BTech Data

Contents

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:

Back to top

- **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
>>> datamodule = BTech(
...     root="./datasets/BTech",
...     category="01",
...     image_size=256,
```

Back to top

```
...     train_batch_size=32,
...     eval_batch_size=32,
...     num_workers=8,
...     transform_config_train=None,
...     transform_config_eval=None,
... )
>>> 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

Back to top

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:

Back to top

- **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'])
```

Back to top

```
>>> 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
0  BTech/01 train 01    BTech/01/train/ok/105.bmp BTech/01/ground_truth/ok/105.
1  BTech/01 train 01    BTech/01/train/ok/017.bmp BTech/01/ground_truth/ok/017.
...
```

Returns:

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

Return type:

DataFrame

Back to top

Previous
Image Data

Next
Folder Data

Back to top