*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashp*

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*)

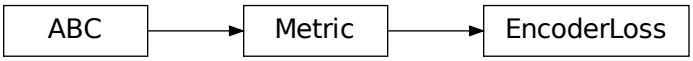
Update the internal state of the metric, using the information from the context object.

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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3*

Bases: *ashpy.metrics.metric.Metric*

Encoder Loss value.

Methods

__init__([name, model_selection_operator, ...])	Initialize the Metric.
update_state(context)	Update the internal state of the metric, using the information from the context object.

Attributes

best_folder	Retrieve the folder used to save the best model when doing model selection.
best_model_sel_file	Retrieve the path to JSON file containing the measured performance of the best model.
logdir	Retrieve the log directory.
metric	Retrieve the tf.keras.metrics.Metric object.
model_selection_operator	Retrieve the operator used for model selection.
name	Retrieve the metric name.
sanitized_name	all / are _.

__init__ (*name='e_loss'*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashp*
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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkou*
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 classified (on the dataset of x).

Methods

__init__(classifier[, name, ...])	Measure the Generator and Encoder performance together.
update_state(context)	Update the internal state of the metric, using the information from the context object.

Attributes

`best_folder`	Retrieve the folder used to save the best model when doing model selection.
`best_model_sel_file`	Retrieve the path to JSON file containing the measured performance of the best model.
`logdir`	Retrieve the log directory.
`metric`	Retrieve the `tf.keras.metrics.Metric` object.
`model_selection_operator`	Retrieve the operator used for model selection.
`name`	Retrieve the metric name.
`sanitized_name`	all / are _.

__init__(*classifier*, *name='encoding_accuracy'*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/docs/source/log'*)

Measure the Generator and Encoder performance together.

This is done by classifying: *G(E(x)), y* using a pre-trained classified (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*)

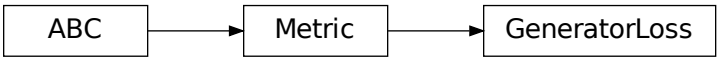
Update the internal state of the metric, using the information from the context object.

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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v*

Bases: *ashpy.metrics.metric.Metric*

Generator loss value.

Methods

__init__([name, model_selection_operator, ...])	Initialize the Metric.
update_state(context)	Update the internal state of the metric, using the information from the context object.

Attributes

best_folder	Retrieve the folder used to save the best model when doing model selection.
best_model_sel_file	Retrieve the path to JSON file containing the measured performance of the best model.
logdir	Retrieve the log directory.
metric	Retrieve the tf.keras.metrics.Metric object.
model_selection_operator	Retrieve the operator used for model selection.
name	Retrieve the metric name.
sanitized_name	all / are _.

__init__(*name='g_loss'*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashp*

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.

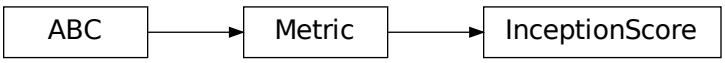
update_state(*context*)
Update the internal state of the metric, using the information from the context object.

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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/*

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 GANs[1].

Methods

__init__(inception[, name, ...])	Initialize the Metric.
get_or_train_inception(dataset, name, ...[, ...])	Restore or train (and save) the Inception model.
inception_score(images)	Compute the Inception Score.
update_state(context)	Update the internal state of the metric, using the information from the context object.

Attributes

best_folder	Retrieve the folder used to save the best model when doing model selection.
best_model_sel_file	Retrieve the path to JSON file containing the measured performance of the best model.
logdir	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

`metric`	Retrieve the `tf.keras.metrics.Metric` object.
`model_selection_operator`	Retrieve the operator used for model selection.
`name`	Retrieve the metric name.
`sanitized_name`	all / are _.

__init__(*inception*, *name='inception_score'*, *model_selection_operator=<built-in function gt>*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkout*

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 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

class ashpy.metrics.gan.**DiscriminatorLoss**(*name='d_loss'*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/check*

Bases: *ashpy.metrics.metric.Metric*

The Discriminator loss value.

__init__(*name='d_loss'*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashp*
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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3*

Bases: *ashpy.metrics.metric.Metric*

Encoder Loss value.

__init__(*name='e_loss'*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashp*
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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkou*

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 classified (on the dataset of x).

__init__(*classifier*, *name='encoding_accuracy'*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/docs/source/log'*)
Measure the Generator and Encoder performance together.

This is done by classifying: *G(E(x)), y* using a pre-trained classified (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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v*

Bases: *ashpy.metrics.metric.Metric*

Generator loss value.

__init__(*name='g_loss'*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashp*
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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/*

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 GANs[1].

__init__(*inception*, *name='inception_score'*, *model_selection_operator=<built-in function gt>*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkout*
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

Metric	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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/*

Bases: abc.ABC

Metric is the abstract class that every ash Metric must implement.

AshPy Metrics wrap and extend Keras Metrics.

Methods

__init__(name, metric[, ...])	Initialize the Metric object.
json_read(filename)	Read a JSON file.
json_write(filename, what_to_write)	Write inside the specified JSON file the mean and stddev.
log(step)	Log the metric.
model_selection(checkpoint, global_step)	Perform model selection.
reset_states()	Reset the state of the metric.
result()	Get the result of the metric.
update_state(context)	Update the internal state of the metric, using the information from the context object.

Attributes

best_folder	Retrieve the folder used to save the best model when doing model selection.
best_model_sel_file	Retrieve the path to JSON file containing the measured performance of the best model.
logdir	Retrieve the log directory.
metric	Retrieve the `tf.keras.metrics.Metric` object.
model_selection_operator	Retrieve the operator used for model selection.
name	Retrieve the metric name.
sanitized_name	all / are _.

__init__ (*name*, *metric*, *model_selection_operator=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/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 `str`

best_model_sel_file
Retrieve the path to JSON file containing the measured performance of the best model.

Return type str

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 str

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 None

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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/*

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='/home/docs/checkouts/readthedocs.org/user_builds/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 str

best_model_sel_file
Retrieve the path to JSON file containing the measured performance of the best model.

Return type str

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 str

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 None

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

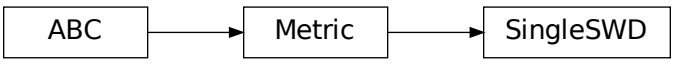
Sliced Wasserstein Distance metric.

Classes

SingleSWD	SlicedWassersteinDistance for a certain level of the pyramid.
SlicedWassersteinDistance	Sliced Wasserstein Distance.

SingleSWD

Inheritance Diagram



---

class ashpy.metrics.sliced_wasserstein_metric.**SingleSWD**(*model_selection_operator=<built-in function lt>*, *logdir='/home/docs/checkouts/readthedocs.org/use*
level_of_pyramid=0, *real_or_fake='fake'*)

Bases: *ashpy.metrics.metric.Metric*

SlicedWassersteinDistance for a certain level of the pyramid.

Methods

__init__([model_selection_operator, logdir, ...])	Initialize the Metric.

Continued on next page

Table 204 – continued from previous page

update_state(context, score)	Update the internal state of the metric, using the information from the context object.

Attributes

best_folder	Retrieve the folder used to save the best model when doing model selection.
best_model_sel_file	Retrieve the path to JSON file containing the measured performance of the best model.
logdir	Retrieve the log directory.
metric	Retrieve the tf.keras.metrics.Metric object.
model_selection_operator	Retrieve the operator used for model selection.
name	Retrieve the metric name.
sanitized_name	all / are _.

__init__(*model_selection_operator=<built-in function lt>*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy*
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=<function lt>*, *logdir='/home/docs/checko*, *resolution=128*, *resolution_min=16*, *patches_per_image=64*, *patch_size=7*, *random_sampling_count=1*, *random_projection_dim=147*, *use_svd=False*)

Bases: *ashpy.metrics.metric.Metric*

Sliced Wasserstein Distance.

