ADN#

Constructs a sequential module of optional activation (A), dropout (D), and normalization (N) layers with an arbitrary order: ordering (str) – a string representing the ordering of activation, dropout, and normalization. Defaults to “NDA”. in_channels (Optional[int]) – C from an expected input of size (N, C, H[, W, D]). act (Union[Tuple, str, None]) – activation type and arguments. Defaults to PReLU. norm (Union[Tuple, str, None]) – feature normalization type and arguments. Defaults to instance norm. norm_dim (Optional[int]) – determine the spatial dimensions of the normalization layer. defaults to dropout_dim if unspecified. dropout (Union[Tuple, str, float, None]) – dropout ratio. Defaults to no dropout. dropout_dim (Optional[int]) – determine the spatial dimensions of dropout. defaults to norm_dim if unspecified. When dropout_dim = 1, randomly zeroes some of the elements for each channel. When dropout_dim = 2, Randomly zeroes out entire channels (a channel is a 2D feature map). When dropout_dim = 3, Randomly zeroes out entire channels (a channel is a 3D feature map). determine the spatial dimensions of dropout. defaults to norm_dim if unspecified. When dropout_dim = 1, randomly zeroes some of the elements for each channel. When dropout_dim = 2, Randomly zeroes out entire channels (a channel is a 2D feature map). When dropout_dim = 3, Randomly zeroes out entire channels (a channel is a 3D feature map). Examples: See also monai.networks.layers.Dropout monai.networks.layers.Act monai.networks.layers.Norm monai.networks.layers.split_args

Convolution#

Constructs a convolution with normalization, optional dropout, and optional activation layers: if conv_only set to True: For example: output: spatial_dims (int) – number of spatial dimensions. in_channels (int) – number of input channels. out_channels (int) – number of output channels. strides (Union[Sequence[int], int]) – convolution stride. Defaults to 1. kernel_size (Union[Sequence[int], int]) – convolution kernel size. Defaults to 3. adn_ordering (str) – a string representing the ordering of activation, normalization, and dropout. Defaults to “NDA”. act (Union[Tuple, str, None]) – activation type and arguments. Defaults to PReLU. norm (Union[Tuple, str, None]) – feature normalization type and arguments. Defaults to instance norm. dropout (Union[Tuple, str, float, None]) – dropout ratio. Defaults to no dropout. dropout_dim (Optional[int]) – determine the spatial dimensions of dropout. Defaults to 1. When dropout_dim = 1, randomly zeroes some of the elements for each channel. When dropout_dim = 2, Randomly zeroes out entire channels (a channel is a 2D feature map). When dropout_dim = 3, Randomly zeroes out entire channels (a channel is a 3D feature map). The value of dropout_dim should be no larger than the value of spatial_dims. determine the spatial dimensions of dropout. Defaults to 1. When dropout_dim = 1, randomly zeroes some of the elements for each channel. When dropout_dim = 2, Randomly zeroes out entire channels (a channel is a 2D feature map). When dropout_dim = 3, Randomly zeroes out entire channels (a channel is a 3D feature map). The value of dropout_dim should be no larger than the value of spatial_dims. dilation (Union[Sequence[int], int]) – dilation rate. Defaults to 1. groups (int) – controls the connections between inputs and outputs. Defaults to 1. bias (bool) – whether to have a bias term. Defaults to True. conv_only (bool) – whether to use the convolutional layer only. Defaults to False. is_transposed (bool) – if True uses ConvTrans instead of Conv. Defaults to False. padding (Union[Sequence[int], int, None]) – controls the amount of implicit zero-paddings on both sides for padding number of points for each dimension. Defaults to None. output_padding

(Union[Sequence[int], int, None]) – controls the additional size added to one side of the output shape. Defaults to None. See also monai.networks.layers.Conv monai.networks.blocks.ADN

CRF#

Conditional Random Field: Combines message passing with a class compatibility convolution into an iterative process designed to successively minimise the energy of the class labeling. In this implementation, the message passing step is a weighted combination of a gaussian filter and a bilateral filter. The bilateral term is included to respect existing structure within the reference tensor. https://arxiv.org/abs/1502.03240 iterations (int) – the number of iterations. bilateral_weight (float) – the weighting of the bilateral term in the message passing step. gaussian_weight (float) – the weighting of the gaussian term in the message passing step. bilateral_spatial_sigma (float) – standard deviation in spatial coordinates for the bilateral term. bilateral_color_sigma (float) – standard deviation in color space for the bilateral term. gaussian_spatial_sigma (float) – standard deviation in spatial coordinates for the gaussian term. update_factor (float) – determines the magnitude of each update. compatibility_matrix (Optional[Tensor]) – a matrix describing class compatibility, should be NxN where N is the number of classes. input_tensor (Tensor) – tensor containing initial class logits. reference_tensor (Tensor) – the reference tensor used to guide the message passing. output tensor. output (torch.Tensor)

ResidualUnit#

Residual module with multiple convolutions and a residual connection. For example: output: spatial_dims (int) – number of spatial dimensions. in_channels (int) – number of input channels. out_channels (int) – number of output channels. strides (Union[Sequence[int], int]) – convolution stride. Defaults to 1. kernel_size (Union[Sequence[int], int]) – convolution kernel size. Defaults to 3. subunits (int) – number of convolutions. Defaults to 2. adn_ordering (str) – a string representing the ordering of activation, normalization, and dropout. Defaults to “NDA”. act (Union[Tuple, str, None]) – activation type and arguments. Defaults to PReLU. norm (Union[Tuple, str, None]) – feature normalization type and arguments. Defaults to instance norm. dropout (Union[Tuple, str, float, None]) – dropout ratio. Defaults to no dropout. dropout_dim (Optional[int]) – determine the dimensions of dropout. Defaults to 1. When dropout_dim = 1, randomly zeroes some of the elements for each channel. When dropout_dim = 2, Randomly zero out entire channels (a channel is a 2D feature map). When dropout_dim = 3, Randomly zero out entire channels (a channel is a 3D feature map). The value of dropout_dim should be no larger than the value of dimensions. determine the dimensions of dropout. Defaults to 1. When dropout_dim = 1, randomly zeroes some of the elements for each channel. When dropout_dim = 2, Randomly zero out entire channels (a channel is a 2D feature map). When dropout_dim = 3, Randomly zero out entire channels (a channel is a 3D feature map). The value of dropout_dim should be no larger than the value of dimensions. dilation (Union[Sequence[int], int]) – dilation rate. Defaults to 1. bias (bool) – whether to have a bias term. Defaults to True. last_conv_only (bool) – for the last subunit, whether to use the convolutional layer only. Defaults to False. padding (Union[Sequence[int], int, None]) – controls the amount of implicit zero-paddings on both sides for padding number of points for each dimension. Defaults to None. See also monai.networks.blocks.Convolution 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. Tensor

Swish#

Applies the element-wise function: Citation: Searching for Activation Functions, Ramachandran et al., 2017, https://arxiv.org/abs/1710.05941. Input: \((N, *)\) where * means, any number of additional dimensions Output: \((N, *)\), same shape as the input Examples: 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. Tensor

MemoryEfficientSwish#

Applies the element-wise function: Memory efficient implementation for training following recommendation from: lukemelas/EfficientNet-PyTorch#18 Results in ~ 30% memory saving during training as compared to Swish() Citation: Searching for Activation Functions, Ramachandran et al., 2017, https://arxiv.org/abs/1710.05941. From Pytorch 1.7.0+, the optimized version of Swish named SiLU is implemented, this class will utilize torch.nn.functional.silu to do the calculation if meets the version. Input: \((N, *)\) where * means, any number of additional dimensions Output: \((N, *)\), same shape as the input Examples: 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.

FPN#

Base class for the extra block in the FPN. Same code as pytorch/vision Compute extended set of results of the FPN and their names. results (List[Tensor]) – the result of the FPN x (List[Tensor]) – the original feature maps names (List[str]) – the names for each one of the original feature maps the extended set of results of the FPN the extended set of names for the results the extended set of results of the FPN the extended set of names for the results Module that adds a FPN from on top of a set of feature maps. This is based on “Feature Pyramid Network for Object Detection”. The feature maps are currently supposed to be in increasing depth order. The input to the model is expected to be an OrderedDict[Tensor], containing the feature maps on top of which the FPN will be added. spatial_dims (int) – 2D or 3D images in_channels_list (List[int]) – number of channels for each feature map that is passed to the module out_channels (int) – number of channels of the FPN representation extra_blocks (Optional[ExtraFPNBlock]) – if provided, extra operations will be performed. It is expected to take the fpn features, the original features and the names of the original features as input, and returns a new list of feature maps and their corresponding names Examples: Computes the FPN for a set of feature maps. x (Dict[str, Tensor]) – feature maps for each feature level. Dict[str, Tensor] feature maps after FPN layers.

They are ordered from highest resolution first. This is equivalent to self.inner_blocks[idx](x), but torchscript doesn't support this yet Tensor This is equivalent to self.layer_blocks[idx](x), but torchscript doesn't support this yet Tensor Applies a max_pool2d or max_pool3d on top of the last feature map. Serves as an extra_blocks in FeaturePyramidNetwork . Compute extended set of results of the FPN and their names. results (List[Tensor]) – the result of the FPN x (List[Tensor]) – the original feature maps names (List[str]) – the names for each one of the original feature maps Tuple[List[Tensor], List[str]] the extended set of results of the FPN the extended set of names for the results the extended set of results of the FPN the extended set of names for the results This module is used in RetinaNet to generate extra layers, P6 and P7. Serves as an extra_blocks in FeaturePyramidNetwork . Compute extended set of results of the FPN and their names. results (List[Tensor]) – the result of the FPN x (List[Tensor]) – the original feature maps names (List[str]) – the names for each one of the original feature maps Tuple[List[Tensor], List[str]] the extended set of results of the FPN the extended set of names for the results the extended set of results of the FPN the extended set of names for the results Adds an FPN on top of a model. Internally, it uses torchvision.models._utils.IntermediateLayerGetter to extract a submodel that returns the feature maps specified in return_layers. The same limitations of IntermediateLayerGetter apply here. Same code as pytorch/vision Except that this class uses spatial_dims backbone (Module) – backbone network return_layers (Dict[str, str]) – a dict containing the names of the modules for which the activations will be returned as the key of the dict, and the value of the dict is the name of the returned activation (which the user can specify). in_channels_list (List[int]) – number of channels for each feature map that is returned, in the order they are present in the OrderedDict out_channels (int) – number of channels in the FPN. spatial_dims (Optional[int]) – 2D or 3D images Computes the resulted feature maps of the network. x (Tensor) – input images Dict[str, Tensor] feature maps after FPN layers. They are ordered from highest resolution first.

Mish#

Applies the element-wise function: Citation: Mish: A Self Regularized Non-Monotonic Activation Function, Diganta Misra, 2019, https://arxiv.org/abs/1908.08681. From Pytorch 1.9.0+, the optimized version of Mish is implemented, this class will utilize torch.nn.functional.mish to do the calculation if meets the version. Input: \((N, *)\) where * means, any number of additional dimensions Output: \((N, *)\), same shape as the input Examples: 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.

GCN Module#

The Global Convolutional Network module using large 1D Kx1 and 1xK kernels to represent 2D kernels. inplanes (int) – number of input channels. planes (int) – number of output channels. ks (int) – kernel size for one dimension. Defaults to 7. x (Tensor) – in shape (batch, inplanes, spatial_1, spatial_2). Tensor

Refinement Module#

Simple residual block to refine the details of the activation maps. planes (int) – number of input channels. x (Tensor) – in shape (batch, planes, spatial_1, spatial_2). Tensor

FCN Module#

2D FCN network with 3 input channels. The small decoder is built with the GCN and Refine modules. The code is adapted from lsqshr's official 2D code. out_channels (int) – number of output channels. Defaults to 1. upsample_mode (str) – ["transpose", "bilinear"] The mode of upsampling manipulations. Using the second mode cannot guarantee the model's reproducibility. Defaults to bilinear. transpose, uses transposed convolution layers. bilinear, uses bilinear interpolation. ["transpose", "bilinear"] The mode of upsampling manipulations. Using the second mode cannot guarantee the model's reproducibility. Defaults to bilinear. transpose, uses transposed convolution layers. bilinear, uses bilinear interpolation. pretrained (bool) – If True, returns a model pre-trained on ImageNet progress (bool) – If True, displays a progress bar of the download to stderr. x (Tensor) – in shape (batch, 3, spatial_1, spatial_2).

Multi-Channel FCN Module#

The multi-channel version of the 2D FCN module. Adds a projection layer to take arbitrary number of inputs. in_channels (int) – number of input channels. Defaults to 3. out_channels (int) – number of output channels. Defaults to 1. upsample_mode (str) – ["transpose", "bilinear"] The mode of upsampling manipulations. Using the second mode cannot guarantee the model's reproducibility. Defaults to bilinear. transpose, uses transposed convolution layers. bilinear, uses bilinear interpolate. ["transpose", "bilinear"] The mode of upsampling manipulations. Using the second mode cannot guarantee the model's reproducibility. Defaults to bilinear. transpose, uses transposed convolution layers. bilinear, uses bilinear interpolate. pretrained (bool) – If True, returns a model pre-trained on ImageNet progress (bool) – If True, displays a progress bar of the download to stderr. x (Tensor) – in shape (batch, in_channels, spatial_1, spatial_2).

Dynamic-Unet Block#

A skip-connection based module that can be used for DynUNet, based on: Automated Design of Deep Learning Methods for Biomedical Image Segmentation. nnU-Net: Self-adapting Framework for U-Net-Based Medical Image Segmentation. spatial_dims (int) – number of spatial dimensions. in_channels (int) – number of input channels. out_channels (int) – number of output channels. kernel_size (Union[Sequence[int], int]) – convolution kernel size. stride (Union[Sequence[int], int]) – convolution stride. norm_name (Union[Tuple, str]) – feature normalization type and arguments. act_name (Union[Tuple, str]) – activation layer type and arguments. dropout (Union[Tuple, str, float, None]) – dropout probability. 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. A CNN module that can be used for DynUNet, based on: Automated Design of Deep Learning Methods for Biomedical Image Segmentation. nnU-Net: Self-adapting Framework for U-Net-Based Medical Image Segmentation. spatial_dims (int) – number of spatial dimensions.

in_channels (int) – number of input channels. out_channels (int) – number of output channels. kernel_size (Union[Sequence[int], int]) – convolution kernel size. stride (Union[Sequence[int], int]) – convolution stride. norm_name (Union[Tuple, str]) – feature normalization type and arguments. act_name (Union[Tuple, str]) – activation layer type and arguments. dropout (Union[Tuple, str, float, None]) – dropout probability. 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. An upsampling module that can be used for DynUNet, based on: Automated Design of Deep Learning Methods for Biomedical Image Segmentation. nnU-Net: Self-adapting Framework for U-Net-Based Medical Image Segmentation. spatial_dims (int) – number of spatial dimensions. in_channels (int) – number of input channels. out_channels (int) – number of output channels. kernel_size (Union[Sequence[int], int]) – convolution kernel size. stride (Union[Sequence[int], int]) – convolution stride. upsample_kernel_size (Union[Sequence[int], int]) – convolution kernel size for transposed convolution layers. norm_name (Union[Tuple, str]) – feature normalization type and arguments. act_name (Union[Tuple, str]) – activation layer type and arguments. dropout (Union[Tuple, str, float, None]) – dropout probability. trans_bias (bool) – transposed convolution bias. 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. 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.

DenseBlock#

A DenseBlock is a sequence of layers where each layer’s outputs are concatenated with their inputs. This has the effect of accumulating outputs from previous layers as inputs to later ones and as the final output of the block. layers (Sequence[Module]) – sequence of nn.Module objects to define the individual layers of the dense block 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.

SegResnet Block#

ResBlock employs skip connection and two convolution blocks and is used in SegResNet based on 3D MRI brain tumor segmentation using autoencoder regularization. spatial_dims (int) – number of spatial dimensions, could be 1, 2 or 3. in_channels (int) – number of input channels. norm (Union[Tuple, str]) – feature normalization type and arguments. kernel_size (int) – convolution kernel size, the value should be an odd number. Defaults to 3. act (Union[Tuple, str]) – activation type and arguments. Defaults to RELU. 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.

SABlock Block#

A self-attention block, based on: “Dosovitskiy et al., An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale ” hidden_size (int) – dimension of hidden layer. num_heads (int) – number of attention heads. dropout_rate (float) – faction of the input units to drop. qkv_bias (bool) – bias term for the qkv linear layer. 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.

