
vformer Documentation

Release 0.1.3

Neelay Shah

Sep 09, 2022

CONTENTS

| | | |
|----------|--|-----------|
| 1 | VFormer | 1 |
| 2 | Installation | 3 |
| 2.1 | From source (recommended) | 3 |
| 2.2 | Stable release | 3 |
| 3 | Attention | 5 |
| 3.1 | Vanilla Self Attention $O(n^2)$ | 5 |
| 3.2 | Cross Attention | 5 |
| 3.3 | Spatial Self Attention | 7 |
| 3.4 | Window Attention | 7 |
| 3.5 | Memory Efficient self Attention $O(n)$ | 8 |
| 3.6 | Gated Positional Self Attention | 9 |
| 3.7 | Self Attention with Convolutional Projection | 10 |
| 4 | Common | 11 |
| 4.1 | Base Classification Model | 11 |
| 4.2 | Blocks | 11 |
| 5 | Decoder | 13 |
| 5.1 | MLP | 13 |
| 5.2 | Perceiver IO Decoder | 13 |
| 5.3 | Task Heads | 14 |
| 6 | Encoders | 15 |
| 6.1 | Embedding Layers | 15 |
| 6.2 | Cross Encoder | 19 |
| 6.3 | FeedForward MLP | 20 |
| 6.4 | Pyramid Encoder | 20 |
| 6.5 | Perceiver IO Encoder | 22 |
| 6.6 | Swin Encoder | 22 |
| 6.7 | Vanilla Encoder | 24 |
| 6.8 | ConViT | 24 |
| 6.9 | ConvVT | 25 |
| 7 | Functional | 27 |
| 7.1 | Patch Merging | 27 |
| 7.2 | Normalization Layers | 27 |
| 8 | Models | 29 |
| 8.1 | Classification | 29 |

| | | |
|-----------|--------------------------------------|-----------|
| 8.2 | Dense Prediction | 43 |
| 9 | Utilities | 51 |
| 9.1 | Generic Utilities | 51 |
| 9.2 | Window Attention Utilities | 51 |
| 10 | Visualisation | 53 |
| 10.1 | Rollout | 53 |
| 10.2 | Gradient Rollout | 53 |
| 11 | Indices and tables | 55 |
| | Python Module Index | 57 |
| | Index | 59 |

VFORMER

A modular PyTorch library for vision transformer models

- Free software: MIT license
- Documentation: <https://vformer.readthedocs.io>.

INSTALLATION

2.1 From source (recommended)

VFormer can be installed from the [GitHub repo](#).

Clone the public repository:

```
$ git clone https://github.com/SforAiDl/vformer.git
```

and then run the following command to install VFormer:

```
$ python setup.py install
```

2.2 Stable release

To install VFormer, run this command in your terminal:

```
$ pip install vformer
```

Note that VFormer is an active project and routinely publishes new releases. In order to upgrade VFormer to the latest version, use pip as follows:

```
$ pip install -U vformer
```


ATTENTION

3.1 Vanilla Self Attention $O(n^2)$

```
class vformer.attention.vanilla.VanillaSelfAttention(dim, num_heads=8, head_dim=64,  
                                                    p_dropout=0.0)
```

Bases: Module

Vanilla $O(n^2)$ Self attention introduced in [An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale](#)

Parameters

- **dim** (*int*) – Dimension of the embedding
- **num_heads** (*int*) – Number of the attention heads
- **head_dim** (*int*) – Dimension of each head
- **p_dropout** (*float*) – Dropout Probability

forward(*x*)

Parameters

x (*torch.Tensor*) – Input tensor

Returns

Returns output tensor by applying self-attention on input tensor

Return type

torch.Tensor

training: *bool*

3.2 Cross Attention

```
class vformer.attention.cross.CrossAttention(query_dim, context_dim, num_heads=8, head_dim=64)
```

Bases: Module

This variant of Cross Attention is iteratively used in Perciever IO.

In Cross-Attention, cls token from one branch and patch token from another branch are fused together.

Parameters

- **query_dim** (*int*) – Dimension of query array

- **context_dim** (*int*) – Dimension of context array
- **num_heads** (*int*) – Number of cross-attention heads
- **head_dim** (*int*) – Dimension of each head

forward(*x, context, mask=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

training: `bool`

class `vformer.attention.cross.CrossAttentionWithClsToken`(*cls_dim, patch_dim, num_heads=8, head_dim=64*)

Bases: `Module`

Cross-Attention with Cls Token introduced in Paper: CrossViT: [Cross-Attention Multi-Scale Vision Transformer for Image Classification](#)

In Cross-Attention, cls token from one branch and patch token from another branch are fused together.

Parameters

- **cls_dim** (*int*) – Dimension of cls token embedding
- **patch_dim** (*int*) – Dimension of patch token embeddings cls token to be fused with
- **num_heads** (*int*) – Number of cross-attention heads
- **head_dim** (*int*) – Dimension of each head

forward(*cls, patches*)

Parameters

- **x** (*torch.Tensor*) – Input tensor
- **cls** (*torch.Tensor*) – CLS token from one branch
- **patch** (*torch.Tensor*) – patch tokens from another branch

Returns

Returns output tensor by applying cross attention on input tensor

Return type

`torch.Tensor`

training: `bool`

3.3 Spatial Self Attention

```
class vformer.attention.spatial.SpatialAttention(dim, num_heads, sr_ratio=1, qkv_bias=False,
                                                qk_scale=None, attn_drop=0.0, proj_drop=0.0,
                                                linear=False, activation=<class
                                                'torch.nn.modules.activation.GELU'>)
```

Bases: Module

Spatial Reduction Attention introduced in : [Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions](#) This class also supports the linear complexity spatial attention in the improved paper

Parameters

- **dim** (*int*) – Dimension of the input tensor
- **num_heads** (*int*) – Number of attention heads
- **sr_ratio** (*int*) – Spatial Reduction ratio
- **qkv_bias** (*bool*) – If True, add a learnable bias to query, key, value, default is True
- **qk_scale** (*float, optional*) – Override default qk scale of head_dim ** -0.5 if set
- **attn_drop** (*float, optional*) – Dropout rate
- **proj_drop** (*float, optional*) – Dropout rate
- **linear** (*bool*) – Whether to use linear Spatial attention, default is False.
- **activation** (*nn.Module*) – Activation function, default is nn.GELU.

forward(*x, H, W*)

Parameters

- **x** (*torch.Tensor*) – Input tensor
- **H** (*int*) – Height of image patches
- **W** (*int*) – Width of image patches

Returns

Returns output tensor by applying spatial attention on input tensor

Return type

torch.Tensor

training: bool

3.4 Window Attention

```
class vformer.attention.window.WindowAttention(dim, window_size, num_heads, qkv_bias=True,
                                                qk_scale=None, attn_dropout=0.0, proj_dropout=0.0)
```

Bases: Module

Implementation of Window Attention introduced in: [Swin Transformer: Hierarchical Vision Transformer using Shifted Windows](#)

Parameters

- **dim** (*int*) – Number of input channels.
- **window_size** (*int or tuple[int]*) – The height and width of the window.
- **num_heads** (*int*) – Number of attention heads.
- **qkv_bias** (*bool*) – If True, add a learnable bias to query, key, value, default is True
- **qk_scale** (*float, optional*) – Override default qk scale of head_dim ** -0.5 if set
- **attn_dropout** (*float, optional*) – Dropout rate, default is 0.0.
- **proj_dropout** (*float, optional*) – Dropout rate, default is 0.0.

forward(*x, mask=None*)

Parameters

- **x** (*torch.Tensor*) – input Tensor
- **mask** (*torch.Tensor*) – Attention mask used for shifted window attention, if None, window attention will be used, else attention mask will be taken into consideration. for better understanding you may refer [this github issue](#).

Returns

Returns output tensor by applying Window-Attention or Shifted-Window-Attention on input tensor

Return type

torch.Tensor

training: bool

3.5 Memory Efficient self Attention O(n)

```
class vformer.attention.memory_efficient.MemoryEfficientAttention(dim, num_heads=8,
                                                                head_dim=64,
                                                                p_dropout=0.0,
                                                                query_chunk_size=1024,
                                                                key_chunk_size=4096)
```

Bases: Module

Memory Efficient attention introduced in paper [Self-attention Does Not Need O\(n²\) Memory](#)

Implementation based on [this repository](#)

Parameters

- **dim** (*int*) – Dimension of the embedding
- **num_heads** (*int*) – Number of the attention heads
- **head_dim** (*int*) – Dimension of each head
- **p_dropout** (*float*) – Dropout Probability

static dynamic_slice(*x, starts, sizes*)

forward(*x*)

Parameters

x (*torch.Tensor*) – Input tensor

Returns

Returns output tensor by applying self-attention on input tensor

Return type

torch.Tensor

static map_pt(*f*, *xs*)**query_chunk_attention**(*query*, *key*, *value*)**static scan**(*f*, *init*, *xs*, *length=None*)**static summarize_chunk**(*query*, *key*, *value*)**training:** bool

3.6 Gated Positional Self Attention

```
class vformer.attention.gated_positional.GatedPositionalSelfAttention(dim, num_heads=8,
                                                                    head_dim=64,
                                                                    p_dropout=0)
```

Bases: *VanillaSelfAttention*Implementation of the Gated Positional Self-Attention from the paper: [ConViT: Improving Vision Transformers with Soft Convolutional Inductive Biases](#)**Parameters**

- **dim** (*int*) – Dimension of the embedding
- **num_heads** (*int*) – Number of the attention heads, default is 8
- **head_dim** (*int*) – Dimension of each head, default is 64
- **p_dropout** (*float*) – Dropout probability, default is 0.0

forward(*x*)**Parameters****x** (*torch.Tensor*) – Input tensor**Returns**

Returns output tensor by applying self-attention on input tensor

Return type

torch.Tensor

rel_embedding(*n*)**training:** bool

3.7 Self Attention with Convolutional Projection

```
class vformer.attention.convvt.ConvVTAttention(dim_in, dim_out, num_heads, img_size,  
                                             attn_dropout=0.0, proj_dropout=0.0, method='dw_bn',  
                                             kernel_size=3, stride_kv=1, stride_q=1,  
                                             padding_kv=1, padding_q=1, with_cls_token=False)
```

Bases: Module

Attention with Convolutional Projection introduced in Paper: [Introducing Convolutions to Vision Transformers](#)

Position-wise linear projection for Multi-Head Self-Attention (MHSA) replaced by Depth-wise separable convolutions

Parameters

- **dim_in** (*int*) – Dimension of input tensor
- **dim_out** (*int*) – Dimension of output tensor
- **num_heads** (*int*) – Number of heads in attention
- **img_size** (*int*) – Size of image
- **attn_dropout** (*float*) – Probability of dropout in attention
- **proj_dropout** (*float*) – Probability of dropout in convolution projection
- **method** (*str*) – Method of projection, 'dw_bn' for depth-wise convolution and batch norm, 'avg' for average pooling. default is 'dw_bn'
- **kernel_size** (*int*) – Size of kernel
- **stride_kv** (*int*) – Size of stride for key value
- **stride_q** (*int*) – Size of stride for query
- **padding_kv** (*int*) – Padding for key value
- **padding_q** (*int*) – Padding for query
- **with_cls_token** (*bool*) – Whether to include classification token, default is `False`.

forward(*x*)

Parameters

x (*torch.Tensor*) – Input tensor

Returns

Returns output tensor by applying self-attention on input tensor

Return type

torch.Tensor

forward_conv(*x*)

training: bool

4.1 Base Classification Model

```
class vformer.common.base_model.BaseClassificationModel(img_size, patch_size, in_channels=3,  
                                                       pool='cls')
```

Parameters

- **img_size** (*int*) – Size of the image
- **patch_size** (*int or tuple(int)*) – Size of the patch
- **in_channels** (*int*) – Number of channels in input image
- **pool** (*str*) – Feature pooling type, must be one of {mean, cls}

4.2 Blocks

```
class vformer.common.blocks.DWConv(dim, kernel_size_dwconv=3, stride_dwconv=1, padding_dwconv=1,  
                                   bias_dwconv=True)
```

Depth Wise Convolution

Parameters

- **dim** (*int*) – Dimension of the input tensor
- **kernel_size_dwconv** (*int, optional*) – Size of the convolution kernel, default is 3
- **stride_dwconv** (*int*) – Stride of the convolution, default is 1
- **padding_dwconv** (*int or tuple or str*) – Padding added to all sides of the input, default is 1
- **bias_dwconv** (*bool*) – Whether to add learnable bias to the output, default is True.

forward(*x, H, W*)

Parameters

- **x** (*torch.Tensor*) – Input tensor
- **H** (*int*) – Height of image patch
- **W** (*int*) – Width of image patch

Returns

Returns output tensor after performing depth-wise convolution operation

Return type
torch.Tensor

5.1 MLP

```
class vformer.decoder.mlp.MLPDecoder(config=(1024,), n_classes=10)
```

Bases: Module

Parameters

- **config** (*int or tuple or list*) – Configuration of the hidden layer(s)
- **n_classes** (*int*) – Number of classes for classification

forward(*x*)

Parameters

x (*torch.Tensor*) – Input tensor

Returns

Returns output tensor of size *n_classes*, Note that *torch.nn.Softmax* is not applied to the output tensor.