Used as metric in Progressive Growing of GANs[1].

Methods

__init__([name, model_selection_operator, ...])	Initialize the Metric.
log(step)	Log the SWD mean and each sub-metric.
model_selection(checkpoint, global_step)	Perform model selection for each sub-metric.
reset_states()	Reset the state of the metric and the state of each child metric.
update_state(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

best_folder	Retrieve the folder used to save the best model when doing model selection.
best_model_sel_file	Retrieve the path to JSON file containing the measured performance of the best model.
logdir	Retrieve the log directory.
metric	Retrieve the tf.keras.metrics.Metric object.
model_selection_operator	Retrieve the operator used for model selection.
name	Retrieve the metric name.
sanitized_name	all / are _.

__init__(*name='SWD'*, *model_selection_operator=<built-in function lt>*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/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='/home/docs/checkouts/readthedocs.org/use*, *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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy*, *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=<built-in function lt>*, *logdir='/home/docs/checko*
resolution=128, *resolution_min=16*, *patches_per_image=64*, *patch_size=7*, *random_sampling_count=1*, *random_projection_dim=147*, *use_svd=False*)

Bases: *ashpy.metrics.metric.Metric*

Sliced Wasserstein Distance.

Used as metric in Progressive Growing of GANs[1].

__init__(*name='SWD'*, *model_selection_operator=<built-in function lt>*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/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.

[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

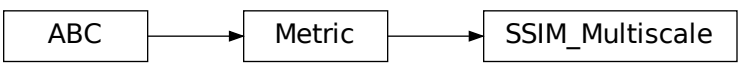
Multiscale Structural Similarity metric.

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='/home/docs/checkouts/readthedocs.org/user_buil*, *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 assessment[1]

Methods

__init__([name, model_selection_operator, ...])	Initialize the Metric.
split_batch(batch)	Split a batch along axis 0 into two tensors having the same size.
update_state(context)	Update the internal state of the metric, using the information from the context object.

Attributes

best_folder	Retrieve the folder used to save the best model when doing model selection.
best_model_sel_file	Retrieve the path to JSON file containing the measured performance of the best model.
logdir	Retrieve the log directory.
metric	Retrieve the tf.keras.metrics.Metric object.
model_selection_operator	Retrieve the operator used for model selection.
name	Retrieve the metric name.
sanitized_name	all / are _.

__init__(*name='SSIM_Multiscale'*, *model_selection_operator=<built-in function lt>*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/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 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** (*int*) – 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='/home/docs/checkouts/readthedocs.org/user_buil*, *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 assessment[1]

__init__(*name='SSIM_Multiscale'*, *model_selection_operator=<built-in function lt>*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/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** (*int*) – 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

convolutional	Collection of Convolutional Models constructors.
fc	Collection of Fully Connected Models constructors.

4.8.1 convolutional

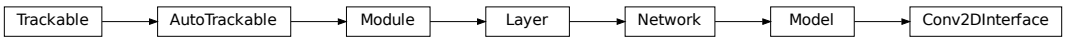
Collection of Convolutional Models constructors.

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

__init__()	Primitive Interface to be used by all *ashpy.models*.
call(inputs[, training, return_features])	Execute the model on input data.

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* APIs.

Continued on next page

Table 214 – continued from previous page

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 *tf.name_scope* 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 *updates* 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__()

Primitive Interface to be used by all *ashpy.models*.

Declares the *self.model_layers* list.

static _get_layer_spec(*initial_filers*, *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_filers** (*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_filers=16,
    filters_cap=128,
    input_res=(512, 256),
    target_res=(32, 16)
)
print([s for s in spec])

spec = T._get_layer_spec(
    initial_filers=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_filers=128,
    filters_cap=16,
    input_res=(32, 16),
    target_res=(512, 256)
)
print([s for s in spec])

spec = T._get_layer_spec(
    initial_filers=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.

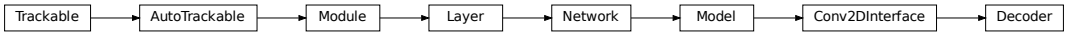
Returns tf.Tensor – The model output.

Decoders

decoders.Decoder	Primitive Model for all decoder (i.e., transpose convolution) based architecture.
decoders.FCNNDecoder	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

__init__(layer_spec_input_res, . . . [, . . .])	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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* from all layers that are stateful.

Continued on next page

Table 217 – continued from previous page

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*, *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*)

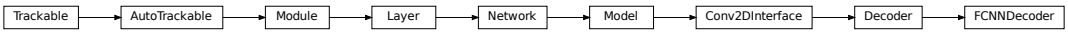
Ingest the tf.keras.Model inputs and prepare them for *_add_building_block()*.

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.LeakyR*

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

__init__(layer_spec_input_res, . . . [, . . .])	Build a Fully Convolutional 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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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*, *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-

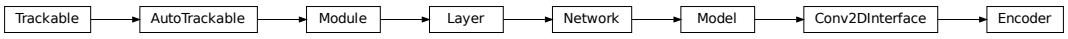
sors.

Encoders

encoders.Encoder	Primitive Model for all encoder (i.e., convolution) based architecture.
encoders.FCNNEncoder	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*, *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,
    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

__init__(layer_spec_input_res, . . . [, . . .])	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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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

`run_eagerly`	Settable attribute indicating whether the model should run eagerly.
`sample_weights`	
`state_updates`	Returns the *updates* 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*, *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 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(*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 1x1xencoding_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

__init__(layer_spec_input_res, ...)	Instantiate the FCNNDecoder.

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.

Continued on next page

Table 224 – continued from previous page

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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*)

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

autoencoders.Autoencoder	Primitive Model for all convolutional autoencoders.
autoencoders.FCNNAutoencoder	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

__init__(layer_spec_input_res, ...)	Instantiate the `BaseAutoEncoder`.
call(inputs[, training])	Execute the model on input data.

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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.

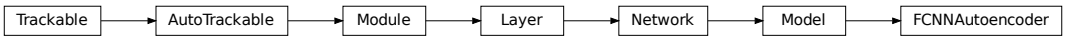
Parameters

- **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*, *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,
)

encoding, reconstruction = autoencoder(tf.zeros((1, 64, 64, 3)))
print(encoding.shape)
print(reconstruction.shape)
```

