Anomalib

Release 2022

Intel OpenVINO

Mar 28, 2024

GET STARTED

1 Installation 3

2 Get Started 5

3 Guides 7
- 3.1 Anomalib in 15 Minutes 7
- 3.2 Migrating from 0.* to 1.0 14
- 3.3 Reference Guide 17
- 3.4 How to Guide 209
- 3.5 Topic Guide 220
- 3.6 Developer Guide 220
- 3.7 Awards and Recognition 232
- 3.8 Community Engagement 233

Python Module Index 235

Index 237

Anomalib is a deep learning library that aims to collect state-of-the-art anomaly detection algorithms for benchmarking on both public and private datasets. Anomalib provides several ready-to-use implementations of anomaly detection algorithms described in the recent literature, as well as a set of tools that facilitate the development and implementation of custom models. The library has a strong focus on visual anomaly detection, where the goal of the algorithm is to detect and/or localize anomalies within images or videos in a dataset. Anomalib is constantly updated with new algorithms and training/inference extensions, so keep checking!

Key Features

- Simple and modular API and CLI for training, inference, benchmarking, and hyperparameter optimization.
- The largest public collection of ready-to-use deep learning anomaly detection algorithms and benchmark datasets.
- Lightning based model implementations to reduce boilerplate code and limit the implementation efforts to the bare essentials.
- All models can be exported to OpenVINO Intermediate Representation (IR) for accelerated inference on intel hardware.
- A set of inference tools for quick and easy deployment of the standard or custom anomaly detection models.

CHAPTER ONE

INSTALLATION

Anomalib provides two ways to install the library. The first is through PyPI, and the second is through a local installation. PyPI installation is recommended if you want to use the library without making any changes to the source code. If you want to make changes to the library, then a local installation is recommended.

PyPI

```
pip install anomalib
```

Source

```
# Use of virtual environment is highy recommended
# Using conda
yes | conda create -n anomalib_env python=3.10
conda activate anomalib_env

# Or using your favorite virtual environment
# ...

# Clone the repository and install in editable mode
git clone https://github.com/openvinotoolkit/anomalib.git
cd anomalib
pip install -e .
```

CHAPTER

TWO

GET STARTED

Anomalib in 15 minutes   Get started with anomalib in 15 minutes.

CHAPTER

THREE

GUIDES

Reference Guide   Learn more about anomalib API and CLI.

How-To Guide   Learn how to use anomalib for your anomaly detection tasks.

Topic Guide   Learn more about the internals of anomalib.

Developer Guide   Learn how to develop and contribute to anomalib.

3.1 Anomalib in 15 Minutes

This section will walk you through the steps to train a model and use it to detect anomalies in a dataset.

3.1.1 Installation

Installation is simple and can be done in two ways. The first is through PyPI, and the second is through a local installation. PyPI installation is recommended if you want to use the library without making any changes to the source code. If you want to make changes to the library, then a local installation is recommended.

Installing the Installer

Anomalib comes with a CLI installer that can be used to install the full package. The installer can be installed using the following commands:

API

```
pip install anomalib
```

Source

```
# Use of virtual environment is highy recommended
# Using conda
yes | conda create -n anomalib_env python=3.10
conda activate anomalib_env

# Or using your favorite virtual environment
# ...

# Clone the repository and install in editable mode
git clone https://github.com/openvinotoolkit/anomalib.git
cd anomalib
pip install -e .
```

The main reason why PyPI and source installer does not install the full package is to keep the installation wheel small. The CLI installer also automates the installation such as finding the torch version with the right CUDA/CUDNN version.

The next section demonstrates how to install the full package using the CLI installer.

Installing the Full Package

After installing anomalib, you can install the full package using the following commands:

```
 anomalib -h
To use other subcommand using `anomalib install`
To use any logger install it using `anomalib install -v`
─ Arguments ───────────────────────────────────────────────────────────────────
│ Usage: anomalib [-h] [-c CONFIG] [--print_config [=flags]] {install} ...     │
│                                                                              │
│                                                                              │
│ Options:                                                                     │
│   -h, --help            Show this help message and exit.                     │
│   -c, --config CONFIG   Path to a configuration file in json or yaml format. │
│   --print_config [=flags]                                                    │
│                         Print the configuration after applying all other     │
│                         arguments and exit. The optional flags customizes the│
│                         output and are one or more keywords separated b comma.
│                         The supported flags are: comments, skip_default,     │
│                         skip_null.                                           │
│                                                                              │
│ Subcommands:                                                                 │
│   For more details of each subcommand, add it as an argument followed by     │
│                         --help.                                              │
│                                                                              │
│                                                                              │
│   Available subcommands:                                                     │
│     install             Install the full-package for anomalib.               │
│                                                                              │
───────────────────────────────────────────────────────────────────────────────
```

As can be seen above, the only available sub-command is `install` at the moment. The `install` sub-command has options to install either the full package or the specific components of the package.

```
 anomalib install -h
To use other subcommand using `anomalib install`
To use any logger install it using `anomalib install -v`
─ Arguments ──────────────────────────────────────────────────────────────────
│ Usage: anomalib [options] install [-h]                                       │
│                        [--option {full,core,dev,loggers,notebooks,openvino}] │
│                        [-v]                                                  │
│                                                                              │
│                                                                              │
│ Options:                                                                     │
│   -h, --help            Show this help message and exit.                     │
│   --option {full,core,dev,loggers,notebooks,openvino}                        │
│                         Install the full or optional-dependencies.           │
│                         (type: None, default: full)                          │
│   -v, --verbose         Set Logger level to INFO (default: False)            │
│                                                                              │
──────────────────────────────────────────────────────────────────────────────
```

By default the `install` sub-command installs the full package. If you want to install only the specific components of the package, you can use the `--option` flag.

```
# Get help for the installation arguments
anomalib install -h

# Install the full package
anomalib install

# Install with verbose output
anomalib install -v

# Install the core package option only to train and evaluate models via Torch and␣
↪Lightning
anomalib install --option core

# Install with OpenVINO option only. This is useful for edge deployment as the wheel␣
↪size is smaller.
anomalib install --option openvino
```

After following these steps, your environment will be ready to use anomalib!

3.1.2 Training

Anomalib supports both API and CLI-based training. The API is more flexible and allows for more customization, while the CLI training utilizes command line interfaces, and might be easier for those who would like to use anomalib off-the-shelf.

API

```python
# Import the required modules
from anomalib.data import MVTec
from anomalib.models import Patchcore
from anomalib.engine import Engine

# Initialize the datamodule, model and engine
datamodule = MVTec()
model = Patchcore()
engine = Engine()

# Train the model
engine.fit(datamodule=datamodule, model=model)
```

CLI

```bash
# Get help about the training arguments, run:
anomalib train -h

# Train by using the default values.
anomalib train --model Patchcore --data anomalib.data.MVTec

# Train by overriding arguments.
anomalib train --model Patchcore --data anomalib.data.MVTec --data.category transistor

# Train by using a config file.
anomalib train --config <path/to/config>
```

3.1.3 Inference

Anomalib includes multiple inferencing scripts, including Torch, Lightning, Gradio, and OpenVINO inferencers to perform inference using the trained/exported model. Here we show an inference example using the Lightning inferencer.

Lightning Inference

API

```
# Assuming the datamodule, model and engine is initialized from the previous step,
# a prediction via a checkpoint file can be performed as follows:
predictions = engine.predict(
    datamodule=datamodule,
    model=model,
    ckpt_path="path/to/checkpoint.ckpt",
)
```

CLI

```
# To get help about the arguments, run:
anomalib predict -h

# Predict by using the default values.
anomalib predict --model anomalib.models.Patchcore \
                 --data anomalib.data.MVTec \
                 --ckpt_path <path/to/model.ckpt>

# Predict by overriding arguments.
anomalib predict --model anomalib.models.Patchcore \
                 --data anomalib.data.MVTec \
                 --ckpt_path <path/to/model.ckpt>
                 --return_predictions

# Predict by using a config file.
anomalib predict --config <path/to/config> --return_predictions
```

Torch Inference

API

```
Python code here.
```

CLI

```
CLI command here.
```

OpenVINO Inference

API

```
Python code here.
```

CLI

```
CLI command here.
```

Gradio Inference

API

```
Python code here.
```

CLI

```
CLI command here.
```

3.1.4 Hyper-Parameter Optimization

Anomalib supports hyper-parameter optimization using wandb and comet.ml. Here we show an example of hyper-parameter optimization using both comet and wandb.

CLI

```
# To perform hpo using wandb sweep
anomalib hpo --backend WANDB  --sweep_config tools/hpo/configs/wandb.yaml

# To perform hpo using comet.ml sweep
anomalib hpo --backend COMET  --sweep_config tools/hpo/configs/comet.yaml
```

API

```
# To be enabled in v1.1
```

3.1.5 Experiment Management

Anomalib is integrated with various libraries for experiment tracking such as comet, tensorboard, and wandb through lighting loggers.

CLI

To run a training experiment with experiment tracking, you will need the following configuration file:

```
# Place the experiment management config here.
```

By using the configuration file above, you can run the experiment with the following command:

```
# Place the Experiment Management CLI command here.
```

API

```
# To be enabled in v1.1
```

3.1.6 Benchmarking

Anomalib provides a benchmarking tool to evaluate the performance of the anomaly detection models on a given dataset. The benchmarking tool can be used to evaluate the performance of the models on a given dataset, or to compare the performance of multiple models on a given dataset.

Each model in anomalib is benchmarked on a set of datasets, and the results are available in `src/anomalib/models/<model_name>README.md`. For example, the MVTec AD results for the Patchcore model are available in the corresponding README.md file.

CLI

To run the benchmarking tool, run the following command:

```
anomalib benchmark --config tools/benchmarking/benchmark_params.yaml
```

API

```
# To be enabled in v1.1
```

3.1.7 Reference

If you use this library and love it, use this to cite it:

```
@inproceedings{akcay2022anomalib,
  title={Anomalib: A deep learning library for anomaly detection},
  author={
    Akcay, Samet and
    Ameln, Dick and
    Vaidya, Ashwin and
    Lakshmanan, Barath
    and Ahuja, Nilesh
    and Genc, Utku
  },
  booktitle={2022 IEEE International Conference on Image Processing (ICIP)},
  pages={1706--1710},
  year={2022},
  organization={IEEE}
}
```

3.2 Migrating from 0.* to 1.0

3.2.1 Overview

The 1.0 release of the Anomaly Detection Library (AnomalyLib) introduces several changes to the library. This guide provides an overview of the changes and how to migrate from 0.* to 1.0.

3.2.2 Installation

For installation instructions, refer to the *installation guide*.

3.2.3 Changes to the CLI

Upgrading the Configuration

There are several changes to the configuration of Anomalib. The configuration file has been updated to include new parameters and remove deprecated parameters. In addition, some parameters have been moved to different sections of the configuration.

Anomalib provides a python script to update the configuration file from 0.* to 1.0. To update the configuration file, run the following command:

```
python tools/upgrade/config.py \
    --input_config <path_to_0.*_config> \
    --output_config <path_to_1.0_config>
```

This script will ensure that the configuration file is updated to the 1.0 format.

In the following sections, we will discuss the changes to the configuration file in more detail.

Changes to the Configuration File

Data

The `data` section of the configuration file has been updated such that the args can be directly used to instantiate the data object. Below are the differences between the old and new configuration files highlighted in a markdown diff format.

```
-dataset:
+data:
-  name: mvtec
-  format: mvtec
+  class_path: anomalib.data.MVTec
+  init_args:
-  path: ./datasets/MVTec
+    root: ./datasets/MVTec
     category: bottle
     image_size: 256
     center_crop: null
     normalization: imagenet
     train_batch_size: 72
     eval_batch_size: 32
     num_workers: 8
     task: segmentation
     test_split_mode: from_dir # options: [from_dir, synthetic]
     test_split_ratio: 0.2 # fraction of train images held out testing (usage depends on␣
↪test_split_mode)
     val_split_mode: same_as_test # options: [same_as_test, from_test, synthetic]
     val_split_ratio: 0.5 # fraction of train/test images held out for validation (usage␣
↪depends on val_split_mode)
     seed: null
-  transform_config:
-    train: null
-    eval: null
+    transform_config_train: null
+    transform_config_eval: null
-  tiling:
-    apply: false
-    tile_size: null
-    stride: null
-    remove_border_count: 0
-    use_random_tiling: False
-    random_tile_count: 16+data:
```

Here is the summary of the changes to the configuration file:

- The `name` and `format keys` from the old configuration are absent in the new configuration, possibly integrated into the design of the class at `class_path`.
- Introduction of a `class_path` key in the new configuration specifies the Python class path for data handling.

- The structure has been streamlined in the new configuration, moving everything under `data` and `init_args` keys, simplifying the hierarchy.
- `transform_config` keys were split into `transform_config_train` and `transform_config_eval` to clearly separate training and evaluation configurations.
- The `tiling` section present in the old configuration has been completely removed in the new configuration. v1.0.0 does not support tiling. This feature will be added back in a future release.

Model

Similar to data configuration, the `model` section of the configuration file has been updated such that the args can be directly used to instantiate the model object. Below are the differences between the old and new configuration files highlighted in a markdown diff format.

```diff
 model:
-  name: patchcore
-  backbone: wide_resnet50_2
-  pre_trained: true
-  layers:
+  class_path: anomalib.models.Patchcore
+  init_args:
+    backbone: wide_resnet50_2
+    pre_trained: true
+    layers:
     - layer2
     - layer3
-  coreset_sampling_ratio: 0.1
-  num_neighbors: 9
+    coreset_sampling_ratio: 0.1
+    num_neighbors: 9
-  normalization_method: min_max # options: [null, min_max, cdf]
+normalization:
+  normalization_method: min_max
```

Here is the summary of the changes to the configuration file:

- Model Identification: Transition from `name` to `class_path` for specifying the model, indicating a more explicit reference to the model's implementation.
- Initialization Structure: Introduction of `init_args` to encapsulate model initialization parameters, suggesting a move towards a more structured and possibly dynamically loaded configuration system.
- Normalization Method: The `normalization_method` key is removed from the `model` section and moved to a separate `normalization` section in the new configuration.

Metrics

The `metrics` section of the configuration file has been updated such that the args can be directly used to instantiate the metrics object. Below are the differences between the old and new configuration files highlighted in a markdown diff format.

```
metrics:
  image:
    - F1Score
    - AUROC
  pixel:
     - F1Score
     - AUROC
   threshold:
-    method: adaptive #options: [adaptive, manual]
-    manual_image: null
-    manual_pixel: null
+    class_path: anomalib.metrics.F1AdaptiveThreshold
+    init_args:
+      default_value: 0.5
```

Here is the summary of the changes to the configuration file:

- Metric Identification: Transition from `method` to `class_path` for specifying the metric, indicating a more explicit reference to the metric's implementation.
- Initialization Structure: Introduction of `init_args` to encapsulate metric initialization parameters, suggesting a move towards a more structured and possibly dynamically loaded configuration system.
- Threshold Method: The `method` key is removed from the `threshold` section and moved to a separate `class_path` section in the new configuration.

3.3 Reference Guide

This section contains the API and CLI reference for anomalib.

Data   Learn more about anomalib datamodules.

Models   Learn more about image and video models.

Engine   Learn more about anomalib Engine.

Metrics   Learn more about anomalib metrics

Loggers   Learn more about anomalib loggers

Callbacks   Learn more about anomalib callbacks

CLI   Learn more about anomalib CLI

Deployment   Learn more about anomalib CLI

Pipelines   Learn more about anomalib hpo, sweep and benchmarking pipelines

3.3.1 Data

Anomalib data can be categorized into four main types: base, image, video, and depth. Image, video and depth datasets are based on the base dataset and datamodule implementations.

Base Classes	Learn more about base anomalib data interfaces.
Image	Learn more about anomalib image datasets.
Video	Learn more about anomalib video datasets.
Depth	Learn more about anomalib depth datasets.

Base Data

Base Dataset	Learn more about base anomalib dataset
Base Datamodule	Learn more about base anomalib datamodule
Video	Learn more about base anomalib video data
Depth	Learn more about base anomalib depth data

Base Dataset

Anomalib dataset base class.

`class anomalib.data.base.dataset.`**`AnomalibDataset`**(*task*, *transform=None*)

Bases: `Dataset`, `ABC`

Anomalib dataset.

The dataset is based on a dataframe that contains the information needed by the dataloader to load each of the dataset items into memory.

The samples dataframe must be set from the subclass using the setter of the *samples* property.

The DataFrame must, at least, include the following columns:

- *split* (str): The subset to which the dataset item is assigned (e.g., 'train', 'test').
- *image_path* (str): Path to the file system location where the image is stored.
- *label_index* (int): Index of the anomaly label, typically 0 for 'normal' and 1 for 'anomalous'.
- *mask_path* (str, optional): Path to the ground truth masks (for the anomalous images only).

Required if task is 'segmentation'.

Example DataFrame:

	image_path	label	label_index	mask_path	split
0	path/to/image.png	anomalous	1	path/to/mask.png	train

Note: The example above is illustrative and may need to be adjusted based on the specific dataset structure.

Parameters

- **task** (*str*) – Task type, either 'classification' or 'segmentation'
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to `None`.

property category: str | None

Get the category of the dataset.

property has_anomalous: bool

Check if the dataset contains any anomalous samples.

property has_normal: bool

Check if the dataset contains any normal samples.

property name: str

Name of the dataset.

property samples: DataFrame

Get the samples dataframe.

subsample(*indices*, *inplace=False*)

Subsamples the dataset at the provided indices.

Parameters

- **indices** (*Sequence[int]*) – Indices at which the dataset is to be subsampled.
- **inplace** (*bool*) – When true, the subsampling will be performed on the instance itself. Defaults to `False`.

Return type

AnomalibDataset

Base Datamodules

Anomalib datamodule base class.

class anomalib.data.base.datamodule.**AnomalibDataModule**(*train_batch_size*, *eval_batch_size*, *num_workers*, *val_split_mode*, *val_split_ratio*, *test_split_mode=None*, *test_split_ratio=None*, *image_size=None*, *transform=None*, *train_transform=None*, *eval_transform=None*, *seed=None*)

Bases: `LightningDataModule`, `ABC`

Base Anomalib data module.

Parameters

- **train_batch_size** (*int*) – Batch size used by the train dataloader.
- **eval_batch_size** (*int*) – Batch size used by the val and test dataloaders.
- **num_workers** (*int*) – Number of workers used by the train, val and test dataloaders.
- **val_split_mode** (`ValSplitMode`) – Determines how the validation split is obtained. Options: [none, same_as_test, from_test, synthetic]
- **val_split_ratio** (*float*) – Fraction of the train or test images held our for validation.

- **test_split_mode** (*Optional[TestSplitMode], optional*) – Determines how the test split is obtained. Options: [none, from_dir, synthetic]. Defaults to `None`.
- **test_split_ratio** (*float*) – Fraction of the train images held out for testing. Defaults to `None`.
- **image_size** (*tuple[int, int], optional*) – Size to which input images should be resized. Defaults to `None`.
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to `None`.
- **train_transform** (*Transform, optional*) – Transforms that should be applied to the input images during training. Defaults to `None`.
- **eval_transform** (*Transform, optional*) – Transforms that should be applied to the input images during evaluation. Defaults to `None`.
- **seed** (*int | None, optional*) – Seed used during random subset splitting. Defaults to `None`.

property category: str

Get the category of the datamodule.

property eval_transform: Transform

Get the transform that will be passed to the val/test/predict datasets.

If the eval_transform is not set, the engine will request the transform from the model.

property name: str

Name of the datamodule.

predict_dataloader()

Use the test dataloader for inference unless overridden.

Return type
`Any`

setup(*stage=None*)

Set up train, validation and test data.

Parameters
stage (`str` | `None`) – str | None: Train/Val/Test stages. Defaults to `None`.

Return type
`None`

test_dataloader()

Get test dataloader.

Return type
`Any`

train_dataloader()

Get train dataloader.

Return type
`Any`

property train_transform: Transform

Get the transforms that will be passed to the train dataset.

If the train_transform is not set, the engine will request the transform from the model.

property transform: Transform

Property that returns the user-specified transform for the datamodule, if any.

This property is accessed by the engine to set the transform for the model. The eval_transform takes precedence over the train_transform, because the transform that we store in the model is the one that should be used during inference.

val_dataloader()

Get validation dataloader.

Return type
`Any`

anomalib.data.base.datamodule.**collate_fn**(*batch*)

Collate bounding boxes as lists.

Bounding boxes are collated as a list of tensors, while the default collate function is used for all other entries.

Parameters
batch (*List*) – list of items in the batch where len(batch) is equal to the batch size.

Returns
Dictionary containing the collated batch information.

Return type
dict[str, Any]

Base Depth Data

Base Depth Dataset.

class anomalib.data.base.depth.**AnomalibDepthDataset**(*task*, *transform=None*)

Bases: *AnomalibDataset*, `ABC`

Base depth anomalib dataset class.

Parameters

- **task** (*str*) – Task type, either 'classification' or 'segmentation'
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to `None`.

Base Video Data

Base Video Dataset.

class anomalib.data.base.video.**AnomalibVideoDataModule**(*train_batch_size*, *eval_batch_size*, *num_workers*, *val_split_mode*, *val_split_ratio*, *test_split_mode=None*, *test_split_ratio=None*, *image_size=None*, *transform=None*, *train_transform=None*, *eval_transform=None*, *seed=None*)

Bases: *AnomalibDataModule*

Base class for video data modules.

`class anomalib.data.base.video.`**`AnomalibVideoDataset`**`(`*task*, *clip_length_in_frames*, *frames_between_clips*, *transform=None*, *target_frame=VideoTargetFrame.LAST*`)`

Bases: *`AnomalibDataset`*, `ABC`

Base video anomalib dataset class.

Parameters

- **`task`** (*`str`*) – Task type, either 'classification' or 'segmentation'
- **`clip_length_in_frames`** (*`int`*) – Number of video frames in each clip.
- **`frames_between_clips`** (*`int`*) – Number of frames between each consecutive video clip.
- **`transform`** (*`Transform, optional`*) – Transforms that should be applied to the input clips. Defaults to `None`.
- **`target_frame`** (`VideoTargetFrame`) – Specifies the target frame in the video clip, used for ground truth retrieval. Defaults to `VideoTargetFrame.LAST`.

`property samples: DataFrame`

Get the samples dataframe.

`class anomalib.data.base.video.`**`VideoTargetFrame`**`(`*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*`)`

Bases: `str`, `Enum`

Target frame for a video-clip.

Used in multi-frame models to determine which frame's ground truth information will be used.

Image Data

BTech Learn more about BTech dataset.

Folder Learn more about custom folder dataset.

Kolektor Learn more about Kolektor dataset.

MVTec 2D Learn more about MVTec 2D dataset

Visa Learn more about Visa dataset.

BTech Data

BTech Dataset.

This script contains PyTorch Lightning DataModule for the BTech dataset.

If the dataset is not on the file system, the script downloads and extracts the dataset and create PyTorch data objects.

`class anomalib.data.image.btech.`**`BTech`**`(`*root='./datasets/BTech'*, *category='01'*, *train_batch_size=32*, *eval_batch_size=32*, *num_workers=8*, *task=TaskType.SEGMENTATION*, *image_size=None*, *transform=None*, *train_transform=None*, *eval_transform=None*, *test_split_mode=TestSplitMode.FROM_DIR*, *test_split_ratio=0.2*, *val_split_mode=ValSplitMode.SAME_AS_TEST*, *val_split_ratio=0.5*, *seed=None*`)`

Bases: *AnomalibDataModule*

BTech Lightning Data Module.

Parameters

- **root** (*Path | str*) – Path to the BTech dataset. Defaults to `"./datasets/BTech"`.
- **category** (*str*) – Name of the BTech category. Defaults to `"01"`.
- **train_batch_size** (*int, optional*) – Training batch size. Defaults to `32`.
- **eval_batch_size** (*int, optional*) – Eval batch size. Defaults to `32`.
- **num_workers** (*int, optional*) – Number of workers. Defaults to `8`.
- **task** (*TaskType, optional*) – Task type. Defaults to `TaskType.SEGMENTATION`.
- **image_size** (*tuple[int, int], optional*) – Size to which input images should be resized. Defaults to `None`.
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to `None`.
- **train_transform** (*Transform, optional*) – Transforms that should be applied to the input images during training. Defaults to `None`.
- **eval_transform** (*Transform, optional*) – Transforms that should be applied to the input images during evaluation. Defaults to `None`.
- **test_split_mode** (*TestSplitMode, optional*) – Setting that determines how the testing subset is obtained. Defaults to `TestSplitMode.FROM_DIR`.
- **test_split_ratio** (*float, optional*) – Fraction of images from the train set that will be reserved for testing. Defaults to `0.2`.
- **val_split_mode** (*ValSplitMode, optional*) – Setting that determines how the validation subset is obtained. Defaults to `ValSplitMode.SAME_AS_TEST`.
- **val_split_ratio** (*float, optional*) – Fraction of train or test images that will be reserved for validation. Defaults to `0.5`.
- **seed** (*int | None, optional*) – Seed which may be set to a fixed value for reproducibility. Defaults to `None`.

Examples

To create the BTech datamodule, we need to instantiate the class, and call the `setup` method.

```
>>> from anomalib.data import BTech
>>> datamodule = BTech(
...     root="./datasets/BTech",
...     category="01",
...     image_size=256,
...     train_batch_size=32,
...     eval_batch_size=32,
...     num_workers=8,
...     transform_config_train=None,
...     transform_config_eval=None,
```

(continues on next page)

(continued from previous page)

```
... )
>>> datamodule.setup()
```

To get the train dataloader and the first batch of data:

```
>>> i, data = next(enumerate(datamodule.train_dataloader()))
>>> data.keys()
dict_keys(['image'])
>>> data["image"].shape
torch.Size([32, 3, 256, 256])
```

To access the validation dataloader and the first batch of data:

```
>>> i, data = next(enumerate(datamodule.val_dataloader()))
>>> data.keys()
dict_keys(['image_path', 'label', 'mask_path', 'image', 'mask'])
>>> data["image"].shape, data["mask"].shape
(torch.Size([32, 3, 256, 256]), torch.Size([32, 256, 256]))
```

Similarly, to access the test dataloader and the first batch of data:

```
>>> i, data = next(enumerate(datamodule.test_dataloader()))
>>> data.keys()
dict_keys(['image_path', 'label', 'mask_path', 'image', 'mask'])
>>> data["image"].shape, data["mask"].shape
(torch.Size([32, 3, 256, 256]), torch.Size([32, 256, 256]))
```

prepare_data()

Download the dataset if not available.

This method checks if the specified dataset is available in the file system. If not, it downloads and extracts the dataset into the appropriate directory.

Return type
`None`

Example

Assume the dataset is not available on the file system. Here's how the directory structure looks before and after calling the *prepare_data* method:

Before:

```
$ tree datasets
datasets
├── dataset1
└── dataset2
```

Calling the method:

```
>> datamodule = BTech(root="./datasets/BTech", category="01")
>> datamodule.prepare_data()
```

After:

```
$ tree datasets
datasets
├── dataset1
├── dataset2
└── BTech
    ├── 01
    ├── 02
    └── 03
```

class `anomalib.data.image.btech.`**`BTechDataset`**(*root*, *category*, *transform=None*, *split=None*, *task=TaskType.SEGMENTATION*)

Bases: *`AnomalibDataset`*

Btech Dataset class.

Parameters

- **`root`** (`str` | `Path`) – Path to the BTech dataset
- **`category`** (`str`) – Name of the BTech category.
- **`transform`** (*`Transform, optional`*) – Transforms that should be applied to the input images. Defaults to `None`.
- **`split`** (`str` | *`Split`* | `None`) – 'train', 'val' or 'test'
- **`task`** (`TaskType` | `str`) – `classification`, `detection` or `segmentation`
- **`create_validation_set`** – Create a validation subset in addition to the train and test subsets

Examples

```
>>> from anomalib.data.image.btech import BTechDataset
>>> from anomalib.data.utils.transforms import get_transforms
>>> transform = get_transforms(image_size=256)
>>> dataset = BTechDataset(
...     task="classification",
...     transform=transform,
...     root='./datasets/BTech',
...     category='01',
... )
>>> dataset[0].keys()
>>> dataset.setup()
dict_keys(['image'])
```

```
>>> dataset.split = "test"
>>> dataset[0].keys()
dict_keys(['image', 'image_path', 'label'])
```

```
>>> dataset.task = "segmentation"
>>> dataset.split = "train"
>>> dataset[0].keys()
dict_keys(['image'])
```

```
>>> dataset.split = "test"
>>> dataset[0].keys()
dict_keys(['image_path', 'label', 'mask_path', 'image', 'mask'])
```

```
>>> dataset[0]["image"].shape, dataset[0]["mask"].shape
(torch.Size([3, 256, 256]), torch.Size([256, 256]))
```

anomalib.data.image.btech.**make_btech_dataset**(*path*, *split=None*)

Create BTech samples by parsing the BTech data file structure.

The files are expected to follow the structure:

```
path/to/dataset/split/category/image_filename.png
path/to/dataset/ground_truth/category/mask_filename.png
```

Parameters

- **path** (*Path*) – Path to dataset
- **split** (*str | Split | None, optional*) – Dataset split (ie., either train or test). Defaults to None.

Example

The following example shows how to get training samples from BTech 01 category:

```
>>> root = Path('./BTech')
>>> category = '01'
>>> path = root / category
>>> path
PosixPath('BTech/01')

>>> samples = make_btech_dataset(path, split='train')
>>> samples.head()
path     split label image_path                  mask_path                         ␣
↪label_index
0  BTech/01 train 01    BTech/01/train/ok/105.bmp BTech/01/ground_truth/ok/105.png ␣
↪    0
1  BTech/01 train 01    BTech/01/train/ok/017.bmp BTech/01/ground_truth/ok/017.png ␣
↪    0
...
```

Returns

an output dataframe containing samples for the requested split (ie., train or test)

Return type

DataFrame

Folder Data

Custom Folder Dataset.

This script creates a custom dataset from a folder.

class `anomalib.data.image.folder.`**`Folder`**(*name*, *normal_dir*, *root=None*, *abnormal_dir=None*, *normal_test_dir=None*, *mask_dir=None*, *normal_split_ratio=0.2*, *extensions=None*, *train_batch_size=32*, *eval_batch_size=32*, *num_workers=8*, *task=TaskType.SEGMENTATION*, *image_size=None*, *transform=None*, *train_transform=None*, *eval_transform=None*, *test_split_mode=TestSplitMode.FROM_DIR*, *test_split_ratio=0.2*, *val_split_mode=ValSplitMode.FROM_TEST*, *val_split_ratio=0.5*, *seed=None*)

Bases: *AnomalibDataModule*

Folder DataModule.

Parameters

- **name** (*str*) – Name of the dataset. This is used to name the datamodule, especially when logging/saving.
- **normal_dir** (*str | Path | Sequence*) – Name of the directory containing normal images.
- **root** (*str | Path | None*) – Path to the root folder containing normal and abnormal dirs. Defaults to `None`.
- **abnormal_dir** (*str | Path | None | Sequence*) – Name of the directory containing abnormal images. Defaults to `None`.
- **normal_test_dir** (*str | Path | Sequence | None, optional*) – Path to the directory containing normal images for the test dataset. Defaults to `None`.
- **mask_dir** (*str | Path | Sequence | None, optional*) – Path to the directory containing the mask annotations. Defaults to `None`.
- **normal_split_ratio** (*float, optional*) – Ratio to split normal training images and add to the test set in case test set doesn't contain any normal images. Defaults to 0.2.
- **extensions** (*tuple[str, ...] | None, optional*) – Type of the image extensions to read from the directory. Defaults to `None`.
- **train_batch_size** (*int, optional*) – Training batch size. Defaults to `32`.
- **eval_batch_size** (*int, optional*) – Validation, test and predict batch size. Defaults to `32`.
- **num_workers** (*int, optional*) – Number of workers. Defaults to `8`.
- **task** (*TaskType, optional*) – Task type. Could be `classification`, `detection` or `segmentation`. Defaults to `segmentation`.
- **image_size** (*tuple[int, int], optional*) – Size to which input images should be resized. Defaults to `None`.
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to `None`.

- **train_transform** (*Transform, optional*) – Transforms that should be applied to the input images during training. Defaults to None.
- **eval_transform** (*Transform, optional*) – Transforms that should be applied to the input images during evaluation. Defaults to None.
- **test_split_mode** (TestSplitMode) – Setting that determines how the testing subset is obtained. Defaults to TestSplitMode.FROM_DIR.
- **test_split_ratio** (*float*) – Fraction of images from the train set that will be reserved for testing. Defaults to 0.2.
- **val_split_mode** (ValSplitMode) – Setting that determines how the validation subset is obtained. Defaults to ValSplitMode.FROM_TEST.
- **val_split_ratio** (*float*) – Fraction of train or test images that will be reserved for validation. Defaults to 0.5.
- **seed** (*int | None, optional*) – Seed used during random subset splitting. Defaults to None.

Examples

The following code demonstrates how to use the Folder datamodule. Assume that the dataset is structured as follows:

```
$ tree sample_dataset
sample_dataset
├── colour
│   ├── 00.jpg
│   ├── ...
│   └── x.jpg
├── crack
│   ├── 00.jpg
│   ├── ...
│   └── y.jpg
├── good
│   ├── ...
│   └── z.jpg
├── LICENSE
└── mask
    ├── colour
    │   ├── ...
    │   └── x.jpg
    └── crack
        ├── ...
        └── y.jpg
```

```
folder_datamodule = Folder(
    root=dataset_root,
    normal_dir="good",
    abnormal_dir="crack",
    task=TaskType.SEGMENTATION,
    mask_dir=dataset_root / "mask" / "crack",
    image_size=256,
```

(continues on next page)

(continued from previous page)

```
    normalization=InputNormalizationMethod.NONE,
)
folder_datamodule.setup()
```

To access the training images,

```
>> i, data = next(enumerate(folder_datamodule.train_dataloader()))
>> print(data.keys(), data["image"].shape)
```

To access the test images,

```
>> i, data = next(enumerate(folder_datamodule.test_dataloader()))
>> print(data.keys(), data["image"].shape)
```

property name: str

Name of the datamodule.

Folder datamodule overrides the name property to provide a custom name.

class anomalib.data.image.folder.**FolderDataset**(*name*, *task*, *normal_dir*, *transform=None*, *root=None*, *abnormal_dir=None*, *normal_test_dir=None*, *mask_dir=None*, *split=None*, *extensions=None*)

Bases: *AnomalibDataset*

Folder dataset.

This class is used to create a dataset from a folder. The class utilizes the Torch Dataset class.

Parameters

- **name** (*str*) – Name of the dataset. This is used to name the datamodule, especially when logging/saving.
- **task** (*TaskType*) – Task type. (classification, detection or segmentation).
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to None.
- **normal_dir** (*str | Path | Sequence*) – Path to the directory containing normal images.
- **root** (*str | Path | None*) – Root folder of the dataset. Defaults to None.
- **abnormal_dir** (*str | Path | Sequence | None, optional*) – Path to the directory containing abnormal images. Defaults to None.
- **normal_test_dir** (*str | Path | Sequence | None, optional*) – Path to the directory containing normal images for the test dataset. Defaults to None.
- **mask_dir** (*str | Path | Sequence | None, optional*) – Path to the directory containing the mask annotations. Defaults to None.
- **split** (*str | Split | None*) – Fixed subset split that follows from folder structure on file system. Choose from [Split.FULL, Split.TRAIN, Split.TEST] Defaults to None.
- **extensions** (*tuple[str, ...] | None, optional*) – Type of the image extensions to read from the directory. Defaults to None.

Raises

ValueError – When task is set to classification and *mask_dir* is provided. When *mask_dir* is provided, *task* should be set to *segmentation*.

Examples

Assume that we would like to use this `FolderDataset` to create a dataset from a folder for a classification task. We could first create the transforms,

```
>>> from anomalib.data.utils import InputNormalizationMethod, get_transforms
>>> transform = get_transforms(image_size=256,␣
↪normalization=InputNormalizationMethod.NONE)
```

We could then create the dataset as follows,

```
folder_dataset_classification_train = FolderDataset(
    normal_dir=dataset_root / "good",
    abnormal_dir=dataset_root / "crack",
    split="train",
    transform=transform,
    task=TaskType.CLASSIFICATION,
)
```

property name: str

Name of the dataset.

Folder dataset overrides the name property to provide a custom name.

`anomalib.data.image.folder.`**`make_folder_dataset`**(*normal_dir*, *root=None*, *abnormal_dir=None*, *normal_test_dir=None*, *mask_dir=None*, *split=None*, *extensions=None*)

Make Folder Dataset.

Parameters

- **normal_dir** (*str | Path | Sequence*) – Path to the directory containing normal images.
- **root** (*str | Path | None*) – Path to the root directory of the dataset. Defaults to `None`.
- **abnormal_dir** (*str | Path | Sequence | None, optional*) – Path to the directory containing abnormal images. Defaults to `None`.
- **normal_test_dir** (*str | Path | Sequence | None, optional*) – Path to the directory containing normal images for the test dataset. Normal test images will be a split of *normal_dir* if *None*. Defaults to `None`.
- **mask_dir** (*str | Path | Sequence | None, optional*) – Path to the directory containing the mask annotations. Defaults to `None`.
- **split** (*str | Split | None, optional*) – Dataset split (ie., Split.FULL, Split.TRAIN or Split.TEST). Defaults to `None`.
- **extensions** (*tuple[str, ...] | None, optional*) – Type of the image extensions to read from the directory. Defaults to `None`.

Returns

an output dataframe containing samples for the requested split (ie., train or test).

Return type

DataFrame

Examples

Assume that we would like to use this `make_folder_dataset` to create a dataset from a folder. We could then create the dataset as follows,

```
folder_df = make_folder_dataset(
    normal_dir=dataset_root / "good",
    abnormal_dir=dataset_root / "crack",
    split="train",
)
folder_df.head()
```

```
          image_path           label  label_index mask_path        split
0  ./toy/good/00.jpg  DirType.NORMAL            0            Split.TRAIN
1  ./toy/good/01.jpg  DirType.NORMAL            0            Split.TRAIN
2  ./toy/good/02.jpg  DirType.NORMAL            0            Split.TRAIN
3  ./toy/good/03.jpg  DirType.NORMAL            0            Split.TRAIN
4  ./toy/good/04.jpg  DirType.NORMAL            0            Split.TRAIN
```

Kolektor Data

Kolektor Surface-Defect Dataset (CC BY-NC-SA 4.0).

Description:
This script provides a PyTorch Dataset, DataLoader, and PyTorch Lightning DataModule for the Kolektor Surface-Defect dataset. The dataset can be accessed at Kolektor Surface-Defect Dataset.

License:
The Kolektor Surface-Defect dataset is released under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License (CC BY-NC-SA 4.0). For more details, visit Creative Commons License.

Reference:
Tabernik, Domen, Samo Šela, Jure Skvarč, and Danijel Skočaj. “Segmentation-based deep-learning approach for surface-defect detection.” Journal of Intelligent Manufacturing 31, no. 3 (2020): 759-776.

class `anomalib.data.image.kolektor.`**`Kolektor`**(*root='./datasets/kolektor'*, *train_batch_size=32*, *eval_batch_size=32*, *num_workers=8*, *task=TaskType.SEGMENTATION*, *image_size=None*, *transform=None*, *train_transform=None*, *eval_transform=None*, *test_split_mode=TestSplitMode.FROM_DIR*, *test_split_ratio=0.2*, *val_split_mode=ValSplitMode.SAME_AS_TEST*, *val_split_ratio=0.5*, *seed=None*)

Bases: *`AnomalibDataModule`*

Kolektor Datamodule.

Parameters

- **root** (*Path | str*) – Path to the root of the dataset
- **train_batch_size** (*int, optional*) – Training batch size. Defaults to 32.
- **eval_batch_size** (*int, optional*) – Test batch size. Defaults to 32.
- **num_workers** (*int, optional*) – Number of workers. Defaults to 8.

- **TaskType)** (*task*) – Task type, 'classification', 'detection' or 'segmentation' Defaults to `TaskType.SEGMENTATION`.
- **image_size** (*tuple[int, int], optional*) – Size to which input images should be resized. Defaults to `None`.
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to `None`.
- **train_transform** (*Transform, optional*) – Transforms that should be applied to the input images during training. Defaults to `None`.
- **eval_transform** (*Transform, optional*) – Transforms that should be applied to the input images during evaluation. Defaults to `None`.
- **test_split_mode** (`TestSplitMode`) – Setting that determines how the testing subset is obtained. Defaults to `TestSplitMode.FROM_DIR`
- **test_split_ratio** (*float*) – Fraction of images from the train set that will be reserved for testing. Defaults to `0.2`
- **val_split_mode** (`ValSplitMode`) – Setting that determines how the validation subset is obtained. Defaults to `ValSplitMode.SAME_AS_TEST`
- **val_split_ratio** (*float*) – Fraction of train or test images that will be reserved for validation. Defaults to `0.5`
- **seed** (*int | None, optional*) – Seed which may be set to a fixed value for reproducibility. Defaults to `None`.

prepare_data()

Download the dataset if not available.

This method checks if the specified dataset is available in the file system. If not, it downloads and extracts the dataset into the appropriate directory.

Return type
`None`

Example

Assume the dataset is not available on the file system. Here's how the directory structure looks before and after calling the *prepare_data* method:

Before:

```
$ tree datasets
datasets
├── dataset1
└── dataset2
```

Calling the method:

```
>> datamodule = Kolektor(root="./datasets/kolektor")
>> datamodule.prepare_data()
```

After:

```
$ tree datasets
datasets
├── dataset1
├── dataset2
└── kolektor
    ├── kolektorsdd
    ├── kos01
    ├── ...
    └── kos50
        ├── Part0.jpg
        ├── Part0_label.bmp
        └── ...
```

class `anomalib.data.image.kolektor.`**`KolektorDataset`**(*task*, *root='./datasets/kolektor'*, *transform=None*, *split=None*)

Bases: `AnomalibDataset`

Kolektor dataset class.

Parameters

- **task** (*TaskType*) – Task type, `classification`, `detection` or `segmentation`
- **root** (*Path | str*) – Path to the root of the dataset Defaults to `./datasets/kolektor`.
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to `None`.
- **split** (*str | Split | None*) – Split of the dataset, usually Split.TRAIN or Split.TEST Defaults to `None`.

`anomalib.data.image.kolektor.`**`make_kolektor_dataset`**(*root*, *train_split_ratio=0.8*, *split=None*)

Create Kolektor samples by parsing the Kolektor data file structure.

The files are expected to follow this structure: - Image files: *path/to/dataset/item/image_filename.jpg*, *path/to/dataset/kos01/Part0.jpg* - Mask files: *path/to/dataset/item/mask_filename.bmp*, *path/to/dataset/kos01/Part0_label.bmp*

This function creates a DataFrame to store the parsed information in the following format:

	path	item	split	label	image_path	mask_path	label_index
0	KolektorSDD	kos01	test	Bad	/path/to/image_file	/path/to/mask_file	1

Parameters

- **root** (*Path*) – Path to the dataset.
- **train_split_ratio** (*float, optional*) – Ratio for splitting good images into train/test sets. Defaults to `0.8`.
- **split** (*str | Split | None, optional*) – Dataset split (either 'train' or 'test'). Defaults to `None`.

Returns

An output DataFrame containing the samples of the dataset.

Return type

pandas.DataFrame

Example

The following example shows how to get training samples from the Kolektor Dataset:

```
>>> from pathlib import Path
>>> root = Path('./KolektorSDD/')
>>> samples = create_kolektor_samples(root, train_split_ratio=0.8)
>>> samples.head()
       path       item  split label   image_path                    mask_path      ␣
↪           label_index
  0   KolektorSDD   kos01  train Good  KolektorSDD/kos01/Part0.jpg  KolektorSDD/
↪kos01/Part0_label.bmp  0
  1   KolektorSDD   kos01  train Good  KolektorSDD/kos01/Part1.jpg  KolektorSDD/
↪kos01/Part1_label.bmp  0
  2   KolektorSDD   kos01  train Good  KolektorSDD/kos01/Part2.jpg  KolektorSDD/
↪kos01/Part2_label.bmp  0
  3   KolektorSDD   kos01  test  Good  KolektorSDD/kos01/Part3.jpg  KolektorSDD/
↪kos01/Part3_label.bmp  0
  4   KolektorSDD   kos01  train Good  KolektorSDD/kos01/Part4.jpg  KolektorSDD/
↪kos01/Part4_label.bmp  0
```

MVTec Data

MVTec AD Dataset (CC BY-NC-SA 4.0).

Description:
This script contains PyTorch Dataset, Dataloader and PyTorch Lightning DataModule for the MVTec AD dataset. If the dataset is not on the file system, the script downloads and extracts the dataset and create PyTorch data objects.

License:
MVTec AD dataset is released under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License (CC BY-NC-SA 4.0)(https://creativecommons.org/licenses/by-nc-sa/4.0/).

References

- Paul Bergmann, Kilian Batzner, Michael Fauser, David Sattlegger, Carsten Steger: The MVTec Anomaly Detection Dataset: A Comprehensive Real-World Dataset for Unsupervised Anomaly Detection; in: International Journal of Computer Vision 129(4):1038-1059, 2021, DOI: 10.1007/s11263-020-01400-4.
- Paul Bergmann, Michael Fauser, David Sattlegger, Carsten Steger: MVTec AD — A Comprehensive Real-World Dataset for Unsupervised Anomaly Detection; in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 9584-9592, 2019, DOI: 10.1109/CVPR.2019.00982.

`class anomalib.data.image.mvtec.MVTec`(*root='./datasets/MVTec'*, *category='bottle'*, *train_batch_size=32*, *eval_batch_size=32*, *num_workers=8*, *task=TaskType.SEGMENTATION*, *image_size=None*, *transform=None*, *train_transform=None*, *eval_transform=None*, *test_split_mode=TestSplitMode.FROM_DIR*, *test_split_ratio=0.2*, *val_split_mode=ValSplitMode.SAME_AS_TEST*, *val_split_ratio=0.5*, *seed=None*)

Bases: `AnomalibDataModule`

MVTec Datamodule.

Parameters

- **root** (*Path | str*) – Path to the root of the dataset. Defaults to `"./datasets/MVTec"`.
- **category** (*str*) – Category of the MVTec dataset (e.g. "bottle" or "cable"). Defaults to `"bottle"`.
- **train_batch_size** (*int, optional*) – Training batch size. Defaults to `32`.
- **eval_batch_size** (*int, optional*) – Test batch size. Defaults to `32`.
- **num_workers** (*int, optional*) – Number of workers. Defaults to `8`.
- **TaskType)** (*task*) – Task type, 'classification', 'detection' or 'segmentation' Defaults to `TaskType.SEGMENTATION`.
- **image_size** (*tuple[int, int], optional*) – Size to which input images should be resized. Defaults to `None`.
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to `None`.
- **train_transform** (*Transform, optional*) – Transforms that should be applied to the input images during training. Defaults to `None`.
- **eval_transform** (*Transform, optional*) – Transforms that should be applied to the input images during evaluation. Defaults to `None`.
- **test_split_mode** (`TestSplitMode`) – Setting that determines how the testing subset is obtained. Defaults to `TestSplitMode.FROM_DIR`.
- **test_split_ratio** (*float*) – Fraction of images from the train set that will be reserved for testing. Defaults to `0.2`.
- **val_split_mode** (`ValSplitMode`) – Setting that determines how the validation subset is obtained. Defaults to `ValSplitMode.SAME_AS_TEST`.
- **val_split_ratio** (*float*) – Fraction of train or test images that will be reserved for validation. Defaults to `0.5`.
- **seed** (*int | None, optional*) – Seed which may be set to a fixed value for reproducibility. Defualts to `None`.

Examples

To create an MVTec AD datamodule with default settings:

```
>>> datamodule = MVTec()
>>> datamodule.setup()
>>> i, data = next(enumerate(datamodule.train_dataloader()))
>>> data.keys()
dict_keys(['image_path', 'label', 'image', 'mask_path', 'mask'])
```

```
>>> data["image"].shape
torch.Size([32, 3, 256, 256])
```

To change the category of the dataset:

```
>>> datamodule = MVTec(category="cable")
```

To change the image and batch size:

```
>>> datamodule = MVTec(image_size=(512, 512), train_batch_size=16, eval_batch_
↪size=8)
```

MVTec AD dataset does not provide a validation set. If you would like to use a separate validation set, you can use the `val_split_mode` and `val_split_ratio` arguments to create a validation set.

```
>>> datamodule = MVTec(val_split_mode=ValSplitMode.FROM_TEST, val_split_ratio=0.1)
```

This will subsample the test set by 10% and use it as the validation set. If you would like to create a validation set synthetically that would not change the test set, you can use the `ValSplitMode.SYNTHETIC` option.

```
>>> datamodule = MVTec(val_split_mode=ValSplitMode.SYNTHETIC, val_split_ratio=0.2)
```

prepare_data()

Download the dataset if not available.

This method checks if the specified dataset is available in the file system. If not, it downloads and extracts the dataset into the appropriate directory.

Return type
`None`

Example

Assume the dataset is not available on the file system. Here's how the directory structure looks before and after calling the *prepare_data* method:

Before:

```
$ tree datasets
datasets
├── dataset1
└── dataset2
```

Calling the method:

```
>> datamodule = MVTec(root="./datasets/MVTec", category="bottle")
>> datamodule.prepare_data()
```

After:

```
$ tree datasets
datasets
├── dataset1
├── dataset2
└── MVTec
    ├── bottle
    ├── ...
    └── zipper
```

class anomalib.data.image.mvtec.**MVTecDataset**(*task*, *root='./datasets/MVTec'*, *category='bottle'*, *transform=None*, *split=None*)

Bases: *AnomalibDataset*

MVTec dataset class.

Parameters

- **task** (*TaskType*) – Task type, `classification`, `detection` or `segmentation`.
- **root** (*Path | str*) – Path to the root of the dataset. Defaults to `./datasets/MVTec`.
- **category** (*str*) – Sub-category of the dataset, e.g. 'bottle' Defaults to `bottle`.
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to `None`.
- **split** (*str | Split | None*) – Split of the dataset, usually Split.TRAIN or Split.TEST Defaults to `None`.

Examples

```
from anomalib.data.image.mvtec import MVTecDataset
from anomalib.data.utils.transforms import get_transforms

transform = get_transforms(image_size=256)
dataset = MVTecDataset(
    task="classification",
    transform=transform,
    root='./datasets/MVTec',
    category='zipper',
)
dataset.setup()
print(dataset[0].keys())
# Output: dict_keys(['image_path', 'label', 'image'])
```

When the task is segmentation, the dataset will also contain the mask:

```
dataset.task = "segmentation"
dataset.setup()
print(dataset[0].keys())
# Output: dict_keys(['image_path', 'label', 'image', 'mask_path', 'mask'])
```

The image is a torch tensor of shape (C, H, W) and the mask is a torch tensor of shape (H, W).

```
print(dataset[0]["image"].shape, dataset[0]["mask"].shape)
# Output: (torch.Size([3, 256, 256]), torch.Size([256, 256]))
```

`anomalib.data.image.mvtec.`**`make_mvtec_dataset`**(*root*, *split=None*, *extensions=None*)

Create MVTec AD samples by parsing the MVTec AD data file structure.

The files are expected to follow the structure:
path/to/dataset/split/category/image_filename.png path/to/dataset/ground_truth/category/mask_filename.png

This function creates a dataframe to store the parsed information based on the following format:

	path	split	label	image_path	mask_path	label_index
0	datasets/name	test	defect	filename.png	ground_truth/defect/filename_mask.png	1

Parameters

- **root** (*Path*) – Path to dataset
- **split** (*str | Split | None, optional*) – Dataset split (ie., either train or test). Defaults to None.
- **extensions** (*Sequence[str] | None, optional*) – List of file extensions to be included in the dataset. Defaults to None.

Examples

The following example shows how to get training samples from MVTec AD bottle category:

```
>>> root = Path('./MVTec')
>>> category = 'bottle'
>>> path = root / category
>>> path
PosixPath('MVTec/bottle')
```

```
>>> samples = make_mvtec_dataset(path, split='train', split_ratio=0.1, seed=0)
>>> samples.head()
   path         split label image_path                           mask_path          ␣
↪         label_index
0  MVTec/bottle train good MVTec/bottle/train/good/105.png MVTec/bottle/ground_
↪truth/good/105_mask.png 0
1  MVTec/bottle train good MVTec/bottle/train/good/017.png MVTec/bottle/ground_
↪truth/good/017_mask.png 0
2  MVTec/bottle train good MVTec/bottle/train/good/137.png MVTec/bottle/ground_
↪truth/good/137_mask.png 0
3  MVTec/bottle train good MVTec/bottle/train/good/152.png MVTec/bottle/ground_
↪truth/good/152_mask.png 0
4  MVTec/bottle train good MVTec/bottle/train/good/109.png MVTec/bottle/ground_
↪truth/good/109_mask.png 0
```

Returns
an output dataframe containing the samples of the dataset.

Return type
DataFrame

Visa Data

Visual Anomaly (VisA) Dataset (CC BY-NC-SA 4.0).

Description:

This script contains PyTorch Dataset, Dataloader and PyTorch
Lightning DataModule for the Visual Anomal (VisA) dataset.

If the dataset is not on the file system, the script downloads and
extracts the dataset and create PyTorch data objects.

License:
The VisA dataset is released under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License (CC BY-NC-SA 4.0)(https://creativecommons.org/licenses/by-nc-sa/4.0/).

Reference:

- Zou, Y., Jeong, J., Pemula, L., Zhang, D., & Dabeer, O. (2022). SPot-the-Difference Self-supervised Pre-training for Anomaly Detection and Segmentation. In European Conference on Computer Vision (pp. 392-408). Springer, Cham.

`class anomalib.data.image.visa.`**`Visa`**(*root='./datasets/visa'*, *category='capsules'*, *train_batch_size=32*, *eval_batch_size=32*, *num_workers=8*, *task=TaskType.SEGMENTATION*, *image_size=None*, *transform=None*, *train_transform=None*, *eval_transform=None*, *test_split_mode=TestSplitMode.FROM_DIR*, *test_split_ratio=0.2*, *val_split_mode=ValSplitMode.SAME_AS_TEST*, *val_split_ratio=0.5*, *seed=None*)

Bases: `AnomalibDataModule`

VisA Datamodule.

Parameters

- **root** (*Path | str*) – Path to the root of the dataset Defaults to `"./datasets/visa"`.
- **category** (*str*) – Category of the Visa dataset such as `candle`. Defaults to `"candle"`.
- **train_batch_size** (*int, optional*) – Training batch size. Defaults to `32`.
- **eval_batch_size** (*int, optional*) – Test batch size. Defaults to `32`.
- **num_workers** (*int, optional*) – Number of workers. Defaults to `8`.
- **task** (*TaskType*) – Task type, 'classification', 'detection' or 'segmentation' Defaults to `TaskType.SEGMENTATION`.
- **image_size** (*tuple[int, int], optional*) – Size to which input images should be resized. Defaults to `None`.
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to `None`.
- **train_transform** (*Transform, optional*) – Transforms that should be applied to the input images during training. Defaults to `None`.
- **eval_transform** (*Transform, optional*) – Transforms that should be applied to the input images during evaluation. Defaults to `None`.
- **test_split_mode** (`TestSplitMode`) – Setting that determines how the testing subset is obtained. Defaults to `TestSplitMode.FROM_DIR`.
- **test_split_ratio** (*float*) – Fraction of images from the train set that will be reserved for testing. Defaults to `0.2`.
- **val_split_mode** (`ValSplitMode`) – Setting that determines how the validation subset is obtained. Defaults to `ValSplitMode.SAME_AS_TEST`.
- **val_split_ratio** (*float*) – Fraction of train or test images that will be reserved for validation. Defatuls to `0.5`.
- **seed** (*int | None, optional*) – Seed which may be set to a fixed value for reproducibility. Defaults to `None`.

`apply_cls1_split`()

Apply the 1-class subset splitting using the fixed split in the csv file.

adapted from https://github.com/amazon-science/spot-diff

Return type
None

`prepare_data()`

Download the dataset if not available.

This method checks if the specified dataset is available in the file system. If not, it downloads and extracts the dataset into the appropriate directory.

Return type
None

Example

Assume the dataset is not available on the file system. Here's how the directory structure looks before and after calling the *prepare_data* method:

Before:

```
$ tree datasets
datasets
├── dataset1
└── dataset2
```

Calling the method:

```
>> datamodule = Visa()
>> datamodule.prepare_data()
```

After:

```
$ tree datasets
datasets
├── dataset1
├── dataset2
└── visa
    ├── candle
    ├── ...
    ├── pipe_fryum
    │   ├── Data
    │   └── image_anno.csv
    ├── split_csv
    │   ├── 1cls.csv
    │   ├── 2cls_fewshot.csv
    │   └── 2cls_highshot.csv
    ├── VisA_20220922.tar
    └── visa_pytorch
        ├── candle
        ├── ...
        ├── pcb4
        └── pipe_fryum
```

`prepare_data` ensures that the dataset is converted to MVTec format. `visa_pytorch` is the directory that contains the dataset in the MVTec format. `visa` is the directory that contains the original dataset.

class anomalib.data.image.visa.**VisaDataset**(*task*, *root*, *category*, *transform=None*, *split=None*)

Bases: `AnomalibDataset`

VisA dataset class.