Squeeze-and-Excitation#

Re-implementation of the Squeeze-and-Excitation block based on: “Hu et al., Squeeze-and-Excitation Networks, https://arxiv.org/abs/1709.01507”. spatial_dims (int) – number of spatial dimensions, could be 1, 2, or 3. in_channels (int) – number of input channels. r (int) – the reduction ratio r in the paper. Defaults to 2. acti_type_1 (Union[Tuple[str, Dict], str]) – activation type of the hidden squeeze layer. Defaults to ("relu", {"inplace": True}). acti_type_2 (Union[Tuple[str, Dict], str]) – activation type of the output squeeze layer. Defaults to “sigmoid”. ValueError – When r is nonpositive or larger than in_channels. See also monai.networks.layers.Act x (Tensor) – in shape (batch, in_channels, spatial_1[, spatial_2, …]). Tensor

Transformer Block#

A transformer block, based on: “Dosovitskiy et al., An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale ” hidden_size (int) – dimension of hidden layer. mlp_dim (int) – dimension of feedforward layer. num_heads (int) – number of attention heads. dropout_rate (float) – faction of the input units to drop. qkv_bias (bool) – apply bias term for the qkv linear layer 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.

UNETR Block#

A CNN module that can be used for UNETR, based on: “Hatamizadeh et al., UNETR: Transformers for 3D Medical Image Segmentation ” spatial_dims (int) – number of spatial dimensions. in_channels (int) – number of input channels. out_channels (int) – number of output channels. kernel_size (Union[Sequence[int], int]) – convolution kernel size. stride (Union[Sequence[int], int]) – convolution stride. norm_name (Union[Tuple, str]) – feature normalization type and arguments. res_block (bool) – bool argument to determine if residual block is used. 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. An upsampling module that can be used for UNETR: “Hatamizadeh et al., UNETR: Transformers for 3D Medical Image Segmentation ” spatial_dims (int) – number of spatial dimensions. in_channels (int) – number of input channels. out_channels (int) – number of output channels. kernel_size (Union[Sequence[int], int]) – convolution kernel size. upsample_kernel_size (Union[Sequence[int], int]) – convolution kernel size for transposed convolution layers. norm_name (Union[Tuple, str]) – feature normalization type and arguments. res_block (bool) – bool argument to determine if residual block is used. 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. A projection upsampling module that can be used for UNETR: “Hatamizadeh et al., UNETR: Transformers for 3D Medical Image Segmentation ” spatial_dims (int) – number of spatial dimensions. in_channels (int) – number of input channels. out_channels (int) – number of output channels. num_layer (int) – number of upsampling blocks. kernel_size (Union[Sequence[int], int]) – convolution kernel size. stride (Union[Sequence[int], int]) – convolution stride. upsample_kernel_size (Union[Sequence[int], int]) – convolution kernel size for transposed convolution layers. norm_name (Union[Tuple, str]) – feature normalization type and arguments. conv_block (bool) – bool argument to determine if convolutional block is used. res_block (bool) – bool argument to determine if residual block is used. 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.

Residual Squeeze-and-Excitation#

A “squeeze-and-excitation”-like layer with a residual connection: spatial_dims (int) – number of spatial dimensions, could be 1, 2, or 3. in_channels (int) – number of input channels. r (int) – the reduction ratio r in the paper. Defaults to 2. acti_type_1 (Union[Tuple[str, Dict], str]) – defaults to “leakyrelu”. acti_type_2 (Union[Tuple[str, Dict], str]) – defaults to “relu”. See also monai.networks.blocks.ChannelSELayer

Squeeze-and-Excitation Block#

Residual module enhanced with Squeeze-and-Excitation: Re-implementation of the SE-Resnet block based on: “Hu et al., Squeeze-and-Excitation Networks, https://arxiv.org/abs/1709.01507”. spatial_dims (int) – number of spatial dimensions, could be 1, 2, or 3. in_channels (int) – number of input channels. n_chns_1 (int) – number of output channels in the 1st convolution. n_chns_2 (int) – number of output channels in the 2nd convolution. n_chns_3 (int) – number of output channels in the 3rd convolution. conv_param_1 (Optional[Dict]) – additional parameters to the 1st convolution. Defaults to {"kernel_size": 1, "norm": Norm.BATCH, "act": ("relu", {"inplace": True})} conv_param_2 (Optional[Dict]) – additional parameters to the 2nd convolution. Defaults to {"kernel_size": 3, "norm": Norm.BATCH, "act": ("relu", {"inplace": True})} conv_param_3 (Optional[Dict]) – additional parameters to the 3rd

convolution. Defaults to {"kernel_size": 1, "norm": Norm.BATCH, "act": None} project (Optional[Convolution]) – in the case of residual chns and output chns doesn't match, a project (Conv) layer/block is used to adjust the number of chns. In SENET, it is consisted with a Conv layer as well as a Norm layer. Defaults to None (chns are matchable) or a Conv layer with kernel size 1. r (int) – the reduction ratio r in the paper. Defaults to 2. acti_type_1 (Union[Tuple[str, Dict], str]) – activation type of the hidden squeeze layer. Defaults to "relu". acti_type_2 (Union[Tuple[str, Dict], str]) – activation type of the output squeeze layer. Defaults to "sigmoid". acti_type_final (Union[Tuple[str, Dict], str, None]) – activation type of the end of the block. Defaults to "relu". See also monai.networks.blocks.ChannelSELayer x (Tensor) – in shape (batch, in_channels, spatial_1[, spatial_2, …]). Tensor

Squeeze-and-Excitation Bottleneck#

Bottleneck for SENet154.

Squeeze-and-Excitation Resnet Bottleneck#

ResNet bottleneck with a Squeeze-and-Excitation module. It follows Caffe implementation and uses strides=stride in conv1 and not in conv2 (the latter is used in the torchvision implementation of ResNet).

Squeeze-and-Excitation ResNeXt Bottleneck#

ResNeXt bottleneck type C with a Squeeze-and-Excitation module.

Simple ASPP#

A simplified version of the atrous spatial pyramid pooling (ASPP) module. Chen et al., Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. https://arxiv.org/abs/1802.02611 Wang et al., A Noise-robust Framework for Automatic Segmentation of COVID-19 Pneumonia Lesions from CT Images. https://ieeexplore.ieee.org/document/9109297 spatial_dims (int) – number of spatial dimensions, could be 1, 2, or 3. in_channels (int) – number of input channels. conv_out_channels (int) – number of output channels of each atrous conv. The final number of output channels is conv_out_channels * len(kernel_sizes). kernel_sizes (Sequence[int]) – a sequence of four convolutional kernel sizes. Defaults to (1, 3, 3, 3) for four (dilated) convolutions. dilations (Sequence[int]) – a sequence of four convolutional dilation parameters. Defaults to (1, 2, 4, 6) for four (dilated) convolutions. norm_type (Union[Tuple, str, None]) – final kernel-size-one convolution normalization type. Defaults to batch norm. acti_type (Union[Tuple, str, None]) – final kernel-size-one convolution activation type. Defaults to leaky ReLU. bias (bool) – whether to have a bias term in convolution blocks. Defaults to False. According to Performance Tuning Guide, if a conv layer is directly followed by a batch norm layer, bias should be False. ValueError – When kernel_sizes length differs from dilations. See also monai.networks.layers.Act monai.networks.layers.Conv monai.networks.layers.Norm x (Tensor) – in shape (batch, channel, spatial_1[, spatial_2, …]). Tensor

MaxAvgPooling#

Downsample with both maxpooling and avgpooling, double the channel size by concatenating the downsampled feature maps. spatial_dims (int) – number of spatial dimensions of the input image. kernel_size (Union[Sequence[int], int]) – the kernel size of both pooling operations. stride (Union[Sequence[int], int, None]) – the stride of the window. Default value is kernel_size. padding (Union[Sequence[int], int]) – implicit zero padding to be added to both pooling operations. ceil_mode (bool) – when True, will use ceil instead of floor to compute the output shape. x (Tensor) – Tensor in shape (batch, channel, spatial_1[, spatial_2, …]). Tensor Tensor in shape (batch, 2*channel, spatial_1[, spatial_2, …]).

Upsampling#

Upsamples data by scale_factor. Supported modes are: “deconv”: uses a transposed convolution. “deconvgroup”: uses a transposed group convolution. “nontrainable”: uses torch.nn.Upsample. “pixelshuffle”: uses monai.networks.blocks.SubpixelUpsample. This module can optionally take a pre-convolution (often used to map the number of features from in_channels to out_channels). spatial_dims (int) – number of spatial dimensions of the input image. in_channels (Optional[int]) – number of channels of the input image. out_channels (Optional[int]) – number of channels of the output image. Defaults to in_channels. scale_factor (Union[Sequence[float], float]) – multiplier for spatial size. Has to match input size if it is a tuple. Defaults to 2. kernel_size (Union[Sequence[float], float, None]) – kernel size used during transposed convolutions. Defaults to scale_factor. size (Union[Tuple[int], int, None]) – spatial size of the output image. Only used when mode is UpsampleMode.NONTRAINABLE. In torch.nn.functional.interpolate, only one of size or scale_factor should be defined, thus if size is defined, scale_factor will not be used. Defaults to None. mode (Union[UpsampleMode, str]) – {"deconv", "deconvgroup", "nontrainable", "pixelshuffle"}. Defaults to "deconv". pre_conv (Union[Module, str, None]) – a conv block applied before upsampling. Defaults to “default”. When conv_block is "default", one reserved conv layer will be utilized when Only used in the “nontrainable” or “pixelshuffle” mode. interp_mode (str) – {"nearest", "linear", "bilinear", "bicubic", "trilinear"} Only used in the “nontrainable” mode. If ends with "linear" will use spatial dims to determine the correct interpolation. This corresponds to linear, bilinear, trilinear for 1D, 2D, and 3D respectively. The interpolation mode. Defaults to "linear". See also: https://pytorch.org/docs/stable/generated/torch.nn.Upsample.html align_corners (Optional[bool]) – set the align_corners parameter of torch.nn.Upsample. Defaults to True. Only used in the “nontrainable” mode. bias (bool) – whether to have a bias term in the default preconv and deconv layers. Defaults to True. apply_pad_pool (bool) – if True the upsampled tensor is padded then average pooling is applied with a kernel the size of scale_factor with a stride of 1. See also: monai.networks.blocks.SubpixelUpsample. Only used in the “pixelshuffle” mode. alias of UpSample Upsample via using a subpixel CNN. This module supports 1D, 2D and 3D input images. The module is consisted with two parts. First of all, a convolutional layer is employed to increase the number of channels into: in_channels * (scale_factor ** dimensions). Secondly, a pixel shuffle manipulation is utilized to aggregates the feature maps from low resolution space and build the super resolution space. The first part of the module is not fixed, a sequential layers can be used to replace the default single layer. See: Shi et al., 2016, “Real-Time Single Image and Video

Super-Resolution Using a nEfficient Sub-Pixel Convolutional Neural Network.” See: Aitken et al., 2017, “Checkerboard artifact free sub-pixel convolution”. The idea comes from: https://arxiv.org/abs/1609.05158 The pixel shuffle mechanism refers to: https://pytorch.org/docs/stable/generated/torch.nn.PixelShuffle.html#torch.nn.PixelShuffle. and: pytorch/pytorch#6340. spatial_dims (int) – number of spatial dimensions of the input image. in_channels (Optional[int]) – number of channels of the input image. out_channels (Optional[int]) – optional number of channels of the output image. scale_factor (int) – multiplier for spatial size. Defaults to 2. conv_block (Union[Module, str, None]) – a conv block to extract feature maps before upsampling. Defaults to None. When conv_block is "default", one reserved conv layer will be utilized. When conv_block is an nn.module, please ensure the output number of channels is divisible (scale_factor ** dimensions). a conv block to extract feature maps before upsampling. Defaults to None. When conv_block is "default", one reserved conv layer will be utilized. When conv_block is an nn.module, please ensure the output number of channels is divisible (scale_factor ** dimensions). apply_pad_pool (bool) – if True the upsampled tensor is padded then average pooling is applied with a kernel the size of scale_factor with a stride of 1. This implements the nearest neighbour resize convolution component of subpixel convolutions described in Aitken et al. bias (bool) – whether to have a bias term in the default conv_block. Defaults to True. x (Tensor) – Tensor in shape (batch, channel, spatial_1[, spatial_2, …). Tensor alias of SubpixelUpsample alias of SubpixelUpsample

Registration Residual Conv Block#

A block with skip links and layer - norm - activation. Only changes the number of channels, the spatial size is kept same. spatial_dims (int) – number of spatial dimensions in_channels (int) – number of input channels out_channels (int) – number of output channels num_layers (int) – number of layers inside the block kernel_size (int) – kernel_size x (Tensor) – Tensor in shape (batch, in_channels, insize_1, insize_2, [insize_3]) Tensor Tensor in shape (batch, out_channels, insize_1, insize_2, [insize_3]), with the same spatial size as x

Registration Down Sample Block#

A down-sample module used in RegUNet to half the spatial size. The number of channels is kept same. DeepReg (DeepRegNet/DeepReg) spatial_dims (int) – number of spatial dimensions. channels (int) – channels pooling (bool) – use MaxPool if True, strided conv if False Halves the spatial dimensions and keeps the same channel. output in shape (batch, channels, insize_1 / 2, insize_2 / 2, [insize_3 / 2]), x (Tensor) – Tensor in shape (batch, channels, insize_1, insize_2, [insize_3]) ValueError – when input spatial dimensions are not even. Tensor

Registration Extraction Block#

The Extraction Block used in RegUNet. Extracts feature from each extract_levels and takes the average. spatial_dims (int) – number of spatial dimensions extract_levels (Tuple[int]) – spatial levels to extract feature from, 0 refers to the input scale num_channels (Union[Tuple[int], List[int]]) – number of channels at each scale level, List or Tuple of length equals to depth of the RegNet out_channels (int) – number of output channels kernel_initializer (Optional[str]) – kernel initializer activation

(Optional[str]) – kernel activation function x (List[Tensor]) – Decoded feature at different spatial levels, sorted from deep to shallow image_size (List[int]) – output image size Tensor Tensor of shape (batch, out_channels, size1, size2, size3), where (size1, size2, size3) = image_size

LocalNet DownSample Block#

A down-sample module that can be used for LocalNet, based on: Weakly-supervised convolutional neural networks for multimodal image registration. Label-driven weakly-supervised learning for multimodal deformable image registration. DeepReg (DeepRegNet/DeepReg) spatial_dims (int) – number of spatial dimensions. in_channels (int) – number of input channels. out_channels (int) – number of output channels. kernel_size (Union[Sequence[int], int]) – convolution kernel size. NotImplementedError – when kernel_size is even Halves the spatial dimensions. A tuple of (x, mid) is returned: x is the downsample result, in shape (batch, out_channels, insize_1 / 2, insize_2 / 2, [insize_3 / 2]), mid is the mid-level feature, in shape (batch, out_channels, insize_1, insize_2, [insize_3]) x – Tensor in shape (batch, in_channels, insize_1, insize_2, [insize_3]) ValueError – when input spatial dimensions are not even. Tuple[Tensor, Tensor]

LocalNet UpSample Block#

A up-sample module that can be used for LocalNet, based on: Weakly-supervised convolutional neural networks for multimodal image registration. Label-driven weakly-supervised learning for multimodal deformable image registration. DeepReg (DeepRegNet/DeepReg) spatial_dims (int) – number of spatial dimensions. in_channels (int) – number of input channels. out_channels (int) – number of output channels. ValueError – when in_channels != 2 * out_channels Halves the channel and doubles the spatial dimensions. x – feature to be up-sampled, in shape (batch, in_channels, insize_1, insize_2, [insize_3]) mid – mid-level feature saved during down-sampling, in shape (batch, out_channels, midsize_1, midsize_2, [midsize_3]) ValueError – when midsize != insize * 2 Tensor

LocalNet Feature Extractor Block#

A feature-extraction module that can be used for LocalNet, based on: Weakly-supervised convolutional neural networks for multimodal image registration. Label-driven weakly-supervised learning for multimodal deformable image registration. DeepReg (DeepRegNet/DeepReg) Args: spatial_dims: number of spatial dimensions. in_channels: number of input channels. out_channels: number of output channels. act: activation type and arguments. Defaults to ReLU. kernel_initializer: kernel initializer. Defaults to None. x – Tensor in shape (batch, in_channels, insize_1, insize_2, [insize_3]) Tensor

MLP Block#

A multi-layer perceptron block, based on: “Dosovitskiy et al., An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale ” hidden_size (int) – dimension of hidden layer. mlp_dim (int) – dimension of feedforward layer. If 0, hidden_size will

be used. dropout_rate (float) – faction of the input units to drop. act (Union[Tuple, str]) – activation type and arguments. Defaults to GELU. dropout_mode – dropout mode, can be “vit” or “swin”. “vit” mode uses two dropout instances as implemented in google-research/vision_transformer “swin” corresponds to one instance as implemented in microsoft/Swin-Transformer 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.