```
```

Methods

__init__(layer_spec_input_res, ...)	Instantiate the `FCNNBaseAutoEncoder`.
call(inputs[, training])	Execute the model on input data.

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 *Layer*.
`metrics`	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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.

Continued on next page

Table 229 – continued from previous page

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 *updates* 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 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.

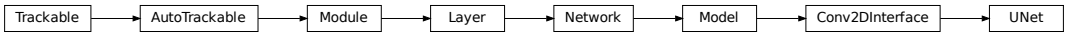
Returns *(encoding, reconstruction)* – Pair of tensors.

UNet

unet.UNet	UNet Architecture.
unet.SUNet	Semantic UNet.
unet.FUNet	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

__init__(input_res, min_res, kernel_size, ...)	Initialize the UNet.
call(inputs[, training])	Forward pass of the UNet model.
get_decoder_block(filters[, use_bn, ...])	Return a block to be used in the decoder part of the UNET.
get_encoder_block(filters[, use_bn, ...])	Return a block to be used in the encoder part of the UNET.

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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*, *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*)

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/v0.3.0/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

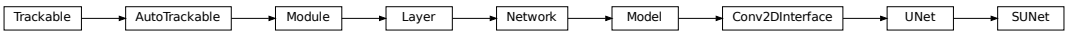
- **filters** – number of filters.
- **use_bn** – whether to use batch normalization.

- **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*, *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'>*, *use_attention=False*)

Bases: *ashpy.models.convolutional.unet.UNet*

Semantic UNet.

Methods

__init__(input_res, min_res, kernel_size, ...)	Build the Semantic UNet model.

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* APIs.

Continued on next page

Table 234 – continued from previous page

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 *tf.name_scope* 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 *updates* 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*, *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'>*, *use_attention=False*)

Build the Semantic UNet model.

ashpy.models.convolutional.unet.FUNet

ashpy.models.convolutional.unet.**FUNet**(*input_res*, *min_res*, *kernel_size*, *initial_filters*, *filters_cap*, *channels*, *input_channels=3*, *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'>*, *last_activation=<function tanh>*, *use_attention=False*)

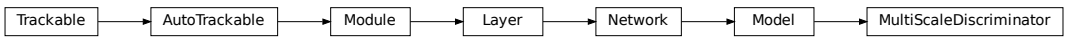
Functional UNET Implementation.

Discriminators

discriminators.MultiScaleDiscriminator	Multi-Scale discriminator.
discriminators.PatchDiscriminator	Pix2Pix discriminator.

MultiScaleDiscriminator

Inheritance Diagram



---

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.adv normalization_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

__init__(input_res, min_res, kernel_size, ...)	Multi Scale Discriminator.
build_discriminator(input_res)	Build a single discriminator using parameters defined in this object.
call(inputs[, training, return_features])	Forward pass of the Multi Scale Discriminator.

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* instance for this class.
non_trainable_variables	
non_trainable_weights	
outbound_nodes	Deprecated, do NOT use! Only for compatibility with external Keras.

Continued on next page

Table 237 – continued from previous page

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 *updates* 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*, *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 Pix2PixHD[1] .

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

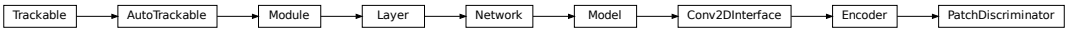
([tf.Tensor]) –

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*, *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*)

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)
```

Methods

__init__(input_res, min_res, kernel_size, ...)	Patch Discriminator used by pix2pix.
call(inputs[, training, return_features])	Forward pass of the PatchDiscriminator.

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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.

Continued on next page

Table 239 – continued from previous page

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 *updates* 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*, *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 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(*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

pix2pixhd.LocalEnhancer	Local Enhancer module of the Pix2PixHD architecture.
pix2pixhd.GlobalGenerator	Global Generator from pix2pixHD paper.

LocalEnhancer

Inheritance Diagram

---

class ashpy.models.convolutional.pix2pixhd.**LocalEnhancer**(*input_res=512*, *min_res=64*, *initial_filters=64*, *filters_cap=512*, *channels=3*, *normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceNor*
non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.Re
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

__init__([input_res, min_res, ...])	Build the LocalEnhancer module of the Pix2PixHD architecture.
call(inputs[, training])	Call the LocalEnhancer model.

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* from all layers that are stateful.
stateful	
submodules	Sequence of all sub-modules.
trainable	

Continued on next page

Table 242 – continued from previous page

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*, *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 GANs[2] 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

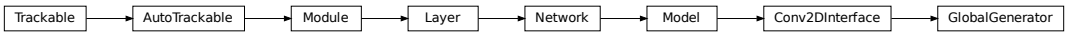
(tf.Tensor) –

[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*, *initial_filters=64*, *filters_cap=512*, *channels=3*, *normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceN*, *non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.*, *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

__init__([input_res, min_res, ...])	Global Generator from Pix2PixHD.
call(inputs[, training])	Call of the Pix2Pix HD model.

Attributes

activity_regularizer	Optional regularizer function for the output of this layer.
dtype	
dynamic	

Continued on next page

Table 244 – continued from previous page

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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*, *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

autoencoders	Collection of Fully Convolutional Autoencoders.
discriminators	Convolutional Discriminators.
decoders	Collection of Decoders (i.e., GANs' Generators) models.
encoders	Collection of Encoders (i.e., GANs' Discriminators) models.
interfaces	Primitive Convolutional interfaces.
unet	UNET implementations.
pix2pixhd	Pix2Pix HD Implementation.

autoencoders

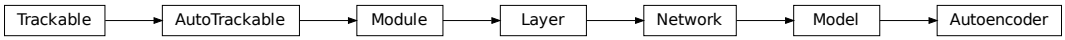
Collection of Fully Convolutional Autoencoders.

Classes

Autoencoder	Primitive Model for all convolutional autoencoders.
FCNNAutoencoder	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

__init__(layer_spec_input_res, ...)	Instantiate the `BaseAutoEncoder`.
call(inputs[, training])	Execute the model on input data.

Attributes

`activity_regularizer`	Optional regularizer function for the output of this layer.
`dtype`	

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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.

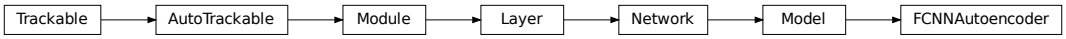
Parameters

- **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*, *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,
)

encoding, reconstruction = autoencoder(tf.zeros((1, 64, 64, 3)))
print(encoding.shape)
print(reconstruction.shape)
```

Methods

__init__(layer_spec_input_res, ...)	Instantiate the `FCNNBaseAutoEncoder`.
call(inputs[, training])	Execute the model on input data.

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 *Layer*.
`metrics`	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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 `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

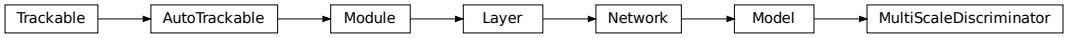
Convolutional Discriminators.

Classes

MultiScaleDiscriminator	Multi-Scale discriminator.
PatchDiscriminator	Pix2Pix discriminator.

MultiScaleDiscriminator

Inheritance Diagram



---

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.ad*
normalization_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,
                                                  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

__init__(input_res, min_res, kernel_size, ...)	Multi Scale Discriminator.
build_discriminator(input_res)	Build a single discriminator using parameters defined in this object.
call(inputs[, training, return_features])	Forward pass of the Multi Scale Discriminator.

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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.

Continued on next page

Table 253 – continued from previous page

sample_weights	
state_updates	Returns the *updates* 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*, *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 Pix2PixHD[1] .

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

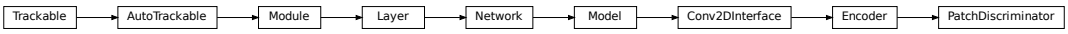
([tf.Tensor]) –

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*, *kernel_size*, *initial_filters*, *filters_cap*, *use_dropout=True*, *dropout_prob=0.3*, *non_linearity=<class 'tensorflow.python.keras.layers.advanced_*, *normalization_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)
```

Methods

__init__(input_res, min_res, kernel_size, ...)	Patch Discriminator used by pix2pix.
call(inputs[, training, return_features])	Forward pass of the PatchDiscriminator.

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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.

Continued on next page

Table 255 – continued from previous page

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 *updates* 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*, *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 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 (*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.ad*
normalization_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,
                                                  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 Pix2PixHD[1] .

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*, *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*)

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 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(*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

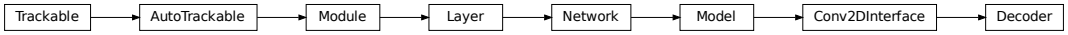
Collection of Decoders (i.e., GANs' Generators) models.

