Feature extractors

Contents

This guide demonstrates how different backbones can be used as feature extractors for anomaly detection models. Most of these models use Timm Feature Extractor except **CSFLOW** which uses TorchFx Feature Extractor. Here we show how to use API and CLI to use different backbones as feature extractors.

> **See also**
>
> For specifics of implementation refer to implementation classes `Timm Feature Extractor` and `TorchFx Feature Extractor`

Available backbones and layers

Timm **TorchFX**

When using TorchFX for feature extraction, you can use either model name, custom model, or instance of model. In this guide, we will cover pretrained models from Torchvision passed by name. For use of the custom model or instance of a model refer to `TorchFxFeatureExtractor class examples`.

Available torchvision models are listed on Torchvision models page.

We can get layer names for selected m[illegible]he following code:

Back to top

```
# Import model and function to list names
from torchvision.models import resnet18
```

```
from torchvision.models.feature_extraction import get_graph_node_names

# Make an instance of model with default (latest) weights
model = resnet18(weights="DEFAULT")

# Get and print node (layer) names
train_nodes, eval_nodes = get_graph_node_names(model)
print(eval_nodes)
>>>['x', 'conv1', 'bn1', 'relu', 'maxpool', ..., 'layer4.1.relu_1', 'avgpool', 'fla
```

As a result, we get a list of all model nodes, which is quite long.

Now for example, if we want only output from the last node in the block named `layer4`, we specify `layer4.1.relu_1`. If we want to avoid writing `layer4.1.relu_1` to get the last output of `layer4` block, we can shorten it to `layer4`.

We can then use selected model name and layer names with either API or using config file.

See also

Additional info about TorchFX feature extraction can be found on PyTorch FX page and feature_extraction documentation page.

Warning

Some models might not support every backbone.

Backbone and layer selection

API CLI

When using API, we need to specify `backbone` and `layers` when instantiating the model with a non-default backbone.

```
1 # Import the required modules
2 from anomalib.data import MVT
3 from anomalib.models import Padim
4 from anomalib.engine import Engine
5
```

Back to top

```python
# Initialize the datamodule, model, and engine
datamodule = MVTec(num_workers=0)
# Specify backbone and layers
model = Padim(backbone="resnet18", layers=["layer1", "layer2"])
engine = Engine(image_metrics=["AUROC"], pixel_metrics=["AUROC"])

# Train the model
engine.fit(datamodule=datamodule, model=model)
```

Previous
Model Tutorials

Next
Topic Guide

Back to top