Patch Embedding Block#

A patch embedding block, based on: “Dosovitskiy et al., An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale ” Example: in_channels (int) – dimension of input channels. img_size (Union[Sequence[int], int]) – dimension of input image. patch_size (Union[Sequence[int], int]) – dimension of patch size. hidden_size (int) – dimension of hidden layer. num_heads (int) – number of attention heads. pos_embed (str) – position embedding layer type. dropout_rate (float) – faction of the input units to drop. spatial_dims (int) – number of spatial dimensions. 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.

FactorizedIncreaseBlock#

Up-sampling the features by two using linear interpolation and convolutions. in_channel (int) – number of input channels out_channel (int) – number of output channels spatial_dims (int) – number of spatial dimensions act_name (Union[Tuple, str]) – activation layer type and arguments. norm_name (Union[Tuple, str]) – feature normalization type and arguments.

FactorizedReduceBlock#

Down-sampling the feature by 2 using stride. The length along each spatial dimension must be a multiple of 2. in_channel (int) – number of input channels out_channel (int) – number of output channels. spatial_dims (int) – number of spatial dimensions. act_name (Union[Tuple, str]) – activation layer type and arguments. norm_name (Union[Tuple, str]) – feature normalization type and arguments. The length along each spatial dimension must be a multiple of 2. Tensor

P3DActiConvNormBlock#

– (act) – (conv) – (norm) – in_channel (int) – number of input channels. out_channel (int) – number of output channels. kernel_size (int) – kernel size to be expanded to 3D. padding (int) – padding size to be expanded to 3D. mode (int) – mode for the anisotropic kernels: 0: (k, k, 1), (1, 1, k), 1: (k, 1, k), (1, k, 1), 2: (1, k, k). (k, 1, 1). mode for the anisotropic kernels: 0: (k, k, 1), (1, 1, k), 1: (k, 1, k), (1, k, 1), 2: (1, k, k). (k, 1, 1). act_name (Union[Tuple, str]) – activation layer type and arguments. norm_name

(Union[Tuple, str]) – feature normalization type and arguments.

ActiConvNormBlock#

– (Acti) – (Conv) – (Norm) – in_channel (int) – number of input channels. out_channel (int) – number of output channels. kernel_size (int) – kernel size of the convolution. padding (int) – padding size of the convolution. spatial_dims (int) – number of spatial dimensions. act_name (Union[Tuple, str]) – activation layer type and arguments. norm_name (Union[Tuple, str]) – feature normalization type and arguments.

Warp#

Warp an image with given dense displacement field (DDF). For pytorch native APIs, the possible values are: mode: "nearest", "bilinear", "bicubic". padding_mode: "zeros", "border", "reflection" See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html For MONAI C++/CUDA extensions, the possible values are: mode: "nearest", "bilinear", "bicubic", 0, 1, … padding_mode: "zeros", "border", "reflection", 0, 1, … See also: monai.networks.layers.grid_pull image (Tensor) – Tensor in shape (batch, num_channels, H, W[, D]) ddf (Tensor) – Tensor in the same spatial size as image, in shape (batch, spatial_dims, H, W[, D]) warped_image in the same shape as image (batch, num_channels, H, W[, D])

DVF2DDF#

Layer calculates a dense displacement field (DDF) from a dense velocity field (DVF) with scaling and squaring. DeepReg (DeepRegNet/DeepReg) dvf (Tensor) – dvf to be transformed, in shape (batch, spatial_dims, H, W[,D]) Tensor a dense displacement field

VarNetBlock#

A variational block based on Sriram et. al., “End-to-end variational networks for accelerated MRI reconstruction”. It applies data consistency and refinement to the intermediate kspace and combines those results. Modified and adopted from: facebookresearch/fastMRI refinement_model (Module) – the model used for refinement (typically a U-Net but can be any deep learning model that performs well when the input and output are in image domain (e.g., a convolutional network). spatial_dims (int) – is 2 for 2D data and is 3 for 3D data current_kspace (Tensor) – Predicted kspace from the previous block. It’s a 2D kspace (B,C,H,W,2) with the last dimension being 2 (for real/imaginary parts) and C denoting the coil dimension. 3D data will have the shape (B,C,H,W,D,2). ref_kspace (Tensor) – reference kspace for applying data consistency (is the under-sampled kspace in MRI reconstruction). Its shape is the same as current_kspace. mask (Tensor) – the under-sampling mask with shape (1,1,1,W,1) for 2D data or (1,1,1,1,D,1) for 3D data. sens_maps (Tensor) – coil sensitivity maps with the same shape as current_kspace Tensor Output of VarNetBlock with the same shape as current_kspace Applies data consistency to input x. Suppose x is an intermediate estimate of the kspace and ref_kspace is the reference under-sampled measurement. This function returns mask * (x - ref_kspace).

View this as the residual between the original under-sampled kspace and the estimate given by the network. x (Tensor) – 2D kspace (B,C,H,W,2) with the last dimension being 2 (for real/imaginary parts) and C denoting the coil dimension. 3D data will have the shape (B,C,H,W,D,2). ref_kspace (Tensor) – original under-sampled kspace with the same shape as x. mask (Tensor) – the under-sampling mask with shape (1,1,1,W,1) for 2D data or (1,1,1,1,D,1) for 3D data. Tensor Output of DC block with the same shape as x

N-Dim Fourier Transform#

Pytorch-based fft for spatial_dims-dim signals. “centered” means this function automatically takes care of the required ifft and fft shifts. This is equivalent to do ifft in numpy based on numpy.fft.fftn, numpy.fft.fftshift, and numpy.fft.ifftshift im (Tensor) – image that can be 1) real-valued: the shape is (C,H,W) for 2D spatial inputs and (C,H,W,D) for 3D, or 2) complex-valued: the shape is (C,H,W,2) for 2D spatial data and (C,H,W,D,2) for 3D. C is the number of channels. spatial_dims (int) – number of spatial dimensions (e.g., is 2 for an image, and is 3 for a volume) is_complex (bool) – if True, then the last dimension of the input im is expected to be 2 (representing real and imaginary channels) Tensor “out” which is the output kspace (fourier of im) Example Pytorch-based ifft for spatial_dims-dim signals. “centered” means this function automatically takes care of the required ifft and fft shifts. This is equivalent to do fft in numpy based on numpy.fft.ifftn, numpy.fft.fftshift, and numpy.fft.ifftshift ksp (Tensor) – k-space data that can be 1) real-valued: the shape is (C,H,W) for 2D spatial inputs and (C,H,W,D) for 3D, or 2) complex-valued: the shape is (C,H,W,2) for 2D spatial data and (C,H,W,D,2) for 3D. C is the number of channels. spatial_dims (int) – number of spatial dimensions (e.g., is 2 for an image, and is 3 for a volume) is_complex (bool) – if True, then the last dimension of the input ksp is expected to be 2 (representing real and imaginary channels) Tensor “out” which is the output image (inverse fourier of ksp) Example Similar to np.roll but applies to PyTorch Tensors x (Tensor) – input data (k-space or image) that can be 1) real-valued: the shape is (C,H,W) for 2D spatial inputs and (C,H,W,D) for 3D, or 2) complex-valued: the shape is (C,H,W,2) for 2D spatial data and (C,H,W,D,2) for 3D. C is the number of channels. shift (List[int]) – the amount of shift along each of shift_dims dimensions shift_dims (List[int]) – dimensions over which the shift is applied Tensor shifted version of x Note This function is called when fftshift and ifftshift are not available in the running pytorch version Similar to roll but for only one dim. x (Tensor) – input data (k-space or image) that can be 1) real-valued: the shape is (C,H,W) for 2D spatial inputs and (C,H,W,D) for 3D, or 2) complex-valued: the shape is (C,H,W,2) for 2D spatial data and (C,H,W,D,2) for 3D. C is the number of channels. shift (int) – the amount of shift along each of shift_dims dimension shift_dim (int) – the dimension over which the shift is applied Tensor 1d-shifted version of x Note This function is called when fftshift and ifftshift are not available in the running pytorch version Similar to np.fft.fftshift but applies to PyTorch Tensors x (Tensor) – input data (k-space or image) that can be 1) real-valued: the shape is (C,H,W) for 2D spatial inputs and (C,H,W,D) for 3D, or 2) complex-valued: the shape is (C,H,W,2) for 2D spatial data and (C,H,W,D,2) for 3D. C is the number of channels. shift_dims (List[int]) – dimensions over which the shift is applied Tensor fft-shifted version of x Note This function is called when fftshift is not available in the running pytorch version Similar to np.fft.ifftshift but applies to PyTorch Tensors x (Tensor) – input data (k-space or image) that can be 1) real-valued: the shape is (C,H,W) for 2D spatial inputs and (C,H,W,D) for 3D, or 2) complex-valued: the shape is (C,H,W,2) for 2D spatial data and (C,H,W,D,2) for 3D. C is the number of channels. shift_dims (List[int]) – dimensions over which the shift is applied Tensor

ifft-shifted version of x Note This function is called when ifftshift is not available in the running pytorch version

Factories#

Defines factories for creating layers in generic, extensible, and dimensionally independent ways. A separate factory object is created for each type of layer, and factory functions keyed to names are added to these objects. Whenever a layer is requested the factory name and any necessary arguments are passed to the factory object. The return value is typically a type but can be any callable producing a layer object. The factory objects contain functions keyed to names converted to upper case, these names can be referred to as members of the factory so that they can function as constant identifiers. eg. instance normalization is named Norm.INSTANCE. For example, to get a transpose convolution layer the name is needed and then a dimension argument is provided which is passed to the factory function: This allows the dimension value to be set in the constructor, for example so that the dimensionality of a network is parameterizable. Not all factories require arguments after the name, the caller must be aware which are required. Defining new factories involves creating the object then associating it with factory functions: Typically the caller of a factory would know what arguments to pass (ie. the dimensionality of the requested type) but can be parameterized with the factory name and the arguments to pass to the created type at instantiation time: Factory object for creating layers, this uses given factory functions to actually produce the types or constructing callables. These functions are referred to by name and can be added at any time. Add the factory function to this object under the given name. None Decorator for adding a factory function with the given name. Callable Get the constructor for the given factory name and arguments. TypeError – When factory_name is not a str. Any Produces all factory names. Tuple[str, …]

split_args#

Split arguments in a way to be suitable for using with the factory types. If args is a string it’s interpreted as the type name. args (str or a tuple of object name and kwarg dict) – input arguments to be parsed. TypeError – When args type is not in Union[str, Tuple[Union[str, Callable], dict]]. Examples:

Dropout#

The supported members are: DROPOUT, ALPHADROPOUT. Please see monai.networks.layers.split_args for additional args parsing.

Act#

The supported members are: ELU, RELU, LEAKYRELU, PRELU, RELU6, SELU, CELU, GELU, SIGMOID, TANH, SOFTMAX, LOGSOFTMAX, SWISH, MEMSWISH, MISH. Please see monai.networks.layers.split_args for additional args parsing.

Norm#

The supported members are: INSTANCE, BATCH, GROUP, LAYER, LOCALRESPONSE, SYNCBATCH, INSTANCE_NVFUSER. Please see monai.networks.layers.split_args for additional args parsing.

Conv#

The supported members are: CONV, CONVTRANS. Please see monai.networks.layers.split_args for additional args parsing.

Pad#

The supported members are: REPLICATIONPAD, CONSTANTPAD. Please see monai.networks.layers.split_args for additional args parsing.

Pool#

The supported members are: MAX, ADAPTIVEMAX, AVG, ADAPTIVEAVG. Please see monai.networks.layers.split_args for additional args parsing.

ChannelPad#

Expand the input tensor’s channel dimension from length in_channels to out_channels, by padding or a projection. spatial_dims (int) – number of spatial dimensions of the input image. in_channels (int) – number of input channels. out_channels (int) – number of output channels. mode (Union[ChannelMatching, str]) – {"pad", "project"} Specifies handling residual branch and conv branch channel mismatches. Defaults to "pad". "pad": with zero padding. "project": with a trainable conv with kernel size one. {"pad", "project"} Specifies handling residual branch and conv branch channel mismatches. Defaults to "pad". "pad": with zero padding. "project": with a trainable conv with kernel size one. 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. Tensor

SkipConnection#

Combine the forward pass input with the result from the given submodule: The available modes are "cat", "add", "mul". submodule – the module defines the trainable branch. dim (int) – the dimension over which the tensors are concatenated. Used when mode is "cat". mode (Union[str, SkipMode]) – "cat", "add", "mul". defaults to "cat". 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. Tensor

Flatten#

Flattens the given input in the forward pass to be [B,-1] in shape. 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. Tensor

Reshape#

Reshapes input tensors to the given shape (minus batch dimension), retaining original batch size. Given a shape list/tuple shape of integers (s0, s1, … , sn), this layer will reshape input tensors of shape (batch, s0 * s1 * … * sn) to shape (batch, s0, s1, … , sn). shape (int) – list/tuple of integer shape dimensions 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. Tensor

separable_filtering#

Apply 1-D convolutions along each spatial dimension of x. x (Tensor) – the input image. must have shape (batch, channels, H[, W, …]). kernels (List[Tensor]) – kernel along each spatial dimension. could be a single kernel (duplicated for all spatial dimensions), or a list of spatial_dims number of kernels. mode (string, optional) – padding mode passed to convolution class. 'zeros', 'reflect', 'replicate' or 'circular'. Default: 'zeros'. See torch.nn.Conv1d() for more information. TypeError – When x is not a torch.Tensor. Examples: Tensor

apply_filter#

Filtering x with kernel independently for each batch and channel respectively. x (Tensor) – the input image, must have shape (batch, channels, H[, W, D]). kernel (Tensor) – kernel must at least have the spatial shape (H_k[, W_k, D_k]). kernel shape must be broadcastable to the batch and channels dimensions of x. kwargs – keyword arguments passed to conv*d() functions. Tensor The filtered x. Examples:

GaussianFilter#

spatial_dims (int) – number of spatial dimensions of the input image. must have shape (Batch, channels, H[, W, …]). sigma (Union[Sequence[float], float, Sequence[Tensor], Tensor]) – std. could be a single value, or spatial_dims number of values. truncated (float) – spreads how many stds. approx (str) – discrete Gaussian kernel type, available options are “erf”, “sampled”, and “scalespace”. erf approximation interpolates the error function; sampled uses a sampled Gaussian kernel; scalespace corresponds to https://en.wikipedia.org/wiki/Scale_space_implementation#The_discrete_Gaussian_kernel based on the modified Bessel functions. discrete Gaussian kernel type, available options are “erf”, “sampled”, and “scalespace”. erf approximation interpolates the error function; sampled uses a sampled Gaussian

kernel; scalespace corresponds to https://en.wikipedia.org/wiki/Scale_space_implementation#The_discrete_Gaussian_kernel based on the modified Bessel functions. requires_grad (bool) – whether to store the gradients for sigma. if True, sigma will be the initial value of the parameters of this module (for example parameters() iterator could be used to get the parameters); otherwise this module will fix the kernels using sigma as the std. x (Tensor) – in shape [Batch, chns, H, W, D]. Tensor

MedianFilter#

Apply median filter to an image. radius (Union[Sequence[int], int]) – the blurring kernel radius (radius of 1 corresponds to 3x3x3 kernel when spatial_dims=3). filtered input tensor. Example: in_tensor (Tensor) – input tensor, median filtering will be applied to the last spatial_dims dimensions. number_of_passes – median filtering will be repeated this many times Tensor

median_filter#

Apply median filter to an image. in_tensor (Tensor) – input tensor; median filtering will be applied to the last spatial_dims dimensions. kernel_size (Sequence[int]) – the convolution kernel size. spatial_dims (int) – number of spatial dimensions to apply median filtering. kernel (Optional[Tensor]) – an optional customized kernel. kwargs – additional parameters to the conv. Tensor the filtered input tensor, shape remains the same as in_tensor Example:

BilateralFilter#

Blurs the input tensor spatially whilst preserving edges. Can run on 1D, 2D, or 3D, tensors (on top of Batch and Channel dimensions). Two implementations are provided, an exact solution and a much faster approximation which uses a permutohedral lattice. https://en.wikipedia.org/wiki/Bilateral_filter https://graphics.stanford.edu/papers/permutohedral/ input – input tensor. sigma (color) – the standard deviation of the spatial blur. Higher values can hurt performance when not using the approximate method (see fast approx). sigma – the standard deviation of the color blur. Lower values preserve edges better whilst higher values tend to a simple gaussian spatial blur. approx (fast) – This flag chooses between two implementations. The approximate method may produce artifacts in some scenarios whereas the exact solution may be intolerably slow for high spatial standard deviations. output tensor. output (torch.Tensor) Defines a formula for differentiating the operation with backward mode automatic differentiation (alias to the vjp function). This function is to be overridden by all subclasses. It must accept a context ctx as the first argument, followed by as many outputs as the forward() returned (None will be passed in for non tensor outputs of the forward function), and it should return as many tensors, as there were inputs to forward(). Each argument is the gradient w.r.t the given output, and each returned value should be the gradient w.r.t. the corresponding input. If an input is not a Tensor or is a Tensor not requiring grads, you can just pass None as a gradient for that input. The context can be used to retrieve tensors saved during the forward pass. It also has an attribute ctx.needs_input_grad as a tuple of booleans representing whether each input needs gradient. E.g., backward() will have ctx.needs_input_grad[0] = True if the first input to forward() needs gradient computated w.r.t. the output. Performs the operation. This function is to be overridden

by all subclasses. It must accept a context ctx as the first argument, followed by any number of arguments (tensors or other types). The context can be used to store arbitrary data that can be then retrieved during the backward pass. Tensors should not be stored directly on ctx (though this is not currently enforced for backward compatibility). Instead, tensors should be saved either with ctx.save_for_backward() if they are intended to be used in backward (equivalently, vjp) or ctx.save_for_forward() if they are intended to be used for in jvp.

