103-folder

March 29, 2024

1 Use `Folder` for Customs Datasets

2 Installing Anomalib

The easiest way to install anomalib is to use pip. You can install it from the command line using the following command:

[ ]:
```
%pip install anomalib
```

2.1 Setting up the Dataset Directory

This cell is to ensure we change the directory to have access to the datasets.

[1]:
```
from pathlib import Path

# NOTE: Provide the path to the dataset root directory.
#   If the datasets is not downloaded, it will be downloaded
#   to this directory.
dataset_root = Path.cwd().parent / "datasets" / "hazelnut_toy"
```

2.2 Use Folder Dataset (for Custom Datasets) via API

Here we show how one can utilize custom datasets to train anomalib models. A custom dataset in this model can be of the following types:

- A dataset with good and bad images.
- A dataset with good and bad images as well as mask ground-truths for pixel-wise evaluation.
- A dataset with good and bad images that is already split into training and testing sets.

To experiment this setting we provide a toy dataset that could be downloaded from the following https://github.com/openvinotoolkit/anomalib/blob/main/docs/source/data/hazelnut_toy.zip. For the rest of the tutorial, we assume that the dataset is downloaded and extracted to `../datasets`, located in the `anomalib` directory.

[2]:
```
# flake8: noqa
import numpy as np
from PIL import Image
from torchvision.transforms.v2 import Resize
from torchvision.transforms.v2.functional import to_pil_image
```

```
from anomalib.data.image.folder import Folder, FolderDataset
from anomalib import TaskType
```

2.2.1 DataModule

Similar to how we created the datamodules for existing benchmarking datasets in the previous tutorials, we can also create an Anomalib datamodule for our custom hazelnut dataset.

In addition to the root folder of the dataset, we now also specify which folder contains the normal images, which folder contains the anomalous images, and which folder contains the ground truth masks for the anomalous images.

[3]:
```
folder_datamodule = Folder(
    name="hazelnut_toy",
    root=dataset_root,
    normal_dir="good",
    abnormal_dir="crack",
    task=TaskType.SEGMENTATION,
    mask_dir=dataset_root / "mask" / "crack",
    image_size=(256, 256),
)
folder_datamodule.setup()
```

[4]:
```
# Train images
i, data = next(enumerate(folder_datamodule.train_dataloader()))
print(data.keys(), data["image"].shape)
```

```
dict_keys(['image_path', 'label', 'image', 'mask']) torch.Size([28, 3, 256,
256])
```

[5]:
```
# Test images
i, data = next(enumerate(folder_datamodule.test_dataloader()))
print(data.keys(), data["image"].shape, data["mask"].shape)
```

```
dict_keys(['image_path', 'label', 'image', 'mask']) torch.Size([6, 3, 256, 256])
torch.Size([6, 256, 256])
```

As can be seen above, creating the dataloaders are pretty straghtforward, which could be directly used for training/testing/inference. We could visualize samples from the dataloaders as well.

[6]:
```
img = to_pil_image(data["image"][0].clone())
msk = to_pil_image(data["mask"][0]).convert("RGB")

Image.fromarray(np.hstack((np.array(img), np.array(msk))))
```

[6]:

`Folder` data module offers much more flexibility cater all different sorts of needs. Please refer to the documentation for more details.

2.2.2 Torch Dataset

As in earlier examples, we can also create a standalone PyTorch dataset instance.

[ ]:
```
FolderDataset??
```

We can add some transforms that will be applied to the images using torchvision. Let's add a transform that resizes the input image to 256x256 pixels.

[7]:
```
image_size = (256, 256)
transform = Resize(image_size, antialias=True)
```

Classification Task

[8]:
```
folder_dataset_classification_train = FolderDataset(
    name="hazelnut_toy",
    normal_dir=dataset_root / "good",
    abnormal_dir=dataset_root / "crack",
    split="train",
    transform=transform,
    task=TaskType.CLASSIFICATION,
)
folder_dataset_classification_train.samples.head()
```

[8]:

	image_path	label	\
0	/home/djameln/datasets/hazelnut_toy/good/00.jpg	DirType.NORMAL	
1	/home/djameln/datasets/hazelnut_toy/good/01.jpg	DirType.NORMAL	

```
2  /home/djameln/datasets/hazelnut_toy/good/02.jpg  DirType.NORMAL
3  /home/djameln/datasets/hazelnut_toy/good/03.jpg  DirType.NORMAL
4  /home/djameln/datasets/hazelnut_toy/good/04.jpg  DirType.NORMAL

   label_index mask_path        split
0            0            Split.TRAIN
1            0            Split.TRAIN
2            0            Split.TRAIN
3            0            Split.TRAIN
4            0            Split.TRAIN
```

Let's look at the first sample in the dataset.

