Custom Data

Contents

This tutorial will show you how to train anomalib models on your custom data. More specifically, we will show you how to use the Folder dataset to train anomalib models on your custom data.

> **Warning**
>
> This tutorial assumes that you have already installed anomalib. If not, please refer to the installation section.

> **Note**
>
> We will use our hazelnut_toy dataset to show the capabilities of the Folder dataset, but you can use any dataset you want.

We will split the section to two tasks: Classification and Segmentation.

Classification Dataset

In certain use-cases, ground-truth masks for the abnormal images may not be available. In such cases, we could use the classification task to train a model that will be able to detect the abnormal images in the test set.

We will split this section into two tasks:

- Classification with normal and abnormal images, and

- Classification with only normal images.

With Normal and Abnormal Images

We could use Folder datamodule to train a model on this dataset. We could run the following python code to create the custom datamodule:

Code Syntax

API CLI

```
# Import the datamodule
from anomalib.data import Folder

# Create the datamodule
datamodule = Folder(
    name="hazelnut_toy",
    root="datasets/hazelnut_toy",
    normal_dir="good",
    abnormal_dir="crack",
    task="classification",
)

# Setup the datamodule
datamodule.setup()
```

Note

As can be seen above, we only need to specify the `task` argument to `classification`. We could have also use `TaskType.CLASSIFICATION` instead of `classification`.

The Folder datamodule will create training, validation, test and prediction datasets and dataloaders for us. We can access the datasets and dataloaders by following the same approach as in the segmentation task.

When we check the samples from the dataloaders, we will see that the `mask` key is not present in the samples. This is because we do not need the masks for the classification task.

```
i, train_data = next(enumerate(datamodule.train_dataloader()))
print(train_data.keys())
# dict_keys(['image_path', 'label', 'image'])

i, val_data = next(enumerate(datamodule.val_dataloader()))
print(val_data.keys())
# dict_keys(['image_path', 'label', 'image'])

i, test_data = next(enumerate(datamodule.test_dataloader()))
print(test_data.keys())
# dict_keys(['image_path', 'label', 'image'])
```

Training the model is as simple as running the following command:

```
# Import the model and engine
from anomalib.models import Patchcore
from anomalib.engine import Engine

# Create the model and engine
model = Patchcore()
engine = Engine(task="classification")

# Train a Patchcore model on the given datamodule
engine.train(datamodule=datamodule, model=model)
```

where we train a Patchcore model on this custom dataset with default model parameters.

With Only Normal Images

There are certain cases where we only have normal images in our dataset but would like to train a classification model.

This could be done in two ways:

- Train the model and skip the validation and test steps, as we do not have abnormal images to validate and test the model on, or
- Use the synthetic anomaly generation feature to create abnormal images from normal images, and perform the validation and test steps.

For now we will focus on the second approach.

With Validation and Testing via Synthetic Anomalies

If we want to check the performance of the model, we will need to have abnormal images to validate and test the model on. During the validation stage, these anomalous images are used to normalize the anomaly scores and find the best threshold that separates normal and abnormal images.

Anomalib provides synthetic anomaly generation capabilities to create abnormal images from normal images so we could check the performance. We could use the Folder datamodule to train a model on this dataset.

Code Syntax

API CLI

```
# Import the datamodule
from anomalib.data import Folder
from anomalib.data.utils import TestSplitMode

# Create the datamodule
datamodule = Folder(
    name="hazelnut_toy",
    root="datasets/hazelnut_toy",
    normal_dir="good",
    test_split_mode=TestSplitMode.SYNTHETIC,
    task="classification",
)

# Setup the datamodule
datamodule.setup()
```

Once the datamodule is setup, the rest of the process is the same as in the previous classification example.

```
# Import the model and engine
from anomalib.models import Patchcore
from anomalib.engine import Engine

# Create the model and engine
model = Patchcore()
```

```
engine = Engine(task="classification")

# Train a Patchcore model on the given datamodule
engine.train(datamodule=datamodule, model=model)
```

where we train a Patchcore model on this custom dataset with default model parameters.

Segmentation Dataset

Assume that we have a dataset in which the training set contains only normal images, and the test set contains both normal and abnormal images. We also have masks for the abnormal images in the test set. We want to train an anomaly segmentation model that will be able to detect the abnormal regions in the test set.

With Normal and Abnormal Images

We could use Folder datamodule to load the hazelnut dataset in a format that is readable by Anomalib's models.