PHLFilter#

Filters input based on arbitrary feature vectors. Uses a permutohedral lattice data structure to efficiently approximate n-dimensional gaussian filtering. Complexity is broadly independent of kernel size. Most applicable to higher filter dimensions and larger kernel sizes. https://graphics.stanford.edu/papers/permutohedral/ input – input tensor to be filtered. features – feature tensor used to filter the input. sigmas – the standard deviations of each feature in the filter. output tensor. output (torch.Tensor)

GaussianMixtureModel#

Takes an initial labeling and uses a mixture of Gaussians to approximate each classes distribution in the feature space. Each unlabeled element is then assigned a probability of belonging to each class based on it’s fit to each classes approximated distribution. https://en.wikipedia.org/wiki/Mixture_model

SavitzkyGolayFilter#

Convolve a Tensor along a particular axis with a Savitzky-Golay kernel. window_length (int) – Length of the filter window, must be a positive odd integer. order (int) – Order of the polynomial to fit to each window, must be less than window_length. axis (optional) – Axis along which to apply the filter kernel. Default 2 (first spatial dimension). mode (string, optional) – padding mode passed to convolution class. 'zeros', 'reflect', 'replicate' or Default ('circular'.) – 'zeros'. See torch.nn.Conv1d() for more information. x (Tensor) – Tensor or array-like to filter. Must be real, in shape [Batch, chns, spatial1, spatial2, ...] and have a device type of 'cpu'. x filtered by Savitzky-Golay kernel with window length self.window_length using polynomials of order self.order, along axis specified in self.axis. torch.Tensor

HilbertTransform#

Determine the analytical signal of a Tensor along a particular axis. axis (int) – Axis along which to apply Hilbert transform. Default 2 (first spatial dimension). n (Optional[int]) – Number of Fourier components (i.e. FFT size). Default: x.shape[axis]. x (Tensor) – Tensor or array-like to transform. Must be real and in shape [Batch, chns, spatial1, spatial2, ...]. Analytical signal of x, transformed along axis specified in self.axis using FFT of size self.N. The absolute value of x_ht relates to the envelope of x along axis self.axis. torch.Tensor

Affine Transform#

Apply affine transformations with a batch of affine matrices. When normalized=False and reverse_indexing=True, it does the commonly used resampling in the ‘pull’ direction following the scipy.ndimage.affine_transform convention. In this case theta is equivalent to (ndim+1, ndim+1) input matrix of scipy.ndimage.affine_transform, operates on homogeneous coordinates. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.affine_transform.html When normalized=True and reverse_indexing=False, it applies theta to the normalized coordinates (coords. in the range of [-1, 1]) directly. This is often used with align_corners=False to achieve resolution-agnostic resampling, thus useful as a part of trainable modules such as the spatial transformer networks. See also: https://pytorch.org/tutorials/intermediate/spatial_transformer_tutorial.html spatial_size (Union[Sequence[int], int, None]) – output spatial shape, the full output shape will be [N, C, *spatial_size] where N and C are inferred from the src input of self.forward. normalized (bool) – indicating whether the provided affine matrix theta is defined for the normalized coordinates. If normalized=False, theta will be converted to operate on normalized coordinates as pytorch affine_grid works with the normalized coordinates. mode (str) – {"bilinear", "nearest"} Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html padding_mode (str) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "zeros". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html align_corners (bool) – see also https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html. reverse_indexing (bool) – whether to reverse the spatial indexing of image and coordinates. set to False if theta follows pytorch’s default “D, H, W” convention. set to True if theta follows scipy.ndimage default “i, j, k” convention. zero_centered (Optional[bool]) – whether the affine is applied to coordinates in a zero-centered value range. With zero_centered=True, for example, the center of rotation will be the spatial center of the input; with zero_centered=False, the center of rotation will be the origin of the input. This option is only available when normalized=False, where the default behaviour is False if unspecified. See also: monai.networks.utils.normalize_transform(). theta must be an affine transformation matrix with shape 3x3 or Nx3x3 or Nx2x3 or 2x3 for spatial 2D transforms, 4x4 or Nx4x4 or Nx3x4 or 3x4 for spatial 3D transforms, where N is the batch size. theta will be converted into float Tensor for the computation. src (array_like) – image in spatial 2D or 3D (N, C, spatial_dims), where N is the batch dim, C is the number of channels. theta (array_like) – Nx3x3, Nx2x3, 3x3, 2x3 for spatial 2D inputs, Nx4x4, Nx3x4, 3x4, 4x4 for spatial 3D inputs. When the batch dimension is omitted, theta will be repeated N times, N is the batch dim of src. spatial_size (Union[Sequence[int], int, None]) – output spatial shape, the full output shape will be [N, C, *spatial_size] where N and C are inferred from the src. TypeError – When theta is not a torch.Tensor. ValueError – When theta is not one of [Nxdxd, dxd]. ValueError – When theta is not one of [Nx3x3, Nx4x4]. TypeError – When src is not a torch.Tensor. ValueError – When src spatially is not one of [2D, 3D]. ValueError – When affine and image batch dimension differ. Tensor

grid_pull#

Sample an image with respect to a deformation field. interpolation can be an int, a string or an InterpolationType. Possible values are: A list of values can be provided, in the order [W, H, D], to specify dimension-specific interpolation orders. bound can be

an int, a string or a BoundType. Possible values are: A list of values can be provided, in the order [W, H, D], to specify dimension-specific boundary conditions. sliding is a specific condition than only applies to flow fields (with as many channels as dimensions). It cannot be dimension-specific. Note that: dft corresponds to circular padding dct2 corresponds to Neumann boundary conditions (symmetric) dst2 corresponds to Dirichlet boundary conditions (antisymmetric) See also https://en.wikipedia.org/wiki/Discrete_cosine_transform https://en.wikipedia.org/wiki/Discrete_sine_transform help(monai._C.BoundType) help(monai._C.InterpolationType) input (Tensor) – Input image. (B, C, Wi, Hi, Di). grid (Tensor) – Deformation field. (B, Wo, Ho, Do, 1|2|3). interpolation (int or list[int] , optional) – Interpolation order. Defaults to 'linear'. bound (BoundType, or list[BoundType], optional) – Boundary conditions. Defaults to 'zero'. extrapolate (bool) – Extrapolate out-of-bound data. Defaults to True. Deformed image (B, C, Wo, Ho, Do). output (torch.Tensor)

grid_push#

Splat an image with respect to a deformation field (pull adjoint). interpolation can be an int, a string or an InterpolationType. Possible values are: A list of values can be provided, in the order [W, H, D], to specify dimension-specific interpolation orders. bound can be an int, a string or a BoundType. Possible values are: A list of values can be provided, in the order [W, H, D], to specify dimension-specific boundary conditions. sliding is a specific condition than only applies to flow fields (with as many channels as dimensions). It cannot be dimension-specific. Note that: dft corresponds to circular padding dct2 corresponds to Neumann boundary conditions (symmetric) dst2 corresponds to Dirichlet boundary conditions (antisymmetric) See also https://en.wikipedia.org/wiki/Discrete_cosine_transform https://en.wikipedia.org/wiki/Discrete_sine_transform help(monai._C.BoundType) help(monai._C.InterpolationType) input (Tensor) – Input image (B, C, Wi, Hi, Di). grid (Tensor) – Deformation field (B, Wi, Hi, Di, 1|2|3). shape – Shape of the source image. interpolation (int or list[int] , optional) – Interpolation order. Defaults to 'linear'. bound (BoundType, or list[BoundType], optional) – Boundary conditions. Defaults to 'zero'. extrapolate (bool) – Extrapolate out-of-bound data. Defaults to True. Splatted image (B, C, Wo, Ho, Do). output (torch.Tensor)

grid_count#

Splatting weights with respect to a deformation field (pull adjoint). This function is equivalent to applying grid_push to an image of ones. interpolation can be an int, a string or an InterpolationType. Possible values are: A list of values can be provided, in the order [W, H, D], to specify dimension-specific interpolation orders. bound can be an int, a string or a BoundType. Possible values are: A list of values can be provided, in the order [W, H, D], to specify dimension-specific boundary conditions. sliding is a specific condition than only applies to flow fields (with as many channels as dimensions). It cannot be dimension-specific. Note that: dft corresponds to circular padding dct2 corresponds to Neumann boundary conditions (symmetric) dst2 corresponds to Dirichlet boundary conditions (antisymmetric) See also https://en.wikipedia.org/wiki/Discrete_cosine_transform https://en.wikipedia.org/wiki/Discrete_sine_transform help(monai._C.BoundType) help(monai._C.InterpolationType) grid (Tensor) – Deformation field (B, Wi, Hi, Di, 2|3). shape – shape of the source image. interpolation (int or list[int] , optional) –

Interpolation order. Defaults to ‘linear’. bound (BoundType, or list[BoundType], optional) – Boundary conditions. Defaults to ‘zero’. extrapolate (bool, optional) – Extrapolate out-of-bound data. Defaults to True. Splat weights (B, 1, Wo, Ho, Do). output (torch.Tensor)

grid_grad#

Sample an image with respect to a deformation field. interpolation can be an int, a string or an InterpolationType. Possible values are: A list of values can be provided, in the order [W, H, D], to specify dimension-specific interpolation orders. bound can be an int, a string or a BoundType. Possible values are: A list of values can be provided, in the order [W, H, D], to specify dimension-specific boundary conditions. sliding is a specific condition than only applies to flow fields (with as many channels as dimensions). It cannot be dimension-specific. Note that: dft corresponds to circular padding dct2 corresponds to Neumann boundary conditions (symmetric) dst2 corresponds to Dirichlet boundary conditions (antisymmetric) See also https://en.wikipedia.org/wiki/Discrete_cosine_transform https://en.wikipedia.org/wiki/Discrete_sine_transform help(monai._C.BoundType) help(monai._C.InterpolationType) input (Tensor) – Input image. (B, C, Wi, Hi, Di). grid (Tensor) – Deformation field. (B, Wo, Ho, Do, 2|3). interpolation (int or list[int] , optional) – Interpolation order. Defaults to ‘linear’. bound (BoundType, or list[BoundType], optional) – Boundary conditions. Defaults to ‘zero’. extrapolate (bool) – Extrapolate out-of-bound data. Defaults to True. Sampled gradients (B, C, Wo, Ho, Do, 1|2|3). output (torch.Tensor)

LLTM#

This recurrent unit is similar to an LSTM, but differs in that it lacks a forget gate and uses an Exponential Linear Unit (ELU) as its internal activation function. Because this unit never forgets, call it LLTM, or Long-Long-Term-Memory unit. It has both C++ and CUDA implementation, automatically switch according to the target device where put this module to. input_features (int) – size of input feature data state_size (int) – size of the state of recurrent unit Referring to: https://pytorch.org/tutorials/advanced/cpp_extension.html 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.

Utilities#

Utilities and types for defining networks, these depend on PyTorch. Utility to convert a model into TorchScript model and save to file, with optional input / output data verification. model (Module) – source PyTorch model to save. filename_or_obj (Optional[Any]) – if not None, specify a file-like object (has to implement write and flush) or a string containing a file path name to save the TorchScript model. extra_files (Optional[Dict]) – map from filename to contents which will be stored as part of the save model file. for more details: https://pytorch.org/docs/stable/generated/torch.jit.save.html. verify (bool) – whether to verify the input and output of TorchScript model. if filename_or_obj is not None, load

the saved TorchScript model and verify. inputs (Optional[Sequence[Any]]) – input test data to verify model, should be a sequence of data, every item maps to a argument of model() function. device (Optional[device]) – target device to verify the model, if None, use CUDA if available. rtol (float) – the relative tolerance when comparing the outputs of PyTorch model and TorchScript model. atol (float) – the absolute tolerance when comparing the outputs of PyTorch model and TorchScript model. kwargs – other arguments except obj for torch.jit.script() to convert model, for more details: https://pytorch.org/docs/master/generated/torch.jit.script.html. Compute a module state_dict, of which the keys are the same as dst. The values of dst are overwritten by the ones from src whenever their keys match. The method provides additional dst_prefix for the dst key when matching them. mapping can be a {“src_key”: “dst_key”} dict, indicating dst[dst_prefix + dst_key] = src[src_key]. This function is mainly to return a model state dict for loading the src model state into the dst model, src and dst can have different dict keys, but their corresponding values normally have the same shape. dst (Union[Module, Mapping]) – a pytorch module or state dict to be updated. src (Union[Module, Mapping]) – a pytorch module or state dist used to get the values used for the update. dst_prefix – dst key prefix, so that dst[dst_prefix + src_key] will be assigned to the value of src[src_key]. mapping – a {“src_key”: “dst_key”} dict, indicating that dst[dst_prefix + dst_key] to be assigned to the value of src[src_key]. exclude_vars – a regular expression to match the dst variable names, so that their values are not overwritten by src. inplace – whether to set the dst module with the updated state_dict via load_state_dict. This option is only available when dst is a torch.nn.Module. Examples Returns: an OrderedDict of the updated dst state, the changed, and unchanged keys. Set network(s) to eval mode and then return to original state at the end. nets (Module) – Input network(s) Examples Get the state dict of input object if has state_dict, otherwise, return object directly. For data parallel model, automatically convert it to regular model first. obj (Union[Module, Mapping]) – input object to check and get the state_dict. ICNR initialization for 2D/3D kernels adapted from Aitken et al.,2017 , “Checkerboard artifact free sub-pixel convolution”. get the named module in mod by the attribute name, for example look_up_named_module(net, "features.3.1.attn") name (str) – a string representing the module attribute. mod – a pytorch module to be searched (in mod.named_modules()). print_all_options – whether to print all named modules when name is not found in mod. Defaults to False. the corresponding pytorch module’s subcomponent such as net.features[3][1].attn Initialize the weight and bias tensors of m’ and its submodules to values from a normal distribution with a stddev of `std’. Weight tensors of convolution and linear modules are initialized with a mean of 0, batch norm modules with a mean of 1. The callable `normal_func’, used to assign values, should have the same arguments as its default normal_(). This can be used with `nn.Module.apply to visit submodules of a network. None Compute an affine matrix according to the input shape. The transform normalizes the homogeneous image coordinates to the range of [-1, 1]. Currently the following source coordinates are supported: align_corners=False, zero_centered=False, normalizing from [-0.5, d-0.5]. align_corners=True, zero_centered=False, normalizing from [0, d-1]. align_corners=False, zero_centered=True, normalizing from [-(d+1)/2, (d-1)/2]. align_corners=True, zero_centered=True, normalizing from [-(d-1)/2, (d-1)/2]. shape – input spatial shape, a sequence of integers. device (Optional[device]) – device on which the returned affine will be allocated. dtype (Optional[dtype]) – data type of the returned affine align_corners (bool) – if True, consider -1 and 1 to refer to the centers of the corner pixels rather than the image corners. See also: https://pytorch.org/docs/stable/nn.functional.html#torch.nn.functional.grid_sample zero_centered (bool) – whether the coordinates are normalized from a zero-centered range, default to False. Setting this flag and align_corners will jointly specify the