Return type

torch.Tensor

5.2 Perceiver IO Decoder

```
class vformer.decoder.perceiver_io.PerceiverIODecoder(dim=32, latent_dim=512, queries_dim=32,  
                                                    num_cross_heads=1, cross_head_dim=64,  
                                                    logits_dim=None, decoder_ff=False)
```

Implementation of the Perceiver IO Decoder

Parameters

- **dim** (*int*) – Size of sequence to be encoded
- **latent_dim** (*int*) – Dimension of latent array
- **queries_dim** (*int*) – Dimension of queries array
- **num_latents** (*int*) – Number of latent arrays
- **num_cross_heads** (*int*) – Number of heads for cross attention
- **cross_head_dim** (*int*) – Dimension of cross attention head
- **logits_dim** (*int, optional*) – Dimension of output logits

- **decoder_ff** (*bool*) – Whether to include a feed forward layer for the decoder attention block

forward(*x*, *mask=None*, *queries=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

5.3 Task Heads

5.3.1 Segmentation

Double Convolution

class `vformer.decoder.task_heads.segmentation.head.DoubleConv`(*in_channels*, *out_channels*)

Bases: `Module`

Module consisting of two convolution layers and activations

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

class `vformer.decoder.task_heads.segmentation.head.SegmentationHead`(*out_channels=1*,
embed_dims=[64, 128,
256, 512])

Bases: `Module`

U-net like up-sampling block

forward(*skip_connections*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

ENCODERS

6.1 Embedding Layers

6.1.1 Linear Embedding

```
class vformer.encoder.embedding.linear.LinearEmbedding(embedding_dim, patch_height, patch_width,  
patch_dim)
```

Projects image patches into embedding space using Linear layer.

Parameters

- **embedding_dim** (*int*) – Dimension of the resultant embedding
- **patch_height** (*int*) – Height of the patch
- **patch_width** (*int*) – Width of the patch
- **patch_dim** (*int*) – Dimension of the patch

forward(*x*)**Parameters**

x (*torch.Tensor*) – Input tensor

Returns

Returns patch embeddings of size *embedding_dim*

Return type

torch.Tensor

6.1.2 Patch Overlap Embedding

```
class vformer.encoder.embedding.overlappatch.OverlapPatchEmbed(img_size, patch_size, stride=4,  
in_channels=3,  
embedding_dim=768,  
norm_layer=<class  
'torch.nn.modules.normalization.LayerNorm'>)
```

Parameters

- **img_size** (*int*) – Image Size
- **patch_size** (*int or tuple(int)*) – Patch Size
- **stride** (*int*) – Stride of the convolution, default is 4

- **in_channels** (*int*) – Number of input channels in the image, default is 3
- **embedding_dim** (*int*) – Number of linear projection output channels, default is 768
- **norm_layer** (*nn.Module, optional*) – Normalization layer, default is `nn.LayerNorm`

forward(*x*)

Parameters

x (*torch.Tensor*) – Input tensor

Returns

- **x** (*torch.Tensor*) – Input tensor
- **H** (*int*) – Height of Patch
- **W** (*int*) – Width of Patch

6.1.3 Patch Embedding

```
class vformer.encoder.embedding.patch.PatchEmbedding(img_size, patch_size, in_channels,
                                                    embedding_dim, norm_layer=<class
                                                    'torch.nn.modules.normalization.LayerNorm'>)
```

Parameters

- **img_size** (*int*) – Image Size
- **patch_size** (*int*) – Patch Size
- **in_channels** (*int*) – Number of input channels in the image
- **embedding_dim** (*int*) – Number of linear projection output channels
- **norm_layer** (*nn.Module,)* – Normalization layer, Default is `nn.LayerNorm`

forward(*x*)

Parameters

x (*torch.Tensor*) – Input tensor

Returns

Returns output tensor by applying convolution operation with same *kernel_size* and *stride* on input tensor.

Return type

`torch.Tensor`

6.1.4 Positional Embedding

```
class vformer.encoder.embedding.pos_embedding.PVTPosEmbedding(pos_shape, pos_dim,
                                                            p_dropout=0.0, std=0.02)
```

Positional Embedding class used in Pyramid vision transformer.

Parameters

- **pos_shape** (*int or tuple(int)*) – The shape of the absolute position embedding.
- **pos_dim** (*int*) – The dimension of the absolute position embedding.
- **p_dropout** (*float, optional*) – Probability of an element to be zeroed, default is 0.2

- **std** (*float*) – Standard deviation for truncated normal distribution

forward(*x*, *H*, *W*, *mode*='bilinear')

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

resize_pos_embed(*pos_embed*, *shape*, *mode*='bilinear', ***kwargs*)

Parameters

- **pos_embed** (*torch.Tensor*) – Position embedding weights
- **shape** (*tuple*) – Required shape
- **mode** (*str* (nearest | linear | bilinear | bicubic | trilinear)) – Algorithm used for up/down sampling, default is bilinear.

class `vformer.encoder.embedding.pos_embedding.PosEmbedding`(*shape*, *dim*, *drop*=None, *sinusoidal*=False, *std*=0.02)

Generalised Positional Embedding class

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

6.1.5 Convvt

class `vformer.encoder.embedding.convvt.ConvEmbedding`(*patch_size*=7, *in_channels*=3, *embedding_dim*=64, *stride*=4, *padding*=2)

Projects image patches into embedding space using convolutional layer.

Parameters

- **patch_size** (*int*, *default is 7*) – Size of a patch
- **in_channels** (*int*, *default is 3*) – Number of input channels
- **embedding_dim** (*int*, *default is 64*) – Dimension of hidden layer
- **stride** (*int or tuple*, *default is 4*) – Stride of the convolution operation
- **padding** (*int*, *default is 2*) – Padding to all sides of the input

forward(*x*)

Parameters

x (*torch.tensor*) – Input tensor

Returns

Returns output tensor (embedding) by applying a convolution operations on input tensor

Return type

torch.Tensor

6.1.6 Video Patch Embeddings

```
class vformer.encoder.embedding.video_patch_embeddings.LinearVideoEmbedding(embedding_dim,
                                                                            patch_height,
                                                                            patch_width,
                                                                            patch_dim)
```

Parameters

- **embedding_dim** (*int*) – Dimension of the resultant embedding
- **patch_height** (*int*) – Height of the patch
- **patch_width** (*int*) – Width of the patch
- **patch_dim** (*int*) – patch_dimension

forward(*x*)**Parameters**

x (*torch.Tensor*) – Input tensor

Returns

Returns patch embeddings of size *embedding_dim*

Return type

torch.Tensor

```
class vformer.encoder.embedding.video_patch_embeddings.TubeletEmbedding(embedding_dim,
                                                                        tubelet_t, tubelet_h,
                                                                        tubelet_w,
                                                                        in_channels)
```

Parameters

- **embedding_dim** (*int*) – Dimension of the resultant embedding
- **tubelet_t** (*int*) – Temporal length of single tube/patch
- **tubelet_h** (*int*) – Height of single tube/patch
- **tubelet_w** (*int*) – Width of single tube/patch
- **in_channels** (*int*) – Number of channels

forward(*x*)**Parameters**

x (*Torch.tensor*) – Input tensor

6.2 Cross Encoder

```
class vformer.encoder.cross.CrossEncoder(embedding_dim_s=1024, embedding_dim_l=1024,
                                         attn_heads_s=16, attn_heads_l=16, cross_head_s=8,
                                         cross_head_l=8, head_dim_s=64, head_dim_l=64,
                                         cross_dim_head_s=64, cross_dim_head_l=64, depth_s=6,
                                         depth_l=6, mlp_dim_s=2048, mlp_dim_l=2048,
                                         p_dropout_s=0.0, p_dropout_l=0.0)
```

Encoder block used in Cross-ViT .

Parameters

- **embedding_dim_s** (*int*) – Dimension of the embedding of smaller patches, default is 1024
- **embedding_dim_l** (*int*) – Dimension of the embedding of larger patches, default is 1024
- **attn_heads_s** (*int*) – Number of self-attention heads for the smaller patches, default is 16
- **attn_heads_l** (*int*) – Number of self-attention heads for the larger patches, default is 16
- **cross_head_s** (*int*) – Number of cross-attention heads for the smaller patches, default is 8
- **cross_head_l** (*int*) – Number of cross-attention heads for the larger patches, default is 8
- **head_dim_s** (*int*) – Dimension of the head of the attention for the smaller patches, default is 64
- **head_dim_l** (*int*) – Dimension of the head of the attention for the larger patches, default is 64
- **cross_dim_head_s** (*int*) – Dimension of the head of the cross-attention for the smaller patches, default is 64
- **cross_dim_head_l** (*int*) – Dimension of the head of the cross-attention for the larger patches, default is 64
- **depth_s** (*int*) – Number of self-attention layers in encoder for the smaller patches, default is 6
- **depth_l** (*int*) – Number of self-attention layers in encoder for the larger patches, default is 6
- **mlp_dim_s** (*int*) – Dimension of the hidden layer in the feed-forward layer for the smaller patches, default is 2048
- **mlp_dim_l** (*int*) – Dimension of the hidden layer in the feed-forward layer for the larger patches, default is 2048
- **p_dropout_s** (*float*) – Dropout probability for the smaller patches, default is 0.0
- **p_dropout_l** (*float*) – Dropout probability for the larger patches, default is 0.0

forward(*emb_s, emb_l*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

6.3 FeedForward MLP

```
class vformer.encoder.nn.FeedForward(dim, hidden_dim=None, out_dim=None, p_dropout=0.0)
```

Parameters

- **dim** (*int*) – Dimension of the input tensor
- **hidden_dim** (*int, optional*) – Dimension of hidden layer
- **out_dim** (*int, optional*) – Dimension of the output tensor
- **p_dropout** (*float*) – Dropout probability, default=0.0

```
forward(x)
```

Parameters

x (*torch.Tensor*) – Input tensor

Returns

Returns output tensor by performing linear operations and activation on input tensor

Return type

torch.Tensor

6.4 Pyramid Encoder

```
class vformer.encoder.pyramid.PVTEncoder(dim, num_heads, mlp_ratio, depth, qkv_bias, qk_scale,
                                         p_dropout, attn_dropout, drop_path, activation, use_dwconv,
                                         sr_ratio, linear=False, drop_path_mode='batch')
```