Classes

Decoder	Primitive Model for all decoder (i.e., transpose convolution) based architecture.
FCNNDecoder	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

__init__(layer_spec_input_res, ... [, ...])	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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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*, *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.LeakyR*

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

__init__(layer_spec_input_res, . . . [, . . .])	Build a Fully Convolutional Decoder.

Attributes

activity_regularizer	Optional regularizer function for the output of this layer.
dtype	
dynamic	

Continued on next page

Table 260 – continued from previous page

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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*, *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*, *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!
```

__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.

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.LeakyR*

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

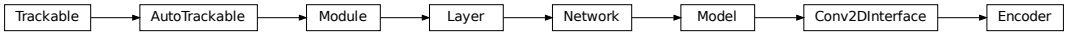
Collection of Encoders (i.e., GANs' Discriminators) models.

Classes

Encoder	Primitive Model for all encoder (i.e., convolution) based architecture.
FCNNEncoder	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*, *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,
    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

__init__(layer_spec_input_res, . . . [, . . .])	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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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*, *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 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(*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 1x1xencoding_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

__init__(layer_spec_input_res, ...)	Instantiate the `FCNNDecoder`.

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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	

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 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(*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*, *encoding_dimension*)

Bases: *ashpy.models.convolutional.encoders.Encoder*

Fully Convolutional Encoder.

Output a 1x1xencoding_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

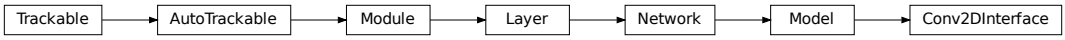
Primitive Convolutional interfaces.

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

__init__()	Primitive Interface to be used by all *ashpy.models*.
call(inputs[, training, return_features])	Execute the model on input data.

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* APIs.

Continued on next page

Table 268 – continued from previous page

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 *tf.name_scope* 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 *updates* 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__()

Primitive Interface to be used by all *ashpy.models*.

Declares the *self.model_layers* list.

static _get_layer_spec(*initial_filers*, *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_filers** (*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_filers=16,
    filters_cap=128,
    input_res=(512, 256),
    target_res=(32, 16)
)
print([s for s in spec])

spec = T._get_layer_spec(
    initial_filers=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_filers=128,
    filters_cap=16,
    input_res=(32, 16),
    target_res=(512, 256)
)
print([s for s in spec])

spec = T._get_layer_spec(
    initial_filers=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_filers*, *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_filers** (*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_filers=16,
    filters_cap=128,
    input_res=(512, 256),
    target_res=(32, 16)
)
print([s for s in spec])

spec = T._get_layer_spec(
    initial_filers=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_filers=128,
    filters_cap=16,
    input_res=(32, 16),
    target_res=(512, 256)
)
print([s for s in spec])

spec = T._get_layer_spec(
    initial_filers=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*, *initial_filters*, *filters_cap*, *channels*, *input_channels=3*, *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'>*, *last_activation=<function tanh>*, *use_attention=False*)

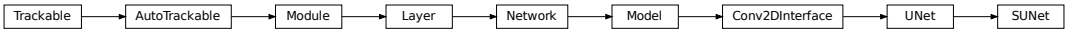
Functional UNET Implementation.

Classes

SUNet	Semantic UNet.
UNet	UNet Architecture.

SUNet

Inheritance Diagram



---

class ashpy.models.convolutional.unet.**SUNet**(*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'>*, *use_attention=False*)

Bases: *ashpy.models.convolutional.unet.UNet*

Semantic UNet.

Methods

__init__(input_res, min_res, kernel_size, ...)	Build the Semantic UNet model.

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	

Continued on next page

Table 272 – continued from previous page

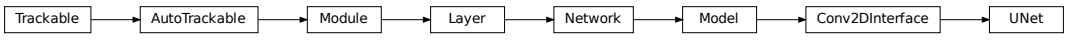
losses	Losses which are associated with this *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* APIs.
metrics_names	Returns the model's display labels for all outputs.
name	Returns the name of this module as passed or determined in the ctor.
name_scope	Returns a *tf.name_scope* 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 *updates* 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*, *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'>*, *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

__init__(input_res, min_res, kernel_size, ...)	Initialize the UNet.
call(inputs[, training])	Forward pass of the UNet model.
get_decoder_block(filters[, use_bn, ...])	Return a block to be used in the decoder part of the UNET.
get_encoder_block(filters[, use_bn, ...])	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

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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*, *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*)

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/v0.3.0/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*, *initial_filters*, *filters_cap*, *channels*, *input_channels=3*, *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'>*, *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*, *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'>*, *use_attention=False*)

Bases: *ashpy.models.convolutional.unet.UNet*

Semantic UNet.

__init__(*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'>*, *use_attention=False*)

Build the Semantic UNet model.

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,
```

(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*, *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*)

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/v0.3.0/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]

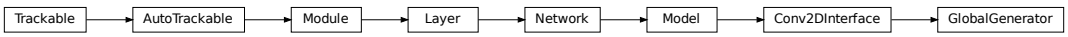
Global Generator + Local Enhancer

Classes

GlobalGenerator	Global Generator from pix2pixHD paper.
LocalEnhancer	Local Enhancer module of the Pix2PixHD architecture.
ResNetBlock	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*, *initial_filters=64*, *filters_cap=512*, *channels=3*, *normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceN*, *non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.*, *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

__init__([input_res, min_res, ...])	Global Generator from Pix2PixHD.
call(inputs[, training])	Call of the Pix2Pix HD model.

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* instance for this class.
non_trainable_variables	
non_trainable_weights	

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 *updates* 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*, *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

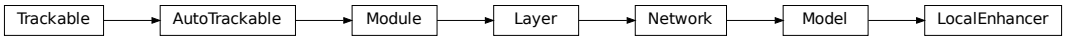
- **inputs** – input tensor(s).

- **training** – If True training phase.

Returns Tuple – Generated images.

LocalEnhancer

Inheritance Diagram



---

class ashpy.models.convolutional.pix2pixhd.**LocalEnhancer**(*input_res=512*, *min_res=64*, *initial_filters=64*, *filters_cap=512*, *channels=3*, *normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceNor*
non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.Re
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

__init__([input_res, min_res, ...])	Build the LocalEnhancer module of the Pix2PixHD architecture.
call(inputs[, training])	Call the LocalEnhancer model.

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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*, *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 GANs[2] 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

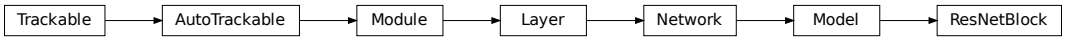
(tf.Tensor) –

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.InstanceNorma*
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

__init__(filters[, normalization_layer, ...])	Build the ResNet block composed by num_blocks.
call(inputs[, training])	Forward pass.

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* instance for this class.
non_trainable_variables	

Continued on next page

Table 281 – continued from previous page

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 *updates* 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*, *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*, *initial_filters=64*, *filters_cap=512*, *channels=3*, *normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceN*, *non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.*, *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*, *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.

class ashpy.models.convolutional.pix2pixhd.**LocalEnhancer**(*input_res=512*, *min_res=64*, *initial_filters=64*, *filters_cap=512*, *channels=3*, *normalization_layer=<class 'ashpy.layers.instance_normalization.InstanceNor*
non_linearity=<class 'tensorflow.python.keras.layers.advanced_activations.Re
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*, *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 GANs[2] 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.InstanceNorma*
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

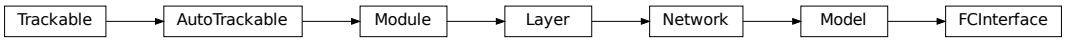
Collection of Fully Connected Models constructors.

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

__init__()	Primitive Interface to be used by all *ashpy.models*.
call(inputs[, training])	Execute the model on input data.

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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__()

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.

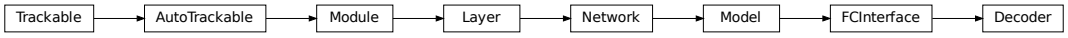
Returns :py_class:‘tf.Tensor‘

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

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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.

