Tiling

Contents

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
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=
(512, 512)
```

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

Previous
Data Transforms

Next
Synthetic Data Utils