Code Syntax

API **CLI**

We could run the following python code to create the custom datamodule:

```
# Import the datamodule
from anomalib.data import Folder

# Create the datamodule
datamodule = Folder(
    name="hazelnut_toy",
    root="datasets/hazelnut_toy",
    normal_dir="good",
    abnormal_dir="crack",
    mask_dir="mask/crack",
    normal_split_ratio=0.2,
)
```

```
# Setup the datamodule
datamodule.setup()
```

The Folder datamodule will create training, validation, test and prediction datasets and dataloaders for us. We can access the datasets and dataloaders using the following attributes:

```
# Access the datasets
train_dataset = datamodule.train_data
val_dataset = datamodule.val_data
test_dataset = datamodule.test_data

# Access the dataloaders
train_dataloader = datamodule.train_dataloader()
val_dataloader = datamodule.val_dataloader()
test_dataloader = datamodule.test_dataloader()
```

To check what individual samples from dataloaders look like, we can run the following command:

```
i, train_data = next(enumerate(datamodule.train_dataloader()))
print(train_data.keys())
# dict_keys(['image_path', 'label', 'image', 'mask_path', 'mask'])

i, val_data = next(enumerate(datamodule.val_dataloader()))
print(val_data.keys())
# dict_keys(['image_path', 'label', 'image', 'mask_path', 'mask'])

i, test_data = next(enumerate(datamodule.test_dataloader()))
print(test_data.keys())
# dict_keys(['image_path', 'label', 'image', 'mask_path', 'mask'])
```

We could check the shape of the images and masks using the following commands:

```
print(train_data["image"].shape)
# torch.Size([32, 3, 256, 256])

print(train_data["mask"].shape)
# torch.Size([32, 256, 256])
```

Training the model is as simple as running the following command:

```
# Import the model and engine
from anomalib.models import Patchcore
from anomalib.engine import Engine

# Create the model and engine
model = Patchcore()
engine = Engine()

# Train a Patchcore model on the given datamodule
engine.train(datamodule=datamodule, model=model)
```

where we train a Patchcore model on this custom dataset with default model parameters.

This example demonstrates how to create a segmentation dataset with normal and abnormal images. We could expand this example to create a segmentation dataset with only normal images.

With Only Normal Images

There are certain cases where we only have normal images in our dataset but would like to train a segmentation model. This could be done in two ways:

- Train the model and skip the validation and test steps, as we do not have abnormal images to validate and test the model on, or
- Use the synthetic anomaly generation feature to create abnormal images from normal images, and perform the validation and test steps.

For now we will focus on the second approach.

With Validation and Testing via Synthetic Anomalies

We could use the synthetic anomaly generation feature again to create abnormal images from normal images. We could then use the Folder datamodule to train a model on this dataset. Here is the python code to create the custom datamodule:

Code Syntax

API CLI

We could run the following python code to create the custom datamodule:

```
# Import the datamodule
from anomalib.data import Folder
from anomalib.data.utils import TestSplitMode

# Create the datamodule
datamodule = Folder(
    name="hazelnut_toy",
    root="datasets/hazelnut_toy",
    normal_dir="good",
    test_split_mode=TestSplitMode.SYNTHETIC,
)

# Setup the datamodule
datamodule.setup()
```

As can be seen from the code above, we only need to specify the `test_split_mode` argument to `SYNTHETIC`. The Folder datamodule will create training, validation, test and prediction datasets and dataloaders for us.

To check what individual samples from dataloaders look like, we can run the following command:

```
i, train_data = next(enumerate(datamodule.train_dataloader()))
print(train_data.keys())
# dict_keys(['image_path', 'label', 'image', 'mask_path', 'mask'])

i, val_data = next(enumerate(datamodule.val_dataloader()))
print(val_data.keys())
# dict_keys(['image_path', 'label', 'image', 'mask_path', 'mask'])

i, test_data = next(enumerate(datamodule.test_dataloader()))
print(test_data.keys())
# dict_keys(['image_path', 'label', 'image', 'mask_path', 'mask'])
```

We could check the shape of the images and masks using the following commands:

```
print(train_data["image"].shape)
```

```
# torch.Size([32, 3, 256, 256])

print(train_data["mask"].shape)
# torch.Size([32, 256, 256])
```

Previous
Data Tutorials

Next
Model Tutorials