Image and Video Utils

Contents

- Path Utils
- Download Utils
- Image Utils
- Video Utils
- Label Utils
- Bounding Box Utils
- Dataset Split 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.p
False
```

```
>>> contains_non_printable_characters("./datasets/MVTec/bottle/train/good/000.p
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"
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="./dat
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]

[default: 1].

- **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.sp
>>>         urllib.request.urlretrieve(url, filename=output_path, reporthook=p_
```

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`]

Previous
Data Utils

Next
Data Transforms