C-Flow

Contents

Cflow.

Real-Time Unsupervised Anomaly Detection via Conditional Normalizing Flows.

For more details, see the paper: Real-Time Unsupervised Anomaly Detection via Conditional Normalizing Flows.

```
class
anomalib.models.image.cflow.lightning_model.Cflow(backbone='wide_resnet50_2',
layers=('layer2', 'layer3', 'layer4'), pre_trained=True,
fiber_batch_size=64, decoder='freia-cflow', condition_vector=128,
coupling_blocks=8, clamp_alpha=1.9, permute_soft=False, lr=0.0001)
```

Bases: `AnomalyModule`

PL Lightning Module for the CFLOW algorithm.

Parameters:

- **backbone** (*str, optional*) – Backbone CNN architecture. Defaults to `"wide_resnet50_2"`.
- **layers** (*Sequence[str], optional*) – Layers to extract features from. Defaults to `("layer2", "layer3", "layer4")`.
- **pre_trained** (*bool, optional*) – Whether to use pre-trained weights. Defaults to `True`.
- **fiber_batch_size** (*int, optional*) – Fiber batch size. Defaults to `64`.
- **decoder** (*str, optional*) – Decoder architecture. Defaults to `"freia-cflow"`.
- **condition_vector** (*int, optional*) – Condition vector size. Defaults to `128`.
- **coupling_blocks** (*int, optional*) – Number of coupling blocks. Defaults to `8`.
- **clamp_alpha** (*float, optional*) – Clamping value for the alpha parameter. Defaults to `1.9`.
- **permute_soft** (*bool, optional*) – Whether to use soft permutation. Defaults to `False`.
- **lr** (*float, optional*) – Learning rate. Defaults to `0.0001`.

`configure_optimizers()`

Configure optimizers for each decoder.

Returns:

Adam optimizer for each decoder

Return type:

Optimizer

property `learning_type`*: LearningType*

Return the learning type of the model.

Returns:

Learning type of the model.

Return type:

LearningType

property `trainer_arguments`*: dict[str, Any]*

C-FLOW specific trainer arguments.

`training_step(batch, *args, **kwargs)`

Perform the training step of CFLOW.

For each batch, decoder layers are trained with a dynamic fiber batch size. Training step is performed manually as multiple training steps are involved

> per batch of input images

Parameters:

- **batch** (*dict[str, str | torch.Tensor]*) – Input batch
- ***args** – Arguments.
- ****kwargs** – Keyword arguments.

Return type:

`Union`[`Tensor`, `Mapping`[`str`, `Any`], `None`]

Returns:

Loss value for the batch

`validation_step(batch, *args, **kwargs)`

Perform the validation step of CFLOW.

> Similar to the training step, encoder features are extracted from the CNN for each batch, and anomaly map is computed.

Parameters:

- **batch** (*dict[str, str | torch.Tensor]*) – Input batch
- ***args** – Arguments.
- ****kwargs** – Keyword arguments.

Return type:

`Union`[`Tensor`, `Mapping`[`str`, `Any`], `None`]

Returns:

Dictionary containing images, anomaly maps, true labels and masks. These are required in *validation_epoch_end* for feature concatenation.

PyTorch model for CFlow model implementation.

```
class anomalib.models.image.cflow.torch_model.CflowModel(backbone, layers,
pre_trained=True, fiber_batch_size=64, decoder='freia-cflow',
condition_vector=128, coupling_blocks=8, clamp_alpha=1.9,
permute_soft=False)
```

Bases: `Module`

CFLOW: Conditional Normalizing Flows.

Parameters:

- **backbone** (*str*) – Backbone CNN architecture.
- **layers** (*Sequence[str]*) – Layers to extract features from.
- **pre_trained** (*bool*) – Whether to use pre-trained weights. Defaults to `True`.
- **fiber_batch_size** (*int*) – Fiber batch size. Defaults to `64`.
- **decoder** (*str*) – Decoder architecture. Defaults to `"freia-cflow"`.
- **condition_vector** (*int*) – Condition vector size. Defaults to `128`.
- **coupling_blocks** (*int*) – Number of coupling blocks. Defaults to `8`.
- **clamp_alpha** (*float*) – Clamping value for the alpha parameter. Defaults to `1.9`.
- **permute_soft** (*bool*) – Whether to use soft permutation. Defaults to `False`.

```
forward(images)
```

Forward-pass images into the network to extract encoder features and compute probability.

Parameters:

images (`Tensor`) – Batch of images.

Return type:

`Tensor`

Returns:

Predicted anomaly maps.

Anomaly Map Generator for CFlow model implementation.

class `anomalib.models.image.cflow.anomaly_map.`**`AnomalyMapGenerator`**`(`*`pool_layers`*`)`

Bases: `Module`

Generate Anomaly Heatmap.

`compute_anomaly_map``(`*`distribution, height, width, image_size`*`)`

Compute the layer map based on likelihood estimation.

Parameters:

- **distribution** (*list[torch.Tensor]*) – List of likelihoods for each layer.
- **height** (*list[int]*) – List of heights of the feature maps.
- **width** (*list[int]*) – List of widths of the feature maps.
- **image_size** (*tuple[int, int]* | *torch.Size* | *None*) – Size of the input image.

Return type:

`Tensor`

Returns:

Final Anomaly Map

`forward``(`*`**kwargs`*`)`

Return anomaly_map.

Expects *distribution*, *height* and 'width' keywords to be passed explicitly

Example

```
>>> anomaly_map_generator = AnomalyMapGenerator(image_size=tuple(hparams.mode
>>>         pool_layers=pool_layers)
>>> output = self.anomaly_map_generator(distribution=dist, height=height, wic
```

Raises:

ValueError – *distribution*, *height* and 'width' keys are not found

Returns:

anomaly map

Return type:

torch.Tensor

Previous
CFA

Next
CS-Flow