Parameters

- **task** (*TaskType*) – Task type, `classification`, `detection` or `segmentation`
- **root** (*str | Path*) – Path to the root of the dataset
- **category** (*str*) – Sub-category of the dataset, e.g. 'candle'
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to `None`.
- **split** (*str | Split | None*) – Split of the dataset, usually Split.TRAIN or Split.TEST Defaults to `None`.

Examples

To create a Visa dataset for classification:

```
from anomalib.data.image.visa import VisaDataset
from anomalib.data.utils.transforms import get_transforms

transform = get_transforms(image_size=256)
dataset = VisaDataset(
    task="classification",
    transform=transform,
    split="train",
    root="./datasets/visa/visa_pytorch/",
    category="candle",
)
dataset.setup()
dataset[0].keys()

# Output
dict_keys(['image_path', 'label', 'image'])
```

If you want to use the dataset for segmentation, you can use the same code as above, with the task set to `segmentation`. The dataset will then have a `mask` key in the output dictionary.

```
from anomalib.data.image.visa import VisaDataset
from anomalib.data.utils.transforms import get_transforms

transform = get_transforms(image_size=256)
dataset = VisaDataset(
    task="segmentation",
    transform=transform,
    split="train",
    root="./datasets/visa/visa_pytorch/",
    category="candle",
)
dataset.setup()
dataset[0].keys()
```

(continues on next page)

(continued from previous page)

```
# Output
dict_keys(['image_path', 'label', 'image', 'mask_path', 'mask'])
```

Video Data

Avenue   Learn more about Avenue dataset.

Shanghai Tech   Learn more about Shanghai Tech dataset.

UCSD   Learn more about UCSD Ped1 and Ped2 datasets.

Avenue Data

CUHK Avenue Dataset.

Description:
This module provides a PyTorch Dataset and PyTorch Lightning DataModule for the CUHK Avenue dataset. If the dataset is not already present on the file system, the DataModule class will download and extract the dataset, converting the .mat mask files to .png format.

Reference:

- Lu, Cewu, Jianping Shi, and Jiaya Jia. "Abnormal event detection at 150 fps in Matlab." In Proceedings of the IEEE International Conference on Computer Vision, 2013.

`class anomalib.data.video.avenue.`**`Avenue`**(*root='./datasets/avenue'*, *gt_dir='./datasets/avenue/ground_truth_demo'*, *clip_length_in_frames=2*, *frames_between_clips=1*, *target_frame=VideoTargetFrame.LAST*, *task=TaskType.SEGMENTATION*, *image_size=None*, *transform=None*, *train_transform=None*, *eval_transform=None*, *train_batch_size=32*, *eval_batch_size=32*, *num_workers=8*, *val_split_mode=ValSplitMode.SAME_AS_TEST*, *val_split_ratio=0.5*, *seed=None*)

Bases: *AnomalibVideoDataModule*

Avenue DataModule class.

Parameters

- **root** (*Path | str*) – Path to the root of the dataset Defaults to `./datasets/avenue`.
- **gt_dir** (*Path | str*) – Path to the ground truth files Defaults to `./datasets/avenue/ground_truth_demo`.
- **clip_length_in_frames** (*int, optional*) – Number of video frames in each clip. Defaults to `2`.
- **frames_between_clips** (*int, optional*) – Number of frames between each consecutive video clip. Defaults to `1`.
- **target_frame** (`VideoTargetFrame`) – Specifies the target frame in the video clip, used for ground truth retrieval Defaults to `VideoTargetFrame.LAST`.

- **task** (*TaskType*) – Task type, 'classification', 'detection' or 'segmentation' Defaults to `TaskType.SEGMENTATION`.
- **image_size** (*tuple[int, int], optional*) – Size to which input images should be resized. Defaults to `None`.
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to `None`.
- **train_transform** (*Transform, optional*) – Transforms that should be applied to the input images during training. Defaults to `None`.
- **eval_transform** (*Transform, optional*) – Transforms that should be applied to the input images during evaluation. Defaults to `None`.
- **train_batch_size** (*int, optional*) – Training batch size. Defaults to `32`.
- **eval_batch_size** (*int, optional*) – Test batch size. Defaults to `32`.
- **num_workers** (*int, optional*) – Number of workers. Defaults to `8`.
- **val_split_mode** (`ValSplitMode`) – Setting that determines how the validation subset is obtained. Defaults to `ValSplitMode.FROM_TEST`.
- **val_split_ratio** (*float*) – Fraction of train or test images that will be reserved for validation. Defaults to `0.5`.
- **seed** (*int | None, optional*) – Seed which may be set to a fixed value for reproducibility. Defaults to `None`.

Examples

To create a DataModule for Avenue dataset with default parameters:

```
datamodule = Avenue()
datamodule.setup()

i, data = next(enumerate(datamodule.train_dataloader()))
data.keys()
# Output: dict_keys(['image', 'video_path', 'frames', 'last_frame', 'original_image'])

i, data = next(enumerate(datamodule.test_dataloader()))
data.keys()
# Output: dict_keys(['image', 'mask', 'video_path', 'frames', 'last_frame', 'original_image
↪', 'label'])

data["image"].shape
# Output: torch.Size([32, 2, 3, 256, 256])
```

Note that the default task type is segmentation and the dataloader returns a mask in addition to the input. Also, it is important to note that the dataloader returns a batch of clips, where each clip is a sequence of frames. The number of frames in each clip is determined by the `clip_length_in_frames` parameter. The `frames_between_clips` parameter determines the number of frames between each consecutive clip. The `target_frame` parameter determines which frame in the clip is used for ground truth retrieval. For example, if `clip_length_in_frames=2`, `frames_between_clips=1` and `target_frame=VideoTargetFrame.LAST`, then the dataloader will return a batch of clips where each clip contains two consecutive frames from the video. The second frame in each clip will be used as the ground truth for the first frame in the clip. The following code shows how to create a dataloader for classification:

```
datamodule = Avenue(
    task="classification",
    clip_length_in_frames=2,
    frames_between_clips=1,
    target_frame=VideoTargetFrame.LAST
)
datamodule.setup()

i, data = next(enumerate(datamodule.train_dataloader()))
data.keys()
# Output: dict_keys(['image', 'video_path', 'frames', 'last_frame', 'original_image'])

data["image"].shape
# Output: torch.Size([32, 2, 3, 256, 256])
```

prepare_data()

Download the dataset if not available.

This method checks if the specified dataset is available in the file system. If not, it downloads and extracts the dataset into the appropriate directory.

Return type
None

Example

Assume the dataset is not available on the file system. Here's how the directory structure looks before and after calling the *prepare_data* method:

Before:

```
$ tree datasets
datasets
├── dataset1
└── dataset2
```

Calling the method:

```
>> datamodule = Avenue()
>> datamodule.prepare_data()
```

After:

```
$ tree datasets
datasets
├── dataset1
├── dataset2
└── avenue
    ├── ground_truth_demo
    │   ├── ground_truth_show.m
    │   ├── Readme.txt
    │   ├── testing_label_mask
    │   └── testing_videos
    ├── testing_videos
```

(continues on next page)

(continued from previous page)

```
    │   ├── ...
    │   └── 21.avi
    ├── testing_vol
    │   ├── ...
    │   └── vol21.mat
    ├── training_videos
    │   ├── ...
    │   └── 16.avi
    └── training_vol
        ├── ...
        └── vol16.mat
```

class anomalib.data.video.avenue.**AvenueDataset**(*task*, *split*, *root='./datasets/avenue'*, *gt_dir='./datasets/avenue/ground_truth_demo'*, *clip_length_in_frames=2*, *frames_between_clips=1*, *transform=None*, *target_frame=VideoTargetFrame.LAST*)

Bases: *AnomalibVideoDataset*

Avenue Dataset class.

Parameters

- **task** (*TaskType*) – Task type, 'classification', 'detection' or 'segmentation'
- **split** (Split) – Split of the dataset, usually Split.TRAIN or Split.TEST
- **root** (*Path | str*) – Path to the root of the dataset Defaults to `./datasets/avenue`.
- **gt_dir** (*Path | str*) – Path to the ground truth files Defaults to `./datasets/avenue/ground_truth_demo`.
- **clip_length_in_frames** (*int, optional*) – Number of video frames in each clip. Defaults to 2.
- **frames_between_clips** (*int, optional*) – Number of frames between each consecutive video clip. Defaults to 1.
- **target_frame** (VideoTargetFrame) – Specifies the target frame in the video clip, used for ground truth retrieval. Defaults to `VideoTargetFrame.LAST`.
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to `None`.

Examples

To create an Avenue dataset to train a classification model:

```
transform = A.Compose([A.Resize(256, 256), A.pytorch.ToTensorV2()])
dataset = AvenueDataset(
    task="classification",
    transform=transform,
    split="train",
    root="./datasets/avenue/",
)
```

(continues on next page)

(continued from previous page)

```
dataset.setup()
dataset[0].keys()

# Output: dict_keys(['image', 'video_path', 'frames', 'last_frame', 'original_image'])
```

If you would like to test a segmentation model, you can use the following code:

```
dataset = AvenueDataset(
    task="segmentation",
    transform=transform,
    split="test",
    root="./datasets/avenue/",
)

dataset.setup()
dataset[0].keys()

# Output: dict_keys(['image', 'mask', 'video_path', 'frames', 'last_frame', 'original_image
↪', 'label'])
```

Avenue video dataset can also be used as an image dataset if you set the clip length to 1. This means that each video frame will be treated as a separate sample. This is useful for training a classification model on the Avenue dataset. The following code shows how to create an image dataset for classification:

```
dataset = AvenueDataset(
    task="classification",
    transform=transform,
    split="test",
    root="./datasets/avenue/",
    clip_length_in_frames=1,
)

dataset.setup()
dataset[0].keys()
# Output: dict_keys(['image', 'video_path', 'frames', 'last_frame', 'original_image',
↪'label'])

dataset[0]["image"].shape
# Output: torch.Size([3, 256, 256])
```

`anomalib.data.video.avenue.`**`make_avenue_dataset`**`(`*root*, *gt_dir*, *split=None*`)`

Create CUHK Avenue dataset by parsing the file structure.

The files are expected to follow the structure:

- path/to/dataset/[training_videos|testing_videos]/video_filename.avi
- path/to/ground_truth/mask_filename.mat

Parameters

- **`root`** (*`Path`*) – Path to dataset
- **`gt_dir`** (*`Path`*) – Path to the ground truth

- **split** (Split | *str* | *None* = *None*, *optional*) – Dataset split (ie., either train or test). Defaults to None.

Example

The following example shows how to get testing samples from Avenue dataset:

```
>>> root = Path('./avenue')
>>> gt_dir = Path('./avenue/masks')
>>> samples = make_avenue_dataset(path, gt_dir, split='test')
>>> samples.head()
   root     folder         image_path                        mask_path                    ␣
↪   split
0  ./avenue testing_videos ./avenue/training_videos/01.avi ./avenue/masks/01_label.
↪mat test
1  ./avenue testing_videos ./avenue/training_videos/02.avi ./avenue/masks/01_label.
↪mat test
...
```

Returns
an output dataframe containing samples for the requested split (ie., train or test)

Return type
DataFrame

Shanghai Tech Data

ShanghaiTech Campus Dataset.

Description:
This module contains PyTorch Dataset and PyTorch Lightning DataModule for the ShanghaiTech Campus dataset. If the dataset is not on the file system, the DataModule class downloads and extracts the dataset and converts video files to a format that is readable by pyav.

License:
ShanghaiTech Campus Dataset is released under the BSD 2-Clause License.

Reference:

- W. Liu and W. Luo, D. Lian and S. Gao. "Future Frame Prediction for Anomaly Detection – A New Baseline." IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2018.

class anomalib.data.video.shanghaitech.**ShanghaiTech**(*root='./datasets/shanghaitech'*, *scene=1*, *clip_length_in_frames=2*, *frames_between_clips=1*, *target_frame=VideoTargetFrame.LAST*, *task=TaskType.SEGMENTATION*, *image_size=None*, *transform=None*, *train_transform=None*, *eval_transform=None*, *train_batch_size=32*, *eval_batch_size=32*, *num_workers=8*, *val_split_mode=ValSplitMode.SAME_AS_TEST*, *val_split_ratio=0.5*, *seed=None*)

Bases: *AnomalibVideoDataModule*

ShanghaiTech DataModule class.

Parameters

- **root** (*Path | str*) – Path to the root of the dataset
- **scene** (*int*) – Index of the dataset scene (category) in range [1, 13]
- **clip_length_in_frames** (*int, optional*) – Number of video frames in each clip.
- **frames_between_clips** (*int, optional*) – Number of frames between each consecutive video clip.
- **target_frame** (VideoTargetFrame) – Specifies the target frame in the video clip, used for ground truth retrieval
- **TaskType)** (*task*) – Task type, 'classification', 'detection' or 'segmentation'
- **image_size** (*tuple[int, int], optional*) – Size to which input images should be resized. Defaults to None.
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to None.
- **train_transform** (*Transform, optional*) – Transforms that should be applied to the input images during training. Defaults to None.
- **eval_transform** (*Transform, optional*) – Transforms that should be applied to the input images during evaluation. Defaults to None.
- **train_batch_size** (*int, optional*) – Training batch size. Defaults to 32.
- **eval_batch_size** (*int, optional*) – Test batch size. Defaults to 32.
- **num_workers** (*int, optional*) – Number of workers. Defaults to 8.
- **val_split_mode** (ValSplitMode) – Setting that determines how the validation subset is obtained.
- **val_split_ratio** (*float*) – Fraction of train or test images that will be reserved for validation.
- **seed** (*int | None, optional*) – Seed which may be set to a fixed value for reproducibility.

prepare_data()

Download the dataset and convert video files.

Return type

None

class anomalib.data.video.shanghaitech.**ShanghaiTechDataset**(*task*, *split*, *root='./datasets/shanghaitech'*, *scene=1*, *clip_length_in_frames=2*, *frames_between_clips=1*, *target_frame=VideoTargetFrame.LAST*, *transform=None*)

Bases: *AnomalibVideoDataset*

ShanghaiTech Dataset class.

Parameters

- **task** (*TaskType*) – Task type, 'classification', 'detection' or 'segmentation'

- **split** (Split) – Split of the dataset, usually Split.TRAIN or Split.TEST
- **root** (*Path | str*) – Path to the root of the dataset
- **scene** (*int*) – Index of the dataset scene (category) in range [1, 13]
- **clip_length_in_frames** (*int, optional*) – Number of video frames in each clip.
- **frames_between_clips** (*int, optional*) – Number of frames between each consecutive video clip.
- **target_frame** (VideoTargetFrame) – Specifies the target frame in the video clip, used for ground truth retrieval.
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to `None`.

class anomalib.data.video.shanghaitech.**ShanghaiTechTestClipsIndexer**(*video_paths*, *mask_paths*, *clip_length_in_frames=2*, *frames_between_clips=1*)

Bases: *ClipsIndexer*

Clips indexer for the test set of the ShanghaiTech Campus dataset.

The train and test subsets of the ShanghaiTech dataset use different file formats, so separate clips indexer implementations are needed.

get_clip(*idx*)

Get a subclip from a list of videos.

Parameters
idx (*int*) – index of the subclip. Must be between 0 and num_clips().

Return type
`tuple[Tensor, Tensor, dict[str, Any], int]`

Returns
video (torch.Tensor) audio (torch.Tensor) info (Dict) video_idx (int): index of the video in *video_paths*

get_mask(*idx*)

Retrieve the masks from the file system.

Return type
`Tensor | None`

class anomalib.data.video.shanghaitech.**ShanghaiTechTrainClipsIndexer**(*video_paths*, *mask_paths*, *clip_length_in_frames=2*, *frames_between_clips=1*)

Bases: *ClipsIndexer*

Clips indexer for ShanghaiTech dataset.

The train and test subsets of the ShanghaiTech dataset use different file formats, so separate clips indexer implementations are needed.

get_mask(*idx*)

No masks available for training set.

Return type
`Tensor | None`

anomalib.data.video.shanghaitech.**make_shanghaitech_dataset**(*root*, *scene*, *split=None*)

Create ShanghaiTech dataset by parsing the file structure.

The files are expected to follow the structure:
path/to/dataset/[training_videos|testing_videos]/video_filename.avi path/to/ground_truth/mask_filename.mat

Parameters

- **root** (*Path*) – Path to dataset
- **scene** (*int*) – Index of the dataset scene (category) in range [1, 13]
- **split** (Split *| str | None, optional*) – Dataset split (ie., either train or test). Defaults to None.

Example

The following example shows how to get testing samples from ShanghaiTech dataset:

```
>>> root = Path('./shanghaiTech')
>>> scene = 1
>>> samples = make_avenue_dataset(path, scene, split='test')
>>> samples.head()
    root            image_path                          split   mask_path
0       shanghaitech    shanghaitech/testing/frames/01_0014     test    ␣
↪shanghaitech/testing/test_pixel_mask/01_0014.npy
1       shanghaitech    shanghaitech/testing/frames/01_0015     test    ␣
↪shanghaitech/testing/test_pixel_mask/01_0015.npy
...
```

Returns
an output dataframe containing samples for the requested split (ie., train or test)

Return type
DataFrame

UCSD Data

UCSD Pedestrian dataset.

class anomalib.data.video.ucsd_ped.**UCSDped**(*root='./datasets/ucsd'*, *category='UCSDped2'*, *clip_length_in_frames=2*, *frames_between_clips=10*, *target_frame=VideoTargetFrame.LAST*, *task=TaskType.SEGMENTATION*, *image_size=None*, *transform=None*, *train_transform=None*, *eval_transform=None*, *train_batch_size=8*, *eval_batch_size=8*, *num_workers=8*, *val_split_mode=ValSplitMode.SAME_AS_TEST*, *val_split_ratio=0.5*, *seed=None*)

Bases: *AnomalibVideoDataModule*

UCSDped DataModule class.

Parameters

- **root** (*Path | str*) – Path to the root of the dataset

- **category** (*str*) – Sub-category of the dataset, e.g. "UCSDped1" or "UCSDped2"
- **clip_length_in_frames** (*int, optional*) – Number of video frames in each clip.
- **frames_between_clips** (*int, optional*) – Number of frames between each consecutive video clip.
- **target_frame** (VideoTargetFrame) – Specifies the target frame in the video clip, used for ground truth retrieval
- **task** (*TaskType*) – Task type, 'classification', 'detection' or 'segmentation'
- **image_size** (*tuple[int, int], optional*) – Size to which input images should be resized. Defaults to `None`.
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to `None`.
- **train_transform** (*Transform, optional*) – Transforms that should be applied to the input images during training. Defaults to `None`.
- **eval_transform** (*Transform, optional*) – Transforms that should be applied to the input images during evaluation. Defaults to `None`.
- **train_batch_size** (*int, optional*) – Training batch size. Defaults to 32.
- **eval_batch_size** (*int, optional*) – Test batch size. Defaults to 32.
- **num_workers** (*int, optional*) – Number of workers. Defaults to 8.
- **val_split_mode** (ValSplitMode) – Setting that determines how the validation subset is obtained.
- **val_split_ratio** (*float*) – Fraction of train or test images that will be reserved for validation.
- **seed** (*int | None, optional*) – Seed which may be set to a fixed value for reproducibility.

prepare_data()

Download the dataset if not available.

Return type

`None`

class anomalib.data.video.ucsd_ped.**UCSDpedClipsIndexer**(*video_paths*, *mask_paths*, *clip_length_in_frames=2*, *frames_between_clips=1*)

Bases: *ClipsIndexer*

Clips class for UCSDped dataset.

get_clip(*idx*)

Get a subclip from a list of videos.

Parameters

idx (*int*) – index of the subclip. Must be between 0 and num_clips().

Return type

`tuple[Tensor, Tensor, dict[str, Any], int]`

Returns

video (torch.Tensor) audio (torch.Tensor) info (dict) video_idx (int): index of the video in *video_paths*

get_mask(*idx*)

Retrieve the masks from the file system.

Return type
ndarray | None

class anomalib.data.video.ucsd_ped.**UCSDpedDataset**(*task*, *root*, *category*, *split*, *clip_length_in_frames=2*, *frames_between_clips=10*, *target_frame=VideoTargetFrame.LAST*, *transform=None*)

Bases: *AnomalibVideoDataset*

UCSDped Dataset class.

Parameters

- **task** (*TaskType*) – Task type, 'classification', 'detection' or 'segmentation'
- **root** (*Path | str*) – Path to the root of the dataset
- **category** (*str*) – Sub-category of the dataset, e.g. "UCSDped1" or "UCSDped2"
- **split** (*str | Split | None*) – Split of the dataset, usually Split.TRAIN or Split.TEST
- **clip_length_in_frames** (*int, optional*) – Number of video frames in each clip.
- **frames_between_clips** (*int, optional*) – Number of frames between each consecutive video clip.
- **target_frame** (VideoTargetFrame) – Specifies the target frame in the video clip, used for ground truth retrieval.
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to None.

anomalib.data.video.ucsd_ped.**make_ucsd_dataset**(*path*, *split=None*)

Create UCSD Pedestrian dataset by parsing the file structure.

The files are expected to follow the structure:
path/to/dataset/category/split/video_id/image_filename.tif path/to/dataset/category/split/video_id_gt/mask_filename.bmp

Parameters

- **path** (*Path*) – Path to dataset
- **split** (*str | Split | None, optional*) – Dataset split (ie., either train or test). Defaults to None.

Example

The following example shows how to get testing samples from UCSDped2 category:

```
>>> root = Path('./UCSDped')
>>> category = 'UCSDped2'
>>> path = root / category
>>> path
PosixPath('UCSDped/UCSDped2')
```

```
>>> samples = make_ucsd_dataset(path, split='test')
>>> samples.head()
   root             folder image_path                      mask_path                   ␣
↪       split
0  UCSDped/UCSDped2 Test   UCSDped/UCSDped2/Test/Test001 UCSDped/UCSDped2/Test/
↪Test001_gt  test
1  UCSDped/UCSDped2 Test   UCSDped/UCSDped2/Test/Test002 UCSDped/UCSDped2/Test/
↪Test002_gt  test
...
```

Returns
an output dataframe containing samples for the requested split (ie., train or test)

Return type
DataFrame

Depth Data

Folder 3D   Learn more about custom folder 3D dataset.

MVTec 3D   Learn more about MVTec 3D dataset

Folder 3D Data

Custom Folder Dataset.

This script creates a custom dataset from a folder.

class `anomalib.data.depth.folder_3d.`**`Folder3D`**(*name*, *normal_dir*, *root*, *abnormal_dir=None*, *normal_test_dir=None*, *mask_dir=None*, *normal_depth_dir=None*, *abnormal_depth_dir=None*, *normal_test_depth_dir=None*, *extensions=None*, *train_batch_size=32*, *eval_batch_size=32*, *num_workers=8*, *task=TaskType.SEGMENTATION*, *image_size=None*, *transform=None*, *train_transform=None*, *eval_transform=None*, *test_split_mode=TestSplitMode.FROM_DIR*, *test_split_ratio=0.2*, *val_split_mode=ValSplitMode.FROM_TEST*, *val_split_ratio=0.5*, *seed=None*)

Bases: `AnomalibDataModule`

Folder DataModule.

Parameters

- **`name`** (*`str`*) – Name of the dataset. This is used to name the datamodule, especially when logging/saving.
- **`normal_dir`** (*`str | Path`*) – Name of the directory containing normal images.
- **`root`** (*`str | Path | None`*) – Path to the root folder containing normal and abnormal dirs. Defaults to `None`.

- **abnormal_dir** (*str | Path | None*) – Name of the directory containing abnormal images. Defaults to `abnormal`.
- **normal_test_dir** (*str | Path | None, optional*) – Path to the directory containing normal images for the test dataset. Defaults to `None`.
- **mask_dir** (*str | Path | None, optional*) – Path to the directory containing the mask annotations. Defaults to `None`.
- **normal_depth_dir** (*str | Path | None, optional*) – Path to the directory containing normal depth images for the test dataset. Normal test depth images will be a split of *normal_dir*
- **abnormal_depth_dir** (*str | Path | None, optional*) – Path to the directory containing abnormal depth images for the test dataset.
- **normal_test_depth_dir** (*str | Path | None, optional*) – Path to the directory containing normal depth images for the test dataset. Normal test images will be a split of *normal_dir* if *None*. Defaults to None.
- **normal_split_ratio** (*float, optional*) – Ratio to split normal training images and add to the test set in case test set doesn't contain any normal images. Defaults to 0.2.
- **extensions** (*tuple[str, ...] | None, optional*) – Type of the image extensions to read from the directory. Defaults to None.
- **train_batch_size** (*int, optional*) – Training batch size. Defaults to `32`.
- **eval_batch_size** (*int, optional*) – Test batch size. Defaults to `32`.
- **num_workers** (*int, optional*) – Number of workers. Defaults to `8`.
- **task** (*TaskType, optional*) – Task type. Could be `classification`, `detection` or `segmentation`. Defaults to `TaskType.SEGMENTATION`.
- **image_size** (*tuple[int, int], optional*) – Size to which input images should be resized. Defaults to `None`.
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to `None`.
- **train_transform** (*Transform, optional*) – Transforms that should be applied to the input images during training. Defaults to `None`.
- **eval_transform** (*Transform, optional*) – Transforms that should be applied to the input images during evaluation. Defaults to `None`.
- **test_split_mode** (`TestSplitMode`) – Setting that determines how the testing subset is obtained. Defaults to `TestSplitMode.FROM_DIR`.
- **test_split_ratio** (*float*) – Fraction of images from the train set that will be reserved for testing. Defaults to `0.2`.
- **val_split_mode** (`ValSplitMode`) – Setting that determines how the validation subset is obtained. Defaults to `ValSplitMode.FROM_TEST`.
- **val_split_ratio** (*float*) – Fraction of train or test images that will be reserved for validation. Defaults to `0.5`.
- **seed** (*int | None, optional*) – Seed used during random subset splitting. Defaults to `None`.

property name: str

Name of the datamodule.

Folder3D datamodule overrides the name property to provide a custom name.

class anomalib.data.depth.folder_3d.**Folder3DDataset**(*name*, *task*, *normal_dir*, *root=None*, *abnormal_dir=None*, *normal_test_dir=None*, *mask_dir=None*, *normal_depth_dir=None*, *abnormal_depth_dir=None*, *normal_test_depth_dir=None*, *transform=None*, *split=None*, *extensions=None*)

Bases: *AnomalibDepthDataset*

Folder dataset.

Parameters

- **name** (*str*) – Name of the dataset.
- **task** (*TaskType*) – Task type. (`classification`, `detection` or `segmentation`).
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images.
- **normal_dir** (*str | Path*) – Path to the directory containing normal images.
- **root** (*str | Path | None*) – Root folder of the dataset. Defaults to `None`.
- **abnormal_dir** (*str | Path | None, optional*) – Path to the directory containing abnormal images. Defaults to `None`.
- **normal_test_dir** (*str | Path | None, optional*) – Path to the directory containing normal images for the test dataset. Defaults to `None`.
- **mask_dir** (*str | Path | None, optional*) – Path to the directory containing the mask annotations. Defaults to `None`.
- **normal_depth_dir** (*str | Path | None, optional*) – Path to the directory containing normal depth images for the test dataset. Normal test depth images will be a split of *normal_dir* Defaults to `None`.
- **abnormal_depth_dir** (*str | Path | None, optional*) – Path to the directory containing abnormal depth images for the test dataset. Defaults to `None`.
- **normal_test_depth_dir** (*str | Path | None, optional*) – Path to the directory containing normal depth images for the test dataset. Normal test images will be a split of *normal_dir* if *None*. Defaults to `None`.
- **transform** – Transforms that should be applied to the input images. Defaults to `None`.
- **split** (*str | Split | None*) – Fixed subset split that follows from folder structure on file system. Choose from [Split.FULL, Split.TRAIN, Split.TEST] Defaults to `None`.
- **extensions** (*tuple[str, ...] | None, optional*) – Type of the image extensions to read from the directory. Defaults to `None`.

Raises

ValueError – When task is set to classification and *mask_dir* is provided. When *mask_dir* is provided, *task* should be set to *segmentation*.

property name: str

Name of the dataset.

Folder3D dataset overrides the name property to provide a custom name.

`anomalib.data.depth.folder_3d.`**`make_folder3d_dataset`**(*normal_dir*, *root=None*, *abnormal_dir=None*, *normal_test_dir=None*, *mask_dir=None*, *normal_depth_dir=None*, *abnormal_depth_dir=None*, *normal_test_depth_dir=None*, *split=None*, *extensions=None*)

Make Folder Dataset.

Parameters

- **`normal_dir`** (*`str | Path`*) – Path to the directory containing normal images.
- **`root`** (*`str | Path | None`*) – Path to the root directory of the dataset. Defaults to `None`.
- **`abnormal_dir`** (*`str | Path | None, optional`*) – Path to the directory containing abnormal images. Defaults to `None`.
- **`normal_test_dir`** (*`str | Path | None, optional`*) – Path to the directory containing normal images for the test
- **`None.`** (*`dataset. Normal test images will be a split of normal_dir if`*) – Defaults to `None`.
- **`mask_dir`** (*`str | Path | None, optional`*) – Path to the directory containing the mask annotations. Defaults to `None`.
- **`normal_depth_dir`** (*`str | Path | None, optional`*) – Path to the directory containing normal depth images for the test dataset. Normal test depth images will be a split of *normal_dir* Defaults to `None`.
- **`abnormal_depth_dir`** (*`str | Path | None, optional`*) – Path to the directory containing abnormal depth images for the test dataset. Defaults to `None`.
- **`normal_test_depth_dir`** (*`str | Path | None, optional`*) – Path to the directory containing normal depth images for the test dataset. Normal test images will be a split of *normal_dir* if *None*. Defaults to `None`.
- **`split`** (*`str | Split | None, optional`*) – Dataset split (ie., Split.FULL, Split.TRAIN or Split.TEST). Defaults to `None`.
- **`extensions`** (*`tuple[str, ...] | None, optional`*) – Type of the image extensions to read from the directory. Defaults to `None`.

Returns

an output dataframe containing samples for the requested split (ie., train or test)

Return type

DataFrame

MVTec 3D Data

MVTec 3D-AD Dataset (CC BY-NC-SA 4.0).

Description:

This script contains PyTorch Dataset, Dataloader and PyTorch Lightning DataModule for the MVTec 3D-AD dataset. If the dataset is not on the file system, the script downloads and extracts the dataset and create PyTorch data objects.

License:

MVTec 3D-AD dataset is released under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International
License (CC BY-NC-SA 4.0)(https://creativecommons.org/licenses/by-nc-sa/4.0/).

Reference:

- **Paul Bergmann, Xin Jin, David Sattlegger, Carsten Steger: The MVTec 3D-AD Dataset for Unsupervised 3D Anomaly**
  Detection and Localization in: Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 5: VISAPP, 202-213, 2022, DOI: 10.5220/ 0010865000003124.

`class anomalib.data.depth.mvtec_3d.`**`MVTec3D`**(*root='./datasets/MVTec3D'*, *category='bagel'*, *train_batch_size=32*, *eval_batch_size=32*, *num_workers=8*, *task=TaskType.SEGMENTATION*, *image_size=None*, *transform=None*, *train_transform=None*, *eval_transform=None*, *test_split_mode=TestSplitMode.FROM_DIR*, *test_split_ratio=0.2*, *val_split_mode=ValSplitMode.SAME_AS_TEST*, *val_split_ratio=0.5*, *seed=None*)

Bases: *AnomalibDataModule*

MVTec Datamodule.

Parameters

- **root** (*Path | str*) – Path to the root of the dataset Defaults to `"./datasets/MVTec3D"`.
- **category** (*str*) – Category of the MVTec dataset (e.g. "bottle" or "cable"). Defaults to `bagel`.
- **train_batch_size** (*int, optional*) – Training batch size. Defaults to `32`.
- **eval_batch_size** (*int, optional*) – Test batch size. Defaults to `32`.
- **num_workers** (*int, optional*) – Number of workers. Defaults to `8`.
- **task** (*TaskType*) – Task type, 'classification', 'detection' or 'segmentation' Defaults to `TaskType.SEGMENTATION`.
- **image_size** (*tuple[int, int], optional*) – Size to which input images should be resized. Defaults to `None`.
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to `None`.
- **train_transform** (*Transform, optional*) – Transforms that should be applied to the input images during training. Defaults to `None`.
- **eval_transform** (*Transform, optional*) – Transforms that should be applied to the input images during evaluation. Defaults to `None`.
- **test_split_mode** (`TestSplitMode`) – Setting that determines how the testing subset is obtained. Defaults to `TestSplitMode.FROM_DIR`.
- **test_split_ratio** (*float*) – Fraction of images from the train set that will be reserved for testing. Defaults to `0.2`.
- **val_split_mode** (`ValSplitMode`) – Setting that determines how the validation subset is obtained. Defaults to `ValSplitMode.SAME_AS_TEST`.
- **val_split_ratio** (*float*) – Fraction of train or test images that will be reserved for validation. Defaults to `0.5`.

- **seed** (*int | None, optional*) – Seed which may be set to a fixed value for reproducibility. Defaults to None.

prepare_data()

Download the dataset if not available.

Return type
None

class anomalib.data.depth.mvtec_3d.**MVTec3DDataset**(*task*, *root='./datasets/MVTec3D'*, *category='bagel'*, *transform=None*, *split=None*)

Bases: *AnomalibDepthDataset*

MVTec 3D dataset class.

Parameters

- **task** (*TaskType*) – Task type, `classification`, `detection` or `segmentation`
- **root** (*Path | str*) – Path to the root of the dataset Defaults to `"./datasets/MVTec3D"`.
- **category** (*str*) – Sub-category of the dataset, e.g. ‘bagel’ Defaults to `"bagel"`.
- **transform** (*Transform, optional*) – Transforms that should be applied to the input images. Defaults to `None`.
- **split** (*str | Split | None*) – Split of the dataset, usually Split.TRAIN or Split.TEST Defaults to `None`.

anomalib.data.depth.mvtec_3d.**make_mvtec_3d_dataset**(*root*, *split=None*, *extensions=None*)

Create MVTec 3D-AD samples by parsing the MVTec AD data file structure.

The files are expected to follow this structure: - *path/to/dataset/split/category/image_filename.png* - *path/to/dataset/ground_truth/category/mask_filename.png*

This function creates a DataFrame to store the parsed information. The DataFrame follows this format:

	path	split	label	image_path	mask_path	label_index
0	datasets/name	test	defect	filename.png	ground_truth/defect/filename_mask.png	1

Parameters

- **root** (*Path*) – Path to the dataset.
- **split** (*str | Split | None, optional*) – Dataset split (e.g., ‘train’ or ‘test’). Defaults to `None`.
- **extensions** (*Sequence[str] | None, optional*) – List of file extensions to be included in the dataset. Defaults to `None`.

Examples

The following example shows how to get training samples from the MVTec 3D-AD 'bagel' category:

```
>>> from pathlib import Path
>>> root = Path('./MVTec3D')
>>> category = 'bagel'
>>> path = root / category
>>> print(path)
PosixPath('MVTec3D/bagel')
```

```
>>> samples = create_mvtec_3d_ad_samples(path, split='train')
>>> print(samples.head())
    path          split label image_path                          mask_path        ␣
↪              label_index
    MVTec3D/bagel train good MVTec3D/bagel/train/good/rgb/105.png MVTec3D/bagel/
↪ground_truth/good/gt/105.png 0
    MVTec3D/bagel train good MVTec3D/bagel/train/good/rgb/017.png MVTec3D/bagel/
↪ground_truth/good/gt/017.png 0
```

Returns
An output DataFrame containing the samples of the dataset.

Return type
DataFrame

Data Utils

Image & Video Utils   Learn more about anomalib API and CLI.

Data Transforms   Learn how to use anomalib for your anomaly detection tasks.

Tiling   Learn more about the internals of anomalib.

Synthetic Data   Learn more about the internals of anomalib.

Image and Video Utils

Path Utils

Path Utils.

class `anomalib.data.utils.path.`**`DirType`**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `str`, `Enum`

Dir type names.

`anomalib.data.utils.path.`**`contains_non_printable_characters`**(*path*)

Check if the path contains non-printable characters.

Parameters
path (*str | Path*) – Path to check.

Returns
True if the path contains non-printable characters, False otherwise.

Return type
bool

Examples

```
>>> contains_non_printable_characters("./datasets/MVTec/bottle/train/good/000.png")
False
```

```
>>> contains_non_printable_characters("./datasets/MVTec/bottle/train/good/000.png\0
↪")
True
```

anomalib.data.utils.path.**is_path_too_long**(*path*, *max_length=512*)

Check if the path contains too long input.

Parameters

- **path** (*str | Path*) – Path to check.
- **max_length** (*int*) – Maximum length a path can be before it is considered too long. Defaults to `512`.

Returns
True if the path contains too long input, False otherwise.

Return type
bool

Examples

```
>>> contains_too_long_input("./datasets/MVTec/bottle/train/good/000.png")
False
```

```
>>> contains_too_long_input("./datasets/MVTec/bottle/train/good/000.png" + "a" *␣
↪4096)
True
```

anomalib.data.utils.path.**resolve_path**(*folder*, *root=None*)

Combine root and folder and returns the absolute path.

This allows users to pass either a root directory and relative paths, or absolute paths to each of the image sources. This function makes sure that the samples dataframe always contains absolute paths.

Parameters

- **folder** (*str | Path | None*) – Folder location containing image or mask data.
- **root** (*str | Path | None*) – Root directory for the dataset.

Return type
`Path`

`anomalib.data.utils.path.`**`validate_and_resolve_path`**(*folder*, *root=None*, *base_dir=None*)

Validate and resolve the path.

Parameters

- **`folder`** (*`str | Path`*) – Folder location containing image or mask data.
- **`root`** (*`str | Path | None`*) – Root directory for the dataset.
- **`base_dir`** (*`str | Path | None`*) – Base directory to restrict file access.

Returns

Validated and resolved path.

Return type

Path

`anomalib.data.utils.path.`**`validate_path`**(*path*, *base_dir=None*, *should_exist=True*)

Validate the path.

Parameters

- **`path`** (*`str | Path`*) – Path to validate.
- **`base_dir`** (*`str | Path`*) – Base directory to restrict file access.
- **`should_exist`** (*`bool`*) – If True, do not raise an exception if the path does not exist.

Returns

Validated path.

Return type

Path

Examples

```
>>> validate_path("./datasets/MVTec/bottle/train/good/000.png")
PosixPath('/abs/path/to/anomalib/datasets/MVTec/bottle/train/good/000.png')
```

```
>>> validate_path("./datasets/MVTec/bottle/train/good/000.png", base_dir="./
↪datasets/MVTec")
PosixPath('/abs/path/to/anomalib/datasets/MVTec/bottle/train/good/000.png')
```

```
>>> validate_path("/path/to/unexisting/file")
Traceback (most recent call last):
File "<string>", line 1, in <module>
File "<string>", line 18, in validate_path
FileNotFoundError: Path does not exist: /path/to/unexisting/file
```

Accessing a file without read permission should raise PermissionError:

Note: Note that, we are using `/usr/local/bin` directory as an example here. If this directory does not exist on your system, this will raise `FileNotFoundError` instead of `PermissionError`. You could change the directory to any directory that you do not have read permission.

```
>>> validate_path("/bin/bash", base_dir="/bin/")
Traceback (most recent call last):
File "<string>", line 1, in <module>
File "<string>", line 18, in validate_path
PermissionError: Read permission denied for the file: /usr/local/bin
```

Download Utils

Helper to show progress bars with *urlretrieve*, check hash of file.

class `anomalib.data.utils.download.`**`DownloadInfo`**(*name*, *url*, *hashsum*, *filename=None*)

Bases: `object`

Info needed to download a dataset from a url.

class `anomalib.data.utils.download.`**`DownloadProgressBar`**(*iterable=None*, *desc=None*, *total=None*, *leave=True*, *file=None*, *ncols=None*, *mininterval=0.1*, *maxinterval=10.0*, *miniters=None*, *use_ascii=None*, *disable=False*, *unit='it'*, *unit_scale=False*, *dynamic_ncols=False*, *smoothing=0.3*, *bar_format=None*, *initial=0*, *position=None*, *postfix=None*, *unit_divisor=1000*, *write_bytes=None*, *lock_args=None*, *nrows=None*, *colour=None*, *delay=0*, *gui=False*, ***kwargs*)

Bases: `tqdm`

Create progress bar for urlretrieve. Subclasses *tqdm*.

For information about the parameters in constructor, refer to *tqdm*'s documentation.

Parameters

- **iterable** (*Iterable | None*) – Iterable to decorate with a progressbar. Leave blank to manually manage the updates.
- **desc** (*str | None*) – Prefix for the progressbar.
- **total** (*int | float | None*) – The number of expected iterations. If unspecified, len(iterable) is used if possible. If float("inf") or as a last resort, only basic progress statistics are displayed (no ETA, no progressbar). If *gui* is True and this parameter needs subsequent updating, specify an initial arbitrary large positive number, e.g. 9e9.
- **leave** (*bool | None*) – upon termination of iteration. If *None*, will leave only if *position* is *0*.
- **file** (*io.TextIOWrapper | io.StringIO | None*) – Specifies where to output the progress messages (default: sys.stderr). Uses *file.write(str)* and *file.flush()* methods. For encoding, see *write_bytes*.
- **ncols** (*int | None*) – The width of the entire output message. If specified, dynamically resizes the progressbar to stay within this bound. If unspecified, attempts to use environment width. The fallback is a meter width of 10 and no limit for the counter and statistics. If 0, will not print any meter (only stats).
- **mininterval** (*float | None*) – Minimum progress display update interval [default: 0.1] seconds.

- **maxinterval** (*float | None*) – Maximum progress display update interval [default: 10] seconds. Automatically adjusts *miniters* to correspond to *mininterval* after long display update lag. Only works if *dynamic_miniters* or monitor thread is enabled.
- **miniters** (*int | float | None*) – Minimum progress display update interval, in iterations. If 0 and *dynamic_miniters*, will automatically adjust to equal *mininterval* (more CPU efficient, good for tight loops). If > 0, will skip display of specified number of iterations. Tweak this and *mininterval* to get very efficient loops. If your progress is erratic with both fast and slow iterations (network, skipping items, etc) you should set miniters=1.
- **use_ascii** (*str | bool | None*) – If unspecified or False, use unicode (smooth blocks) to fill the meter. The fallback is to use ASCII characters " 123456789#".
- **disable** (*bool | None*) – Whether to disable the entire progressbar wrapper [default: False]. If set to None, disable on non-TTY.
- **unit** (*str | None*) – String that will be used to define the unit of each iteration [default: it].
- **unit_scale** (*int | float | bool*) – If 1 or True, the number of iterations will be reduced/scaled automatically and a metric prefix following the International System of Units standard will be added (kilo, mega, etc.) [default: False]. If any other non-zero number, will scale *total* and *n*.
- **dynamic_ncols** (*bool | None*) – If set, constantly alters *ncols* and *nrows* to the environment (allowing for window resizes) [default: False].
- **smoothing** (*float | None*) – Exponential moving average smoothing factor for speed estimates (ignored in GUI mode). Ranges from 0 (average speed) to 1 (current/instantaneous speed) [default: 0.3].
- **bar_format** (*str | None*) – Specify a custom bar string formatting. May impact performance. [default: '{l_bar}{bar}{r_bar}'], where l_bar='{desc}: {percentage:3.0f}%|' and r_bar='| {n_fmt}/{total_fmt} [{elapsed}<{remaining}, ' '{rate_fmt}{postfix}]' Possible vars: l_bar, bar, r_bar, n, n_fmt, total, total_fmt, percentage, elapsed, elapsed_s, ncols, nrows, desc, unit, rate, rate_fmt, rate_noinv, rate_noinv_fmt, rate_inv, rate_inv_fmt, postfix, unit_divisor, remaining, remaining_s, eta. Note that a trailing ": " is automatically removed after {desc} if the latter is empty.
- **initial** (*int | float | None*) – The initial counter value. Useful when restarting a progress bar [default: 0]. If using float, consider specifying *{n:.3f}* or similar in *bar_format*, or specifying *unit_scale*.
- **position** (*int | None*) – Specify the line offset to print this bar (starting from 0) Automatic if unspecified. Useful to manage multiple bars at once (eg, from threads).
- **postfix** (*dict | None*) – Specify additional stats to display at the end of the bar. Calls *set_postfix(**postfix)* if possible (dict).
- **unit_divisor** (*float | None*) – [default: 1000], ignored unless *unit_scale* is True.
- **write_bytes** (*bool | None*) – If (default: None) and *file* is unspecified, bytes will be written in Python 2. If *True* will also write bytes. In all other cases will default to unicode.
- **lock_args** (*tuple | None*) – Passed to *refresh* for intermediate output (initialisation, iterating, and updating). nrows (int | None): The screen height. If specified, hides nested bars outside this bound. If unspecified, attempts to use environment height. The fallback is 20.
- **colour** (*str | None*) – Bar colour (e.g. 'green', '#00ff00').
- **delay** (*float | None*) – Don't display until [default: 0] seconds have elapsed.

- **gui** (*bool | None*) – WARNING: internal parameter - do not use. Use tqdm.gui.tqdm(…) instead. If set, will attempt to use matplotlib animations for a graphical output [default: False].

Example

```
>>> with DownloadProgressBar(unit='B', unit_scale=True, miniters=1, desc=url.split(
↪'/')[-1]) as p_bar:
>>>         urllib.request.urlretrieve(url, filename=output_path, reporthook=p_bar.
↪update_to)
```

update_to(*chunk_number=1*, *max_chunk_size=1*, *total_size=None*)

Progress bar hook for tqdm.

Based on https://stackoverflow.com/a/53877507 The implementor does not have to bother about passing parameters to this as it gets them from urlretrieve. However the context needs a few parameters. Refer to the example.

Parameters

- **chunk_number** (*int, optional*) – The current chunk being processed. Defaults to 1.
- **max_chunk_size** (*int, optional*) – Maximum size of each chunk. Defaults to 1.
- **total_size** (*int, optional*) – Total download size. Defaults to None.

Return type

None

anomalib.data.utils.download.**check_hash**(*file_path*, *expected_hash*, *algorithm='sha256'*)

Raise value error if hash does not match the calculated hash of the file.

Parameters

- **file_path** (*Path*) – Path to file.
- **expected_hash** (*str*) – Expected hash of the file.
- **algorithm** (*str*) – Hashing algorithm to use ('sha256', 'sha3_512', etc.).

Return type

None

anomalib.data.utils.download.**download_and_extract**(*root*, *info*)

Download and extract a dataset.

Parameters

- **root** (*Path*) – Root directory where the dataset will be stored.
- **info** (*DownloadInfo*) – Info needed to download the dataset.

Return type

None

anomalib.data.utils.download.**extract**(*file_name*, *root*)

Extract a dataset.

Parameters

- **file_name** (*Path*) – Path of the file to be extracted.
- **root** (*Path*) – Root directory where the dataset will be stored.

Return type
`None`

`anomalib.data.utils.download.`**`generate_hash`**`(`*file_path*, *algorithm='sha256'*`)`

Generate a hash of a file using the specified algorithm.

Parameters

- **`file_path`** (*`str | Path`*) – Path to the file to hash.
- **`algorithm`** (*`str`*) – The hashing algorithm to use (e.g., 'sha256', 'sha3_512').

Returns
The hexadecimal hash string of the file.

Return type
str

Raises
`ValueError` – If the specified hashing algorithm is not supported.

`anomalib.data.utils.download.`**`is_file_potentially_dangerous`**`(`*file_name*`)`

Check if a file is potentially dangerous.

Parameters
`file_name` (*`str`*) – Filename.

Returns
True if the member is potentially dangerous, False otherwise.

Return type
bool

`anomalib.data.utils.download.`**`is_within_directory`**`(`*directory*, *target*`)`

Check if a target path is located within a given directory.

Parameters

- **`directory`** (*`Path`*) – path of the parent directory
- **`target`** (*`Path`*) – path of the target

Returns
True if the target is within the directory, False otherwise

Return type
(bool)

`anomalib.data.utils.download.`**`safe_extract`**`(`*tar_file*, *root*, *members*`)`

Extract safe members from a tar archive.

Parameters

- **`tar_file`** (*`TarFile`*) – TarFile object.
- **`root`** (*`Path`*) – Root directory where the dataset will be stored.
- **`members`** (*`List[TarInfo]`*) – List of safe members to be extracted.

Return type
`None`

Image Utils

Image Utils.

anomalib.data.utils.image.**duplicate_filename**(*path*)

Check and duplicate filename.

This function checks the path and adds a suffix if it already exists on the file system.

Parameters
path (*str | Path*) – Input Path

Examples

```
>>> path = Path("datasets/MVTec/bottle/test/broken_large/000.png")
>>> path.exists()
True
```

If we pass this to `duplicate_filename` function we would get the following: >>> duplicate_filename(path) PosixPath('datasets/MVTec/bottle/test/broken_large/000_1.png')

Returns
Duplicated output path.

Return type
Path

anomalib.data.utils.image.**figure_to_array**(*fig*)

Convert a matplotlib figure to a numpy array.

Parameters
fig (*Figure*) – Matplotlib figure.

Returns
Numpy array containing the image.

Return type
np.ndarray

anomalib.data.utils.image.**generate_output_image_filename**(*input_path*, *output_path*)

Generate an output filename to save the inference image.

This function generates an output filaname by checking the input and output filenames. Input path is the input to infer, and output path is the path to save the output predictions specified by the user.

The function expects `input_path` to always be a file, not a directory. `output_path` could be a filename or directory. If it is a filename, the function checks if the specified filename exists on the file system. If yes, the function calls `duplicate_filename` to duplicate the filename to avoid overwriting the existing file. If `output_path` is a directory, this function adds the parent and filenames of `input_path` to `output_path`.

Parameters

- **input_path** (*str | Path*) – Path to the input image to infer.
- **output_path** (*str | Path*) – Path to output to save the predictions. Could be a filename or a directory.

Examples

```
>>> input_path = Path("datasets/MVTec/bottle/test/broken_large/000.png")
>>> output_path = Path("datasets/MVTec/bottle/test/broken_large/000.png")
>>> generate_output_image_filename(input_path, output_path)
PosixPath('datasets/MVTec/bottle/test/broken_large/000_1.png')
```

```
>>> input_path = Path("datasets/MVTec/bottle/test/broken_large/000.png")
>>> output_path = Path("results/images")
>>> generate_output_image_filename(input_path, output_path)
PosixPath('results/images/broken_large/000.png')
```

Raises
`ValueError` – When the `input_path` is not a file.

Returns
The output filename to save the output predictions from the inferencer.

Return type
Path

`anomalib.data.utils.image.`**`get_image_filename`**`(`*filename*`)`

Get image filename.

Parameters
`filename` (*`str | Path`*) – Filename to check.

Returns
Image filename.

Return type
Path

Examples

Assume that we have the following files in the directory:

```
$ ls
000.png  001.jpg  002.JPEG  003.tiff  004.png  005.txt
```

```
>>> get_image_filename("000.png")
PosixPath('000.png')
```

```
>>> get_image_filename("001.jpg")
PosixPath('001.jpg')
```

```
>>> get_image_filename("009.tiff")
Traceback (most recent call last):
File "<string>", line 1, in <module>
File "<string>", line 18, in get_image_filename
FileNotFoundError: File not found: 009.tiff
```

```
>>> get_image_filename("005.txt")
Traceback (most recent call last):
File "<string>", line 1, in <module>
File "<string>", line 18, in get_image_filename
ValueError: ``filename`` is not an image file. 005.txt
```

anomalib.data.utils.image.**get_image_filenames**(*path*, *base_dir=None*)

Get image filenames.

Parameters

- **path** (*str | Path*) – Path to image or image-folder.
- **base_dir** (*Path*) – Base directory to restrict file access.

Returns

List of image filenames.

Return type

list[Path]

Examples

Assume that we have the following files in the directory:

```
$ tree images
images
├── bad
│   ├── 003.png
│   └── 004.jpg
└── good
    ├── 000.png
    └── 001.tiff
```

We can get the image filenames with various ways:

```
>>> get_image_filenames("images/bad/003.png")
PosixPath('/home/sakcay/Projects/anomalib/images/bad/003.png')]
```

It is possible to recursively get the image filenames from a directory:

```
>>> get_image_filenames("images")
[PosixPath('/home/sakcay/Projects/anomalib/images/bad/003.png'),
PosixPath('/home/sakcay/Projects/anomalib/images/bad/004.jpg'),
PosixPath('/home/sakcay/Projects/anomalib/images/good/001.tiff'),
PosixPath('/home/sakcay/Projects/anomalib/images/good/000.png')]
```

If we want to restrict the file access to a specific directory, we can use `base_dir` argument.

```
>>> get_image_filenames("images", base_dir="images/bad")
Traceback (most recent call last):
File "<string>", line 1, in <module>
File "<string>", line 18, in get_image_filenames
ValueError: Access denied: Path is outside the allowed directory.
```

anomalib.data.utils.image.**get_image_filenames_from_dir**(*path*)

Get image filenames from directory.

Parameters
: **path** (*str | Path*) – Path to image directory.

Raises
: **ValueError** – When path is not a directory.

Returns
: Image filenames.

Return type
: list[Path]

Examples

Assume that we have the following files in the directory: $ ls 000.png 001.jpg 002.JPEG 003.tiff 004.png 005.png

```
>>> get_image_filenames_from_dir(".")
[PosixPath('000.png'), PosixPath('001.jpg'), PosixPath('002.JPEG'),
PosixPath('003.tiff'), PosixPath('004.png'), PosixPath('005.png')]
```

```
>>> get_image_filenames_from_dir("009.tiff")
Traceback (most recent call last):
File "<string>", line 1, in <module>
File "<string>", line 18, in get_image_filenames_from_dir
ValueError: ``path`` is not a directory: 009.tiff
```

anomalib.data.utils.image.**get_image_height_and_width**(*image_size*)

Get image height and width from image_size variable.

Parameters
: **image_size** (*int | Sequence[int] | None, optional*) – Input image size.

Raises
: **ValueError** – Image size not None, int or Sequence of values.

Examples

```
>>> get_image_height_and_width(image_size=256)
(256, 256)
```

```
>>> get_image_height_and_width(image_size=(256, 256))
(256, 256)
```

```
>>> get_image_height_and_width(image_size=(256, 256, 3))
(256, 256)
```

```
>>> get_image_height_and_width(image_size=256.)
Traceback (most recent call last):
File "<string>", line 1, in <module>
File "<string>", line 18, in get_image_height_and_width
ValueError: ``image_size`` could be either int or tuple[int, int]
```

Returns
A tuple containing image height and width values.

Return type
tuple[int | None, int | None]

anomalib.data.utils.image.**is_image_file**(*filename*)

Check if the filename is an image file.

Parameters
filename (*str | Path*) – Filename to check.

Returns
True if the filename is an image file.

Return type
bool

Examples

```
>>> is_image_file("000.png")
True
```

```
>>> is_image_file("002.JPEG")
True
```

```
>>> is_image_file("009.tiff")
True
```

```
>>> is_image_file("002.avi")
False
```

anomalib.data.utils.image.**pad_nextpow2**(*batch*)

Compute required padding from input size and return padded images.

Finds the largest dimension and computes a square image of dimensions that are of the power of 2. In case the image dimension is odd, it returns the image with an extra padding on one side.

Parameters
batch (*torch.Tensor*) – Input images

Returns
Padded batch

Return type
batch

anomalib.data.utils.image.**read_depth_image**(*path*)

Read tiff depth image from disk.

Parameters
path (*str, Path*) – path to the image file

Example

```
>>> image = read_depth_image("test_image.tiff")
```

Return type
`ndarray`

Returns
image as numpy array

`anomalib.data.utils.image.`**`read_image`**`(`*path*, *as_tensor=False*`)`

Read image from disk in RGB format.

Parameters

- **`path`** (*`str, Path`*) – path to the image file
- **`as_tensor`** (*`bool, optional`*) – If True, returns the image as a tensor. Defaults to False.

Example

```
>>> image = read_image("test_image.jpg")
>>> type(image)
<class 'numpy.ndarray'>
>>>
>>> image = read_image("test_image.jpg", as_tensor=True)
>>> type(image)
<class 'torch.Tensor'>
```

Return type
`Tensor` | `ndarray`

Returns
image as numpy array

`anomalib.data.utils.image.`**`read_mask`**`(`*path*, *as_tensor=False*`)`

Read mask from disk.

Parameters

- **`path`** (*`str, Path`*) – path to the mask file
- **`as_tensor`** (*`bool, optional`*) – If True, returns the mask as a tensor. Defaults to False.

Return type
`Tensor` | `ndarray`

Example

```
>>> mask = read_mask("test_mask.png")
>>> type(mask)
<class 'numpy.ndarray'>
>>>
>>> mask = read_mask("test_mask.png", as_tensor=True)
>>> type(mask)
<class 'torch.Tensor'>
```

`anomalib.data.utils.image.`**`save_image`**(*filename*, *image*, *root=None*)

Save an image to the file system.

Parameters

- **filename** (*Path | str*) – Path or filename to which the image will be saved.
- **image** (*np.ndarray | Figure*) – Image that will be saved to the file system.
- **root** (*Path, optional*) – Root directory to save the image. If provided, the top level directory of an absolute filename will be overwritten. Defaults to None.

