DFM

Contents

DFM: Deep Feature Modeling.

https://arxiv.org/abs/1909.11786

class `anomalib.models.image.dfm.lightning_model.Dfm(backbone='resnet50', layer='layer3', pre_trained=True, pooling_kernel_size=4, pca_level=0.97, score_type='fre')`

Bases: `MemoryBankMixin`, `AnomalyModule`

DFM: Deep Featured Kernel Density Estimation.

Parameters:

- **backbone** (*str*) – Backbone CNN network Defaults to `"resnet50"`.
- **layer** (*str*) – Layer to extract features from the backbone CNN Defaults to `"layer3"`.
- **pre_trained** (*bool, optional*) – Boolean to check whether to use a pre_trained backbone. Defaults to `True`.
- **pooling_kernel_size** (*int, optional*) – Kernel size to pool features extracted from the CNN. Defaults to `4`.
- **pca_level** (*float, optional*) – Ratio from which number of components for PCA are calculated. Defaults to `0.97`.
- **score_type** (*str, optional*) – Scoring type. Options are *fre* and *nll*. Defaults to `fre`.

static configure_optimizers()

DFM doesn’t require optimization, therefore returns no optimizers.

Return type:

`None`

fit()

Fit a PCA transformation and a Gaussian model to dataset.

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 DFM-specific trainer arguments.

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

Perform the training step of DFM.

For each batch, features are extracted from the CNN.

Parameters:

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

Return type:

`None`

Returns:

Deep CNN features.

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

Perform the validation step of DFM.

Similar to the training step, features are extracted from the CNN for each batch.

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 FRE anomaly scores and anomaly maps.

PyTorch model for DFM model implementation.

`class anomalib.models.image.dfm.torch_model.DFMModel(backbone, layer, pre_trained=True, pooling_kernel_size=4, n_comps=0.97, score_type='fre')`

Bases: `Module`

Model for the DFM algorithm.

Parameters:

- **backbone** (*str*) – Pre-trained model backbone.
- **layer** (*str*) – Layer from which to extract features.
- **pre_trained** (*bool, optional*) – Boolean to check whether to use a pre_trained backbone. Defaults to `True`.
- **pooling_kernel_size** (*int, optional*) – Kernel size to pool features extracted from the CNN. Defaults to `4`.
- **n_comps** (*float, optional*) – Ratio from which number of components for PCA are calculated. Defaults to `0.97`.
- **score_type** (*str, optional*) – Scoring type. Options are *fre* and *nll*. Anomaly Defaults to `fre`. Segmentation is supported with *fre* only. If using *nll*, set *task* in config.yaml to classification Defaults to `classification`.

`fit(dataset)`

Fit a pca transformation and a Gaussian model to dataset.

Parameters:

dataset (*torch.Tensor*) – Input dataset to fit the model.

Return type:

`None`

`forward(batch)`

Compute score from input images.

Parameters:

batch (*torch.Tensor*) – Input images

Returns:

Scores

Return type:

Tensor

`get_features(batch)`

Extract features from the pretrained network.

Parameters:

batch (*torch.Tensor*) – Image batch.

Returns:

torch.Tensor containing extracted features.

Return type:

Tensor

score(*features*, *feature_shapes*)

Compute scores.

Scores are either PCA-based feature reconstruction error (FRE) scores or the Gaussian density-based NLL scores

Parameters:

- **features** (*torch.Tensor*) – semantic features on which PCA and density modeling is performed.
- **feature_shapes** (*tuple*) – shape of *features* tensor. Used to generate anomaly map of correct shape.

Returns:

numpy array of scores

Return type:

score (torch.Tensor)

class anomalib.models.image.dfm.torch_model.SingleClassGaussian

Bases: `DynamicBufferMixin`

Model Gaussian distribution over a set of points.

fit(*dataset*)

Fit a Gaussian model to dataset X.

Covariance matrix is not calculated directly using: `C = X.X^T` Instead, it is represented in terms of the Singular Value Decomposition of X: `X = U.S.V^T` Hence, `C = U.S^2.U^T` This simplifies the calculation of the log-likelihood without requiring full matrix inversion.

Parameters:

dataset (*torch.Tensor*) – Input dataset to fit the model.

Return type:

`None`

`forward(dataset)`

Provide the same functionality as *fit*.

Transforms the input dataset based on singular values calculated earlier.

Parameters:

dataset (*torch.Tensor*) – Input dataset

Return type:

`None`

`score_samples(features)`

Compute the NLL (negative log likelihood) scores.

Parameters:

features (*torch.Tensor*) – semantic features on which density modeling is performed.

Returns:

Torch tensor of scores

Return type:

nll (torch.Tensor)

‹ Previous
DFKDE

Next ›
DRAEM