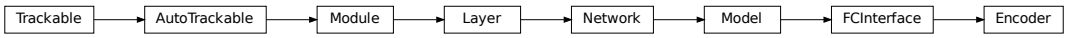
Returns None

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

__init__(hidden_units, output_shape)	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.

Continued on next page

Table 290 – continued from previous page

layers	
losses	Losses which are associated with this *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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`.

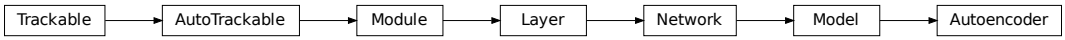
Returns `None`

Autoencoders

autoencoders.Autoencoder	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

__init__(hidden_units, encoding_dimension, ...)	Instantiate the Decoder.
call(inputs[, training])	Execute the model on input data.

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.

Continued on next page

Table 293 – continued from previous page

input_spec	Gets the network's input specs.
layers	
losses	Losses which are associated with this *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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*, *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

autoencoders	Collection of Fully Connected Autoencoders.
decoders	Collection of Decoders (i.e., GANs' Discriminators) models.
encoders	Collection of Encoders (i.e., GANs' Discriminators) models.
interfaces	Primitive Fully Connected interfaces.

autoencoders

Collection of Fully Connected Autoencoders.

Classes

Autoencoder	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

__init__(hidden_units, encoding_dimension, ...)	Instantiate the `Decoder`.
call(inputs[, training])	Execute the model on input data.

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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*, *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

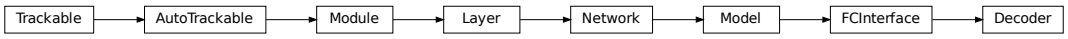
Collection of Decoders (i.e., GANs' Discriminators) models.

Classes

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

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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

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

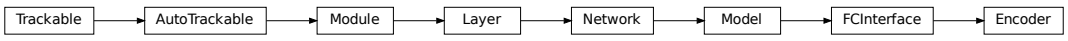
Collection of Encoders (i.e., GANs' Discriminators) models.

Classes

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

__init__(hidden_units, output_shape)	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 *Layer*.
`metrics`	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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

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

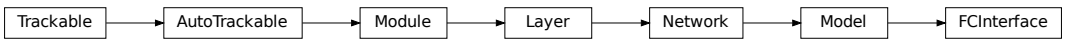
Primitive Fully Connected interfaces.

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

__init__()	Primitive Interface to be used by all *ashpy.models*.
call(inputs[, training])	Execute the model on input data.

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 *Layer*.
metrics	Returns the model's metrics added using *compile*, *add_metric* 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 *tf.name_scope* 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 *updates* 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.

Continued on next page

Table 306 – continued from previous page

weights	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‘

class ashpy.models.fc.interfaces.**FCInterface**
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._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‘

gans	GANs Models.

4.8.3 gans

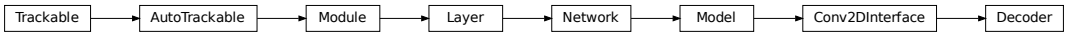
GANs Models.

Generators

ConvGenerator	alias of *ashpy.models.convolutional.decoders.Decoder*
DenseGenerator	alias of *ashpy.models.fc.decoders.Decoder*

ConvGenerator

Inheritance Diagram

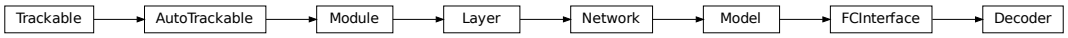


---

ashpy.models.gans.**ConvGenerator**
alias of *ashpy.models.convolutional.decoders.Decoder*

DenseGenerator

Inheritance Diagram



---

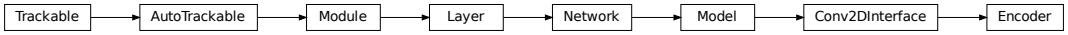
ashpy.models.gans.**DenseGenerator**
alias of *ashpy.models.fc.decoders.Decoder*

Discriminators

ConvDiscriminator	alias of *ashpy.models.convolutional.encoders.Encoder*
DenseDiscriminator	alias of *ashpy.models.fc.encoders.Encoder*

ConvDiscriminator

Inheritance Diagram

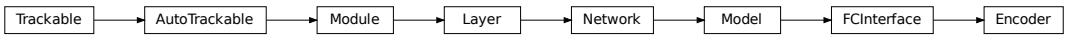


---

ashpy.models.gans.**ConvDiscriminator**
alias of *ashpy.models.convolutional.encoders.Encoder*

DenseDiscriminator

Inheritance Diagram



---

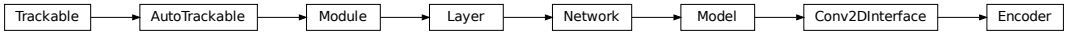
ashpy.models.gans.**DenseDiscriminator**
alias of *ashpy.models.fc.encoders.Encoder*

Encoders

ConvEncoder	alias of *ashpy.models.convolutional.encoders.Encoder*
DenseEncoder	alias of *ashpy.models.fc.encoders.Encoder*

ConvEncoder

Inheritance Diagram

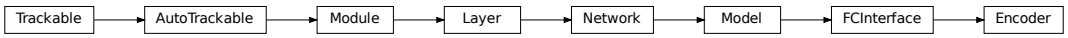


---

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

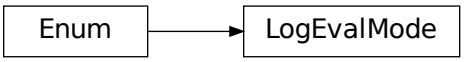
Various modalities used to configure certain ash behaviours.

Classes

LogEvalMode	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

Restorer	*Restorer* provide a way to restore objects from tf.train.Checkpoint.
AdversarialRestorer	Convenience *Restorer* for ease of use with the AdversairalTrainer.
AdversarialEncoderRestorer	Convenience *Restorer* for ease of use with the EncoderTrainer.
ClassifierRestorer	Convenience *Restorer* for ease of use with the ClassifierTrainer.

4.10.1 Restorer

Inheritance Diagram

class ashpy.restorers.**Restorer**(*logdir='log'*, *ckpts_dir='ckpts'*, *expect_partial=True*)
Bases: object

Restorer provide a way to restore objects from tf.train.Checkpoint.

Can be standalone.

Methods

__init__([logdir, ckpts_dir, expect_partial])	Initialize the Restorer.
get_global_step()	Return the restored global_step.
get_steps_per_epoch()	Return the restored global_step.
restore_callback(callback, callback_ckpt_id)	Return the restored callbacks.
restore_object(placeholder, object_ckpt_id)	Restore a placeholder from a checkpoint using the specified id.

Attributes

checkpoint_map	Get the map of the ids in the checkpoint.

__init__ (*logdir='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`

_restore_checkpoint (*checkpoint*, *partial=True*)
Restore or initialize the persistence layer (checkpoint).

checkpoint_map
Get the map of the ids in the checkpoint.

Return type `Dict[str, str]`

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]`

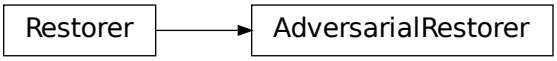
restore_object (*placeholder*, *object_ckpt_id*)
Restore a placeholder from a checkpoint using the specified id.

> **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='log'*, *ckpts_dir='ckpts'*, *expect_partial=True*)

Bases: *ashpy.restorers.restorer.Restorer*

Convenience *Restorer* for ease of use with the `AdversairalTrainer`.

Methods

restore_discriminator(model)	Restore the Discriminator model.
restore_discriminator_optimizer(optimizer)	Restore the Optimizer used to train the Discriminator model.
restore_generator(model)	Restore the Generator model.
restore_generator_optimizer(optimizer)	Restore the Optimizer used to train the Generator model.

Attributes

`checkpoint_map`	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.