Return type

`None`

`anomalib.data.utils.image.`**`show_image`**(*image*, *title='Image'*)

Show an image on the screen.

Parameters

- **image** (*np.ndarray | Figure*) – Image that will be shown in the window.
- **title** (*str, optional*) – Title that will be given to that window. Defaults to "Image".

Return type

`None`

Video Utils

Video utils.

class `anomalib.data.utils.video.`**`ClipsIndexer`**(*video_paths*, *mask_paths*, *clip_length_in_frames=2*, *frames_between_clips=1*)

Bases: `VideoClips`, `ABC`

Extension of torchvision's VideoClips class that also returns the masks for each clip.

Subclasses should implement the get_mask method. By default, the class inherits the functionality of VideoClips, which assumes that video_paths is a list of video files. If custom behaviour is required (e.g. video_paths is a list of folders with single-frame images), the subclass should implement at least get_clip and _compute_frame_pts.

Parameters

- **video_paths** (*list[str]*) – List of video paths that make up the dataset.
- **mask_paths** (*list[str]*) – List of paths to the masks for each video in the dataset.

`get_item`(*idx*)

Return a dictionary containing the clip, mask, video path and frame indices.

Return type
`dict[str, Any]`

`abstract get_mask`(*idx*)
Return the masks for the given index.

Return type
`Tensor | None`

`last_frame_idx`(*video_idx*)
Return the index of the last frame for a given video.

Return type
`int`

`anomalib.data.utils.video.convert_video`(*input_path*, *output_path*, *codec='MP4V'*)
Convert video file to a different codec.

Parameters

- **input_path** (*Path*) – Path to the input video.
- **output_path** (*Path*) – Path to the target output video.
- **codec** (*str*) – fourcc code of the codec that will be used for compression of the output file.

Return type
`None`

Label Utils

Label name enum class.

`class anomalib.data.utils.label.LabelName`(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `int`, `Enum`

Name of label.

Bounding Box Utils

Helper functions for processing bounding box detections and annotations.

`anomalib.data.utils.boxes.boxes_to_anomaly_maps`(*boxes*, *scores*, *image_size*)
Convert bounding box coordinates to anomaly heatmaps.

Parameters

- **boxes** (*list[torch.Tensor]*) – A list of length B where each element is a tensor of shape (N, 4) containing the bounding box coordinates of the regions of interest in xyxy format.
- **scores** (*list[torch.Tensor]*) – A list of length B where each element is a 1D tensor of length N containing the anomaly scores for each region of interest.
- **image_size** (*tuple[int, int]*) – Image size of the output masks in (H, W) format.

Returns

torch.Tensor of shape (B, H, W). The pixel locations within each bounding box are collectively
assigned the anomaly score of the bounding box. In the case of overlapping bounding boxes, the highest score is used.

Return type
Tensor

`anomalib.data.utils.boxes.`**`boxes_to_masks`**`(`*boxes*, *image_size*`)`

Convert bounding boxes to segmentations masks.

Parameters

- **boxes** (*list[torch.Tensor]*) – A list of length B where each element is a tensor of shape (N, 4) containing the bounding box coordinates of the regions of interest in xyxy format.
- **image_size** (*tuple[int, int]*) – Image size of the output masks in (H, W) format.

Returns

torch.Tensor of shape (B, H, W) in which each slice is a binary mask showing the pixels contained by a
bounding box.

Return type
Tensor

`anomalib.data.utils.boxes.`**`masks_to_boxes`**`(`*masks*, *anomaly_maps=None*`)`

Convert a batch of segmentation masks to bounding box coordinates.

Parameters

- **masks** (*torch.Tensor*) – Input tensor of shape (B, 1, H, W), (B, H, W) or (H, W)
- **anomaly_maps** (*Tensor | None, optional*) – Anomaly maps of shape (B, 1, H, W), (B, H, W) or (H, W) which are used to determine an anomaly score for the converted bounding boxes.

Returns

A list of length B where each element is a tensor of shape (N, 4)
containing the bounding box coordinates of the objects in the masks in xyxy format.

list[torch.Tensor]: A list of length B where each element is a tensor of length (N)
containing an anomaly score for each of the converted boxes.

Return type
list[torch.Tensor]

`anomalib.data.utils.boxes.`**`scale_boxes`**`(`*boxes*, *image_size*, *new_size*`)`

Scale bbox coordinates to a new image size.

Parameters

- **boxes** (*torch.Tensor*) – Boxes of shape (N, 4) - (x1, y1, x2, y2).
- **image_size** (*Size*) – Size of the original image in which the bbox coordinates were retrieved.
- **new_size** (*Size*) – New image size to which the bbox coordinates will be scaled.

Returns
Updated boxes of shape (N, 4) - (x1, y1, x2, y2).

Return type
Tensor

Dataset Split Utils

Dataset Split Utils.

This module contains function in regards to splitting normal images in training set, and creating validation sets from test sets.

These function are useful

- when the test set does not contain any normal images.
- when the dataset doesn't have a validation set.

`class anomalib.data.utils.split.`**`Split`**`(`*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*`)`

Bases: `str`, `Enum`

Split of a subset.

`class anomalib.data.utils.split.`**`TestSplitMode`**`(`*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*`)`

Bases: `str`, `Enum`

Splitting mode used to obtain subset.

`class anomalib.data.utils.split.`**`ValSplitMode`**`(`*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*`)`

Bases: `str`, `Enum`

Splitting mode used to obtain validation subset.

`anomalib.data.utils.split.`**`concatenate_datasets`**`(`*datasets*`)`

Concatenate multiple datasets into a single dataset object.

Parameters
`datasets` (*`Sequence[AnomalibDataset]`*) – Sequence of at least two datasets.

Returns
Dataset that contains the combined samples of all input datasets.

Return type
AnomalibDataset

`anomalib.data.utils.split.`**`random_split`**`(`*dataset*, *split_ratio*, *label_aware=False*, *seed=None*`)`

Perform a random split of a dataset.

Parameters

- **`dataset`** (`AnomalibDataset`) – Source dataset
- **`split_ratio`** (*`Union[float, Sequence[float]]`*) – Fractions of the splits that will be produced. The values in the sequence must sum to 1. If a single value is passed, the ratio will be converted to [1-split_ratio, split_ratio].
- **`label_aware`** (*`bool`*) – When True, the relative occurrence of the different class labels of the source dataset will be maintained in each of the subsets.
- **`seed`** (*`int | None, optional`*) – Seed that can be passed if results need to be reproducible

Return type
list[*AnomalibDataset*]

anomalib.data.utils.split.**split_by_label**(*dataset*)

Split the dataset into the normal and anomalous subsets.

Return type
tuple[*AnomalibDataset*, *AnomalibDataset*]

Data Transforms

Tiling

Image Tiler.

class anomalib.data.utils.tiler.**ImageUpscaleMode**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: str, Enum

Type of mode when upscaling image.

exception anomalib.data.utils.tiler.**StrideSizeError**

Bases: Exception

StrideSizeError to raise exception when stride size is greater than the tile size.

class anomalib.data.utils.tiler.**Tiler**(*tile_size*, *stride=None*, *remove_border_count=0*, *mode=ImageUpscaleMode.PADDING*)

Bases: object

Tile Image into (non)overlapping Patches. Images are tiled in order to efficiently process large images.

Parameters

- **tile_size** (int | Sequence) – Tile dimension for each patch
- **stride** (int | Sequence | None) – Stride length between patches
- **remove_border_count** (int) – Number of border pixels to be removed from tile before untiling
- **mode** (*ImageUpscaleMode*) – Upscaling mode for image resize.Supported formats: padding, interpolation

Examples

```
>>> import torch
>>> from torchvision import transforms
>>> from skimage.data import camera
>>> tiler = Tiler(tile_size=256,stride=128)
>>> image = transforms.ToTensor()(camera())
>>> tiles = tiler.tile(image)
>>> image.shape, tiles.shape
(torch.Size([3, 512, 512]), torch.Size([9, 3, 256, 256]))
```

```
>>> # Perform your operations on the tiles.
```

```
>>> # Untile the patches to reconstruct the image
>>> reconstructed_image = tiler.untile(tiles)
>>> reconstructed_image.shape
torch.Size([1, 3, 512, 512])
```

tile(*image*, *use_random_tiling=False*)

Tiles an input image to either overlapping, non-overlapping or random patches.

Parameters

- **image** (Tensor) – Input image to tile.
- **use_random_tiling** (bool) – If True, randomly crops tiles from the image. If False, tiles the image in a regular grid.

Examples

```
>>> from anomalib.data.utils.tiler import Tiler
>>> tiler = Tiler(tile_size=512,stride=256)
>>> image = torch.rand(size=(2, 3, 1024, 1024))
>>> image.shape
torch.Size([2, 3, 1024, 1024])
>>> tiles = tiler.tile(image)
>>> tiles.shape
torch.Size([18, 3, 512, 512])
```

Return type
Tensor

Returns
Tiles generated from the image.

untile(*tiles*)

Untiles patches to reconstruct the original input image.

If patches, are overlapping patches, the function averages the overlapping pixels, and return the reconstructed image.

Parameters
tiles (Tensor) – Tiles from the input image, generated via tile()..

Examples

```
>>> from anomalib.data.utils.tiler import Tiler
>>> tiler = Tiler(tile_size=512,stride=256)
>>> image = torch.rand(size=(2, 3, 1024, 1024))
>>> image.shape
torch.Size([2, 3, 1024, 1024])
>>> tiles = tiler.tile(image)
>>> tiles.shape
```

(continues on next page)

(continued from previous page)

```
torch.Size([18, 3, 512, 512])
>>> reconstructed_image = tiler.untile(tiles)
>>> reconstructed_image.shape
torch.Size([2, 3, 1024, 1024])
>>> torch.equal(image, reconstructed_image)
True
```

Return type
`Tensor`

Returns
Output that is the reconstructed version of the input tensor.

`anomalib.data.utils.tiler.`**`compute_new_image_size`**(*image_size*, *tile_size*, *stride*)

Check if image size is divisible by tile size and stride.

If not divisible, it resizes the image size to make it divisible.

Parameters

- **`image_size`** (*`tuple`*) – Original image size
- **`tile_size`** (*`tuple`*) – Tile size
- **`stride`** (*`tuple`*) – Stride

Examples

```
>>> compute_new_image_size(image_size=(512, 512), tile_size=(256, 256), stride=(128,
↪ 128))
(512, 512)
```

```
>>> compute_new_image_size(image_size=(512, 512), tile_size=(222, 222), stride=(111,
↪ 111))
(555, 555)
```

Returns
Updated image size that is divisible by tile size and stride.

Return type
tuple

`anomalib.data.utils.tiler.`**`downscale_image`**(*image*, *size*, *mode=ImageUpscaleMode.PADDING*)

Opposite of upscaling. This image downscales image to a desired size.

Parameters

- **`image`** (*`torch.Tensor`*) – Input image
- **`size`** (*`tuple`*) – Size to which image is down scaled.
- **`mode`** (*`str, optional`*) – Downscaling mode. Defaults to "padding".

Examples

```
>>> x = torch.rand(1, 3, 512, 512)
>>> y = upscale_image(image, upscale_size=(555, 555), mode="padding")
>>> y = downscale_image(y, size=(512, 512), mode='padding')
>>> torch.allclose(x, y)
True
```

Returns
Downscaled image

Return type
Tensor

`anomalib.data.utils.tiler.`**`upscale_image`**(*image*, *size*, *mode=ImageUpscaleMode.PADDING*)

Upscale image to the desired size via either padding or interpolation.

Parameters

- **image** (*torch.Tensor*) – Image
- **size** (*tuple*) – tuple to which image is upscaled.
- **mode** (*str, optional*) – Upscaling mode. Defaults to "padding".

Examples

```
>>> image = torch.rand(1, 3, 512, 512)
>>> image = upscale_image(image, size=(555, 555), mode="padding")
>>> image.shape
torch.Size([1, 3, 555, 555])
```

```
>>> image = torch.rand(1, 3, 512, 512)
>>> image = upscale_image(image, size=(555, 555), mode="interpolation")
>>> image.shape
torch.Size([1, 3, 555, 555])
```

Returns
Upscaled image.

Return type
Tensor

Synthetic Data Utils

Utilities to generate synthetic data.

`anomalib.data.utils.generators.`**`random_2d_perlin`**(*shape*, *res*, *fade=<function <lambda>>*)

Returns a random 2d perlin noise array.

Parameters

- **shape** (*tuple*) – Shape of the 2d map.

- **res** (*tuple[int | torch.Tensor, int | torch.Tensor]*) – Tuple of scales for perlin noise for height and width dimension.
- **fade** (*_type_, optional*) – Function used for fading the resulting 2d map. Defaults to equation 6*t**5-15*t**4+10*t**3.

Returns
Random 2d-array/tensor generated using perlin noise.

Return type
np.ndarray | torch.Tensor

Augmenter module to generates out-of-distribution samples for the DRAEM implementation.

`class anomalib.data.utils.augmenter.`**`Augmenter`**(*anomaly_source_path=None*, *p_anomalous=0.5*, *beta=(0.2, 1.0)*)

Bases: `object`

Class that generates noisy augmentations of input images.

Parameters

- **anomaly_source_path** (*str | None*) – Path to a folder of images that will be used as source of the anomalous
- **specified** (*noise. If not*) –
- **instead.** (*random noise will be used*) –
- **p_anomalous** (*float*) – Probability that the anomalous perturbation will be applied to a given image.
- **beta** (*float*) – Parameter that determines the opacity of the noise mask.

`augment_batch`(*batch*)

Generate anomalous augmentations for a batch of input images.

Parameters
batch (*torch.Tensor*) – Batch of input images

Return type
`tuple`[`Tensor`, `Tensor`]

Returns

- Augmented image to which anomalous perturbations have been added.
- Ground truth masks corresponding to the anomalous perturbations.

`generate_perturbation`(*height*, *width*, *anomaly_source_path=None*)

Generate an image containing a random anomalous perturbation using a source image.

Parameters

- **height** (*int*) – height of the generated image.
- **width** (int) – (int): width of the generated image.
- **anomaly_source_path** (*Path | str | None*) – Path to an image file. If not provided, random noise will be used
- **instead.** –

Return type
`tuple`[`ndarray`, `ndarray`]

Returns
Image containing a random anomalous perturbation, and the corresponding ground truth anomaly mask.

`rand_augmenter()`
Select 3 random transforms that will be applied to the anomaly source images.

Return type
`Sequential`

Returns
A selection of 3 transforms.

`anomalib.data.utils.augmenter.nextpow2(`*value*`)`
Return the smallest power of 2 greater than or equal to the input value.

Return type
`int`

Dataset that generates synthetic anomalies.

This dataset can be used when there is a lack of real anomalous data.

`class anomalib.data.utils.synthetic.SyntheticAnomalyDataset(`*task*, *transform*, *source_samples*`)`
Bases: *AnomalibDataset*

Dataset which reads synthetically generated anomalous images from a temporary folder.

Parameters

- **task** (*str*) – Task type, either "classification" or "segmentation".
- **transform** (*A.Compose*) – Transform object describing the transforms that are applied to the inputs.
- **source_samples** (*DataFrame*) – Normal samples to which the anomalous augmentations will be applied.

`classmethod from_dataset(`*dataset*`)`
Create a synthetic anomaly dataset from an existing dataset of normal images.

Parameters
dataset (AnomalibDataset) – Dataset consisting of only normal images that will be converrted to a synthetic anomalous dataset with a 50/50 normal anomalous split.

Return type
SyntheticAnomalyDataset

`anomalib.data.utils.synthetic.make_synthetic_dataset(`*source_samples*, *image_dir*, *mask_dir*, *anomalous_ratio=0.5*`)`
Convert a set of normal samples into a mixed set of normal and synthetic anomalous samples.

The synthetic images will be saved to the file system in the specified root directory under <root>/images. For the synthetic anomalous images, the masks will be saved under <root>/ground_truth.

Parameters

- **source_samples** (*DataFrame*) – Normal images that will be used as source for the synthetic anomalous images.
- **image_dir** (*Path*) – Directory to which the synthetic anomalous image files will be written.
- **mask_dir** (*Path*) – Directory to which the ground truth anomaly masks will be written.

- **anomalous_ratio** (*float*) – Fraction of source samples that will be converted into anomalous samples.

Return type
`DataFrame`

3.3.2 Models

Model Components	Learn more about components to design your own anomaly detection models.
Image Models	Learn more about image anomaly detection models.
Video Models	Learn more about video anomaly detection models.

Model Components

Feature Extractors	Learn more about anomalib feature extractors to extract features from backbones.
Dimensionality Reduction	Learn more about dimensionality reduction models.
Normalizing Flows	Learn more about `freia` normalizing flows model components.
Sampling Components	Learn more about various sampling components.
Filters	Learn more about filters for post-processing.
Classification	Learn more about classification model components.
Cluster	Learn more about cluster model components.
Statistical Components	Learn more about classification model components.

Feature Extractors

Feature extractors.

`class anomalib.models.components.feature_extractors.`**`BackboneParams`**(*class_path*, *init_args=<factory>*)

Bases: `object`

Used for serializing the backbone.

`class anomalib.models.components.feature_extractors.`**`TimmFeatureExtractor`**(*backbone*, *layers*, *pre_trained=True*, *requires_grad=False*)

Bases: `Module`

Extract features from a CNN.

Parameters

- **backbone** (*nn.Module*) – The backbone to which the feature extraction hooks are attached.
- **layers** (*Iterable[str]*) – List of layer names of the backbone to which the hooks are attached.

- **pre_trained** (*bool*) – Whether to use a pre-trained backbone. Defaults to True.
- **requires_grad** (*bool*) – Whether to require gradients for the backbone. Defaults to False. Models like `stfpm` use the feature extractor model as a trainable network. In such cases gradient computation is required.

Example

```
import torch
from anomalib.models.components.feature_extractors import TimmFeatureExtractor

model = TimmFeatureExtractor(model="resnet18", layers=['layer1', 'layer2', 'layer3
↪'])
input = torch.rand((32, 3, 256, 256))
features = model(input)

print([layer for layer in features.keys()])
# Output: ['layer1', 'layer2', 'layer3']

print([feature.shape for feature in features.values()]()
# Output: [torch.Size([32, 64, 64, 64]), torch.Size([32, 128, 32, 32]), torch.
↪Size([32, 256, 16, 16])]
```

forward(*inputs*)

Forward-pass input tensor into the CNN.

Parameters
inputs (*torch.Tensor*) – Input tensor

Return type
dict[str, Tensor]

Returns
Feature map extracted from the CNN

Example

```
model = TimmFeatureExtractor(model="resnet50", layers=['layer3'])
input = torch.rand((32, 3, 256, 256))
features = model.forward(input)
```

class anomalib.models.components.feature_extractors.**TorchFXFeatureExtractor**(*backbone*, *return_nodes*, *weights=None*, *requires_grad=False*, *tracer_kwargs=None*)

Bases: `Module`

Extract features from a CNN.

Parameters

- **backbone** (*str | BackboneParams | dict | nn.Module*) – The backbone to which the feature extraction hooks are attached. If the name is provided, the model is loaded from torchvision. Otherwise, the model class can be provided and it will try to load the weights from the provided weights file. Last, an instance of nn.Module can also be passed directly.

- **return_nodes** (*Iterable[str]*) – List of layer names of the backbone to which the hooks are attached. You can find the names of these nodes by using `get_graph_node_names` function.
- **weights** (*str | WeightsEnum | None*) – Weights enum to use for the model. Torchvision models require `WeightsEnum`. These enums are defined in `torchvision.models.<model>`. You can pass the weights path for custom models.
- **requires_grad** (*bool*) – Models like `stfpm` use the feature extractor for training. In such cases we should set `requires_grad` to `True`. Default is `False`.
- **tracer_kwargs** (*dict | None*) – a dictionary of keyword arguments for NodePathTracer (which passes them onto it's parent class torch.fx.Tracer). Can be used to allow not tracing through a list of problematic modules, by passing a list of *leaf_modules* as one of the *tracer_kwargs*.

Example

With torchvision models:

```python
import torch
from anomalib.models.components.feature_extractors import TorchFXFeatureExtractor
from torchvision.models.efficientnet import EfficientNet_B5_Weights

feature_extractor = TorchFXFeatureExtractor(
    backbone="efficientnet_b5",
    return_nodes=["features.6.8"],
    weights=EfficientNet_B5_Weights.DEFAULT
)

input = torch.rand((32, 3, 256, 256))
features = feature_extractor(input)

print([layer for layer in features.keys()])
# Output: ["features.6.8"]

print([feature.shape for feature in features.values()])
# Output: [torch.Size([32, 304, 8, 8])]
```

With custom models:

```python
import torch
from anomalib.models.components.feature_extractors import TorchFXFeatureExtractor

feature_extractor = TorchFXFeatureExtractor(
    "path.to.CustomModel", ["linear_relu_stack.3"], weights="path/to/weights.pth"
)

input = torch.randn(1, 1, 28, 28)
features = feature_extractor(input)
```

(continues on next page)

(continued from previous page)

```
print([layer for layer in features.keys()])
# Output: ["linear_relu_stack.3"]
```

with model instances:

```
import torch
from anomalib.models.components.feature_extractors import TorchFXFeatureExtractor
from timm import create_model

model = create_model("resnet18", pretrained=True)
feature_extractor = TorchFXFeatureExtractor(model, ["layer1"])

input = torch.rand((32, 3, 256, 256))
features = feature_extractor(input)

print([layer for layer in features.keys()])
# Output: ["layer1"]

print([feature.shape for feature in features.values()])
# Output: [torch.Size([32, 64, 64, 64])]
```

forward(*inputs*)

Extract features from the input.

Return type

`dict[str, Tensor]`

initialize_feature_extractor(*backbone*, *return_nodes*, *weights=None*, *requires_grad=False*, *tracer_kwargs=None*)

Extract features from a CNN.

Parameters

- **backbone** (`BackboneParams` | *nn.Module*) – The backbone to which the feature extraction hooks are attached. If the name is provided for BackboneParams, the model is loaded from torchvision. Otherwise, the model class can be provided and it will try to load the weights from the provided weights file. Last, an instance of the model can be provided as well, which will be used as-is.
- **return_nodes** (*Iterable[str]*) – List of layer names of the backbone to which the hooks are attached. You can find the names of these nodes by using `get_graph_node_names` function.
- **weights** (*str* | *WeightsEnum* | *None*) – Weights enum to use for the model. Torchvision models require `WeightsEnum`. These enums are defined in `torchvision.models.<model>`. You can pass the weights path for custom models.
- **requires_grad** (*bool*) – Models like `stfpm` use the feature extractor for training. In such cases we should set `requires_grad` to `True`. Default is `False`.
- **tracer_kwargs** (*dict* | *None*) – a dictionary of keyword arguments for NodePathTracer (which passes them onto it's parent class torch.fx.Tracer). Can be used to allow not tracing through a list of problematic modules, by passing a list of *leaf_modules* as one of the *tracer_kwargs*.

Return type

`GraphModule`

Returns
Feature Extractor based on TorchFX.

`anomalib.models.components.feature_extractors.`**`dryrun_find_featuremap_dims`**`(`*feature_extractor*, *input_size*, *layers*`)`

Dry run an empty image of *input_size* size to get the featuremap tensors' dimensions (num_features, resolution).

Returns

maping of *layer -> dimensions dict*
Each *dimension dict* has two keys: *num_features* (int) and `resolution`(tuple[int, int]).

Return type
tuple[int, int]

Dimensionality Reduction

Algorithms for decomposition and dimensionality reduction.

`class` `anomalib.models.components.dimensionality_reduction.`**`PCA`**`(`*n_components*`)`

Bases: `DynamicBufferMixin`

Principle Component Analysis (PCA).

Parameters
`n_components` (*`float`*) – Number of components. Can be either integer number of components or a ratio between 0-1.

Example

```
>>> import torch
>>> from anomalib.models.components import PCA
```

Create a PCA model with 2 components:

```
>>> pca = PCA(n_components=2)
```

Create a random embedding and fit a PCA model.

```
>>> embedding = torch.rand(1000, 5).cuda()
>>> pca = PCA(n_components=2)
>>> pca.fit(embedding)
```

Apply transformation:

```
>>> transformed = pca.transform(embedding)
>>> transformed.shape
torch.Size([1000, 2])
```

`fit``(`*dataset*`)`

Fits the PCA model to the dataset.

Parameters
`dataset` (*`torch.Tensor`*) – Input dataset to fit the model.

Return type
`None`

Example

```
>>> pca.fit(embedding)
>>> pca.singular_vectors
tensor([9.6053, 9.2763], device='cuda:0')
```

```
>>> pca.mean
tensor([0.4859, 0.4959, 0.4906, 0.5010, 0.5042], device='cuda:0')
```

fit_transform(*dataset*)

Fit and transform PCA to dataset.

Parameters
`dataset` (*torch.Tensor*) – Dataset to which the PCA if fit and transformed

Return type
`Tensor`

Returns
Transformed dataset

Example

```
>>> pca.fit_transform(embedding)
>>> transformed_embedding = pca.fit_transform(embedding)
>>> transformed_embedding.shape
torch.Size([1000, 2])
```

forward(*features*)

Transform the features.

Parameters
`features` (*torch.Tensor*) – Input features

Return type
`Tensor`

Returns
Transformed features

Example

```
>>> pca(embedding).shape
torch.Size([1000, 2])
```

inverse_transform(*features*)

Inverses the transformed features.

Parameters
`features` (*torch.Tensor*) – Transformed features

Return type
`Tensor`

Returns
Inverse features

Example

```
>>> inverse_embedding = pca.inverse_transform(transformed_embedding)
>>> inverse_embedding.shape
torch.Size([1000, 5])
```

transform(*features*)

Transform the features based on singular vectors calculated earlier.

Parameters
`features` (*torch.Tensor*) – Input features

Return type
`Tensor`

Returns
Transformed features

Example

```
>>> pca.transform(embedding)
>>> transformed_embedding = pca.transform(embedding)
```

```
>>> embedding.shape
torch.Size([1000, 5])
#
>>> transformed_embedding.shape
torch.Size([1000, 2])
```

class anomalib.models.components.dimensionality_reduction.**SparseRandomProjection**(*eps=0.1*, *random_state=None*)

Bases: `object`

Sparse Random Projection using PyTorch operations.

Parameters

- **eps** (*float, optional*) – Minimum distortion rate parameter for calculating Johnson-Lindenstrauss minimum dimensions. Defaults to `0.1`.
- **random_state** (*int | None, optional*) – Uses the seed to set the random state for sample_without_replacement function. Defaults to `None`.

Example

To fit and transform the embedding tensor, use the following code:

```
import torch
from anomalib.models.components import SparseRandomProjection

sparse_embedding = torch.rand(1000, 5).cuda()
model = SparseRandomProjection(eps=0.1)
```

Fit the model and transform the embedding tensor:

```
model.fit(sparse_embedding)
projected_embedding = model.transform(sparse_embedding)

print(projected_embedding.shape)
# Output: torch.Size([1000, 5920])
```

`fit`(*embedding*)

Generate sparse matrix from the embedding tensor.

Parameters
embedding (*torch.Tensor*) – embedding tensor for generating embedding

Returns

Return self to be used as

```
>>> model = SparseRandomProjection()
>>> model = model.fit()
```

Return type
(*SparseRandomProjection*)

`transform`(*embedding*)

Project the data by using matrix product with the random matrix.

Parameters
embedding (*torch.Tensor*) – Embedding of shape (n_samples, n_features) The input data to project into a smaller dimensional space

Returns

Sparse matrix of shape
(n_samples, n_components) Projected array.

Return type
projected_embedding (torch.Tensor)

Example

```
>>> projected_embedding = model.transform(embedding)
>>> projected_embedding.shape
torch.Size([1000, 5920])
```

Normalizing Flows

All In One Block Layer.

class `anomalib.models.components.flow.AllInOneBlock`(*dims_in*, *dims_c=None*, *subnet_constructor=None*, *affine_clamping=2.0*, *gin_block=False*, *global_affine_init=1.0*, *global_affine_type='SOFTPLUS'*, *permute_soft=False*, *learned_householder_permutation=0*, *reverse_permutation=False*)

Bases: `InvertibleModule`

Module combining the most common operations in a normalizing flow or similar model.

It combines affine coupling, permutation, and global affine transformation ('ActNorm'). It can also be used as GIN coupling block, perform learned householder permutations, and use an inverted pre-permutation. The affine transformation includes a soft clamping mechanism, first used in Real-NVP. The block as a whole performs the following computation:

$$y = VR\;\Psi(s_{\mathrm{global}}) \odot \mathrm{Coupling}\Big(R^{-1}V^{-1}x\Big) + t_{\mathrm{global}}$$

- The inverse pre-permutation of x (i.e. $R^{-1}V^{-1}$) is optional (see `reverse_permutation` below).
- The learned householder reflection matrix V is also optional all together (see `learned_householder_permutation` below).
- For the coupling, the input is split into x_1, x_2 along the channel dimension. Then the output of the coupling operation is the two halves $u = \mathrm{concat}(u_1, u_2)$.

$$\begin{aligned} u_1 &= x_1 \odot \exp\Big(\alpha\,\tanh\big(s(x_2)\big)\Big) + t(x_2) \\ u_2 &= x_2 \end{aligned}$$

Because $\tanh(s) \in [-1, 1]$, this clamping mechanism prevents exploding values in the exponential. The hyperparameter α can be adjusted.

Parameters

- **subnet_constructor** (`Callable | None`) – class or callable `f`, called as `f(channels_in, channels_out)` and should return a torch.nn.Module. Predicts coupling coefficients s, t.
- **affine_clamping** (`float`) – clamp the output of the multiplicative coefficients before exponentiation to +/- `affine_clamping` (see α above).
- **gin_block** (`bool`) – Turn the block into a GIN block from Sorrenson et al, 2019. Makes it so that the coupling operations as a whole is volume preserving.
- **global_affine_init** (`float`) – Initial value for the global affine scaling s_{global}.
- **global_affine_init** – `'SIGMOID'`, `'SOFTPLUS'`, or `'EXP'`. Defines the activation to be used on the beta for the global affine scaling (Ψ above).
- **permute_soft** (`bool`) – bool, whether to sample the permutation matrix R from $SO(N)$, or to use hard permutations instead. Note, `permute_soft=True` is very slow when working with >512 dimensions.
- **learned_householder_permutation** (`int`) – Int, if >0, turn on the matrix V above, that represents multiple learned householder reflections. Slow if large number. Dubious whether it actually helps network performance.
- **reverse_permutation** (`bool`) – Reverse the permutation before the block, as introduced by Putzky et al, 2019. Turns on the $R^{-1}V^{-1}$ pre-multiplication above.

forward(*x*, *c=None*, *rev=False*, *jac=True*)

See base class docstring.

Return type

`tuple[tuple[Tensor], Tensor]`

output_dims(*input_dims*)

Output dimensions of the layer.

Parameters

input_dims (*list[tuple[int]]*) – Input dimensions.

Returns

Output dimensions.

Return type

list[tuple[int]]

Sampling Components

Sampling methods.

class anomalib.models.components.sampling.**KCenterGreedy**(*embedding*, *sampling_ratio*)

Bases: object

Implements k-center-greedy method.

Parameters

- **embedding** (*torch.Tensor*) – Embedding vector extracted from a CNN
- **sampling_ratio** (*float*) – Ratio to choose coreset size from the embedding size.

Example

```
>>> embedding.shape
torch.Size([219520, 1536])
>>> sampler = KCenterGreedy(embedding=embedding)
>>> sampled_idxs = sampler.select_coreset_idxs()
>>> coreset = embedding[sampled_idxs]
>>> coreset.shape
torch.Size([219, 1536])
```

get_new_idx()

Get index value of a sample.

Based on minimum distance of the cluster

Returns

Sample index

Return type

int

reset_distances()

Reset minimum distances.

Return type

None

sample_coreset(*selected_idxs=None*)

Select coreset from the embedding.

Parameters
`selected_idxs` (list[int] | None) – index of samples already selected. Defaults to an empty set.

Returns
Output coreset

Return type
Tensor

Example

```
>>> embedding.shape
torch.Size([219520, 1536])
>>> sampler = KCenterGreedy(...)
>>> coreset = sampler.sample_coreset()
>>> coreset.shape
torch.Size([219, 1536])
```

`select_coreset_idxs`(*selected_idxs=None*)

Greedily form a coreset to minimize the maximum distance of a cluster.

Parameters
`selected_idxs` (list[int] | None) – index of samples already selected. Defaults to an empty set.

Return type
`list[int]`

Returns
indices of samples selected to minimize distance to cluster centers

`update_distances`(*cluster_centers*)

Update min distances given cluster centers.

Parameters
`cluster_centers` (*list[int]*) – indices of cluster centers

Return type
`None`

Filters

Implements filters used by models.

`class anomalib.models.components.filters.`**`GaussianBlur2d`**(*sigma, channels=1, kernel_size=None, normalize=True, border_type='reflect', padding='same'*)

Bases: `Module`

Compute GaussianBlur in 2d.

Makes use of kornia functions, but most notably the kernel is not computed during the forward pass, and does not depend on the input size. As a caveat, the number of channels that are expected have to be provided during initialization.

`forward`(*input_tensor*)

Blur the input with the computed Gaussian.

Parameters
`input_tensor` (*torch.Tensor*) – Input tensor to be blurred.

Returns
Blurred output tensor.

Return type
Tensor

Classification

Classification modules.

`class anomalib.models.components.classification.FeatureScalingMethod`(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `str`, `Enum`

Determines how the feature embeddings are scaled.

`class anomalib.models.components.classification.KDEClassifier`(*n_pca_components=16*, *feature_scaling_method=FeatureScalingMethod.SCALE*, *max_training_points=40000*)

Bases: `Module`

Classification module for KDE-based anomaly detection.

Parameters

- **n_pca_components** (*int, optional*) – Number of PCA components. Defaults to 16.
- **feature_scaling_method** (`FeatureScalingMethod`, *optional*) – Scaling method applied to features before passing to KDE. Options are *norm* (normalize to unit vector length) and *scale* (scale to max length observed in training).
- **max_training_points** (*int, optional*) – Maximum number of training points to fit the KDE model. Defaults to 40000.

`compute_kde_scores`(*features*, *as_log_likelihood=False*)

Compute the KDE scores.

The scores calculated from the KDE model are converted to densities. If *as_log_likelihood* is set to true then
the log of the scores are calculated.

Parameters

- **features** (*torch.Tensor*) – Features to which the PCA model is fit.
- **as_log_likelihood** (*bool | None, optional*) – If true, gets log likelihood scores. Defaults to False.

Returns
Score

Return type
(torch.Tensor)

static compute_probabilities(*scores*)

Convert density scores to anomaly probabilities (see https://www.desmos.com/calculator/ifju7eesg7).

Parameters
scores (*torch.Tensor*) – density of an image.

Return type
`Tensor`

Returns
probability that image with {density} is anomalous

fit(*embeddings*)

Fit a kde model to embeddings.

Parameters
embeddings (*torch.Tensor*) – Input embeddings to fit the model.

Return type
`bool`

Returns
Boolean confirming whether the training is successful.

forward(*features*)

Make predictions on extracted features.

Return type
`Tensor`

pre_process(*feature_stack*, *max_length=None*)

Pre-process the CNN features.

Parameters

- **feature_stack** (*torch.Tensor*) – Features extracted from CNN
- **max_length** (*Tensor | None*) – Used to unit normalize the feature_stack vector. If `max_len` is not provided, the length is calculated from the `feature_stack`. Defaults to None.

Returns
Stacked features and length

Return type
(Tuple)

predict(*features*)

Predicts the probability that the features belong to the anomalous class.

Parameters
features (*torch.Tensor*) – Feature from which the output probabilities are detected.

Return type
`Tensor`

Returns
Detection probabilities

Cluster

Clustering algorithm implementations using PyTorch.

class anomalib.models.components.cluster.**GaussianMixture**(*n_components*, *n_iter=100*, *tol=0.001*)

Bases: `DynamicBufferMixin`

Gaussian Mixture Model.

Parameters

- **n_components** (*int*) – Number of components.
- **n_iter** (*int*) – Maximum number of iterations to perform. Defaults to `100`.
- **tol** (*float*) – Convergence threshold. Defaults to `1e-3`.

Example

The following examples shows how to fit a Gaussian Mixture Model to some data and get the cluster means and predicted labels and log-likelihood scores of the data.

```
>>> import torch
>>> from anomalib.models.components.cluster import GaussianMixture
>>> model = GaussianMixture(n_components=2)
>>> data = torch.tensor(
...     [
...             [2, 1], [2, 2], [2, 3],
...             [7, 5], [8, 5], [9, 5],
...     ]
... ).float()
>>> model.fit(data)
>>> model.means  # get the means of the gaussians
tensor([[8., 5.],
        [2., 2.]])
>>> model.predict(data)  # get the predicted cluster label of each sample
tensor([1, 1, 1, 0, 0, 0])
>>> model.score_samples(data)  # get the log-likelihood score of each sample
tensor([3.8295, 4.5795, 3.8295, 3.8295, 4.5795, 3.8295])
```

fit(*data*)

Fit the model to the data.

Parameters
data (*Tensor*) – Data to fit the model to. Tensor of shape (n_samples, n_features).

Return type
None

predict(*data*)

Predict the cluster labels of the data.

Parameters
data (*Tensor*) – Samples to assign to clusters. Tensor of shape (n_samples, n_features).

Returns
Tensor of shape (n_samples,) containing the predicted cluster label of each sample.

Return type
Tensor

`score_samples`(*data*)

Assign a likelihood score to each sample in the data.

Parameters
`data` (*Tensor*) – Samples to assign scores to. Tensor of shape (n_samples, n_features).

Returns
Tensor of shape (n_samples,) containing the log-likelihood score of each sample.

Return type
Tensor

class `anomalib.models.components.cluster.KMeans`(*n_clusters*, *max_iter=10*)

Bases: `object`

Initialize the KMeans object.

Parameters

- **n_clusters** (*int*) – The number of clusters to create.
- **max_iter** (*int, optional)*) – The maximum number of iterations to run the algorithm. Defaults to 10.

`fit`(*inputs*)

Fit the K-means algorithm to the input data.

Parameters
`inputs` (*torch.Tensor*) – Input data of shape (batch_size, n_features).

Returns
A tuple containing the labels of the input data with respect to the identified clusters and the cluster centers themselves. The labels have a shape of (batch_size,) and the cluster centers have a shape of (n_clusters, n_features).

Return type
tuple

Raises
`ValueError` – If the number of clusters is less than or equal to 0.

`predict`(*inputs*)

Predict the labels of input data based on the fitted model.

Parameters
`inputs` (*torch.Tensor*) – Input data of shape (batch_size, n_features).

Returns
The predicted labels of the input data with respect to the identified clusters.

Return type
torch.Tensor

Raises
`AttributeError` – If the KMeans object has not been fitted to input data.

Stats Components

Statistical functions.

class `anomalib.models.components.stats.`**`GaussianKDE`**(*dataset=None*)

Bases: `DynamicBufferMixin`

Gaussian Kernel Density Estimation.

Parameters
`dataset` (*`Tensor | None, optional`*) – Dataset on which to fit the KDE model. Defaults to None.

static cov(*tensor*)

Calculate the unbiased covariance matrix.

Parameters
`tensor` (*`torch.Tensor`*) – Input tensor from which covariance matrix is computed.

Return type
`Tensor`

Returns
Output covariance matrix.

`fit`(*dataset*)

Fit a KDE model to the input dataset.

Parameters
`dataset` (*`torch.Tensor`*) – Input dataset.

Return type
`None`

Returns
None

`forward`(*features*)

Get the KDE estimates from the feature map.

Parameters
`features` (*`torch.Tensor`*) – Feature map extracted from the CNN

Return type
`Tensor`

Returns: KDE Estimates

class `anomalib.models.components.stats.`**`MultiVariateGaussian`**

Bases: `DynamicBufferMixin`, `Module`

Multi Variate Gaussian Distribution.

`fit`(*embedding*)

Fit multi-variate gaussian distribution to the input embedding.

Parameters
`embedding` (*`torch.Tensor`*) – Embedding vector extracted from CNN.

Return type
`list[Tensor]`

Returns
Mean and the covariance of the embedding.

`forward`(*embedding*)

Calculate multivariate Gaussian distribution.

Parameters
`embedding` (*torch.Tensor*) – CNN features whose dimensionality is reduced via either random sampling or PCA.

Return type
`list[Tensor]`

Returns
mean and inverse covariance of the multi-variate gaussian distribution that fits the features.

Image Models

CFA	Coupled-hypersphere-based Feature Adaptation for Target-Oriented Anomaly Localization
C-Flow	Real-Time Unsupervised Anomaly Detection via Conditional Normalizing Flows
CS-Flow	Fully Convolutional Cross-Scale-Flows for Image-based Defect Detection
DFKDE	Deep Feature Kernel Density Estimation
DFM	Probabilistic Modeling of Deep Features for Out-of-Distribution and Adversarial Detection
DRAEM	DRÆM – A discriminatively trained reconstruction embedding for surface anomaly detection
DSR	DSR – A Dual Subspace Re-Projection Network for Surface Anomaly Detection
Efficient AD	EfficientAD: Accurate Visual Anomaly Detection at Millisecond-Level Latencies
FastFlow	FastFlow: Unsupervised Anomaly Detection and Localization via 2D Normalizing Flows
GANomaly	GANomaly: Semi-Supervised Anomaly Detection via Adversarial Training
PaDiM	PaDiM: A Patch Distribution Modeling Framework for Anomaly Detection and Localization
Patchcore	Towards Total Recall in Industrial Anomaly Detection
Reverse Distillation	Anomaly Detection via Reverse Distillation from One-Class Embedding.
R-KDE	Region-Based Kernel Density Estimation (RKDE)
STFPM	Student-Teacher Feature Pyramid Matching for Unsupervised Anomaly Detection
U-Flow	U-Flow: A U-shaped Normalizing Flow for Anomaly Detection with Unsupervised Threshold
WinCLIP	WinCLIP: Zero-/Few-Shot Anomaly Classification and Segmentation

CFA

Lightning Implementatation of the CFA Model.

CFA: Coupled-hypersphere-based Feature Adaptation for Target-Oriented Anomaly Localization

Paper https://arxiv.org/abs/2206.04325

class `anomalib.models.image.cfa.lightning_model.`**`Cfa`**(*backbone='wide_resnet50_2'*, *gamma_c=1*, *gamma_d=1*, *num_nearest_neighbors=3*, *num_hard_negative_features=3*, *radius=1e-05*)

Bases: `AnomalyModule`

CFA: Coupled-hypersphere-based Feature Adaptation for Target-Oriented Anomaly Localization.

Parameters

- **backbone** (*str*) – Backbone CNN network Defaults to `"wide_resnet50_2"`.
- **gamma_c** (*int, optional*) – gamma_c value from the paper. Defaults to `1`.
- **gamma_d** (*int, optional*) – gamma_d value from the paper. Defaults to `1`.
- **num_nearest_neighbors** (*int*) – Number of nearest neighbors. Defaults to `3`.
- **num_hard_negative_features** (*int*) – Number of hard negative features. Defaults to `3`.
- **radius** (*float*) – Radius of the hypersphere to search the soft boundary. Defaults to `1e-5`.

backward(*loss*, **args*, ***kwargs*)

Perform backward-pass for the CFA model.

Parameters

- **loss** (*torch.Tensor*) – Loss value.
- ***args** – Arguments.
- ****kwargs** – Keyword arguments.

Return type
`None`

configure_optimizers()

Configure optimizers for the CFA Model.

Returns
Adam optimizer for each decoder

Return type
Optimizer

property learning_type: LearningType

Return the learning type of the model.

Returns
Learning type of the model.

Return type
LearningType

on_train_start()

Initialize the centroid for the memory bank computation.

Return type
`None`

`property trainer_arguments: dict[str, Any]`

CFA specific trainer arguments.

training_step(*batch*, **args*, ***kwargs*)

Perform the training step for the CFA model.

Parameters

- **batch** (*dict[str, str | torch.Tensor]*) – Batch input.
- ***args** – Arguments.
- ****kwargs** – Keyword arguments.

Returns
Loss value.

Return type
STEP_OUTPUT

validation_step(*batch*, **args*, ***kwargs*)

Perform the validation step for the CFA model.

Parameters

- **batch** (*dict[str, str | torch.Tensor]*) – Input batch.
- ***args** – Arguments.
- ****kwargs** – Keyword arguments.

Returns
Anomaly map computed by the model.

Return type
dict

Torch Implementatation of the CFA Model.

CFA: Coupled-hypersphere-based Feature Adaptation for Target-Oriented Anomaly Localization

Paper https://arxiv.org/abs/2206.04325

`class anomalib.models.image.cfa.torch_model.`**CfaModel**(*backbone*, *gamma_c*, *gamma_d*, *num_nearest_neighbors*, *num_hard_negative_features*, *radius*)

Bases: `DynamicBufferMixin`

Torch implementation of the CFA Model.

Parameters

- **backbone** (*str*) – Backbone CNN network.
- **gamma_c** (*int*) – gamma_c parameter from the paper.
- **gamma_d** (*int*) – gamma_d parameter from the paper.
- **num_nearest_neighbors** (*int*) – Number of nearest neighbors.
- **num_hard_negative_features** (*int*) – Number of hard negative features.
- **radius** (*float*) – Radius of the hypersphere to search the soft boundary.

`compute_distance`(*target_oriented_features*)

Compute distance using target oriented features.

Parameters

`target_oriented_features` (*`torch.Tensor`*) – Target oriented features computed using the descriptor.

Returns

Distance tensor.

Return type

Tensor

`forward`(*input_tensor*)

Forward pass.

Parameters

`input_tensor` (*`torch.Tensor`*) – Input tensor.

Raises

`ValueError` – When the memory bank is not initialized.

Returns

Loss or anomaly map depending on the train/eval mode.

Return type

Tensor

`get_scale`(*input_size*)

Get the scale of the feature map.

Parameters

`input_size` (*`tuple[int, int]`*) – Input size of the image tensor.

Return type

`Size`

`initialize_centroid`(*data_loader*)

Initialize the Centroid of the Memory Bank.

Parameters

`data_loader` (*`DataLoader`*) – Train Dataloader.

Returns

Memory Bank.

Return type

Tensor

Loss function for the Cfa Model Implementation.

`class` `anomalib.models.image.cfa.loss.`**`CfaLoss`**(*num_nearest_neighbors*, *num_hard_negative_features*, *radius*)

Bases: `Module`

Cfa Loss.

Parameters

- **`num_nearest_neighbors`** (*`int`*) – Number of nearest neighbors.
- **`num_hard_negative_features`** (*`int`*) – Number of hard negative features.
- **`radius`** (*`float`*) – Radius of the hypersphere to search the soft boundary.

`forward`(*distance*)

Compute the CFA loss.

Parameters
`distance` (*torch.Tensor*) – Distance computed using target oriented features.

Returns
CFA loss.

Return type
Tensor

Anomaly Map Generator for the CFA model implementation.

`class anomalib.models.image.cfa.anomaly_map.AnomalyMapGenerator`(*num_nearest_neighbors*, *sigma=4*)

Bases: `Module`

Generate Anomaly Heatmap.

`compute_anomaly_map`(*score*, *image_size=None*)

Compute anomaly map based on the score.

Parameters

- `score` (*torch.Tensor*) – Score tensor.
- `image_size` (*tuple[int, int] | torch.Size | None, optional*) – Size of the input image.

Returns
Anomaly map.

Return type
Tensor

`compute_score`(*distance*, *scale*)

Compute score based on the distance.

Parameters

- `distance` (*torch.Tensor*) – Distance tensor computed using target oriented features.
- `scale` (*tuple[int, int]*) – Height and width of the largest feature map.

Returns
Score value.

Return type
Tensor

`forward`(***kwargs*)

Return anomaly map.

Raises
distance` and scale keys are not found –

Returns
Anomaly heatmap.

Return type
Tensor

C-Flow

Cflow.

Real-Time Unsupervised Anomaly Detection via Conditional Normalizing Flows.

For more details, see the paper: Real-Time Unsupervised Anomaly Detection via Conditional Normalizing Flows.

`class anomalib.models.image.cflow.lightning_model.`**`Cflow`**(*backbone='wide_resnet50_2', layers=('layer2', 'layer3', 'layer4'), pre_trained=True, fiber_batch_size=64, decoder='freia-cflow', condition_vector=128, coupling_blocks=8, clamp_alpha=1.9, permute_soft=False, lr=0.0001*)

Bases: `AnomalyModule`

PL Lightning Module for the CFLOW algorithm.

Parameters

- **backbone** (*str, optional*) – Backbone CNN architecture. Defaults to `"wide_resnet50_2"`.
- **layers** (*Sequence[str], optional*) – Layers to extract features from. Defaults to `("layer2", "layer3", "layer4")`.
- **pre_trained** (*bool, optional*) – Whether to use pre-trained weights. Defaults to `True`.
- **fiber_batch_size** (*int, optional*) – Fiber batch size. Defaults to `64`.
- **decoder** (*str, optional*) – Decoder architecture. Defaults to `"freia-cflow"`.
- **condition_vector** (*int, optional*) – Condition vector size. Defaults to `128`.
- **coupling_blocks** (*int, optional*) – Number of coupling blocks. Defaults to `8`.
- **clamp_alpha** (*float, optional*) – Clamping value for the alpha parameter. Defaults to `1.9`.
- **permute_soft** (*bool, optional*) – Whether to use soft permutation. Defaults to `False`.
- **lr** (*float, optional*) – Learning rate. Defaults to `0.0001`.

`configure_optimizers()`

Configure optimizers for each decoder.

Returns
Adam optimizer for each decoder

Return type
Optimizer

`property` **`learning_type: LearningType`**

Return the learning type of the model.

Returns
Learning type of the model.

Return type
LearningType

`property trainer_arguments: dict[str, Any]`

C-FLOW specific trainer arguments.

`training_step`(*batch*, **args*, ***kwargs*)

Perform the training step of CFLOW.

For each batch, decoder layers are trained with a dynamic fiber batch size. Training step is performed manually as multiple training steps are involved

per batch of input images

Parameters

- **batch** (*dict[str, str | torch.Tensor]*) – Input batch
- ***args** – Arguments.
- ****kwargs** – Keyword arguments.

Return type

`Union`[`Tensor`, `Mapping`[`str`, `Any`], `None`]

Returns

Loss value for the batch

`validation_step`(*batch*, **args*, ***kwargs*)

Perform the validation step of CFLOW.

Similar to the training step, encoder features are extracted from the CNN for each batch, and anomaly map is computed.

Parameters

- **batch** (*dict[str, str | torch.Tensor]*) – Input batch
- ***args** – Arguments.
- ****kwargs** – Keyword arguments.

Return type

`Union`[`Tensor`, `Mapping`[`str`, `Any`], `None`]

Returns

Dictionary containing images, anomaly maps, true labels and masks. These are required in *validation_epoch_end* for feature concatenation.

PyTorch model for CFlow model implementation.

`class anomalib.models.image.cflow.torch_model.CflowModel`(*backbone*, *layers*, *pre_trained=True*, *fiber_batch_size=64*, *decoder='freia-cflow'*, *condition_vector=128*, *coupling_blocks=8*, *clamp_alpha=1.9*, *permute_soft=False*)

Bases: `Module`

CFLOW: Conditional Normalizing Flows.

Parameters

- **backbone** (*str*) – Backbone CNN architecture.

- **layers** (*Sequence[str]*) – Layers to extract features from.
- **pre_trained** (*bool*) – Whether to use pre-trained weights. Defaults to `True`.
- **fiber_batch_size** (*int*) – Fiber batch size. Defaults to `64`.
- **decoder** (*str*) – Decoder architecture. Defaults to `"freia-cflow"`.
- **condition_vector** (*int*) – Condition vector size. Defaults to `128`.
- **coupling_blocks** (*int*) – Number of coupling blocks. Defaults to `8`.
- **clamp_alpha** (*float*) – Clamping value for the alpha parameter. Defaults to `1.9`.
- **permute_soft** (*bool*) – Whether to use soft permutation. Defaults to `False`.

forward(*images*)

Forward-pass images into the network to extract encoder features and compute probability.

Parameters

images (`Tensor`) – Batch of images.

Return type

`Tensor`

Returns

Predicted anomaly maps.

Anomaly Map Generator for CFlow model implementation.

class `anomalib.models.image.cflow.anomaly_map.`**AnomalyMapGenerator**(*pool_layers*)

Bases: `Module`

Generate Anomaly Heatmap.

compute_anomaly_map(*distribution*, *height*, *width*, *image_size*)

Compute the layer map based on likelihood estimation.

Parameters

- **distribution** (*list[torch.Tensor]*) – List of likelihoods for each layer.
- **height** (*list[int]*) – List of heights of the feature maps.
- **width** (*list[int]*) – List of widths of the feature maps.
- **image_size** (*tuple[int, int] | torch.Size | None*) – Size of the input image.

Return type

`Tensor`

Returns

Final Anomaly Map

forward(***kwargs*)

Return anomaly_map.

Expects *distribution*, *height* and 'width' keywords to be passed explicitly

Example

```
>>> anomaly_map_generator = AnomalyMapGenerator(image_size=tuple(hparams.model.
↪input_size),
>>>         pool_layers=pool_layers)
>>> output = self.anomaly_map_generator(distribution=dist, height=height,␣
↪width=width)
```

Raises
`ValueError` – *distribution*, *height* and ‘width’ keys are not found

Returns
anomaly map

Return type
torch.Tensor

CS-Flow

Fully Convolutional Cross-Scale-Flows for Image-based Defect Detection.

https://arxiv.org/pdf/2110.02855.pdf

class `anomalib.models.image.csflow.lightning_model.`**`Csflow`**(*cross_conv_hidden_channels=1024*, *n_coupling_blocks=4*, *clamp=3*, *num_channels=3*)

Bases: `AnomalyModule`

Fully Convolutional Cross-Scale-Flows for Image-based Defect Detection.

Parameters

- **n_coupling_blocks** (*int*) – Number of coupling blocks in the model. Defaults to `4`.
- **cross_conv_hidden_channels** (*int*) – Number of hidden channels in the cross convolution. Defaults to `1024`.
- **clamp** (*int*) – Clamp value for glow layer. Defaults to `3`.
- **num_channels** (*int*) – Number of channels in the model. Defaults to `3`.

`configure_optimizers()`

Configure optimizers.

Returns
Adam optimizer

Return type
Optimizer

`property learning_type: LearningType`

Return the learning type of the model.

Returns
Learning type of the model.

Return type
LearningType

property trainer_arguments: dict[str, Any]

CS-Flow-specific trainer arguments.

training_step(*batch*, **args*, ***kwargs*)

Perform the training step of CS-Flow.

Parameters

- **batch** (*dict[str, str | torch.Tensor]*) – Input batch
- **args** – Arguments.
- **kwargs** – Keyword arguments.

Return type

Union[Tensor, Mapping[str, Any], None]

Returns

Loss value

validation_step(*batch*, **args*, ***kwargs*)

Perform the validation step for CS Flow.

Parameters

- **batch** (*torch.Tensor*) – Input batch
- **args** – Arguments.
- **kwargs** – Keyword arguments.

Returns

Dictionary containing the anomaly map, scores, etc.

Return type

dict[str, torch.Tensor]

PyTorch model for CS-Flow implementation.

class anomalib.models.image.csflow.torch_model.**CsFlowModel**(*input_size*, *cross_conv_hidden_channels*, *n_coupling_blocks=4*, *clamp=3*, *num_channels=3*)

Bases: Module

CS Flow Module.

Parameters

- **input_size** (*tuple[int, int]*) – Input image size.
- **cross_conv_hidden_channels** (*int*) – Number of hidden channels in the cross convolution.
- **n_coupling_blocks** (*int*) – Number of coupling blocks. Defaults to 4.
- **clamp** (*float*) – Clamp value for the coupling blocks. Defaults to 3.
- **num_channels** (*int*) – Number of channels in the input image. Defaults to 3.

forward(*images*)

Forward method of the model.

Parameters

images (*torch.Tensor*) – Input images.

Returns

During training: tuple containing the z_distribution for three scales
and the sum of log determinant of the Jacobian. During evaluation: tuple containing anomaly maps and anomaly scores

Return type
tuple[torch.Tensor, torch.Tensor]

Loss function for the CS-Flow Model Implementation.

`class anomalib.models.image.csflow.loss.`**`CsFlowLoss`**`(`**args*, ***kwargs*`)`

Bases: `Module`

Loss function for the CS-Flow Model Implementation.

`forward``(`*z_dist*, *jacobians*`)`

Compute the loss CS-Flow.

Parameters

- **`z_dist`** (*`torch.Tensor`*) – Latent space image mappings from NF.
- **`jacobians`** (*`torch.Tensor`*) – Jacobians of the distribution

Return type
`Tensor`

Returns
Loss value

Anomaly Map Generator for CS-Flow model.

`class anomalib.models.image.csflow.anomaly_map.`**`AnomalyMapGenerator`**`(`*input_dims*, *mode=AnomalyMapMode.ALL*`)`

Bases: `Module`

Anomaly Map Generator for CS-Flow model.

Parameters

- **`input_dims`** (*`tuple[int, int, int]`*) – Input dimensions.
- **`mode`** (`AnomalyMapMode`) – Anomaly map mode. Defaults to `AnomalyMapMode.ALL`.

