PatchCore

Contents

Towards Total Recall in Industrial Anomaly Detection.

Paper https://arxiv.org/abs/2106.08265.

class **anomalib.models.image.patchcore.lightning_model.Patchcore(*backbone='wide_resnet50_2', layers=('layer2', 'layer3'), pre_trained=True, coreset_sampling_ratio=0.1, num_neighbors=9*)**

Bases: `MemoryBankMixin`, `AnomalyModule`

PatchcoreLightning Module to train PatchCore algorithm.

Parameters:

- **backbone** (*str*) – Backbone CNN network Defaults to `wide_resnet50_2`.
- **layers** (*list[str]*) – Layers to extract features from the backbone CNN Defaults to `["layer2", "layer3"]`.
- **pre_trained** (*bool, optional*) – Boolean to check whether to use a pre_trained backbone. Defaults to `True`.
- **coreset_sampling_ratio** (*float, optional*) – Coreset sampling ratio to subsample embedding. Defaults to `0.1`.
- **num_neighbors** (*int, optional*) – Number of nearest neighbors. Defaults to `9`.

configure_optimizers()

Configure optimizers.

Returns:

Do not set optimizers by returning None.

Return type:

None

configure_transforms(image_size=None)

Default transform for Padim.

Return type:

`Transform`

fit()

Apply subsampling to the embedding collected from the training set.

Return type:

`None`

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 Patchcore trainer arguments.

training_step(batch, *args, **kwargs)

Generate feature embedding of the batch.

Parameters:

- **batch** (*dict[str, str | torch.Tensor]*) – Batch containing image filename, image, label and mask
- **args** – Additional arguments.
- **kwargs** – Additional keyword arguments.

Returns:

Embedding Vector

Return type:

dict[str, np.ndarray]

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

Get batch of anomaly maps from input image batch.

Parameters:

- **batch** (*dict[str, str | torch.Tensor]*) – Batch containing image filename, image, label and mask
- **args** – Additional arguments.
- **kwargs** – Additional keyword arguments.

Returns:

Image filenames, test images, GT and predicted label/masks

Return type:

dict[str, Any]

PyTorch model for the PatchCore model implementation.

class anomalib.models.image.patchcore.torch_model.PatchcoreModel(*layers*, *backbone='wide_resnet50_2'*, *pre_trained=True*, *num_neighbors=9*)

Bases: `DynamicBufferMixin`, `Module`

Patchcore Module.

Parameters:

- **layers** (*list[str]*) – Layers used for feature extraction
- **backbone** (*str, optional*) – Pre-trained model backbone. Defaults to `resnet18`.
- **pre_trained** (*bool, optional*) – Boolean to check whether to use a pre_trained backbone. Defaults to `True`.
- **num_neighbors** (*int, optional*) – Number of nearest neighbors. Defaults to `9`.

compute_anomaly_score(*patch_scores*, *locations*, *embedding*)

Compute Image-Level Anomaly Score.

Parameters:

- **patch_scores** (*torch.Tensor*) – Patch-level anomaly scores
- **locations** (`Tensor`) – Memory bank locations of the nearest neighbor for each patch location
- **embedding** (`Tensor`) – The feature embeddings that generated the patch scores

Returns:

Image-level anomaly scores

Return type:

Tensor

static euclidean_dist(*x*, *y*)

Calculate pair-wise distance between row vectors in x and those in y.

Replaces torch cdist with p=2, as cdist is not properly exported to onnx and openvino format. Resulting matrix is indexed by x vectors in rows and y vectors in columns.

Parameters:

- **x** (`Tensor`) – input tensor 1
- **y** (`Tensor`) – input tensor 2

Return type:

`Tensor`

Returns:

Matrix of distances between row vectors in x and y.

forward(*input_tensor*)

Return Embedding during training, or a tuple of anomaly map and anomaly score during testing.

Steps performed: 1. Get features from a CNN. 2. Generate embedding based on the features. 3. Compute anomaly map in test mode.

Parameters:

input_tensor (*torch.Tensor*) – Input tensor

Returns:

Embedding for training, anomaly map and anomaly score for testing.

Return type:

Tensor | dict[str, torch.Tensor]

generate_embedding(*features*)

Generate embedding from hierarchical feature map.

Parameters:

- **features** (`dict`[`str`, `Tensor`]) – Hierarchical feature map from a CNN (ResNet18 or WideResnet)
- **features** – dict[str:Tensor]:

Return type:

`Tensor`

Returns:

Embedding vector

nearest_neighbors(*embedding*, *n_neighbors*)

Nearest Neighbours using brute force method and euclidean norm.

Parameters:

- **embedding** (*torch.Tensor*) – Features to compare the distance with the memory bank.
- **n_neighbors** (*int*) – Number of neighbors to look at

Returns:

Patch scores. Tensor: Locations of the nearest neighbor(s).

Return type:

Tensor

static reshape_embedding(*embedding*)

Reshape Embedding.

Reshapes Embedding to the following format:

- [Batch, Embedding, Patch, Patch] to [Batch*Patch*Patch, Embedding]

Parameters:

embedding (*torch.Tensor*) – Embedding tensor extracted from CNN features.

Returns:

Reshaped embedding tensor.

Return type:

Tensor

subsample_embedding(*embedding*, *sampling_ratio*)

Subsample embedding based on coreset sampling and store to memory.

Parameters:

- **embedding** (*np.ndarray*) – Embedding tensor from the CNN
- **sampling_ratio** (*float*) – Coreset sampling ratio

Return type:

None

Previous
Padim

Next
Reverse Distillation