Parameters

- **dim** (*int*) – Dimension of the input tensor
- **num_heads** (*int*) – Number of attention heads
- **mlp_ratio** – Ratio of MLP hidden dimension to embedding dimension
- **depth** (*int*) – Number of attention layers in the encoder
- **qkv_bias** (*bool*) – Whether to add a bias vector to the q,k, and v matrices
- **qk_scale** (*float, optional*) – Override default qk scale of head_dim ** -0.5 in Spatial Attention if set
- **p_dropout** (*float*) – Dropout probability
- **attn_dropout** (*float*) – Dropout probability
- **drop_path** (*tuple(float)*) – List of stochastic drop rate
- **activation** (*nn.Module*) – Activation layer
- **use_dwconv** (*bool*) – Whether to use depth-wise convolutions in overlap-patch embedding
- **sr_ratio** (*float*) – Spatial Reduction ratio
- **linear** (*bool*) – Whether to use linear Spatial attention, default is `False`.
- **drop_path_mode** (*str*) – Mode for *StochasticDepth* https://pytorch.org/vision/main/generated/torchvision.ops.stochastic_depth.html, must be one of {'batch' or 'row'}

forward(*x*, ***kwargs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

```
class vformer.encoder.pyramid.PVTFeedForward(dim, hidden_dim=None, out_dim=None,
                                             activation=<class 'torch.nn.modules.activation.GELU'>,
                                             p_dropout=0.0, linear=False, use_dwconv=False,
                                             **kwargs)
```

Parameters

- **dim** (*int*) – Dimension of the input tensor
- **hidden_dim** (*int*, *optional*) – Dimension of hidden layer
- **out_dim** (*int*, *optional*) – Dimension of output tensor
- **activation** (*nn.Module*) – Activation Layer, default is `nn.GELU`
- **p_dropout** (*float*) – Dropout probability/rate, default is 0.0
- **linear** (*bool*) – Whether to use linear Spatial attention, default is `False`
- **use_dwconv** (*bool*) – Whether to use Depth-wise convolutions, default is `False`
- **kernel_size_dwconv** (*int*) – *kernel_size* parameter for 2D convolution used in Depth wise convolution
- **stride_dwconv** (*int*) – *stride* parameter for 2D convolution used in Depth wise convolution
- **padding_dwconv** (*int*) – *padding* parameter for 2D convolution used in Depth wise convolution
- **bias_dwconv** (*bool*) – *bias* parameter for 2D convolution used in Depth wise convolution

forward(*x*, ***kwargs*)

Parameters

- **x** (*torch.Tensor*) – Input tensor
- **H** (*int*) – Height of image patch
- **W** (*int*) – Width of image patch

Returns

Returns output tensor

Return type

`torch.Tensor`

6.5 Perceiver IO Encoder

```
class vformer.encoder.perceiver_io.PerceiverIOEncoder(dim=32, depth=6, latent_dim=512,
                                                    num_latents=512, num_cross_heads=1,
                                                    num_latent_heads=8, cross_head_dim=64,
                                                    latent_head_dim=64)
```

Implementation of the Perceiver IO Encoder containing Iterative Cross Attention and Processor

Parameters

- **dim** (*int*) – Size of sequence to be encoded
- **depth** (*int*) – Depth of latent attention blocks
- **latent_dim** (*int*) – Dimension of latent array
- **num_latents** (*int*) – Number of latent arrays
- **num_cross_heads** (*int*) – Number of heads for cross attention
- **num_latent_heads** (*int*) – Number of heads for latent attention
- **cross_head_dim** (*int*) – Dimension of cross attention head
- **latent_head_dim** (*int*) – Dimension of latent attention head

forward(*x*, *mask=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

6.6 Swin Encoder

```
class vformer.encoder.swin.SwinEncoder(dim, input_resolution, depth, num_heads, window_size,
                                       mlp_ratio=4.0, qkv_bias=True, qkv_scale=None, p_dropout=0.0,
                                       attn_dropout=0.0, drop_path=0.0, norm_layer=<class
                                       'torch.nn.modules.normalization.LayerNorm'>,
                                       downsample=None, use_checkpoint=False)
```

Parameters

- **dim** (*int*) – Number of input channels.
- **input_resolution** (*tuple[int]*) – Input resolution.
- **depth** (*int*) – Number of blocks.
- **num_heads** (*int*) – Number of attention heads.
- **window_size** (*int*) – Local window size.
- **mlp_ratio** (*float*) – Ratio of MLP hidden dim to embedding dim.
- **qkv_bias** (*bool*, *default is True*) – Whether to add a bias vector to the q,k, and v matrices

- **qk_scale** (*float, optional*) – Override default qk scale of $\text{head_dim}^{**} -0.5$ in Window Attention if set
- **p_dropout** (*float, ,*) – Dropout rate.
- **attn_dropout** (*float, optional*) – Attention dropout rate
- **drop_path_rate** (*float or tuple[float]*) – Stochastic depth rate.
- **norm_layer** (*nn.Module*) – Normalization layer. default is `nn.LayerNorm`
- **downsample** (*nn.Module, optional*) – Downsample layer(like PatchMerging) at the end of the layer, default is `None`

forward(*x*)

Parameters

x (*torch.Tensor*) –

Returns

Returns output tensor

Return type

`torch.Tensor`

```
class vformer.encoder.swin.SwinEncoderBlock(dim, input_resolution, num_heads, window_size=7,
shift_size=0, mlp_ratio=4.0, qkv_bias=True,
qk_scale=None, p_dropout=0.0, attn_dropout=0.0,
drop_path_rate=0.0, norm_layer=<class
'torch.nn.modules.normalization.LayerNorm'>,
drop_path_mode='batch'))
```

Parameters

- **dim** (*int*) – Number of the input channels
- **input_resolution** (*int or tuple[int]*) – Input resolution of patches
- **num_heads** (*int*) – Number of attention heads
- **window_size** (*int*) – Window size
- **shift_size** (*int*) – Shift size for Shifted Window Masked Self Attention (SW_MSA)
- **mlp_ratio** (*float*) – Ratio of MLP hidden dimension to embedding dimension
- **qkv_bias** (*bool, default= True*) – Whether to add a bias vector to the q,k, and v matrices
- **qk_scale** (*float, Optional*) –
- **p_dropout** (*float*) – Dropout rate
- **attn_dropout** (*float*) – Dropout rate
- **drop_path_rate** (*float*) – Stochastic depth rate
- **norm_layer** (*nn.Module*) – Normalization layer, default is `nn.LayerNorm`

forward(*x*)

Parameters

x (*torch.Tensor*) –

Returns

Returns output tensor

Return type
torch.Tensor

6.7 Vanilla Encoder

```
class vformer.encoder.vanilla.VanillaEncoder(embedding_dim, depth, num_heads, head_dim, mlp_dim,  
                                             p_dropout=0.0, attn_dropout=0.0, drop_path_rate=0.0,  
                                             drop_path_mode='batch')
```

Parameters

- **embedding_dim** (*int*) – Dimension of the embedding
- **depth** (*int*) – Number of self-attention layers
- **num_heads** (*int*) – Number of the attention heads
- **head_dim** (*int*) – Dimension of each head
- **mlp_dim** (*int*) – Dimension of the hidden layer in the feed-forward layer
- **p_dropout** (*float*) – Dropout Probability
- **attn_dropout** (*float*) – Dropout Probability
- **drop_path_rate** (*float*) – Stochastic drop path rate

forward(*x*)

Parameters
x (*torch.Tensor*) –

Returns
Returns output tensor

Return type
torch.Tensor

6.8 ConViT

```
class vformer.encoder.convit.ConViTEncoder(embedding_dim, depth, num_heads, head_dim, mlp_dim,  
                                             p_dropout=0, attn_dropout=0, drop_path_rate=0,  
                                             drop_path_mode='batch')
```

Parameters

- **embedding_dim** (*int*) – Dimension of the embedding
- **depth** (*int*) – Number of self-attention layers
- **num_heads** (*int*) – Number of the attention heads
- **head_dim** (*int*) – Dimension of each head
- **mlp_dim** (*int*) – Dimension of the hidden layer in the feed-forward layer
- **p_dropout** (*float*) – Dropout Probability
- **attn_dropout** (*float*) – Dropout Probability
- **drop_path_rate** (*float*) – Stochastic drop path rate

6.9 ConvVT

```
class vformer.encoder.convvt.ConvVTBlock(dim_in, dim_out, mlp_ratio=4.0, p_dropout=0.0,  
                                         drop_path=0.0, drop_path_mode='batch', **kwargs)
```

Implementation of a Attention MLP block in CVT

Parameters

- **dim_in** (*int*) – Input dimensions
- **dim_out** (*int*) – Output dimensions
- **num_heads** (*int*) – Number of heads in attention
- **img_size** (*int*) – Size of image
- **mlp_ratio** (*float*) – Feature dimension expansion ratio in MLP, default is 4.
- **p_dropout** (*float*) – Probability of dropout in MLP, default is 0.0
- **attn_dropout** (*float*) – Probability of dropout in attention, default is 0.0
- **drop_path** (*float*) – Probability of droppath, default is 0.0
- **with_cls_token** (*bool*) – Whether to include classification token, default is False

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

```
class vformer.encoder.convvt.ConvVTStage(patch_size=7, patch_stride=4, patch_padding=0,  
                                         in_channels=3, embedding_dim=64, depth=1, p_dropout=0.0,  
                                         drop_path_rate=0.0, with_cls_token=False, init='trunc_norm',  
                                         **kwargs)
```

Implementation of a Stage in CVT

Parameters

- **patch_size** (*int*) – Size of patch, default is 16
- **patch_stride** (*int*) – Stride of patch, default is 4
- **patch_padding** (*int*) – Padding for patch, default is 0
- **in_channels** (*int*) – Number of input channels in image, default is 3
- **img_size** (*int*) – Size of the image, default is 224
- **embedding_dim** (*int*) – Embedding dimensions, default is 64
- **depth** (*int*) – Number of CVT Attention blocks in each stage, default is 1
- **num_heads** (*int*) – Number of heads in attention, default is 6
- **mlp_ratio** (*float*) – Feature dimension expansion ratio in MLP, default is 4.0
- **p_dropout** (*float*) – Probability of dropout in MLP, default is 0.0

- **attn_dropout** (*float*) – Probability of dropout in attention, default is 0.0
- **drop_path_rate** (*float*) – Probability for droppath, default is 0.0
- **with_cls_token** (*bool*) – Whether to include classification token, default is False
- **kernel_size** (*int*) – Size of kernel, default is 3
- **padding_q** (*int*) – Size of padding in q, default is 1
- **padding_kv** (*int*) – Size of padding in kv, default is 2
- **stride_kv** (*int*) – Stride in kv, default is 2
- **stride_q** (*int*) – Stride in q, default is 1
- **init** (*str*) – Initialization method, one of { `trunc_norm` or `xavier` } default is `trunc_norm`

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

7.1 Patch Merging

```
class vformer.functional.merge.PatchMerging(input_resolution, dim, norm_layer=<class  
                                         'torch.nn.modules.normalization.LayerNorm'>)
```

Parameters

- **input_resolution** (*int* or *tuple[int]*) – Resolution of input features
- **dim** (*int*) –

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

7.2 Normalization Layers

```
class vformer.functional.norm.PreNorm(dim, fn, context_dim=None)
```

Parameters

- **dim** (*int*) – Dimension of the embedding
- **fn** (*nn.Module*) – Attention class
- **context_dim** (*int*) – Dimension of the context array used in cross attention

forward(*x*, ***kwargs*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

8.1 Classification

8.1.1 Compact Convolutional Transformer

```
class vformer.models.classification.cct.CCT(img_size=224, patch_size=4, in_channels=3,  
                                           seq_pool=True, embedding_dim=768, num_layers=1,  
                                           head_dim=96, num_heads=1, mlp_ratio=4.0,  
                                           n_classes=1000, p_dropout=0.1, attn_dropout=0.1,  
                                           drop_path=0.1, positional_embedding='learnable',  
                                           decoder_config=(768, 1024), pooling_kernel_size=3,  
                                           pooling_stride=2, pooling_padding=1)
```

Implementation of [Escaping the Big Data Paradigm with Compact Transformers](#)