`forward``(`*inputs*`)`

Get anomaly maps by taking mean of the z-distributions across channels.

By default it computes anomaly maps for all the scales as it gave better performance on initial tests. Use `AnomalyMapMode.MAX` for the largest scale as mentioned in the paper.

Parameters

- **`inputs`** (*`torch.Tensor`*) – z-distributions for the three scales.
- **`mode`** (`AnomalyMapMode`) – Anomaly map mode.

Returns
Anomaly maps.

Return type
Tensor

`class anomalib.models.image.csflow.anomaly_map.AnomalyMapMode(`*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*`)`

Bases: `str`, `Enum`

Generate anomaly map from all the scales or the max.

DFKDE

DFKDE: Deep Feature Kernel Density Estimation.

`class anomalib.models.image.dfkde.lightning_model.Dfkde(`*backbone='resnet18'*, *layers=('layer4',)*, *pre_trained=True*, *n_pca_components=16*, *feature_scaling_method=FeatureScalingMethod.SCALE*, *max_training_points=40000*`)`

Bases: `MemoryBankMixin`, `AnomalyModule`

DFKDE: Deep Feature Kernel Density Estimation.

Parameters

- **backbone** (*str*) – Pre-trained model backbone. Defaults to `"resnet18"`.
- **layers** (*Sequence[str], optional*) – Layers to extract features from. Defaults to `("layer4",)`.
- **pre_trained** (*bool, optional*) – Boolean to check whether to use a pre_trained backbone. Defaults to `True`.
- **n_pca_components** (*int, optional*) – Number of PCA components. Defaults to `16`.
- **feature_scaling_method** (*FeatureScalingMethod, optional*) – Feature scaling method. Defaults to `FeatureScalingMethod.SCALE`.
- **max_training_points** (*int, optional*) – Number of training points to fit the KDE model. Defaults to `40000`.

`static configure_optimizers()`

DFKDE doesn't require optimization, therefore returns no optimizers.

Return type
`None`

`fit()`

Fit a KDE Model to the embedding collected from the training set.

Return type
`None`

`property learning_type: LearningType`

Return the learning type of the model.

Returns
Learning type of the model.

Return type
LearningType

property trainer_arguments: dict[str, Any]

Return DFKDE-specific trainer arguments.

training_step(*batch*, **args*, ***kwargs*)

Perform the training step of DFKDE. For each batch, features are extracted from the CNN.

Parameters

- **(batch** (*batch*) – dict[str, str | torch.Tensor]): Batch containing image filename, image, label and mask
- **args** – Arguments.
- **kwargs** – Keyword arguments.

Return type

`None`

Returns

Deep CNN features.

validation_step(*batch*, **args*, ***kwargs*)

Perform the validation step of DFKDE.

Similar to the training step, features are extracted from the CNN for each batch.

Parameters

- **batch** (*dict[str, str | torch.Tensor]*) – Input batch
- **args** – Arguments.
- **kwargs** – Keyword arguments.

Return type

`Union`[`Tensor`, `Mapping`[`str`, `Any`], `None`]

Returns

Dictionary containing probability, prediction and ground truth values.

Normality model of DFKDE.

class anomalib.models.image.dfkde.torch_model.**DfkdeModel**(*backbone*, *layers*, *pre_trained=True*, *n_pca_components=16*, *feature_scaling_method=FeatureScalingMethod.SCALE*, *max_training_points=40000*)

Bases: `Module`

Normality Model for the DFKDE algorithm.

Parameters

- **backbone** (*str*) – Pre-trained model backbone.
- **layers** (*Sequence[str]*) – Layers to extract features from.
- **pre_trained** (*bool, optional*) – Boolean to check whether to use a pre_trained backbone. Defaults to `True`.
- **n_pca_components** (*int, optional*) – Number of PCA components. Defaults to `16`.
- **feature_scaling_method** (*FeatureScalingMethod, optional*) – Feature scaling method. Defaults to `FeatureScalingMethod.SCALE`.

- **max_training_points** (*int, optional*) – Number of training points to fit the KDE model. Defaults to `40000`.

forward(*batch*)

Prediction by normality model.

Parameters
batch (*torch.Tensor*) – Input images.

Returns
Predictions

Return type
Tensor

get_features(*batch*)

Extract features from the pretrained network.

Parameters
batch (*torch.Tensor*) – Image batch.

Returns
torch.Tensor containing extracted features.

Return type
Tensor

DFM

DFM: Deep Feature Modeling.

https://arxiv.org/abs/1909.11786

class anomalib.models.image.dfm.lightning_model.**Dfm**(*backbone='resnet50'*, *layer='layer3'*, *pre_trained=True*, *pooling_kernel_size=4*, *pca_level=0.97*, *score_type='fre'*)

Bases: `MemoryBankMixin`, `AnomalyModule`

DFM: Deep Featured Kernel Density Estimation.

Parameters

- **backbone** (*str*) – Backbone CNN network Defaults to `"resnet50"`.
- **layer** (*str*) – Layer to extract features from the backbone CNN Defaults to `"layer3"`.
- **pre_trained** (*bool, optional*) – Boolean to check whether to use a pre_trained backbone. Defaults to `True`.
- **pooling_kernel_size** (*int, optional*) – Kernel size to pool features extracted from the CNN. Defaults to `4`.
- **pca_level** (*float, optional*) – Ratio from which number of components for PCA are calculated. Defaults to `0.97`.
- **score_type** (*str, optional*) – Scoring type. Options are *fre* and *nll*. Defaults to `fre`.

static **configure_optimizers**()

DFM doesn't require optimization, therefore returns no optimizers.

Return type
`None`

`fit()`

Fit a PCA transformation and a Gaussian model to dataset.

Return type
`None`

`property learning_type: LearningType`

Return the learning type of the model.

Returns
Learning type of the model.

Return type
LearningType

`property trainer_arguments: dict[str, Any]`

Return DFM-specific trainer arguments.

`training_step`(*batch*, **args*, ***kwargs*)

Perform the training step of DFM.

For each batch, features are extracted from the CNN.

Parameters

- **batch** (*dict[str, str | torch.Tensor]*) – Input batch
- **args** – Arguments.
- **kwargs** – Keyword arguments.

Return type
`None`

Returns
Deep CNN features.

`validation_step`(*batch*, **args*, ***kwargs*)

Perform the validation step of DFM.

Similar to the training step, features are extracted from the CNN for each batch.

Parameters

- **batch** (*dict[str, str | torch.Tensor]*) – Input batch
- **args** – Arguments.
- **kwargs** – Keyword arguments.

Return type
`Union[Tensor, Mapping[str, Any], None]`

Returns
Dictionary containing FRE anomaly scores and anomaly maps.

PyTorch model for DFM model implementation.

`class anomalib.models.image.dfm.torch_model.DFMModel`(*backbone*, *layer*, *pre_trained=True*, *pooling_kernel_size=4*, *n_comps=0.97*, *score_type='fre'*)

Bases: `Module`

Model for the DFM algorithm.

Parameters

- `backbone` (*str*) – Pre-trained model backbone.
- `layer` (*str*) – Layer from which to extract features.
- `pre_trained` (*bool, optional*) – Boolean to check whether to use a pre_trained backbone. Defaults to `True`.
- `pooling_kernel_size` (*int, optional*) – Kernel size to pool features extracted from the CNN. Defaults to `4`.
- `n_comps` (*float, optional*) – Ratio from which number of components for PCA are calculated. Defaults to `0.97`.
- `score_type` (*str, optional*) – Scoring type. Options are *fre* and *nll*. Anomaly Defaults to `fre`. Segmentation is supported with *fre* only. If using *nll*, set *task* in config.yaml to classification Defaults to `classification`.

`fit`(*dataset*)

Fit a pca transformation and a Gaussian model to dataset.

Parameters
`dataset` (*torch.Tensor*) – Input dataset to fit the model.

Return type
`None`

`forward`(*batch*)

Compute score from input images.

Parameters
`batch` (*torch.Tensor*) – Input images

Returns
Scores

Return type
Tensor

`get_features`(*batch*)

Extract features from the pretrained network.

Parameters
`batch` (*torch.Tensor*) – Image batch.

Returns
torch.Tensor containing extracted features.

Return type
Tensor

`score`(*features*, *feature_shapes*)

Compute scores.

Scores are either PCA-based feature reconstruction error (FRE) scores or the Gaussian density-based NLL scores

Parameters

- `features` (*torch.Tensor*) – semantic features on which PCA and density modeling is performed.

- `feature_shapes` (*tuple*) – shape of *features* tensor. Used to generate anomaly map of correct shape.

Returns
numpy array of scores

Return type
score (torch.Tensor)

class `anomalib.models.image.dfm.torch_model.`**`SingleClassGaussian`**

Bases: `DynamicBufferMixin`

Model Gaussian distribution over a set of points.

`fit`(*dataset*)

Fit a Gaussian model to dataset X.

Covariance matrix is not calculated directly using: `C = X.X^T` Instead, it is represented in terms of the Singular Value Decomposition of X: `X = U.S.V^T` Hence, `C = U.S^2.U^T` This simplifies the calculation of the log-likelihood without requiring full matrix inversion.

Parameters
`dataset` (*torch.Tensor*) – Input dataset to fit the model.

Return type
`None`

`forward`(*dataset*)

Provide the same functionality as *fit*.

Transforms the input dataset based on singular values calculated earlier.

Parameters
`dataset` (*torch.Tensor*) – Input dataset

Return type
`None`

`score_samples`(*features*)

Compute the NLL (negative log likelihood) scores.

Parameters
`features` (*torch.Tensor*) – semantic features on which density modeling is performed.

Returns
Torch tensor of scores

Return type
nll (torch.Tensor)

DRAEM

DRÆM - A discriminatively trained reconstruction embedding for surface anomaly detection.

Paper https://arxiv.org/abs/2108.07610

class `anomalib.models.image.draem.lightning_model.`**`Draem`**(*enable_sspcab=False*, *sspcab_lambda=0.1*, *anomaly_source_path=None*, *beta=(0.1, 1.0)*)

Bases: `AnomalyModule`

DRÆM: A discriminatively trained reconstruction embedding for surface anomaly detection.

Parameters

- **enable_sspcab** (*bool*) – Enable SSPCAB training. Defaults to `False`.
- **sspcab_lambda** (*float*) – SSPCAB loss weight. Defaults to `0.1`.
- **anomaly_source_path** (*str | None*) – Path to folder that contains the anomaly source images. Random noise will be used if left empty. Defaults to `None`.

configure_optimizers()

Configure the Adam optimizer.

Return type
`Optimizer`

property learning_type: LearningType

Return the learning type of the model.

Returns
Learning type of the model.

Return type
LearningType

setup_sspcab()

Prepare the model for the SSPCAB training step by adding forward hooks for the SSPCAB layer activations.

Return type
`None`

property trainer_arguments: dict[str, Any]

Return DRÆM-specific trainer arguments.

training_step(*batch*, **args*, ***kwargs*)

Perform the training step of DRAEM.

Feeds the original image and the simulated anomaly image through the network and computes the training loss.

Parameters

- **batch** (*dict[str, str | torch.Tensor]*) – Batch containing image filename, image, label and mask
- **args** – Arguments.
- **kwargs** – Keyword arguments.

Return type
`Union[Tensor, Mapping[str, Any], None]`

Returns
Loss dictionary

validation_step(*batch*, **args*, ***kwargs*)

Perform the validation step of DRAEM. The Softmax predictions of the anomalous class are used as anomaly map.

Parameters

- **batch** (*dict[str, str | torch.Tensor]*) – Batch of input images
- **args** – Arguments.
- **kwargs** – Keyword arguments.

Return type
`Union[Tensor, Mapping[str, Any], None]`

Returns
Dictionary to which predicted anomaly maps have been added.

PyTorch model for the DRAEM model implementation.

`class anomalib.models.image.draem.torch_model.DraemModel(sspcab=False)`

Bases: `Module`

DRAEM PyTorch model consisting of the reconstructive and discriminative sub networks.

Parameters
sspcab (*bool*) – Enable SSPCAB training. Defaults to `False`.

`forward(batch)`

Compute the reconstruction and anomaly mask from an input image.

Parameters
batch (*torch.Tensor*) – batch of input images

Return type
`Tensor | tuple[Tensor, Tensor]`

Returns
Predicted confidence values of the anomaly mask. During training the reconstructed input images are returned as well.

Loss function for the DRAEM model implementation.

`class anomalib.models.image.draem.loss.DraemLoss`

Bases: `Module`

Overall loss function of the DRAEM model.

The total loss consists of the sum of the L2 loss and Focal loss between the reconstructed image and the input image, and the Structural Similarity loss between the predicted and GT anomaly masks.

`forward(input_image, reconstruction, anomaly_mask, prediction)`

Compute the loss over a batch for the DRAEM model.

Return type
`Tensor`

DSR

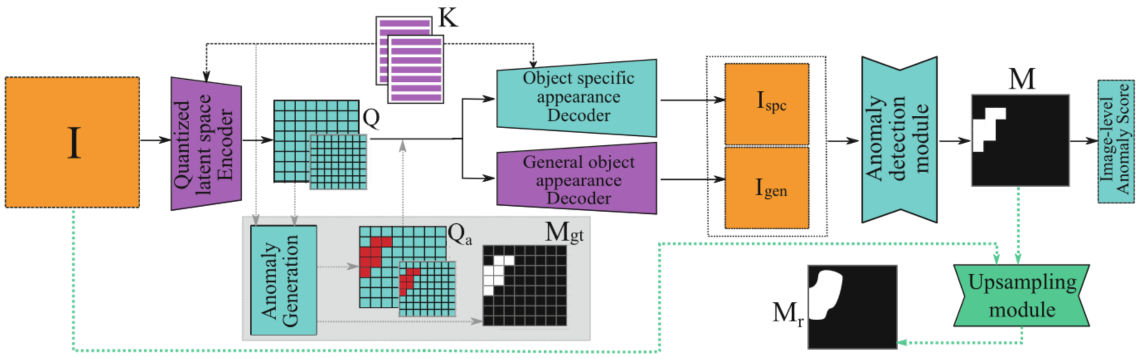
This is the implementation of the DSR paper.

Model Type: Segmentation

Description

DSR is a quantized-feature based algorithm that consists of an autoencoder with one encoder and two decoders, coupled with an anomaly detection module. DSR learns a codebook of quantized representations on ImageNet, which are then used to encode input images. These quantized representations also serve to sample near-in-distribution anomalies, since they do not rely on external datasets. Training takes place in three phases. The encoder and "general object decoder", as well as the codebook, are pretrained on ImageNet. Defects are then generated at the feature level using the codebook on the quantized representations, and are used to train the object-specific decoder as well as the anomaly detection module. In the final phase of training, the upsampling module is trained on simulated image-level smudges in order to output more robust anomaly maps.

Architecture


K
I
Quantized latent space Encoder
Q
Object specific appearance Decoder
General object appearance Decoder
I_{spc}
I_{gen}
Anomaly detection module
M
Image-level Anomaly Score
Anomaly Generation
Q_a
M_{gt}
M_r
Upsampling module

PyTorch model for the DSR model implementation.

class anomalib.models.image.dsr.torch_model.**AnomalyDetectionModule**(*in_channels*, *out_channels*, *base_width*)

Bases: `Module`

Anomaly detection module.

Module that detects the preseßnce of an anomaly by comparing two images reconstructed by the object specific decoder and the general object decoder.

Parameters

- **in_channels** (*int*) – Number of input channels.
- **out_channels** (*int*) – Number of output channels.
- **base_width** (*int*) – Base dimensionality of the layers of the autoencoder.

forward(*batch_real*, *batch_anomaly*)

Computes the anomaly map over corresponding real and anomalous images.

Parameters

- **batch_real** (*torch.Tensor*) – Batch of real, non defective images.
- **batch_anomaly** (*torch.Tensor*) – Batch of potentially anomalous images.

Return type

`Tensor`

Returns
The anomaly segmentation map.

class anomalib.models.image.dsr.torch_model.**DecoderBot**(*in_channels*, *num_hiddens*, *num_residual_layers*, *num_residual_hiddens*)

Bases: `Module`

General appearance decoder module to reconstruct images while keeping possible anomalies.

Parameters

- **in_channels** (*int*) – Number of input channels.
- **num_hiddens** (*int*) – Number of hidden channels.
- **num_residual_layers** (*int*) – Number of residual layers in residual stack.
- **num_residual_hiddens** (*int*) – Number of channels in residual layers.

forward(*inputs*)

Decode quantized feature maps into an image.

Parameters
inputs (*torch.Tensor*) – Quantized feature maps.

Return type
`Tensor`

Returns
Decoded image.

class anomalib.models.image.dsr.torch_model.**DiscreteLatentModel**(*num_hiddens*, *num_residual_layers*, *num_residual_hiddens*, *num_embeddings*, *embedding_dim*)

Bases: `Module`

Discrete Latent Model.

Autoencoder quantized model that encodes the input images into quantized feature maps and generates a reconstructed image using the general appearance decoder.

Parameters

- **num_hiddens** (*int*) – Number of hidden channels.
- **num_residual_layers** (*int*) – Number of residual layers in residual stacks.
- **num_residual_hiddens** (*int*) – Number of channels in residual layers.
- **num_embeddings** (*int*) – Size of embedding dictionary.
- **embedding_dim** (*int*) – Dimension of embeddings.

forward(*batch*, *anomaly_mask=None*, *anom_str_lo=None*, *anom_str_hi=None*)

Generate quantized feature maps.

Generates quantized feature maps of batch of input images as well as their reconstruction based on the general appearance decoder.

Parameters

- **batch** (*Tensor*) – Batch of input images.

- **anomaly_mask** (*Tensor | None*) – Anomaly mask to be used to generate anomalies on the quantized feature maps.
- **anom_str_lo** (*torch.Tensor | None*) – Strength of generated anomaly lo.
- **anom_str_hi** (*torch.Tensor | None*) – Strength of generated anomaly hi.

Returns

If generating an anomaly mask:

- General object decoder-decoded anomalous image
- Reshaped ground truth anomaly map
- Non defective quantized lo feature
- Non defective quantized hi feature
- Non quantized subspace encoded defective lo feature
- Non quantized subspace encoded defective hi feature

Else:

- General object decoder-decoded image
- Quantized lo feature
- Quantized hi feature

Return type
dict[str, torch.Tensor]

generate_fake_anomalies_joined(*features*, *embeddings*, *memory_torch_original*, *mask*, *strength*)

Generate quantized anomalies.

Parameters

- **features** (*torch.Tensor*) – Features on which the anomalies will be generated.
- **embeddings** (*torch.Tensor*) – Embeddings to use to generate the anomalies.
- **memory_torch_original** (*torch.Tensor*) – Weight of embeddings.
- **mask** (*torch.Tensor*) – Original anomaly mask.
- **strength** (*float*) – Strength of generated anomaly.

Returns
Anomalous embedding.

Return type
torch.Tensor

property vq_vae_bot: *VectorQuantizer*

Return `self._vq_vae_bot`.

property vq_vae_top: *VectorQuantizer*

Return `self._vq_vae_top`.

class anomalib.models.image.dsr.torch_model.**DsrModel**(*latent_anomaly_strength=0.2*, *embedding_dim=128*, *num_embeddings=4096*, *num_hiddens=128*, *num_residual_layers=2*, *num_residual_hiddens=64*)

Bases: `Module`

DSR PyTorch model.

Consists of the discrete latent model, image reconstruction network, subspace restriction modules, anomaly detection module and upsampling module.

Parameters

- **embedding_dim** (*int*) – Dimension of codebook embeddings.
- **num_embeddings** (*int*) – Number of embeddings.
- **latent_anomaly_strength** (*float*) – Strength of the generated anomalies in the latent space.
- **num_hiddens** (*int*) – Number of output channels in residual layers.
- **num_residual_layers** (*int*) – Number of residual layers.
- **num_residual_hiddens** (*int*) – Number of intermediate channels.

forward(*batch*, *anomaly_map_to_generate=None*)

Compute the anomaly mask from an input image.

Parameters

- **batch** (*torch.Tensor*) – Batch of input images.
- **anomaly_map_to_generate** (*torch.Tensor | None*) – anomaly map to use to generate quantized defects.
- **2** (*If not training phase*) –
- **None.** (*should be*) –

Returns

If testing:

- ”anomaly_map”: Upsampled anomaly map
- ”pred_score”: Image score

If training phase 2:

- ”recon_feat_hi”: Reconstructed non-quantized hi features of defect (F~_hi)
- ”recon_feat_lo”: Reconstructed non-quantized lo features of defect (F~_lo)
- ”embedding_bot”: Quantized features of non defective img (Q_hi)
- ”embedding_top”: Quantized features of non defective img (Q_lo)
- ”obj_spec_image”: Object-specific-decoded image (I_spc)
- ”anomaly_map”: Predicted segmentation mask (M)
- ”true_mask”: Resized ground-truth anomaly map (M_gt)

If training phase 3:

- ”anomaly_map”: Reconstructed anomaly map

Return type

dict[str, torch.Tensor]

load_pretrained_discrete_model_weights(*ckpt*)

Load pre-trained model weights.

Return type
`None`

class anomalib.models.image.dsr.torch_model.**EncoderBot**(*in_channels*, *num_hiddens*, *num_residual_layers*, *num_residual_hiddens*)

Bases: `Module`

Encoder module for bottom quantized feature maps.

Parameters

- **in_channels** (*int*) – Number of input channels.
- **num_hiddens** (*int*) – Number of hidden channels.
- **num_residual_layers** (*int*) – Number of residual layers in residual stacks.
- **num_residual_hiddens** (*int*) – Number of channels in residual layers.

forward(*batch*)

Encode inputs to be quantized into the bottom feature map.

Parameters
batch (*torch.Tensor*) – Batch of input images.

Return type
`Tensor`

Returns
Encoded feature maps.

class anomalib.models.image.dsr.torch_model.**EncoderTop**(*in_channels*, *num_hiddens*, *num_residual_layers*, *num_residual_hiddens*)

Bases: `Module`

Encoder module for top quantized feature maps.

Parameters

- **in_channels** (*int*) – Number of input channels.
- **num_hiddens** (*int*) – Number of hidden channels.
- **num_residual_layers** (*int*) – Number of residual layers in residual stacks.
- **num_residual_hiddens** (*int*) – Number of channels in residual layers.

forward(*batch*)

Encode inputs to be quantized into the top feature map.

Parameters
batch (*torch.Tensor*) – Batch of input images.

Return type
`Tensor`

Returns
Encoded feature maps.

class anomalib.models.image.dsr.torch_model.**FeatureDecoder**(*base_width*, *out_channels=1*)

Bases: `Module`

Feature decoder for the subspace restriction network.

Parameters

- **base_width** (*int*) – Base dimensionality of the layers of the autoencoder.
- **out_channels** (*int*) – Number of output channels.

forward(_, __, *b3*)

Decode a batch of latent features to a non-quantized representation.

Parameters

- **_** (*torch.Tensor*) – Top latent feature layer.
- **__** (*torch.Tensor*) – Middle latent feature layer.
- **b3** (*torch.Tensor*) – Bottom latent feature layer.

Return type

`Tensor`

Returns

Decoded non-quantized representation.

class anomalib.models.image.dsr.torch_model.**FeatureEncoder**(*in_channels*, *base_width*)

Bases: `Module`

Feature encoder for the subspace restriction network.

Parameters

- **in_channels** (*int*) – Number of input channels.
- **base_width** (*int*) – Base dimensionality of the layers of the autoencoder.

forward(*batch*)

Encode a batch of input features to the latent space.

Parameters

batch (*torch.Tensor*) – Batch of input images.

Return type

`tuple[Tensor, Tensor, Tensor]`

Returns: Encoded feature maps.

class anomalib.models.image.dsr.torch_model.**ImageReconstructionNetwork**(*in_channels*, *num_hiddens*, *num_residual_layers*, *num_residual_hiddens*)

Bases: `Module`

Image Reconstruction Network.

Image reconstruction network that reconstructs the image from a quantized representation.

Parameters

- **in_channels** (*int*) – Number of input channels.
- **num_hiddens** (*int*) – Number of output channels in residual layers.

- **num_residual_layers** (*int*) – Number of residual layers.
- **num_residual_hiddens** (*int*) – Number of intermediate channels.

forward(*inputs*)

Reconstructs an image from a quantized representation.

Parameters
inputs (*torch.Tensor*) – Quantized features.

Return type
`Tensor`

Returns
Reconstructed image.

class anomalib.models.image.dsr.torch_model.**Residual**(*in_channels*, *out_channels*, *num_residual_hiddens*)

Bases: `Module`

Residual layer.

Parameters

- **in_channels** (*int*) – Number of input channels.
- **out_channels** (*int*) – Number of output channels.
- **num_residual_hiddens** (*int*) – Number of intermediate channels.

forward(*batch*)

Compute residual layer.

Parameters
batch (*torch.Tensor*) – Batch of input images.

Return type
`Tensor`

Returns
Computed feature maps.

class anomalib.models.image.dsr.torch_model.**ResidualStack**(*in_channels*, *num_hiddens*, *num_residual_layers*, *num_residual_hiddens*)

Bases: `Module`

Stack of residual layers.

Parameters

- **in_channels** (*int*) – Number of input channels.
- **num_hiddens** (*int*) – Number of output channels in residual layers.
- **num_residual_layers** (*int*) – Number of residual layers.
- **num_residual_hiddens** (*int*) – Number of intermediate channels.

forward(*batch*)

Compute residual stack.

Parameters
batch (*torch.Tensor*) – Batch of input images.

Return type
`Tensor`

Returns
Computed feature maps.

`class anomalib.models.image.dsr.torch_model.`**`SubspaceRestrictionModule`**(*base_width*)

Bases: `Module`

Subspace Restriction Module.

Subspace restriction module that restricts the appearance subspace into configurations that agree with normal appearances and applies quantization.

Parameters
`base_width` (*`int`*) – Base dimensionality of the layers of the autoencoder.

`forward`(*batch*, *quantization*)

Generate the quantized anomaly-free representation of an anomalous image.

Parameters

- **`batch`** (*`torch.Tensor`*) – Batch of input images.
- **`quantization`** (*`function | object`*) – Quantization function.

Return type
`tuple`[`Tensor`, `Tensor`]

Returns
Reconstructed batch of non-quantized features and corresponding quantized features.

`class anomalib.models.image.dsr.torch_model.`**`SubspaceRestrictionNetwork`**(*in_channels=64*, *out_channels=64*, *base_width=64*)

Bases: `Module`

Subspace Restriction Network.

Subspace restriction network that reconstructs the input image into a non-quantized configuration that agrees with normal appearances.

Parameters

- **`in_channels`** (*`int`*) – Number of input channels.
- **`out_channels`** (*`int`*) – Number of output channels.
- **`base_width`** (*`int`*) – Base dimensionality of the layers of the autoencoder.

`forward`(*batch*)

Reconstruct non-quantized representation from batch.

Generate non-quantized feature maps from potentially anomalous images, to be quantized into non-anomalous quantized representations.

Parameters
`batch` (*`torch.Tensor`*) – Batch of input images.

Return type
`Tensor`

Returns
Reconstructed non-quantized representation.

class anomalib.models.image.dsr.torch_model.**UnetDecoder**(*base_width*, *out_channels=1*)

Bases: `Module`

Decoder of the Unet network.

Parameters

- **base_width** (*int*) – Base dimensionality of the layers of the autoencoder.
- **out_channels** (*int*) – Number of output channels.

forward(*b1*, *b2*, *b3*, *b4*)

Decodes latent represnetations into an image.

Parameters

- **b1** (*torch.Tensor*) – First (top level) quantized feature map.
- **b2** (*torch.Tensor*) – Second quantized feature map.
- **b3** (*torch.Tensor*) – Third quantized feature map.
- **b4** (*torch.Tensor*) – Fourth (bottom level) quantized feature map.

Return type

`Tensor`

Returns

Reconstructed image.

class anomalib.models.image.dsr.torch_model.**UnetEncoder**(*in_channels*, *base_width*)

Bases: `Module`

Encoder of the Unet network.

Parameters

- **in_channels** (*int*) – Number of input channels.
- **base_width** (*int*) – Base dimensionality of the layers of the autoencoder.

forward(*batch*)

Encodes batch of images into a latent representation.

Parameters

batch (*torch.Tensor*) – Quantized features.

Return type

`tuple[Tensor, Tensor, Tensor, Tensor]`

Returns

Latent representations of the input batch.

class anomalib.models.image.dsr.torch_model.**UnetModel**(*in_channels=64*, *out_channels=64*, *base_width=64*)

Bases: `Module`

Autoencoder model that reconstructs the input image.

Parameters

- **in_channels** (*int*) – Number of input channels.
- **out_channels** (*int*) – Number of output channels.
- **base_width** (*int*) – Base dimensionality of the layers of the autoencoder.

forward(*batch*)

Reconstructs an input batch of images.

Parameters
`batch` (*torch.Tensor*) – Batch of input images.

Return type
`Tensor`

Returns
Reconstructed images.

`class anomalib.models.image.dsr.torch_model.`**UpsamplingModule**(*in_channels=8*, *out_channels=2*, *base_width=64*)

Bases: `Module`

Module that upsamples the generated anomaly mask to full resolution.

Parameters

- **in_channels** (*int*) – Number of input channels.
- **out_channels** (*int*) – Number of output channels.
- **base_width** (*int*) – Base dimensionality of the layers of the autoencoder.

forward(*batch_real*, *batch_anomaly*, *batch_segmentation_map*)

Computes upsampled segmentation maps.

Parameters

- **batch_real** (*torch.Tensor*) – Batch of real, non defective images.
- **batch_anomaly** (*torch.Tensor*) – Batch of potentially anomalous images.
- **batch_segmentation_map** (*torch.Tensor*) – Batch of anomaly segmentation maps.

Return type
`Tensor`

Returns
Upsampled anomaly segmentation maps.

`class anomalib.models.image.dsr.torch_model.`**VectorQuantizer**(*num_embeddings*, *embedding_dim*)

Bases: `Module`

Module that quantizes a given feature map using learned quantization codebooks.

Parameters

- **num_embeddings** (*int*) – Size of embedding codebook.
- **embedding_dim** (*int*) – Dimension of embeddings.

property embedding: Tensor

Return embedding.

forward(*inputs*)

Calculates quantized feature map.

Parameters
`inputs` (*torch.Tensor*) – Non-quantized feature maps.

Return type
`Tensor`

Returns
Quantized feature maps.

DSR - A Dual Subspace Re-Projection Network for Surface Anomaly Detection.

Paper https://link.springer.com/chapter/10.1007/978-3-031-19821-2_31

`class anomalib.models.image.dsr.lightning_model.Dsr`(*latent_anomaly_strength=0.2*, *upsampling_train_ratio=0.7*)

Bases: `AnomalyModule`

DSR: A Dual Subspace Re-Projection Network for Surface Anomaly Detection.

Parameters

- **latent_anomaly_strength** (*float*) – Strength of the generated anomalies in the latent space. Defaults to 0.2
- **upsampling_train_ratio** (*float*) – Ratio of training steps for the upsampling module. Defaults to 0.7

`configure_optimizers()`

Configure the Adam optimizer for training phases 2 and 3.

Does not train the discrete model (phase 1)

Returns
Dictionary of optimizers

Return type
dict[str, torch.optim.Optimizer | torch.optim.lr_scheduler.LRScheduler]

`property learning_type: LearningType`

Return the learning type of the model.

Returns
Learning type of the model.

Return type
LearningType

`on_train_epoch_start()`

Display a message when starting to train the upsampling module.

Return type
`None`

`on_train_start()`

Load pretrained weights of the discrete model when starting training.

Return type
`None`

`prepare_pretrained_model()`

Download pre-trained models if they don't exist.

Return type
`Path`

`property trainer_arguments: dict[str, Any]`

Required trainer arguments.

`training_step`(*batch*)

Training Step of DSR.

Feeds the original image and the simulated anomaly mask during first phase. During second phase, feeds a generated anomalous image to train the upsampling module.

Parameters
`batch` (*dict[str, str | Tensor]*) – Batch containing image filename, image, label and mask

Returns
Loss dictionary

Return type
STEP_OUTPUT

`validation_step`(*batch*, **args*, ***kwargs*)

Validation step of DSR.

The Softmax predictions of the anomalous class are used as anomaly map.

Parameters

- **batch** (*dict[str, str | Tensor]*) – Batch of input images
- ***args** – unused
- ****kwargs** – unused

Returns
Dictionary to which predicted anomaly maps have been added.

Return type
STEP_OUTPUT

Anomaly generator for the DSR model implementation.

class `anomalib.models.image.dsr.anomaly_generator.DsrAnomalyGenerator`(*p_anomalous=0.5*)

Bases: `Module`

Anomaly generator of the DSR model.

The anomaly is generated using a Perlin noise generator on the two quantized representations of an image. This generator is only used during the second phase of training! The third phase requires generating smudges over the input images.

Parameters
`p_anomalous` (*float, optional*) – Probability to generate an anomalous image.

`augment_batch`(*batch*)

Generate anomalous augmentations for a batch of input images.

Parameters
`batch` (*Tensor*) – Batch of input images

Returns
Ground truth masks corresponding to the anomalous perturbations.

Return type
Tensor

`generate_anomaly`(*height*, *width*)

Generate an anomalous mask.

Parameters

- **height** (*int*) – Height of generated mask.
- **width** (*int*) – Width of generated mask.

Returns
Generated mask.

Return type
Tensor

Loss function for the DSR model implementation.

class anomalib.models.image.dsr.loss.**DsrSecondStageLoss**

Bases: `Module`

Overall loss function of the second training phase of the DSR model.

The total loss consists of:

- MSE loss between non-anomalous quantized input image and anomalous subspace-reconstructed non-quantized input (hi and lo)
- MSE loss between input image and reconstructed image through object-specific decoder,
- Focal loss between computed segmentation mask and ground truth mask.

forward(*recon_nq_hi*, *recon_nq_lo*, *qu_hi*, *qu_lo*, *input_image*, *gen_img*, *seg*, *anomaly_mask*)

Compute the loss over a batch for the DSR model.

Parameters

- **recon_nq_hi** (*Tensor*) – Reconstructed non-quantized hi feature
- **recon_nq_lo** (*Tensor*) – Reconstructed non-quantized lo feature
- **qu_hi** (*Tensor*) – Non-defective quantized hi feature
- **qu_lo** (*Tensor*) – Non-defective quantized lo feature
- **input_image** (*Tensor*) – Original image
- **gen_img** (*Tensor*) – Object-specific decoded image
- **seg** (*Tensor*) – Computed anomaly map
- **anomaly_mask** (*Tensor*) – Ground truth anomaly map

Returns
Total loss

Return type
Tensor

class anomalib.models.image.dsr.loss.**DsrThirdStageLoss**

Bases: `Module`

Overall loss function of the third training phase of the DSR model.

The loss consists of a focal loss between the computed segmentation mask and the ground truth mask.

forward(*pred_mask*, *true_mask*)

Compute the loss over a batch for the DSR model.

Parameters

- **pred_mask** (*Tensor*) – Computed anomaly map
- **true_mask** (*Tensor*) – Ground truth anomaly map

Returns
Total loss

Return type
Tensor

Efficient AD

EfficientAd: Accurate Visual Anomaly Detection at Millisecond-Level Latencies.

https://arxiv.org/pdf/2303.14535.pdf.

class `anomalib.models.image.efficient_ad.lightning_model.`**`EfficientAd`**(*imagenet_dir='./datasets/imagenette'*, *teacher_out_channels=384*, *model_size=EfficientAdModelSize.S*, *lr=0.0001*, *weight_decay=1e-05*, *padding=False*, *pad_maps=True*, *batch_size=1*)

Bases: `AnomalyModule`

PL Lightning Module for the EfficientAd algorithm.

Parameters

- **imagenet_dir** (*Path | str*) – directory path for the Imagenet dataset Defaults to `./datasets/imagenette`.
- **teacher_out_channels** (*int*) – number of convolution output channels Defaults to `384`.
- **model_size** (*str*) – size of student and teacher model Defaults to `EfficientAdModelSize.S`.
- **lr** (*float*) – learning rate Defaults to `0.0001`.
- **weight_decay** (*float*) – optimizer weight decay Defaults to `0.00001`.
- **padding** (*bool*) – use padding in convoluional layers Defaults to `False`.
- **pad_maps** (*bool*) – relevant if padding is set to False. In this case, pad_maps = True pads the output anomaly maps so that their size matches the size in the padding = True case. Defaults to `True`.
- **batch_size** (*int*) – batch size for imagenet dataloader Defaults to 1.

`configure_optimizers`()

Configure optimizers.

Return type
`Optimizer`

`configure_transforms`(*image_size=None*)

Default transform for Padim.

Return type
`Transform`

property learning_type: LearningType

Return the learning type of the model.

Returns
Learning type of the model.

Return type
LearningType

map_norm_quantiles(*dataloader*)

Calculate 90% and 99.5% quantiles of the student(st) and autoencoder(ae).

Parameters
dataloader (*DataLoader*) – Dataloader of the respective dataset.

Returns
Dictionary of both the 90% and 99.5% quantiles of both the student and autoencoder feature maps.

Return type
dict[str, torch.Tensor]

on_train_start()

Called before the first training epoch.

First sets up the pretrained teacher model, then prepares the imagenette data, and finally calculates or loads the channel-wise mean and std of the training dataset and push to the model.

Return type
`None`

on_validation_start()

Calculate the feature map quantiles of the validation dataset and push to the model.

Return type
`None`

prepare_imagenette_data(*image_size*)

Prepare ImageNette dataset transformations.

Parameters
image_size (*tuple[int, int] | torch.Size*) – Image size.

Return type
`None`

prepare_pretrained_model()

Prepare the pretrained teacher model.

Return type
`None`

teacher_channel_mean_std(*dataloader*)

Calculate the mean and std of the teacher models activations.

Adapted from https://math.stackexchange.com/a/2148949

Parameters
dataloader (*DataLoader*) – Dataloader of the respective dataset.

Returns
Dictionary of channel-wise mean and std

Return type
dict[str, torch.Tensor]

`property trainer_arguments: dict[str, Any]`

Return EfficientAD trainer arguments.

`training_step`(*batch*, **args*, ***kwargs*)

Perform the training step for EfficientAd returns the student, autoencoder and combined loss.

Parameters

- **(batch** (*batch*) – dict[str, str | torch.Tensor]): Batch containing image filename, image, label and mask
- **args** – Additional arguments.
- **kwargs** – Additional keyword arguments.

Return type
`dict`[`str`, `Tensor`]

Returns
Loss.

`validation_step`(*batch*, **args*, ***kwargs*)

Perform the validation step of EfficientAd returns anomaly maps for the input image batch.

Parameters

- **batch** (*dict[str, str | torch.Tensor]*) – Input batch
- **args** – Additional arguments.
- **kwargs** – Additional keyword arguments.

Return type
`Union`[`Tensor`, `Mapping`[`str`, `Any`], `None`]

Returns
Dictionary containing anomaly maps.

Torch model for student, teacher and autoencoder model in EfficientAd.

`class anomalib.models.image.efficient_ad.torch_model.EfficientAdModel`(*teacher_out_channels*, *model_size=EfficientAdModelSize.S*, *padding=False*, *pad_maps=True*)

Bases: `Module`

EfficientAd model.

Parameters

- **teacher_out_channels** (*int*) – number of convolution output channels of the pre-trained teacher model
- **model_size** (*str*) – size of student and teacher model
- **padding** (*bool*) – use padding in convoluional layers Defaults to `False`.
- **pad_maps** (*bool*) – relevant if padding is set to False. In this case, pad_maps = True pads the output anomaly maps so that their size matches the size in the padding = True case. Defaults to `True`.

choose_random_aug_image(*image*)

Choose a random augmentation function and apply it to the input image.

Parameters
image (*torch.Tensor*) – Input image.

Returns
Augmented image.

Return type
Tensor

forward(*batch*, *batch_imagenet=None*, *normalize=True*)

Perform the forward-pass of the EfficientAd models.

Parameters

- **batch** (*torch.Tensor*) – Input images.
- **batch_imagenet** (*torch.Tensor*) – ImageNet batch. Defaults to None.
- **normalize** (*bool*) – Normalize anomaly maps or not

Returns
Predictions

Return type
Tensor

is_set(*p_dic*)

Check if any of the parameters in the parameter dictionary is set.

Parameters
p_dic (*nn.ParameterDict*) – Parameter dictionary.

Returns
Boolean indicating whether any of the parameters in the parameter dictionary is set.

Return type
bool

FastFlow

FastFlow Lightning Model Implementation.

https://arxiv.org/abs/2111.07677

class anomalib.models.image.fastflow.lightning_model.**Fastflow**(*backbone='resnet18'*, *pre_trained=True*, *flow_steps=8*, *conv3x3_only=False*, *hidden_ratio=1.0*)

Bases: `AnomalyModule`

PL Lightning Module for the FastFlow algorithm.

Parameters

- **backbone** (*str*) – Backbone CNN network Defaults to `resnet18`.
- **pre_trained** (*bool, optional*) – Boolean to check whether to use a pre_trained backbone. Defaults to `True`.

- **flow_steps** (*int, optional*) – Flow steps. Defaults to 8.
- **conv3x3_only** (*bool, optinoal*) – Use only conv3x3 in fast_flow model. Defaults to `False`.
- **hidden_ratio** (*float, optional*) – Ratio to calculate hidden var channels. Defaults to ``1.0`.

configure_optimizers()

Configure optimizers for each decoder.

Returns
Adam optimizer for each decoder

Return type
Optimizer

property learning_type: LearningType

Return the learning type of the model.

Returns
Learning type of the model.

Return type
LearningType

property trainer_arguments: dict[str, Any]

Return FastFlow trainer arguments.

training_step(*batch*, **args*, ***kwargs*)

Perform the training step input and return the loss.

Parameters

- **(batch** (*batch*) – dict[str, str | torch.Tensor]): Input batch
- **args** – Additional arguments.
- **kwargs** – Additional keyword arguments.

Returns
Dictionary containing the loss value.

Return type
STEP_OUTPUT

validation_step(*batch*, **args*, ***kwargs*)

Perform the validation step and return the anomaly map.

Parameters

- **batch** (*dict[str, str | torch.Tensor]*) – Input batch
- **args** – Additional arguments.
- **kwargs** – Additional keyword arguments.

Returns
batch dictionary containing anomaly-maps.

Return type
STEP_OUTPUT | None

FastFlow Torch Model Implementation.

class anomalib.models.image.fastflow.torch_model.**FastflowModel**(*input_size*, *backbone*, *pre_trained=True*, *flow_steps=8*, *conv3x3_only=False*, *hidden_ratio=1.0*)

Bases: `Module`

FastFlow.

Unsupervised Anomaly Detection and Localization via 2D Normalizing Flows.

Parameters

- **input_size** (*tuple[int, int]*) – Model input size.
- **backbone** (*str*) – Backbone CNN network
- **pre_trained** (*bool, optional*) – Boolean to check whether to use a pre_trained backbone. Defaults to `True`.
- **flow_steps** (*int, optional*) – Flow steps. Defaults to `8`.
- **conv3x3_only** (*bool, optinoal*) – Use only conv3x3 in fast_flow model. Defaults to `False`.
- **hidden_ratio** (*float, optional*) – Ratio to calculate hidden var channels. Defaults to `1.0`.

Raises

ValueError – When the backbone is not supported.

forward(*input_tensor*)

Forward-Pass the input to the FastFlow Model.

Parameters

input_tensor (*torch.Tensor*) – Input tensor.

Returns

During training, return

(hidden_variables, log-of-the-jacobian-determinants). During the validation/test, return the anomaly map.

Return type

Tensor | list[torch.Tensor] | tuple[list[torch.Tensor]]

Loss function for the FastFlow Model Implementation.

class anomalib.models.image.fastflow.loss.**FastflowLoss**(**args*, ***kwargs*)

Bases: `Module`

FastFlow Loss.

forward(*hidden_variables*, *jacobians*)

Calculate the Fastflow loss.

Parameters

- **hidden_variables** (*list[torch.Tensor]*) – Hidden variables from the fastflow model. f: X -> Z
- **jacobians** (*list[torch.Tensor]*) – Log of the jacobian determinants from the fastflow model.

Returns
Fastflow loss computed based on the hidden variables and the log of the Jacobians.

Return type
Tensor

FastFlow Anomaly Map Generator Implementation.

`class anomalib.models.image.fastflow.anomaly_map.`**`AnomalyMapGenerator`**(*input_size*)

Bases: `Module`

Generate Anomaly Heatmap.

Parameters
`input_size` (*`ListConfig | tuple`*) – Input size.

`forward`(*hidden_variables*)

Generate Anomaly Heatmap.

This implementation generates the heatmap based on the flow maps computed from the normalizing flow (NF) FastFlow blocks. Each block yields a flow map, which overall is stacked and averaged to an anomaly map.

Parameters
`hidden_variables` (*`list[torch.Tensor]`*) – List of hidden variables from each NF FastFlow block.

Returns
Anomaly Map.

Return type
Tensor

GANomaly

GANomaly: Semi-Supervised Anomaly Detection via Adversarial Training.

https://arxiv.org/abs/1805.06725

`class anomalib.models.image.ganomaly.lightning_model.`**`Ganomaly`**(*batch_size=32*, *n_features=64*, *latent_vec_size=100*, *extra_layers=0*, *add_final_conv_layer=True*, *wadv=1*, *wcon=50*, *wenc=1*, *lr=0.0002*, *beta1=0.5*, *beta2=0.999*)

Bases: `AnomalyModule`

PL Lightning Module for the GANomaly Algorithm.

Parameters

- **`batch_size`** (*`int`*) – Batch size. Defaults to `32`.
- **`n_features`** (*`int`*) – Number of features layers in the CNNs. Defaults to `64`.
- **`latent_vec_size`** (*`int`*) – Size of autoencoder latent vector. Defaults to `100`.
- **`extra_layers`** (*`int, optional`*) – Number of extra layers for encoder/decoder. Defaults to `0`.

- **add_final_conv_layer** (*bool, optional*) – Add convolution layer at the end. Defaults to `True`.
- **wadv** (*int, optional*) – Weight for adversarial loss. Defaults to 1.
- **wcon** (*int, optional*) – Image regeneration weight. Defaults to `50`.
- **wenc** (*int, optional*) – Latent vector encoder weight. Defaults to 1.
- **lr** (*float, optional*) – Learning rate. Defaults to `0.0002`.
- **beta1** (*float, optional*) – Adam beta1. Defaults to `0.5`.
- **beta2** (*float, optional*) – Adam beta2. Defaults to `0.999`.

configure_optimizers()

Configure optimizers for each decoder.

Returns
Adam optimizer for each decoder

Return type
Optimizer

property learning_type: LearningType

Return the learning type of the model.

Returns
Learning type of the model.

Return type
LearningType

on_test_batch_end(*outputs*, *batch*, *batch_idx*, *dataloader_idx=0*)

Normalize outputs based on min/max values.

Return type
`None`

on_test_start()

Reset min max values before test batch starts.

Return type
`None`

on_validation_batch_end(*outputs*, *batch*, *batch_idx*, *dataloader_idx=0*)

Normalize outputs based on min/max values.

Return type
`None`

on_validation_start()

Reset min and max values for current validation epoch.

Return type
`None`

test_step(*batch*, *batch_idx*, **args*, ***kwargs*)

Update min and max scores from the current step.

Return type
`Union[Tensor, Mapping[str, Any], None]`

property trainer_arguments: dict[str, Any]

Return GANomaly trainer arguments.

training_step(*batch*, *batch_idx*)

Perform the training step.

Parameters

- **batch** (*dict[str, str | torch.Tensor]*) – Input batch containing images.
- **batch_idx** (*int*) – Batch index.
- **optimizer_idx** (*int*) – Optimizer which is being called for current training step.

Returns
Loss

Return type
STEP_OUTPUT

validation_step(*batch*, **args*, ***kwargs*)

Update min and max scores from the current step.

Parameters

- **batch** (*dict[str, str | torch.Tensor]*) – Predicted difference between z and z_hat.
- **args** – Additional arguments.
- **kwargs** – Additional keyword arguments.

Returns
Output predictions.

Return type
(STEP_OUTPUT)

Torch models defining encoder, decoder, Generator and Discriminator.

Code adapted from https://github.com/samet-akcay/ganomaly.

class anomalib.models.image.ganomaly.torch_model.**GanomalyModel**(*input_size*, *num_input_channels*, *n_features*, *latent_vec_size*, *extra_layers=0*, *add_final_conv_layer=True*)

Bases: `Module`

Ganomaly Model.

Parameters

- **input_size** (*tuple[int, int]*) – Input dimension.
- **num_input_channels** (*int*) – Number of input channels.
- **n_features** (*int*) – Number of features layers in the CNNs.
- **latent_vec_size** (*int*) – Size of autoencoder latent vector.
- **extra_layers** (*int, optional*) – Number of extra layers for encoder/decoder. Defaults to `0`.
- **add_final_conv_layer** (*bool, optional*) – Add convolution layer at the end. Defaults to `True`.

`forward`(*batch*)

Get scores for batch.

Parameters
`batch` (*torch.Tensor*) – Images

Returns
Regeneration scores.

Return type
Tensor

`static weights_init`(*module*)

Initialize DCGAN weights.

Parameters
`module` (*nn.Module*) – [description]

Return type
`None`

Loss function for the GANomaly Model Implementation.

`class anomalib.models.image.ganomaly.loss.DiscriminatorLoss`

Bases: `Module`

Discriminator loss for the GANomaly model.

`forward`(*pred_real*, *pred_fake*)

Compute the loss for a predicted batch.

Parameters

- `pred_real` (*torch.Tensor*) – Discriminator predictions for the real image.
- `pred_fake` (*torch.Tensor*) – Discriminator predictions for the fake image.

Returns
The computed discriminator loss.

Return type
Tensor

`class anomalib.models.image.ganomaly.loss.GeneratorLoss`(*wadv=1*, *wcon=50*, *wenc=1*)

Bases: `Module`

Generator loss for the GANomaly model.

Parameters

- `wadv` (*int, optional*) – Weight for adversarial loss. Defaults to 1.
- `wcon` (*int, optional*) – Image regeneration weight. Defaults to `50`.
- `wenc` (*int, optional*) – Latent vector encoder weight. Defaults to 1.

`forward`(*latent_i*, *latent_o*, *images*, *fake*, *pred_real*, *pred_fake*)

Compute the loss for a batch.

Parameters

- `latent_i` (*torch.Tensor*) – Latent features of the first encoder.
- `latent_o` (*torch.Tensor*) – Latent features of the second encoder.
- `images` (*torch.Tensor*) – Real image that served as input of the generator.

- **fake** (*torch.Tensor*) – Generated image.
- **pred_real** (*torch.Tensor*) – Discriminator predictions for the real image.
- **pred_fake** (*torch.Tensor*) – Discriminator predictions for the fake image.

Returns
The computed generator loss.

Return type
Tensor

Padim

PaDiM: a Patch Distribution Modeling Framework for Anomaly Detection and Localization.

Paper https://arxiv.org/abs/2011.08785

`class anomalib.models.image.padim.lightning_model.`**`Padim`**`(`*backbone='resnet18', layers=['layer1', 'layer2', 'layer3'], pre_trained=True, n_features=None*`)`

Bases: `MemoryBankMixin`, `AnomalyModule`

PaDiM: a Patch Distribution Modeling Framework for Anomaly Detection and Localization.

Parameters

- **backbone** (*str*) – Backbone CNN network Defaults to `resnet18`.
- **layers** (*list[str]*) – Layers to extract features from the backbone CNN Defaults to `["layer1", "layer2", "layer3"]`.
- **pre_trained** (*bool, optional*) – Boolean to check whether to use a pre_trained backbone. Defaults to `True`.
- **n_features** (*int, optional*) – Number of features to retain in the dimension reduction step. Default values from the paper are available for: resnet18 (100), wide_resnet50_2 (550). Defaults to `None`.

`static` **`configure_optimizers`**`()`

PADIM doesn't require optimization, therefore returns no optimizers.

Return type
`None`

`configure_transforms``(`*image_size=None*`)`

Default transform for Padim.

Return type
`Transform`

`fit``()`

Fit a Gaussian to the embedding collected from the training set.

Return type
`None`

`property` **`learning_type: LearningType`**

Return the learning type of the model.

Returns
Learning type of the model.

Return type
LearningType

`property trainer_arguments: dict[str, int | float]`

Return PADIM trainer arguments.

Since the model does not require training, we limit the max_epochs to 1. Since we need to run training epoch before validation, we also set the sanity steps to 0

`training_step`(*batch*, **args*, ***kwargs*)

Perform the training step of PADIM. For each batch, hierarchical features are extracted from the CNN.

Parameters

- `batch` (*dict[str, str | torch.Tensor]*) – Batch containing image filename, image, label and mask
- `args` – Additional arguments.
- `kwargs` – Additional keyword arguments.

Return type
`None`

Returns
Hierarchical feature map

`validation_step`(*batch*, **args*, ***kwargs*)

Perform a validation step of PADIM.

Similar to the training step, hierarchical features are extracted from the CNN for each batch.

Parameters

- `batch` (*dict[str, str | torch.Tensor]*) – Input batch
- `args` – Additional arguments.
- `kwargs` – Additional keyword arguments.

Return type
`Union`[`Tensor`, `Mapping`[`str`, `Any`], `None`]

Returns
Dictionary containing images, features, true labels and masks. These are required in *validation_epoch_end* for feature concatenation.

PyTorch model for the PaDiM model implementation.

`class anomalib.models.image.padim.torch_model.PadimModel`(*layers*, *backbone='resnet18'*, *pre_trained=True*, *n_features=None*)

Bases: `Module`

Padim Module.

Parameters

- `layers` (*list[str]*) – Layers used for feature extraction
- `backbone` (*str, optional*) – Pre-trained model backbone. Defaults to "resnet18". Defaults to `resnet18`.
- `pre_trained` (*bool, optional*) – Boolean to check whether to use a pre_trained backbone. Defaults to `True`.

- **n_features** (*int, optional*) – Number of features to retain in the dimension reduction step. Default values from the paper are available for: resnet18 (100), wide_resnet50_2 (550). Defaults to None.

forward(*input_tensor*)

Forward-pass image-batch (N, C, H, W) into model to extract features.

Parameters

- **input_tensor** (Tensor) – Image-batch (N, C, H, W)
- **input_tensor** – torch.Tensor:

Return type
Tensor

Returns
Features from single/multiple layers.

Example

```
>>> x = torch.randn(32, 3, 224, 224)
>>> features = self.extract_features(input_tensor)
>>> features.keys()
dict_keys(['layer1', 'layer2', 'layer3'])
```

```
>>> [v.shape for v in features.values()]
[torch.Size([32, 64, 56, 56]),
torch.Size([32, 128, 28, 28]),
torch.Size([32, 256, 14, 14])]
```

generate_embedding(*features*)

Generate embedding from hierarchical feature map.

Parameters
features (*dict[str, torch.Tensor]*) – Hierarchical feature map from a CNN (ResNet18 or WideResnet)

Return type
Tensor

Returns
Embedding vector

PatchCore

Towards Total Recall in Industrial Anomaly Detection.

Paper https://arxiv.org/abs/2106.08265.

class anomalib.models.image.patchcore.lightning_model.**Patchcore**(*backbone='wide_resnet50_2', layers=('layer2', 'layer3'), pre_trained=True, coreset_sampling_ratio=0.1, num_neighbors=9*)

Bases: `MemoryBankMixin`, `AnomalyModule`

PatchcoreLightning Module to train PatchCore algorithm.

Parameters

- **`backbone`** (*`str`*) – Backbone CNN network Defaults to `wide_resnet50_2`.
- **`layers`** (*`list[str]`*) – Layers to extract features from the backbone CNN Defaults to `["layer2", "layer3"]`.
- **`pre_trained`** (*`bool, optional`*) – Boolean to check whether to use a pre_trained backbone. Defaults to `True`.
- **`coreset_sampling_ratio`** (*`float, optional`*) – Coreset sampling ratio to subsample embedding. Defaults to `0.1`.
- **`num_neighbors`** (*`int, optional`*) – Number of nearest neighbors. Defaults to `9`.

`configure_optimizers()`

Configure optimizers.

Returns
Do not set optimizers by returning None.

Return type
None

`configure_transforms`(*image_size=None*)

Default transform for Padim.

Return type
`Transform`

`fit()`

Apply subsampling to the embedding collected from the training set.

Return type
`None`

`property learning_type: LearningType`

Return the learning type of the model.

Returns
Learning type of the model.

Return type
LearningType

`property trainer_arguments: dict[str, Any]`

Return Patchcore trainer arguments.

`training_step`(*batch*, **args*, ***kwargs*)

Generate feature embedding of the batch.

Parameters

- **`batch`** (*`dict[str, str | torch.Tensor]`*) – Batch containing image filename, image, label and mask
- **`args`** – Additional arguments.
- **`kwargs`** – Additional keyword arguments.

Returns
Embedding Vector

Return type
dict[str, np.ndarray]

`validation_step`(*batch*, **args*, ***kwargs*)

Get batch of anomaly maps from input image batch.

Parameters

- **batch** (*dict[str, str | torch.Tensor]*) – Batch containing image filename, image, label and mask
- **args** – Additional arguments.
- **kwargs** – Additional keyword arguments.