normalization source range. Tensor For every value v in labels, the value in the output will be either 1 or 0. Each vector along the dim-th dimension has the “one-hot” format, i.e., it has a total length of num_classes, with a one and num_class-1 zeros. Note that this will include the background label, thus a binary mask should be treated as having two classes. labels (Tensor) – input tensor of integers to be converted into the ‘one-hot’ format. Internally labels will be converted into integers labels.long(). num_classes (int) – number of output channels, the corresponding length of labels[dim] will be converted to num_classes from 1. dtype (dtype) – the data type of the output one_hot label. dim (int) – the dimension to be converted to num_classes channels from 1 channel, should be non-negative number. Example: For a tensor labels of dimensions [B]1[spatial_dims], return a tensor of dimensions [B]N[spatial_dims] when num_classes=N number of classes and dim=1. Tensor Apply pixel shuffle to the tensor x with spatial dimensions spatial_dims and scaling factor scale_factor. See: Shi et al., 2016, “Real-Time Single Image and Video Super-Resolution Using a nEfficient Sub-Pixel Convolutional Neural Network.” See: Aitken et al., 2017, “Checkerboard artifact free sub-pixel convolution”. x (Tensor) – Input tensor spatial_dims (int) – number of spatial dimensions, typically 2 or 3 for 2D or 3D scale_factor (int) – factor to rescale the spatial dimensions by, must be >=1 Tensor Reshuffled version of x. ValueError – When input channels of x are not divisible by (scale_factor ** spatial_dims) Given the logits from a network, computing the segmentation by thresholding all values above 0 if multi-labels task, computing the argmax along the channel axis if multi-classes task, logits has shape BCHW[D]. logits (Tensor) – raw data of model output. mutually_exclusive (bool) – if True, logits will be converted into a binary matrix using a combination of argmax, which is suitable for multi-classes task. Defaults to False. threshold (float) – thresholding the prediction values if multi-labels task. Any Replace sub-module(s) in a parent module. The name of the module to be replace can be nested e.g., features.denseblock1.denselayer1.layers.relu1. If this is the case (there are “.” in the module name), then this function will recursively call itself. parent (Module) – module that contains the module to be replaced name (str) – name of module to be replaced. Can include “.”. new_module (Module) – torch.nn.Module to be placed at position name inside parent. This will be deep copied if strict_match == False multiple instances are independent. strict_match (bool) – if True, module name must == name. If false then name in named_modules() will be used. True can be used to change just one module, whereas False can be used to replace all modules with similar name (e.g., relu). match_device (bool) – if True, the device of the new module will match the model. Requires all of parent to be on the same device. List[Tuple[str, Module]] List of tuples of replaced modules. Element 0 is module name, element 1 is the replaced module. AttributeError – if strict_match is True and name is not a named module in parent. Temporarily replace sub-module(s) in a parent module (context manager). See monai.networks.utils.replace_modules. Save the state dict of input source data with PyTorch save. It can save nn.Module, state_dict, a dictionary of nn.Module or state_dict. And automatically convert the data parallel module to regular module. For example: Refer to: https://pytorch.org/ignite/v0.4.8/generated/ignite.handlers.DiskSaver.html. src (Union[Module, Dict]) – input data to save, can be nn.Module, state_dict, a dictionary of nn.Module or state_dict. path (Union[str, PathLike]) – target file path to save the input object. kwargs – other args for the save_obj except for the obj and path. default func is torch.save(), details of the args: https://pytorch.org/docs/stable/generated/torch.save.html. look up name in mod and replace the layer with new_layer, return the updated mod. mod – a pytorch module to be updated. name (str) – a string representing the target module attribute. new_layer – a new module replacing the corresponding layer at mod.name. an updated mod See

also: monai.networks.utils.look_up_named_module(). Deprecated since version 0.8.0: Use monai.utils.misc.sample_slices instead. Tensor Given affine defined for coordinates in the pixel space, compute the corresponding affine for the normalized coordinates. affine (Tensor) – Nxdxd batched square matrix src_size (Sequence[int]) – source image spatial shape dst_size (Sequence[int]) – target image spatial shape align_corners (bool) – if True, consider -1 and 1 to refer to the centers of the corner pixels rather than the image corners. See also: https://pytorch.org/docs/stable/nn.functional.html#torch.nn.functional.grid_sample zero_centered (bool) – whether the coordinates are normalized from a zero-centered range, default to False. See also: monai.networks.utils.normalize_transform(). TypeError – When affine is not a torch.Tensor. ValueError – When affine is not Nxdxd. ValueError – When src_size or dst_size dimensions differ from affine. Tensor Set network(s) to train mode and then return to original state at the end. nets (Module) – Input network(s) Examples This script contains utility functions for developing new networks/blocks in PyTorch. Performs layer mean-std normalization for complex data. Normalization is done for each batch member along each part (part refers to real and imaginary parts), separately. x (Tensor) – input of shape (B,C,H,W) for 2D data or (B,C,H,W,D) for 3D data Tuple[Tensor, Tensor, Tensor] A tuple containing normalized output of shape (B,C,H,W) for 2D data or (B,C,H,W,D) for 3D data mean std normalized output of shape (B,C,H,W) for 2D data or (B,C,H,W,D) for 3D data mean std Pad input to feed into the network (torch script compatible) x (Tensor) – input of shape (B,C,H,W) for 2D data or (B,C,H,W,D) for 3D data k (int) – padding factor. each padded dimension will be divisible by k. Tuple[Tensor, Tuple[Tuple[int, int], Tuple[int, int], Tuple[int, int], int, int, int]] A tuple containing padded input pad sizes (in order to reverse padding if needed) padded input pad sizes (in order to reverse padding if needed) Example Returns floor and ceil of the input n (float) – input number floor(n) ceil(n) floor(n) ceil(n) A tuple containing De-pad network output to match its original shape x (Tensor) – input of shape (B,C,H,W) for 2D data or (B,C,H,W,D) for 3D data pad_sizes (Tuple[Tuple[int, int], Tuple[int, int], Tuple[int, int], int, int, int]) – padding values Tensor de-padded input Detaches batch and channel dimensions. x (Tensor) – input of shape (B*C,1,H,W,2) for 2D data or (B*C,1,H,W,D,2) for 3D data batch_size (int) – batch size Tensor output of shape (B,C,…) Swaps the complex dimension with the channel dimension so that the network output has 2 as its last dimension x (Tensor) – input of shape (B,C*2,H,W) for 2D data or (B,C*2,H,W,D) for 3D data Tensor output of shape (B,C,H,W,2) for 2D data or (B,C,H,W,D,2) for 3D data Combines batch and channel dimensions. x (Tensor) – input of shape (B,C,H,W,2) for 2D data or (B,C,H,W,D,2) for 3D data output of shape (B*C,1,…) batch size output of shape (B*C,1,…) batch size A tuple containing Swaps the complex dimension with the channel dimension so that the network treats real/imaginary parts as two separate channels. x (Tensor) – input of shape (B,C,H,W,2) for 2D data or (B,C,H,W,D,2) for 3D data Tensor output of shape (B,C*2,H,W) for 2D data or (B,C*2,H,W,D) for 3D data Expands an image to its corresponding coil images based on the given sens_maps. Let’s say there are C coils. This function multiples image img with each coil sensitivity map in sens_maps and stacks the resulting C coil images along the channel dimension which is reserved for coils. img (Tensor) – 2D image (B,1,H,W,2) with the last dimension being 2 (for real/imaginary parts). 3D data will have the shape (B,1,H,W,D,2). sens_maps (Tensor) – Sensitivity maps for combining coil images. The shape is (B,C,H,W,2) for 2D data or (B,C,H,W,D,2) for 3D data (C denotes the coil dimension). spatial_dims (int) – is 2 for 2D data and is 3 for 3D data Tensor Expansion of x to (B,C,H,W,2) for 2D data and (B,C,H,W,D,2) for 3D data. The output is transferredto the frequency domain to yield coil measurements. to the frequency domain to yield coil measurements. Reduces coil measurements to a corresponding image based on the given sens_maps. Let’s say there are C coil measurements

inside kspace, then this function multiplies the conjugate of each coil sensitivity map with the corresponding coil image. The result of this process will be C images. Summing those images together gives the resulting “reduced image.” kspace (Tensor) – 2D kspace (B,C,H,W,2) with the last dimension being 2 (for real/imaginary parts) and C denoting the coil dimension. 3D data will have the shape (B,C,H,W,D,2). sens_maps (Tensor) – sensitivity maps of the same shape as input x. spatial_dims (int) – is 2 for 2D data and is 3 for 3D data Tensor reduction of x to (B,1,H,W,2) for 2D data or (B,1,H,W,D,2) for 3D data. previous Loss functions next Metrics © Copyright MONAI Consortium.

AHNet#

AHNet based on Anisotropic Hybrid Network. Adapted from lsqshr’s official code. Except from the original network that supports 3D inputs, this implementation also supports 2D inputs. According to the tests for deconvolutions, using "transpose" rather than linear interpolations is faster. Therefore, this implementation sets "transpose" as the default upsampling method. To meet the requirements of the structure, the input size for each spatial dimension (except the last one) should be: divisible by 2 ** (psp_block_num + 3) and no less than 32 in transpose mode, and should be divisible by 32 and no less than 2 ** (psp_block_num + 3) in other upsample modes. In addition, the input size for the last spatial dimension should be divisible by 32, and at least one spatial size should be no less than 64. layers (tuple) – number of residual blocks for 4 layers of the network (layer1…layer4). Defaults to (3, 4, 6, 3). spatial_dims (int) – spatial dimension of the input data. Defaults to 3. in_channels (int) – number of input channels for the network. Default to 1. out_channels (int) – number of output channels for the network. Defaults to 1. psp_block_num (int) – the number of pyramid volumetric pooling modules used at the end of the network before the final output layer for extracting multiscale features. The number should be an integer that belongs to [0,4]. Defaults to 4. upsample_mode (str) – ["transpose", "bilinear", "trilinear", nearest] The mode of upsampling manipulations. Using the last two modes cannot guarantee the model’s reproducibility. Defaults to transpose. "transpose", uses transposed convolution layers. "bilinear", uses bilinear interpolate. "trilinear", uses trilinear interpolate. "nearest", uses nearest interpolate. ["transpose", "bilinear", "trilinear", nearest] The mode of upsampling manipulations. Using the last two modes cannot guarantee the model’s reproducibility. Defaults to transpose. "transpose", uses transposed convolution layers. "bilinear", uses bilinear interpolate. "trilinear", uses trilinear interpolate. "nearest", uses nearest interpolate. pretrained (bool) – whether to load pretrained weights from ResNet50 to initialize convolution layers, default to False. progress (bool) – If True, displays a progress bar of the download of pretrained weights to stderr. 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.

DenseNet#

Densenet based on: Densely Connected Convolutional Networks. Adapted from PyTorch Hub 2D version: https://pytorch.org/vision/stable/models.html#id16. This network is non-deterministic When spatial_dims is 3 and CUDA is enabled. Please check the link below for more details: https://pytorch.org/docs/stable/generated/torch.

use_deterministic_algorithms.html#torch.use_deterministic_algorithms spatial_dims (int) – number of spatial dimensions of the input image. in_channels (int) – number of the input channel. out_channels (int) – number of the output classes. init_features (int) – number of filters in the first convolution layer. growth_rate (int) – how many filters to add each layer (k in paper). block_config (Sequence[int]) – how many layers in each pooling block. bn_size (int) – multiplicative factor for number of bottle neck layers. (i.e. bn_size * k features in the bottleneck layer) act (Union[str, tuple]) – activation type and arguments. Defaults to relu. norm (Union[str, tuple]) – feature normalization type and arguments. Defaults to batch norm. dropout_prob (float) – dropout rate after each dense layer. 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. Tensor

DenseNet121#

DenseNet121 with optional pretrained support when spatial_dims is 2.

DenseNet169#

DenseNet169 with optional pretrained support when spatial_dims is 2.

DenseNet201#

DenseNet201 with optional pretrained support when spatial_dims is 2.

EfficientNet#

EfficientNet based on Rethinking Model Scaling for Convolutional Neural Networks. Adapted from EfficientNet-PyTorch. blocks_args_str (List[str]) – block definitions. spatial_dims (int) – number of spatial dimensions. in_channels (int) – number of input channels. num_classes (int) – number of output classes. width_coefficient (float) – width multiplier coefficient (w in paper). depth_coefficient (float) – depth multiplier coefficient (d in paper). dropout_rate (float) – dropout rate for dropout layers. image_size (int) – input image resolution. norm (Union[str, tuple]) – feature normalization type and arguments. drop_connect_rate (float) – dropconnect rate for drop connection (individual weights) layers. depth_divisor (int) – depth divisor for channel rounding. inputs (Tensor) – input should have spatially N dimensions `` (Batch, in_channels, dim_0[, dim_1, ..., dim_N]) – a torch Tensor of classification prediction in shape (Batch, num_classes). Sets swish function as memory efficient (for training) or standard (for JIT export). memory_efficient (bool) – whether to use memory-efficient version of swish. None

BlockArgs#

of arguments for MBConvBlock definition. Alias for field number 3 Get a BlockArgs object from a string notation of arguments. block_string (str) – A string notation of

arguments. Examples: “r1_k3_s11_e1_i32_o16_se0.25”. namedtuple defined at the top of this function. BlockArgs Alias for field number 6 Alias for field number 4 Alias for field number 1 Alias for field number 0 Alias for field number 5 Alias for field number 7 Alias for field number 2 Return a block string notation for current BlockArgs object A string notation of BlockArgs object arguments.Example:
“r1_k3_s11_e1_i32_o16_se0.25_noskip”. Example:
“r1_k3_s11_e1_i32_o16_se0.25_noskip”.

EfficientNetBN#

Generic wrapper around EfficientNet, used to initialize EfficientNet-B0 to EfficientNet-B7 models model_name is mandatory argument as there is no EfficientNetBN itself, it needs the N in [0, 1, 2, 3, 4, 5, 6, 7, 8] to be a model model_name (str) – name of model to initialize, can be from [efficientnet-b0, …, efficientnet-b8, efficientnet-l2]. pretrained (bool) – whether to initialize pretrained ImageNet weights, only available for spatial_dims=2 and batch norm is used. progress (bool) – whether to show download progress for pretrained weights download. spatial_dims (int) – number of spatial dimensions. in_channels (int) – number of input channels. num_classes (int) – number of output classes. norm (Union[str, tuple]) – feature normalization type and arguments. adv_prop (bool) – whether to use weights trained with adversarial examples. This argument only works when pretrained is True. Examples:

EfficientNetBNFeatures#

Initialize EfficientNet-B0 to EfficientNet-B7 models as a backbone, the backbone can be used as an encoder for segmentation and objection models. Compared with the class EfficientNetBN, the only different place is the forward function. This class refers to PyTorch image models. inputs (Tensor) – input should have spatially N dimensions `` (Batch, in_channels, dim_0[, dim_1, ..., dim_N]) – a list of torch Tensors.