Parameters

- **img_size** (*int*) – Size of the image
- **patch_size** (*int*) – Size of the single patch in the image
- **in_channels** (*int*) – Number of input channels in image
- **seq_pool** (*bool*) – Whether to use sequence pooling or not
- **embedding_dim** (*int*) – Patch embedding dimension
- **num_layers** (*int*) – Number of Encoders in encoder block
- **num_heads** (*int*) – Number of heads in each transformer layer
- **mlp_ratio** (*float*) – Ratio of mlp heads to embedding dimension
- **n_classes** (*int*) – Number of classes for classification
- **p_dropout** (*float*) – Dropout probability
- **attn_dropout** (*float*) – Dropout probability
- **drop_path** (*float*) – Stochastic depth rate, default is 0.1
- **positional_embedding** (*str*) – One of the string values {'learnable', 'sine', None}, default is 'learnable'.
- **decoder_config** (*tuple(int) or int*) – Configuration of the decoder. If None, the default configuration is used.
- **pooling_kernel_size** (*int or tuple(int)*) – Size of the kernel in MaxPooling operation

- **pooling_stride** (*int* or *tuple(int)*) – Stride of MaxPooling operation
- **pooling_padding** (*int*) – Padding in MaxPooling operation

forward(*x*)

Parameters

x (*torch.Tensor*) – Input tensor

Returns

Returns tensor of size *n_classes*

Return type

torch.Tensor

8.1.2 Cross-Attention Transformer

```
class vformer.models.classification.cross.CrossViT(img_size, patch_size_s, patch_size_l, n_classes,
                                                  cross_dim_head_s=64, cross_dim_head_l=64,
                                                  latent_dim_s=1024, latent_dim_l=1024,
                                                  head_dim_s=64, head_dim_l=64, depth_s=6,
                                                  depth_l=6, attn_heads_s=16, attn_heads_l=16,
                                                  cross_head_s=8, cross_head_l=8,
                                                  encoder_mlp_dim_s=2048,
                                                  encoder_mlp_dim_l=2048, in_channels=3,
                                                  decoder_config_s=None,
                                                  decoder_config_l=None, pool_s='cls',
                                                  pool_l='cls', p_dropout_encoder_s=0.0,
                                                  p_dropout_encoder_l=0.0,
                                                  p_dropout_embedding_s=0.0,
                                                  p_dropout_embedding_l=0.0)
```

Implementation of CrossViT: Cross-Attention Multi-Scale Vision Transformer for Image Classification

Parameters

- **img_size** (*int*) – Size of the image
- **patch_size_s** (*int*) – Size of the smaller patches
- **patch_size_l** (*int*) – Size of the larger patches
- **n_classes** (*int*) – Number of classes for classification
- **cross_dim_head_s** (*int*) – Dimension of the head of the cross-attention for the smaller patches
- **cross_dim_head_l** (*int*) – Dimension of the head of the cross-attention for the larger patches
- **latent_dim_s** (*int*) – Dimension of the hidden layer for the smaller patches
- **latent_dim_l** (*int*) – Dimension of the hidden layer for the larger patches
- **head_dim_s** (*int*) – Dimension of the head of the attention for the smaller patches
- **head_dim_l** (*int*) – Dimension of the head of the attention for the larger patches
- **depth_s** (*int*) – Number of attention layers in encoder for the smaller patches
- **depth_l** (*int*) – Number of attention layers in encoder for the larger patches
- **attn_heads_s** (*int*) – Number of attention heads for the smaller patches

- **attn_heads_l** (*int*) – Number of attention heads for the larger patches
- **cross_head_s** (*int*) – Number of CrossAttention heads for the smaller patches
- **cross_head_l** (*int*) – Number of CrossAttention heads for the larger patches
- **encoder_mlp_dim_s** (*int*) – Dimension of hidden layer in the encoder for the smaller patches
- **encoder_mlp_dim_l** (*int*) – Dimension of hidden layer in the encoder for the larger patches
- **in_channels** (*int*) – Number of input channels
- **decoder_config_s** (*int or tuple or list, optional*) – Configuration of the decoder for the smaller patches
- **decoder_config_l** (*int or tuple or list, optional*) – Configuration of the decoder for the larger patches
- **pool_s** (*str*) – Feature pooling type for the smaller patches, one of {cls, ``mean``}
- **pool_l** (*str*) – Feature pooling type for the larger patches, one of {cls, ``mean``}
- **p_dropout_encoder_s** (*float*) – Dropout probability in the encoder for the smaller patches
- **p_dropout_encoder_l** (*float*) – Dropout probability in the encoder for the larger patches
- **p_dropout_embedding_s** (*float*) – Dropout probability in the embedding layer for the smaller patches
- **p_dropout_embedding_l** (*float*) – Dropout probability in the embedding layer for the larger patches

forward(*img*)

Parameters

img (*torch.Tensor*) – Input tensor

Returns

Returns tensor of size *n_classes*

Return type

torch.Tensor

8.1.3 Compact Vision Transformer

```
class vformer.models.classification.cvt.CVT(img_size=224, patch_size=4, in_channels=3,
                                          seq_pool=True, embedding_dim=768, head_dim=96,
                                          num_layers=1, num_heads=1, mlp_ratio=4.0,
                                          n_classes=1000, p_dropout=0.1, attn_dropout=0.1,
                                          drop_path=0.1, positional_embedding='learnable',
                                          decoder_config=(768, 1024))
```

Implementation of [Escaping the Big Data Paradigm with Compact Transformers](#)

Parameters

- **img_size** (*int*) – Size of the image, default is 224
- **patch_size** (*int*) – Size of the single patch in the image, default is 4
- **in_channels** (*int*) – Number of input channels in image, default is 3

- **seq_pool** (*bool*) – Whether to use sequence pooling, default is True
- **embedding_dim** (*int*) – Patch embedding dimension, default is 768
- **num_layers** (*int*) – Number of Encoders in encoder block, default is 1
- **num_heads** (*int*) – Number of heads in each transformer layer, default is 1
- **mlp_ratio** (*float*) – Ratio of mlp heads to embedding dimension, default is 4.0
- **n_classes** (*int*) – Number of classes for classification, default is 1000
- **p_dropout** (*float*) – Dropout probability, default is 0.0
- **attn_dropout** (*float*) – Dropout probability, default is 0.0
- **drop_path** (*float*) – Stochastic depth rate, default is 0.1
- **positional_embedding** (*str*) – One of the string values {'learnable', 'sine', None}, default is 'learnable'
- **decoder_config** (*tuple(int) or int*) – Configuration of the decoder. If None, the default configuration is used.

forward(*x*)

Parameters

x (*torch.Tensor*) – Input tensor

Returns

Returns tensor of size *n_classes*

Return type

torch.Tensor

8.1.4 Pyramid Vision Transformer

```
class vformer.models.classification.pyramid.PVTClassification(img_size=224, patch_size=[7, 3, 3,
3], in_channels=3,
n_classes=1000, embed_dims=[64,
128, 256, 512], num_heads=[1, 2,
4, 8], mlp_ratio=[4, 4, 4, 4],
qkv_bias=False, qk_scale=None,
p_dropout=0.0, attn_dropout=0.0,
drop_path_rate=0.0,
norm_layer=<class
'torch.nn.modules.normalization.LayerNorm'>,
depths=[3, 4, 6, 3], sr_ratios=[8, 4,
2, 1], decoder_config=None,
linear=False, use_dwconv=False,
ape=True)
```

Implementation of Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolution

Parameters

- **img_size** (*int*) – Image size
- **patch_size** (*list(int)*) – List of patch size
- **in_channels** (*int*) – Input channels in image, default=3

- **n_classes** (*int*) – Number of classes for classification
- **embed_dims** (*int*) – Patch Embedding dimension
- **num_heads** (*tuple[int]*) – Number of heads in each transformer layer
- **depths** (*tuple[int]*) – Depth in each Transformer layer
- **mlp_ratio** (*float*) – Ratio of mlp heads to embedding dimension
- **qkv_bias** (*bool*, *default= True*) – Adds bias to the qkv if true
- **qk_scale** (*float*, *optional*) – Override default qk scale of head_dim ** -0.5 Spatial Attention if set
- **p_dropout** (*float*,) – Dropout rate,default is 0.0
- **attn_dropout** (*float*,) – Attention dropout rate, default is 0.0
- **drop_path_rate** (*float*) – Stochastic depth rate, default is 0.1
- **norm_layer** – Normalization layer, default is nn.LayerNorm
- **sr_ratios** (*float*) – Spatial reduction ratio
- **decoder_config** (*int or tuple[int]*, *optional*) – Configuration of the decoder. If None, the default configuration is used.
- **linear** (*bool*) – Whether to use linear Spatial attention, default is False
- **use_dwconv** (*bool*) – Whether to use Depth-wise convolutions, default is False
- **ape** (*bool*) – Whether to use absolute position embedding, default is True

forward(*x*)

Parameters

x (*torch.Tensor*) – Input tensor

Returns

Returns tensor of size *n_classes*

Return type

torch.Tensor

```
class vformer.models.classification.pyramid.PVTClassificationV2(img_size=224, patch_size=[7, 3, 3], in_channels=3, n_classes=1000, embedding_dims=[64, 128, 256, 512], num_heads=[1, 2, 4, 8], mlp_ratio=[4, 4, 4, 4], qkv_bias=False, qk_scale=0.0, p_dropout=0.0, attn_dropout=0.0, drop_path_rate=0.0, norm_layer=<class 'torch.nn.modules.normalization.LayerNorm'>, depths=[3, 4, 6, 3], sr_ratios=[8, 4, 2, 1], decoder_config=None, use_dwconv=True, linear=False, ape=False)
```

Implementation of PVT v2: Improved Baselines with Pyramid Vision Transformer

Parameters

- **img_size** (*int*) – Image size
- **patch_size** (*list(int)*) – List of patch size
- **in_channels** (*int*) – Input channels in image, default is 3
- **n_classes** (*int*) – Number of classes for classification
- **embedding_dims** (*int*) – Patch Embedding dimension
- **num_heads** (*tuple[int]*) – Number of heads in each transformer layer
- **depths** (*tuple[int]*) – Depth in each Transformer layer
- **mlp_ratio** (*float*) – Ratio of mlp heads to embedding dimension
- **qkv_bias** (*bool*, *default= True*) – Adds bias to the qkv if true
- **qk_scale** (*float*, *optional*) – Override default qk scale of head_dim ** -0.5 in Spatial Attention if set
- **p_dropout** (*float*,) – Dropout rate,default is 0.0
- **attn_dropout** (*float*,) – Attention dropout rate, default is 0.0
- **drop_path_rate** (*float*) – Stochastic depth rate, default is 0.1
- **norm_layer** (*nn.Module*) – Normalization layer, default is nn.LayerNorm
- **sr_ratios** (*float*) – Spatial reduction ratio
- **decoder_config** (*int or tuple[int]*, *optional*) – Configuration of the decoder. If None, the default configuration is used.
- **linear** (*bool*) – Whether to use linear Spatial attention, default is False
- **use_dwconv** (*bool*) – Whether to use Depth-wise convolutions, default is True
- **ape** (*bool*) – Whether to use absolute position embedding, default is false

8.1.5 Swin Transformer

```
class vformer.models.classification.swin.SwinTransformer(img_size, patch_size, in_channels,  
                                                    n_classes, embedding_dim=96,  
                                                    depths=[2, 2, 6, 2], num_heads=[3, 6, 12,  
                                                    24], window_size=8, mlp_ratio=4.0,  
                                                    qkv_bias=True, qk_scale=None,  
                                                    p_dropout=0.0, attn_dropout=0.0,  
                                                    drop_path_rate=0.1, norm_layer=<class  
'torch.nn.modules.normalization.LayerNorm'>,  
                                                    ape=True, decoder_config=None,  
                                                    patch_norm=True)
```

Implementation of Swin Transformer: Hierarchical Vision Transformer using Shifted Windows <<https://arxiv.org/abs/2103.14030v1>>`_

Parameters

- **img_size** (*int*) – Size of an Image
- **patch_size** (*int*) – Patch Size
- **in_channels** (*int*) – Input channels in image, default=3
- **n_classes** (*int*) – Number of classes for classification