Returns
Image filenames, test images, GT and predicted label/masks

Return type
dict[str, Any]

PyTorch model for the PatchCore model implementation.

`class anomalib.models.image.patchcore.torch_model.PatchcoreModel`(*layers*, *backbone='wide_resnet50_2'*, *pre_trained=True*, *num_neighbors=9*)

Bases: `DynamicBufferMixin`, `Module`

Patchcore Module.

Parameters

- **layers** (*list[str]*) – Layers used for feature extraction
- **backbone** (*str, optional*) – Pre-trained model backbone. Defaults to `resnet18`.
- **pre_trained** (*bool, optional*) – Boolean to check whether to use a pre_trained backbone. Defaults to `True`.
- **num_neighbors** (*int, optional*) – Number of nearest neighbors. Defaults to `9`.

`compute_anomaly_score`(*patch_scores*, *locations*, *embedding*)

Compute Image-Level Anomaly Score.

Parameters

- **patch_scores** (*torch.Tensor*) – Patch-level anomaly scores
- **locations** (`Tensor`) – Memory bank locations of the nearest neighbor for each patch location
- **embedding** (`Tensor`) – The feature embeddings that generated the patch scores

Returns
Image-level anomaly scores

Return type
Tensor

static `euclidean_dist`(*x*, *y*)

Calculate pair-wise distance between row vectors in x and those in y.

Replaces torch cdist with p=2, as cdist is not properly exported to onnx and openvino format. Resulting matrix is indexed by x vectors in rows and y vectors in columns.

Parameters

- **x** (`Tensor`) – input tensor 1
- **y** (`Tensor`) – input tensor 2

Return type
`Tensor`

Returns
Matrix of distances between row vectors in x and y.

`forward`(*input_tensor*)

Return Embedding during training, or a tuple of anomaly map and anomaly score during testing.

Steps performed: 1. Get features from a CNN. 2. Generate embedding based on the features. 3. Compute anomaly map in test mode.

Parameters
input_tensor (*torch.Tensor*) – Input tensor

Returns
Embedding for training, anomaly map and anomaly score for testing.

Return type
Tensor | dict[str, torch.Tensor]

`generate_embedding`(*features*)

Generate embedding from hierarchical feature map.

Parameters

- **features** (`dict[str, Tensor]`) – Hierarchical feature map from a CNN (ResNet18 or WideResnet)
- **features** – dict[str:Tensor]:

Return type
`Tensor`

Returns
Embedding vector

`nearest_neighbors`(*embedding*, *n_neighbors*)

Nearest Neighbours using brute force method and euclidean norm.

Parameters

- **embedding** (*torch.Tensor*) – Features to compare the distance with the memory bank.
- **n_neighbors** (*int*) – Number of neighbors to look at

Returns
Patch scores. Tensor: Locations of the nearest neighbor(s).

Return type
Tensor

static reshape_embedding(*embedding*)

Reshape Embedding.

Reshapes Embedding to the following format:

- [Batch, Embedding, Patch, Patch] to [Batch*Patch*Patch, Embedding]

Parameters
embedding (*torch.Tensor*) – Embedding tensor extracted from CNN features.

Returns
Reshaped embedding tensor.

Return type
Tensor

subsample_embedding(*embedding*, *sampling_ratio*)

Subsample embedding based on coreset sampling and store to memory.

Parameters

- **embedding** (*np.ndarray*) – Embedding tensor from the CNN
- **sampling_ratio** (*float*) – Coreset sampling ratio

Return type
`None`

Reverse Distillation

Anomaly Detection via Reverse Distillation from One-Class Embedding.

https://arxiv.org/abs/2201.10703v2

class anomalib.models.image.reverse_distillation.lightning_model.**ReverseDistillation**(*backbone='wide_resnet*
lay-
ers=('layer1',
'layer2',
'layer3'),
anomaly_map_mode=A
pre_trained=True)

Bases: `AnomalyModule`

PL Lightning Module for Reverse Distillation Algorithm.

Parameters

- **backbone** (*str*) – Backbone of CNN network Defaults to `wide_resnet50_2`.
- **layers** (*list[str]*) – Layers to extract features from the backbone CNN Defaults to `["layer1", "layer2", "layer3"]`.
- **anomaly_map_mode** (*AnomalyMapGenerationMode, optional*) – Mode to generate anomaly map. Defaults to `AnomalyMapGenerationMode.ADD`.
- **pre_trained** (*bool, optional*) – Boolean to check whether to use a pre_trained backbone. Defaults to `True`.

`configure_optimizers()`

Configure optimizers for decoder and bottleneck.

Returns
Adam optimizer for each decoder

Return type
Optimizer

`property learning_type: LearningType`

Return the learning type of the model.

Returns
Learning type of the model.

Return type
LearningType

`property trainer_arguments: dict[str, Any]`

Return Reverse Distillation trainer arguments.

`training_step`(*batch*, **args*, ***kwargs*)

Perform a training step of Reverse Distillation Model.

Features are extracted from three layers of the Encoder model. These are passed to the bottleneck layer that are passed to the decoder network. The loss is then calculated based on the cosine similarity between the encoder and decoder features.

Parameters

- **(batch** (*batch*) – dict[str, str | torch.Tensor]): Input batch
- **args** – Additional arguments.
- **kwargs** – Additional keyword arguments.

Return type
`Union[Tensor, Mapping[str, Any], None]`

Returns
Feature Map

`validation_step`(*batch*, **args*, ***kwargs*)

Perform a validation step of Reverse Distillation Model.

Similar to the training step, encoder/decoder features are extracted from the CNN for each batch, and anomaly map is computed.

Parameters

- **batch** (*dict[str, str | torch.Tensor]*) – Input batch
- **args** – Additional arguments.
- **kwargs** – Additional keyword arguments.

Return type
`Union[Tensor, Mapping[str, Any], None]`

Returns
Dictionary containing images, anomaly maps, true labels and masks. These are required in *validation_epoch_end* for feature concatenation.

PyTorch model for Reverse Distillation.

class anomalib.models.image.reverse_distillation.torch_model.**ReverseDistillationModel**(*backbone*, *input_size*, *layers*, *anomaly_map_mode*, *pre_trained=True*)

Bases: `Module`

Reverse Distillation Model.

To reproduce results in the paper, use torchvision model for the encoder:
self.encoder = torchvision.models.wide_resnet50_2(pretrained=True)

Parameters

- **backbone** (*str*) – Name of the backbone used for encoder and decoder.
- **input_size** (*tuple[int, int]*) – Size of input image.
- **layers** (*list[str]*) – Name of layers from which the features are extracted.
- **anomaly_map_mode** (*str*) – Mode used to generate anomaly map. Options are between `multiply` and `add`.
- **pre_trained** (*bool, optional*) – Boolean to check whether to use a pre_trained backbone. Defaults to `True`.

forward(*images*)

Forward-pass images to the network.

During the training mode the model extracts features from encoder and decoder networks. During evaluation mode, it returns the predicted anomaly map.

Parameters
images (*torch.Tensor*) – Batch of images

Returns

Encoder and decoder features
in training mode, else anomaly maps.

Return type
torch.Tensor | list[torch.Tensor] | tuple[list[torch.Tensor]]

Loss function for Reverse Distillation.

class anomalib.models.image.reverse_distillation.loss.**ReverseDistillationLoss**(**args*, ***kwargs*)

Bases: `Module`

Loss function for Reverse Distillation.

forward(*encoder_features*, *decoder_features*)

Compute cosine similarity loss based on features from encoder and decoder.

Based on the official code: https://github.com/hq-deng/RD4AD/blob/6554076872c65f8784f6ece8cfb39ce77e1aee12/main.py#L33C25-L33C25 Calculates loss from flattened arrays of features, see https://github.com/hq-deng/RD4AD/issues/22

Parameters

- **encoder_features** (*list[torch.Tensor]*) – List of features extracted from encoder
- **decoder_features** (*list[torch.Tensor]*) – List of features extracted from decoder

Returns
Cosine similarity loss

Return type
Tensor

Compute Anomaly map.

`class anomalib.models.image.reverse_distillation.anomaly_map.`**`AnomalyMapGenerationMode`**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `str`, `Enum`

Type of mode when generating anomaly imape.

`class anomalib.models.image.reverse_distillation.anomaly_map.`**`AnomalyMapGenerator`**(*image_size*, *sigma=4*, *mode=AnomalyMapGenerati*

Bases: `Module`

Generate Anomaly Heatmap.

Parameters

- **image_size** (*ListConfig, tuple*) – Size of original image used for upscaling the anomaly map.
- **sigma** (*int*) – Standard deviation of the gaussian kernel used to smooth anomaly map. Defaults to 4.
- **mode** (AnomalyMapGenerationMode, *optional*) – Operation used to generate anomaly map. Options are `AnomalyMapGenerationMode.ADD` and `AnomalyMapGenerationMode.MULTIPLY`. Defaults to `AnomalyMapGenerationMode.MULTIPLY`.

Raises
ValueError – In case modes other than multiply and add are passed.

forward(*student_features*, *teacher_features*)
Compute anomaly map given encoder and decoder features.

Parameters

- **student_features** (*list[torch.Tensor]*) – List of encoder features
- **teacher_features** (*list[torch.Tensor]*) – List of decoder features

Returns
Anomaly maps of length batch.

Return type
Tensor

R-KDE

Region Based Anomaly Detection With Real-Time Training and Analysis.

https://ieeexplore.ieee.org/abstract/document/8999287

class `anomalib.models.image.rkde.lightning_model.`**`Rkde`**(*roi_stage=RoiStage.RCNN*, *roi_score_threshold=0.001*, *min_box_size=25*, *iou_threshold=0.3*, *max_detections_per_image=100*, *n_pca_components=16*, *feature_scaling_method=FeatureScalingMethod.SCALE*, *max_training_points=40000*)

Bases: `MemoryBankMixin`, `AnomalyModule`

Region Based Anomaly Detection With Real-Time Training and Analysis.

Parameters

- **roi_stage** (`RoiStage`, *optional*) – Processing stage from which rois are extracted. Defaults to `RoiStage.RCNN`.
- **roi_score_threshold** (*float, optional*) – Mimumum confidence score for the region proposals. Defaults to `0.001`.
- **min_size** (*int, optional*) – Minimum size in pixels for the region proposals. Defaults to `25`.
- **iou_threshold** (*float, optional*) – Intersection-Over-Union threshold used during NMS. Defaults to `0.3`.
- **max_detections_per_image** (*int, optional*) – Maximum number of region proposals per image. Defaults to `100`.
- **n_pca_components** (*int, optional*) – Number of PCA components. Defaults to `16`.
- **feature_scaling_method** (`FeatureScalingMethod`, *optional*) – Scaling method applied to features before passing to KDE. Options are *norm* (normalize to unit vector length) and *scale* (scale to max length observed in training). Defaults to `FeatureScalingMethod.SCALE`.
- **max_training_points** (*int, optional*) – Maximum number of training points to fit the KDE model. Defaults to `40000`.

static **`configure_optimizers`**()

RKDE doesn't require optimization, therefore returns no optimizers.

Return type
`None`

`fit`()

Fit a KDE Model to the embedding collected from the training set.

Return type
`None`

property learning_type: LearningType

Return the learning type of the model.

Returns
Learning type of the model.

Return type
LearningType

property trainer_arguments: dict[str, Any]

Return R-KDE trainer arguments.

Returns
Arguments for the trainer.

Return type
dict[str, Any]

training_step(*batch*, **args*, ***kwargs*)

Perform a training Step of RKDE. For each batch, features are extracted from the CNN.

Parameters

- **batch** (*dict[str, str | torch.Tensor]*) – Batch containing image filename, image, label and mask
- **args** – Additional arguments.
- **kwargs** – Additional keyword arguments.

Return type
`None`

Returns
Deep CNN features.

validation_step(*batch*, **args*, ***kwargs*)

Perform a validation Step of RKde.

Similar to the training step, features are extracted from the CNN for each batch.

Parameters

- **batch** (*dict[str, str | torch.Tensor]*) – Batch containing image filename, image, label and mask
- **args** – Additional arguments.
- **kwargs** – Additional keyword arguments.

Return type
`Union`[`Tensor`, `Mapping`[`str`, `Any`], `None`]

Returns
Dictionary containing probability, prediction and ground truth values.

Torch model for region-based anomaly detection.

class anomalib.models.image.rkde.torch_model.**RkdeModel**(*roi_stage=RoiStage.RCNN*, *roi_score_threshold=0.001*, *min_box_size=25*, *iou_threshold=0.3*, *max_detections_per_image=100*, *n_pca_components=16*, *feature_scaling_method=FeatureScalingMethod.SCALE*, *max_training_points=40000*)

Bases: `Module`

Torch Model for the Region-based Anomaly Detection Model.

Parameters

- **roi_stage** (`RoiStage`, *optional*) – Processing stage from which rois are extracted. Defaults to `RoiStage.RCNN`.
- **roi_score_threshold** (*float*, *optional*) – Mimumum confidence score for the region proposals. Defaults to `0.001`.
- **min_size** (*int*, *optional*) – Minimum size in pixels for the region proposals. Defaults to `25`.
- **iou_threshold** (*float*, *optional*) – Intersection-Over-Union threshold used during NMS. Defaults to `0.3`.
- **max_detections_per_image** (*int*, *optional*) – Maximum number of region proposals per image. Defaults to `100`.
- **n_pca_components** (*int*, *optional*) – Number of PCA components. Defaults to `16`.
- **feature_scaling_method** (`FeatureScalingMethod`, *optional*) – Scaling method applied to features before passing to KDE. Options are *norm* (normalize to unit vector length) and *scale* (scale to max length observed in training). Defaults to `FeatureScalingMethod.SCALE`.
- **max_training_points** (*int*, *optional*) – Maximum number of training points to fit the KDE model. Defaults to `40000`.

`fit`(*embeddings*)

Fit the model using a set of collected embeddings.

Parameters
embeddings (*torch.Tensor*) – Input embeddings to fit the model.

Return type
`bool`

Returns
Boolean confirming whether the training is successful.

`forward`(*batch*)

Prediction by normality model.

Parameters
batch (*torch.Tensor*) – Input images.

Returns

The extracted features (when in training mode),
or the predicted rois and corresponding anomaly scores.

Return type
Tensor | tuple[torch.Tensor, torch.Tensor]

Region-based Anomaly Detection with Real Time Training and Analysis.

Feature Extractor.

class anomalib.models.image.rkde.feature_extractor.**FeatureExtractor**

Bases: `Module`

Feature Extractor module for Region-based anomaly detection.

forward(*batch*, *rois*)

Perform a forward pass of the feature extractor.

Parameters

- **batch** (*torch.Tensor*) – Batch of input images of shape [B, C, H, W].
- **rois** (*torch.Tensor*) – torch.Tensor of shape [N, 5] describing the regions-of-interest in the batch.

Returns
torch.Tensor containing a 4096-dimensional feature vector for every RoI location.

Return type
Tensor

Region-based Anomaly Detection with Real Time Training and Analysis.

Region Extractor.

class anomalib.models.image.rkde.region_extractor.**RegionExtractor**(*stage=RoiStage.RCNN*, *score_threshold=0.001*, *min_size=25*, *iou_threshold=0.3*, *max_detections_per_image=100*)

Bases: `Module`

Extracts regions from the image.

Parameters

- **stage** (RoiStage, *optional*) – Processing stage from which rois are extracted. Defaults to `RoiStage.RCNN`.
- **score_threshold** (*float, optional*) – Mimumum confidence score for the region proposals. Defaults to `0.001`.
- **min_size** (*int, optional*) – Minimum size in pixels for the region proposals. Defaults to `25`.
- **iou_threshold** (*float, optional*) – Intersection-Over-Union threshold used during NMS. Defaults to `0.3`.
- **max_detections_per_image** (*int, optional*) – Maximum number of region proposals per image. Defaults to `100`.

forward(*batch*)

Forward pass of the model.

Parameters
batch (*torch.Tensor*) – Batch of input images of shape [B, C, H, W].

Raises
ValueError – When `stage` is not one of `rcnn` or `rpn`.

Returns

Predicted regions, tensor of shape [N, 5] where N is the number of predicted regions in the batch,
and where each row describes the index of the image in the batch and the 4 bounding box coordinates.

Return type
Tensor

post_process_box_predictions(*pred_boxes*, *pred_scores*)

Post-processes the box predictions.

The post-processing consists of removing small boxes, applying nms, and keeping only the k boxes with the highest confidence score.

Parameters

- **pred_boxes** (*torch.Tensor*) – Box predictions of shape (N, 4).
- **pred_scores** (*torch.Tensor*) – torch.Tensor of shape () with a confidence score for each box prediction.

Returns
Post-processed box predictions of shape (N, 4).

Return type
list[torch.Tensor]

class anomalib.models.image.rkde.region_extractor.**RoiStage**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `str`, `Enum`

Processing stage from which rois are extracted.

STFPM

STFPM: Student-Teacher Feature Pyramid Matching for Unsupervised Anomaly Detection.

https://arxiv.org/abs/2103.04257

class anomalib.models.image.stfpm.lightning_model.**Stfpm**(*backbone='resnet18'*, *layers=('layer1', 'layer2', 'layer3')*)

Bases: `AnomalyModule`

PL Lightning Module for the STFPM algorithm.

Parameters

- **backbone** (*str*) – Backbone CNN network Defaults to `resnet18`.
- **layers** (*list[str]*) – Layers to extract features from the backbone CNN Defaults to `["layer1", "layer2", "layer3"]`.

configure_optimizers()

Configure optimizers.

Returns
SGD optimizer

Return type
Optimizer

property learning_type: LearningType

Return the learning type of the model.

Returns
Learning type of the model.

Return type
LearningType

property trainer_arguments: dict[str, Any]

Required trainer arguments.

training_step(*batch*, **args*, ***kwargs*)

Perform a training step of STFPM.

For each batch, teacher and student and teacher features are extracted from the CNN.

Parameters

- **batch** (*dict[str, str | torch.Tensor]*) – Input batch.
- **args** – Additional arguments.
- **kwargs** – Additional keyword arguments.

Return type
`Union[Tensor, Mapping[str, Any], None]`

Returns
Loss value

validation_step(*batch*, **args*, ***kwargs*)

Perform a validation Step of STFPM.

Similar to the training step, student/teacher features are extracted from the CNN for each batch, and anomaly map is computed.

Parameters

- **batch** (*dict[str, str | torch.Tensor]*) – Input batch
- **args** – Additional arguments
- **kwargs** – Additional keyword arguments

Return type
`Union[Tensor, Mapping[str, Any], None]`

Returns
Dictionary containing images, anomaly maps, true labels and masks. These are required in *validation_epoch_end* for feature concatenation.

PyTorch model for the STFPM model implementation.

class anomalib.models.image.stfpm.torch_model.**STFPMModel**(*layers*, *backbone='resnet18'*)

Bases: `Module`

STFPM: Student-Teacher Feature Pyramid Matching for Unsupervised Anomaly Detection.

Parameters

- **layers** (*list[str]*) – Layers used for feature extraction.
- **backbone** (*str, optional*) – Pre-trained model backbone. Defaults to `resnet18`.

forward(*images*)

Forward-pass images into the network.

During the training mode the model extracts the features from the teacher and student networks. During the evaluation mode, it returns the predicted anomaly map.

Parameters
images (*torch.Tensor*) – Batch of images.

Return type
`Tensor | dict[str, Tensor] | tuple[dict[str, Tensor]]`

Returns
Teacher and student features when in training mode, otherwise the predicted anomaly maps.

Loss function for the STFPM Model Implementation.

class anomalib.models.image.stfpm.loss.**STFPMLoss**

Bases: `Module`

Feature Pyramid Loss This class implmenents the feature pyramid loss function proposed in STFPM paper.

Example

```
>>> from anomalib.models.components.feature_extractors import TimmFeatureExtractor
>>> from anomalib.models.stfpm.loss import STFPMLoss
>>> from torchvision.models import resnet18
```

```
>>> layers = ['layer1', 'layer2', 'layer3']
>>> teacher_model = TimmFeatureExtractor(model=resnet18(pretrained=True),␣
↪layers=layers)
>>> student_model = TimmFeatureExtractor(model=resnet18(pretrained=False),␣
↪layers=layers)
>>> loss = Loss()
```

```
>>> inp = torch.rand((4, 3, 256, 256))
>>> teacher_features = teacher_model(inp)
>>> student_features = student_model(inp)
>>> loss(student_features, teacher_features)
    tensor(51.2015, grad_fn=<SumBackward0>)
```

compute_layer_loss(*teacher_feats*, *student_feats*)

Compute layer loss based on Equation (1) in Section 3.2 of the paper.

Parameters

- **teacher_feats** (*torch.Tensor*) – Teacher features
- **student_feats** (*torch.Tensor*) – Student features

Return type
`Tensor`

Returns
L2 distance between teacher and student features.

forward(*teacher_features*, *student_features*)

Compute the overall loss via the weighted average of the layer losses computed by the cosine similarity.

Parameters

- **teacher_features** (*dict[str, torch.Tensor]*) – Teacher features
- **student_features** (*dict[str, torch.Tensor]*) – Student features

Return type
`Tensor`

Returns
Total loss, which is the weighted average of the layer losses.

Anomaly Map Generator for the STFPM model implementation.

class anomalib.models.image.stfpm.anomaly_map.**AnomalyMapGenerator**

Bases: `Module`

Generate Anomaly Heatmap.

compute_anomaly_map(*teacher_features*, *student_features*, *image_size*)

Compute the overall anomaly map via element-wise production the interpolated anomaly maps.

Parameters

- **teacher_features** (*dict[str, torch.Tensor]*) – Teacher features
- **student_features** (*dict[str, torch.Tensor]*) – Student features
- **image_size** (*tuple[int, int]*) – Image size to which the anomaly map should be resized.

Return type
`Tensor`

Returns
Final anomaly map

compute_layer_map(*teacher_features*, *student_features*, *image_size*)

Compute the layer map based on cosine similarity.

Parameters

- **teacher_features** (*torch.Tensor*) – Teacher features
- **student_features** (*torch.Tensor*) – Student features
- **image_size** (*tuple[int, int]*) – Image size to which the anomaly map should be resized.

Return type
`Tensor`

Returns
Anomaly score based on cosine similarity.

forward(***kwargs*)

Return anomaly map.

Expects *teach_features* and *student_features* keywords to be passed explicitly.

Parameters
kwargs (*dict[str, torch.Tensor]*) – Keyword arguments

Example

```
>>> anomaly_map_generator = AnomalyMapGenerator(image_size=tuple(hparams.model.
↪input_size))
>>> output = self.anomaly_map_generator(
        teacher_features=teacher_features,
        student_features=student_features
    )
```

Raises
`ValueError` – *teach_features* and *student_features* keys are not found

Returns
anomaly map

Return type
torch.Tensor

U-Flow

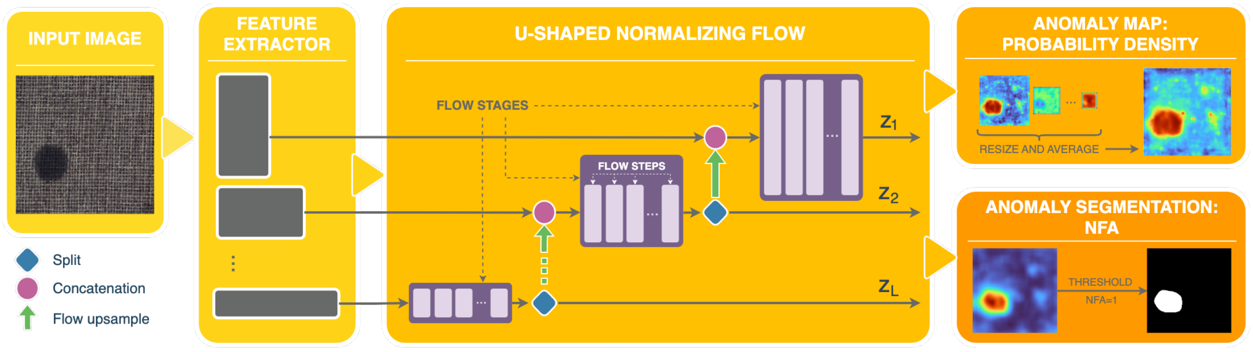
This is the implementation of the U-Flow paper.

Model Type: Segmentation

Description

U-Flow is a U-Shaped normalizing flow-based probability distribution estimator. The method consists of three phases. (1) Multi-scale feature extraction: a rich multi-scale representation is obtained with MSCaiT, by combining pre-trained image Transformers acting at different image scales. It can also be used any other feature extractor, such as ResNet. (2) U-shaped Normalizing Flow: by adapting the widely used U-like architecture to NFs, a fully invertible architecture is designed. This architecture is capable of merging the information from different scales while ensuring independence both intra- and inter-scales. To make it fully invertible, split and invertible up-sampling operations are used. (3) Anomaly score and segmentation computation: besides generating the anomaly map based on the likelihood of test data, we also propose to adapt the a contrario framework to obtain an automatic threshold by controlling the allowed number of false alarms.

Architecture



U-Flow torch model.

class anomalib.models.image.uflow.torch_model.**AffineCouplingSubnet**(*kernel_size*, *subnet_channels_ratio*)

Bases: `object`

Class for building the Affine Coupling subnet.

It is passed as an argument to the *AllInOneBlock* module.

Parameters

- **kernel_size** (*int*) – Kernel size.
- **subnet_channels_ratio** (*float*) – Subnet channels ratio.

class anomalib.models.image.uflow.torch_model.**UflowModel**(*input_size=(448, 448)*, *flow_steps=4*, *backbone='mcait'*, *affine_clamp=2.0*, *affine_subnet_channels_ratio=1.0*, *permute_soft=False*)

Bases: `Module`

U-Flow model.

Parameters

- **input_size** (*tuple[int, int]*) – Input image size.
- **flow_steps** (*int*) – Number of flow steps.
- **backbone** (*str*) – Backbone name.
- **affine_clamp** (*float*) – Affine clamp.
- **affine_subnet_channels_ratio** (*float*) – Affine subnet channels ratio.
- **permute_soft** (*bool*) – Whether to use soft permutation.

build_flow(*flow_steps*)

Build the flow model.

First we start with the input nodes, which have to match the feature extractor output. Then, we build the U-Shaped flow. Starting from the bottom (the coarsest scale), the flow is built as follows:

1. Pass the input through a Flow Stage (*build_flow_stage*).
2. Split the output of the flow stage into two parts, one that goes directly to the output,
3. and the other is up-sampled, and will be concatenated with the output of the next flow stage (next scale)
4. Repeat steps 1-3 for the next scale.

Finally, we build the Flow graph using the input nodes, the flow stages, and the output nodes.

Parameters
flow_steps (*int*) – Number of flow steps.

Returns
Flow model.

Return type
ff.GraphINN

build_flow_stage(*in_node*, *flow_steps*, *condition_node=None*)

Build a flow stage, which is a sequence of flow steps.

Each flow stage is essentially a sequence of *flow_steps* Glow blocks (*AllInOneBlock*).

Parameters

- **in_node** (*ff.Node*) – Input node.
- **flow_steps** (*int*) – Number of flow steps.
- **condition_node** (*ff.Node*) – Condition node.

Returns
List of flow steps.

Return type
List[ff.Node]

encode(*features*)

Return

Return type
tuple[Tensor, Tensor]

forward(*image*)

Return anomaly map.

Return type
Tensor

U-Flow: A U-shaped Normalizing Flow for Anomaly Detection with Unsupervised Threshold.

https://arxiv.org/pdf/2211.12353.pdf

class anomalib.models.image.uflow.lightning_model.**Uflow**(*backbone='mcait'*, *flow_steps=4*, *affine_clamp=2.0*, *affine_subnet_channels_ratio=1.0*, *permute_soft=False*)

Bases: AnomalyModule

PL Lightning Module for the UFLOW algorithm.

configure_optimizers()

Return optimizer and scheduler.

Return type
tuple[list[LightningOptimizer], list[LRScheduler]]

configure_transforms(*image_size=None*)

Default transform for Padim.

Return type
Transform

property learning_type: LearningType

Return the learning type of the model.

Returns
Learning type of the model.

Return type
LearningType

property trainer_arguments: dict[str, Any]

Return EfficientAD trainer arguments.

`training_step`(*batch*, **args*, ***kwargs*)

Training step.

Return type
`Union[Tensor, Mapping[str, Any], None]`

`validation_step`(*batch*, **args*, ***kwargs*)

Validation step.

Return type
`Union[Tensor, Mapping[str, Any], None]`

UFlow Anomaly Map Generator Implementation.

`class anomalib.models.image.uflow.anomaly_map.AnomalyMapGenerator`(*input_size*)

Bases: `Module`

Generate Anomaly Heatmap and segmentation.

`static binomial_test`(*z*, *window_size*, *probability_thr*, *high_precision=False*)

The binomial test applied to validate or reject the null hypothesis that the pixel is normal.

The null hypothesis is that the pixel is normal, and the alternative hypothesis is that the pixel is anomalous. The binomial test is applied to a window around the pixel, and the number of pixels in the window that ares anomalous is compared to the number of pixels that are expected to be anomalous under the null hypothesis.

Parameters

- **z** (`Tensor`) – Latent variable from the UFlow model. Tensor of shape (N, Cl, Hl, Wl), where N is the batch size, Cl is
- **channels** (*the number of*) –
- **variables** (*and Hl and Wl are the height and width of the latent*) –
- **respectively.** –
- **window_size** (*int*) – Window size for the binomial test.
- **probability_thr** (`float`) – Probability threshold for the binomial test.
- **high_precision** (`bool`) – Whether to use high precision for the binomial test.

Return type
`Tensor`

Returns
Log of the probability of the null hypothesis.

`compute_anomaly_map`(*latent_variables*)

Generate a likelihood-based anomaly map, from latent variables.

Parameters

- **latent_variables** (`list[Tensor]`) – List of latent variables from the UFlow model. Each element is a tensor of shape
- **(N** –
- **Cl** –
- **Hl** –
- **Wl)** –
- **size** (*where N is the batch*) –

- **channels** (*Cl is the number of*) –
- **and** (*and Hl and Wl are the height*) –
- **variables** (*width of the latent*) –
- **respectively** –
- **l.** (*for each scale*) –

Return type
Tensor

Returns
Final Anomaly Map. Tensor of shape (N, 1, H, W), where N is the batch size, and H and W are the height and width of the input image, respectively.

compute_anomaly_mask(*z*, *window_size=7*, *binomial_probability_thr=0.5*, *high_precision=False*)

This method is not used in the basic functionality of training and testing.

It is a bit slow, so we decided to leave it as an option for the user. It is included as it is part of the U-Flow paper, and can be called separately if an unsupervised anomaly segmentation is needed.

Generate an anomaly mask, from latent variables. It is based on the NFA (Number of False Alarms) method, which is a statistical method to detect anomalies. The NFA is computed as the log of the probability of the null hypothesis, which is that all pixels are normal. First, we compute a list of candidate pixels, with suspiciously high values of z^2, by applying a binomial test to each pixel, looking at a window around it. Then, to compute the NFA values (actually the log-NFA), we evaluate how probable is that a pixel belongs to the normal distribution. The null-hypothesis is that under normality assumptions, all candidate pixels are uniformly distributed. Then, the detection is based on the concentration of candidate pixels.

Parameters

- **z** (*list[torch.Tensor]*) – List of latent variables from the UFlow model. Each element is a tensor of shape (N, Cl, Hl, Wl), where N is the batch size, Cl is the number of channels, and Hl and Wl are the height and width of the latent variables, respectively, for each scale l.
- **window_size** (*int*) – Window size for the binomial test. Defaults to 7.
- **binomial_probability_thr** (*float*) – Probability threshold for the binomial test. Defaults to 0.5
- **high_precision** (*bool*) – Whether to use high precision for the binomial test. Defaults to False.

Return type
Tensor

Returns
Anomaly mask. Tensor of shape (N, 1, H, W), where N is the batch size, and H and W are the height and width of the input image, respectively.

forward(*latent_variables*)

Return anomaly map.

Return type
Tensor

WinCLIP

WinCLIP: Zero-/Few-Shot Anomaly Classification and Segmentation.

Paper https://arxiv.org/abs/2303.14814

`class anomalib.models.image.winclip.lightning_model.WinClip`(*class_name=None*, *k_shot=0*, *scales=(2, 3)*, *few_shot_source=None*)

Bases: `AnomalyModule`

WinCLIP Lightning model.

Parameters

- **class_name** (*str, optional*) – The name of the object class used in the prompt ensemble. Defaults to `None`.
- **k_shot** (*int*) – The number of reference images for few-shot inference. Defaults to `0`.
- **scales** (*tuple[int], optional*) – The scales of the sliding windows used for multiscale anomaly detection. Defaults to `(2, 3)`.
- **few_shot_source** (*str | Path, optional*) – Path to a folder of reference images used for few-shot inference. Defaults to `None`.

`collect_reference_images`(*dataloader*)

Collect reference images for few-shot inference.

The reference images are collected by iterating the training dataset until the required number of images are collected.

Returns
A tensor containing the reference images.

Return type
ref_images (Tensor)

`static configure_optimizers()`

WinCLIP doesn't require optimization, therefore returns no optimizers.

Return type
`None`

`configure_transforms`(*image_size=None*)

Configure the default transforms used by the model.

Return type
`Transform`

`property learning_type: LearningType`

The learning type of the model.

WinCLIP is a zero-/few-shot model, depending on the user configuration. Therefore, the learning type is set to `LearningType.FEW_SHOT` when `k_shot` is greater than zero and `LearningType.ZERO_SHOT` otherwise.

`load_state_dict`(*state_dict*, *strict=True*)

Load the state dict of the model.

Before loading the state dict, we restore the parameters of the frozen backbone to ensure that the model is loaded correctly. We also restore the auxiliary objects like threshold classes and normalization metrics.

Return type
`Any`

`state_dict()`

Return the state dict of the model.

Before returning the state dict, we remove the parameters of the frozen backbone to reduce the size of the checkpoint.

Return type
`OrderedDict[str, Any]`

`property trainer_arguments: dict[str, int | float]`

Set model-specific trainer arguments.

`validation_step`(*batch*, **args*, ***kwargs*)

Validation Step of WinCLIP.

Return type
`dict`

PyTorch model for the WinCLIP implementation.

`class anomalib.models.image.winclip.torch_model.WinClipModel`(*class_name=None*, *reference_images=None*, *scales=(2, 3)*, *apply_transform=False*)

Bases: `DynamicBufferMixin`, `BufferListMixin`, `Module`

PyTorch module that implements the WinClip model for image anomaly detection.

Parameters

- **class_name** (*str, optional*) – The name of the object class used in the prompt ensemble. Defaults to `None`.
- **reference_images** (*torch.Tensor, optional*) – Tensor of shape `(K, C, H, W)` containing the reference images. Defaults to `None`.
- **scales** (*tuple[int], optional*) – The scales of the sliding windows used for multi-scale anomaly detection. Defaults to `(2, 3)`.
- **apply_transform** (*bool, optional*) – Whether to apply the default CLIP transform to the input images. Defaults to `False`.

`clip`

The CLIP model used for image and text encoding.

Type
CLIP

`grid_size`

The size of the feature map grid.

Type
tuple[int]

`k_shot`

The number of reference images used for few-shot anomaly detection.

Type
int

`scales`

The scales of the sliding windows used for multi-scale anomaly detection.

Type
tuple[int]

`masks`

The masks representing the sliding window locations.

Type
list[torch.Tensor] | None

`_text_embeddings`

The text embeddings for the compositional prompt ensemble.

Type
torch.Tensor | None

`_visual_embeddings`

The multi-scale embeddings for the reference images.

Type
list[torch.Tensor] | None

`_patch_embeddings`

The patch embeddings for the reference images.

Type
torch.Tensor | None

`encode_image`(*batch*)

Encode the batch of images to obtain image embeddings, window embeddings, and patch embeddings.

The image embeddings and patch embeddings are obtained by passing the batch of images through the model. The window embeddings are obtained by masking the feature map and passing it through the transformer. A forward hook is used to retrieve the intermediate feature map and share computation between the image and window embeddings.

Parameters
batch (*torch.Tensor*) – Batch of input images of shape `(N, C, H, W)`.

Returns
A tuple containing the image embeddings, window embeddings, and patch embeddings respectively.

Return type
Tuple[torch.Tensor, List[torch.Tensor], torch.Tensor]

Examples

```
>>> model = WinClipModel()
>>> model.prepare_masks()
>>> batch = torch.rand((1, 3, 240, 240))
>>> image_embeddings, window_embeddings, patch_embeddings = model.encode_
↪image(batch)
>>> image_embeddings.shape
torch.Size([1, 640])
>>> [embedding.shape for embedding in window_embeddings]
```

(continues on next page)

(continued from previous page)

```
[torch.Size([1, 196, 640]), torch.Size([1, 169, 640])]
>>> patch_embeddings.shape
torch.Size([1, 225, 896])
```

forward(*batch*)

Forward-pass through the model to obtain image and pixel scores.

Parameters
batch (*torch.Tensor*) – Batch of input images of shape `(batch_size, C, H, W)`.

Returns
Tuple containing the image scores and pixel scores.

Return type
Tuple[torch.Tensor, torch.Tensor]

property patch_embeddings: Tensor

The patch embeddings used by the model.

setup(*class_name=None*, *reference_images=None*)

Setup WinCLIP.

WinCLIP's setup stage consists of collecting the text and visual embeddings used during inference. The following steps are performed, depending on the arguments passed to the model: - Collect text embeddings for zero-shot inference. - Collect reference images for few-shot inference. The k_shot attribute is updated based on the number of reference images.

The setup method is called internally by the constructor. However, it can also be called manually to update the text and visual embeddings after the model has been initialized.

Parameters

- **class_name** (*str*) – The name of the object class used in the prompt ensemble.
- **reference_images** (*torch.Tensor*) – Tensor of shape `(batch_size, C, H, W)` containing the reference images.

Return type
None

Examples

```
>>> model = WinClipModel()
>>> model.setup("transistor")
>>> model.text_embeddings.shape
torch.Size([2, 640])
```

```
>>> ref_images = torch.rand(2, 3, 240, 240)
>>> model = WinClipModel()
>>> model.setup("transistor", ref_images)
>>> model.k_shot
2
>>> model.visual_embeddings[0].shape
torch.Size([2, 196, 640])
```

```
>>> model = WinClipModel("transistor")
>>> model.k_shot
0
>>> model.setup(reference_images=ref_images)
>>> model.k_shot
2
```

```
>>> model = WinClipModel(class_name="transistor", reference_images=ref_images)
>>> model.text_embeddings.shape
torch.Size([2, 640])
>>> model.visual_embeddings[0].shape
torch.Size([2, 196, 640])
```

property text_embeddings: Tensor

The text embeddings used by the model.

property transform: Compose

The transform used by the model.

To obtain the transforms, we retrieve the transforms from the clip backbone. Since the original transforms are intended for PIL images, we prepend a ToPILImage transform to the list of transforms.

property visual_embeddings: list[Tensor]

The visual embeddings used by the model.

Video Models

AI VAD

AI VAD

Attribute-based Representations for Accurate and Interpretable Video Anomaly Detection.

Paper https://arxiv.org/pdf/2212.00789.pdf

class anomalib.models.video.ai_vad.lightning_model.**AiVad**(*box_score_thresh=0.7*, *persons_only=False*, *min_bbox_area=100*, *max_bbox_overlap=0.65*, *enable_foreground_detections=True*, *foreground_kernel_size=3*, *foreground_binary_threshold=18*, *n_velocity_bins=1*, *use_velocity_features=True*, *use_pose_features=True*, *use_deep_features=True*, *n_components_velocity=2*, *n_neighbors_pose=1*, *n_neighbors_deep=1*)

Bases: `MemoryBankMixin`, `AnomalyModule`

AI-VAD: Attribute-based Representations for Accurate and Interpretable Video Anomaly Detection.

Parameters

- **box_score_thresh** (*float*) – Confidence threshold for bounding box predictions. Defaults to `0.7`.
- **persons_only** (*bool*) – When enabled, only regions labeled as person are included. Defaults to `False`.
- **min_bbox_area** (*int*) – Minimum bounding box area. Regions with a surface area lower than this value are excluded. Defaults to `100`.
- **max_bbox_overlap** (*float*) – Maximum allowed overlap between bounding boxes. Defaults to `0.65`.
- **enable_foreground_detections** (*bool*) – Add additional foreground detections based on pixel difference between consecutive frames. Defaults to `True`.
- **foreground_kernel_size** (*int*) – Gaussian kernel size used in foreground detection. Defaults to `3`.
- **foreground_binary_threshold** (*int*) – Value between 0 and 255 which acts as binary threshold in foreground detection. Defaults to `18`.
- **n_velocity_bins** (*int*) – Number of discrete bins used for velocity histogram features. Defaults to `1`.
- **use_velocity_features** (*bool*) – Flag indicating if velocity features should be used. Defaults to `True`.
- **use_pose_features** (*bool*) – Flag indicating if pose features should be used. Defaults to `True`.
- **use_deep_features** (*bool*) – Flag indicating if deep features should be used. Defaults to `True`.
- **n_components_velocity** (*int*) – Number of components used by GMM density estimation for velocity features. Defaults to `2`.
- **n_neighbors_pose** (*int*) – Number of neighbors used in KNN density estimation for pose features. Defaults to `1`.
- **n_neighbors_deep** (*int*) – Number of neighbors used in KNN density estimation for deep features. Defaults to `1`.

`static configure_optimizers()`

AI-VAD training does not involve fine-tuning of NN weights, no optimizers needed.

Return type
`None`

`configure_transforms(`*image_size=None*`)`

AI-VAD does not need a transform, as the region- and feature-extractors apply their own transforms.

Return type
`Transform | None`

`fit()`

Fit the density estimators to the extracted features from the training set.

Return type
`None`

`property learning_type: LearningType`

Return the learning type of the model.

Returns
Learning type of the model.

Return type
LearningType

`property trainer_arguments: dict[str, Any]`

AI-VAD specific trainer arguments.

`training_step`(*batch*)

Training Step of AI-VAD.

Extract features from the batch of clips and update the density estimators.

Parameters
`batch` (`dict[str, str | torch.Tensor]`) – Batch containing image filename, image, label and mask

Return type
`None`

`validation_step`(*batch*, **args*, ***kwargs*)

Perform the validation step of AI-VAD.

Extract boxes and box scores..

Parameters

- `batch` (`dict[str, str | torch.Tensor]`) – Input batch
- `*args` – Arguments.
- `**kwargs` – Keyword arguments.

Return type
`Union`[`Tensor`, `Mapping`[`str`, `Any`], `None`]

Returns
Batch dictionary with added boxes and box scores.

PyTorch model for AI-VAD model implementation.

Paper https://arxiv.org/pdf/2212.00789.pdf

`class anomalib.models.video.ai_vad.torch_model.AiVadModel`(*box_score_thresh=0.8*, *persons_only=False*, *min_bbox_area=100*, *max_bbox_overlap=0.65*, *enable_foreground_detections=True*, *foreground_kernel_size=3*, *foreground_binary_threshold=18*, *n_velocity_bins=8*, *use_velocity_features=True*, *use_pose_features=True*, *use_deep_features=True*, *n_components_velocity=5*, *n_neighbors_pose=1*, *n_neighbors_deep=1*)

Bases: `Module`

AI-VAD model.

Parameters

- **box_score_thresh** (*float*) – Confidence threshold for region extraction stage. Defaults to `0.8`.
- **persons_only** (*bool*) – When enabled, only regions labeled as person are included. Defaults to `False`.
- **min_bbox_area** (*int*) – Minimum bounding box area. Regions with a surface area lower than this value are excluded. Defaults to `100`.
- **max_bbox_overlap** (*float*) – Maximum allowed overlap between bounding boxes. Defaults to `0.65`.
- **enable_foreground_detections** (*bool*) – Add additional foreground detections based on pixel difference between consecutive frames. Defaults to `True`.
- **foreground_kernel_size** (*int*) – Gaussian kernel size used in foreground detection. Defaults to `3`.
- **foreground_binary_threshold** (*int*) – Value between 0 and 255 which acts as binary threshold in foreground detection. Defaults to `18`.
- **n_velocity_bins** (*int*) – Number of discrete bins used for velocity histogram features. Defaults to `8`.
- **use_velocity_features** (*bool*) – Flag indicating if velocity features should be used. Defaults to `True`.
- **use_pose_features** (*bool*) – Flag indicating if pose features should be used. Defaults to `True`.
- **use_deep_features** (*bool*) – Flag indicating if deep features should be used. Defaults to `True`.
- **n_components_velocity** (*int*) – Number of components used by GMM density estimation for velocity features. Defaults to `5`.
- **n_neighbors_pose** (*int*) – Number of neighbors used in KNN density estimation for pose features. Defaults to `1`.
- **n_neighbors_deep** (*int*) – Number of neighbors used in KNN density estimation for deep features. Defaults to `1`.

forward(*batch*)

Forward pass through AI-VAD model.

Parameters
batch (*torch.Tensor*) – Input image of shape (N, L, C, H, W)

Returns
List of bbox locations for each image. list[torch.Tensor]: List of per-bbox anomaly scores for each image. list[torch.Tensor]: List of per-image anomaly scores.

Return type
list[torch.Tensor]

Feature extraction module for AI-VAD model implementation.

`class anomalib.models.video.ai_vad.features.`**DeepExtractor**

Bases: `Module`

Deep feature extractor.

Extracts the deep (appearance) features from the input regions.

`forward`(*batch*, *boxes*, *batch_size*)

Extract deep features using CLIP encoder.

Parameters

- **batch** (*torch.Tensor*) – Batch of RGB input images of shape (N, 3, H, W)
- **boxes** (*torch.Tensor*) – Bounding box coordinates of shaspe (M, 5). First column indicates batch index of the bbox.
- **batch_size** (*int*) – Number of images in the batch.

Returns
Deep feature tensor of shape (M, 512)

Return type
Tensor

class anomalib.models.video.ai_vad.features.**FeatureExtractor**(*n_velocity_bins=8*, *use_velocity_features=True*, *use_pose_features=True*, *use_deep_features=True*)

Bases: `Module`

Feature extractor for AI-VAD.

Parameters

- **n_velocity_bins** (*int*) – Number of discrete bins used for velocity histogram features. Defaults to `8`.
- **use_velocity_features** (*bool*) – Flag indicating if velocity features should be used. Defaults to `True`.
- **use_pose_features** (*bool*) – Flag indicating if pose features should be used. Defaults to `True`.
- **use_deep_features** (*bool*) – Flag indicating if deep features should be used. Defaults to `True`.

`forward`(*rgb_batch*, *flow_batch*, *regions*)

Forward pass through the feature extractor.

Extract any combination of velocity, pose and deep features depending on configuration.

Parameters

- **rgb_batch** (*torch.Tensor*) – Batch of RGB images of shape (N, 3, H, W)
- **flow_batch** (*torch.Tensor*) – Batch of optical flow images of shape (N, 2, H, W)
- **regions** (*list[dict]*) – Region information per image in batch.

Returns
Feature dictionary per image in batch.

Return type
list[dict]

class anomalib.models.video.ai_vad.features.**FeatureType**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `str`, `Enum`

Names of the different feature streams used in AI-VAD.

`class anomalib.models.video.ai_vad.features.`**`PoseExtractor`**`(`**args*, ***kwargs*`)`

Bases: `Module`

Pose feature extractor.

Extracts pose features based on estimated body landmark keypoints.

`forward``(`*batch*, *boxes*`)`

Extract pose features using a human keypoint estimation model.

Parameters

- **batch** (*torch.Tensor*) – Batch of RGB input images of shape (N, 3, H, W)
- **boxes** (*torch.Tensor*) – Bounding box coordinates of shaspe (M, 5). First column indicates batch index of the bbox.

Returns

list of pose feature tensors for each image.

Return type

list[torch.Tensor]

`class anomalib.models.video.ai_vad.features.`**`VelocityExtractor`**`(`*n_bins=8*`)`

Bases: `Module`

Velocity feature extractor.

Extracts histograms of optical flow magnitude and direction.

Parameters

n_bins (*int*) – Number of direction bins used for the feature histograms.

`forward``(`*flows*, *boxes*`)`

Extract velocioty features by filling a histogram.

Parameters

- **flows** (*torch.Tensor*) – Batch of optical flow images of shape (N, 2, H, W)
- **boxes** (*torch.Tensor*) – Bounding box coordinates of shaspe (M, 5). First column indicates batch index of the bbox.

Returns

Velocity feature tensor of shape (M, n_bins)

Return type

Tensor

Regions extraction module of AI-VAD model implementation.

`class anomalib.models.video.ai_vad.regions.`**`RegionExtractor`**`(`*box_score_thresh=0.8*, *persons_only=False*, *min_bbox_area=100*, *max_bbox_overlap=0.65*, *enable_foreground_detections=True*, *foreground_kernel_size=3*, *foreground_binary_threshold=18*`)`

Bases: `Module`

Region extractor for AI-VAD.

Parameters

- **box_score_thresh** (*float*) – Confidence threshold for bounding box predictions. Defaults to `0.8`.
- **persons_only** (*bool*) – When enabled, only regions labeled as person are included. Defaults to `False`.
- **min_bbox_area** (*int*) – Minimum bounding box area. Regions with a surface area lower than this value are excluded. Defaults to `100`.
- **max_bbox_overlap** (*float*) – Maximum allowed overlap between bounding boxes. Defaults to `0.65`.
- **enable_foreground_detections** (*bool*) – Add additional foreground detections based on pixel difference between consecutive frames. Defaults to `True`.
- **foreground_kernel_size** (*int*) – Gaussian kernel size used in foreground detection. Defaults to `3`.
- **foreground_binary_threshold** (*int*) – Value between 0 and 255 which acts as binary threshold in foreground detection. Defaults to `18`.

`add_foreground_boxes`(*regions*, *first_frame*, *last_frame*, *kernel_size*, *binary_threshold*)

Add any foreground regions that were not detected by the region extractor.

This method adds regions that likely belong to the foreground of the video scene, but were not detected by the region extractor module. The foreground pixels are determined by taking the pixel difference between two consecutive video frames and applying a binary threshold. The final detections consist of all connected components in the foreground that do not fall in one of the bounding boxes predicted by the region extractor.

Parameters

- **regions** (*list[dict[str, torch.Tensor]]*) – Region detections for a batch of images, generated by the region extraction module.
- **first_frame** (*torch.Tensor*) – video frame at time t-1
- **last_frame** (*torch.Tensor*) – Video frame time t
- **kernel_size** (*int*) – Kernel size for Gaussian smoothing applied to input frames
- **binary_threshold** (*int*) – Binary threshold used in foreground detection, should be in range [0, 255]

Returns

region detections with foreground regions appended

Return type

list[dict[str, torch.Tensor]]

`forward`(*first_frame*, *last_frame*)

Perform forward-pass through region extractor.

Parameters

- **first_frame** (*torch.Tensor*) – Batch of input images of shape (N, C, H, W) forming the first frames in the clip.
- **last_frame** (*torch.Tensor*) – Batch of input images of shape (N, C, H, W) forming the last frame in the clip.

Returns

List of Mask RCNN predictions for each image in the batch.

Return type
list[dict]

`post_process_bbox_detections`(*regions*)

Post-process the region detections.

The region detections are filtered based on class label, bbox area and overlap with other regions.

Parameters
regions (*list[dict[str, torch.Tensor]]*) – Region detections for a batch of images, generated by the region extraction module.

Returns
Filtered regions

Return type
list[dict[str, torch.Tensor]]

`static subsample_regions`(*regions*, *indices*)

Subsample the items in a region dictionary based on a Tensor of indices.

Parameters

- **regions** (*dict[str, torch.Tensor]*) – Region detections for a single image in the batch.
- **indices** (*torch.Tensor*) – Indices of region detections that should be kept.

Returns
Subsampled region detections.

Return type
dict[str, torch.Tensor]

Optical Flow extraction module for AI-VAD implementation.

`class anomalib.models.video.ai_vad.flow.FlowExtractor`(**args*, ***kwargs*)

Bases: `Module`

Optical Flow extractor.

Computes the pixel displacement between 2 consecutive frames from a video clip.

`forward`(*first_frame*, *last_frame*)

Forward pass through the flow extractor.

Parameters

- **first_frame** (*torch.Tensor*) – Batch of starting frames of shape (N, 3, H, W).
- **last_frame** (*torch.Tensor*) – Batch of last frames of shape (N, 3, H, W).

Returns
Estimated optical flow map of shape (N, 2, H, W).

Return type
Tensor

`pre_process`(*first_frame*, *last_frame*)

Resize inputs to dimensions required by backbone.

Parameters

- **first_frame** (*torch.Tensor*) – Starting frame of optical flow computation.

- **last_frame** (*torch.Tensor*) – Last frame of optical flow computation.

Returns
Preprocessed first and last frame.

Return type
tuple[torch.Tensor, torch.Tensor]

Density estimation module for AI-VAD model implementation.

class anomalib.models.video.ai_vad.density.**BaseDensityEstimator**(**args*, ***kwargs*)

Bases: `Module`, `ABC`

Base density estimator.

abstract **fit**()

Compose model using collected features.

Return type
`None`

forward(*features*)

Update or predict depending on training status.

Return type
`Tensor` | `tuple[Tensor, Tensor]` | `None`

abstract **predict**(*features*)

Predict the density of a set of features.

Return type
`Tensor` | `tuple[Tensor, Tensor]`

abstract **update**(*features*, *group=None*)

Update the density model with a new set of features.

Return type
`None`

class anomalib.models.video.ai_vad.density.**CombinedDensityEstimator**(*use_pose_features=True*, *use_deep_features=True*, *use_velocity_features=False*, *n_neighbors_pose=1*, *n_neighbors_deep=1*, *n_components_velocity=5*)

Bases: *BaseDensityEstimator*

Density estimator for AI-VAD.

Combines density estimators for the different feature types included in the model.

Parameters

- **use_pose_features** (*bool*) – Flag indicating if pose features should be used. Defaults to `True`.
- **use_deep_features** (*bool*) – Flag indicating if deep features should be used. Defaults to `True`.
- **use_velocity_features** (*bool*) – Flag indicating if velocity features should be used. Defaults to `False`.

- **n_neighbors_pose** (*int*) – Number of neighbors used in KNN density estimation for pose features. Defaults to 1.
- **n_neighbors_deep** (*int*) – Number of neighbors used in KNN density estimation for deep features. Defaults to 1.
- **n_components_velocity** (*int*) – Number of components used by GMM density estimation for velocity features. Defaults to 5.

fit()

Fit the density estimation models on the collected features.

Return type
None

predict(*features*)

Predict the region- and image-level anomaly scores for an image based on a set of features.

Parameters
features (*dict[Tensor]*) – Dictionary containing extracted features for a single frame.

Returns
Region-level anomaly scores for all regions withing the frame. Tensor: Frame-level anomaly score for the frame.

Return type
Tensor

update(*features*, *group=None*)

Update the density estimators for the different feature types.

Parameters

- **features** (*dict[FeatureType, torch.Tensor]*) – Dictionary containing extracted features for a single frame.
- **group** (*str*) – Identifier of the video from which the frame was sampled. Used for grouped density estimation.

Return type
None

class anomalib.models.video.ai_vad.density.**GMMEstimator**(*n_components=2*)

Bases: *BaseDensityEstimator*

Density estimation based on Gaussian Mixture Model.

Parameters
n_components (*int*) – Number of components used in the GMM. Defaults to 2.

fit()

Fit the GMM and compute normalization statistics.

Return type
None

predict(*features*, *normalize=True*)

Predict the density of a set of feature vectors.

Parameters

- **features** (*torch.Tensor*) – Input feature vectors.

- **normalize** (*bool*) – Flag indicating if the density should be normalized to min-max stats of the feature bank. Defaults to `True`.

Returns
Density scores of the input feature vectors.

Return type
Tensor

`update`(*features*, *group=None*)

Update the feature bank.

Return type
`None`

`class anomalib.models.video.ai_vad.density.`**`GroupedKNNEstimator`**(*n_neighbors*)

Bases: `DynamicBufferMixin`, *`BaseDensityEstimator`*

Grouped KNN density estimator.

Keeps track of the group (e.g. video id) from which the features were sampled for normalization purposes.

Parameters
n_neighbors (*int*) – Number of neighbors used in KNN search.

`fit`()

Fit the KNN model by stacking the feature vectors and computing the normalization statistics.

Return type
`None`

`predict`(*features*, *group=None*, *n_neighbors=1*, *normalize=True*)

Predict the (normalized) density for a set of features.

Parameters

- **features** (*torch.Tensor*) – Input features that will be compared to the density model.
- **group** (*str, optional*) – Group (video id) from which the features originate. If passed, all features of the same group in the memory bank will be excluded from the density estimation. Defaults to `None`.
- **n_neighbors** (*int*) – Number of neighbors used in the KNN search. Defaults to `1`.
- **normalize** (*bool*) – Flag indicating if the density should be normalized to min-max stats of the feature bank. Defatuls to `True`.

Returns
Mean (normalized) distances of input feature vectors to k nearest neighbors in feature bank.

Return type
Tensor

`update`(*features*, *group=None*)

Update the internal feature bank while keeping track of the group.

Parameters

- **features** (*torch.Tensor*) – Feature vectors extracted from a video frame.
- **group** (*str*) – Identifier of the group (video) from which the frame was sampled.

Return type
`None`

3.3.3 Engine

Anomalib engine.