SegResNet#

SegResNet based on 3D MRI brain tumor segmentation using autoencoder regularization. The module does not include the variational autoencoder (VAE). The model supports 2D or 3D inputs. spatial_dims (int) – spatial dimension of the input data. Defaults to 3. init_filters (int) – number of output channels for initial convolution layer. Defaults to 8. in_channels (int) – number of input channels for the network. Defaults to 1. out_channels (int) – number of output channels for the network. Defaults to 2. dropout_prob (Optional[float]) – probability of an element to be zero-ed. Defaults to None. act (Union[Tuple, str]) – activation type and arguments. Defaults to RELU. norm (Union[Tuple, str]) – feature normalization type and arguments. Defaults to GROUP. norm_name (str) – deprecating option for feature normalization type. num_groups (int) – deprecating option for group norm. parameters. use_conv_final (bool) – if add a final convolution block to output. Defaults to True. blocks_down (tuple) – number of down sample blocks in each layer. Defaults to [1,2,2,4]. blocks_up (tuple) – number of up sample blocks in each layer. Defaults to [1,1,1]. upsample_mode (Union[UpsampleMode, str]) – ["deconv", "nontrainable", "pixelshuffle"] The mode of upsampling manipulations. Using the nontrainable modes cannot guarantee the model’s reproducibility. Defaults to``nontrainable``. deconv,

uses transposed convolution layers. nontrainable, uses non-trainable linear interpolation. pixelshuffle, uses monai.networks.blocks.SubpixelUpsample. ["deconv", "nontrainable", "pixelshuffle"] The mode of upsampling manipulations. Using the nontrainable modes cannot guarantee the model’s reproducibility. Defaults to``nontrainable``. deconv, uses transposed convolution layers. nontrainable, uses non-trainable linear interpolation. pixelshuffle, uses monai.networks.blocks.SubpixelUpsample. 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. Tensor

SegResNetVAE#

SegResNetVAE based on 3D MRI brain tumor segmentation using autoencoder regularization. The module contains the variational autoencoder (VAE). The model supports 2D or 3D inputs. input_image_size (Sequence[int]) – the size of images to input into the network. It is used to determine the in_features of the fc layer in VAE. vae_estimate_std (bool) – whether to estimate the standard deviations in VAE. Defaults to False. vae_default_std (float) – if not to estimate the std, use the default value. Defaults to 0.3. vae_nz (int) – number of latent variables in VAE. Defaults to 256. Where, 128 to represent mean, and 128 to represent std. spatial_dims (int) – spatial dimension of the input data. Defaults to 3. init_filters (int) – number of output channels for initial convolution layer. Defaults to 8. in_channels (int) – number of input channels for the network. Defaults to 1. out_channels (int) – number of output channels for the network. Defaults to 2. dropout_prob (Optional[float]) – probability of an element to be zero-ed. Defaults to None. act (Union[str, tuple]) – activation type and arguments. Defaults to RELU. norm (Union[Tuple, str]) – feature normalization type and arguments. Defaults to GROUP. use_conv_final (bool) – if add a final convolution block to output. Defaults to True. blocks_down (tuple) – number of down sample blocks in each layer. Defaults to [1,2,2,4]. blocks_up (tuple) – number of up sample blocks in each layer. Defaults to [1,1,1]. upsample_mode (Union[UpsampleMode, str]) – ["deconv", "nontrainable", "pixelshuffle"] The mode of upsampling manipulations. Using the nontrainable modes cannot guarantee the model’s reproducibility. Defaults to``nontrainable``. deconv, uses transposed convolution layers. nontrainable, uses non-trainable linear interpolation. pixelshuffle, uses monai.networks.blocks.SubpixelUpsample. ["deconv", "nontrainable", "pixelshuffle"] The mode of upsampling manipulations. Using the nontrainable modes cannot guarantee the model’s reproducibility. Defaults to``nontrainable``. deconv, uses transposed convolution layers. nontrainable, uses non-trainable linear interpolation. pixelshuffle, uses monai.networks.blocks.SubpixelUpsample. 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.

ResNet#

ResNet based on: Deep Residual Learning for Image Recognition and Can Spatiotemporal 3D CNNs Retrace the History of 2D CNNs and ImageNet?. Adapted from kenshohara/3D-ResNets-PyTorch. block (Union[Type[Union[ResNetBlock,

ResNetBottleneck]], str]) – which ResNet block to use, either Basic or Bottleneck. ResNet block class or str. for Basic: ResNetBlock or ‘basic’ for Bottleneck: ResNetBottleneck or ‘bottleneck’ layers (List[int]) – how many layers to use. block_inplanes (List[int]) – determine the size of planes at each step. Also tunable with widen_factor. spatial_dims (int) – number of spatial dimensions of the input image. n_input_channels (int) – number of input channels for first convolutional layer. conv1_t_size (Union[Tuple[int], int]) – size of first convolution layer, determines kernel and padding. conv1_t_stride (Union[Tuple[int], int]) – stride of first convolution layer. no_max_pool (bool) – bool argument to determine if to use maxpool layer. shortcut_type (str) – which downsample block to use. Options are ‘A’, ‘B’, default to ‘B’. - ‘A’: using self._downsample_basic_block. - ‘B’: kernel_size 1 conv + norm. widen_factor (float) – widen output for each layer. num_classes (int) – number of output (classifications). feed_forward (bool) – whether to add the FC layer for the output, default to True. bias_downsample (bool) – whether to use bias term in the downsampling block when shortcut_type is ‘B’, default to True. 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. Tensor

SENet#

SENet based on Squeeze-and-Excitation Networks. Adapted from Cadene Hub 2D version. spatial_dims (int) – spatial dimension of the input data. in_channels (int) – channel number of the input data. block (Union[Type[Union[SEBottleneck, SEResNetBottleneck, SEResNeXtBottleneck]], str]) – SEBlock class or str. for SENet154: SEBottleneck or ‘se_bottleneck’ for SE-ResNet models: SEResNetBottleneck or ‘se_resnet_bottleneck’ for SE-ResNeXt models: SEResNeXtBottleneck or ‘se_resnetxt_bottleneck’ layers (Sequence[int]) – number of residual blocks for 4 layers of the network (layer1…layer4). groups (int) – number of groups for the 3x3 convolution in each bottleneck block. for SENet154: 64 for SE-ResNet models: 1 for SE-ResNeXt models: 32 reduction (int) – reduction ratio for Squeeze-and-Excitation modules. for all models: 16 dropout_prob (Optional[float]) – drop probability for the Dropout layer. if None the Dropout layer is not used. for SENet154: 0.2 for SE-ResNet models: None for SE-ResNeXt models: None dropout_dim (int) – determine the dimensions of dropout. Defaults to 1. When dropout_dim = 1, randomly zeroes some of the elements for each channel. When dropout_dim = 2, Randomly zeroes out entire channels (a channel is a 2D feature map). When dropout_dim = 3, Randomly zeroes out entire channels (a channel is a 3D feature map). inplanes (int) – number of input channels for layer1. for SENet154: 128 for SE-ResNet models: 64 for SE-ResNeXt models: 64 downsample_kernel_size (int) – kernel size for downsampling convolutions in layer2, layer3 and layer4. for SENet154: 3 for SE-ResNet models: 1 for SE-ResNeXt models: 1 input_3x3 (bool) – If True, use three 3x3 convolutions instead of a single 7x7 convolution in layer0. - For SENet154: True - For SE-ResNet models: False - For SE-ResNeXt models: False num_classes (int) – number of outputs in last_linear layer. for all models: 1000 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. Tensor

SENet154#

SENet154 based on Squeeze-and-Excitation Networks with optional pretrained support when spatial_dims is 2.

SEResNet50#

SEResNet50 based on Squeeze-and-Excitation Networks with optional pretrained support when spatial_dims is 2.

SEResNet101#

SEResNet101 based on Squeeze-and-Excitation Networks with optional pretrained support when spatial_dims is 2.

SEResNet152#

SEResNet152 based on Squeeze-and-Excitation Networks with optional pretrained support when spatial_dims is 2.

SEResNext50#

SEResNext50 based on Squeeze-and-Excitation Networks with optional pretrained support when spatial_dims is 2.

SEResNext101#

SEResNext101 based on Squeeze-and-Excitation Networks with optional pretrained support when spatial_dims is 2.

HighResNet#

Reimplementation of highres3dnet based on Li et al., “On the compactness, efficiency, and representation of 3D convolutional networks: Brain parcellation as a pretext task”, IPMI ‘17 Adapted from: NifTK/NiftyNet fepegar/highresnet spatial_dims (int) – number of spatial dimensions of the input image. in_channels (int) – number of input channels. out_channels (int) – number of output channels. norm_type (Union[str, tuple]) – feature normalization type and arguments. Defaults to ("batch", {"affine": True}). acti_type (Union[str, tuple]) – activation type and arguments. Defaults to ("relu", {"inplace": True}). dropout_prob (Union[Tuple, str, float, None]) – probability of the feature map to be zeroed (only applies to the penultimate conv layer). bias (bool) – whether to have a bias term in convolution blocks. Defaults to False. According to Performance Tuning Guide, if a conv layer is directly followed by a batch norm layer, bias should be False. whether to have a bias term in convolution blocks. Defaults to False. According to Performance Tuning Guide, if a conv layer is directly followed by a batch norm layer, bias should be False. layer_params (Sequence[Dict]) – specifying

key parameters of each layer/block. channel_matching (Union[ChannelMatching, str]) – {"pad", "project"} Specifies handling residual branch and conv branch channel mismatches. Defaults to "pad". "pad": with zero padding. "project": with a trainable conv with kernel size one. {"pad", "project"} Specifies handling residual branch and conv branch channel mismatches. Defaults to "pad". "pad": with zero padding. "project": with a trainable conv with kernel size one. 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. Tensor spatial_dims (int) – number of spatial dimensions of the input image. in_channels (int) – number of input channels. out_channels (int) – number of output channels. kernels (Sequence[int]) – each integer k in kernels corresponds to a convolution layer with kernel size k. dilation (Union[Sequence[int], int]) – spacing between kernel elements. norm_type (Union[Tuple, str]) – feature normalization type and arguments. Defaults to ("batch", {"affine": True}). acti_type (Union[Tuple, str]) – {"relu", "prelu", "relu6"} Non-linear activation using ReLU or PReLU. Defaults to "relu". bias (bool) – whether to have a bias term in convolution blocks. Defaults to False. According to Performance Tuning Guide, if a conv layer is directly followed by a batch norm layer, bias should be False. whether to have a bias term in convolution blocks. Defaults to False. According to Performance Tuning Guide, if a conv layer is directly followed by a batch norm layer, bias should be False. channel_matching (Union[ChannelMatching, str]) – {"pad", "project"} Specifies handling residual branch and conv branch channel mismatches. Defaults to "pad". "pad": with zero padding. "project": with a trainable conv with kernel size one. {"pad", "project"} Specifies handling residual branch and conv branch channel mismatches. Defaults to "pad". "pad": with zero padding. "project": with a trainable conv with kernel size one. ValueError – When channel_matching=pad and in_channels > out_channels. Incompatible values. 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. Tensor

DynUNet#

This reimplementation of a dynamic UNet (DynUNet) is based on: Automated Design of Deep Learning Methods for Biomedical Image Segmentation. nnU-Net: Self-adapting Framework for U-Net-Based Medical Image Segmentation. Optimized U-Net for Brain Tumor Segmentation. This model is more flexible compared with monai.networks.nets.UNet in three places: Residual connection is supported in conv blocks. Anisotropic kernel sizes and strides can be used in each layers. Deep supervision heads can be added. The model supports 2D or 3D inputs and is consisted with four kinds of blocks: one input block, n downsample blocks, one bottleneck and n+1 upsample blocks. Where, n>0. The first and last kernel and stride values of the input sequences are used for input block and bottleneck respectively, and the rest value(s) are used for downsample and upsample blocks. Therefore, pleasure ensure that the length of input sequences (kernel_size and strides) is no less than 3 in order to have at least one downsample and upsample blocks. To meet the requirements of the structure, the input size for each spatial dimension should be divisible by the product of all strides in the corresponding dimension. In addition, the minimal spatial size should have at least one dimension that has twice the size of the product of all strides. For example, if strides=((1, 2, 4), 2, 2, 1), the spatial size should

be divisible by (4, 8, 16), and the minimal spatial size is (8, 8, 16) or (4, 16, 16) or (4, 8, 32). The output size for each spatial dimension equals to the input size of the corresponding dimension divided by the stride in strides[0]. For example, if strides=((1, 2, 4), 2, 2, 1) and the input size is (64, 32, 32), the output size is (64, 16, 8). For backwards compatibility with old weights, please set strict=False when calling load_state_dict. Usage example with medical segmentation decathlon dataset is available at: Project-MONAI/tutorials. spatial_dims (int) – number of spatial dimensions. in_channels (int) – number of input channels. out_channels (int) – number of output channels. kernel_size (Sequence[Union[Sequence[int], int]]) – convolution kernel size. strides (Sequence[Union[Sequence[int], int]]) – convolution strides for each blocks. upsample_kernel_size (Sequence[Union[Sequence[int], int]]) – convolution kernel size for transposed convolution layers. The values should equal to strides[1:]. filters (Optional[Sequence[int]]) – number of output channels for each blocks. Different from nnU-Net, in this implementation we add this argument to make the network more flexible. As shown in the third reference, one way to determine this argument is like: [64, 96, 128, 192, 256, 384, 512, 768, 1024][: len(strides)]. The above way is used in the network that wins task 1 in the BraTS21 Challenge. If not specified, the way which nnUNet used will be employed. Defaults to None. dropout (Union[Tuple, str, float, None]) – dropout ratio. Defaults to no dropout. norm_name (Union[Tuple, str]) – feature normalization type and arguments. Defaults to INSTANCE. INSTANCE_NVFUSER is a faster version of the instance norm layer, it can be used when: 1) spatial_dims=3, 2) CUDA device is available, 3) apex is installed and 4) non-Windows OS is used. act_name (Union[Tuple, str]) – activation layer type and arguments. Defaults to leakyrelu. deep_supervision (bool) – whether to add deep supervision head before output. Defaults to False. If True, in training mode, the forward function will output not only the final feature map (from output_block), but also the feature maps that come from the intermediate up sample layers. In order to unify the return type (the restriction of TorchScript), all intermediate feature maps are interpolated into the same size as the final feature map and stacked together (with a new dimension in the first axis)into one single tensor. For instance, if there are two intermediate feature maps with shapes: (1, 2, 16, 12) and (1, 2, 8, 6), and the final feature map has the shape (1, 2, 32, 24), then all intermediate feature maps will be interpolated into (1, 2, 32, 24), and the stacked tensor will has the shape (1, 3, 2, 32, 24). When calculating the loss, you can use torch.unbind to get all feature maps can compute the loss one by one with the ground truth, then do a weighted average for all losses to achieve the final loss. deep_supr_num (int) – number of feature maps that will output during deep supervision head. The value should be larger than 0 and less than the number of up sample layers. Defaults to 1. res_block (bool) – whether to use residual connection based convolution blocks during the network. Defaults to False. trans_bias (bool) – whether to set the bias parameter in transposed convolution layers. Defaults to False. 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. alias of DynUNet alias of DynUNet

UNet#

Enhanced version of UNet which has residual units implemented with the ResidualUnit class. The residual part uses a convolution to change the input dimensions to match the output dimensions if this is necessary but will use nn.Identity if not. Refer to: https://link.springer.com/chapter/10.1007/978-3-030-12029-0_40.

Each layer of the network has a encode and decode path with a skip connection between them. Data in the encode path is downsampled using strided convolutions (if strides is given values greater than 1) and in the decode path upsampled using strided transpose convolutions. These down or up sampling operations occur at the beginning of each block rather than afterwards as is typical in UNet implementations. To further explain this consider the first example network given below. This network has 3 layers with strides of 2 for each of the middle layers (the last layer is the bottom connection which does not down/up sample). Input data to this network is immediately reduced in the spatial dimensions by a factor of 2 by the first convolution of the residual unit defining the first layer of the encode part. The last layer of the decode part will upsample its input (data from the previous layer concatenated with data from the skip connection) in the first convolution. this ensures the final output of the network has the same shape as the input. Padding values for the convolutions are chosen to ensure output sizes are even divisors/multiples of the input sizes if the strides value for a layer is a factor of the input sizes. A typical case is to use strides values of 2 and inputs that are multiples of powers of 2. An input can thus be downsampled evenly however many times its dimensions can be divided by 2, so for the example network inputs would have to have dimensions that are multiples of 4. In the second example network given below the input to the bottom layer will have shape (1, 64, 15, 15) for an input of shape (1, 1, 240, 240) demonstrating the input being reduced in size spatially by 2**4. spatial_dims (int) – number of spatial dimensions. in_channels (int) – number of input channels. out_channels (int) – number of output channels. channels (Sequence[int]) – sequence of channels. Top block first. The length of channels should be no less than 2. strides (Sequence[int]) – sequence of convolution strides. The length of stride should equal to len(channels) - 1. kernel_size (Union[Sequence[int], int]) – convolution kernel size, the value(s) should be odd. If sequence, its length should equal to dimensions. Defaults to 3. up_kernel_size (Union[Sequence[int], int]) – upsampling convolution kernel size, the value(s) should be odd. If sequence, its length should equal to dimensions. Defaults to 3. num_res_units (int) – number of residual units. Defaults to 0. act (Union[Tuple, str]) – activation type and arguments. Defaults to PReLU. norm (Union[Tuple, str]) – feature normalization type and arguments. Defaults to instance norm. dropout (float) – dropout ratio. Defaults to no dropout. bias (bool) – whether to have a bias term in convolution blocks. Defaults to True. According to Performance Tuning Guide, if a conv layer is directly followed by a batch norm layer, bias should be False. whether to have a bias term in convolution blocks. Defaults to True. According to Performance Tuning Guide, if a conv layer is directly followed by a batch norm layer, bias should be False. adn_ordering (str) – a string representing the ordering of activation (A), normalization (N), and dropout (D). Defaults to “NDA”. See also: monai.networks.blocks.ADN. Examples: Deprecated since version 0.6.0: dimensions is deprecated, use spatial_dims instead. to set appropriate spatial size, please check the tutorial for more details: Project-MONAI/tutorials. Typically, when using a stride of 2 in down / up sampling, the output dimensions are either half of the input when downsampling, or twice when upsampling. In this case with N numbers of layers in the network, the inputs must have spatial dimensions that are all multiples of 2^N. Usually, applying resize, pad or crop transforms can help adjust the spatial size of input data. 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. Tensor alias of UNet alias of

AttentionUnet#

Attention Unet based on Otkay et al. “Attention U-Net: Learning Where to Look for the Pancreas” https://arxiv.org/abs/1804.03999 spatial_dims (int) – number of spatial dimensions of the input image. in_channels (int) – number of the input channel. out_channels (int) – number of the output classes. channels (Sequence[int]) – sequence of channels. Top block first. The length of channels should be no less than 2. strides (Sequence[int]) – stride to use for convolutions. kernel_size (Union[Sequence[int], int]) – convolution kernel size. up_kernel_size (Union[Sequence[int], int]) – convolution kernel size for transposed convolution layers. dropout (float) – dropout ratio. Defaults to no dropout. 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. Tensor

UNETR#

UNETR based on: “Hatamizadeh et al., UNETR: Transformers for 3D Medical Image Segmentation ” in_channels (int) – dimension of input channels. out_channels (int) – dimension of output channels. img_size (Union[Sequence[int], int]) – dimension of input image. feature_size (int) – dimension of network feature size. hidden_size (int) – dimension of hidden layer. mlp_dim (int) – dimension of feedforward layer. num_heads (int) – number of attention heads. pos_embed (str) – position embedding layer type. norm_name (Union[Tuple, str]) – feature normalization type and arguments. conv_block (bool) – bool argument to determine if convolutional block is used. res_block (bool) – bool argument to determine if residual block is used. dropout_rate (float) – faction of the input units to drop. spatial_dims (int) – number of spatial dims. qkv_bias (bool) – apply the bias term for the qkv linear layer in self attention block Examples: 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.