- **embedding_dim** (*int*) – Patch Embedding dimension
- **depths** (*tuple[int]*) – Depth in each Transformer layer
- **num_heads** (*tuple[int]*) – Number of heads in each transformer layer
- **window_size** (*int*) – Window Size
- **mlp_ratio** (*float*) – Ratio of mlp heads to embedding dimension
- **qkv_bias** (*bool*, *default= True*) – Adds bias to the qkv if true
- **qk_scale** (*float*, *optional*) – Override default qk scale of head_dim ** -0.5 in Window Attention if set
- **p_dropout** (*float*) – Dropout rate, default is 0.0
- **attn_dropout** (*float*) – Attention dropout rate, default is 0.0
- **drop_path_rate** (*float*) – Stochastic depth rate, default is 0.1
- **norm_layer** (*nn.Module*) – Normalization layer, default is nn.LayerNorm
- **ape** (*bool*, *optional*) – Whether to add relative/absolute position embedding to patch embedding, default is True
- **decoder_config** (*int or tuple[int]*, *optional*) – Configuration of the decoder. If None, the default configuration is used.
- **patch_norm** (*bool*, *optional*) – Whether to add Normalization layer in PatchEmbedding, default is True

forward(*x*)

Parameters

x (*torch.Tensor*) – Input tensor

Returns

Returns tensor of size *n_classes*

Return type

torch.Tensor

8.1.6 Vanilla Vision Transformer

```
class vformer.models.classification.vanilla.VanillaViT(img_size, patch_size, n_classes,
                                                    embedding_dim=1024, head_dim=64,
                                                    depth=6, num_heads=16,
                                                    encoder_mlp_dim=2048, in_channels=3,
                                                    decoder_config=None, pool='cls',
                                                    p_dropout_encoder=0.0,
                                                    p_dropout_embedding=0.0)
```

Implementation of [An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale](#)

Parameters

- **img_size** (*int*) – Size of the image
- **patch_size** (*int*) – Size of a patch
- **n_classes** (*int*) – Number of classes for classification
- **embedding_dim** (*int*) – Dimension of hidden layer

- **head_dim** (*int*) – Dimension of the attention head
- **depth** (*int*) – Number of attention layers in the encoder
- **num_heads** (*int*) – Number of the attention heads
- **encoder_mlp_dim** (*int*) – Dimension of hidden layer in the encoder
- **in_channels** (*int*) – Number of input channels
- **decoder_config** (*int or tuple or list, optional*) – Configuration of the decoder. If None, the default configuration is used.
- **pool** (*str*) – Feature pooling type, one of {cls, ``mean``}
- **p_dropout_encoder** (*float*) – Dropout probability in the encoder
- **p_dropout_embedding** (*float*) – Dropout probability in the embedding layer

forward(*x*)

Parameters

x (*torch.Tensor*) – Input tensor

Returns

Returns tensor of size *n_classes*

Return type

torch.Tensor

8.1.7 Vision-friendly Transformer

```
class vformer.models.classification.visformer.Visformer(img_size, n_classes, depth: tuple, config: tuple, channel_config: tuple, num_heads=8, conv_group=8, p_dropout_conv=0.0, p_dropout_attn=0.0, activation=<class 'torch.nn.modules.activation.GELU'>, pos_embedding=True)
```

A builder to construct a Vision-Friendly transformer model as in the paper :[Visformer: The Vision-friendly Transformer](#)

Parameters

- **img_size** (*int, tuple*) – Size of the input image
- **n_classes** (*int*) – Number of classes in the dataset
- **depth** (*tuple[int]*) – Number of layers before each embedding reduction
- **config** (*tuple[int]*) – Choice of convolution block (0) or attention block (1) for corresponding layer
- **channel_config** (*tuple[int]*) – Number of channels for each layer
- **num_heads** (*int*) – Number of heads for attention block, default is 8
- **conv_group** (*int*) – Number of groups for convolution block, default is 8
- **p_dropout_conv** (*float*) – Dropout rate for convolution block, default is 0.0
- **p_dropout_attn** (*float*) – Dropout rate for attention block, default is 0.0
- **activation** (*nn.Module*) – Activation function between layers, default is nn.GELU

- **pos_embedding** (*bool*) – Whether to use positional embedding, default is True

forward(*x*)

Parameters

x (*torch.Tensor*) – Input tensor

Returns

Returns tensor of size *n_classes*

Return type

torch.Tensor

```
class vformer.models.classification.visformer.VisformerAttentionBlock(in_channels,
                                                                    num_heads=8,
                                                                    activation=<class
                                                                    'torch.nn.modules.activation.GELU'>,
                                                                    p_dropout=0.0)
```

Attention Block for Vision-Friendly transformers <https://arxiv.org/abs/2104.12533>

Parameters

- **in_channels** (*int*) – Number of input channels
- **num_heads** (*int*) – Number of heads for attention, default is 8
- **activation** (*torch.nn.Module*) – Activation function between layers, default is *nn.GELU*
- **p_dropout** (*float*) – Dropout rate, default is 0.0

forward(*x*)

Parameters

x (*torch.Tensor*) – Input tensor

Returns

Returns tensor of same size as input

Return type

torch.Tensor

```
class vformer.models.classification.visformer.VisformerConvBlock(in_channels, group=8,
                                                                    activation=<class
                                                                    'torch.nn.modules.activation.GELU'>,
                                                                    p_dropout=0.0)
```

Convolution Block for Vision-Friendly transformers <https://arxiv.org/abs/2104.12533>

Parameters

- **in_channels** (*int*) – Number of input channels
- **group** (*int*) – Number of groups for convolution, default is 8
- **activation** (*torch.nn.Module*) – Activation function between layers, default is *nn.GELU*
- **p_dropout** (*float*) – Dropout rate, default is 0.0

forward(*x*)

Parameters

x (*torch.Tensor*) – Input tensor

Returns

Returns tensor of same size as input

Return type

torch.Tensor

`vformer.models.classification.visformer.VisformerV2_S(img_size, n_classes, in_channels=3)`

VisformerV2-S model from the paper: [Visformer: The Vision-friendly Transformer](#)

Parameters

- **img_size** (*int, tuple*) – Size of the input image
- **n_classes** (*int*) – Number of classes in the dataset
- **in_channels** (*int*) – Number of channels in the input

`vformer.models.classification.visformer.VisformerV2_Ti(img_size, n_classes, in_channels=3)`

VisformerV2-Ti model from the paper: [Visformer: The Vision-friendly Transformer](#)

Parameters

- **img_size** (*int, tuple*) – Size of the input image
- **n_classes** (*int*) – Number of classes in the dataset
- **in_channels** (*int*) – Number of channels in the input

`vformer.models.classification.visformer.Visformer_S(img_size, n_classes, in_channels=3)`

Visformer-S model from the paper: [Visformer: The Vision-friendly Transformer](#)

Parameters

- **img_size** (*int, tuple*) – Size of the input image
- **n_classes** (*int*) – Number of classes in the dataset
- **in_channels** (*int*) – Number of channels in the input

`vformer.models.classification.visformer.Visformer_Ti(img_size, n_classes, in_channels=3)`

Visformer-Ti model from the paper: [Visformer: The Vision-friendly Transformer](#)

Parameters

- **img_size** (*int, tuple*) – Size of the input image
- **n_classes** (*int*) – Number of classes in the dataset
- **in_channels** (*int*) – Number of channels in the input

8.1.8 ConViT

```
class vformer.models.classification.convit.ConViT(img_size, patch_size, n_classes,
                                                embedding_dim=1024, head_dim=64, depth_sa=6,
                                                depth_gpsa=6, attn_heads_sa=16,
                                                attn_heads_gpsa=16, encoder_mlp_dim=2048,
                                                in_channels=3, decoder_config=None, pool='cls',
                                                p_dropout_encoder=0, p_dropout_embedding=0)
```

Implementation of ConViT: [Improving Vision Transformers with Soft Convolutional Inductive Biases](#)

Parameters

- **img_size** (*int*) – Size of the image

- **patch_size** (*int*) – Size of a patch
- **n_classes** (*int*) – Number of classes for classification
- **embedding_dim** (*int*) – Dimension of hidden layer
- **head_dim** (*int*) – Dimension of the attention head
- **depth_sa** (*int*) – Number of attention layers in the encoder for self attention layers
- **depth_gpsa** (*int*) – Number of attention layers in the encoder for global positional self attention layers
- **attn_heads_sa** (*int*) – Number of the attention heads for self attention layers
- **attn_heads_gpsa** (*int*) – Number of the attention heads for global positional self attention layers
- **encoder_mlp_dim** (*int*) – Dimension of hidden layer in the encoder
- **in_channels** (*int*) – Number of input channels
- **decoder_config** (*int or tuple or list, optional*) – Configuration of the decoder. If None, the default configuration is used.
- **pool** (*str*) – Feature pooling type, one of {cls, ``mean``}
- **p_dropout_encoder** (*float*) – Dropout probability in the encoder
- **p_dropout_embedding** (*float*) – Dropout probability in the embedding layer

forward(*x*)

Parameters

x (*torch.Tensor*) – Input tensor

Returns

Returns tensor of size *n_classes*

Return type

torch.Tensor

8.1.9 Convolutional Vision Transformer

```
class vformer.models.classification.convvt.ConvVT(img_size=224, patch_size=[7, 3, 3],
                                                patch_stride=[4, 2, 2], patch_padding=[2, 1, 1],
                                                embedding_dim=[64, 192, 384], num_heads=[1, 3,
                                                6], depth=[1, 2, 10], mlp_ratio=[4.0, 4.0, 4.0],
                                                p_dropout=[0, 0, 0], attn_dropout=[0, 0, 0],
                                                drop_path_rate=[0, 0, 0.1], kernel_size=[3, 3, 3],
                                                padding_q=[1, 1, 1], padding_kv=[1, 1, 1],
                                                stride_kv=[2, 2, 2], stride_q=[1, 1, 1],
                                                in_channels=3, num_stages=3, n_classes=1000)
```

Implementation of CvT: [Introducing Convolutions to Vision Transformers](#)

Parameters

- **img_size** (*int*) – Size of the image, default is 224
- **in_channels** (*int*) – Number of input channels in image, default is 3
- **num_stages** (* *The following are all in list of int/float with length*) – Number of stages in encoder block, default is 3

- **n_classes** (*int*) – Number of classes for classification, default is 1000
- **num_stages** –
- **patch_size** (*list[int]*) – Size of patch, default is [7, 3, 3]
- **patch_stride** (*list[int]*) – Stride of patch, default is [4, 2, 2]
- **patch_padding** (*list[int]*) – Padding for patch, default is [2, 1, 1]
- **embedding_dim** (*list[int]*) – Embedding dimensions, default is [64, 192, 384]
- **depth** (*list[int]*) – Number of CVT Attention blocks in each stage, default is [1, 2, 10]
- **num_heads** (*list[int]*) – Number of heads in attention, default is [1, 3, 6]
- **mlp_ratio** (*list[float]*) – Feature dimension expansion ratio in MLP, default is [4.0, 4.0, 4.0]
- **p_dropout** (*list[float]*) – Probability of dropout in MLP, default is [0, 0, 0]
- **attn_dropout** (*list[float]*) – Probability of dropout in attention, default is [0, 0, 0]
- **drop_path_rate** (*list[float]*) – Probability for droppath, default is [0, 0, 0.1]
- **kernel_size** (*list[int]*) – Size of kernel, default is [3, 3, 3]
- **padding_q** (*list[int]*) – Size of padding in q, default is [1, 1, 1]
- **padding_kv** (*list[int]*) – Size of padding in kv, default is [2, 2, 2]
- **stride_kv** (*list[int]*) – Stride in kv, default is [2, 2, 2]
- **stride_q** (*list[int]*) – Stride in q, default is [1, 1, 1]

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

8.1.10 Video Vision Transformer

```
class vformer.models.classification.vivit.ViViTModel2(img_size, in_channels, patch_size,  
                                                    embedding_dim, num_frames, depth,  
                                                    num_heads, head_dim, n_classes,  
                                                    mlp_dim=None, pool='cls', p_dropout=0.0,  
                                                    attn_dropout=0.0, drop_path_rate=0.02)
```

Model 2 implementation of: [ViViT: A Video Vision Transformer](#)

