# Feature Extractors

Print to PDF

## Contents

Feature extractors.

*class* **anomalib.models.components.feature_extractors.BackboneParams**(*class_path*, *init_args=<factory>*)

Bases: `object`

Used for serializing the backbone.

*class* **anomalib.models.components.feature_extractors.TimmFeatureExtractor**(*backbone*, *layers*, *pre_trained=True*, *requires_grad=False*)

Bases: `Module`

Extract features from a CNN.

**Parameters:**

- **backbone** (*nn.Module*) – The backbone to which the feature extraction hooks are attached.
- **layers** (*Iterable[str]*) – List of layer names of the backbone to which the hooks are attached.
- **pre_trained** (*bool*) – Whether to use a pre-trained backbone. Defaults to True.
- **requires_grad** (*bool*) – Whether to require gradients for the backbone. Defaults to False. Models like `stfpm` use the feature extractor model as a trainable network. In such cases gradient computation is required.

**Example**

```
import torch
from anomalib.models.components.feature_extractors import TimmFeatureExtractor

model = TimmFeatureExtractor(model="resnet18", layers=['layer1', 'layer2', 'layer3'])
input = torch.rand((32, 3, 256, 256))
features = model(input)

print([layer for layer in features.keys()])
```

```
# Output: ['layer1', 'layer2', 'layer3']

print([feature.shape for feature in features.values()]()
# Output: [torch.Size([32, 64, 64, 64]), torch.Size([32, 128, 32, 32]), torch.Size([32, 256, 16, 16])
```

**forward**(*inputs*)

Forward-pass input tensor into the CNN.

**Parameters:**

**inputs** (*torch.Tensor*) – Input tensor

**Return type:**

`dict`[`str`, `Tensor`]

**Returns:**

Feature map extracted from the CNN

**Example**

```
model = TimmFeatureExtractor(model="resnet50", layers=['layer3'])
input = torch.rand((32, 3, 256, 256))
features = model.forward(input)
```

*class* **anomalib.models.components.feature_extractors.TorchFXFeatureExtractor**(*backbone*, *return_nodes*, *weights=None*, *requires_grad=False*, *tracer_kwargs=None*)

Bases: `Module`

Extract features from a CNN.

**Parameters:**

- **backbone** (*str* | *BackboneParams* | *dict* | *nn.Module*) – The backbone to which the feature extraction hooks are attached. If the name is provided, the model is loaded from torchvision. Otherwise, the model class can be provided and it will try to load the weights from the provided weights file. Last, an instance of nn.Module can also be passed directly.
- **return_nodes** (*Iterable[str]*) – List of layer names of the backbone to which the hooks are attached. You can find the names of these nodes by using `get_graph_node_names` function.
- **weights** (*str* | *WeightsEnum* | *None*) – Weights enum to use for the model. Torchvision models require `WeightsEnum`. These enums are defined in `torchvision.models.<model>`. You can pass the weights path for custom models.
- **requires_grad** (*bool*) – Models like `stfpm` use the feature extractor for training. In such cases we should set `requires_grad` to `True`. Default is `False`.
- **tracer_kwargs** (*dict* | *None*) – a dictionary of keyword arguments for NodePathTracer (which passes them onto it's parent class torch.fx.Tracer). Can be used to allow not tracing through a list of problematic modules, by passing a list of *leaf_modules* as one of the *tracer_kwargs*.

### Example

With torchvision models:

```python
import torch
from anomalib.models.components.feature_extractors import TorchFXFeatureExtractor
from torchvision.models.efficientnet import EfficientNet_B5_Weights

feature_extractor = TorchFXFeatureExtractor(
    backbone="efficientnet_b5",
    return_nodes=["features.6.8"],
    weights=EfficientNet_B5_Weights.DEFAULT
)

input = torch.rand((32, 3, 256, 256))
features = feature_extractor(input)

print([layer for layer in features.keys()])
# Output: ["features.6.8"]

print([feature.shape for feature in features.values()])
# Output: [torch.Size([32, 304, 8, 8])]
```

With custom models:

```python
import torch
from anomalib.models.components.feature_extractors import TorchFXFeatureExtractor

feature_extractor = TorchFXFeatureExtractor(
    "path.to.CustomModel", ["linear_relu_stack.3"], weights="path/to/weights.pth"
)

input = torch.randn(1, 1, 28, 28)
features = feature_extractor(input)

print([layer for layer in features.keys()])
# Output: ["linear_relu_stack.3"]
```

with model instances:

```python
import torch
from anomalib.models.components.feature_extractors import TorchFXFeatureExtractor
from timm import create_model

model = create_model("resnet18", pretrained=True)
feature_extractor = TorchFXFeatureExtractor(model, ["layer1"])

input = torch.rand((32, 3, 256, 256))
features = feature_extractor(input)

print([layer for layer in features.keys()])
# Output: ["layer1"]

print([feature.shape for feature in features.values()])
# Output: [torch.Size([32, 64, 64, 64])]
```

**forward**(*inputs*)

Extract features from the input.

**Return type:**

`dict`[`str`, `Tensor`]

`initialize_feature_extractor`(*backbone*, *return_nodes*, *weights*=None, *requires_grad*=False, *tracer_kwargs*=None)

Extract features from a CNN.

**Parameters:**

- **backbone** (*BackboneParams* | *nn.Module*) – The backbone to which the feature extraction hooks are attached. If the name is provided for BackboneParams, the model is loaded from torchvision. Otherwise, the model class can be provided and it will try to load the weights from the provided weights file. Last, an instance of the model can be provided as well, which will be used as-is.
- **return_nodes** (*Iterable[str]*) – List of layer names of the backbone to which the hooks are attached. You can find the names of these nodes by using `get_graph_node_names` function.
- **weights** (*str* | *WeightsEnum* | *None*) – Weights enum to use for the model. Torchvision models require `WeightsEnum`. These enums are defined in `torchvision.models.<model>`. You can pass the weights path for custom models.
- **requires_grad** (*bool*) – Models like `stfpm` use the feature extractor for training. In such cases we should set `requires_grad` to `True`. Default is `False`.
- **tracer_kwargs** (*dict* | *None*) – a dictionary of keyword arguments for NodePathTracer (which passes them onto it's parent class torch.fx.Tracer). Can be used to allow not tracing through a list of problematic modules, by passing a list of *leaf_modules* as one of the *tracer_kwargs*.

**Return type:**

`GraphModule`

**Returns:**

Feature Extractor based on TorchFX.

`anomalib.models.components.feature_extractors.dryrun_find_featuremap_dims`(*feature_extractor*, *input_size*, *layers*)

Dry run an empty image of *input_size* size to get the featuremap tensors' dimensions (num_features, resolution).

**Returns:**

**maping of*layer* -> *dimensions dict***

Each *dimension dict* has two keys: *num_features* (int) and `resolution`(tuple[int, int]).

**Return type:**

tuple[int, int]

Previous
Model Components

Next
Dimensionality Reduction