SwinUNETR#

Swin UNETR based on: “Hatamizadeh et al., Swin UNETR: Swin Transformers for Semantic Segmentation of Brain Tumors in MRI Images ” img_size (Union[Sequence[int], int]) – dimension of input image. in_channels (int) – dimension of input channels. out_channels (int) – dimension of output channels. feature_size (int) – dimension of network feature size. depths (Sequence[int]) – number of layers in each stage. num_heads (Sequence[int]) – number of attention heads. norm_name (Union[Tuple, str]) – feature normalization type and arguments. drop_rate (float) – dropout rate. attn_drop_rate (float) – attention dropout rate. dropout_path_rate (float) – drop path rate. normalize (bool) – normalize output intermediate features in each stage. use_checkpoint (bool) – use gradient checkpointing for reduced memory usage. spatial_dims (int) – number of spatial dims. downsample – module used for downsampling, available options are “mergingv2”, “merging” and a user-specified nn.Module following the API defined in monai.networks.nets.PatchMerging. The default is currently “merging” (the original version defined in v0.9.0). Examples:

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.

BasicUNet#

A UNet implementation with 1D/2D/3D supports. Based on: Falk et al. “U-Net – Deep Learning for Cell Counting, Detection, and Morphometry”. Nature Methods 16, 67–70 (2019), DOI: http://dx.doi.org/10.1038/s41592-018-0261-2 spatial_dims (int) – number of spatial dimensions. Defaults to 3 for spatial 3D inputs. in_channels (int) – number of input channels. Defaults to 1. out_channels (int) – number of output channels. Defaults to 2. features (Sequence[int]) – six integers as numbers of features. Defaults to (32, 32, 64, 128, 256, 32), the first five values correspond to the five-level encoder feature sizes. the last value corresponds to the feature size after the last upsampling. six integers as numbers of features. Defaults to (32, 32, 64, 128, 256, 32), the first five values correspond to the five-level encoder feature sizes. the last value corresponds to the feature size after the last upsampling. act (Union[str, tuple]) – activation type and arguments. Defaults to LeakyReLU. norm (Union[str, tuple]) – feature normalization type and arguments. Defaults to instance norm. bias (bool) – whether to have a bias term in convolution blocks. Defaults to True. According to Performance Tuning Guide, if a conv layer is directly followed by a batch norm layer, bias should be False. whether to have a bias term in convolution blocks. Defaults to True. According to Performance Tuning Guide, if a conv layer is directly followed by a batch norm layer, bias should be False. dropout (Union[float, tuple]) – dropout ratio. Defaults to no dropout. upsample (str) – upsampling mode, available options are "deconv", "pixelshuffle", "nontrainable". Deprecated since version 0.6.0: dimensions is deprecated, use spatial_dims instead. Examples: See Also monai.networks.nets.DynUNet monai.networks.nets.UNet x (Tensor) – input should have spatially N dimensions (Batch, in_channels, dim_0[, dim_1, ..., dim_N-1]), N is defined by spatial_dims. It is recommended to have dim_n % 16 == 0 to ensure all maxpooling inputs have even edge lengths. A torch Tensor of “raw” predictions in shape (Batch, out_channels, dim_0[, dim_1, ..., dim_N-1]). alias of BasicUNet alias of BasicUNet

BasicUNetPlusPlus#

A UNet++ implementation with 1D/2D/3D supports. Based on: Zhou et al. “UNet++: A Nested U-Net Architecture for Medical Image Segmentation”. 4th Deep Learning in Medical Image Analysis (DLMIA) Workshop, DOI: https://doi.org/10.48550/arXiv.1807.10165 spatial_dims (int) – number of spatial dimensions. Defaults to 3 for spatial 3D inputs. in_channels (int) – number of input channels. Defaults to 1. out_channels (int) – number of output channels. Defaults to 2. features (Sequence[int]) – six integers as numbers of features. Defaults to (32, 32, 64, 128, 256, 32), the first five values correspond to the five-level encoder feature sizes. the last value corresponds to the feature size after the last upsampling. six integers as numbers of features. Defaults to (32, 32, 64, 128, 256, 32), the first five values correspond to the five-level encoder feature sizes. the last value corresponds to the feature size after the last upsampling. deep_supervision (bool) – whether to prune the network at inference time. Defaults to False. If true, returns a list, whose elements

correspond to outputs at different nodes. act (Union[str, tuple]) – activation type and arguments. Defaults to LeakyReLU. norm (Union[str, tuple]) – feature normalization type and arguments. Defaults to instance norm. bias (bool) – whether to have a bias term in convolution blocks. Defaults to True. According to Performance Tuning Guide, if a conv layer is directly followed by a batch norm layer, bias should be False. whether to have a bias term in convolution blocks. Defaults to True. According to Performance Tuning Guide, if a conv layer is directly followed by a batch norm layer, bias should be False. dropout (Union[float, tuple]) – dropout ratio. Defaults to no dropout. upsample (str) – upsampling mode, available options are "deconv", "pixelshuffle", "nontrainable". Examples: monai.networks.nets.BasicUNet monai.networks.nets.DynUNet monai.networks.nets.UNet x (Tensor) – input should have spatially N dimensions (Batch, in_channels, dim_0[, dim_1, ..., dim_N-1]), N is defined by dimensions. It is recommended to have dim_n % 16 == 0 to ensure all maxpooling inputs have even edge lengths. A torch Tensor of “raw” predictions in shape (Batch, out_channels, dim_0[, dim_1, ..., dim_N-1]). alias of BasicUNetPlusPlus alias of BasicUNetPlusPlus

FlexibleUNet#

A flexible implementation of UNet-like encoder-decoder architecture. A flexible implement of UNet, in which the backbone/encoder can be replaced with any efficient network. Currently the input must have a 2 or 3 spatial dimension and the spatial size of each dimension must be a multiple of 32 if is_pad parameter is False. Please notice each output of backbone must be 2x downsample in spatial dimension of last output. For example, if given a 512x256 2D image and a backbone with 4 outputs. Spatial size of each encoder output should be 256x128, 128x64, 64x32 and 32x16. in_channels (int) – number of input channels. out_channels (int) – number of output channels. backbone (str) – name of backbones to initialize, only support efficientnet right now, can be from [efficientnet-b0,…, efficientnet-b8, efficientnet-l2]. pretrained (bool) – whether to initialize pretrained ImageNet weights, only available for spatial_dims=2 and batch norm is used, default to False. decoder_channels (Tuple) – number of output channels for all feature maps in decoder. len(decoder_channels) should equal to len(encoder_channels) - 1,default to (256, 128, 64, 32, 16). spatial_dims (int) – number of spatial dimensions, default to 2. norm (Union[str, tuple]) – normalization type and arguments, default to (“batch”, {“eps”: 1e-3, “momentum”: 0.1}). act (Union[str, tuple]) – activation type and arguments, default to (“relu”, {“inplace”: True}). dropout (Union[float, tuple]) – dropout ratio, default to 0.0. decoder_bias (bool) – whether to have a bias term in decoder’s convolution blocks. upsample (str) – upsampling mode, available options are``”deconv”, ``"pixelshuffle", "nontrainable". interp_mode (str) – {"nearest", "linear", "bilinear", "bicubic", "trilinear"} Only used in the “nontrainable” mode. is_pad (bool) – whether to pad upsampling features to fit features from encoder. Default to True. If this parameter is set to “True”, the spatial dim of network input can be arbitrary size, which is not supported by TensorRT. Otherwise, it must be a multiple of 32. Do a typical encoder-decoder-header inference. inputs (Tensor) – input should have spatially N dimensions (Batch, in_channels, dim_0[, dim_1, ..., dim_N]), N is defined by dimensions. A torch Tensor of “raw” predictions in shape (Batch, out_channels, dim_0[, dim_1, ..., dim_N]).

VNet#

V-Net based on Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation. Adapted from the official Caffe implementation. and another pytorch implementation. The model supports 2D or 3D inputs. spatial_dims (int) – spatial dimension of the input data. Defaults to 3. in_channels (int) – number of input channels for the network. Defaults to 1. The value should meet the condition that 16 % in_channels == 0. out_channels (int) – number of output channels for the network. Defaults to 1. act (Union[Tuple[str, Dict], str]) – activation type in the network. Defaults to ("elu", {"inplace": True}). dropout_prob (float) – dropout ratio. Defaults to 0.5. dropout_dim (int) – determine the dimensions of dropout. Defaults to 3. dropout_dim = 1, randomly zeroes some of the elements for each channel. dropout_dim = 2, Randomly zeroes out entire channels (a channel is a 2D feature map). dropout_dim = 3, Randomly zeroes out entire channels (a channel is a 3D feature map). determine the dimensions of dropout. Defaults to 3. dropout_dim = 1, randomly zeroes some of the elements for each channel. dropout_dim = 2, Randomly zeroes out entire channels (a channel is a 2D feature map). dropout_dim = 3, Randomly zeroes out entire channels (a channel is a 3D feature map). bias (bool) – whether to have a bias term in convolution blocks. Defaults to False. According to Performance Tuning Guide, if a conv layer is directly followed by a batch norm layer, bias should be False. whether to have a bias term in convolution blocks. Defaults to False. According to Performance Tuning Guide, if a conv layer is directly followed by a batch norm layer, bias should be False. 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.

RegUNet#

Class that implements an adapted UNet. This class also serve as the parent class of LocalNet and GlobalNet O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,”, Lecture Notes in Computer Science, 2015, vol. 9351, pp. 234–241. https://arxiv.org/abs/1505.04597 DeepReg (DeepRegNet/DeepReg) spatial_dims (int) – number of spatial dims in_channels (int) – number of input channels num_channel_initial (int) – number of initial channels depth (int) – input is at level 0, bottom is at level depth. out_kernel_initializer (Optional[str]) – kernel initializer for the last layer out_activation (Optional[str]) – activation at the last layer out_channels (int) – number of channels for the output extract_levels (Optional[Tuple[int]]) – list, which levels from net to extract. The maximum level must equal to depth pooling (bool) – for down-sampling, use non-parameterized pooling if true, otherwise use conv3d concat_skip (bool) – when up-sampling, concatenate skipped tensor if true, otherwise use addition encode_kernel_sizes (Union[int, List[int]]) – kernel size for down-sampling x – Tensor in shape (batch, in_channels, insize_1, insize_2, [insize_3]) Tensor in shape (batch, out_channels, insize_1, insize_2, [insize_3]), with the same spatial size as x

GlobalNet#

Build GlobalNet for image registration. Hu, Yipeng, et al. “Label-driven weakly-supervised learning for multimodal deformable image registration,” https://arxiv.org/abs/1711.01666

LocalNet#

Reimplementation of LocalNet, based on: Weakly-supervised convolutional neural networks for multimodal image registration. Label-driven weakly-supervised learning for multimodal deformable image registration. DeepReg (DeepRegNet/DeepReg) spatial_dims (int) – number of spatial dims in_channels (int) – number of input channels num_channel_initial (int) – number of initial channels out_kernel_initializer (Optional[str]) – kernel initializer for the last layer out_activation (Optional[str]) – activation at the last layer out_channels (int) – number of channels for the output extract_levels (Tuple[int]) – list, which levels from net to extract. The maximum level must equal to depth pooling (bool) – for down-sampling, use non-parameterized pooling if true, otherwise use conv3d use_addictive_sampling (bool) – whether use additive up-sampling layer for decoding. concat_skip (bool) – when up-sampling, concatenate skipped tensor if true, otherwise use addition

AutoEncoder#

Simple definition of an autoencoder and base class for the architecture implementing monai.networks.nets.VarAutoEncoder. The network is composed of an encode sequence of blocks, followed by an intermediary sequence of blocks, and finally a decode sequence of blocks. The encode and decode blocks are default monai.networks.blocks.Convolution instances with the encode blocks having the given stride and the decode blocks having transpose convolutions with the same stride. If num_res_units is given residual blocks are used instead. By default the intermediary sequence is empty but if inter_channels is given to specify the output channels of blocks then this will be become a sequence of Convolution blocks or of residual blocks if num_inter_units is given. The optional parameter inter_dilations can be used to specify the dilation values of the convolutions in these blocks, this allows a network to use dilated kernels in this middle section. Since the intermediary section isn’t meant to change the size of the output the strides for all these kernels is 1. spatial_dims (int) – number of spatial dimensions. in_channels (int) – number of input channels. out_channels (int) – number of output channels. channels (Sequence[int]) – sequence of channels. Top block first. The length of channels should be no less than 2. strides (Sequence[int]) – sequence of convolution strides. The length of stride should equal to len(channels) - 1. kernel_size (Union[Sequence[int], int]) – convolution kernel size, the value(s) should be odd. If sequence, its length should equal to dimensions. Defaults to 3. up_kernel_size (Union[Sequence[int], int]) – upsampling convolution kernel size, the value(s) should be odd. If sequence, its length should equal to dimensions. Defaults to 3. num_res_units (int) – number of residual units. Defaults to 0. inter_channels (Optional[list]) – sequence of channels defining the blocks in the intermediate layer between encode and decode. inter_dilations (Optional[list]) – defines the dilation value for each block of the intermediate layer. Defaults to 1. num_inter_units (int) – number of residual units for each block of the intermediate layer. Defaults to 0. act (Union[Tuple, str, None]) – activation type and arguments. Defaults to PReLU. norm (Union[Tuple, str]) – feature normalization type and arguments. Defaults to instance norm. dropout (Union[Tuple, str, float, None]) – dropout ratio. Defaults to no dropout. bias (bool) – whether to have a bias term in convolution blocks. Defaults to True. According to Performance Tuning Guide, if a conv layer is directly followed by a batch norm layer, bias should be False. whether to have a bias term in convolution blocks. Defaults to True. According to Performance Tuning Guide, if a conv layer is directly followed by a batch norm layer, bias should be

False. Examples: 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. Any

VarAutoEncoder#

Variational Autoencoder based on the paper - https://arxiv.org/abs/1312.6114 spatial_dims (int) – number of spatial dimensions. in_shape (Sequence[int]) – shape of input data starting with channel dimension. out_channels (int) – number of output channels. latent_size (int) – size of the latent variable. channels (Sequence[int]) – sequence of channels. Top block first. The length of channels should be no less than 2. strides (Sequence[int]) – sequence of convolution strides. The length of stride should equal to len(channels) - 1. kernel_size (Union[Sequence[int], int]) – convolution kernel size, the value(s) should be odd. If sequence, its length should equal to dimensions. Defaults to 3. up_kernel_size (Union[Sequence[int], int]) – upsampling convolution kernel size, the value(s) should be odd. If sequence, its length should equal to dimensions. Defaults to 3. num_res_units (int) – number of residual units. Defaults to 0. inter_channels (Optional[list]) – sequence of channels defining the blocks in the intermediate layer between encode and decode. inter_dilations (Optional[list]) – defines the dilation value for each block of the intermediate layer. Defaults to 1. num_inter_units (int) – number of residual units for each block of the intermediate layer. Defaults to 0. act (Union[Tuple, str, None]) – activation type and arguments. Defaults to PReLU. norm (Union[Tuple, str]) – feature normalization type and arguments. Defaults to instance norm. dropout (Union[Tuple, str, float, None]) – dropout ratio. Defaults to no dropout. bias (bool) – whether to have a bias term in convolution blocks. Defaults to True. According to Performance Tuning Guide, if a conv layer is directly followed by a batch norm layer, bias should be False. whether to have a bias term in convolution blocks. Defaults to True. According to Performance Tuning Guide, if a conv layer is directly followed by a batch norm layer, bias should be False. use_sigmoid (bool) – whether to use the sigmoid function on final output. Defaults to True. Examples: See also Variational autoencoder network with MedNIST Dataset Project-MONAI/tutorials 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. Tuple[Tensor, Tensor, Tensor, Tensor]

ViT#

Vision Transformer (ViT), based on: “Dosovitskiy et al., An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale ” ViT supports Torchscript but only works for Pytorch after 1.8. in_channels (int) – dimension of input channels. img_size (Union[Sequence[int], int]) – dimension of input image. patch_size (Union[Sequence[int], int]) – dimension of patch size. hidden_size (int) – dimension of hidden layer. mlp_dim (int) – dimension of feedforward layer. num_layers (int) – number of transformer blocks. num_heads (int) – number of attention heads. pos_embed (str) – position embedding layer type. classification (bool) – bool argument to determine if classification is used. num_classes (int) – number of classes if classification is used. dropout_rate (float) – faction of the input units to drop.

spatial_dims (int) – number of spatial dimensions. post_activation – add a final acivation function to the classification head when classification is True. Default to “Tanh” for nn.Tanh(). Set to other values to remove this function. qkv_bias (bool) – apply bias to the qkv linear layer in self attention block Examples: 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.