Parameters

- **img_size** (*int*) – Size of single frame/ image in video
- **in_channels** (*int*) – Number of channels
- **patch_size** (*int*) – Patch size
- **embedding_dim** (*int*) – Embedding dimension of a patch

- **num_frames** (*int*) – Number of seconds in each Video
- **depth** (*int*) – Number of encoder layers
- **num_heads** (*int*) – Number of attention heads
- **head_dim** (*int*) – Dimension of head
- **n_classes** (*int*) – Number of classes
- **mlp_dim** (*int*) – Dimension of hidden layer
- **pool** (*str*) – Pooling operation, must be one of {"cls", "mean"}, default is "cls"
- **p_dropout** (*float*) – Dropout probability
- **attn_dropout** (*float*) – Dropout probability
- **drop_path_rate** (*float*) – Stochastic drop path rate

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

```
class vformer.models.classification.vivit.ViViTModel3(img_size, patch_t, patch_h, patch_w,
in_channels, n_classes, num_frames,
embedding_dim, depth, num_heads,
head_dim, p_dropout, mlp_dim=None)
```

Model 3 Implementation from : [ViViT: A Video Vision Transformer](#)

Parameters

- **img_size** (*int or tuple[int]*) – size of a frame
- **patch_t** (*int*) – Temporal length of single tube/patch in tubelet embedding
- **patch_h** (*int*) – Height of single tube/patch in tubelet embedding
- **patch_w** (*int*) – Width of single tube/patch in tubelet embedding
- **in_channels** (*int*) – Number of input channels, default is 3
- **n_classes** (*int*) – Number of classes
- **num_frames** (*int*) – Number of seconds in each Video
- **embedding_dim** (*int*) – Embedding dimension of a patch
- **depth** (*int*) – Number of Encoder layers
- **num_heads** (*int*) – Number of attention heads
- **head_dim** (*int*) – Dimension of attention head
- **p_dropout** (*float*) – Dropout rate/probability, default is 0.0
- **mlp_dim** (*int*) – Hidden dimension, optional

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

8.1.11 Perceiver IO

```
class vformer.models.classification.perceiver_io.PerceiverIO(dim=32, depth=6, latent_dim=512,
                                                            num_latents=512,
                                                            num_cross_heads=1,
                                                            num_latent_heads=8,
                                                            cross_head_dim=64,
                                                            latent_head_dim=64,
                                                            queries_dim=32, logits_dim=None,
                                                            decoder_ff=False)
```

Bases: `Module`

Implementation of [Perceiver IO: A General Architecture for Structured Inputs & Outputs](#)

Code Implementation based on: <https://github.com/lucidrains/perceiver-pytorch>

Parameters

- **dim** (*int*) – Size of sequence to be encoded
- **depth** (*int*) – Depth of latent attention blocks
- **latent_dim** (*int*) – Dimension of latent array
- **num_latents** (*int*) – Number of latent arrays
- **num_cross_heads** (*int*) – Number of heads for cross attention
- **num_latent_heads** (*int*) – Number of heads for latent attention
- **cross_head_dim** (*int*) – Dimension of cross attention head
- **latent_head_dim** (*int*) – Dimension of latent attention head
- **queries_dim** (*int*) – Dimension of queries array
- **logits_dim** (*int*, *optional*) – Dimension of output logits
- **decoder_ff** (*bool*) – Whether to include a feed forward layer for the decoder attention block

forward(*x*, *queries*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

training: `bool`

8.2 Dense Prediction

8.2.1 Vision Transformers for Dense Prediction

class `vformer.models.dense.dpt.AddReadout` (*start_index=1*)

Handles readout operation when *readout* parameter is *add*. Removes *cls_token* or *readout_token* from tensor and adds it to the rest of tensor

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

class `vformer.models.dense.dpt.DPTDepth` (*backbone*, *in_channels=3*, *img_size=(384, 384)*, *readout='project'*, *hooks=(2, 5, 8, 11)*, *channels_last=False*, *use_bn=False*, *enable_attention_hooks=False*, *non_negative=True*, *scale=1.0*, *shift=0.0*, *invert=False*)

Implementation of " Vision Transformers for Dense Prediction " <https://arxiv.org/abs/2103.13413>

Parameters

- **backbone** (*str*) – Name of ViT model to be used as backbone, must be one of `{vitb16,`vitl16`,`vit_tiny`}`
- **in_channels** (*int*) – Number of channels in input image, default is 3
- **img_size** (*tuple[int]*) – Input image size, default is (384,384)
- **readout** (*str*) – Method to handle the *readout_token* or *cls_token* Must be one of `{add, ignore,`project`}`, default is *project*
- **hooks** (*list[int]*) – List representing index of encoder blocks on which hooks will be registered. These hooks extract features from different ViT blocks, eg attention, default is (2,5,8,11).
- **channels_last** (*bool*) – Alters the memory format of storing tensors, default is `False`. For more information visit, this *blog-post* https://pytorch.org/tutorials/intermediate/memory_format_tutorial.html
- **use_bn** (*bool*) – If `True`, BatchNormalisation is used in *FeatureFusionBlock_custom*, default is `False`
- **enable_attention_hooks** (*bool*) – If `True`, *get_attention* hook is registered, default is `false`
- **non_negative** (*bool*) – If `True`, Relu operation will be applied in *DPTDepth.model.head* block, default is `True`
- **invert** (*bool*) – If `True`, forward pass output of *DPTDepth.model.head* will be transformed (inverted) according to *scale* and *shift* parameters, default is `False`

- **scale** (*float*) – Float value that will be multiplied with forward pass output from *DPT-Depth.model.head*, default is 1.0
- **shift** (*float*) – Float value that will be added with forward pass output from *DPT-Depth.model.head* after scaling, default is 0.0

forward(*x*)

Forward pass of DPTDepth

Parameters

x (*torch.Tensor*) – Input image tensor

forward_vit(*x*)

Performs forward pass on backbone ViT model and fetches output from different encoder blocks with the help of hooks

Parameters

x (*torch.Tensor*) – Input image tensor

```
class vformer.models.dense.dpt.FeatureFusionBlock_custom(features, activation, deconv=False,
                                                         bn=False, expand=False,
                                                         align_corners=True)
```

Feature fusion block.

forward(**x*s)

Forward pass

```
class vformer.models.dense.dpt.Interpolate(scale_factor, mode, align_corners=False)
```

Interpolation module

Parameters

- **scale_factor** (*float*) – Scaling factor used in interpolation
- **mode** (*str*) – Interpolation mode
- **align_corners** (*bool*) – Whether to align corners in Interpolation operation

forward(*x*)

Forward pass

```
class vformer.models.dense.dpt.ProjectReadout(in_features, start_index=1)
```

Another class that handles readout operation. Used when *readout* parameter is *project*

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

```
class vformer.models.dense.dpt.ResidualConvUnit_custom(features, activation=<class
                                                         'torch.nn.modules.activation.GELU'>,
                                                         bn=True)
```

Residual convolution module

Parameters

- **features** (*int*) – Number of features
- **activation** (*nn.Module*) – Activation module, default is nn.GELU
- **bn** (*bool*) – Whether to use batch normalisation

forward(*x*)

forward pass

class vformer.models.dense.dpt.Slice(*start_index=1*)

Handles readout operation when *readout* parameter is *ignore*. Removes *cls_token* or *readout_token* by index slicing

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

class vformer.models.dense.dpt.Transpose(*dim0, dim1*)

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

8.2.2 Pyramid Vision Transformer

Detection

class vformer.models.dense.PVT.detection.PVTDetection(*img_size=224, patch_size=[7, 3, 3, 3], in_channels=3, embedding_dims=[64, 128, 256, 512], num_heads=[1, 2, 4, 8], mlp_ratio=[4, 4, 4, 4], qkv_bias=False, qk_scale=None, p_dropout=0.0, attn_dropout=0.0, drop_path_rate=0.0, norm_layer=<class 'torch.nn.modules.normalization.LayerNorm'>, depths=[3, 4, 6, 3], sr_ratios=[8, 4, 2, 1], linear=False, use_dwconv=False, ape=True)*

Implementation of Pyramid Vision Transformer: <https://arxiv.org/abs/2102.12122v1>

Parameters

- **img_size** (*int*) – Image size
- **patch_size** (*list(int)*) – List of patch size

- **in_channels** (*int*) – Input channels in image, default=3
- **n_classes** (*int*) – Number of classes for classification
- **embedding_dims** (*int*) – Patch Embedding dimension
- **num_heads** (*tuple[int]*) – Number of heads in each transformer layer
- **depths** (*tuple[int]*) – Depth in each Transformer layer
- **mlp_ratio** (*float*) – Ratio of mlp heads to embedding dimension
- **qkv_bias** (*bool*, *default= True*) – Adds bias to the qkv if true
- **qk_scale** (*float*, *optional*) – Override default qk scale of head_dim ** -0.5 in Spatial Attention if set
- **p_dropout** (*float*,) – Dropout rate,default is 0.0
- **attn_dropout** (*float*,) – Attention dropout rate, default is 0.0
- **drop_path_rate** (*float*) – Stochastic depth rate, default is 0.1
- **sr_ratios** (*float*) – Spatial reduction ratio
- **linear** (*bool*) – Whether to use linear spatial attention, default is False
- **use_dwconv** (*bool*) – Whether to use Depth-wise convolutions in Overlap-patch embedding, default is False
- **ape** (*bool*) – Whether to use absolute position embedding, default is True

forward(*x*)

Parameters

x (*torch.Tensor*) – Input tensor

Returns

Returns list containing output features from all pyramid stages

Return type

torch.Tensor

```
class vformer.models.dense.PVT.detection.PVTDetectionV2(img_size=224, patch_size=[7, 3, 3, 3],  
                                                    in_channels=3, embedding_dims=[64, 128,  
                                                    256, 512], num_heads=[1, 2, 4, 8],  
                                                    mlp_ratio=[4, 4, 4, 4], qkv_bias=False,  
                                                    qk_scale=0.0, p_dropout=0.0,  
                                                    attn_dropout=0.0, drop_path_rate=0.0,  
                                                    norm_layer=<class  
                                                    'torch.nn.modules.normalization.LayerNorm'>,  
                                                    depths=[3, 4, 6, 3], sr_ratios=[8, 4, 2, 1],  
                                                    ape=False, use_dwconv=True,  
                                                    linear=False)
```

Implementation of Pyramid Vision Transformer: <https://arxiv.org/abs/2102.12122v2>

Parameters

- **img_size** (*int*) – Image size
- **patch_size** (*list(int)*) – List of patch size
- **in_channels** (*int*) – Input channels in image, default=3
- **n_classes** (*int*) – Number of classes for classification

- **embedding_dims** (*int*) – Patch Embedding dimension
- **num_heads** (*tuple[int]*) – Number of heads in each transformer layer
- **depths** (*tuple[int]*) – Depth in each Transformer layer
- **mlp_ratio** (*float*) – Ratio of mlp heads to embedding dimension
- **qkv_bias** (*bool*, *default= True*) – Adds bias to the qkv if true
- **qk_scale** (*float*, *optional*) – Override default qk scale of head_dim ** -0.5 in Spatial Attention if set
- **p_dropout** (*float*,) – Dropout rate,default is 0.0
- **attn_dropout** (*float*,) – Attention dropout rate, default is 0.0
- **drop_path_rate** (*float*) – Stochastic depth rate, default is 0.1
- **sr_ratios** (*float*) – Spatial reduction ratio
- **linear** (*bool*) – Whether to use linear spatial attention
- **use_dwconv** (*bool*) – Whether to use Depth-wise convolutions in Overlap-patch embedding
- **ape** (*bool*) – Whether to use absolute position embedding

Segmentation

```
class vformer.models.dense.PVT.segmentation.PVTSegmentation(img_size=224, patch_size=[7, 3, 3,
3], in_channels=3,
embedding_dims=[64, 128, 256, 512],
num_heads=[1, 2, 4, 8],
mlp_ratio=[4, 4, 4, 4],
qkv_bias=False, qk_scale=None,
p_dropout=0.0, attn_dropout=0.0,
drop_path_rate=0.0,
norm_layer=<class
'torch.nn.modules.normalization.LayerNorm'>,
depths=[3, 4, 6, 3], sr_ratios=[8, 4, 2,
1], linear=False, out_channels=1,
use_dwconv=False, ape=True,
return_pyramid=False)
```