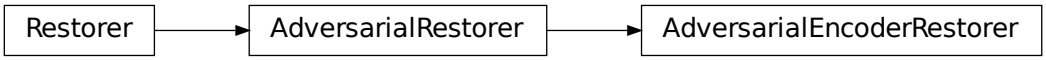
Parameters **model** (tf.keras.optimizers.Optimizer) – The placeholder Optimizer in which values from the checkpoint will be restored.

Return type OptimizerV2

Returns Restored optimizer.

4.10.3 AdversarialEncoderRestorer

Inheritance Diagram



---

class ashpy.restorers.**AdversarialEncoderRestorer**(*logdir='log'*, *ckpts_dir='ckpts'*, *expect_partial=True*)
Bases: *ashpy.restorers.gan.AdversarialRestorer*

Convenience *Restorer* for ease of use with the EncoderTrainer.

Methods

restore_encoder(model)	Restore the Encoder model.
restore_encoder_optimizer(optimizer)	Restore the Optimizer used to train the Encoder model.

Attributes

checkpoint_map	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.

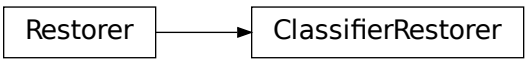
Parameters **model** (tf.keras.optimizers.Optimizer) – The placeholder Optimizer in which values from the checkpoint will be restored.

Return type OptimizerV2

Returns Restored optimizer.

4.10.4 ClassifierRestorer

Inheritance Diagram



class ashpy.restorers.**ClassifierRestorer**(*logdir='log'*, *ckpts_dir='ckpts'*, *expect_partial=True*)

Bases: *ashpy.restorers.restorer.Restorer*

Convenience *Restorer* for ease of use with the `ClassifierTrainer`.

Methods

restore_model(model)	Restore the Classifier model.
restore_optimizer(optimizer)	Restore the Optimizer used to train the Classifier model.

Attributes

`checkpoint_map`	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

restorer	Primitive Restorer, can be used standalone.
classifier	Convenience *Restorer* to be used with *ashpy.trainers.classifier*.
gan	Convenience *Restorer* to be used with *ashpy.trainers.gan*.

4.10.5 restorer

Primitive Restorer, can be used standalone.

Classes

Restorer	*Restorer* provide a way to restore objects from tf.train.Checkpoint.

Restorer

Inheritance Diagram

---

class ashpy.restorers.restorer.**Restorer**(*logdir='log'*, *ckpts_dir='ckpts'*, *expect_partial=True*)

Bases: object

Restorer provide a way to restore objects from tf.train.Checkpoint.

Can be standalone.

Methods

__init__([logdir, ckpts_dir, expect_partial])	Initialize the Restorer.
get_global_step()	Return the restored global_step.
get_steps_per_epoch()	Return the restored global_step.
restore_callback(callback, callback_ckpt_id)	Return the restored callbacks.
restore_object(placeholder, object_ckpt_id)	Restore a placeholder from a checkpoint using the specified id.

Attributes

checkpoint_map	Get the map of the ids in the checkpoint.

__init__(*logdir='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

_restore_checkpoint(*checkpoint*, *partial=True*)
Restore or initialize the persistence layer (checkpoint).

checkpoint_map
Get the map of the ids in the checkpoint.

Return type Dict[str, str]

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

class ashpy.restorers.restorer.**Restorer**(*logdir='log'*, *ckpts_dir='ckpts'*, *expect_partial=True*)

Bases: object

Restorer provide a way to restore objects from `tf.train.Checkpoint`.

Can be standalone.

__init__(*logdir='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

_restore_checkpoint(*checkpoint*, *partial=True*)
Restore or initialize the persistence layer (checkpoint).

checkpoint_map
Get the map of the ids in the checkpoint.

Return type Dict[str, str]

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

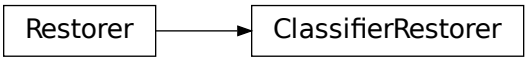
Convenience Restorer to be used with *ashpy.trainers.classifier*.

Classes

ClassifierRestorer	Convenience Restorer for ease of use with the ClassifierTrainer.

ClassifierRestorer

Inheritance Diagram



---

class ashpy.restorers.classifier.**ClassifierRestorer**(*logdir='log'*, *ckpts_dir='ckpts'*, *expect_partial=True*)

Bases: *ashpy.restorers.restorer.Restorer*

Convenience `Restorer` for ease of use with the `ClassifierTrainer`.

Methods

restore_model(model)	Restore the Classifier model.
restore_optimizer(optimizer)	Restore the Optimizer used to train the Classifier model.

Attributes

checkpoint_map	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='log'*, *ckpts_dir='ckpts'*, *expect_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

AdversarialEncoderRestorer	Convenience `Restorer` for ease of use with the `EncoderTrainer`.
AdversarialRestorer	Convenience `Restorer` for ease of use with the `AdversairalTrainer`.

AdversarialEncoderRestorer

Inheritance Diagram

---

class ashpy.restorers.gan.**AdversarialEncoderRestorer**(*logdir='log'*, *ckpts_dir='ckpts'*, *expect_partial=True*)

Bases: *ashpy.restorers.gan.AdversarialRestorer*

Convenience `Restorer` for ease of use with the `EncoderTrainer`.

Methods

restore_encoder(model)	Restore the Encoder model.
restore_encoder_optimizer(optimizer)	Restore the Optimizer used to train the Encoder model.

Attributes

checkpoint_map	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.

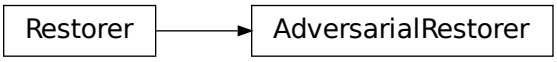
Parameters **model** (`tf.keras.optimizers.Optimizer`) – The placeholder Optimizer in which values from the checkpoint will be restored.

Return type `OptimizerV2`

Returns Restored optimizer.

AdversarialRestorer

Inheritance Diagram



---

class ashpy.restorers.gan.**AdversarialRestorer**(*logdir='log'*, *ckpts_dir='ckpts'*, *expect_partial=True*)

Bases: *ashpy.restorers.restorer.Restorer*

Convenience `Restorer` for ease of use with the `AdversairalTrainer`.

Methods

restore_discriminator(model)	Restore the Discriminator model.
restore_discriminator_optimizer(optimizer)	Restore the Optimizer used to train the Discriminator model.
restore_generator(model)	Restore the Generator model.
restore_generator_optimizer(optimizer)	Restore the Optimizer used to train the Generator model.

Attributes

checkpoint_map	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='log'*, *ckpts_dir='ckpts'*, *expect_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='log'*, *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.

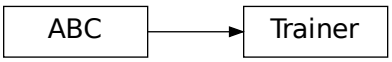
The module contains a primitive Interface and specific trainers that inherits from it.

Classes

Trainer	*Trainer* provide an interface for all trainers to inherit from.
AdversarialTrainer	Primitive Trainer for GANs subclassed from *ashpy.trainers.Trainer*.
EncoderTrainer	Primitive Trainer for GANs using an Encoder subnetwork.

4.11.1 Trainer

Inheritance Diagram



---

class ashpy.trainers.**Trainer**(*epochs*, *example_dim*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/che*
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

__init__(epochs, example_dim[, logdir, ...])	Primitive trainer interface.
call(*args, **kwargs)	Execute the training process.

Continued on next page

Table 335 – continued from previous page

local_example(example, dims)	Return a local example from a distributed example.
measure_metrics()	Measure the metrics.
model_selection()	Use the metrics to perform model selection.

Attributes

ckpt_id_callbacks
ckpt_id_global_step
ckpt_id_steps_per_epoch
context — Return the training context.

__init__(*epochs*, *example_dim*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/docs/sou*
log_eval_mode=<LogEvalMode.TEST: 1>, *global_step=None*, *metrics=None*, *callbacks=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[`ashpy.metrics.Metric`]]) – list of metrics.
- **callbacks** (Optional[List[`ashpy.callbacks.Callback`]]) – list of callbacks to handle events.

Return type `None`

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.Executo*
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.