`class anomalib.engine.`**`Engine`**`(`*callbacks=None*, *normalization=NormalizationMethod.MIN_MAX*, *threshold='F1AdaptiveThreshold'*, *task=TaskType.SEGMENTATION*, *image_metrics=None*, *pixel_metrics=None*, *logger=None*, *default_root_dir='results'*, ***kwargs*`)`

Bases: `object`

Anomalib Engine.

Note: Refer to PyTorch Lightning's Trainer for a list of parameters for details on other Trainer parameters.

Parameters

- **callbacks** (*list[Callback]*) – Add a callback or list of callbacks.
- **normalization** (*NORMALIZATION, optional*) – Normalization method. Defaults to NormalizationMethod.MIN_MAX.
- **threshold** (*THRESHOLD*) – Thresholding method. Defaults to "F1AdaptiveThreshold".
- **task** (*TaskType, optional*) – Task type. Defaults to TaskType.SEGMENTATION.
- **image_metrics** (*list[str] | str | dict[str, dict[str, Any]] | None, optional*) – Image metrics to be used for evaluation. Defaults to None.
- **pixel_metrics** (*list[str] | str | dict[str, dict[str, Any]] | None, optional*) – Pixel metrics to be used for evaluation. Defaults to None.
- **default_root_dir** (*str, optional*) – Default root directory for the trainer. The results will be saved in this directory. Defaults to `results`.
- ****kwargs** – PyTorch Lightning Trainer arguments.

`export``(`*model*, *export_type*, *export_root=None*, *transform=None*, *ov_args=None*, *ckpt_path=None*`)`

Export the model in PyTorch, ONNX or OpenVINO format.

Parameters

- **model** (*AnomalyModule*) – Trained model.
- **export_type** (`ExportType`) – Export type.
- **export_root** (*str | Path | None, optional*) – Path to the output directory. If it is not set, the model is exported to trainer.default_root_dir. Defaults to None.
- **transform** (*Transform | None, optional*) – Input transform to include in the exported model. If not provided, the engine will try to use the transform from the datamodule or dataset. Defaults to None.
- **ov_args** (*dict[str, Any] | None, optional*) – This is optional and used only for OpenVINO's model optimizer. Defaults to None.
- **ckpt_path** (*str | Path | None*) – Checkpoint path. If provided, the model will be loaded from this path.

Returns

Path to the exported model.

Return type
Path

Raises

- `ValueError` – If Dataset, Datamodule, and transform are not provided.
- `TypeError` – If path to the transform file is not a string or Path.

CLI Usage:

1. **To export as a torch `.pt` file you can run the following command.**
   `` `python anomalib export --model Padim --export_mode TORCH --data MVTec ` ``
2. **To export as an ONNX `.onnx` file you can run the following command.**
   `` `python anomalib export --model Padim --export_mode ONNX --data Visa --input_size "[256,256]" ` ``
3. **To export as an OpenVINO `.xml` and `.bin` file you can run the following command.**
   `` `python anomalib export --model Padim --export_mode OPENVINO --data Visa --input_size "[256,256]" ` ``
4. **You can also overrride OpenVINO model optimizer by adding the `--mo_args.<key>` arguments.**
   `` `python anomalib export --model Padim --export_mode OPENVINO --data Visa --input_size "[256,256]" --mo_args.compress_to_fp16 False ` ``

`fit`(*model*, *train_dataloaders=None*, *val_dataloaders=None*, *datamodule=None*, *ckpt_path=None*)

Fit the model using the trainer.

Parameters

- `model` (*AnomalyModule*) – Model to be trained.
- `train_dataloaders` (*TRAIN_DATALOADERS | None, optional*) – Train dataloaders. Defaults to None.
- `val_dataloaders` (*EVAL_DATALOADERS | None, optional*) – Validation dataloaders. Defaults to None.
- `datamodule` (AnomalibDataModule *| None, optional*) – Lightning datamodule. If provided, dataloaders will be instantiated from this. Defaults to None.
- `ckpt_path` (*str | None, optional*) – Checkpoint path. If provided, the model will be loaded from this path. Defaults to None.

Return type
`None`

CLI Usage:

1. **you can pick a model, and you can run through the MVTec dataset.**
   `` `python anomalib fit --model anomalib.models.Padim ` ``
2. **Of course, you can override the various values with commands.**
   `` `python anomalib fit --model anomalib.models.Padim --data <CONFIG | CLASS_PATH_OR_NAME> --trainer.max_epochs 3 ` ``

4. **If you have a ready configuration file, run it like this.**
   `` `python anomalib fit --config <config_file_path> ` ``

property model: AnomalyModule

Property to get the model.

Raises
`UnassignedError` – When the model is not assigned yet.

Returns
Anomaly model.

Return type
AnomalyModule

property normalization_callback: NormalizationCallback | None

The `NormalizationCallback` callback in the trainer.callbacks list, or `None` if it doesn't exist.

Returns
Normalization callback, if available.

Return type
NormalizationCallback | None

Raises
`ValueError` – If there are multiple normalization callbacks.

predict(*model=None*, *dataloaders=None*, *datamodule=None*, *dataset=None*, *return_predictions=None*, *ckpt_path=None*)

Predict using the model using the trainer.

Sets up the trainer and the dataset task if not already set up. Then validates the model if needed and a validation dataloader is available. Finally, predicts using the model.

Parameters

- **model** (*AnomalyModule | None, optional*) – Model to be used for prediction. Defaults to None.
- **dataloaders** (*EVAL_DATALOADERS | None, optional*) – An iterable or collection of iterables specifying predict samples. Defaults to None.
- **datamodule** (*AnomalibDataModule | None, optional*) – A `AnomalibDataModule` that defines the `predict_dataloader` hook. The datamodule can also be a dataset that will be wrapped in a torch Dataloader. Defaults to None.
- **dataset** (*Dataset | PredictDataset | None, optional*) – A `Dataset` or `PredictDataset` that will be used to create a dataloader. Defaults to None.
- **return_predictions** (*bool | None, optional*) – Whether to return predictions. `True` by default except when an accelerator that spawns processes is used (not supported). Defaults to None.
- **ckpt_path** (*str | None, optional*) – Either `"best"`, `"last"`, `"hpc"` or path to the checkpoint you wish to predict. If `None` and the model instance was passed, use the current weights. Otherwise, the best model checkpoint from the previous `trainer.fit` call will be loaded if a checkpoint callback is configured. Defaults to None.

Returns
Predictions.

Return type
_PREDICT_OUTPUT | None

CLI Usage:

1. **you can pick a model.**
   `` `python anomalib predict --model anomalib.models.Padim anomalib predict --model Padim --data datasets/MVTec/bottle/test/broken_large ` ``
2. **Of course, you can override the various values with commands.**
   `` `python anomalib predict --model anomalib.models.Padim --data <CONFIG | CLASS_PATH_OR_NAME> ` ``

4. **If you have a ready configuration file, run it like this.**
   `` `python anomalib predict --config <config_file_path> --return_predictions ` ``
5. **You can also point to a folder with image or a single image instead of passing a dataset.**
   `` `python anomalib predict --model Padim --data <PATH_TO_IMAGE_OR_FOLDER> --ckpt_path <PATH_TO_CHECKPOINT> ` ``

`test`(*model=None*, *dataloaders=None*, *ckpt_path=None*, *verbose=True*, *datamodule=None*)

Test the model using the trainer.

Sets up the trainer and the dataset task if not already set up. Then validates the model if needed and finally tests the model.

Parameters

- **model** (*AnomalyModule | None, optional*) – The model to be tested. Defaults to None.
- **dataloaders** (*EVAL_DATALOADERS | None, optional*) – An iterable or collection of iterables specifying test samples. Defaults to None.
- **ckpt_path** (*str | None, optional*) – Either `"best"`, `"last"`, `"hpc"` or path to the checkpoint you wish to test. If `None` and the model instance was passed, use the current weights. Otherwise, the best model checkpoint from the previous `trainer.fit` call will be loaded if a checkpoint callback is configured. Defaults to None.
- **verbose** (*bool, optional*) – If True, prints the test results. Defaults to True.
- **datamodule** (*AnomalibDataModule | None, optional*) – A `AnomalibDataModule` that defines the `test_dataloader` hook. Defaults to None.

Returns

A List of dictionaries containing the test results. 1 dict per dataloader.

Return type

_EVALUATE_OUTPUT

Examples

# fit and test a one-class model >>> from anomalib.data import MVTec >>> from anomalib.models import Padim >>> from anomalib.engine import Engine

```
>>> datamodule = MVTec()
>>> model = Padim()
>>> model.learning_type
<LearningType.ONE_CLASS: 'one_class'>
```

```
>>> engine = Engine()
>>> engine.fit(model, datamodule=datamodule)
>>> engine.test(model, datamodule=datamodule)
```

# Test a zero-shot model >>> from anomalib.data import MVTec >>> from anomalib.models import Padim >>> from anomalib.engine import Engine

```
>>> datamodule = MVTec(image_size=240, normalization="clip")
>>> model = Padim()
>>> model.learning_type
<LearningType.ZERO_SHOT: 'zero_shot'>
```

```
>>> engine = Engine()
>>> engine.test(model, datamodule=datamodule)
```

CLI Usage:

1. **you can pick a model.**
   `python anomalib test --model anomalib.models.Padim `
2. **Of course, you can override the various values with commands.**
   `python anomalib test --model anomalib.models.Padim --data <CONFIG | CLASS_PATH_OR_NAME> `

4. **If you have a ready configuration file, run it like this.**
   `python anomalib test --config <config_file_path> `

property threshold_callback: _ThresholdCallback | None

The `ThresholdCallback` callback in the trainer.callbacks list, or `None` if it doesn't exist.

Returns
Threshold callback, if available.

Return type
_ThresholdCallback | None

Raises
ValueError – If there are multiple threshold callbacks.

train(*model*, *train_dataloaders=None*, *val_dataloaders=None*, *test_dataloaders=None*, *datamodule=None*, *ckpt_path=None*)

Fits the model and then calls test on it.

Parameters

- **model** (*AnomalyModule*) – Model to be trained.
- **train_dataloaders** (*TRAIN_DATALOADERS | None, optional*) – Train dataloaders. Defaults to None.
- **val_dataloaders** (*EVAL_DATALOADERS | None, optional*) – Validation dataloaders. Defaults to None.
- **test_dataloaders** (*EVAL_DATALOADERS | None, optional*) – Test dataloaders. Defaults to None.
- **datamodule** (AnomalibDataModule *| None, optional*) – Lightning datamodule. If provided, dataloaders will be instantiated from this. Defaults to None.

- **ckpt_path** (*str | None, optional*) – Checkpoint path. If provided, the model will be loaded from this path. Defaults to None.

Return type
List[Mapping[str, float]]

CLI Usage:

1. **you can pick a model, and you can run through the MVTec dataset.**
   `` `python anomalib train --model anomalib.models.Padim --data MVTec ` ``
2. **Of course, you can override the various values with commands.**
   `` `python anomalib train --model anomalib.models.Padim --data <CONFIG | CLASS_PATH_OR_NAME> --trainer.max_epochs 3 ` ``

4. **If you have a ready configuration file, run it like this.**
   `` `python anomalib train --config <config_file_path> ` ``

property trainer: Trainer

Property to get the trainer.

Raises
`UnassignedError` – When the trainer is not assigned yet.

Returns
Lightning Trainer.

Return type
Trainer

validate(*model=None*, *dataloaders=None*, *ckpt_path=None*, *verbose=True*, *datamodule=None*)

Validate the model using the trainer.

Parameters

- **model** (*AnomalyModule | None, optional*) – Model to be validated. Defaults to None.
- **dataloaders** (*EVAL_DATALOADERS | None, optional*) – Dataloaders to be used for validation. Defaults to None.
- **ckpt_path** (*str | None, optional*) – Checkpoint path. If provided, the model will be loaded from this path. Defaults to None.
- **verbose** (*bool, optional*) – Boolean to print the validation results. Defaults to True.
- **datamodule** (*AnomalibDataModule | None, optional*) – A datamodule `AnomalibDataModule` that defines the `val_dataloader` hook. Defaults to None.

Returns
Validation results.

Return type
_EVALUATE_OUTPUT | None

CLI Usage:

1. **you can pick a model.**
   `` `python anomalib validate --model anomalib.models.Padim ` ``

2. **Of course, you can override the various values with commands.**
   `` `python anomalib validate --model anomalib.models.Padim --data <CONFIG | CLASS_PATH_OR_NAME> ` ``

4. **If you have a ready configuration file, run it like this.**
   `` `python anomalib validate --config <config_file_path> ` ``

3.3.4 Metrics

Custom anomaly evaluation metrics.

class anomalib.metrics.**AUPR**(***kwargs*)

Bases: `PrecisionRecallCurve`

Area under the PR curve.

This metric computes the area under the precision-recall curve.

Parameters
kwargs (Any) – Additional arguments to the TorchMetrics base class.

Examples

To compute the metric for a set of predictions and ground truth targets:

```
>>> true = torch.tensor([0, 1, 1, 1, 0, 0, 0, 0, 1, 1])
>>> pred = torch.tensor([0.59, 0.35, 0.72, 0.33, 0.73, 0.81, 0.30, 0.05, 0.04, 0.
↪48])
```

```
>>> metric = AUPR()
>>> metric(pred, true)
tensor(0.4899)
```

It is also possible to update the metric state incrementally within batches:

```
>>> for batch in dataloader:
...     # Compute prediction and target tensors
...     metric.update(pred, true)
>>> metric.compute()
```

Once the metric has been computed, we can plot the PR curve:

```
>>> figure, title = metric.generate_figure()
```

compute()

First compute PR curve, then compute area under the curve.

Return type
Tensor

Returns
Value of the AUPR metric

generate_figure()

Generate a figure containing the PR curve as well as the random baseline and the AUC.

Returns
Tuple containing both the PR curve and the figure title to be used for logging

Return type
tuple[Figure, str]

update(*preds*, *target*)

Update state with new values.

Need to flatten new values as PrecicionRecallCurve expects them in this format for binary classification.

Parameters

- **preds** (*torch.Tensor*) – predictions of the model
- **target** (*torch.Tensor*) – ground truth targets

Return type
None

class anomalib.metrics.**AUPRO**(*dist_sync_on_step=False*, *process_group=None*, *dist_sync_fn=None*, *fpr_limit=0.3*, *num_thresholds=None*)

Bases: `Metric`

Area under per region overlap (AUPRO) Metric.

Parameters

- **dist_sync_on_step** (*bool*) – Synchronize metric state across processes at each `forward()` before returning the value at the step. Default: `False`
- **process_group** (*Optional[Any]*) – Specify the process group on which synchronization is called. Default: `None` (which selects the entire world)
- **dist_sync_fn** (*Optional[Callable]*) – Callback that performs the allgather operation on the metric state. When `None`, DDP will be used to perform the allgather. Default: `None`
- **fpr_limit** (*float*) – Limit for the false positive rate. Defaults to `0.3`.
- **num_thresholds** (*int*) – Number of thresholds to use for computing the roc curve. Defaults to `None`. If `None`, the roc curve is computed with the thresholds returned by `torchmetrics.functional.classification.thresholds`.

Examples

```
>>> import torch
>>> from anomalib.metrics import AUPRO
...
>>> labels = torch.randint(low=0, high=2, size=(1, 10, 5), dtype=torch.float32)
>>> preds = torch.rand_like(labels)
...
>>> aupro = AUPRO(fpr_limit=0.3)
>>> aupro(preds, labels)
tensor(0.4321)
```

Increasing the fpr_limit will increase the AUPRO value:

```
>>> aupro = AUPRO(fpr_limit=0.7)
>>> aupro(preds, labels)
tensor(0.5271)
```

compute()

Fist compute PRO curve, then compute and scale area under the curve.

Returns
Value of the AUPRO metric

Return type
Tensor

compute_pro(*cca*, *target*, *preds*)

Compute the pro/fpr value-pairs until the fpr specified by self.fpr_limit.

It leverages the fact that the overlap corresponds to the tpr, and thus computes the overall PRO curve by aggregating per-region tpr/fpr values produced by ROC-construction.

Returns
tuple containing final fpr and tpr values.

Return type
tuple[torch.Tensor, torch.Tensor]

generate_figure()

Generate a figure containing the PRO curve and the AUPRO.

Returns
Tuple containing both the figure and the figure title to be used for logging

Return type
tuple[Figure, str]

static **interp1d**(*old_x*, *old_y*, *new_x*)

Interpolate a 1D signal linearly to new sampling points.

Parameters

- **old_x** (*torch.Tensor*) – original 1-D x values (same size as y)
- **old_y** (*torch.Tensor*) – original 1-D y values (same size as x)
- **new_x** (*torch.Tensor*) – x-values where y should be interpolated at

Returns
y-values at corresponding new_x values.

Return type
Tensor

perform_cca()

Perform the Connected Component Analysis on the self.target tensor.

Raises
ValueError – ValueError is raised if self.target doesn't conform with requirements imposed by kornia for connected component analysis.

Returns
Components labeled from 0 to N.

Return type
Tensor

update(*preds*, *target*)
Update state with new values.

Parameters

- **preds** (*torch.Tensor*) – predictions of the model
- **target** (*torch.Tensor*) – ground truth targets

Return type
None

class anomalib.metrics.**AUROC**(*thresholds=None*, *ignore_index=None*, *validate_args=True*, ***kwargs*)
Bases: BinaryROC

Area under the ROC curve.

Examples

```
>>> import torch
>>> from anomalib.metrics import AUROC
...
>>> preds = torch.tensor([0.13, 0.26, 0.08, 0.92, 0.03])
>>> target = torch.tensor([0, 0, 1, 1, 0])
...
>>> auroc = AUROC()
>>> auroc(preds, target)
tensor(0.6667)
```

It is possible to update the metric state incrementally:

```
>>> auroc.update(preds[:2], target[:2])
>>> auroc.update(preds[2:], target[2:])
>>> auroc.compute()
tensor(0.6667)
```

To plot the ROC curve, use the generate_figure method:

```
>>> fig, title = auroc.generate_figure()
```

compute()
First compute ROC curve, then compute area under the curve.

Returns
Value of the AUROC metric

Return type
Tensor

generate_figure()
Generate a figure containing the ROC curve, the baseline and the AUROC.

Returns
Tuple containing both the figure and the figure title to be used for logging

Return type
tuple[Figure, str]

`update`(*preds*, *target*)

Update state with new values.

Need to flatten new values as ROC expects them in this format for binary classification.

Parameters

- **preds** (*torch.Tensor*) – predictions of the model
- **target** (*torch.Tensor*) – ground truth targets

Return type
`None`

`class anomalib.metrics.`**`AnomalyScoreDistribution`**`(`***kwargs*`)`

Bases: `Metric`

Mean and standard deviation of the anomaly scores of normal training data.

`compute`()

Compute stats.

Return type
`tuple[Tensor, Tensor, Tensor, Tensor]`

`update`(**args*, *anomaly_scores=None*, *anomaly_maps=None*, ***kwargs*)

Update the precision-recall curve metric.

Return type
`None`

`class anomalib.metrics.`**`BinaryPrecisionRecallCurve`**`(`*thresholds=None*, *ignore_index=None*, *validate_args=True*, ***kwargs*`)`

Bases: `BinaryPrecisionRecallCurve`

Binary precision-recall curve with without threshold prediction normalization.

`update`(*preds*, *target*)

Update metric state with new predictions and targets.

Unlike the base class, this accepts raw predictions and targets.

Parameters

- **preds** (*Tensor*) – Predicted probabilities
- **target** (*Tensor*) – Ground truth labels

Return type
`None`

`class anomalib.metrics.`**`F1AdaptiveThreshold`**`(`*default_value=0.5*, ***kwargs*`)`

Bases: *BinaryPrecisionRecallCurve*, `BaseThreshold`

Anomaly Score Threshold.

This class computes/stores the threshold that determines the anomalous label given anomaly scores. It initially computes the adaptive threshold to find the optimal f1_score and stores the computed adaptive threshold value.

Parameters
default_value (`float`) – Default value of the threshold. Defaults to `0.5`.

Examples

To find the best threshold that maximizes the F1 score, we could run the following:

```
>>> from anomalib.metrics import F1AdaptiveThreshold
>>> import torch
...
>>> labels = torch.tensor([0, 0, 0, 1, 1])
>>> preds = torch.tensor([2.3, 1.6, 2.6, 7.9, 3.3])
...
>>> adaptive_threshold = F1AdaptiveThreshold(default_value=0.5)
>>> threshold = adaptive_threshold(preds, labels)
>>> threshold
tensor(3.3000)
```

`compute``()`

Compute the threshold that yields the optimal F1 score.

Compute the F1 scores while varying the threshold. Store the optimal threshold as attribute and return the maximum value of the F1 score.

Return type
`Tensor`

Returns
Value of the F1 score at the optimal threshold.

`class anomalib.metrics.`**`F1Score`**`(`*threshold=0.5*, *multidim_average='global'*, *ignore_index=None*, *validate_args=True*, ***kwargs*`)`

Bases: `BinaryF1Score`

This is a wrapper around torchmetrics' BinaryF1Score.

The idea behind this is to retain the current configuration otherwise the one from torchmetrics requires `task` as a parameter.

`class anomalib.metrics.`**`ManualThreshold`**`(`*default_value=0.5*, ***kwargs*`)`

Bases: `BaseThreshold`

Initialize Manual Threshold.

Parameters

- **`default_value`** (*`float, optional`*) – Default threshold value. Defaults to `0.5`.
- **`kwargs`** – Any keyword arguments.

Examples

```
>>> from anomalib.metrics import ManualThreshold
>>> import torch
...
>>> manual_threshold = ManualThreshold(default_value=0.5)
...
>>> labels = torch.randint(low=0, high=2, size=(5,))
>>> preds = torch.rand(5)
...
```

(continues on next page)

(continued from previous page)

```
>>> threshold = manual_threshold(preds, labels)
>>> threshold
tensor(0.5000, dtype=torch.float64)
```

As the threshold is manually set, the threshold value is the same as the `default_value`.

```
>>> labels = torch.randint(low=0, high=2, size=(5,))
>>> preds = torch.rand(5)
>>> threshold = manual_threshold(preds2, labels2)
>>> threshold
tensor(0.5000, dtype=torch.float64)
```

The threshold value remains the same even if the inputs change.

`compute``()`

Compute the threshold.

In case of manual thresholding, the threshold is already set and does not need to be computed.

Returns
Value of the optimal threshold.

Return type
torch.Tensor

`update``(*args, **kwargs)`

Do nothing.

Parameters

- **`*args`** – Any positional arguments.
- **`**kwargs`** – Any keyword arguments.

Return type
`None`

`class anomalib.metrics.`**`MinMax`**`(**kwargs)`

Bases: `Metric`

Track the min and max values of the observations in each batch.

Parameters

- **`full_state_update`** (*`bool, optional`*) – Whether to update the state with the new values. Defaults to `True`.
- **`kwargs`** – Any keyword arguments.

Examples

```
>>> from anomalib.metrics import MinMax
>>> import torch
...
>>> predictions = torch.tensor([0.0807, 0.6329, 0.0559, 0.9860, 0.3595])
>>> minmax = MinMax()
>>> minmax(predictions)
(tensor(0.0559), tensor(0.9860))
```

It is possible to update the minmax values with a new tensor of predictions.

```
>>> new_predictions = torch.tensor([0.3251, 0.3169, 0.3072, 0.6247, 0.9999])
>>> minmax.update(new_predictions)
>>> minmax.compute()
(tensor(0.0559), tensor(0.9999))
```

compute()

Return min and max values.

Return type
`tuple[Tensor, Tensor]`

update(*predictions*, **args*, ***kwargs*)

Update the min and max values.

Return type
`None`

class anomalib.metrics.**PRO**(*threshold=0.5*, ***kwargs*)

Bases: `Metric`

Per-Region Overlap (PRO) Score.

This metric computes the macro average of the per-region overlap between the predicted anomaly masks and the ground truth masks.

Parameters

- **threshold** (*float*) – Threshold used to binarize the predictions. Defaults to `0.5`.
- **kwargs** – Additional arguments to the TorchMetrics base class.

Example

Import the metric from the package:

```
>>> import torch
>>> from anomalib.metrics import PRO
```

Create random `preds` and `labels` tensors:

```
>>> labels = torch.randint(low=0, high=2, size=(1, 10, 5), dtype=torch.float32)
>>> preds = torch.rand_like(labels)
```

Compute the PRO score for labels and preds:

```
>>> pro = PRO(threshold=0.5)
>>> pro.update(preds, labels)
>>> pro.compute()
tensor(0.5433)
```

Note: Note that the example above shows random predictions and labels. Therefore, the PRO score above may not be reproducible.

compute()

Compute the macro average of the PRO score across all regions in all batches.

Return type
`Tensor`

Example

To compute the metric based on the state accumulated from multiple batches, use the `compute` method:

```
>>> pro.compute()
tensor(0.5433)
```

update(*predictions*, *targets*)

Compute the PRO score for the current batch.

Parameters

- **predictions** (*torch.Tensor*) – Predicted anomaly masks (Bx1xHxW)
- **targets** (*torch.Tensor*) – Ground truth anomaly masks (Bx1xHxW)

Return type
`None`

Example

To update the metric state for the current batch, use the `update` method:

```
>>> pro.update(preds, labels)
```

3.3.5 Loggers

Load PyTorch Lightning Loggers.

class anomalib.loggers.**AnomalibCometLogger**(*api_key=None*, *save_dir=None*, *project_name=None*, *rest_api_key=None*, *experiment_name=None*, *experiment_key=None*, *offline=False*, *prefix=''*, ***kwargs*)

Bases: `ImageLoggerBase`, `CometLogger`

Logger for comet.

Adds interface for `add_image` in the logger rather than calling the experiment object.

Note: Same as the CometLogger provided by PyTorch Lightning and the doc string is reproduced below.

Track your parameters, metrics, source code and more using Comet.

Install it with pip:

```
pip install comet-ml
```

Comet requires either an API Key (online mode) or a local directory path (offline mode).

Parameters

- **api_key** (str | None) – Required in online mode. API key, found on Comet.ml. If not given, this will be loaded from the environment variable COMET_API_KEY or ~/.comet.config if either exists. Defaults to None.
- **save_dir** (str | None) – Required in offline mode. The path for the directory to save local comet logs. If given, this also sets the directory for saving checkpoints. Defaults to None.
- **project_name** (str | None) – Optional. Send your experiment to a specific project. Otherwise will be sent to Uncategorized Experiments. If the project name does not already exist, Comet.ml will create a new project. Defaults to None.
- **rest_api_key** (str | None) – Optional. Rest API key found in Comet.ml settings. This is used to determine version number Defaults to None.
- **experiment_name** (str | None) – Optional. String representing the name for this particular experiment on Comet.ml. Defaults to None.
- **experiment_key** (str | None) – Optional. If set, restores from existing experiment. Defaults to None.
- **offline** (bool) – If api_key and save_dir are both given, this determines whether the experiment will be in online or offline mode. This is useful if you use save_dir to control the checkpoints directory and have a ~/.comet.config file but still want to run offline experiments. Defaults to None.
- **prefix** (str) – A string to put at the beginning of metric keys. Defaults to "".
- **kwargs** – Additional arguments like *workspace*, *log_code*, etc. used by CometExperiment can be passed as keyword arguments in this logger.

Raises

- **ModuleNotFoundError** – If required Comet package is not installed on the device.
- **MisconfigurationException** – If neither api_key nor save_dir are passed as arguments.

Example

```
>>> from anomalib.loggers import AnomalibCometLogger
>>> from anomalib.engine import Engine
...
>>> comet_logger = AnomalibCometLogger()
>>> engine =  Engine(logger=comet_logger)
```

See also:

- Comet Documentation

add_image(*image*, *name=None*, ***kwargs*)

Interface to add image to comet logger.

Parameters

- **image** (*np.ndarray | Figure*) – Image to log.
- **name** (*str | None*) – The tag of the image Defaults to `None`.
- **kwargs** – Accepts only *global_step* (int). The step at which to log the image.

Return type

None

class anomalib.loggers.**AnomalibTensorBoardLogger**(*save_dir*, *name='default'*, *version=None*, *log_graph=False*, *default_hp_metric=True*, *prefix=''*, ***kwargs*)

Bases: `ImageLoggerBase`, `TensorBoardLogger`

Logger for tensorboard.

Adds interface for *add_image* in the logger rather than calling the experiment object.

Note: Same as the Tensorboard Logger provided by PyTorch Lightning and the doc string is reproduced below.

Logs are saved to `os.path.join(save_dir, name, version)`. This is the default logger in Lightning, it comes preinstalled.

Example

```
>>> from anomalib.engine import Engine
>>> from anomalib.loggers import AnomalibTensorBoardLogger
...
>>> logger = AnomalibTensorBoardLogger("tb_logs", name="my_model")
>>> engine =  Engine(logger=logger)
```

Parameters

- **save_dir** (*str*) – Save directory
- **name** (*str | None*) – Experiment name. Defaults to `'default'`. If it is the empty string then no per-experiment subdirectory is used. Default: `'default'`.
- **version** (*int | str | None*) – Experiment version. If version is not specified the logger inspects the save directory for existing versions, then automatically assigns the next available version. If it is a string then it is used as the run-specific subdirectory name, otherwise `'version_${version}'` is used. Defaults to `None`
- **log_graph** (*bool*) – Adds the computational graph to tensorboard. This requires that the user has defined the *self.example_input_array* attribute in their model. Defaults to `False`.
- **default_hp_metric** (*bool*) – Enables a placeholder metric with key `hp_metric` when `log_hyperparams` is called without a metric (otherwise calls to log_hyperparams without a metric are ignored). Defaults to `True`.
- **prefix** (*str*) – A string to put at the beginning of metric keys. Defaults to `''`.

- ****kwargs** – Additional arguments like *comment*, *filename_suffix*, etc. used by `SummaryWriter` can be passed as keyword arguments in this logger.

add_image(*image*, *name=None*, ***kwargs*)

Interface to add image to tensorboard logger.

Parameters

- **image** (*np.ndarray | Figure*) – Image to log
- **name** (*str | None*) – The tag of the image Defaults to `None`.
- **kwargs** – Accepts only *global_step* (int). The step at which to log the image.

Return type

None

class anomalib.loggers.**AnomalibWandbLogger**(*name=None*, *save_dir=None*, *offline=False*, *id=None*, *anonymous=None*, *version=None*, *project=None*, *log_model=False*, *experiment=None*, *prefix=''*, ***kwargs*)

Bases: `ImageLoggerBase`, `WandbLogger`

Logger for wandb.

Adds interface for *add_image* in the logger rather than calling the experiment object.

Note: Same as the wandb Logger provided by PyTorch Lightning and the doc string is reproduced below.

Log using Weights and Biases.

Install it with pip:

```
$ pip install wandb
```

Parameters

- **name** (str | None) – Display name for the run. Defaults to `None`.
- **save_dir** (str | None) – Path where data is saved (wandb dir by default). Defaults to `None`.
- **offline** (bool | None) – Run offline (data can be streamed later to wandb servers). Defaults to `False`.
- **id** (str | None) – Sets the version, mainly used to resume a previous run. Defaults to `None`.
- **anonymous** (bool | None) – Enables or explicitly disables anonymous logging. Defaults to `None`.
- **version** (str | None) – Same as id. Defaults to `None`.
- **project** (str | None) – The name of the project to which this run will belong. Defaults to `None`.
- **log_model** (str | bool) – Save checkpoints in wandb dir to upload on W&B servers. Defaults to `False`.
- **experiment** (type[Run] | type[RunDisabled] | None) – WandB experiment object. Automatically set when creating a run. Defaults to `None`.
- **prefix** (str | None) – A string to put at the beginning of metric keys. Defaults to `''`.
- ****kwargs** – Arguments passed to `wandb.init()` like *entity*, *group*, *tags*, etc.

Raises

- **ImportError** – If required WandB package is not installed on the device.
- **MisconfigurationException** – If both log_model and offline``is set to ``True.

Example

```
>>> from anomalib.loggers import AnomalibWandbLogger
>>> from anomalib.engine import Engine
...
>>> wandb_logger = AnomalibWandbLogger()
>>> engine =  Engine(logger=wandb_logger)
```

Note: When logging manually through *wandb.log* or *trainer.logger.experiment.log*, make sure to use *commit=False* so the logging step does not increase.

See also:

- Tutorial on how to use W&B with PyTorch Lightning
- W&B Documentation

add_image(*image*, *name=None*, ***kwargs*)

Interface to add image to wandb logger.

Parameters

- **image** (*np.ndarray | Figure*) – Image to log
- **name** (*str | None*) – The tag of the image Defaults to None.
- **kwargs** – Additional arguments to *wandb.Image*

Return type

None

save()

Upload images to wandb server. :rtype: None

Note: There is a limit on the number of images that can be logged together to the *wandb* server.

3.3.6 Callbacks

Callbacks for Anomalib models.

class anomalib.callbacks.**GraphLogger**

Bases: Callback

Log model graph to respective logger.

Examples

Log model graph to Tensorboard

```
>>> from anomalib.callbacks import GraphLogger
>>> from anomalib.loggers import AnomalibTensorBoardLogger
>>> from anomalib.engine import Engine
...
>>> logger = AnomalibTensorBoardLogger()
>>> callbacks = [GraphLogger()]
>>> engine = Engine(logger=logger, callbacks=callbacks)
```

Log model graph to Comet

```
>>> from anomalib.loggers import AnomalibCometLogger
>>> from anomalib.engine import Engine
...
>>> logger = AnomalibCometLogger()
>>> callbacks = [GraphLogger()]
>>> engine = Engine(logger=logger, callbacks=callbacks)
```

on_train_end(*trainer*, *pl_module*)

Unwatch model if configured for wandb and log it model graph in Tensorboard if specified.

Parameters

- **trainer** (Trainer) – Trainer object which contans reference to loggers.
- **pl_module** (LightningModule) – LightningModule object which is logged.

Return type

None

on_train_start(*trainer*, *pl_module*)

Log model graph to respective logger.

Parameters

- **trainer** (Trainer) – Trainer object which contans reference to loggers.
- **pl_module** (LightningModule) – LightningModule object which is logged.

Return type

None

class anomalib.callbacks.**LoadModelCallback**(*weights_path*)

Bases: Callback

Callback that loads the model weights from the state dict.

Examples

```
>>> from anomalib.callbacks import LoadModelCallback
>>> from anomalib.engine import Engine
...
>>> callbacks = [LoadModelCallback(weights_path="path/to/weights.pt")]
>>> engine = Engine(callbacks=callbacks)
```

setup(*trainer*, *pl_module*, *stage=None*)

Call when inference begins.

Loads the model weights from `weights_path` into the PyTorch module.

Return type
`None`

class anomalib.callbacks.**ModelCheckpoint**(*dirpath=None*, *filename=None*, *monitor=None*, *verbose=False*, *save_last=None*, *save_top_k=1*, *save_weights_only=False*, *mode='min'*, *auto_insert_metric_name=True*, *every_n_train_steps=None*, *train_time_interval=None*, *every_n_epochs=None*, *save_on_train_epoch_end=None*, *enable_version_counter=True*)

Bases: `ModelCheckpoint`

Anomalib Model Checkpoint Callback.

This class overrides the Lightning ModelCheckpoint callback to enable saving checkpoints without running any training steps. This is useful for zero-/few-shot models, where the fit sequence only consists of validation.

To enable saving checkpoints without running any training steps, we need to override two checks which are being called in the `on_validation_end` method of the parent class: - `_should_save_on_train_epoch_end`: This method checks whether the checkpoint should be saved at the end of a

training epoch, or at the end of the validation sequence. We modify this method to default to saving at the end of the validation sequence when the model is of zero- or few-shot type, unless `save_on_train_epoch_end` is specifically set by the user.

- **`_should_skip_saving_checkpoint`: This method checks whether the checkpoint should be saved at all. We modify**
  this method to allow saving during both the `FITTING` and `VALIDATING` states. In addition, we allow saving if the global step has not changed since the last checkpoint, but only for zero- and few-shot models. This is needed because both the last global step and the last checkpoint remain unchanged during zero-/few-shot training, which would otherwise prevent saving checkpoints during validation.

class anomalib.callbacks.**TimerCallback**

Bases: `Callback`

Callback that measures the training and testing time of a PyTorch Lightning module.

Examples

```
>>> from anomalib.callbacks import TimerCallback
>>> from anomalib.engine import Engine
...
>>> callbacks = [TimerCallback()]
>>> engine = Engine(callbacks=callbacks)
```

on_fit_end(*trainer*, *pl_module*)

Call when fit ends.

Prints the time taken for training.

Parameters

- **trainer** (*Trainer*) – PyTorch Lightning trainer.
- **pl_module** (*LightningModule*) – Current training module.

Return type
`None`

Returns
None

on_fit_start(*trainer*, *pl_module*)

Call when fit begins.

Sets the start time to the time training started.

Parameters

- **trainer** (*Trainer*) – PyTorch Lightning trainer.
- **pl_module** (*LightningModule*) – Current training module.

Return type
`None`

Returns
None

on_test_end(*trainer*, *pl_module*)

Call when the test ends.

Prints the time taken for testing and the throughput in frames per second.

Parameters

- **trainer** (*Trainer*) – PyTorch Lightning trainer.
- **pl_module** (*LightningModule*) – Current training module.

Return type
`None`

Returns
None

on_test_start(*trainer*, *pl_module*)

Call when the test begins.

Sets the start time to the time testing started. Goes over all the test dataloaders and adds the number of images in each.

Parameters

- **trainer** (*Trainer*) – PyTorch Lightning trainer.
- **pl_module** (*LightningModule*) – Current training module.

Return type
`None`

Returns
None

3.3.7 CLI

Anomalib CLI.

class `anomalib.cli.`**`AnomalibCLI`**(*args=None*)

Bases: `object`

Implementation of a fully configurable CLI tool for anomalib.

The advantage of this tool is its flexibility to configure the pipeline from both the CLI and a configuration file (.yaml or .json). It is even possible to use both the CLI and a configuration file simultaneously. For more details, the reader could refer to PyTorch Lightning CLI documentation.

`save_config_kwargs` is set to `overwrite=True` so that the `SaveConfigCallback` overwrites the config if it already exists.

`add_arguments_to_parser`(*parser*)

Extend trainer's arguments to add engine arguments. :rtype: `None`

Note: Since `Engine` parameters are manually added, any change to the `Engine` class should be reflected manually.

`add_export_arguments`(*parser*)

Add export arguments to the parser.

Return type
`None`

`add_predict_arguments`(*parser*)

Add predict arguments to the parser.

Return type
`None`

`add_subcommands`(***kwargs*)

Initialize base subcommands and add anomalib specific on top of it.

Return type
`None`

`add_train_arguments`(*parser*)

Add train arguments to the parser.

Return type
`None`

`add_trainer_arguments`(*parser*, *subcommand*)

Add train arguments to the parser.

Return type
`None`

`static anomalib_subcommands()`

Return a dictionary of subcommands and their description.

Return type
`dict[str, dict[str, str]]`

`before_instantiate_classes()`

Modify the configuration to properly instantiate classes and sets up tiler.

Return type
`None`

`property export: Callable`

Export the model using engine's export method.

`property fit: Callable`

Fit the model using engine's fit method.

`init_parser`(***kwargs*)

Method that instantiates the argument parser.

Return type
`ArgumentParser`

`instantiate_classes()`

Instantiate classes depending on the subcommand.

For trainer related commands it instantiates all the model, datamodule and trainer classes. But for subcommands we do not want to instantiate any trainer specific classes such as datamodule, model, etc This is because the subcommand is responsible for instantiating and executing code based on the passed config

Return type
`None`

`instantiate_engine()`

Instantiate the engine. :rtype: `None`

Note: Most of the code in this method is taken from `LightningCLI`'s `instantiate_trainer` method. Refer to that method for more details.

`property predict: Callable`

Predict using engine's predict method.

`static subcommands()`

Skip predict subcommand as it is added later.

Return type
`dict[str, set[str]]`

`property test: Callable`

Test the model using engine's test method.

property train: Callable

Train the model using engine's train method.

property validate: Callable

Validate the model using engine's validate method.

3.3.8 Deployment

Functions for Inference and model deployment.

`class anomalib.deploy.`**`ExportType`**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `str`, `Enum`

Model export type.

Examples

```
>>> from anomalib.deploy import ExportType
>>> ExportType.ONNX
'onnx'
>>> ExportType.OPENVINO
'openvino'
>>> ExportType.TORCH
'torch'
```

`class anomalib.deploy.`**`OpenVINOInferencer`**(*path*, *metadata=None*, *device='AUTO'*, *task=None*, *config=None*)

Bases: `Inferencer`

OpenVINO implementation for the inference.

Parameters

- **path** (*str | Path*) – Path to the openvino onnx, xml or bin file.
- **metadata** (*str | Path | dict, optional*) – Path to metadata file or a dict object defining the metadata. Defaults to `None`.
- **device** (*str | None, optional*) – Device to run the inference on (AUTO, CPU, GPU, NPU). Defaults to `AUTO`.
- **task** (*TaskType | None, optional*) – Task type. Defaults to `None`.
- **config** (*dict | None, optional*) – Configuration parameters for the inference Defaults to `None`.

Examples

Assume that we have an OpenVINO IR model and metadata files in the following structure:

```
$ tree weights
./weights
├── model.bin
├── model.xml
└── metadata.json
```

We could then create `OpenVINOInferencer` as follows:

```
>>> from anomalib.deploy.inferencers import OpenVINOInferencer
>>> inferencer = OpenVINOInferencer(
...     path="weights/model.xml",
...     metadata="weights/metadata.json",
...     device="CPU",
... )
```

This will ensure that the model is loaded on the CPU device and the metadata is loaded from the `metadata.json` file. To make a prediction, we can simply call the `predict` method:

```
>>> prediction = inferencer.predict(image="path/to/image.jpg")
```

Alternatively we can also pass the image as a PIL image or numpy array:

```
>>> from PIL import Image
>>> image = Image.open("path/to/image.jpg")
>>> prediction = inferencer.predict(image=image)
```

```
>>> import numpy as np
>>> image = np.random.rand(224, 224, 3)
>>> prediction = inferencer.predict(image=image)
```

`prediction` will be an `ImageResult` object containing the prediction results. For example, to visualize the heatmap, we can do the following:

```
>>> from matplotlib import pyplot as plt
>>> plt.imshow(result.heatmap)
```

It is also possible to visualize the true and predicted masks if the task is `TaskType.SEGMENTATION`:

```
>>> plt.imshow(result.gt_mask)
>>> plt.imshow(result.pred_mask)
```

forward(*image*)

Forward-Pass input tensor to the model.

Parameters

image (*np.ndarray*) – Input tensor.

Returns

Output predictions.

Return type

np.ndarray

`load_model`(*path*)

Load the OpenVINO model.

Parameters

`path` (*str | Path | tuple[bytes, bytes]*) – Path to the onnx or xml and bin files or tuple of .xml and .bin data as bytes.

Returns

Input and Output blob names
together with the Executable network.

Return type

[tuple[str, str, ExecutableNetwork]]

`post_process`(*predictions*, *metadata=None*)

Post process the output predictions.

Parameters

- `predictions` (*np.ndarray*) – Raw output predicted by the model.
- `metadata` (*Dict, optional*) – Metadata. Post-processing step sometimes requires additional metadata such as image shape. This variable comprises such info. Defaults to None.

Returns

Post processed prediction results.

Return type

dict[str, Any]

`pre_process`(*image*)

Pre-process the input image by applying transformations.

Parameters

`image` (*np.ndarray*) – Input image.

Returns

pre-processed image.

Return type

np.ndarray

`predict`(*image*, *metadata=None*)

Perform a prediction for a given input image.

The main workflow is (i) pre-processing, (ii) forward-pass, (iii) post-process.

Parameters

- `image` (*Union[str, np.ndarray]*) – Input image whose output is to be predicted. It could be either a path to image or numpy array itself.
- `metadata` (`dict[str, Any] | None`) – Metadata information such as shape, threshold.

Returns

Prediction results to be visualized.

Return type

ImageResult

`class anomalib.deploy.`**`TorchInferencer`**(*path*, *device='auto'*)

Bases: `Inferencer`

PyTorch implementation for the inference.

Parameters

- **path** (*str | Path*) – Path to Torch model weights.
- **device** (*str*) – Device to use for inference. Options are `auto`, `cpu`, `cuda`. Defaults to `auto`.

Examples

Assume that we have a Torch `pt` model and metadata files in the following structure:

```
>>> from anomalib.deploy.inferencers import TorchInferencer
>>> inferencer = TorchInferencer(path="path/to/torch/model.pt", device="cpu")
```

This will ensure that the model is loaded on the `CPU` device. To make a prediction, we can simply call the `predict` method:

```
>>> from anomalib.data.utils import read_image
>>> image = read_image("path/to/image.jpg")
>>> result = inferencer.predict(image)
```

`result` will be an `ImageResult` object containing the prediction results. For example, to visualize the heatmap, we can do the following:

```
>>> from matplotlib import pyplot as plt
>>> plt.imshow(result.heatmap)
```

It is also possible to visualize the true and predicted masks if the task is `TaskType.SEGMENTATION`:

```
>>> plt.imshow(result.gt_mask)
>>> plt.imshow(result.pred_mask)
```

`forward`(*image*)

Forward-Pass input tensor to the model.

Parameters
image (*torch.Tensor*) – Input tensor.

Returns
Output predictions.

Return type
Tensor

`load_model`(*path*)

Load the PyTorch model.

Parameters
path (*str | Path*) – Path to the Torch model.

Returns
Torch model.

Return type
(nn.Module)

`post_process`(*predictions*, *metadata=None*)

Post process the output predictions.

Parameters

- **predictions** (*Tensor | list[torch.Tensor] | dict[str, torch.Tensor]*) – Raw output predicted by the model.
- **metadata** (*dict, optional*) – Meta data. Post-processing step sometimes requires additional meta data such as image shape. This variable comprises such info. Defaults to None.

Returns

Post processed prediction results.

Return type

dict[str, str | float | np.ndarray]

`pre_process`(*image*)

Pre process the input image.

Parameters

image (*np.ndarray*) – Input image

Returns

pre-processed image.

Return type

Tensor

`predict`(*image*, *metadata=None*)

Perform a prediction for a given input image.

The main workflow is (i) pre-processing, (ii) forward-pass, (iii) post-process.

Parameters

- **image** (*Union[str, np.ndarray]*) – Input image whose output is to be predicted. It could be either a path to image or numpy array itself.
- **metadata** (`dict[str, Any] | None`) – Metadata information such as shape, threshold.

Returns

Prediction results to be visualized.

Return type

ImageResult

`anomalib.deploy.export_to_onnx`(*model*, *export_root*, *transform=None*, *task=None*, *export_type=ExportType.ONNX*)

Export model to onnx.

Parameters

- **model** (*AnomalyModule*) – Model to export.
- **export_root** (*Path*) – Path to the root folder of the exported model.
- **transform** (*Transform, optional*) – Input transforms used for the model. If not provided, the transform is taken from the model. Defaults to `None`.
- **task** (*TaskType | None*) – Task type. Defaults to `None`.

- **export_type** (ExportType) – Mode to export the model. Since this method is used by OpenVINO export as well, we need to pass the export type so that the right export path is created. Defaults to `ExportType.ONNX`.

Returns
Path to the exported onnx model.

Return type
Path

Examples

Export the Lightning Model to ONNX:

```
>>> from anomalib.models import Patchcore
>>> from anomalib.data import Visa
>>> from anomalib.deploy import export_to_onnx
...
>>> datamodule = Visa()
>>> model = Patchcore()
...
>>> export_to_onnx(
...     model=model,
...     export_root="path/to/export",
...     transform=datamodule.test_data.transform,
...     task=datamodule.test_data.task
... )
```

Using Custom Transforms: This example shows how to use a custom `Compose` object for the `transform` argument.

```
>>> export_to_onnx(
...     model=model,
...     export_root="path/to/export",
...     task="segmentation",
... )
```

`anomalib.deploy.`**`export_to_openvino`**(*export_root*, *model*, *transform=None*, *ov_args=None*, *task=None*)
Convert onnx model to OpenVINO IR.

Parameters

- **export_root** (*Path*) – Path to the export folder.
- **model** (*AnomalyModule*) – AnomalyModule to export.
- **transform** (*Transform, optional*) – Input transforms used for the model. If not provided, the transform is taken from the model. Defaults to `None`.
- **ov_args** (dict[str, Any] | None) – Model optimizer arguments for OpenVINO model conversion. Defaults to `None`.
- **task** (*TaskType | None*) – Task type. Defaults to `None`.

Returns
Path to the exported onnx model.

Return type
Path

Raises
`ModuleNotFoundError` – If OpenVINO is not installed.

Returns
Path to the exported OpenVINO IR.

Return type
Path

Examples

Export the Lightning Model to OpenVINO IR: This example demonstrates how to export the Lightning Model to OpenVINO IR.

```
>>> from anomalib.models import Patchcore
>>> from anomalib.data import Visa
>>> from anomalib.deploy import export_to_openvino
...
>>> datamodule = Visa()
>>> model = Patchcore()
...
>>> export_to_openvino(
...     export_root="path/to/export",
...     model=model,
...     input_size=(224, 224),
...     transform=datamodule.test_data.transform,
...     task=datamodule.test_data.task
... )
```

Using Custom Transforms: This example shows how to use a custom `Transform` object for the `transform` argument.

```
>>> from torchvision.transforms.v2 import Resize
>>> transform = Resize(224, 224)
...
>>> export_to_openvino(
...     export_root="path/to/export",
...     model=model,
...     transform=transform,
...     task="segmentation",
... )
```

`anomalib.deploy.`**`export_to_torch`**(*model*, *export_root*, *transform=None*, *task=None*)

Export AnomalibModel to torch.

Parameters

- **`model`** (*`AnomalyModule`*) – Model to export.
- **`export_root`** (*`Path`*) – Path to the output folder.
- **`transform`** (*`Transform`*, *`optional`*) – Input transforms used for the model. If not provided, the transform is taken from the model. Defaults to `None`.
- **`task`** (*`TaskType | None`*) – Task type. Defaults to `None`.

Returns
Path to the exported pytorch model.

Return type
Path

Examples

Assume that we have a model to train and we want to export it to torch format.

```
>>> from anomalib.data import Visa
>>> from anomalib.models import Patchcore
>>> from anomalib.engine import Engine
...
>>> datamodule = Visa()
>>> model = Patchcore()
>>> engine = Engine()
...
>>> engine.fit(model, datamodule)
```

Now that we have a model trained, we can export it to torch format.

```
>>> from anomalib.deploy import export_to_torch
...
>>> export_to_torch(
...     model=model,
...     export_root="path/to/export",
...     transform=datamodule.test_data.transform,
...     task=datamodule.test_data.task,
... )
```

3.4 How to Guide

This section contains how-to guides for anomalib.

Warning: This section is under construction

3.4.1 Tutorials

Data Learn more about anomalib datamodules.

Models Learn more about image and video models.

Engine Learn more about anomalib Engine.

Metrics Learn more about anomalib metrics

Loggers Learn more about anomalib loggers

Callbacks Learn more about anomalib callbacks

CLI Learn more about anomalib CLI

Deployment Learn more about anomalib's deployment capabilities

Pipelines Learn more about anomalib hpo, sweep and benchmarking pipelines

Data Tutorials

This section contains tutorials on how to fully utilize the data components of anomalib.

Train on Custom Data. Learn more about how to use `Folder` dataset to train anomalib models on your custom data.

Custom Data

This tutorial will show you how to train anomalib models on your custom data. More specifically, we will show you how to use the *Folder* dataset to train anomalib models on your custom data.

Warning: This tutorial assumes that you have already installed anomalib. If not, please refer to the installation section.

Note: We will use our hazelnut_toy dataset to show the capabilities of the *Folder* dataset, but you can use any dataset you want.

We will split the section to two tasks: Classification and Segmentation.

Classification Dataset

In certain use-cases, ground-truth masks for the abnormal images may not be available. In such cases, we could use the classification task to train a model that will be able to detect the abnormal images in the test set.

We will split this section into two tasks:

- Classification with normal and abnormal images, and
- Classification with only normal images.

With Normal and Abnormal Images

We could use *Folder* datamodule to train a model on this dataset. We could run the following python code to create the custom datamodule:

Code Syntax

API

```python
# Import the datamodule
from anomalib.data import Folder

# Create the datamodule
datamodule = Folder(
    name="hazelnut_toy",
    root="datasets/hazelnut_toy",
    normal_dir="good",
```

(continues on next page)

(continued from previous page)

```
    abnormal_dir="crack",
    task="classification",
)

# Setup the datamodule
datamodule.setup()
```

Note: As can be seen above, we only need to specify the `task` argument to `classification`. We could have also use `TaskType.CLASSIFICATION` instead of `classification`.

The *Folder* datamodule will create training, validation, test and prediction datasets and dataloaders for us. We can access the datasets and dataloaders by following the same approach as in the segmentation task.

When we check the samples from the dataloaders, we will see that the `mask` key is not present in the samples. This is because we do not need the masks for the classification task.

```
i, train_data = next(enumerate(datamodule.train_dataloader()))
print(train_data.keys())
# dict_keys(['image_path', 'label', 'image'])

i, val_data = next(enumerate(datamodule.val_dataloader()))
print(val_data.keys())
# dict_keys(['image_path', 'label', 'image'])

i, test_data = next(enumerate(datamodule.test_dataloader()))
print(test_data.keys())
# dict_keys(['image_path', 'label', 'image'])
```

Training the model is as simple as running the following command:

```
# Import the model and engine
from anomalib.models import Patchcore
from anomalib.engine import Engine

# Create the model and engine
model = Patchcore()
engine = Engine(task="classification")

# Train a Patchcore model on the given datamodule
engine.train(datamodule=datamodule, model=model)
```

where we train a Patchcore model on this custom dataset with default model parameters.

CLI

Here is the dataset config to create the same custom datamodule:

```yaml
class_path: anomalib.data.Folder
init_args:
  name: "MVTec"
  root: "datasets/MVTec/transistor"
  normal_dir: "train/good"
  abnormal_dir: "test/bent_lead"
  normal_test_dir: "test/good"
  normal_split_ratio: 0
  extensions: [".png"]
  image_size: [256, 256]
  train_batch_size: 32
  eval_batch_size: 32
  num_workers: 8
  task: classification
  train_transform: null
  eval_transform: null
  test_split_mode: from_dir
  test_split_ratio: 0.2
  val_split_mode: same_as_test
  val_split_ratio: 0.5
  seed: null
```

Assume that we have saved the above config file as `classification.yaml`. We could run the following CLI command to train a Patchcore model on above dataset:

```
anomalib train --data classification.yaml --model anomalib.models.Patchcore --task␣
↪CLASSIFICATION
```

Note: As can be seen above, we also need to specify the `task` argument to `CLASSIFICATION` to explicitly tell anomalib that we want to train a classification model. This is because the default `task` is `SEGMENTATION` within `Engine`.

With Only Normal Images

There are certain cases where we only have normal images in our dataset but would like to train a classification model.

This could be done in two ways:

- Train the model and skip the validation and test steps, as we do not have abnormal images to validate and test the model on, or
- Use the synthetic anomaly generation feature to create abnormal images from normal images, and perform the validation and test steps.

For now we will focus on the second approach.

With Validation and Testing via Synthetic Anomalies

If we want to check the performance of the model, we will need to have abnormal images to validate and test the model on. During the validation stage, these anomalous images are used to normalize the anomaly scores and find the best threshold that separates normal and abnormal images.

Anomalib provides synthetic anomaly generation capabilities to create abnormal images from normal images so we could check the performance. We could use the *Folder* datamodule to train a model on this dataset.

Code Syntax

API

```python
# Import the datamodule
from anomalib.data import Folder
from anomalib.data.utils import TestSplitMode

# Create the datamodule
datamodule = Folder(
    name="hazelnut_toy",
    root="datasets/hazelnut_toy",
    normal_dir="good",
    test_split_mode=TestSplitMode.SYNTHETIC,
    task="classification",
)

# Setup the datamodule
datamodule.setup()
```

Once the datamodule is setup, the rest of the process is the same as in the previous classification example.

```python
# Import the model and engine
from anomalib.models import Patchcore
from anomalib.engine import Engine

# Create the model and engine
model = Patchcore()
engine = Engine(task="classification")

# Train a Patchcore model on the given datamodule
engine.train(datamodule=datamodule, model=model)
```

where we train a Patchcore model on this custom dataset with default model parameters.

CLI

Here is the CLI command to create the same custom datamodule with only normal images. We only need to change the `test_split_mode` argument to `SYNTHETIC` to generate synthetic anomalies.

```yaml
class_path: anomalib.data.Folder
init_args:
  root: "datasets/hazelnut_toy"
  normal_dir: "good"
  normalization: imagenet
  test_split_mode: synthetic
  task: classification
```

Assume that we have saved the above config file as `normal.yaml`. We could run the following CLI command to train a Patchcore model on above dataset:

```
anomalib train --data normal.yaml --model anomalib.models.Patchcore --task CLASSIFICATION
```

Note: As shown in the previous classification example, we, again, need to specify the `task` argument to `CLASSIFICATION` to explicitly tell anomalib that we want to train a classification model. This is because the default `task` is `SEGMENTATION` within `Engine`.

