Reverse Distillation

Contents

Anomaly Detection via Reverse Distillation from One-Class Embedding.

https://arxiv.org/abs/2201.10703v2

class **anomalib.models.image.reverse_distillation.lightning_model.ReverseDistillation**(*backbone='wide_resnet50_2'*, *layers=('layer1', 'layer2', 'layer3')*, *anomaly_map_mode=AnomalyMapGenerationMode.ADD*, *pre_trained=True*)

Bases: `AnomalyModule`

PL Lightning Module for Reverse Distillation Algorithm.

Parameters:

- **backbone** (*str*) – Backbone of CNN network Defaults to `wide_resnet50_2`.
- **layers** (*list[str]*) – Layers to extract features from the backbone CNN Defaults to `["layer1", "layer2", "layer3"]`.
- **anomaly_map_mode** (*AnomalyMapGenerationMode, optional*) – Mode to generate anomaly map. Defaults to `AnomalyMapGenerationMode.ADD`.
- **pre_trained** (*bool, optional*) – Boolean to check whether to use a pre_trained backbone. Defaults to `True`.

configure_optimizers()

Configure optimizers for decoder and bottleneck.

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]*

Return Reverse Distillation trainer arguments.

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

Perform a training step of Reverse Distillation Model.

Features are extracted from three layers of the Encoder model. These are passed to the bottleneck layer that are passed to the decoder network. The loss is then calculated based on the cosine similarity between the encoder and decoder features.

Parameters:

- **(batch** (*batch*) – dict[str, str | torch.Tensor]): Input batch
- **args** – Additional arguments.
- **kwargs** – Additional keyword arguments.

Return type:

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

Returns:

Feature Map

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

Perform a validation step of Reverse Distillation Model.

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

Parameters:

- **batch** (*dict[str, str | torch.Tensor]*) – Input batch
- **args** – Additional arguments.
- **kwargs** – Additional 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 Reverse Distillation.

class `anomalib.models.image.reverse_distillation.torch_model.ReverseDistillationModel`(*backbone*, *input_size*, *layers*, *anomaly_map_mode*, *pre_trained=True*)

Bases: `Module`

Reverse Distillation Model.

To reproduce results in the paper, use torchvision model for the encoder:

self.encoder = torchvision.models.wide_resnet50_2(pretrained=True)

Parameters:

- **backbone** (*str*) – Name of the backbone used for encoder and decoder.
- **input_size** (*tuple[int, int]*) – Size of input image.
- **layers** (*list[str]*) – Name of layers from which the features are extracted.
- **anomaly_map_mode** (*str*) – Mode used to generate anomaly map. Options are between `multiply` and `add`.
- **pre_trained** (*bool, optional*) – Boolean to check whether to use a pre_trained backbone. Defaults to `True`.

forward(*images*)

Forward-pass images to the network.

During the training mode the model extracts features from encoder and decoder networks. During evaluation mode, it returns the predicted anomaly map.

Parameters:

images (*torch.Tensor*) – Batch of images

Returns:

Encoder and decoder features

in training mode, else anomaly maps.

Return type:

torch.Tensor | list[torch.Tensor] | tuple[list[torch.Tensor]]

Loss function for Reverse Distillation.

class **anomalib.models.image.reverse_distillation.loss.ReverseDistillationLoss**(**args, **kwargs*)

Bases: `Module`

Loss function for Reverse Distillation.

forward(*encoder_features, decoder_features*)

Compute cosine similarity loss based on features from encoder and decoder.

Based on the official code: hq-deng/RD4AD Calculates loss from flattened arrays of features, see hq-deng/RD4AD#22

Parameters:

- **encoder_features** (*list[torch.Tensor]*) – List of features extracted from encoder
- **decoder_features** (*list[torch.Tensor]*) – List of features extracted from decoder

Returns:

Cosine similarity loss

Return type:

Tensor

Compute Anomaly map.

class **anomalib.models.image.reverse_distillation.anomaly_map.AnomalyMapGenerationMode**(*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: `str`, `Enum`

Type of mode when generating anomaly imape.

class **anomalib.models.image.reverse_distillation.anomaly_map.AnomalyMapGenerator**(*image_size*, *sigma=4*, *mode=AnomalyMapGenerationMode.MULTIPLY*)

Bases: `Module`

Generate Anomaly Heatmap.

Parameters:

- **image_size** (*ListConfig, tuple*) – Size of original image used for upscaling the anomaly map.
- **sigma** (*int*) – Standard deviation of the gaussian kernel used to smooth anomaly map. Defaults to `4`.
- **mode** (*AnomalyMapGenerationMode, optional*) – Operation used to generate anomaly map. Options are `AnomalyMapGenerationMode.ADD` and `AnomalyMapGenerationMode.MULTIPLY`. Defaults to `AnomalyMapGenerationMode.MULTIPLY`.

Raises:

ValueError – In case modes other than multiply and add are passed.

forward(*student_features*, *teacher_features*)

Compute anomaly map given encoder and decoder features.

Parameters:

- **student_features** (*list[torch.Tensor]*) – List of encoder features
- **teacher_features** (*list[torch.Tensor]*) – List of decoder features

Returns:

Anomaly maps of length batch.

Return type:

Tensor

Previous PatchCore

Next R-KDE