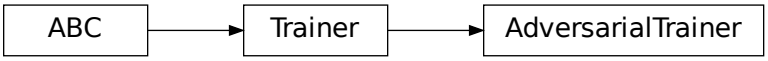
Return type None

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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkout*
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,
    epochs=epochs,
    metrics=metrics,
```

(continues on next page)

(continued from previous page)

```
    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

__init__(generator, discriminator, ... [, ...])	Instantiate a *AdversarialTrainer*.
call(dataset[, log_freq, ...])	Perform the adversarial training.
train_step(real_xy, g_inputs)	Train step for the AdversarialTrainer.

Attributes

ckpt_id_callbacks	
ckpt_id_discriminator	
ckpt_id_generator	
ckpt_id_global_step	
ckpt_id_optimizer_discriminator	
ckpt_id_optimizer_generator	
ckpt_id_steps_per_epoch	
context	Return the training context.

__init__(*generator*, *discriminator*, *generator_optimizer*, *discriminator_optimizer*, *generator_loss*, *discriminator_loss*, *epochs*, *metrics=None*, *callbacks=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/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** (Optional[List[*Metric*]]) – (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** (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

_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*, *generator_optimizer*, *discriminator_optimizer*, *encoder_optimizer*, *generator_loss*, *discriminator_loss*, *encoder_loss*, *epochs*, *metrics=None*, *callbacks=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3*
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

```
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)
```

(continues on next page)

(continued from previous page)

```
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 = "testlog/adversarial_encoder"

if os.path.exists(logdir):
    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.
```

Methods

__init__(generator, discriminator, encoder, ...)	Instantiate a *EncoderTrainer*.
call(dataset[, log_freq, ...])	Perform the adversarial training.
train_step(real_xy, g_inputs)	Adversarial training step.

Attributes

ckpt_id_callbacks	
ckpt_id_discriminator	
ckpt_id_encoder	
ckpt_id_generator	
ckpt_id_global_step	
ckpt_id_optimizer_discriminator	
ckpt_id_optimizer_encoder	
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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/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** (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.

_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

trainer	Primitive Trainer Interface.
classifier	Primitive Trainer Interface.
gan	Collection of GANs trainers.

4.11.4 trainer

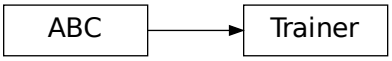
Primitive Trainer Interface.

Classes

Trainer	*Trainer* provide an interface for all trainers to inherit from.

Trainer

Inheritance Diagram



---

class ashpy.trainers.trainer.**Trainer**(*epochs*, *example_dim*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.*, *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

__init__(epochs, example_dim[, logdir, ...])	Primitive trainer interface.
call(*args, **kwargs)	Execute the training process.
local_example(example, dims)	Return a local example from a distributed example.
measure_metrics()	Measure the metrics.
model_selection()	Use the metrics to perform model selection.

Attributes

ckpt_id_callbacks	
ckpt_id_global_step	
ckpt_id_steps_per_epoch	
context	Return the training context.

__init__(*epochs*, *example_dim*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/docs/sou*
log_eval_mode=<LogEvalMode.TEST: 1>, *global_step=None*, *metrics=None*, *callbacks=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[`ashpy.metrics.Metric`]]) – list of metrics.
- **callbacks** (Optional[List[`ashpy.callbacks.Callback`]]) – list of callbacks to handle events.

Return type `None`

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.Executo*
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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.*, *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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/docs/sou*, *log_eval_mode=<LogEvalMode.TEST: 1>*, *global_step=None*, *metrics=None*, *callbacks=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[ashpy.metrics.Metric]]) – list of metrics.
- **callbacks** (Optional[List[ashpy.callbacks.Callback]]) – list of callbacks to handle events.

Return type None

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.Executo*
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

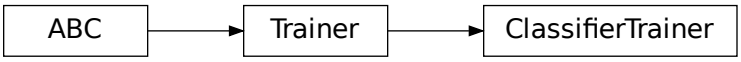
Primitive Trainer Interface.

Classes

ClassifierTrainer	*ClassifierTrainer* 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='/home/docs/checkouts/readthedocs.org/user_builds/*
global_step=None)

Bases: *ashpy.trainers.trainer.Trainer*

ClassifierTrainer provide the standard training loop for a classifier.

Methods

__init__(model, optimizer, loss, epochs[, ...])	Instantiate the *ClassifierTrainer* trainer.
call(training_set, validation_set[, ...])	Start the training.
train_step(features, labels)	Train step.

Attributes

ckpt_id_callbacks	
ckpt_id_global_step	
ckpt_id_model	
ckpt_id_optimizer	
ckpt_id_steps_per_epoch	
context	Return the training context.

__init__(*model*, *optimizer*, *loss*, *epochs*, *metrics=None*, *callbacks=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/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** (Optional[List[*Metric*]]) – (List): List of *ashpy.metrics.metric.Metric* to measure on training and validation data.

- **callbacks** (*List*) – List of *ashpy.callbacks.callback.Callback* to 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
    )
    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
```

(continues on next page)

(continued from previous page)

```
Epoch 2 completed.
Training finished after 2 epochs.
```

_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='/home/docs/checkouts/readthedocs.org/user_builds/*
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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/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** (Optional[List[*Metric*]]) – (List): List of *ashpy.metrics.metric.Metric* to measure on training and validation data.

- **callbacks** (*List*) – List of *ashpy.callbacks.callback.Callback* to 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
    )
    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
```

(continues on next page)

(continued from previous page)

```
Epoch 2 completed.
Training finished after 2 epochs.
```

_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

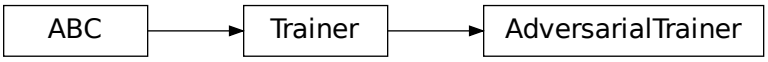
Collection of GANs trainers.

Classes

AdversarialTrainer	Primitive Trainer for GANs subclassed from *ashpy.trainers.Trainer*.
EncoderTrainer	Primitive Trainer for GANs using an Encoder subnetwork.

AdversarialTrainer

Inheritance Diagram



---

class ashpy.trainers.gan.**AdversarialTrainer**(*generator*, *discriminator*, *generator_optimizer*, *discriminator_optimizer*, *generator_loss*, *discriminator_loss*, *epochs*, *metrics=None*, *callbacks=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/ch*
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
```

(continues on next page)

(continued from previous page)

```
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,
    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

__init__(generator, discriminator, . . . [, . . .])	Instantiate a *AdversarialTrainer*.
call(dataset[, log_freq, . . .])	Perform the adversarial training.
train_step(real_xy, g_inputs)	Train step for the AdversarialTrainer.

Attributes

ckpt_id_callbacks	
ckpt_id_discriminator	
ckpt_id_generator	
ckpt_id_global_step	
ckpt_id_optimizer_discriminator	
ckpt_id_optimizer_generator	
ckpt_id_steps_per_epoch	
context	Return the training context.