[9]:
```
data = folder_dataset_classification_train[0]
print(data.keys(), data["image"].shape)
```

```
dict_keys(['image_path', 'label', 'image']) torch.Size([3, 256, 256])
```

As can be seen above, when we choose `classification` task and `train` split, the dataset only returns `image`. This is mainly because training only requires normal images and no labels. Now let's try `test` split for the `classification` task

[10]:
```
# Folder Classification Test Set
folder_dataset_classification_test = FolderDataset(
    name="hazelnut_toy",
    normal_dir=dataset_root / "good",
    abnormal_dir=dataset_root / "crack",
    split="test",
    transform=transform,
    task=TaskType.CLASSIFICATION,
)
folder_dataset_classification_test.samples.head()
```

[10]:
```
                                        image_path             label  \
0  /home/djameln/datasets/hazelnut_toy/crack/01.jpg  DirType.ABNORMAL
1  /home/djameln/datasets/hazelnut_toy/crack/02.jpg  DirType.ABNORMAL
2  /home/djameln/datasets/hazelnut_toy/crack/03.jpg  DirType.ABNORMAL
3  /home/djameln/datasets/hazelnut_toy/crack/04.jpg  DirType.ABNORMAL
4  /home/djameln/datasets/hazelnut_toy/crack/05.jpg  DirType.ABNORMAL

   label_index mask_path       split
0            1            Split.TEST
1            1            Split.TEST
2            1            Split.TEST
3            1            Split.TEST
4            1            Split.TEST
```

[11]:
```
data = folder_dataset_classification_test[0]
print(data.keys(), data["image"].shape, data["image_path"], data["label"])
```

```
dict_keys(['image_path', 'label', 'image']) torch.Size([3, 256, 256])
/home/djameln/datasets/hazelnut_toy/crack/01.jpg 1
```

Segmentation Task It is also possible to configure the Folder dataset for the segmentation task, where the dataset object returns image and ground-truth mask.

[12]:
```
# Folder Segmentation Train Set
folder_dataset_segmentation_train = FolderDataset(
    name="hazelnut_toy",
    normal_dir=dataset_root / "good",
    abnormal_dir=dataset_root / "crack",
    split="train",
    transform=transform,
    mask_dir=dataset_root / "mask" / "crack",
    task=TaskType.SEGMENTATION,
)
folder_dataset_segmentation_train.samples.head()
```

[12]:

	image_path	label	label_index	mask_path	split
0	/home/djameln/datasets/hazelnut_toy/good/00.jpg	DirType.NORMAL	0		Split.TRAIN
1	/home/djameln/datasets/hazelnut_toy/good/01.jpg	DirType.NORMAL	0		Split.TRAIN
2	/home/djameln/datasets/hazelnut_toy/good/02.jpg	DirType.NORMAL	0		Split.TRAIN
3	/home/djameln/datasets/hazelnut_toy/good/03.jpg	DirType.NORMAL	0		Split.TRAIN
4	/home/djameln/datasets/hazelnut_toy/good/04.jpg	DirType.NORMAL	0		Split.TRAIN

[13]:
```
# Folder Segmentation Test Set
folder_dataset_segmentation_test = FolderDataset(
    name="hazelnut_toy",
    normal_dir=dataset_root / "good",
    abnormal_dir=dataset_root / "crack",
    split="test",
    transform=transform,
    mask_dir=dataset_root / "mask" / "crack",
    task=TaskType.SEGMENTATION,
)
folder_dataset_segmentation_test.samples.head(10)
```

[13]:

	image_path	label	label_index	mask_path	split
0	/home/djameln/datasets/hazelnut_toy/crack/01.jpg	DirType.ABNORMAL	1	/home/djameln/datasets/hazelnut_toy/mask/crack…	Split.TEST
1	/home/djameln/datasets/hazelnut_toy/crack/02.jpg	DirType.ABNORMAL	1	/home/djameln/datasets/hazelnut_toy/mask/crack…	Split.TEST
2	/home/djameln/datasets/hazelnut_toy/crack/03.jpg	DirType.ABNORMAL	1	/home/djameln/datasets/hazelnut_toy/mask/crack…	Split.TEST
3	/home/djameln/datasets/hazelnut_toy/crack/04.jpg	DirType.ABNORMAL	1	/home/djameln/datasets/hazelnut_toy/mask/crack…	Split.TEST
4	/home/djameln/datasets/hazelnut_toy/crack/05.jpg	DirType.ABNORMAL	1	/home/djameln/datasets/hazelnut_toy/mask/crack…	Split.TEST

[14]:
```
data = folder_dataset_segmentation_test[3]
print(data.keys(), data["image"].shape, data["mask"].shape)
```

```
dict_keys(['image_path', 'label', 'image', 'mask']) torch.Size([3, 256, 256])
torch.Size([256, 256])
```

Let's visualize the image and the mask…

[15]:
```
img = to_pil_image(data["image"].clone())
msk = to_pil_image(data["mask"]).convert("RGB")

Image.fromarray(np.hstack((np.array(img), np.array(msk))))
```

[15]: