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

```python
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:

```python
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).

```python
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.