Segmentation Dataset

Assume that we have a dataset in which the training set contains only normal images, and the test set contains both normal and abnormal images. We also have masks for the abnormal images in the test set. We want to train an anomaly segmentation model that will be able to detect the abnormal regions in the test set.

With Normal and Abnormal Images

We could use *Folder* datamodule to load the hazelnut dataset in a format that is readable by Anomalib's models.

Code Syntax

API

We could run the following python code to create the custom datamodule:

```python
# Import the datamodule
from anomalib.data import Folder

# Create the datamodule
datamodule = Folder(
    name="hazelnut_toy",
    root="datasets/hazelnut_toy",
    normal_dir="good",
    abnormal_dir="crack",
    mask_dir="mask/crack",
    normal_split_ratio=0.2,
```

(continues on next page)

(continued from previous page)

```
)

# Setup the datamodule
datamodule.setup()
```

The *Folder* datamodule will create training, validation, test and prediction datasets and dataloaders for us. We can access the datasets and dataloaders using the following attributes:

```
# Access the datasets
train_dataset = datamodule.train_data
val_dataset = datamodule.val_data
test_dataset = datamodule.test_data

# Access the dataloaders
train_dataloader = datamodule.train_dataloader()
val_dataloader = datamodule.val_dataloader()
test_dataloader = datamodule.test_dataloader()
```

To check what individual samples from dataloaders look like, we can run the following command:

```
i, train_data = next(enumerate(datamodule.train_dataloader()))
print(train_data.keys())
# dict_keys(['image_path', 'label', 'image', 'mask_path', 'mask'])

i, val_data = next(enumerate(datamodule.val_dataloader()))
print(val_data.keys())
# dict_keys(['image_path', 'label', 'image', 'mask_path', 'mask'])

i, test_data = next(enumerate(datamodule.test_dataloader()))
print(test_data.keys())
# dict_keys(['image_path', 'label', 'image', 'mask_path', 'mask'])
```

We could check the shape of the images and masks using the following commands:

```
print(train_data["image"].shape)
# torch.Size([32, 3, 256, 256])

print(train_data["mask"].shape)
# torch.Size([32, 256, 256])
```

Training the model is as simple as running the following command:

```
# Import the model and engine
from anomalib.models import Patchcore
from anomalib.engine import Engine

# Create the model and engine
model = Patchcore()
engine = Engine()

# Train a Patchcore model on the given datamodule
engine.train(datamodule=datamodule, model=model)
```

where we train a Patchcore model on this custom dataset with default model parameters.

CLI

Here is the CLI command to create the same custom datamodule:

```yaml
class_path: anomalib.data.Folder
init_args:
  root: "datasets/hazelnut_toy"
  normal_dir: "good"
  abnormal_dir: "crack"
  mask_dir: "mask/crack"
  normal_split_ratio: 0.2
  normalization: imagenet
```

Assume that we have saved the above config file as `segmentation.yaml`. We could run the following CLI command to train a Patchcore model on above dataset:

```
anomalib train --data segmentation.yaml --model anomalib.models.Patchcore
```

This example demonstrates how to create a segmentation dataset with normal and abnormal images. We could expand this example to create a segmentation dataset with only normal images.

With Only Normal Images

There are certain cases where we only have normal images in our dataset but would like to train a segmentation model. This could be done in two ways:

- Train the model and skip the validation and test steps, as we do not have abnormal images to validate and test the model on, or
- Use the synthetic anomaly generation feature to create abnormal images from normal images, and perform the validation and test steps.

For now we will focus on the second approach.

With Validation and Testing via Synthetic Anomalies

We could use the synthetic anomaly generation feature again to create abnormal images from normal images. We could then use the *Folder* datamodule to train a model on this dataset. Here is the python code to create the custom datamodule:

Code Syntax

API

We could run the following python code to create the custom datamodule:

```python
# Import the datamodule
from anomalib.data import Folder
from anomalib.data.utils import TestSplitMode

# Create the datamodule
datamodule = Folder(
```

(continues on next page)

(continued from previous page)

```
    name="hazelnut_toy",
    root="datasets/hazelnut_toy",
    normal_dir="good",
    test_split_mode=TestSplitMode.SYNTHETIC,
)

# Setup the datamodule
datamodule.setup()
```

As can be seen from the code above, we only need to specify the `test_split_mode` argument to `SYNTHETIC`. The *Folder* datamodule will create training, validation, test and prediction datasets and dataloaders for us.

To check what individual samples from dataloaders look like, we can run the following command:

```
i, train_data = next(enumerate(datamodule.train_dataloader()))
print(train_data.keys())
# dict_keys(['image_path', 'label', 'image', 'mask_path', 'mask'])

i, val_data = next(enumerate(datamodule.val_dataloader()))
print(val_data.keys())
# dict_keys(['image_path', 'label', 'image', 'mask_path', 'mask'])

i, test_data = next(enumerate(datamodule.test_dataloader()))
print(test_data.keys())
# dict_keys(['image_path', 'label', 'image', 'mask_path', 'mask'])
```

We could check the shape of the images and masks using the following commands:

```
print(train_data["image"].shape)
# torch.Size([32, 3, 256, 256])

print(train_data["mask"].shape)
# torch.Size([32, 256, 256])
```

CLI

Here is the CLI command to create the same custom datamodule with only normal images. We only need to change the `test_split_mode` argument to `SYNTHETIC` to generate synthetic anomalies.

```
class_path: anomalib.data.Folder
init_args:
  root: "datasets/hazelnut_toy"
  normal_dir: "good"
  normalization: imagenet
  train_batch_size: 32
  eval_batch_size: 32
  num_workers: 8
  task: segmentation
  train_transform: null
  eval_transform: null
  test_split_mode: synthetic
  test_split_ratio: 0.2
```

(continues on next page)

(continued from previous page)

```
  val_split_mode: same_as_test
  val_split_ratio: 0.5
  seed: null
```

Assume that we have saved the above config file as `synthetic.yaml`. We could run the following CLI command to train a Patchcore model on above dataset:

```
anomalib train --data synthetic.yaml --model anomalib.models.Patchcore
```

Model Tutorials

This section contains tutorials on how to use the different model components of anomalib.

Feature Extractors   Learn more about how to use the different backbones as feature extractors.

Feature extractors

This guide demonstrates how different backbones can be used as feature extractors for anomaly detection models. Most of these models use Timm Feature Extractor except **CSFLOW** which uses TorchFx Feature Extractor. Here we show how to use API and CLI to use different backbones as feature extractors.

See also:

For specifics of implementation refer to implementation classes *`Timm Feature Extractor`* and *`TorchFx Feature Extractor`*

Available backbones and layers

Timm

Available Timm models are listed on Timm GitHub page.

In most cases, we want to use a pretrained backbone, so can get a list of all such models using the following code:

```
import timm
# list all pretrained models in timm
for model_name in timm.list_models(pretrained=True):
    print(model_name)
```

Once we have a model selected we can obtain available layer names using the following code:

```
import timm
model = timm.create_model("resnet18", features_only=True)
# Print module names
print(model.feature_info.module_name())
>>>['act1', 'layer1', 'layer2', 'layer3', 'layer4']

model = timm.create_model("mobilenetv3_large_100", features_only=True)
print(model.feature_info.module_name())
>>>['blocks.0.0', 'blocks.1.1', 'blocks.2.2', 'blocks.4.1', 'blocks.6.0']
```

We can then use selected model name and layer names with either API or using config file.

TorchFX

When using TorchFX for feature extraction, you can use either model name, custom model, or instance of model. In this guide, we will cover pretrained models from Torchvision passed by name. For use of the custom model or instance of a model refer to *TorchFxFeatureExtractor class examples*.

Available torchvision models are listed on Torchvision models page.

We can get layer names for selected model using the following code:

```
# Import model and function to list names
from torchvision.models import resnet18
from torchvision.models.feature_extraction import get_graph_node_names

# Make an instance of model with default (latest) weights
model = resnet18(weights="DEFAULT")

# Get and print node (layer) names
train_nodes, eval_nodes = get_graph_node_names(model)
print(eval_nodes)
>>>['x', 'conv1', 'bn1', 'relu', 'maxpool', ..., 'layer4.1.relu_1', 'avgpool', 'flatten',
↪ 'fc']
```

As a result, we get a list of all model nodes, which is quite long.

Now for example, if we want only output from the last node in the block named `layer4`, we specify `layer4.1.relu_1`. If we want to avoid writing `layer4.1.relu_1` to get the last output of `layer4` block, we can shorten it to `layer4`.

We can then use selected model name and layer names with either API or using config file.

See also:

Additional info about TorchFX feature extraction can be found on PyTorch FX page and feature_extraction documentation page.

Warning: Some models might not support every backbone.

Backbone and layer selection

API

When using API, we need to specify `backbone` and `layers` when instantiating the model with a non-default backbone.

```
# Import the required modules
from anomalib.data import MVTec
from anomalib.models import Padim
from anomalib.engine import Engine

# Initialize the datamodule, model, and engine
datamodule = MVTec(num_workers=0)
# Specify backbone and layers
```

(continues on next page)

(continued from previous page)

```python
model = Padim(backbone="resnet18", layers=["layer1", "layer2"])
engine = Engine(image_metrics=["AUROC"], pixel_metrics=["AUROC"])

# Train the model
engine.fit(datamodule=datamodule, model=model)
```

CLI

In the following example config, we can see that we need to specify two parameters: the `backbone` and `layers` list.

```yaml
model:
  class_path: anomalib.models.Padim
  init_args:
    layers:
      - blocks.1.1
      - blocks.2.2
    input_size: null
    backbone: mobilenetv3_large_100
    pre_trained: true
    n_features: 50
```

Then we can train using:

```
anomalib train --config <path/to/config>
```

3.5 Topic Guide

This section contains design documents and other internals of anomalib.

Warning: This section is under construction

3.6 Developer Guide

This section contains information for developers who want to contribute to Anomalib.

Anomalib Software Design Document. Learn more about the architecture, components, and design decisions of Anomalib.

Contribution Guidelies. Learn how to contribute to Anomalib.

Code Review Checklist. Learn the criteria for reviewing code.

3.6.1 Software Design Document

The Software Design Document (SDD) provides a comprehensive overview of the design and architecture of Anomalib. It is intended for developers, architects, and anyone interested in the technical underpinnings of the software.

1. Introduction

Purpose

The purpose of this document is to give a detailed overview of the architecture, components, and design decisions of Anomalib. It serves as a guide for new developers and a reference for all stakeholders involved in development.

Scope

This section outlines the scope of the software, including its intended functionality, user base, and integration with other systems or technologies.

2. Overall Description

Product Perspective

Anomalib is a deep learning library that aims to collect state-of-the-art anomaly detection algorithms for benchmarking on both public and private datasets. Anomalib provides several ready-to-use implementations of anomaly detection algorithms described in the recent literature, as well as a set of tools that facilitate the development and implementation of custom models. The library has a strong focus on visual anomaly detection, where the goal of the algorithm is to detect and/or localize anomalies within images or videos in a dataset.

Product Features

- Modular Design

  Offers a plug-and-play architecture that allows users to easily modify the existing anomaly detection algorithms or add new ones based on their specific requirements.

- Wide Algorithm Support

  Includes implementations of the state-of-the-art (sota) anomaly detection algorithms, covering a variety of techniques such as autoencoders, GANs, flow-based models, and more.

- Real-time Detection

  With a support of multiple inferencers such as Torch, Lightning, ONNX, OpenVINO and Gradio, Anomalib is capable of performing real-time anomaly detection on streaming data.

- Visualization Tools

  Comes with a suite of visualization tools for analyzing anomalies, understanding model decisions, and interpreting results.

- Customization and Extensibility

  Provides APIs for customization and extension, enabling researchers and practitioners to experiment with sota algorithms and techniques.

- Comprehensive Documentation and Community Support

  Features detailed documentation and a supportive community forum to assist users in implementation and troubleshooting.

User Classes and Characteristics

- Research Scientists

  Academics and researchers conducting studies in anomaly detection, requiring a flexible toolset for experimenting with different models and techniques.

- Data Scientists and Analysts

  Professionals looking to incorporate anomaly detection into their data analysis workflows without diving deeper into algorithmic complexities.

- Machine Learning Engineers

  Individuals focused on developing and deploying anomaly detection models, seeking a library that offers sota algorithms.

- Industry Practitioners

  Specialists in sectors like manufacturing, healthcare and security, who need to detect and respond to anomalies as part of risk management and quality control processes.

Operating Environment

Anomalib is designed to be environment-agnostic, capable of running on a variety of platforms including standalone desktop environments, on-premises servers, and cloud-based infrastructures. The library supports both CPU and GPU-based processing, with the latter providing significant performance improvements for large-scale anomaly detection tasks. The library also supports edge devices for real-time anomaly detection in IoT and embedded systems.

Assumptions and Dependencies

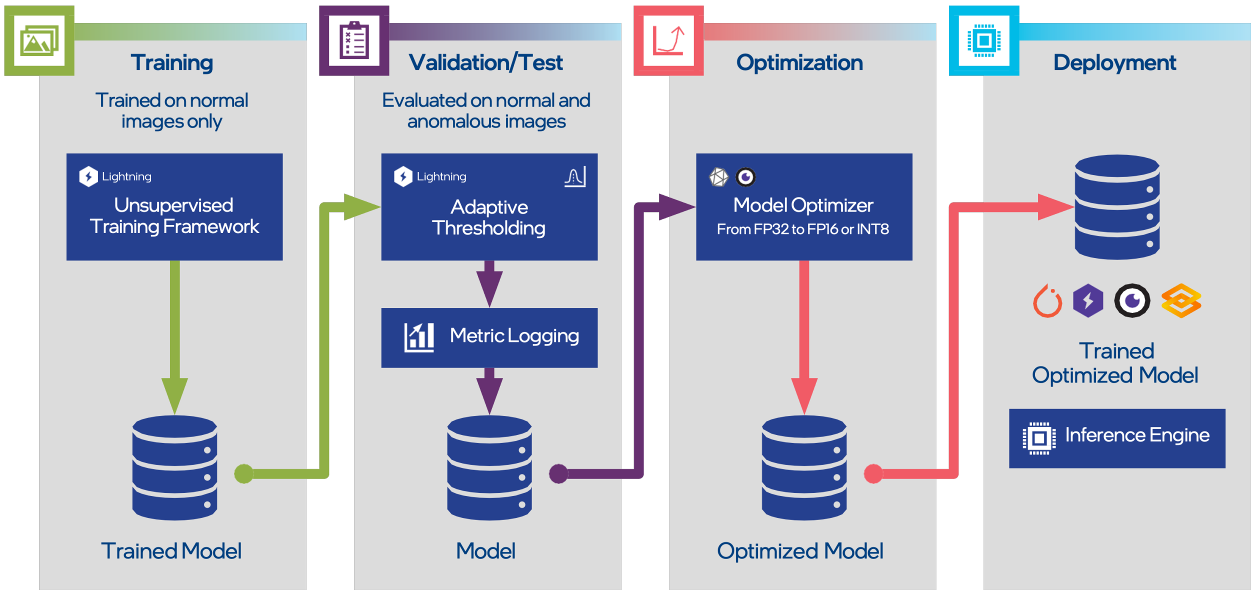
Anomalib's effectiveness and performance are dependent on the quality and granularity of the input data, as well as the appropriateness of the chosen anomaly detection algorithms for the specific use case. It assumes that users have a basic understanding of anomaly detection principles and are familiar with Python programming for custom implementations.

3. System Architecture

Architectural Overview

Anomalib is designed with a focus on modular, scalable, and flexible architecture to meet the diverse needs of anomaly detection across various domains.

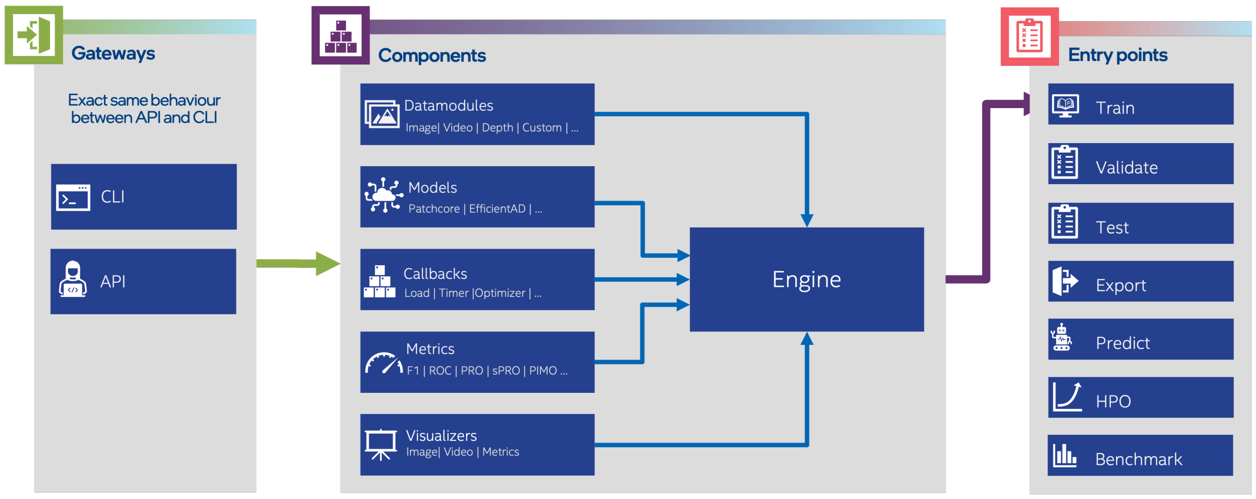
High-Level Workflow



As shown in Figure *1*, Anomalib's high-level workflow consists of the following components:

- **Training [Optional]**: Involves training the anomaly detection model to learn patterns and characteristics of normal behavior. This step is optional for unsupervised algorithms. For zero-shot learning, the model is not trained on any specific dataset.
- **Validation/Test**: Validates the model by finding the threshold for anomaly detection and testing the model's performance on a validation or test dataset.
- **Optimization [Optional]**: Involves optimizing the model by converting it to a more efficient form, such as a compressed or quantized model. This step utilizes tools such as ONNX, OpenVINO for optimization.
- **Deployment**: Deploys the model to a production environment for real-time or batch anomaly detection. This step uses multiple tools such as Torch, Lightning, ONNX, OpenVINO and Gradio for deployment.

High-Level Architecture


Gateways
Exact same behaviour between API and CLI
CLI
API
Components
Datamodules
Image| Video | Depth | Custom | ...
Models
Patchcore | EfficientAD | ...
Callbacks
Load | Timer |Optimizer | ...
Metrics
F1 | ROC | PRO | sPRO | PIMO ...
Visualizers
Image| Video | Metrics
Engine
Entry points
Train
Validate
Test
Export
Predict
HPO
Benchmark

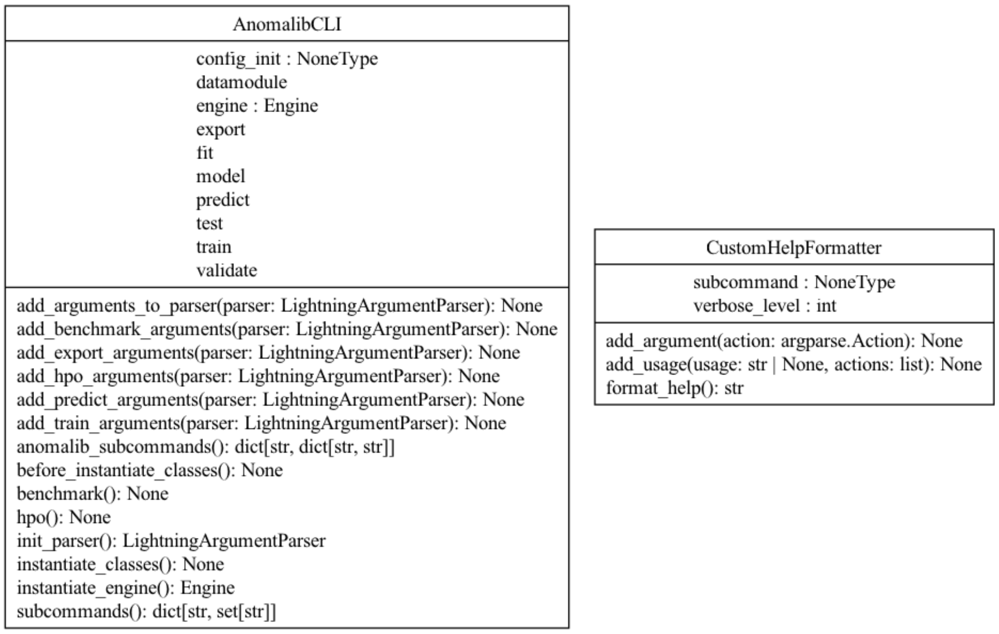
As shown in Figure 2, Anomalib's high-level architecture consists of the following components: API/CLI, Components and Entry Points.

- **API/CLI**: Provides an interface for users to interact with the library, including APIs for data ingestion, configuration, and result export, as well as a command-line interface for running anomaly detection tasks. Both API and CLI provides the same functionality.
- **Components**: Consists of the core components of Anomalib, including, Datamodules, Models, Callbacks, Metrics, Visualizers and Engine.
- **Entry Points**: Includes the main entry points for running anomaly detection tasks, such as `train`, `validate`, `test`, `export`, and `predict`.

4. Detailed System Design

Here we will provide a detailed overview of the design and architecture of Anomalib, including the CLI, core components and entyrpoints.

CLI


AnomalibCLI
config_init : NoneType
datamodule
engine : Engine
export
fit
model
predict
test
train
validate
add_arguments_to_parser(parser: LightningArgumentParser): None
add_benchmark_arguments(parser: LightningArgumentParser): None
add_export_arguments(parser: LightningArgumentParser): None
add_hpo_arguments(parser: LightningArgumentParser): None
add_predict_arguments(parser: LightningArgumentParser): None
add_train_arguments(parser: LightningArgumentParser): None
anomalib_subcommands(): dict[str, dict[str, str]]
before_instantiate_classes(): None
benchmark(): None
hpo(): None
init_parser(): LightningArgumentParser
instantiate_classes(): None
instantiate_engine(): Engine
subcommands(): dict[str, set[str]]
CustomHelpFormatter
subcommand : NoneType
verbose_level : int
add_argument(action: argparse.Action): None
add_usage(usage: str | None, actions: list): None
format_help(): str

As shown in Figure 3, Anomalib's command-line interface (CLI) provides a user-friendly way to interact with the library, allowing users to run anomaly detection tasks, configure algorithms, and visualize results. The CLI is implemented using LightningCLI, a command-line interface framework for PyTorch Lightning that uses jsonargparse for argument parsing. This allows users to sync their CLI arguments with their Python code, making it easier to maintain and update.

Datamodules

Datamodules in anomalib are responsible for preparing the data for training, validation, and testing. They are implemented using PyTorch Lightning's `LightningDataModule` class, which provides a set of APIs for loading, preprocessing, and splitting the data. The datamodules are designed to be modular and extensible, allowing users to easily modify the data loading and preprocessing steps based on their specific requirements. Anomalib has a base datamodule, which is sub-classes by image, video and depth datamodules to provide the necessary functionality for loading and preprocessing image, video and depth data.

Models

Anomalib provides a collection of anomaly models within the image and video domains. The models are implemented sub-classing PyTorch Lightning's `LightningModule` class, which is called `AnomalyModule`, which provides a set of APIs for defining the model architecture, loss function, and optimization algorithm. The models are designed to be modular and extensible, allowing users to easily modify the model architecture and training workflow based on their specific requirements. Anomalib has a base model, which is sub-classes by algorithm implementations such as autoencoders, GANs, flow-based models, and more. The figure below demonstrates the high-level overview of model architecture, and the implementation of Patchcore model.

Callbacks

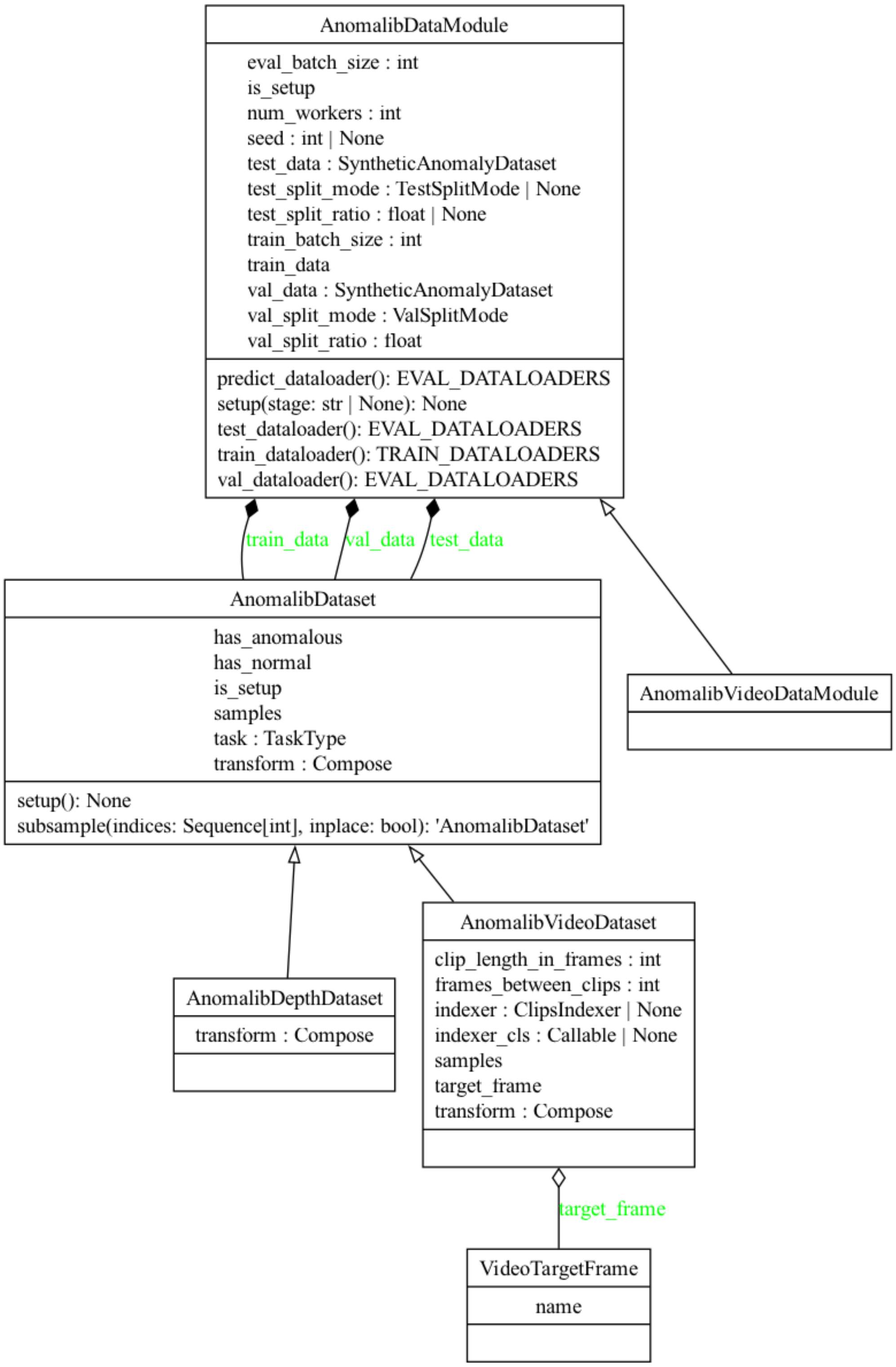
We use LightningCallbacks to add functionality to the training loop, such as logging, early stopping, and model checkpointing. For more details, refer to the Lightning Callbacks documentation. In addition to the built-in callbacks, we have implemented custom callbacks for anomaly detection tasks, which could be accessed through the `callbacks` module.

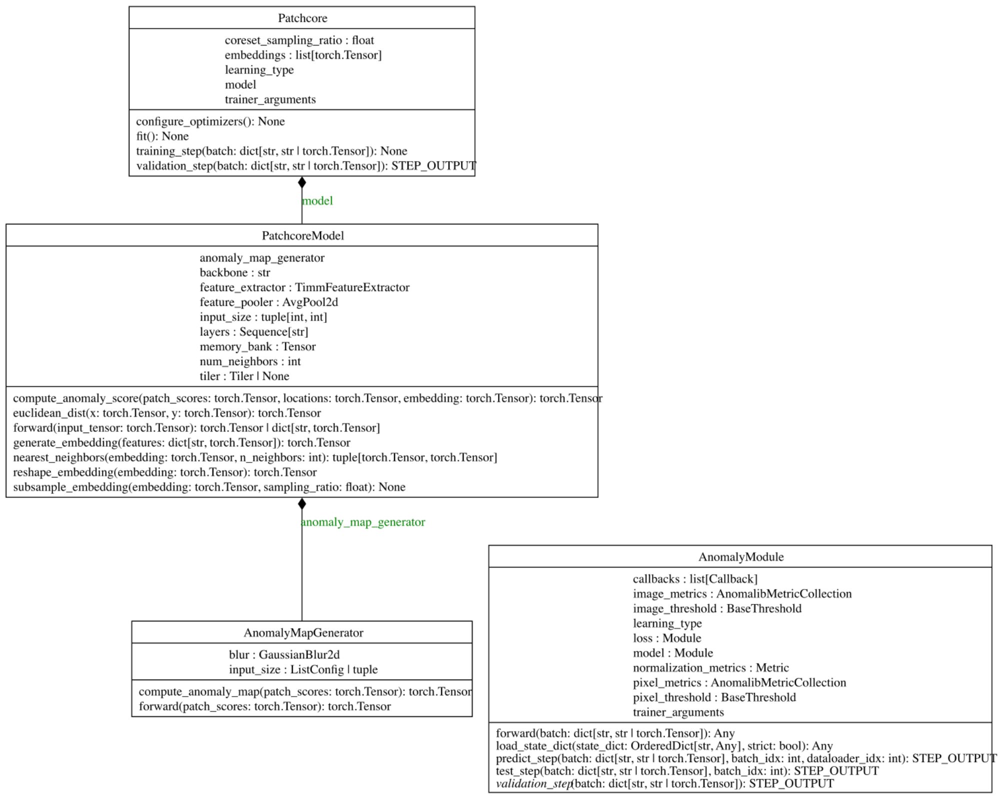
Metrics

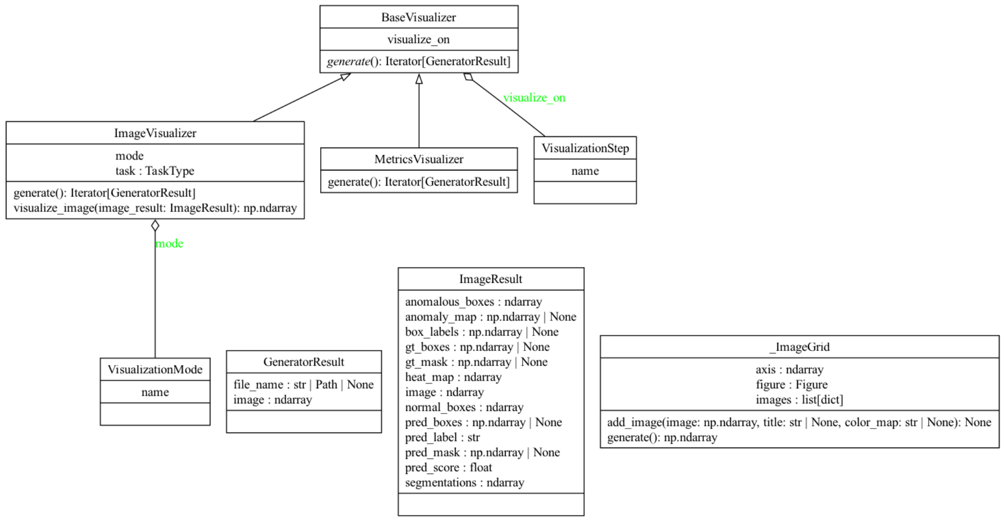
Similar to callbacks, we make use of Lightning Torchmetrics package, which provides a collection of metrics for evaluating model performance. For more details, refer to the Torchmetrics documentation. We have also implemented custom metrics such as PRO, sPRO and PIMO for anomaly detection tasks, which could be accessed through the `metrics` module.

Visualizers

Anomalib provides a suite of visualization tools for analyzing anomalies, understanding model decisions, and interpreting results. Currently we have implemented visualizers for image and metrics, which could be accessed through `ImageVisualizer` and `MetricsVisualizer` classes, respectively. The figure below demonstrates the high-level overview of visualizers.


AnomalibDataModule
eval_batch_size : int
is_setup
num_workers : int
seed : int | None
test_data : SyntheticAnomalyDataset
test_split_mode : TestSplitMode | None
test_split_ratio : float | None
train_batch_size : int
train_data
val_data : SyntheticAnomalyDataset
val_split_mode : ValSplitMode
val_split_ratio : float
predict_dataloader(): EVAL_DATALOADERS
setup(stage: str | None): None
test_dataloader(): EVAL_DATALOADERS
train_dataloader(): TRAIN_DATALOADERS
val_dataloader(): EVAL_DATALOADERS
train_data
val_data
test_data
AnomalibDataset
has_anomalous
has_normal
is_setup
samples
task : TaskType
transform : Compose
setup(): None
subsample(indices: Sequence[int], inplace: bool): 'AnomalibDataset'
AnomalibVideoDataModule
AnomalibVideoDataset
clip_length_in_frames : int
frames_between_clips : int
indexer : ClipsIndexer | None
indexer_cls : Callable | None
samples
target_frame
transform : Compose
AnomalibDepthDataset
transform : Compose
target_frame
VideoTargetFrame
name


Patchcore
coreset_sampling_ratio : float
embeddings : list[torch.Tensor]
learning_type
model
trainer_arguments
configure_optimizers(): None
fit(): None
training_step(batch: dict[str, str | torch.Tensor]): None
validation_step(batch: dict[str, str | torch.Tensor]): STEP_OUTPUT
model
PatchcoreModel
anomaly_map_generator
backbone : str
feature_extractor : TimmFeatureExtractor
feature_pooler : AvgPool2d
input_size : tuple[int, int]
layers : Sequence[str]
memory_bank : Tensor
num_neighbors : int
tiler : Tiler | None
compute_anomaly_score(patch_scores: torch.Tensor, locations: torch.Tensor, embedding: torch.Tensor): torch.Tensor
euclidean_dist(x: torch.Tensor, y: torch.Tensor): torch.Tensor
forward(input_tensor: torch.Tensor): torch.Tensor | dict[str, torch.Tensor]
generate_embedding(features: dict[str, torch.Tensor]): torch.Tensor
nearest_neighbors(embedding: torch.Tensor, n_neighbors: int): tuple[torch.Tensor, torch.Tensor]
reshape_embedding(embedding: torch.Tensor): torch.Tensor
subsample_embedding(embedding: torch.Tensor, sampling_ratio: float): None
anomaly_map_generator
AnomalyMapGenerator
blur : GaussianBlur2d
input_size : ListConfig | tuple
compute_anomaly_map(patch_scores: torch.Tensor): torch.Tensor
forward(patch_scores: torch.Tensor): torch.Tensor
AnomalyModule
callbacks : list[Callback]
image_metrics : AnomalibMetricCollection
image_threshold : BaseThreshold
learning_type
loss : Module
model : Module
normalization_metrics : Metric
pixel_metrics : AnomalibMetricCollection
pixel_threshold : BaseThreshold
trainer_arguments
forward(batch: dict[str, str | torch.Tensor]): Any
load_state_dict(state_dict: OrderedDict[str, Any], strict: bool): Any
predict_step(batch: dict[str, str | torch.Tensor], batch_idx: int, dataloader_idx: int): STEP_OUTPUT
test_step(batch: dict[str, str | torch.Tensor], batch_idx: int): STEP_OUTPUT
validation_step(batch: dict[str, str | torch.Tensor]): STEP_OUTPUT

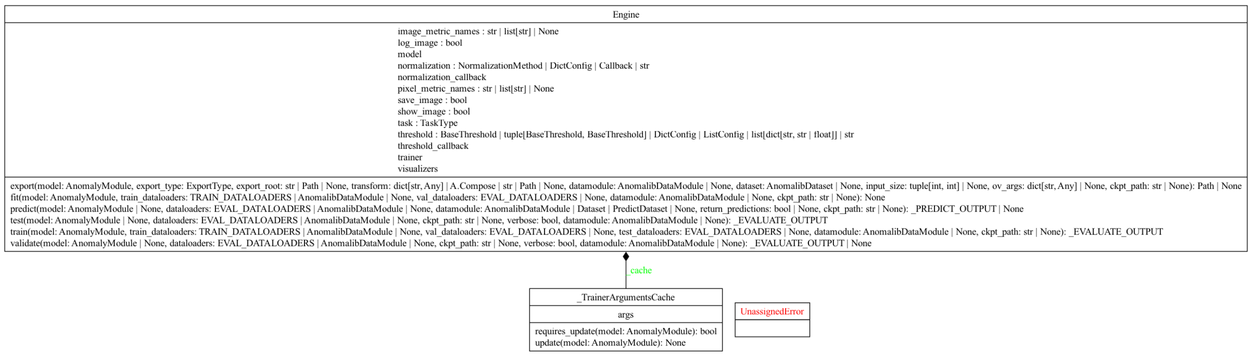

BaseVisualizer
visualize_on
generate(): Iterator[GeneratorResult]
visualize_on
ImageVisualizer
mode
task : TaskType
generate(): Iterator[GeneratorResult]
visualize_image(image_result: ImageResult): np.ndarray
MetricsVisualizer
generate(): Iterator[GeneratorResult]
VisualizationStep
name
mode
VisualizationMode
name
GeneratorResult
file_name : str | Path | None
image : ndarray
ImageResult
anomalous_boxes : ndarray
anomaly_map : np.ndarray | None
box_labels : np.ndarray | None
gt_boxes : np.ndarray | None
gt_mask : np.ndarray | None
heat_map : ndarray
image : ndarray
normal_boxes : ndarray
pred_boxes : np.ndarray | None
pred_label : str
pred_mask : np.ndarray | None
pred_score : float
segmentations : ndarray
_ImageGrid
axis : ndarray
figure : Figure
images : list[dict]
add_image(image: np.ndarray, title: str | None, color_map: str | None): None
generate(): np.ndarray

Engine

The figure below demonstrates the high-level architecture of the engine, which is responsible for orchestrating the anomaly detection workflow, including data preparation, model training, validation, testing, export and predict.

Under the hood, `Engine` uses Lightning's `Trainer` to manage the training workflow. One of the main distinguishing features of the `Engine` is its ability of caching arguments to create the Trainer class after the datamodule and model are created or modified. This allows the user to modify the datamodule and model without having to re-create the Trainer class.

Another important feature of the `Engine` is its ability to export the trained model to OpenVINO format for deployment on edge devices. This is achieved by using the `openvino` module, which provides a set of tools for converting PyTorch models to OpenVINO format.


Engine
image_metric_names : str | list[str] | None
log_image : bool
model
normalization : NormalizationMethod | DictConfig | Callback | str
normalization_callback
pixel_metric_names : str | list[str] | None
save_image : bool
show_image : bool
task : TaskType
threshold : BaseThreshold | tuple[BaseThreshold, BaseThreshold] | DictConfig | ListConfig | list[dict[str, str | float]] | str
threshold_callback
trainer
visualizers
export(model: AnomalyModule, export_type: ExportType, export_root: str | Path | None, transform: dict[str, Any] | A.Compose | str | Path | None, datamodule: AnomalibDataModule | None, dataset: AnomalibDataset | None, input_size: tuple[int, int] | None, ov_args: dict[str, Any] | None, ckpt_path: str | None): Path | None
fit(model: AnomalyModule, train_dataloaders: TRAIN_DATALOADERS | AnomalibDataModule | None, val_dataloaders: EVAL_DATALOADERS | None, datamodule: AnomalibDataModule | None, ckpt_path: str | None): None
predict(model: AnomalyModule | None, dataloaders: EVAL_DATALOADERS | AnomalibDataModule | None, datamodule: AnomalibDataModule | Dataset | PredictDataset | None, return_predictions: bool | None, ckpt_path: str | None): _PREDICT_OUTPUT | None
test(model: AnomalyModule | None, dataloaders: EVAL_DATALOADERS | AnomalibDataModule | None, ckpt_path: str | None, verbose: bool, datamodule: AnomalibDataModule | None): _EVALUATE_OUTPUT
train(model: AnomalyModule, train_dataloaders: TRAIN_DATALOADERS | AnomalibDataModule | None, val_dataloaders: EVAL_DATALOADERS | None, test_dataloaders: EVAL_DATALOADERS | None, datamodule: AnomalibDataModule | None, ckpt_path: str | None): _EVALUATE_OUTPUT
validate(model: AnomalyModule | None, dataloaders: EVAL_DATALOADERS | AnomalibDataModule | None, ckpt_path: str | None, verbose: bool, datamodule: AnomalibDataModule | None): _EVALUATE_OUTPUT | None
_cache
_TrainerArgumentsCache
args
requires_update(model: AnomalyModule): bool
update(model: AnomalyModule): None
UnassignedError

Overall, Anomalib is designed with a focus on modular, scalable, and flexible architecture to meet the diverse needs of anomaly detection across various domains. The library provides a comprehensive set of tools for data preparation, model training, validation, testing, export, and predict, as well as a suite of visualization tools for analyzing anomalies, understanding model decisions, and interpreting results.

3.6.2 Contribution Guidelines

Getting Started

To get started with Anomalib development, follow these steps:

Fork and Clone the Repository

First, fork the Anomalib repository by following the GitHub documentation on forking a repo. Then, clone your forked repository to your local machine and create a new branch from `main`.

Set Up Your Development Environment

Set up your development environment to start contributing. This involves installing the required dependencies and setting up pre-commit hooks for code quality checks. Note that this guide assumes you are using Conda for package management. However, the steps are similar for other package managers.

Development Environment Setup Instructions

1. Create and activate a new Conda environment:

   ```
   conda create -n anomalib_dev python=3.10
   conda activate anomalib_dev
   ```

   Install the base and development requirements:

   ```
   pip install -r requirements/installer.txt -r requirements/dev.txt
   anomalib install -v
   ```

   Optionally, for a full installation with all dependencies:

   ```
   pip install -e .[all]
   ```

2. Install and configure pre-commit hooks:

   ```
   pre-commit install
   ```

   Pre-commit hooks help ensure code quality and consistency. After each commit, `pre-commit` will automatically run the configured checks for the changed file. If you would like to manually run the checks for all files, use:

   ```
   pre-commit run --all-files
   ```

   To bypass pre-commit hooks temporarily (e.g., for a work-in-progress commit), use:

   ```
   git commit -m 'WIP commit' --no-verify
   ```

   However, make sure to address any pre-commit issues before finalizing your pull request.

Making Changes

1. **Write Code:** Follow the project's coding standards and write your code with clear intent. Ensure your code is well-documented and includes examples where appropriate. For code quality we use ruff, whose configuration is in `pyproject.toml` file.
2. **Add Tests:** If your code includes new functionality, add corresponding tests using pytest to maintain coverage and reliability.
3. **Update Documentation:** If you've changed APIs or added new features, update the documentation accordingly. Ensure your docstrings are clear and follow Google's docstring guide.
4. **Pass Tests and Quality Checks:** Ensure the test suite passes and that your code meets quality standards by running:

   ```
   pre-commit run --all-files
   pytest tests/
   ```

5. **Update the Changelog:** For significant changes, add a summary to the CHANGELOG.
6. **Check Licensing:** Ensure you own the code or have rights to use it, adhering to appropriate licensing.
7. **Sign Your Commits:** Use signed commits to certify that you have the right to submit the code under the project's license:

```
git commit -S -m "Your detailed commit message"
```

For more on signing commits, see GitHub's guide on signing commits.

Submitting Pull Requests

Once you've followed the above steps and are satisfied with your changes:

1. Push your changes to your forked repository.
2. Go to the original Anomalib repository you forked and click "New pull request".
3. Choose your fork and the branch with your changes to open a pull request.
4. Fill in the pull request template with the necessary details about your changes.

We look forward to your contributions!

3.6.3 Code Review Checklist

This checklist is a guide for reviewing code changes. It can be used as a reference for both authors and reviewers to ensure that the code meets the project's standards and requirements.

Code Quality

- Is the code clear and understandable?
- Does the code follow the project's coding conventions and style guide (naming conventions, spacing, indentation, etc.)?
- Are there any redundant or unnecessary parts of the code?
- Is there duplicated code that could be refactored into a reusable function/method?
- Are there any magic numbers or strings that should be constants or configurations?

Architecture and Design

- Is the code change consistent with the overall architecture of the system?
- Are the classes, modules, and functions well-organized and appropriately sized?
- Are design patterns used appropriately and consistently?
- Does the change introduce any potential scalability issues?
- Is there a clear separation of concerns (e.g., UI, business logic, data access)?

Functionality

- Does the code do what it’s supposed to do?
- Are all edge cases considered and handled?
- Is there any dead or commented-out code that should be removed?
- Are there any debugging or logging statements that need to be removed or adjusted?

Security

- Are all data inputs validated and sanitized to prevent SQL injection, XSS, etc.?
- Are passwords and sensitive data properly encrypted or secured?
- Are there any potential security vulnerabilities introduced or exposed by the code change?
- Is authentication and authorization handled properly?

Performance

- Are there any obvious performance issues or bottlenecks?
- Is the code optimized for time and space complexity where necessary?
- Are large data sets or files handled efficiently?
- Is caching implemented appropriately?

Testing

- Are there unit tests covering the new functionality or changes?
- Do the existing tests need to be updated or extended?
- Is there appropriate error handling and logging in the tests?
- Do all tests pass?
- Is there enough coverage for critical paths in the code?

Documentation and Comments

- Is the new code adequately commented for clarity?
- Is the documentation (README, API docs, inline comments) updated to reflect the changes?
- Are complex algorithms or decisions well-explained?
- Are there any assumptions or limitations that need to be documented?

Compatibility

- Is the code compatible with all targeted environments (operating systems, browsers, devices)?
- Does the change maintain backward compatibility or is a migration path provided?
- Are there any dependencies added or updated? If so, are they necessary and properly vetted?

Reviewer's General Feedback

- Provide any general feedback or suggestions for improvements.
- Highlight any areas of excellence or particularly clever solutions.

3.7 Awards and Recognition

Anomalib is committed to excellence and innovation in open-source software development. Over the years, our efforts have been recognized by various prestigious organizations and communities. This page highlights the awards and recognitions we have received, reflecting our dedication to our goals and mission.

3.7.1 2024

Passing Badge - OpenSSF Best Practices

- **Date:** January 25, 2024
- **Details:** Anomalib received the passing grade for the Open Source Initiative Best Practices Badge. This badge recognizes our commitment to high-quality coding standards and community engagement.

Learn more about this award

3.7.2 2023

Anomalib as a Pytorch EcoSystem Project

- **Date:** December 15, 2023
- **Details:** Anomalib has been accepted as a Pytorch Ecosystem Project. This recognition highlights our commitment to the Pytorch community and our efforts to make anomaly detection accessible to everyone.

3.7.3 Acknowledgements

We extend our heartfelt thanks to our contributors, users, and the community who have supported us throughout our journey. These recognitions belong not only to the core team but to everyone who has been a part of this project.

Last Updated: 29-Jan-24

3.8 Community Engagement

Anomalib has recently been gaining traction in the developer community. It has recently been the subject of several hackathons, events, and challenges, showcasing its adaptability and impact in real-world scenarios.

3.8.1 Events

Keynote - Visual Anomaly Detection Workshop, CVPR24

- **Date:** June 2024
- **Location:** Seattle, Washington, USA
- **Details:** A keynote showcasing anomalib at the Anomaly Detection workshop at CVPR. Link to the event.

Keynote - Anomaly Detection Workshop, NVPHBV

- **Date:** November 2023
- **Location:** Amsterdam, Netherlands
- **Details:** A keynote showcasing anomalib at the Anomaly Detection workshop at NVPHBV. The workshop was organized by the Dutch Society for Pattern Recognition (NVPHBV). Link to the event.

Talk - AI Grunn Tech Event

- **Date:** November 2023
- **Location:** Groningen, Netherlands
- **Details:** A talk showcasing anomalib at the tech event. Link to the event.

3.8.2 Challenges

Chips Challenge on Hackster.io

- **Date:** December 2023
- **Overview:** A coding marathon where participants created innovative solutions using Anomalib and OpenVINO.
- **Outcome:** Help Chip find all the bad produce at his farm-to-table business using anomaly detection by sharing your knowledge, running a Jupyter notebook or creating an app with OpenVINO.
- **Details:** Chips Challenge

PYTHON MODULE INDEX

a

anomalib.callbacks, 196
anomalib.cli, 200
anomalib.data.base.datamodule, 19
anomalib.data.base.dataset, 18
anomalib.data.base.depth, 21
anomalib.data.base.video, 21
anomalib.data.depth.folder_3d, 53
anomalib.data.depth.mvtec_3d, 56
anomalib.data.image.btech, 22
anomalib.data.image.folder, 27
anomalib.data.image.kolektor, 31
anomalib.data.image.mvtec, 34
anomalib.data.image.visa, 38
anomalib.data.utils.augmenter, 80
anomalib.data.utils.boxes, 73
anomalib.data.utils.download, 62
anomalib.data.utils.generators, 79
anomalib.data.utils.image, 66
anomalib.data.utils.label, 73
anomalib.data.utils.path, 59
anomalib.data.utils.split, 75
anomalib.data.utils.synthetic, 81
anomalib.data.utils.tiler, 76
anomalib.data.utils.video, 72
anomalib.data.video.avenue, 42
anomalib.data.video.shanghaitech, 47
anomalib.data.video.ucsd_ped, 50
anomalib.deploy, 202
anomalib.engine, 178
anomalib.loggers, 192
anomalib.metrics, 184
anomalib.models.components.classification, 93
anomalib.models.components.cluster, 95
anomalib.models.components.dimensionality_reduction, 86
anomalib.models.components.feature_extractors, 82
anomalib.models.components.filters, 92
anomalib.models.components.flow, 89
anomalib.models.components.sampling, 91
anomalib.models.components.stats, 97
anomalib.models.image.cfa.anomaly_map, 102
anomalib.models.image.cfa.lightning_model, 99
anomalib.models.image.cfa.loss, 101
anomalib.models.image.cfa.torch_model, 100
anomalib.models.image.cflow.anomaly_map, 105
anomalib.models.image.cflow.lightning_model, 103
anomalib.models.image.cflow.torch_model, 104
anomalib.models.image.csflow.anomaly_map, 108
anomalib.models.image.csflow.lightning_model, 106
anomalib.models.image.csflow.loss, 108
anomalib.models.image.csflow.torch_model, 107
anomalib.models.image.dfkde.lightning_model, 109
anomalib.models.image.dfkde.torch_model, 110
anomalib.models.image.dfm.lightning_model, 111
anomalib.models.image.dfm.torch_model, 112
anomalib.models.image.draem.lightning_model, 114
anomalib.models.image.draem.loss, 116
anomalib.models.image.draem.torch_model, 116
anomalib.models.image.dsr.anomaly_generator, 128
anomalib.models.image.dsr.lightning_model, 127
anomalib.models.image.dsr.loss, 129
anomalib.models.image.dsr.torch_model, 117
anomalib.models.image.efficient_ad.lightning_model, 130
anomalib.models.image.efficient_ad.torch_model, 132
anomalib.models.image.fastflow.anomaly_map, 136
anomalib.models.image.fastflow.lightning_model, 133
anomalib.models.image.fastflow.loss, 135
anomalib.models.image.fastflow.torch_model, 134
anomalib.models.image.ganomaly.lightning_model, 136

anomalib.models.image.ganomaly.loss, 139
anomalib.models.image.ganomaly.torch_model, 138
anomalib.models.image.padim.lightning_model, 140
anomalib.models.image.padim.torch_model, 141
anomalib.models.image.patchcore.lightning_model, 142
anomalib.models.image.patchcore.torch_model, 144
anomalib.models.image.reverse_distillation.anomaly_map, 149
anomalib.models.image.reverse_distillation.lightning_model, 146
anomalib.models.image.reverse_distillation.loss, 148
anomalib.models.image.reverse_distillation.torch_model, 147
anomalib.models.image.rkde.feature_extractor, 152
anomalib.models.image.rkde.lightning_model, 150
anomalib.models.image.rkde.region_extractor, 153
anomalib.models.image.rkde.torch_model, 151
anomalib.models.image.stfpm.anomaly_map, 157
anomalib.models.image.stfpm.lightning_model, 154
anomalib.models.image.stfpm.loss, 156
anomalib.models.image.stfpm.torch_model, 155
anomalib.models.image.uflow.anomaly_map, 161
anomalib.models.image.uflow.lightning_model, 160
anomalib.models.image.uflow.torch_model, 158
anomalib.models.image.winclip.lightning_model, 163
anomalib.models.image.winclip.torch_model, 164
anomalib.models.video.ai_vad.density, 175
anomalib.models.video.ai_vad.features, 170
anomalib.models.video.ai_vad.flow, 174
anomalib.models.video.ai_vad.lightning_model, 167
anomalib.models.video.ai_vad.regions, 172
anomalib.models.video.ai_vad.torch_model, 169

INDEX

Symbols

`_patch_embeddings` (*anomalib.models.image.winclip.torch_model.WinClipModel attribute*), 165

`_text_embeddings` (*anomalib.models.image.winclip.torch_model.WinClipModel attribute*), 165

`_visual_embeddings` (*anomalib.models.image.winclip.torch_model.WinClipModel attribute*), 165

A

`add_arguments_to_parser()` (*anomalib.cli.AnomalibCLI method*), 200

`add_export_arguments()` (*anomalib.cli.AnomalibCLI method*), 200

`add_foreground_boxes()` (*anomalib.models.video.ai_vad.regions.RegionExtractor method*), 173