Using Pyramid Vision Transformer as a backbone for a segmentation model with help of U-Net like segmentation head. <https://arxiv.org/abs/2102.12122v1>

Parameters

- **img_size** (*int*) – Image size
- **patch_size** (*list(int)*) – List of patch size
- **in_channels** (*int*) – Input channels in image, default=3
- **embedding_dims** (*int*) – Patch Embedding dimension
- **num_heads** (*tuple[int]*) – Number of heads in each transformer layer
- **depths** (*tuple[int]*) – Depth in each Transformer layer
- **mlp_ratio** (*float*) – Ratio of mlp heads to embedding dimension
- **qkv_bias** (*bool*, *default= True*) – Adds bias to the qkv if true

- **qk_scale** (*float, optional*) – Override default qk scale of $\text{head_dim}^{**} -0.5$ in Spatial Attention if set
- **p_dropout** (*float*) – Dropout rate, default is 0.0
- **attn_dropout** (*float*) – Attention dropout rate, default is 0.0
- **drop_path_rate** (*float*) – Stochastic depth rate, default is 0.1
- **sr_ratios** (*float*) – Spatial reduction ratio
- **linear** (*bool*) – Whether to use linear spatial attention
- **use_dwconv** (*bool*) – Whether to use Depth-wise convolutions in Overlap-patch embedding
- **ape** (*bool*) – Whether to use absolute position embedding
- **return_pyramid** (*bool*) – Whether to use all pyramid feature layers for up-sampling, default is False

forward(*x*)

Parameters

x (*torch.Tensor*) – Input tensor

Returns

Returns output tensor

Return type

torch.Tensor

```
class vformer.models.dense.PVT.segmentation.PVTSegmentationV2(img_size=224, patch_size=[7, 3, 3,  
3], in_channels=3,  
embedding_dims=[64, 128, 256,  
512], num_heads=[1, 2, 4, 8],  
mlp_ratio=[4, 4, 4, 4],  
qkv_bias=False, qk_scale=0.0,  
p_dropout=0.0, attn_dropout=0.0,  
drop_path_rate=0.0,  
norm_layer=<class  
'torch.nn.modules.normalization.LayerNorm'>,  
depths=[3, 4, 6, 3], sr_ratios=[8, 4,  
2, 1], ape=False,  
use_dwconv=True, linear=False,  
return_pyramid=False)
```

Using Pyramid Vision Transformer as a backbone for a segmentation model with help of U-Net like segmentation head.

<https://arxiv.org/abs/2106.13797>

Parameters

- **img_size** (*int*) – Image size
- **patch_size** (*list(int)*) – List of patch size
- **in_channels** (*int*) – Input channels in image, default=3
- **embedding_dims** (*int*) – Patch Embedding dimension
- **num_heads** (*tuple[int]*) – Number of heads in each transformer layer
- **depths** (*tuple[int]*) – Depth in each Transformer layer

- **mlp_ratio** (*float*) – Ratio of mlp heads to embedding dimension
- **qkv_bias** (*bool*, *default= True*) – Adds bias to the qkv if true
- **qk_scale** (*float*, *optional*) – Override default qk scale of head_dim ** -0.5 in Spatial Attention if set
- **p_dropout** (*float*,) – Dropout rate,default is 0.0
- **attn_dropout** (*float*,) – Attention dropout rate, default is 0.0
- **drop_path_rate** (*float*) – Stochastic depth rate, default is 0.1
- **sr_ratios** (*float*) – Spatial reduction ratio
- **linear** (*bool*) – Whether to use linear spatial attention, default is False
- **use_dwconv** (*bool*) – Whether to use Depth-wise convolutions in Overlap-patch embedding, default is True
- **ape** (*bool*) – Whether to use absolute position embedding, default is False
- **return_pyramid** (*bool*) – Whether to use all pyramid feature layers for up-sampling, default is true

9.1 Generic Utilities

`vformer.utils.utils.pair(t)`

Parameters

`t` (*tuple[int] or int*) –

9.2 Window Attention Utilities

`vformer.utils.window_utils.create_mask(window_size, shift_size, H, W)`

Parameters

- `window_size` (*int*) – Window Size
- `shift_size` (*int*) – Shift_size

`vformer.utils.window_utils.cyclicshift(input, shift_size, dims=None)`

Parameters

- `input` (*torch.Tensor*) – input tensor
- `shift_size` (*int or tuple(int)*) – Number of places by which input tensor is shifted
- `dims` (*int or tuple(int), optional*) – Axis along which to roll

`vformer.utils.window_utils.get_relative_position_bias_index(window_size)`

Parameters

`window_size` (*int or tuple[int]*) – Window size

`vformer.utils.window_utils.window_partition(x, window_size)`

Parameters

- `x` (*torch.Tensor*) – input tensor
- `window_size` (*int*) – window size

`vformer.utils.window_utils.window_reverse(windows, window_size, H, W)`

Parameters

- `windows` (*torch.Tensor*) –
- `window_size` (*int*) –

VISUALISATION

10.1 Rollout

10.2 Gradient Rollout

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

V

- vformer.attention.convvt, 10
- vformer.attention.cross, 5
- vformer.attention.gated_positional, 9
- vformer.attention.memory_efficient, 8
- vformer.attention.spatial, 7
- vformer.attention.vanilla, 5
- vformer.attention.window, 7
- vformer.common.base_model, 11
- vformer.common.blocks, 11
- vformer.decoder.mlp, 13
- vformer.decoder.perceiver_io, 13
- vformer.decoder.task_heads.segmentation.head,
14
- vformer.encoder.convit, 24
- vformer.encoder.convvt, 25
- vformer.encoder.cross, 19
- vformer.encoder.embedding.convvt, 17
- vformer.encoder.embedding.linear, 15
- vformer.encoder.embedding.overlappatch, 15
- vformer.encoder.embedding.patch, 16
- vformer.encoder.embedding.pos_embedding, 16
- vformer.encoder.embedding.video_patch_embeddings,
18
- vformer.encoder.nn, 20
- vformer.encoder.perceiver_io, 22
- vformer.encoder.pyramid, 20
- vformer.encoder.swin, 22
- vformer.encoder.vanilla, 24
- vformer.functional.merge, 27
- vformer.functional.norm, 27
- vformer.models.classification.cct, 29
- vformer.models.classification.convit, 38
- vformer.models.classification.convvt, 39
- vformer.models.classification.cross, 30
- vformer.models.classification.cvt, 31
- vformer.models.classification.perceiver_io,
42
- vformer.models.classification.pyramid, 32
- vformer.models.classification.swin, 34
- vformer.models.classification.vanilla, 35
- vformer.models.classification.visformer, 36
- vformer.models.classification.vivit, 40
- vformer.models.dense.dpt, 43
- vformer.models.dense.PVT.detection, 45
- vformer.models.dense.PVT.segmentation, 47
- vformer.utils.utils, 51
- vformer.utils.window_utils, 51
- vformer.viz.vit_grad_rollout, 53
- vformer.viz.vit_rollout, 53

INDEX

A

AddReadout (class in *vformer.models.dense.dpt*), 43

B

BaseClassificationModel (class in *vformer.common.base_model*), 11

C

CCT (class in *vformer.models.classification.cct*), 29

ConvEmbedding (class in *vformer.encoder.embedding.convvt*), 17

ConViT (class in *vformer.models.classification.convit*), 38

ConViTEncoder (class in *vformer.encoder.convit*), 24

ConvVT (class in *vformer.models.classification.convvt*), 39

ConvVTAttention (class in *vformer.attention.convvt*), 10

ConvVTBlock (class in *vformer.encoder.convvt*), 25

ConvVTStage (class in *vformer.encoder.convvt*), 25

create_mask() (in module *vformer.utils.window_utils*), 51

CrossAttention (class in *vformer.attention.cross*), 5

CrossAttentionWithClsToken (class in *vformer.attention.cross*), 6

CrossEncoder (class in *vformer.encoder.cross*), 19

CrossViT (class in *vformer.models.classification.cross*), 30

CVT (class in *vformer.models.classification.cvt*), 31

cyclicshift() (in module *vformer.utils.window_utils*), 51

D

DoubleConv (class in *vformer.decoder.task_heads.segmentation.head*), 14

DPTDepth (class in *vformer.models.dense.dpt*), 43

DWConv (class in *vformer.common.blocks*), 11

dynamic_slice() (*vformer.attention.memory_efficient.MemoryEfficientAttention* static method), 8

F

FeatureFusionBlock_custom (class in *vformer.models.dense.dpt*), 44

FeedForward (class in *vformer.encoder.nn*), 20

forward() (*vformer.attention.convvt.ConvVTAttention* method), 10

forward() (*vformer.attention.cross.CrossAttention* method), 6

forward() (*vformer.attention.cross.CrossAttentionWithClsToken* method), 6

forward() (*vformer.attention.gated_positional.GatedPositionalSelfAttention* method), 9

forward() (*vformer.attention.memory_efficient.MemoryEfficientAttention* method), 8

forward() (*vformer.attention.spatial.SpatialAttention* method), 7

forward() (*vformer.attention.vanilla.VanillaSelfAttention* method), 5

forward() (*vformer.attention.window.WindowAttention* method), 8

forward() (*vformer.common.blocks.DWConv* method), 11

forward() (*vformer.decoder.mlp.MLPDecoder* method), 13

forward() (*vformer.decoder.perceiver_io.PerceiverIODecoder* method), 14

forward() (*vformer.decoder.task_heads.segmentation.head.DoubleConv* method), 14

forward() (*vformer.decoder.task_heads.segmentation.head.SegmentationFusion* method), 14

forward() (*vformer.encoder.convvt.ConvVTBlock* method), 25

forward() (*vformer.encoder.convvt.ConvVTStage* method), 26

forward() (*vformer.encoder.cross.CrossEncoder* method), 19

forward() (*vformer.encoder.embedding.convvt.ConvEmbedding* method), 17

forward() (*vformer.encoder.embedding.linear.LinearEmbedding* method), 15

forward() (*vformer.encoder.embedding.overlappatch.OverlapPatchEmbedding* method), 16

