WinCLIP

Contents

WinCLIP: Zero-/Few-Shot Anomaly Classification and Segmentation.

Paper https://arxiv.org/abs/2303.14814

class **anomalib.models.image.winclip.lightning_model.WinClip(*class_name=None*, *k_shot=0*, *scales=(2, 3)*, *few_shot_source=None*)**

Bases: `AnomalyModule`

WinCLIP Lightning model.

Parameters:

- **class_name** (*str, optional*) – The name of the object class used in the prompt ensemble. Defaults to `None`.
- **k_shot** (*int*) – The number of reference images for few-shot inference. Defaults to `0`.
- **scales** (*tuple[int], optional*) – The scales of the sliding windows used for multiscale anomaly detection. Defaults to `(2, 3)`.
- **few_shot_source** (*str | Path, optional*) – Path to a folder of reference images used for few-shot inference. Defaults to `None`.

collect_reference_images(*dataloader*)

Collect reference images for few-shot inference.

The reference images are collected by iterating the training dataset until the

required number of images are collected.

Returns:

A tensor containing the reference images.

Return type:

ref_images (Tensor)

static configure_optimizers()

WinCLIP doesn't require optimization, therefore returns no optimizers.

Return type:

`None`

configure_transforms(*image_size=None*)

Configure the default transforms used by the model.

Return type:

`Transform`

property learning_type: *LearningType*

The learning type of the model.

WinCLIP is a zero-/few-shot model, depending on the user configuration. Therefore, the learning type is set to `LearningType.FEW_SHOT` when `k_shot` is greater than zero and `LearningType.ZERO_SHOT` otherwise.

load_state_dict(*state_dict*, *strict=True*)

Load the state dict of the model.

Before loading the state dict, we restore the parameters of the frozen backbone to ensure that the model is loaded correctly. We also restore the auxiliary objects like threshold classes and normalization metrics.

Return type:

`Any`

state_dict()

Return the state dict of the model.

Before returning the state dict, we remove the parameters of the frozen backbone to reduce the size of the checkpoint.

Return type:

`OrderedDict`[`str`, `Any`]

property trainer_arguments: dict[str, int | float]

Set model-specific trainer arguments.

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

Validation Step of WinCLIP.

Return type:

`dict`

PyTorch model for the WinCLIP implementation.

class anomalib.models.image.winclip.torch_model.WinClipModel(*class_name=None*, *reference_images=None*, *scales=(2, 3)*, *apply_transform=False*)

Bases: `DynamicBufferMixin`, `BufferListMixin`, `Module`

PyTorch module that implements the WinClip model for image anomaly detection.

Parameters:

- **class_name** (*str, optional*) – The name of the object class used in the prompt ensemble. Defaults to `None`.
- **reference_images** (*torch.Tensor, optional*) – Tensor of shape `(K, C, H, W)` containing the reference images. Defaults to `None`.
- **scales** (*tuple[int], optional*) – The scales of the sliding windows used for multi-scale anomaly detection. Defaults to `(2, 3)`.
- **apply_transform** (*bool, optional*) – Whether to apply the default CLIP transform to the input images. Defaults to `False`.

clip

The CLIP model used for image and text encoding.

Type:

CLIP

grid_size

The size of the feature map grid.

Type:

tuple[int]

k_shot

The number of reference images used for few-shot anomaly detection.

Type:

int

scales

The scales of the sliding windows used for multi-scale anomaly detection.

Type:

tuple[int]

masks

The masks representing the sliding window locations.

Type:

list[torch.Tensor] | None

_text_embeddings

The text embeddings for the compositional prompt ensemble.

Type:

torch.Tensor | None

_visual_embeddings

The multi-scale embeddings for the reference images.

Type:

list[torch.Tensor] | None

_patch_embeddings

The patch embeddings for the reference images.

Type:

torch.Tensor | None

encode_image(*batch*)

Encode the batch of images to obtain image embeddings, window embeddings, and patch embeddings.

The image embeddings and patch embeddings are obtained by passing the batch of images through the model. The window embeddings are obtained by masking the feature map and passing it through the transformer. A forward hook is used to retrieve the intermediate feature map and share computation between the image

and window embeddings.

Parameters:

batch (*torch.Tensor*) – Batch of input images of shape `(N, C, H, W)`.

Returns:

A tuple containing the image embeddings, window embeddings, and patch embeddings respectively.

Return type:

Tuple[torch.Tensor, List[torch.Tensor], torch.Tensor]

Examples

```
>>> model = WinClipModel()
>>> model.prepare_masks()
>>> batch = torch.rand((1, 3, 240, 240))
>>> image_embeddings, window_embeddings, patch_embeddings = model.encode_im
>>> image_embeddings.shape
torch.Size([1, 640])
>>> [embedding.shape for embedding in window_embeddings]
[torch.Size([1, 196, 640]), torch.Size([1, 169, 640])]
>>> patch_embeddings.shape
torch.Size([1, 225, 896])
```

forward(*batch*)

Forward-pass through the model to obtain image and pixel scores.

Parameters:

batch (*torch.Tensor*) – Batch of input images of shape `(batch_size, C, H, W)`.

Returns:

Tuple containing the image scores and pixel scores.

Return type:

Tuple[torch.Tensor, torch.Tensor]

property patch_embeddings: *Tensor*

The patch embeddings used by the model.

setup(*class_name=None*, *reference_images=None*)

Setup WinCLIP.

WinCLIP's setup stage consists of collecting the text and visual embeddings used during inference. The following steps are performed, depending on the arguments passed to the model: - Collect text embeddings for zero-shot inference. - Collect reference images for few-shot inference. The k_shot attribute is updated based on the number of reference images.

The setup method is called internally by the constructor. However, it can also be called manually to update the text and visual embeddings after the model has been initialized.

Parameters:

- **class_name** (*str*) – The name of the object class used in the prompt ensemble.
- **reference_images** (*torch.Tensor*) – Tensor of shape `(batch_size, C, H, W)` containing the reference images.

Return type:

`None`

Examples

```
>>> model = WinClipModel()
>>> model.setup("transistor")
>>> model.text_embeddings.shape
torch.Size([2, 640])
```

```
>>> ref_images = torch.rand(2, 3, 240, 240)
>>> model = WinClipModel()
>>> model.setup("transistor", ref_images)
>>> model.k_shot
2
>>> model.visual_embeddings[0].shape
torch.Size([2, 196, 640])
```

```
>>> model = WinClipModel("transistor")
>>> model.k_shot
0
>>> model.setup(reference_images=ref_images)
>>> model.k_shot
2
```

```
>>> model = WinClipModel(class_name="transistor", reference_images=ref_imag
>>> model.text_embeddings.shape
torch.Size([2, 640])
>>> model.visual_embeddings[0].shape
torch.Size([2, 196, 640])
```

property* text_embeddings: *Tensor

The text embeddings used by the model.

property* transform: *Compose

The transform used by the model.

To obtain the transforms, we retrieve the transforms from the clip backbone. Since the original transforms are intended for PIL images, we prepend a ToPILImage transform to the list of transforms.

property* visual_embeddings: *list[Tensor]

The visual embeddings used by the model.

Previous
U-Flow

Next
Video Models