__init__(*generator*, *discriminator*, *generator_optimizer*, *discriminator_optimizer*, *generator_loss*, *discriminator_loss*, *epochs*, *metrics=None*, *callbacks=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/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** (Optional[List[*Metric*]]) – (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** (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

_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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkout*
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

```
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 = "testlog/adversarial_encoder"

if os.path.exists(logdir):
    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),
```

(continues on next page)

(continued from previous page)

```
    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

__init__(generator, discriminator, encoder, ...)	Instantiate a *EncoderTrainer*.
call(dataset[, log_freq, ...])	Perform the adversarial training.
train_step(real_xy, g_inputs)	Adversarial training step.

Attributes

ckpt_id_callbacks	
ckpt_id_discriminator	
ckpt_id_encoder	
ckpt_id_generator	
ckpt_id_global_step	
ckpt_id_optimizer_discriminator	
ckpt_id_optimizer_encoder	
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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/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** (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.

_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*, *generator_optimizer*, *discriminator_optimizer*, *generator_loss*, *discriminator_loss*, *epochs*, *metrics=None*, *callbacks=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/ch*
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,))
```

(continues on next page)

(continued from previous page)

```
# 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,
    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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/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** (Optional[List[*Metric*]]) – (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** (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

_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*, *generator_optimizer*, *discriminator_optimizer*, *encoder_optimizer*, *generator_loss*, *discriminator_loss*, *encoder_loss*, *epochs*, *metrics=None*, *callbacks=None*, *logdir='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkout*
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

```
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()
```

(continues on next page)

(continued from previous page)

```
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 = "testlog/adversarial_encoder"

if os.path.exists(logdir):
    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='/home/docs/checkouts/readthedocs.org/user_builds/ashpy/checkouts/v0.3.0/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** (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.

_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.

CHAPTER 5

AshPy Internals

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 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:

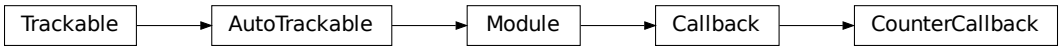
- `Executor.reduce_loss` decoration: uses the Executor decorator in order to correctly reduce the loss
- `tf.reduce_mean(value, axis=1)` (last line), we perform only the mean over the axis 1. The output of the *call* function

should be a `tf.Tensor` with shape (N, 1) or (N,). This is because the decorator performs the mean over the axis 0.

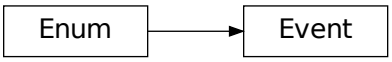
CHAPTER 6

Dependencies Graph

6.1 ashpy.callbacks

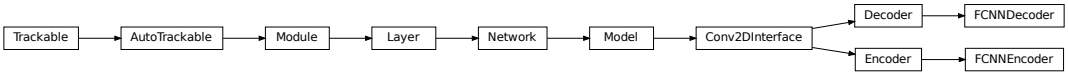


6.1.1 Events



6.2 ashpy.models

6.2.1 Convolutional



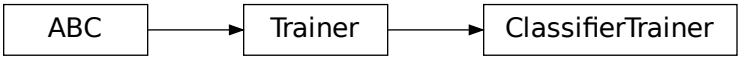
6.2.2 GANs

GANs models are just aliases.

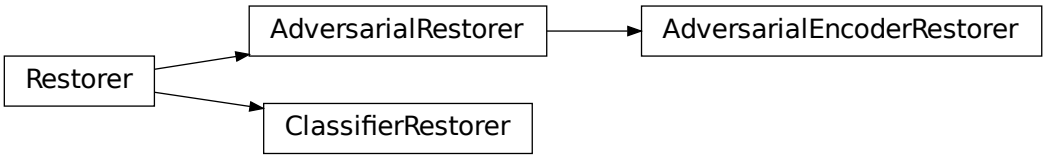
6.3 ashpy.trainers

6.3.1 Adversarial

6.3.2 Classifier

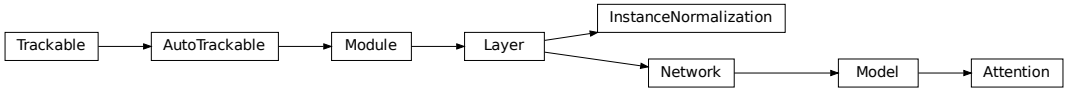


6.4 ashpy.restorers



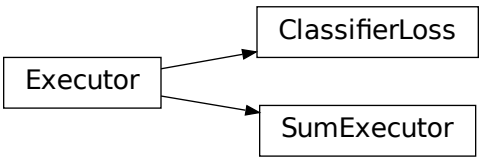
6.5 ashpy.layers

6.5.1 Layers

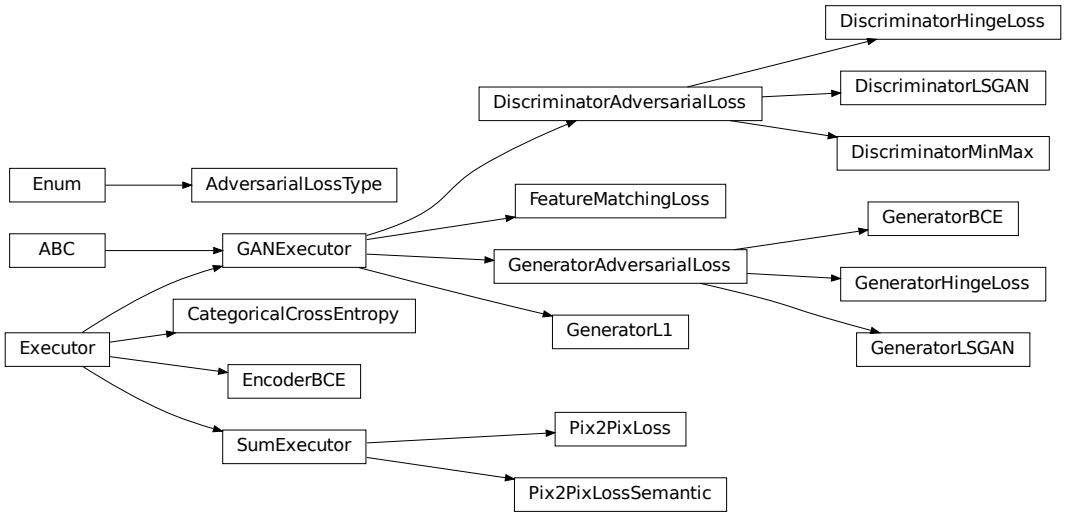


6.6 ashpy.losses

6.6.1 Classifier Losses

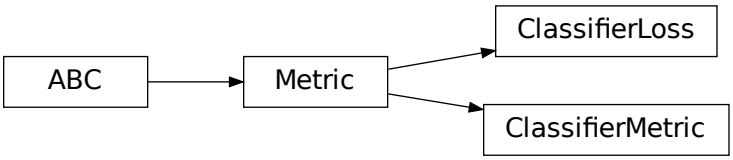


6.6.2 GAN Losses

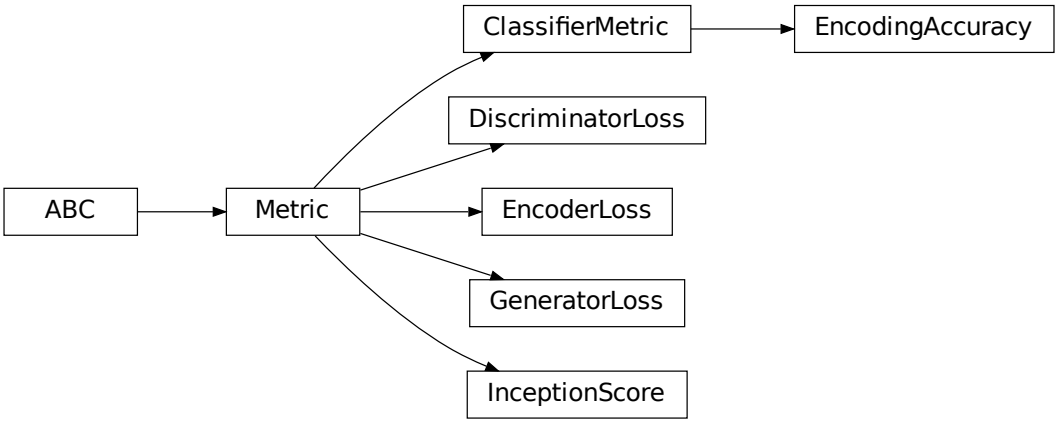


6.7 ashpy.metrics

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

Python Module Index

a

Index

Symbols

A

B

C

D

E

F

G

P

R

S

T

U

V

W