forward() (*vformer.encoder.embedding.patch.PatchEmbedding* method), 16

forward() (*vformer.encoder.embedding.pos_embedding.PosEmbedding* method), 16

- `method`), 17
 - `forward()` (`vformer.encoder.embedding.pos_embedding.PVTPositionalEmbedding` `method`), 17
 - `forward()` (`vformer.encoder.embedding.video_patch_embeddings.VideoPatchEmbeddings` `method`), 18
 - `forward()` (`vformer.encoder.embedding.video_patch_embeddings.VideoPatchEmbeddings` `method`), 18
 - `forward()` (`vformer.encoder.nn.FeedForward` `method`), 20
 - `forward()` (`vformer.encoder.perceiver_io.PerceiverIOEncoder` `method`), 22
 - `forward()` (`vformer.encoder.pyramid.PVTEncoder` `method`), 21
 - `forward()` (`vformer.encoder.pyramid.PVTFeedForward` `method`), 21
 - `forward()` (`vformer.encoder.swin.SwinEncoder` `method`), 23
 - `forward()` (`vformer.encoder.swin.SwinEncoderBlock` `method`), 23
 - `forward()` (`vformer.encoder.vanilla.VanillaEncoder` `method`), 24
 - `forward()` (`vformer.functional.merge.PatchMerging` `method`), 27
 - `forward()` (`vformer.functional.norm.PreNorm` `method`), 27
 - `forward()` (`vformer.models.classification.cct.CCT` `method`), 30
 - `forward()` (`vformer.models.classification.convit.ConvIT` `method`), 39
 - `forward()` (`vformer.models.classification.convvt.ConvVT` `method`), 40
 - `forward()` (`vformer.models.classification.cross.CrossViT` `method`), 31
 - `forward()` (`vformer.models.classification.cvt.CVT` `method`), 32
 - `forward()` (`vformer.models.classification.perceiver_io.PerceiverIO` `method`), 42
 - `forward()` (`vformer.models.classification.pyramid.PVTClassification` `method`), 33
 - `forward()` (`vformer.models.classification.swin.SwinTransformer` `method`), 35
 - `forward()` (`vformer.models.classification.vanilla.VanillaViT` `method`), 36
 - `forward()` (`vformer.models.classification.visformer.Visformer` `method`), 37
 - `forward()` (`vformer.models.classification.visformer.VisformerAttentionBlock` `method`), 37
 - `forward()` (`vformer.models.classification.visformer.VisformerConvBlock` `method`), 37
 - `forward()` (`vformer.models.classification.vivit.ViViTModel2` `method`), 41
 - `forward()` (`vformer.models.classification.vivit.ViViTModel3` `method`), 41
 - `forward()` (`vformer.models.dense.dpt.AddReadout` `method`), 43
 - `forward()` (`vformer.models.dense.dpt.DPTDepth` `method`), 44
 - `forward()` (`vformer.models.dense.dpt.FeatureFusionBlock_custom` `method`), 44
 - `forward()` (`vformer.models.dense.dpt.Interpolate` `method`), 44
 - `forward()` (`vformer.models.dense.dpt.ProjectReadout` `method`), 44
 - `forward()` (`vformer.models.dense.dpt.ResidualConvUnit_custom` `method`), 45
 - `forward()` (`vformer.models.dense.dpt.Slice` `method`), 45
 - `forward()` (`vformer.models.dense.dpt.Transpose` `method`), 45
 - `forward()` (`vformer.models.dense.PVT.detection.PVTDetection` `method`), 46
 - `forward()` (`vformer.models.dense.PVT.segmentation.PVTSegmentation` `method`), 48
 - `forward_conv()` (`vformer.attention.convvt.ConvVTAttention` `method`), 10
 - `forward_vit()` (`vformer.models.dense.dpt.DPTDepth` `method`), 44
- ## G
- `GatedPositionalSelfAttention` (class in `vformer.attention.gated_positional`), 9
 - `get_relative_position_bias_index()` (in module `vformer.utils.window_utils`), 51
- ## I
- `Interpolate` (class in `vformer.models.dense.dpt`), 44
- ## L
- `LinearEmbedding` (class in `vformer.encoder.embedding.linear`), 15
 - `LinearVideoEmbedding` (class in `vformer.encoder.embedding.video_patch_embeddings`), 18
- ## M
- `map_pt()` (`vformer.attention.memory_efficient.MemoryEfficientAttention` `static method`), 9
 - `MemoryEfficientAttention` (class in `vformer.attention.memory_efficient`), 8
 - `MLPDecoder` (class in `vformer.decoder.mlp`), 13
- ## vformer
- `vformer.attention.convvt`, 10
 - `vformer.attention.cross`, 5
 - `vformer.attention.gated_positional`, 9
 - `vformer.attention.memory_efficient`, 8
 - `vformer.attention.spatial`, 7
 - `vformer.attention.vanilla`, 5
 - `vformer.attention.window`, 7

- [vformer.common.base_model](#), 11
[vformer.common.blocks](#), 11
[vformer.decoder.mlp](#), 13
[vformer.decoder.perceiver_io](#), 13
[vformer.decoder.task_heads.segmentation.head](#),
 14
[vformer.encoder.convit](#), 24
[vformer.encoder.convvt](#), 25
[vformer.encoder.cross](#), 19
[vformer.encoder.embedding.convvt](#), 17
[vformer.encoder.embedding.linear](#), 15
[vformer.encoder.embedding.overlappatch](#),
 15
[vformer.encoder.embedding.patch](#), 16
[vformer.encoder.embedding.pos_embedding](#),
 16
[vformer.encoder.embedding.video_patch_embedding](#),
 18
[vformer.encoder.nn](#), 20
[vformer.encoder.perceiver_io](#), 22
[vformer.encoder.pyramid](#), 20
[vformer.encoder.swin](#), 22
[vformer.encoder.vanilla](#), 24
[vformer.functional.merge](#), 27
[vformer.functional.norm](#), 27
[vformer.models.classification.cct](#), 29
[vformer.models.classification.convit](#), 38
[vformer.models.classification.convvt](#), 39
[vformer.models.classification.cross](#), 30
[vformer.models.classification.cvt](#), 31
[vformer.models.classification.perceiver_io](#),
 42
[vformer.models.classification.pyramid](#), 32
[vformer.models.classification.swin](#), 34
[vformer.models.classification.vanilla](#), 35
[vformer.models.classification.visformer](#),
 36
[vformer.models.classification.vivit](#), 40
[vformer.models.dense.dpt](#), 43
[vformer.models.dense.PVT.detection](#), 45
[vformer.models.dense.PVT.segmentation](#), 47
[vformer.utils.utils](#), 51
[vformer.utils.window_utils](#), 51
[vformer.viz.vit_grad_rollout](#), 53
[vformer.viz.vit_rollout](#), 53
- O**
- [OverlapPatchEmbed](#) (class in *vformer.encoder.embedding.overlappatch*),
 15
- P**
- [pair\(\)](#) (in module *vformer.utils.utils*), 51
- [PatchEmbedding](#) (class in *vformer.encoder.embedding.patch*), 16
[PatchMerging](#) (class in *vformer.functional.merge*), 27
[PerceiverIO](#) (class in *vformer.models.classification.perceiver_io*), 42
[PerceiverIODecoder](#) (class in *vformer.decoder.perceiver_io*), 13
[PerceiverIOEncoder](#) (class in *vformer.encoder.perceiver_io*), 22
[PosEmbedding](#) (class in *vformer.encoder.embedding.pos_embedding*),
 17
[PreNorm](#) (class in *vformer.functional.norm*), 27
[ProjectReadout](#) (class in *vformer.models.dense.dpt*), 44
[PVTClassification](#) (class in *vformer.models.classification.pyramid*), 32
[PVTClassificationV2](#) (class in *vformer.models.classification.pyramid*), 33
[PVTDetection](#) (class in *vformer.models.dense.PVT.detection*), 45
[PVTDetectionV2](#) (class in *vformer.models.dense.PVT.detection*), 46
[PVTDecoder](#) (class in *vformer.encoder.pyramid*), 20
[PVTFeedForward](#) (class in *vformer.encoder.pyramid*), 21
[PVTPosEmbedding](#) (class in *vformer.encoder.embedding.pos_embedding*),
 16
[PVTSegmentation](#) (class in *vformer.models.dense.PVT.segmentation*),
 47
[PVTSegmentationV2](#) (class in *vformer.models.dense.PVT.segmentation*),
 48
- Q**
- [query_chunk_attention\(\)](#) (*vformer.attention.memory_efficient.MemoryEfficientAttention*
method), 9
- R**
- [rel_embedding\(\)](#) (*vformer.attention.gated_positional.GatedPositionalSelf*
method), 9
[ResidualConvUnit_custom](#) (class in *vformer.models.dense.dpt*), 44
[resize_pos_embed\(\)](#) (*vformer.encoder.embedding.pos_embedding.PVTPos*
method), 17
- S**
- [scan\(\)](#) (*vformer.attention.memory_efficient.MemoryEfficientAttention*
static method), 9
[SegmentationHead](#) (class in *vformer.decoder.task_heads.segmentation.head*),
 14
[Slice](#) (class in *vformer.models.dense.dpt*), 45

| | |
|--|---|
| SpatialAttention (class in vformer.attention.spatial), 7 | module, 11 |
| summarize_chunk() (vformer.attention.memory_efficient.MemoryEfficientAttention static method), 9 | vformer.common.blocks module, 11 |
| SwinEncoder (class in vformer.encoder.swin), 22 | vformer.decoder.mlp module, 13 |
| SwinEncoderBlock (class in vformer.encoder.swin), 23 | vformer.decoder.perceiver_io module, 13 |
| SwinTransformer (class in vformer.models.classification.swin), 34 | vformer.decoder.task_heads.segmentation.head module, 14 |
| T | vformer.encoder.convit module, 24 |
| training (vformer.attention.convvt.ConvVTAttention attribute), 10 | vformer.encoder.convvt module, 25 |
| training (vformer.attention.cross.CrossAttention attribute), 6 | vformer.encoder.cross module, 19 |
| training (vformer.attention.cross.CrossAttentionWithClsToken attribute), 6 | vformer.encoder.embedding.convvt module, 17 |
| training (vformer.attention.gated_positional.GatedPositionalSelfAttention attribute), 9 | vformer.encoder.embedding.linear module, 15 |
| training (vformer.attention.memory_efficient.MemoryEfficientAttention attribute), 9 | vformer.encoder.embedding.overlappatch module, 15 |
| training (vformer.attention.spatial.SpatialAttention attribute), 7 | vformer.encoder.embedding.patch module, 16 |
| training (vformer.attention.vanilla.VanillaSelfAttention attribute), 5 | vformer.encoder.embedding.pos_embedding module, 16 |
| training (vformer.attention.window.WindowAttention attribute), 8 | vformer.encoder.embedding.video_patch_embeddings module, 18 |
| training (vformer.models.classification.perceiver_io.PerceiverIo attribute), 42 | vformer.encoder.nn module, 20 |
| Transpose (class in vformer.models.dense.dpt), 45 | vformer.encoder.perceiver_io module, 22 |
| TubeletEmbedding (class in vformer.encoder.embedding.video_patch_embeddings), 18 | vformer.encoder.pyramid module, 20 |
| V | vformer.encoder.swin module, 22 |
| VanillaEncoder (class in vformer.encoder.vanilla), 24 | vformer.encoder.vanilla module, 24 |
| VanillaSelfAttention (class in vformer.attention.vanilla), 5 | vformer.functional.merge module, 27 |
| VanillaViT (class in vformer.models.classification.vanilla), 35 | vformer.functional.norm module, 27 |
| vformer.attention.convvt module, 10 | vformer.models.classification.cct module, 29 |
| vformer.attention.cross module, 5 | vformer.models.classification.convit module, 38 |
| vformer.attention.gated_positional module, 9 | vformer.models.classification.convvt module, 39 |
| vformer.attention.memory_efficient module, 8 | vformer.models.classification.cross module, 30 |
| vformer.attention.spatial module, 7 | vformer.models.classification.cvt module, 31 |
| vformer.attention.vanilla module, 5 | vformer.models.classification.perceiver_io module, 42 |
| vformer.attention.window module, 7 | vformer.models.classification.pyramid |
| vformer.common.base_model | |

module, 32
 vformer.models.classification.swin
 module, 34
 vformer.models.classification.vanilla
 module, 35
 vformer.models.classification.visformer
 module, 36
 vformer.models.classification.vivit
 module, 40
 vformer.models.dense.dpt
 module, 43
 vformer.models.dense.PVT.detection
 module, 45
 vformer.models.dense.PVT.segmentation
 module, 47
 vformer.utils.utils
 module, 51
 vformer.utils.window_utils
 module, 51
 vformer.viz.vit_grad_rollout
 module, 53
 vformer.viz.vit_rollout
 module, 53
 Visformer (class in vformer.models.classification.visformer),
 36
 Visformer_S() (in module
 vformer.models.classification.visformer),
 38
 Visformer_Ti() (in module
 vformer.models.classification.visformer),
 38
 VisformerAttentionBlock (class in
 vformer.models.classification.visformer),
 37
 VisformerConvBlock (class in
 vformer.models.classification.visformer),
 37
 VisformerV2_S() (in module
 vformer.models.classification.visformer),
 38
 VisformerV2_Ti() (in module
 vformer.models.classification.visformer),
 38
 ViViTModel2 (class in
 vformer.models.classification.vivit), 40
 ViViTModel3 (class in
 vformer.models.classification.vivit), 41

W

window_partition() (in module
 vformer.utils.window_utils), 51
 window_reverse() (in module
 vformer.utils.window_utils), 51
 WindowAttention (class in vformer.attention.window), 7