`add_image()` (*anomalib.loggers.AnomalibCometLogger method*), 194

`add_image()` (*anomalib.loggers.AnomalibTensorBoardLogger method*), 195

`add_image()` (*anomalib.loggers.AnomalibWandbLogger method*), 196

`add_predict_arguments()` (*anomalib.cli.AnomalibCLI method*), 200

`add_subcommands()` (*anomalib.cli.AnomalibCLI method*), 200

`add_train_arguments()` (*anomalib.cli.AnomalibCLI method*), 200

`add_trainer_arguments()` (*anomalib.cli.AnomalibCLI method*), 200

`AffineCouplingSubnet` (*class in anomalib.models.image.uflow.torch_model*), 158

`AiVad` (*class in anomalib.models.video.ai_vad.lightning_model*), 167

`AiVadModel` (*class in anomalib.models.video.ai_vad.torch_model*), 169

`AllInOneBlock` (*class in anomalib.models.components.flow*), 89

`anomalib.callbacks`
    module, 196

`anomalib.cli`
    module, 200

`anomalib.data.base.datamodule`
    module, 19

`anomalib.data.base.dataset`
    module, 18

`anomalib.data.base.depth`
    module, 21

`anomalib.data.base.video`
    module, 21

`anomalib.data.depth.folder_3d`
    module, 53

`anomalib.data.depth.mvtec_3d`
    module, 56

`anomalib.data.image.btech`
    module, 22

`anomalib.data.image.folder`
    module, 27

`anomalib.data.image.kolektor`
    module, 31

`anomalib.data.image.mvtec`
    module, 34

`anomalib.data.image.visa`
    module, 38

`anomalib.data.utils.augmenter`
    module, 80

`anomalib.data.utils.boxes`
    module, 73

`anomalib.data.utils.download`
    module, 62

`anomalib.data.utils.generators`
    module, 79

`anomalib.data.utils.image`
    module, 66

`anomalib.data.utils.label`
    module, 73

`anomalib.data.utils.path`
    module, 59

anomalib.data.utils.split
    module, 75
anomalib.data.utils.synthetic
    module, 81
anomalib.data.utils.tiler
    module, 76
anomalib.data.utils.video
    module, 72
anomalib.data.video.avenue
    module, 42
anomalib.data.video.shanghaitech
    module, 47
anomalib.data.video.ucsd_ped
    module, 50
anomalib.deploy
    module, 202
anomalib.engine
    module, 178
anomalib.loggers
    module, 192
anomalib.metrics
    module, 184
anomalib.models.components.classification
    module, 93
anomalib.models.components.cluster
    module, 95
anomalib.models.components.dimensionality_reduction
    module, 86
anomalib.models.components.feature_extractors
    module, 82
anomalib.models.components.filters
    module, 92
anomalib.models.components.flow
    module, 89
anomalib.models.components.sampling
    module, 91
anomalib.models.components.stats
    module, 97
anomalib.models.image.cfa.anomaly_map
    module, 102
anomalib.models.image.cfa.lightning_model
    module, 99
anomalib.models.image.cfa.loss
    module, 101
anomalib.models.image.cfa.torch_model
    module, 100
anomalib.models.image.cflow.anomaly_map
    module, 105
anomalib.models.image.cflow.lightning_model
    module, 103
anomalib.models.image.cflow.torch_model
    module, 104
anomalib.models.image.csflow.anomaly_map
    module, 108
anomalib.models.image.csflow.lightning_model
    module, 106
anomalib.models.image.csflow.loss
    module, 108
anomalib.models.image.csflow.torch_model
    module, 107
anomalib.models.image.dfkde.lightning_model
    module, 109
anomalib.models.image.dfkde.torch_model
    module, 110
anomalib.models.image.dfm.lightning_model
    module, 111
anomalib.models.image.dfm.torch_model
    module, 112
anomalib.models.image.draem.lightning_model
    module, 114
anomalib.models.image.draem.loss
    module, 116
anomalib.models.image.draem.torch_model
    module, 116
anomalib.models.image.dsr.anomaly_generator
    module, 128
anomalib.models.image.dsr.lightning_model
    module, 127
anomalib.models.image.dsr.loss
    module, 129
anomalib.models.image.dsr.torch_model
    module, 117
anomalib.models.image.efficient_ad.lightning_model
    module, 130
anomalib.models.image.efficient_ad.torch_model
    module, 132
anomalib.models.image.fastflow.anomaly_map
    module, 136
anomalib.models.image.fastflow.lightning_model
    module, 133
anomalib.models.image.fastflow.loss
    module, 135
anomalib.models.image.fastflow.torch_model
    module, 134
anomalib.models.image.ganomaly.lightning_model
    module, 136
anomalib.models.image.ganomaly.loss
    module, 139
anomalib.models.image.ganomaly.torch_model
    module, 138
anomalib.models.image.padim.lightning_model
    module, 140
anomalib.models.image.padim.torch_model
    module, 141
anomalib.models.image.patchcore.lightning_model
    module, 142
anomalib.models.image.patchcore.torch_model
    module, 144

anomalib.models.image.reverse_distillation.anomaly_map module, 149
anomalib.models.image.reverse_distillation.lightning_model module, 146
anomalib.models.image.reverse_distillation.loss module, 148
anomalib.models.image.reverse_distillation.torch_model module, 147
anomalib.models.image.rkde.feature_extractor module, 152
anomalib.models.image.rkde.lightning_model module, 150
anomalib.models.image.rkde.region_extractor module, 153
anomalib.models.image.rkde.torch_model module, 151
anomalib.models.image.stfpm.anomaly_map module, 157
anomalib.models.image.stfpm.lightning_model module, 154
anomalib.models.image.stfpm.loss module, 156
anomalib.models.image.stfpm.torch_model module, 155
anomalib.models.image.uflow.anomaly_map module, 161
anomalib.models.image.uflow.lightning_model module, 160
anomalib.models.image.uflow.torch_model module, 158
anomalib.models.image.winclip.lightning_model module, 163
anomalib.models.image.winclip.torch_model module, 164
anomalib.models.video.ai_vad.density module, 175
anomalib.models.video.ai_vad.features module, 170
anomalib.models.video.ai_vad.flow module, 174
anomalib.models.video.ai_vad.lightning_model module, 167
anomalib.models.video.ai_vad.regions module, 172
anomalib.models.video.ai_vad.torch_model module, 169
anomalib_subcommands() (*anomalib.cli.AnomalibCLI static method*), 201
AnomalibCLI (*class in anomalib.cli*), 200
AnomalibCometLogger (*class in anomalib.loggers*), 192
AnomalibDataModule (*class in anomalib.data.base.datamodule*), 19
AnomalibDataset (*class in anomalib.data.base.dataset*), 18
AnomalibDepthDataset (*class in anomalib.data.base.depth*), 21
AnomalibTensorBoardLogger (*class in anomalib.loggers*), 194
AnomalibVideoDataModule (*class in anomalib.data.base.video*), 21
AnomalibVideoDataset (*class in anomalib.data.base.video*), 21
AnomalibWandbLogger (*class in anomalib.loggers*), 195
AnomalyDetectionModule (*class in anomalib.models.image.dsr.torch_model*), 117
AnomalyMapGenerationMode (*class in anomalib.models.image.reverse_distillation.anomaly_map*), 149
AnomalyMapGenerator (*class in anomalib.models.image.cfa.anomaly_map*), 102
AnomalyMapGenerator (*class in anomalib.models.image.cflow.anomaly_map*), 105
AnomalyMapGenerator (*class in anomalib.models.image.csflow.anomaly_map*), 108
AnomalyMapGenerator (*class in anomalib.models.image.fastflow.anomaly_map*), 136
AnomalyMapGenerator (*class in anomalib.models.image.reverse_distillation.anomaly_map*), 149
AnomalyMapGenerator (*class in anomalib.models.image.stfpm.anomaly_map*), 157
AnomalyMapGenerator (*class in anomalib.models.image.uflow.anomaly_map*), 161
AnomalyMapMode (*class in anomalib.models.image.csflow.anomaly_map*), 108
AnomalyScoreDistribution (*class in anomalib.metrics*), 188
apply_cls1_split() (*anomalib.data.image.visa.Visa method*), 39
augment_batch() (*anomalib.data.utils.augmenter.Augmenter method*), 80
augment_batch() (*anomalib.models.image.dsr.anomaly_generator.DsrAnomalyGenerator method*), 128
Augmenter (*class in anomalib.data.utils.augmenter*), 80
AUPR (*class in anomalib.metrics*), 184
AUPRO (*class in anomalib.metrics*), 185
AUROC (*class in anomalib.metrics*), 187
Avenue (*class in anomalib.data.video.avenue*), 42
AvenueDataset (*class in anomalib.data.video.avenue*), 45

B

BackboneParams (*class in anomalib.models.components.feature_extractors*), 82

backward() (*anomalib.models.image.cfa.lightning_model.Cfa method*), 99
BaseDensityEstimator (*class in anomalib.models.video.ai_vad.density*), 175
before_instantiate_classes() (*anomalib.cli.AnomalibCLI method*), 201
BinaryPrecisionRecallCurve (*class in anomalib.metrics*), 188
binomial_test() (*anomalib.models.image.uflow.anomaly_map.AnomalyMapGenerator static method*), 161
boxes_to_anomaly_maps() (*in module anomalib.data.utils.boxes*), 73
boxes_to_masks() (*in module anomalib.data.utils.boxes*), 74
BTech (*class in anomalib.data.image.btech*), 22
BTechDataset (*class in anomalib.data.image.btech*), 25
build_flow() (*anomalib.models.image.uflow.torch_model.UflowModel method*), 159
build_flow_stage() (*anomalib.models.image.uflow.torch_model.UflowModel method*), 159

C

category (*anomalib.data.base.datamodule.AnomalibDataModule property*), 20
category (*anomalib.data.base.dataset.AnomalibDataset property*), 19
Cfa (*class in anomalib.models.image.cfa.lightning_model*), 99
CfaLoss (*class in anomalib.models.image.cfa.loss*), 101
CfaModel (*class in anomalib.models.image.cfa.torch_model*), 100
Cflow (*class in anomalib.models.image.cflow.lightning_model*), 103
CflowModel (*class in anomalib.models.image.cflow.torch_model*), 104
check_hash() (*in module anomalib.data.utils.download*), 64
choose_random_aug_image() (*anomalib.models.image.efficient_ad.torch_model.EfficientAdModel method*), 132
clip (*anomalib.models.image.winclip.torch_model.WinClipModel attribute*), 164
ClipsIndexer (*class in anomalib.data.utils.video*), 72
collate_fn() (*in module anomalib.data.base.datamodule*), 21
collect_reference_images() (*anomalib.models.image.winclip.lightning_model.WinClip method*), 163
CombinedDensityEstimator (*class in anomalib.models.video.ai_vad.density*), 175
compute() (*anomalib.metrics.AnomalyScoreDistribution method*), 188
compute() (*anomalib.metrics.AUPR method*), 184
compute() (*anomalib.metrics.AUPRO method*), 186
compute() (*anomalib.metrics.AUROC method*), 187
compute() (*anomalib.metrics.F1AdaptiveThreshold method*), 189
compute() (*anomalib.metrics.ManualThreshold method*), 190
compute() (*anomalib.metrics.MinMax method*), 191
compute() (*anomalib.metrics.PRO method*), 192
compute_anomaly_map() (*anomalib.models.image.cfa.anomaly_map.AnomalyMapGenerator method*), 102
compute_anomaly_map() (*anomalib.models.image.cflow.anomaly_map.AnomalyMapGenerator method*), 105
compute_anomaly_map() (*anomalib.models.image.stfpm.anomaly_map.AnomalyMapGenerator method*), 157
compute_anomaly_map() (*anomalib.models.image.uflow.anomaly_map.AnomalyMapGenerator method*), 161
compute_anomaly_mask() (*anomalib.models.image.uflow.anomaly_map.AnomalyMapGenerator method*), 162
compute_anomaly_score() (*anomalib.models.image.patchcore.torch_model.PatchcoreModel method*), 144
compute_distance() (*anomalib.models.image.cfa.torch_model.CfaModel method*), 100
compute_kde_scores() (*anomalib.models.components.classification.KDEClassifier method*), 93
compute_layer_loss() (*anomalib.models.image.stfpm.loss.STFPMLoss method*), 156
compute_layer_map() (*anomalib.models.image.stfpm.anomaly_map.AnomalyMapGenerator method*), 157
compute_new_image_size() (*in module anomalib.data.utils.tiler*), 78
compute_pro() (*anomalib.metrics.AUPRO method*), 186
compute_probabilities() (*anomalib.models.components.classification.KDEClassifier static method*), 94
compute_score() (*anomalib.models.image.cfa.anomaly_map.AnomalyMapGenerator method*), 102
concatenate_datasets() (*in module anomalib.data.utils.split*), 75
configure_optimizers() (*anoma-

lib.models.image.cfa.lightning_model.Cfa method), 99
configure_optimizers() (*anomalib.models.image.cflow.lightning_model.Cflow method*), 103
configure_optimizers() (*anomalib.models.image.csflow.lightning_model.Csflow method*), 106
configure_optimizers() (*anomalib.models.image.dfkde.lightning_model.Dfkde static method*), 109
configure_optimizers() (*anomalib.models.image.dfm.lightning_model.Dfm static method*), 111
configure_optimizers() (*anomalib.models.image.draem.lightning_model.Draem method*), 115
configure_optimizers() (*anomalib.models.image.dsr.lightning_model.Dsr method*), 127
configure_optimizers() (*anomalib.models.image.efficient_ad.lightning_model.EfficientAd method*), 130
configure_optimizers() (*anomalib.models.image.fastflow.lightning_model.Fastflow method*), 134
configure_optimizers() (*anomalib.models.image.ganomaly.lightning_model.Ganomaly method*), 137
configure_optimizers() (*anomalib.models.image.padim.lightning_model.Padim static method*), 140
configure_optimizers() (*anomalib.models.image.patchcore.lightning_model.Patchcore method*), 143
configure_optimizers() (*anomalib.models.image.reverse_distillation.lightning_model.ReverseDistillation method*), 146
configure_optimizers() (*anomalib.models.image.rkde.lightning_model.Rkde static method*), 150
configure_optimizers() (*anomalib.models.image.stfpm.lightning_model.Stfpm method*), 154
configure_optimizers() (*anomalib.models.image.uflow.lightning_model.Uflow method*), 160
configure_optimizers() (*anomalib.models.image.winclip.lightning_model.WinClip static method*), 163
configure_optimizers() (*anomalib.models.video.ai_vad.lightning_model.AiVad static method*), 168
configure_transforms() (*anomalib.models.image.efficient_ad.lightning_model.EfficientAd method*), 130
configure_transforms() (*anomalib.models.image.padim.lightning_model.Padim method*), 140
configure_transforms() (*anomalib.models.image.patchcore.lightning_model.Patchcore method*), 143
configure_transforms() (*anomalib.models.image.uflow.lightning_model.Uflow method*), 160
configure_transforms() (*anomalib.models.image.winclip.lightning_model.WinClip method*), 163
configure_transforms() (*anomalib.models.video.ai_vad.lightning_model.AiVad method*), 168
contains_non_printable_characters() (*in module anomalib.data.utils.path*), 59
convert_video() (*in module anomalib.data.utils.video*), 73
cov() (*anomalib.models.components.stats.GaussianKDE static method*), 97
Csflow (*class in anomalib.models.image.csflow.lightning_model*), 106
CsFlowLoss (*class in anomalib.models.image.csflow.loss*), 108
CsFlowModel (*class in anomalib.models.image.csflow.torch_model*), 107

D

DecoderBot (*class in anomalib.models.image.dsr.torch_model*), 118
DeepExtractor (*class in anomalib.models.video.ai_vad.features*), 170
Dfkde (*class in anomalib.models.image.dfkde.lightning_model*), 109
DfkdeModel (*class in anomalib.models.image.dfkde.torch_model*), 110
Dfm (*class in anomalib.models.image.dfm.lightning_model*), 111
DFMModel (*class in anomalib.models.image.dfm.torch_model*), 112
DirType (*class in anomalib.data.utils.path*), 59
DiscreteLatentModel (*class in anomalib.models.image.dsr.torch_model*), 118
DiscriminatorLoss (*class in anomalib.models.image.ganomaly.loss*), 139
download_and_extract() (*in module anomalib.data.utils.download*), 64
DownloadInfo (*class in anomalib.data.utils.download*), 62

DownloadProgressBar (*class in anomalib.data.utils.download*), 62
downscale_image() (*in module anomalib.data.utils.tiler*), 78
Draem (*class in anomalib.models.image.draem.lightning_model*), 114
DraemLoss (*class in anomalib.models.image.draem.loss*), 116
DraemModel (*class in anomalib.models.image.draem.torch_model*), 116
dryrun_find_featuremap_dims() (*in module anomalib.models.components.feature_extractors*), 86
Dsr (*class in anomalib.models.image.dsr.lightning_model*), 127
DsrAnomalyGenerator (*class in anomalib.models.image.dsr.anomaly_generator*), 128
DsrModel (*class in anomalib.models.image.dsr.torch_model*), 119
DsrSecondStageLoss (*class in anomalib.models.image.dsr.loss*), 129
DsrThirdStageLoss (*class in anomalib.models.image.dsr.loss*), 129
duplicate_filename() (*in module anomalib.data.utils.image*), 66

E

EfficientAd (*class in anomalib.models.image.efficient_ad.lightning_model*), 130
EfficientAdModel (*class in anomalib.models.image.efficient_ad.torch_model*), 132
embedding (*anomalib.models.image.dsr.torch_model.VectorQuantizer property*), 126
encode() (*anomalib.models.image.uflow.torch_model.UflowModel method*), 160
encode_image() (*anomalib.models.image.winclip.torch_model.WinClipModel method*), 165
EncoderBot (*class in anomalib.models.image.dsr.torch_model*), 121
EncoderTop (*class in anomalib.models.image.dsr.torch_model*), 121
Engine (*class in anomalib.engine*), 178
euclidean_dist() (*anomalib.models.image.patchcore.torch_model.PatchcoreModel static method*), 144
eval_transform (*anomalib.data.base.datamodule.AnomalibDataModule property*), 20
export (*anomalib.cli.AnomalibCLI property*), 201
export() (*anomalib.engine.Engine method*), 178
export_to_onnx() (*in module anomalib.deploy*), 206
export_to_openvino() (*in module anomalib.deploy*), 207
export_to_torch() (*in module anomalib.deploy*), 208
ExportType (*class in anomalib.deploy*), 202
extract() (*in module anomalib.data.utils.download*), 64

F

F1AdaptiveThreshold (*class in anomalib.metrics*), 188
F1Score (*class in anomalib.metrics*), 189
Fastflow (*class in anomalib.models.image.fastflow.lightning_model*), 133
FastflowLoss (*class in anomalib.models.image.fastflow.loss*), 135
FastflowModel (*class in anomalib.models.image.fastflow.torch_model*), 134
FeatureDecoder (*class in anomalib.models.image.dsr.torch_model*), 121
FeatureEncoder (*class in anomalib.models.image.dsr.torch_model*), 122
FeatureExtractor (*class in anomalib.models.image.rkde.feature_extractor*), 152
FeatureExtractor (*class in anomalib.models.video.ai_vad.features*), 171
FeatureScalingMethod (*class in anomalib.models.components.classification*), 93
FeatureType (*class in anomalib.models.video.ai_vad.features*), 171
figure_to_array() (*in module anomalib.data.utils.image*), 66
fit (*anomalib.cli.AnomalibCLI property*), 201
fit() (*anomalib.engine.Engine method*), 179
fit() (*anomalib.models.components.classification.KDEClassifier method*), 94
fit() (*anomalib.models.components.cluster.GaussianMixture method*), 95
fit() (*anomalib.models.components.cluster.KMeans method*), 96
fit() (*anomalib.models.components.dimensionality_reduction.PCA method*), 86
fit() (*anomalib.models.components.dimensionality_reduction.SparseRand method*), 89
fit() (*anomalib.models.components.stats.GaussianKDE method*), 97
fit() (*anomalib.models.components.stats.MultiVariateGaussian method*), 97
fit() (*anomalib.models.image.dfkde.lightning_model.Dfkde method*), 109
fit() (*anomalib.models.image.dfm.lightning_model.Dfm method*), 111

fit() (*anomalib.models.image.dfm.torch_model.DFMModel method*), 113
fit() (*anomalib.models.image.dfm.torch_model.SingleClassGaussian method*), 114
fit() (*anomalib.models.image.padim.lightning_model.Padim method*), 140
fit() (*anomalib.models.image.patchcore.lightning_model.Patchcore method*), 143
fit() (*anomalib.models.image.rkde.lightning_model.Rkde method*), 150
fit() (*anomalib.models.image.rkde.torch_model.RkdeModel method*), 152
fit() (*anomalib.models.video.ai_vad.density.BaseDensityEstimator method*), 175
fit() (*anomalib.models.video.ai_vad.density.CombinedDensityEstimator method*), 176
fit() (*anomalib.models.video.ai_vad.density.GMMEstimator method*), 176
fit() (*anomalib.models.video.ai_vad.density.GroupedKNNEstimator method*), 177
fit() (*anomalib.models.video.ai_vad.lightning_model.AiVad method*), 168
fit_transform() (*anomalib.models.components.dimensionality_reduction.PCA method*), 87
FlowExtractor (*class in anomalib.models.video.ai_vad.flow*), 174
Folder (*class in anomalib.data.image.folder*), 27
Folder3D (*class in anomalib.data.depth.folder_3d*), 53
Folder3DDataset (*class in anomalib.data.depth.folder_3d*), 55
FolderDataset (*class in anomalib.data.image.folder*), 29
forward() (*anomalib.deploy.OpenVINOInferencer method*), 203
forward() (*anomalib.deploy.TorchInferencer method*), 205
forward() (*anomalib.models.components.classification.KDEClassifier method*), 94
forward() (*anomalib.models.components.dimensionality_reduction.PCA method*), 87
forward() (*anomalib.models.components.feature_extractors.TimmFeatureExtractor method*), 83
forward() (*anomalib.models.components.feature_extractors.TorchFXFeatureExtractor method*), 85
forward() (*anomalib.models.components.filters.GaussianBlur2d method*), 92
forward() (*anomalib.models.components.flow.AllInOneBlock method*), 90
forward() (*anomalib.models.components.stats.GaussianKDE method*), 97
forward() (*anomalib.models.components.stats.MultiVariateGaussian method*), 98
forward() (*anomalib.models.image.cfa.anomaly_map.AnomalyMapGenerator method*), 102
forward() (*anomalib.models.image.cfa.loss.CfaLoss method*), 102
forward() (*anomalib.models.image.cfa.torch_model.CfaModel method*), 101
forward() (*anomalib.models.image.cflow.anomaly_map.AnomalyMapGen method*), 105
forward() (*anomalib.models.image.cflow.torch_model.CflowModel method*), 105
forward() (*anomalib.models.image.csflow.anomaly_map.AnomalyMapGe method*), 108
forward() (*anomalib.models.image.csflow.loss.CsFlowLoss method*), 108
forward() (*anomalib.models.image.csflow.torch_model.CsFlowModel method*), 107
forward() (*anomalib.models.image.dfkde.torch_model.DfkdeModel method*), 111
forward() (*anomalib.models.image.dfm.torch_model.DFMModel method*), 113
forward() (*anomalib.models.image.dfm.torch_model.SingleClassGaussian method*), 114
forward() (*anomalib.models.image.draem.loss.DraemLoss method*), 116
forward() (*anomalib.models.image.draem.torch_model.DraemModel method*), 116
forward() (*anomalib.models.image.dsr.loss.DsrSecondStageLoss method*), 129
forward() (*anomalib.models.image.dsr.loss.DsrThirdStageLoss method*), 129
forward() (*anomalib.models.image.dsr.torch_model.AnomalyDetectionMo method*), 117
forward() (*anomalib.models.image.dsr.torch_model.DecoderBot method*), 118
forward() (*anomalib.models.image.dsr.torch_model.DiscreteLatentModel method*), 118
forward() (*anomalib.models.image.dsr.torch_model.DsrModel method*), 120
forward() (*anomalib.models.image.dsr.torch_model.EncoderBot method*), 121
forward() (*anomalib.models.image.dsr.torch_model.EncoderTop method*), 121
forward() (*anomalib.models.image.dsr.torch_model.FeatureDecoder method*), 122
forward() (*anomalib.models.image.dsr.torch_model.FeatureEncoder method*), 122
forward() (*anomalib.models.image.dsr.torch_model.ImageReconstruction method*), 123
forward() (*anomalib.models.image.dsr.torch_model.Residual method*), 123
forward() (*anomalib.models.image.dsr.torch_model.ResidualStack method*), 123
forward() (*anomalib.models.image.dsr.torch_model.SubspaceRestrictionM method*), 124
forward() (*anomalib.models.image.dsr.torch_model.SubspaceRestrictionN*

method), 124
forward() (*anomalib.models.image.dsr.torch_model.UnetDecoder method*), 125
forward() (*anomalib.models.image.dsr.torch_model.UnetEncoder method*), 125
forward() (*anomalib.models.image.dsr.torch_model.UnetModel method*), 125
forward() (*anomalib.models.image.dsr.torch_model.UpsamplingModule method*), 126
forward() (*anomalib.models.image.dsr.torch_model.VectorQuantizer method*), 126
forward() (*anomalib.models.image.efficient_ad.torch_model.EfficientAdModel method*), 133
forward() (*anomalib.models.image.fastflow.anomaly_map.AnomalyMapGenerator method*), 136
forward() (*anomalib.models.image.fastflow.loss.FastflowLoss method*), 135
forward() (*anomalib.models.image.fastflow.torch_model.FastflowModel method*), 135
forward() (*anomalib.models.image.ganomaly.loss.DiscriminatorLoss method*), 139
forward() (*anomalib.models.image.ganomaly.loss.GeneratorLoss method*), 139
forward() (*anomalib.models.image.ganomaly.torch_model.GanomalyModel method*), 138
forward() (*anomalib.models.image.padim.torch_model.PadimModel method*), 142
forward() (*anomalib.models.image.patchcore.torch_model.PatchcoreModel method*), 145
forward() (*anomalib.models.image.reverse_distillation.anomaly_map.AnomalyMapGenerator method*), 149
forward() (*anomalib.models.image.reverse_distillation.loss.ReverseDistillationLoss method*), 148
forward() (*anomalib.models.image.reverse_distillation.torch_model.ReverseDistillationModel method*), 148
forward() (*anomalib.models.image.rkde.feature_extractor.FeatureExtractor method*), 153
forward() (*anomalib.models.image.rkde.region_extractor.RegionExtractor method*), 153
forward() (*anomalib.models.image.rkde.torch_model.RkdeModel method*), 152
forward() (*anomalib.models.image.stfpm.anomaly_map.AnomalyMapGenerator method*), 157
forward() (*anomalib.models.image.stfpm.loss.STFPMLoss method*), 156
forward() (*anomalib.models.image.stfpm.torch_model.STFPMModel method*), 155
forward() (*anomalib.models.image.uflow.anomaly_map.AnomalyMapGenerator method*), 162
forward() (*anomalib.models.image.uflow.torch_model.UflowModel method*), 160
forward() (*anomalib.models.image.winclip.torch_model.WinClipModel method*), 166
forward() (*anomalib.models.video.ai_vad.density.BaseDensityEstimator method*), 175
forward() (*anomalib.models.video.ai_vad.features.DeepExtractor method*), 170
forward() (*anomalib.models.video.ai_vad.features.FeatureExtractor method*), 171
forward() (*anomalib.models.video.ai_vad.features.PoseExtractor method*), 172
forward() (*anomalib.models.video.ai_vad.features.VelocityExtractor method*), 172
forward() (*anomalib.models.video.ai_vad.flow.FlowExtractor method*), 174
forward() (*anomalib.models.video.ai_vad.regions.RegionExtractor method*), 173
forward() (*anomalib.models.video.ai_vad.torch_model.AiVadModel method*), 170
from_dataset() (*anomalib.data.utils.synthetic.SyntheticAnomalyDataset class method*), 81

G

Ganomaly (*class in anomalib.models.image.ganomaly.lightning_model*), 136
GanomalyModel (*class in anomalib.models.image.ganomaly.torch_model*), 138
GaussianBlur2d (*class in anomalib.models.components.filters*), 92
GaussianKDE (*class in anomalib.models.components.stats*), 97
GaussianMixture (*class in anomalib.models.components.cluster*), 95
generate_anomaly() (*anomalib.models.image.dsr.anomaly_generator.DsrAnomalyGenerator method*), 128
generate_embedding() (*anomalib.models.image.padim.torch_model.PadimModel method*), 142
generate_embedding() (*anomalib.models.image.patchcore.torch_model.PatchcoreModel method*), 145
generate_fake_anomalies_joined() (*anomalib.models.image.dsr.torch_model.DiscreteLatentModel method*), 119
generate_figure() (*anomalib.metrics.AUPR method*), 184
generate_figure() (*anomalib.metrics.AUPRO method*), 186
generate_figure() (*anomalib.metrics.AUROC method*), 187
generate_hash() (*in module anomalib.data.utils.download*), 65
generate_output_image_filename() (*in module anomalib.data.utils.image*), 66

generate_perturbation() (*anomalib.data.utils.augmenter.Augmenter method*), 80
GeneratorLoss (*class in anomalib.models.image.ganomaly.loss*), 139
get_clip() (*anomalib.data.video.shanghaitech.ShanghaiTechTestClipsIndexer method*), 49
get_clip() (*anomalib.data.video.ucsd_ped.UCSDpedClipsIndexer method*), 51
get_features() (*anomalib.models.image.dfkde.torch_model.DfkdeModel method*), 111
get_features() (*anomalib.models.image.dfm.torch_model.DFMModel method*), 113
get_image_filename() (*in module anomalib.data.utils.image*), 67
get_image_filenames() (*in module anomalib.data.utils.image*), 68
get_image_filenames_from_dir() (*in module anomalib.data.utils.image*), 68
get_image_height_and_width() (*in module anomalib.data.utils.image*), 69
get_item() (*anomalib.data.utils.video.ClipsIndexer method*), 72
get_mask() (*anomalib.data.utils.video.ClipsIndexer method*), 73
get_mask() (*anomalib.data.video.shanghaitech.ShanghaiTechTestClipsIndexer method*), 49
get_mask() (*anomalib.data.video.shanghaitech.ShanghaiTechTrainClipsIndexer method*), 49
get_mask() (*anomalib.data.video.ucsd_ped.UCSDpedClipsIndexer method*), 51
get_new_idx() (*anomalib.models.components.sampling.KCenterGreedy method*), 91
get_scale() (*anomalib.models.image.cfa.torch_model.CfaModel method*), 101
GMMEstimator (*class in anomalib.models.video.ai_vad.density*), 176
GraphLogger (*class in anomalib.callbacks*), 196
grid_size (*anomalib.models.image.winclip.torch_model.WinClipModel attribute*), 164
GroupedKNNEstimator (*class in anomalib.models.video.ai_vad.density*), 177

H

has_anomalous (*anomalib.data.base.dataset.AnomalibDataset property*), 19
has_normal (*anomalib.data.base.dataset.AnomalibDataset property*), 19

I

ImageReconstructionNetwork (*class in anomalib.models.image.dsr.torch_model*), 122
ImageUpscaleMode (*class in anomalib.data.utils.tiler*), 76
init_parser() (*anomalib.cli.AnomalibCLI method*), 201
initialize_centroid() (*anomalib.models.image.cfa.torch_model.CfaModel method*), 101
initialize_feature_extractor() (*anomalib.models.components.feature_extractors.TorchFXFeatureExtract method*), 85
instantiate_classes() (*anomalib.cli.AnomalibCLI method*), 201
instantiate_engine() (*anomalib.cli.AnomalibCLI method*), 201
interp1d() (*anomalib.metrics.AUPRO static method*), 186
inverse_transform() (*anomalib.models.components.dimensionality_reduction.PCA method*), 87
is_file_potentially_dangerous() (*in module anomalib.data.utils.download*), 65
is_image_file() (*in module anomalib.data.utils.image*), 70
is_path_too_long() (*in module anomalib.data.utils.path*), 60
is_set() (*anomalib.models.image.efficient_ad.torch_model.EfficientAdMo method*), 133
is_within_directory() (*in module anomalib.data.utils.download*), 65

K

k_shot (*anomalib.models.image.winclip.torch_model.WinClipModel attribute*), 164
KCenterGreedy (*class in anomalib.models.components.sampling*), 91
KDEClassifier (*class in anomalib.models.components.classification*), 93
KMeans (*class in anomalib.models.components.cluster*), 96
Kolektor (*class in anomalib.data.image.kolektor*), 31
KolektorDataset (*class in anomalib.data.image.kolektor*), 33

L

LabelName (*class in anomalib.data.utils.label*), 73
last_frame_idx() (*anomalib.data.utils.video.ClipsIndexer method*), 73
learning_type (*anomalib.models.image.cfa.lightning_model.Cfa property*), 99

learning_type (*anomalib.models.image.cflow.lightning_model.Cflow property*), 103
learning_type (*anomalib.models.image.csflow.lightning_model.Csflow property*), 106
learning_type (*anomalib.models.image.dfkde.lightning_model.Dfkde property*), 109
learning_type (*anomalib.models.image.dfm.lightning_model.Dfm property*), 112
learning_type (*anomalib.models.image.draem.lightning_model.Draem property*), 115
learning_type (*anomalib.models.image.dsr.lightning_model.Dsr property*), 127
learning_type (*anomalib.models.image.efficient_ad.lightning_model.EfficientAd property*), 130
learning_type (*anomalib.models.image.fastflow.lightning_model.Fastflow property*), 134
learning_type (*anomalib.models.image.ganomaly.lightning_model.Ganomaly property*), 137
learning_type (*anomalib.models.image.padim.lightning_model.Padim property*), 140
learning_type (*anomalib.models.image.patchcore.lightning_model.Patchcore property*), 143
learning_type (*anomalib.models.image.reverse_distillation.lightning_model.ReverseDistillation property*), 147
learning_type (*anomalib.models.image.rkde.lightning_model.Rkde property*), 150
learning_type (*anomalib.models.image.stfpm.lightning_model.Stfpm property*), 154
learning_type (*anomalib.models.image.uflow.lightning_model.Uflow property*), 160
learning_type (*anomalib.models.image.winclip.lightning_model.WinClip property*), 163
learning_type (*anomalib.models.video.ai_vad.lightning_model.AiVad property*), 168
load_model() (*anomalib.deploy.OpenVINOInferencer method*), 203
load_model() (*anomalib.deploy.TorchInferencer method*), 205
load_pretrained_discrete_model_weights() (*anomalib.models.image.dsr.torch_model.DsrModel method*), 120
load_state_dict() (*anomalib.models.image.winclip.lightning_model.WinClip method*), 163
LoadModelCallback (*class in anomalib.callbacks*), 197

M

make_avenue_dataset() (*in module anomalib.data.video.avenue*), 46
make_btech_dataset() (*in module anomalib.data.image.btech*), 26
make_folder3d_dataset() (*in module anomalib.data.depth.folder_3d*), 56
make_folder_dataset() (*in module anomalib.data.image.folder*), 30
make_kolektor_dataset() (*in module anomalib.data.image.kolektor*), 33
make_mvtec_3d_dataset() (*in module anomalib.data.depth.mvtec_3d*), 58
make_mvtec_dataset() (*in module anomalib.data.image.mvtec*), 37
make_shanghaitech_dataset() (*in module anomalib.data.video.shanghaitech*), 49
make_synthetic_dataset() (*in module anomalib.data.utils.synthetic*), 81
make_ucsd_dataset() (*in module anomalib.data.video.ucsd_ped*), 52
ManualThreshold (*class in anomalib.metrics*), 189
map_norm_quantiles() (*anomalib.models.image.efficient_ad.lightning_model.EfficientAd method*), 131
masks (*anomalib.models.image.winclip.torch_model.WinClipModel attribute*), 165
masks_to_boxes() (*in module anomalib.data.utils.boxes*), 74
MinMax (*class in anomalib.metrics*), 190
model (*anomalib.engine.Engine property*), 179
ModelCheckpoint (*class in anomalib.callbacks*), 198
module
- anomalib.callbacks, 196
- anomalib.cli, 200
- anomalib.data.base.datamodule, 19
- anomalib.data.base.dataset, 18
- anomalib.data.base.depth, 21
- anomalib.data.base.video, 21
- anomalib.data.depth.folder_3d, 53
- anomalib.data.depth.mvtec_3d, 56
- anomalib.data.image.btech, 22
- anomalib.data.image.folder, 27
- anomalib.data.image.kolektor, 31
- anomalib.data.image.mvtec, 34

anomalib.data.image.visa, 38
anomalib.data.utils.augmenter, 80
anomalib.data.utils.boxes, 73
anomalib.data.utils.download, 62
anomalib.data.utils.generators, 79
anomalib.data.utils.image, 66
anomalib.data.utils.label, 73
anomalib.data.utils.path, 59
anomalib.data.utils.split, 75
anomalib.data.utils.synthetic, 81
anomalib.data.utils.tiler, 76
anomalib.data.utils.video, 72
anomalib.data.video.avenue, 42
anomalib.data.video.shanghaitech, 47
anomalib.data.video.ucsd_ped, 50
anomalib.deploy, 202
anomalib.engine, 178
anomalib.loggers, 192
anomalib.metrics, 184
anomalib.models.components.classification, 93
anomalib.models.components.cluster, 95
anomalib.models.components.dimensionality_reduction, 86
anomalib.models.components.feature_extractors, 82
anomalib.models.components.filters, 92
anomalib.models.components.flow, 89
anomalib.models.components.sampling, 91
anomalib.models.components.stats, 97
anomalib.models.image.cfa.anomaly_map, 102
anomalib.models.image.cfa.lightning_model, 99
anomalib.models.image.cfa.loss, 101
anomalib.models.image.cfa.torch_model, 100
anomalib.models.image.cflow.anomaly_map, 105
anomalib.models.image.cflow.lightning_model, 103
anomalib.models.image.cflow.torch_model, 104
anomalib.models.image.csflow.anomaly_map, 108
anomalib.models.image.csflow.lightning_model, 106
anomalib.models.image.csflow.loss, 108
anomalib.models.image.csflow.torch_model, 107
anomalib.models.image.dfkde.lightning_model, 109
anomalib.models.image.dfkde.torch_model, 110
anomalib.models.image.dfm.lightning_model, 111
anomalib.models.image.dfm.torch_model, 112
anomalib.models.image.draem.lightning_model, 114
anomalib.models.image.draem.loss, 116
anomalib.models.image.draem.torch_model, 116
anomalib.models.image.dsr.anomaly_generator, 128
anomalib.models.image.dsr.lightning_model, 127
anomalib.models.image.dsr.loss, 129
anomalib.models.image.dsr.torch_model, 117
anomalib.models.image.efficient_ad.lightning_model, 130
anomalib.models.image.efficient_ad.torch_model, 132
anomalib.models.image.fastflow.anomaly_map, 136
anomalib.models.image.fastflow.lightning_model, 133
anomalib.models.image.fastflow.loss, 135
anomalib.models.image.fastflow.torch_model, 134
anomalib.models.image.ganomaly.lightning_model, 136
anomalib.models.image.ganomaly.loss, 139
anomalib.models.image.ganomaly.torch_model, 138
anomalib.models.image.padim.lightning_model, 140
anomalib.models.image.padim.torch_model, 141
anomalib.models.image.patchcore.lightning_model, 142
anomalib.models.image.patchcore.torch_model, 144
anomalib.models.image.reverse_distillation.anomaly_map
149
anomalib.models.image.reverse_distillation.lightning_m
146
anomalib.models.image.reverse_distillation.loss, 148
anomalib.models.image.reverse_distillation.torch_model
147
anomalib.models.image.rkde.feature_extractor, 152
anomalib.models.image.rkde.lightning_model, 150
anomalib.models.image.rkde.region_extractor, 153

anomalib.models.image.rkde.torch_model, 151
anomalib.models.image.stfpm.anomaly_map, 157
anomalib.models.image.stfpm.lightning_model, 154
anomalib.models.image.stfpm.loss, 156
anomalib.models.image.stfpm.torch_model, 155
anomalib.models.image.uflow.anomaly_map, 161
anomalib.models.image.uflow.lightning_model, 160
anomalib.models.image.uflow.torch_model, 158
anomalib.models.image.winclip.lightning_model, 163
anomalib.models.image.winclip.torch_model, 164
anomalib.models.video.ai_vad.density, 175
anomalib.models.video.ai_vad.features, 170
anomalib.models.video.ai_vad.flow, 174
anomalib.models.video.ai_vad.lightning_model, 167
anomalib.models.video.ai_vad.regions, 172
anomalib.models.video.ai_vad.torch_model, 169

MultiVariateGaussian (*class in anomalib.models.components.stats*), 97
MVTec (*class in anomalib.data.image.mvtec*), 34
MVTec3D (*class in anomalib.data.depth.mvtec_3d*), 57
MVTec3DDataset (*class in anomalib.data.depth.mvtec_3d*), 58
MVTecDataset (*class in anomalib.data.image.mvtec*), 36

N

name (*anomalib.data.base.datamodule.AnomalibDataModule property*), 20
name (*anomalib.data.base.dataset.AnomalibDataset property*), 19
name (*anomalib.data.depth.folder_3d.Folder3D property*), 54
name (*anomalib.data.depth.folder_3d.Folder3DDataset property*), 55
name (*anomalib.data.image.folder.Folder property*), 29
name (*anomalib.data.image.folder.FolderDataset property*), 30
nearest_neighbors() (*anomalib.models.image.patchcore.torch_model.PatchcoreModel method*), 145
nextpow2() (*in module anomalib.data.utils.augmenter*), 81
normalization_callback (*anomalib.engine.Engine property*), 180

O

on_fit_end() (*anomalib.callbacks.TimerCallback method*), 199
on_fit_start() (*anomalib.callbacks.TimerCallback method*), 199
on_test_batch_end() (*anomalib.models.image.ganomaly.lightning_model.Ganomaly method*), 137
on_test_end() (*anomalib.callbacks.TimerCallback method*), 199
on_test_start() (*anomalib.callbacks.TimerCallback method*), 199
on_test_start() (*anomalib.models.image.ganomaly.lightning_model.Ganomaly method*), 137
on_train_end() (*anomalib.callbacks.GraphLogger method*), 197
on_train_epoch_start() (*anomalib.models.image.dsr.lightning_model.Dsr method*), 127
on_train_start() (*anomalib.callbacks.GraphLogger method*), 197
on_train_start() (*anomalib.models.image.cfa.lightning_model.Cfa method*), 99
on_train_start() (*anomalib.models.image.dsr.lightning_model.Dsr method*), 127
on_train_start() (*anomalib.models.image.efficient_ad.lightning_model.EfficientAd method*), 131
on_validation_batch_end() (*anomalib.models.image.ganomaly.lightning_model.Ganomaly method*), 137
on_validation_start() (*anomalib.models.image.efficient_ad.lightning_model.EfficientAd method*), 131
on_validation_start() (*anomalib.models.image.ganomaly.lightning_model.Ganomaly method*), 137
OpenVINOInferencer (*class in anomalib.deploy*), 202
output_dims() (*anomalib.models.components.flow.AllInOneBlock method*), 90

P

pad_nextpow2() (*in module anomalib.data.utils.image*), 70
Padim (*class in anomalib.models.image.padim.lightning_model*), 140

PadimModel (*class in anomalib.models.image.padim.torch_model*), 141
patch_embeddings (*anomalib.models.image.winclip.torch_model.WinClipModel property*), 166
Patchcore (*class in anomalib.models.image.patchcore.lightning_model*), 142
PatchcoreModel (*class in anomalib.models.image.patchcore.torch_model*), 144
PCA (*class in anomalib.models.components.dimensionality_reduction*), 86
perform_cca() (*anomalib.metrics.AUPRO method*), 186
PoseExtractor (*class in anomalib.models.video.ai_vad.features*), 172
post_process() (*anomalib.deploy.OpenVINOInferencer method*), 204
post_process() (*anomalib.deploy.TorchInferencer method*), 205
post_process_bbox_detections() (*anomalib.models.video.ai_vad.regions.RegionExtractor method*), 174
post_process_box_predictions() (*anomalib.models.image.rkde.region_extractor.RegionExtractor method*), 154
pre_process() (*anomalib.deploy.OpenVINOInferencer method*), 204
pre_process() (*anomalib.deploy.TorchInferencer method*), 206
pre_process() (*anomalib.models.components.classification.KDEClassifier method*), 94
pre_process() (*anomalib.models.video.ai_vad.flow.FlowExtractor method*), 174
predict (*anomalib.cli.AnomalibCLI property*), 201
predict() (*anomalib.deploy.OpenVINOInferencer method*), 204
predict() (*anomalib.deploy.TorchInferencer method*), 206
predict() (*anomalib.engine.Engine method*), 180
predict() (*anomalib.models.components.classification.KDEClassifier method*), 94
predict() (*anomalib.models.components.cluster.GaussianMixture method*), 95
predict() (*anomalib.models.components.cluster.KMeans method*), 96
predict() (*anomalib.models.video.ai_vad.density.BaseDensityEstimator method*), 175
predict() (*anomalib.models.video.ai_vad.density.CombinedDensityEstimator method*), 176
predict() (*anomalib.models.video.ai_vad.density.GMMEstimator method*), 176
predict() (*anomalib.models.video.ai_vad.density.GroupedKNNEstimator method*), 177
predict_dataloader() (*anomalib.data.base.datamodule.AnomalibDataModule method*), 20
prepare_data() (*anomalib.data.depth.mvtec_3d.MVTec3D method*), 58
prepare_data() (*anomalib.data.image.btech.BTech method*), 24
prepare_data() (*anomalib.data.image.kolektor.Kolektor method*), 32
prepare_data() (*anomalib.data.image.mvtec.MVTec method*), 36
prepare_data() (*anomalib.data.image.visa.Visa method*), 40
prepare_data() (*anomalib.data.video.avenue.Avenue method*), 44
prepare_data() (*anomalib.data.video.shanghaitech.ShanghaiTech method*), 48
prepare_data() (*anomalib.data.video.ucsd_ped.UCSDped method*), 51
prepare_imagenette_data() (*anomalib.models.image.efficient_ad.lightning_model.EfficientAd method*), 131
prepare_pretrained_model() (*anomalib.models.image.dsr.lightning_model.Dsr method*), 127
prepare_pretrained_model() (*anomalib.models.image.efficient_ad.lightning_model.EfficientAd method*), 131
PRO (*class in anomalib.metrics*), 191

R

rand_augmenter() (*anomalib.data.utils.augmenter.Augmenter method*), 81
random_2d_perlin() (*in module anomalib.data.utils.generators*), 79
random_split() (*in module anomalib.data.utils.split*), 75
read_depth_image() (*in module anomalib.data.utils.image*), 70
read_image() (*in module anomalib.data.utils.image*), 71
read_mask() (*in module anomalib.data.utils.image*), 71
RegionExtractor (*class in anomalib.models.image.rkde.region_extractor*),

153
RegionExtractor (*class in anomalib.models.video.ai_vad.regions*), 172
reset_distances() (*anomalib.models.components.sampling.KCenterGreedy method*), 91
reshape_embedding() (*anomalib.models.image.patchcore.torch_model.PatchcoreModel static method*), 145
Residual (*class in anomalib.models.image.dsr.torch_model*), 123
ResidualStack (*class in anomalib.models.image.dsr.torch_model*), 123
resolve_path() (*in module anomalib.data.utils.path*), 60
ReverseDistillation (*class in anomalib.models.image.reverse_distillation.lightning_model*), 146
ReverseDistillationLoss (*class in anomalib.models.image.reverse_distillation.loss*), 148
ReverseDistillationModel (*class in anomalib.models.image.reverse_distillation.torch_model*), 147
Rkde (*class in anomalib.models.image.rkde.lightning_model*), 150
RkdeModel (*class in anomalib.models.image.rkde.torch_model*), 151
RoiStage (*class in anomalib.models.image.rkde.region_extractor*), 154

S

safe_extract() (*in module anomalib.data.utils.download*), 65
sample_coreset() (*anomalib.models.components.sampling.KCenterGreedy method*), 91
samples (*anomalib.data.base.dataset.AnomalibDataset property*), 19
samples (*anomalib.data.base.video.AnomalibVideoDataset property*), 22
save() (*anomalib.loggers.AnomalibWandbLogger method*), 196
save_image() (*in module anomalib.data.utils.image*), 72
scale_boxes() (*in module anomalib.data.utils.boxes*), 74
scales (*anomalib.models.image.winclip.torch_model.WinClipModel attribute*), 164
score() (*anomalib.models.image.dfm.torch_model.DFMModel method*), 113
score_samples() (*anomalib.models.components.cluster.GaussianMixture method*), 96
score_samples() (*anomalib.models.image.dfm.torch_model.SingleClassGaussian method*), 114
select_coreset_idxs() (*anomalib.models.components.sampling.KCenterGreedy method*), 92
setup() (*anomalib.callbacks.LoadModelCallback method*), 198
setup() (*anomalib.data.base.datamodule.AnomalibDataModule method*), 20
setup() (*anomalib.models.image.winclip.torch_model.WinClipModel method*), 166
setup_sspcab() (*anomalib.models.image.draem.lightning_model.Draem method*), 115
ShanghaiTech (*class in anomalib.data.video.shanghaitech*), 47
ShanghaiTechDataset (*class in anomalib.data.video.shanghaitech*), 48
ShanghaiTechTestClipsIndexer (*class in anomalib.data.video.shanghaitech*), 49
ShanghaiTechTrainClipsIndexer (*class in anomalib.data.video.shanghaitech*), 49
show_image() (*in module anomalib.data.utils.image*), 72
SingleClassGaussian (*class in anomalib.models.image.dfm.torch_model*), 114
SparseRandomProjection (*class in anomalib.models.components.dimensionality_reduction*), 88
Split (*class in anomalib.data.utils.split*), 75
split_by_label() (*in module anomalib.data.utils.split*), 76
state_dict() (*anomalib.models.image.winclip.lightning_model.WinClip method*), 164
Stfpm (*class in anomalib.models.image.stfpm.lightning_model*), 154
STFPMLoss (*class in anomalib.models.image.stfpm.loss*), 156
STFPMModel (*class in anomalib.models.image.stfpm.torch_model*), 155
StrideSizeError, 76
subcommands() (*anomalib.cli.AnomalibCLI static method*), 201
subsample() (*anomalib.data.base.dataset.AnomalibDataset method*), 19
subsample_embedding() (*anomalib.models.image.patchcore.torch_model.PatchcoreModel*

method), 146
subsample_regions() (*anomalib.models.video.ai_vad.regions.RegionExtractor static method*), 174
SubspaceRestrictionModule (*class in anomalib.models.image.dsr.torch_model*), 124
SubspaceRestrictionNetwork (*class in anomalib.models.image.dsr.torch_model*), 124
SyntheticAnomalyDataset (*class in anomalib.data.utils.synthetic*), 81

T

teacher_channel_mean_std() (*anomalib.models.image.efficient_ad.lightning_model.EfficientAd method*), 131
test (*anomalib.cli.AnomalibCLI property*), 201
test() (*anomalib.engine.Engine method*), 181
test_dataloader() (*anomalib.data.base.datamodule.AnomalibDataModule method*), 20
test_step() (*anomalib.models.image.ganomaly.lightning_model.Ganomaly method*), 137
TestSplitMode (*class in anomalib.data.utils.split*), 75
text_embeddings (*anomalib.models.image.winclip.torch_model.WinClipModel property*), 167
threshold_callback (*anomalib.engine.Engine property*), 182
tile() (*anomalib.data.utils.tiler.Tiler method*), 77
Tiler (*class in anomalib.data.utils.tiler*), 76
TimerCallback (*class in anomalib.callbacks*), 198
TimmFeatureExtractor (*class in anomalib.models.components.feature_extractors*), 82
TorchFXFeatureExtractor (*class in anomalib.models.components.feature_extractors*), 83
TorchInferencer (*class in anomalib.deploy*), 204
train (*anomalib.cli.AnomalibCLI property*), 201
train() (*anomalib.engine.Engine method*), 182
train_dataloader() (*anomalib.data.base.datamodule.AnomalibDataModule method*), 20
train_transform (*anomalib.data.base.datamodule.AnomalibDataModule property*), 20
trainer (*anomalib.engine.Engine property*), 183
trainer_arguments (*anomalib.models.image.cfa.lightning_model.Cfa property*), 100
trainer_arguments (*anomalib.models.image.cflow.lightning_model.Cflow property*), 103
trainer_arguments (*anomalib.models.image.csflow.lightning_model.Csflow property*), 106
trainer_arguments (*anomalib.models.image.dfkde.lightning_model.Dfkde property*), 109
trainer_arguments (*anomalib.models.image.dfm.lightning_model.Dfm property*), 112
trainer_arguments (*anomalib.models.image.draem.lightning_model.Draem property*), 115
trainer_arguments (*anomalib.models.image.dsr.lightning_model.Dsr property*), 127
trainer_arguments (*anomalib.models.image.efficient_ad.lightning_model.EfficientAd property*), 132
trainer_arguments (*anomalib.models.image.fastflow.lightning_model.Fastflow property*), 134
trainer_arguments (*anomalib.models.image.ganomaly.lightning_model.Ganomaly property*), 137
trainer_arguments (*anomalib.models.image.padim.lightning_model.Padim property*), 141
trainer_arguments (*anomalib.models.image.patchcore.lightning_model.Patchcore property*), 143
trainer_arguments (*anomalib.models.image.reverse_distillation.lightning_model.ReverseDis property*), 147
trainer_arguments (*anomalib.models.image.rkde.lightning_model.Rkde property*), 151
trainer_arguments (*anomalib.models.image.stfpm.lightning_model.Stfpm property*), 155
trainer_arguments (*anomalib.models.image.uflow.lightning_model.Uflow property*), 160
trainer_arguments (*anomalib.models.image.winclip.lightning_model.WinClip property*), 164
trainer_arguments (*anomalib.models.video.ai_vad.lightning_model.AiVad property*), 169
training_step() (*anomalib.models.image.cfa.lightning_model.Cfa method*), 100
training_step() (*anomalib.models.image.cflow.lightning_model.Cflow method*), 104

training_step() (*anomalib.models.image.csflow.lightning_model.Csflow method*), 107
training_step() (*anomalib.models.image.dfkde.lightning_model.Dfkde method*), 110
training_step() (*anomalib.models.image.dfm.lightning_model.Dfm method*), 112
training_step() (*anomalib.models.image.draem.lightning_model.Draem method*), 115
training_step() (*anomalib.models.image.dsr.lightning_model.Dsr method*), 127
training_step() (*anomalib.models.image.efficient_ad.lightning_model.EfficientAd method*), 132
training_step() (*anomalib.models.image.fastflow.lightning_model.Fastflow method*), 134
training_step() (*anomalib.models.image.ganomaly.lightning_model.Ganomaly method*), 138
training_step() (*anomalib.models.image.padim.lightning_model.Padim method*), 141
training_step() (*anomalib.models.image.patchcore.lightning_model.Patchcore method*), 143
training_step() (*anomalib.models.image.reverse_distillation.lightning_model.ReverseDistillation method*), 147
training_step() (*anomalib.models.image.rkde.lightning_model.Rkde method*), 151
training_step() (*anomalib.models.image.stfpm.lightning_model.Stfpm method*), 155
training_step() (*anomalib.models.image.uflow.lightning_model.Uflow method*), 160
training_step() (*anomalib.models.video.ai_vad.lightning_model.AiVad method*), 169
transform (*anomalib.data.base.datamodule.AnomalibDataModule property*), 21
transform (*anomalib.models.image.winclip.torch_model.WinClipModel property*), 167
transform() (*anomalib.models.components.dimensionality_reduction.PCA method*), 88
transform() (*anomalib.models.components.dimensionality_reduction.SparseRandomProjection method*), 89

U

UCSDped (*class in anomalib.data.video.ucsd_ped*), 50
UCSDpedClipsIndexer (*class in anomalib.data.video.ucsd_ped*), 51
UCSDpedDataset (*class in anomalib.data.video.ucsd_ped*), 52
Uflow (*class in anomalib.models.image.uflow.lightning_model*), 160
UflowModel (*class in anomalib.models.image.uflow.torch_model*), 159
UnetDecoder (*class in anomalib.models.image.dsr.torch_model*), 124
UnetEncoder (*class in anomalib.models.image.dsr.torch_model*), 125
UnetModel (*class in anomalib.models.image.dsr.torch_model*), 125
untile() (*anomalib.data.utils.tiler.Tiler method*), 77
update() (*anomalib.metrics.AnomalyScoreDistribution method*), 188
update() (*anomalib.metrics.AUPR method*), 185
update() (*anomalib.metrics.AUPRO method*), 187
update() (*anomalib.metrics.AUROC method*), 188
update() (*anomalib.metrics.BinaryPrecisionRecallCurve method*), 188
update() (*anomalib.metrics.ManualThreshold method*), 190
update() (*anomalib.metrics.MinMax method*), 191
update() (*anomalib.metrics.PRO method*), 192
update() (*anomalib.models.video.ai_vad.density.BaseDensityEstimator method*), 175
update() (*anomalib.models.video.ai_vad.density.CombinedDensityEstima method*), 176
update() (*anomalib.models.video.ai_vad.density.GMMEstimator method*), 177
update() (*anomalib.models.video.ai_vad.density.GroupedKNNEstimator method*), 177
update_distances() (*anomalib.models.components.sampling.KCenterGreedy method*), 92
update_to() (*anomalib.data.utils.download.DownloadProgressBar method*), 64
UpsamplingModule (*class in anomalib.models.image.dsr.torch_model*), 126
upscale_image() (*in module anomalib.data.utils.tiler*), 79

V

val_dataloader() (*anomalib.data.base.datamodule.AnomalibDataModule method*), 21

`validate` (*anomalib.cli.AnomalibCLI property*), 202
`validate()` (*anomalib.engine.Engine method*), 183
`validate_and_resolve_path()` (*in module anomalib.data.utils.path*), 60
`validate_path()` (*in module anomalib.data.utils.path*), 61
`validation_step()` (*anomalib.models.image.cfa.lightning_model.Cfa method*), 100
`validation_step()` (*anomalib.models.image.cflow.lightning_model.Cflow method*), 104
`validation_step()` (*anomalib.models.image.csflow.lightning_model.Csflow method*), 107
`validation_step()` (*anomalib.models.image.dfkde.lightning_model.Dfkde method*), 110
`validation_step()` (*anomalib.models.image.dfm.lightning_model.Dfm method*), 112
`validation_step()` (*anomalib.models.image.draem.lightning_model.Draem method*), 115
`validation_step()` (*anomalib.models.image.dsr.lightning_model.Dsr method*), 128
`validation_step()` (*anomalib.models.image.efficient_ad.lightning_model.EfficientAd method*), 132
`validation_step()` (*anomalib.models.image.fastflow.lightning_model.Fastflow method*), 134
`validation_step()` (*anomalib.models.image.ganomaly.lightning_model.Ganomaly method*), 138
`validation_step()` (*anomalib.models.image.padim.lightning_model.Padim method*), 141
`validation_step()` (*anomalib.models.image.patchcore.lightning_model.Patchcore method*), 144
`validation_step()` (*anomalib.models.image.reverse_distillation.lightning_model.ReverseDistillation method*), 147
`validation_step()` (*anomalib.models.image.rkde.lightning_model.Rkde method*), 151
`validation_step()` (*anomalib.models.image.stfpm.lightning_model.Stfpm method*), 155
`validation_step()` (*anomalib.models.image.uflow.lightning_model.Uflow method*), 161
`validation_step()` (*anomalib.models.image.winclip.lightning_model.WinClip method*), 164
`validation_step()` (*anomalib.models.video.ai_vad.lightning_model.AiVad method*), 169
`ValSplitMode` (*class in anomalib.data.utils.split*), 75
`VectorQuantizer` (*class in anomalib.models.image.dsr.torch_model*), 126
`VelocityExtractor` (*class in anomalib.models.video.ai_vad.features*), 172
`VideoTargetFrame` (*class in anomalib.data.base.video*), 22
`Visa` (*class in anomalib.data.image.visa*), 39
`VisaDataset` (*class in anomalib.data.image.visa*), 40
`visual_embeddings` (*anomalib.models.image.winclip.torch_model.WinClipModel property*), 167
`vq_vae_bot` (*anomalib.models.image.dsr.torch_model.DiscreteLatentMode property*), 119
`vq_vae_top` (*anomalib.models.image.dsr.torch_model.DiscreteLatentMode property*), 119

W

`weights_init()` (*anomalib.models.image.ganomaly.torch_model.GanomalyModel static method*), 139
`WinClip` (*class in anomalib.models.image.winclip.lightning_model*), 163
`WinClipModel` (*class in anomalib.models.image.winclip.torch_model*), 164