ViTAutoEnc#

Vision Transformer (ViT), based on: “Dosovitskiy et al., An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale ” Modified to also give same dimension outputs as the input size of the image in_channels (int) – dimension of input channels or the number of channels for input img_size (Union[Sequence[int], int]) – dimension of input image. patch_size (Union[Sequence[int], int]) – dimension of patch size. hidden_size (int) – dimension of hidden layer. out_channels (int) – number of output channels. deconv_chns (int) – number of channels for the deconvolution layers. mlp_dim (int) – dimension of feedforward layer. num_layers (int) – number of transformer blocks. num_heads (int) – number of attention heads. pos_embed (str) – position embedding layer type. dropout_rate (float) – faction of the input units to drop. spatial_dims (int) – number of spatial dimensions. Examples: x – input tensor must have isotropic spatial dimensions, such as [batch_size, channels, sp_size, sp_size[, sp_size]].

FullyConnectedNet#

Simple full-connected layer neural network composed of a sequence of linear layers with PReLU activation and dropout. The network accepts input with in_channels channels, has output with out_channels channels, and hidden layer output channels given in hidden_channels. If bias is True then linear units have a bias term. in_channels (int) – number of input channels. out_channels (int) – number of output channels. hidden_channels (Sequence[int]) – number of output channels for each hidden layer. dropout (Union[Tuple, str, float, None]) – dropout ratio. Defaults to no dropout. act (Union[Tuple, str, None]) – activation type and arguments. Defaults to PReLU. bias (bool) – whether to have a bias term in linear units. Defaults to True. adn_ordering (Optional[str]) – order of operations in monai.networks.blocks.ADN. Examples: Defines a network accept input with in_channels channels, output of out_channels channels, and hidden layers with channels given in hidden_channels. If bias is True then linear units have a bias term.

VarFullyConnectedNet#

Variational fully-connected network. This is composed of an encode layer, reparameterization layer, and then a decode layer. in_channels (int) – number of input channels. out_channels (int) – number of output channels. latent_size (int) – number of latent variables to use. encode_channels (Sequence[int]) – number of output channels for each hidden layer of the encode half. decode_channels (Sequence[int]) – number of output channels for each hidden layer of the decode half. dropout (Union[Tuple, str, float, None]) – dropout ratio. Defaults to no dropout. act

(Union[Tuple, str, None]) – activation type and arguments. Defaults to PReLU. bias (bool) – whether to have a bias term in linear units. Defaults to True. adn_ordering (Optional[str]) – order of operations in monai.networks.blocks.ADN. Examples: 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. Tuple[Tensor, Tensor, Tensor, Tensor]

Generator#

Defines a simple generator network accepting a latent vector and through a sequence of convolution layers constructs an output tensor of greater size and high dimensionality. The method _get_layer is used to create each of these layers, override this method to define layers beyond the default monai.networks.blocks.Convolution or monai.networks.blocks.ResidualUnit layers. The layers are constructed using the values in the channels and strides arguments, the number being defined by the length of these (which must match). Input is first passed through a torch.nn.Linear layer to convert the input vector to an image tensor with dimensions start_shape. This passes through the convolution layers and is progressively upsampled if the strides values are greater than 1 using transpose convolutions. The size of the final output is defined by the start_shape dimension and the amount of upsampling done through strides. In the default definition the size of the output's spatial dimensions will be that of start_shape multiplied by the product of strides, thus the example network below upsamples an starting size of (64, 8, 8) to (1, 64, 64) since its strides are (2, 2, 2). latent_shape (Sequence[int]) – tuple of integers stating the dimension of the input latent vector (minus batch dimension) start_shape (Sequence[int]) – tuple of integers stating the dimension of the tensor to pass to convolution subnetwork channels (Sequence[int]) – tuple of integers stating the output channels of each convolutional layer strides (Sequence[int]) – tuple of integers stating the stride (upscale factor) of each convolutional layer kernel_size (Union[Sequence[int], int]) – integer or tuple of integers stating size of convolutional kernels num_res_units (int) – integer stating number of convolutions in residual units, 0 means no residual units act – name or type defining activation layers norm – name or type defining normalization layers dropout (Optional[float]) – optional float value in range [0, 1] stating dropout probability for layers, None for no dropout bias (bool) – boolean stating if convolution layers should have a bias component Examples: 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. Tensor

Regressor#

This defines a network for relating large-sized input tensors to small output tensors, ie. regressing large values to a prediction. An output of a single dimension can be used as value regression or multi-label classification prediction, an output of a single value can be used as a discriminator or critic prediction. The network is constructed as a sequence of layers, either monai.networks.blocks.Convolution or monai.networks.blocks.ResidualUnit, with a final fully-connected layer resizing the

output from the blocks to the final size. Each block is defined with a stride value typically used to downsample the input using strided convolutions. In this way each block progressively condenses information from the input into a deep representation the final fully-connected layer relates to a final result. in_shape (Sequence[int]) – tuple of integers stating the dimension of the input tensor (minus batch dimension) out_shape (Sequence[int]) – tuple of integers stating the dimension of the final output tensor (minus batch dimension) channels (Sequence[int]) – tuple of integers stating the output channels of each convolutional layer strides (Sequence[int]) – tuple of integers stating the stride (downscale factor) of each convolutional layer kernel_size (Union[Sequence[int], int]) – integer or tuple of integers stating size of convolutional kernels num_res_units (int) – integer stating number of convolutions in residual units, 0 means no residual units act – name or type defining activation layers norm – name or type defining normalization layers dropout (Optional[float]) – optional float value in range [0, 1] stating dropout probability for layers, None for no dropout bias (bool) – boolean stating if convolution layers should have a bias component Examples: 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. Tensor

Classifier#

Defines a classification network from Regressor by specifying the output shape as a single dimensional tensor with size equal to the number of classes to predict. The final activation function can also be specified, eg. softmax or sigmoid. in_shape (Sequence[int]) – tuple of integers stating the dimension of the input tensor (minus batch dimension) classes (int) – integer stating the dimension of the final output tensor channels (Sequence[int]) – tuple of integers stating the output channels of each convolutional layer strides (Sequence[int]) – tuple of integers stating the stride (downscale factor) of each convolutional layer kernel_size (Union[Sequence[int], int]) – integer or tuple of integers stating size of convolutional kernels num_res_units (int) – integer stating number of convolutions in residual units, 0 means no residual units act – name or type defining activation layers norm – name or type defining normalization layers dropout (Optional[float]) – optional float value in range [0, 1] stating dropout probability for layers, None for no dropout bias (bool) – boolean stating if convolution layers should have a bias component last_act (Optional[str]) – name defining the last activation layer

Discriminator#

Defines a discriminator network from Classifier with a single output value and sigmoid activation by default. This is meant for use with GANs or other applications requiring a generic discriminator network. in_shape (Sequence[int]) – tuple of integers stating the dimension of the input tensor (minus batch dimension) channels (Sequence[int]) – tuple of integers stating the output channels of each convolutional layer strides (Sequence[int]) – tuple of integers stating the stride (downscale factor) of each convolutional layer kernel_size (Union[Sequence[int], int]) – integer or tuple of integers stating size of convolutional kernels num_res_units (int) – integer stating number of convolutions in residual units, 0 means no residual units act – name or type defining activation layers norm – name or type defining normalization layers dropout

(Optional[float]) – optional float value in range [0, 1] stating dropout probability for layers, None for no dropout bias (bool) – boolean stating if convolution layers should have a bias component last_act – name defining the last activation layer

Critic#

Defines a critic network from Classifier with a single output value and no final activation. The final layer is nn.Flatten instead of nn.Linear, the final result is computed as the mean over the first dimension. This is meant to be used with Wasserstein GANs. in_shape (Sequence[int]) – tuple of integers stating the dimension of the input tensor (minus batch dimension) channels (Sequence[int]) – tuple of integers stating the output channels of each convolutional layer strides (Sequence[int]) – tuple of integers stating the stride (downscale factor) of each convolutional layer kernel_size (Union[Sequence[int], int]) – integer or tuple of integers stating size of convolutional kernels num_res_units (int) – integer stating number of convolutions in residual units, 0 means no residual units act – name or type defining activation layers norm – name or type defining normalization layers dropout (Optional[float]) – optional float value in range [0, 1] stating dropout probability for layers, None for no dropout bias (bool) – boolean stating if convolution layers should have a bias component 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. Tensor

Transchex#

TransChex based on: “Hatamizadeh et al.,TransCheX: Self-Supervised Pretraining of Vision-Language Transformers for Chest X-ray Analysis” in_channels (int) – dimension of input channels. img_size (Union[Sequence[int], int]) – dimension of input image. patch_size (Union[int, Tuple[int, int]]) – dimension of patch size. num_classes (int) – number of classes if classification is used. num_language_layers (int) – number of language transformer layers. num_vision_layers (int) – number of vision transformer layers. num_mixed_layers (int) – number of mixed transformer layers. drop_out (float) – faction of the input units to drop. The other parameters are part of the bert_config to MultiModal.from_pretrained. Examples: 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.

NetAdapter#

Wrapper to replace the last layer of model by convolutional layer or FC layer. See also: monai.networks.nets.TorchVisionFCModel model (Module) – a PyTorch model, which can be both 2D and 3D models. typically, it can be a pretrained model in Torchvision, like: resnet18, resnet34, resnet50, resnet101, resnet152, etc. more details: https://pytorch.org/vision/stable/models.html. num_classes (int) – number of classes for the last classification layer. Default to 1. dim (int) – number of supported spatial dimensions in the specified model, depends on the model implementation.

default to 2 as most Torchvision models are for 2D image processing. in_channels (Optional[int]) – number of the input channels of last layer. if None, get it from in_features of last layer. use_conv (bool) – whether to use convolutional layer to replace the last layer, default to False. pool (Optional[Tuple[str, Dict[str, Any]]]) – parameters for the pooling layer, it should be a tuple, the first item is name of the pooling layer, the second item is dictionary of the initialization args. if None, will not replace the layers[-2]. default to (“avg”, {“kernel_size”: 7, “stride”: 1}). bias (bool) – the bias value when replacing the last layer. if False, the layer will not learn an additive bias, default to True. fc_name (str) – the corresponding layer attribute of the last fully connected layer. Defaults to "fc". node_name (str) – the corresponding feature extractor node name of model. Defaults to “”, the extractor is not in use. 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.

TorchVisionFCModel#

Customize the fully connected layer of (pretrained) TorchVision model or replace it by convolutional layer. This class supports two primary use cases: use pool=None to indicate no modification in the pooling layers. It should be used with fc_name to locate the target FC layer to modify: In this case, the class will load a torchvision classification model, replace the last fully connected (FC) layer with a new FC layer with num_classes outputs, example input arguments: use_conv=False, pool=None, fc_name="heads.head". The heads.head specifies the target FC of the input model, could be found by model.named_modules(), for example: use pool="" or set it to a tuple of pooling parameters to indicate modifications of both the pooling and the FC layer. It should be used with node_name to locate the model feature outputs: In this case, the class will load a torchvision model, remove the existing pooling and FC layers, and append an additional convolution layer: use_conv=True, pool="", node_name="permute" append an additional pooling and FC layers: use_conv=False, pool=("avg", {"kernel_size": 7, "stride": 1}), node_name="permute" append an additional pooling and convolution layers: use_conv=True, pool=("avg", {"kernel_size": 7, "stride": 1}), node_name="permute" The permute in the example is the target feature extraction node of the input model_name, could be found by using the torchvision feature extraction utilities, for example: model_name (str) – name of any torchvision model with fully connected layer at the end. resnet18 (default), resnet34, resnet50, resnet101, resnet152, resnext50_32x4d, resnext101_32x8d, wide_resnet50_2, wide_resnet101_2, inception_v3. model details: https://pytorch.org/vision/stable/models.html. num_classes (int) – number of classes for the last classification layer. Default to 1. dim (int) – number of supported spatial dimensions in the specified model, depends on the model implementation. default to 2 as most Torchvision models are for 2D image processing. in_channels (Optional[int]) – number of the input channels of last layer. if None, get it from in_features of last layer. use_conv (bool) – whether to use convolutional layer to replace the last layer, default to False. pool (Optional[Tuple[str, Dict[str, Any]]]) – parameters for the pooling layer, when it’s a tuple, the first item is name of the pooling layer, the second item is dictionary of the initialization args. If None, will not replace the layers[-2]. default to (“avg”, {“kernel_size”: 7, “stride”: 1}). "" indicates not adding a pooling layer. bias (bool) – the bias value when replacing the last layer. if False, the layer will not learn an additive bias, default to True. pretrained (bool) – whether to use the imagenet pretrained weights. Default to False. fc_name (str) – the corresponding layer attribute

of the last fully connected layer. Defaults to "fc". node_name (str) – the corresponding feature extractor node name of model. Defaults to “”, not in use. weights – additional weights enum for the torchvision model. kwargs – additional parameters for the torchvision model. Example:

MILModel#

Multiple Instance Learning (MIL) model, with a backbone classification model. Currently, it only works for 2D images, a typical use case is for classification of the digital pathology whole slide images. The expected shape of input data is [B, N, C, H, W], where B is the batch_size of PyTorch Dataloader and N is the number of instances extracted from every original image in the batch. A tutorial example is available at: Project-MONAI/tutorials. num_classes (int) – number of output classes. mil_mode (str) – MIL algorithm, available values (Defaults to "att"): "mean" - average features from all instances, equivalent to pure CNN (non MIL). "max" - retain only the instance with the max probability for loss calculation. "att" - attention based MIL https://arxiv.org/abs/1802.04712. "att_trans" - transformer MIL https://arxiv.org/abs/2111.01556. "att_trans_pyramid" - transformer pyramid MIL https://arxiv.org/abs/2111.01556. MIL algorithm, available values (Defaults to "att"): "mean" - average features from all instances, equivalent to pure CNN (non MIL). "max" - retain only the instance with the max probability for loss calculation. "att" - attention based MIL https://arxiv.org/abs/1802.04712. "att_trans" - transformer MIL https://arxiv.org/abs/2111.01556. "att_trans_pyramid" - transformer pyramid MIL https://arxiv.org/abs/2111.01556. pretrained (bool) – init backbone with pretrained weights, defaults to True. backbone (Union[Module, str, None]) – Backbone classifier CNN (either None, a nn.Module that returns features, or a string name of a torchvision model). Defaults to None, in which case ResNet50 is used. backbone_num_features (Optional[int]) – Number of output features of the backbone CNN Defaults to None (necessary only when using a custom backbone) trans_blocks (int) – number of the blocks in TransformEncoder layer. trans_dropout (float) – dropout rate in TransformEncoder layer. 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. Tensor

DiNTS#

Reimplementation of DiNTS based on “DiNTS: Differentiable Neural Network Topology Search for 3D Medical Image Segmentation ”. The model contains a pre-defined multi-resolution stem block (defined in this class) and a DiNTS space (defined in monai.networks.nets.TopologyInstance and monai.networks.nets.TopologySearch). The stem block is for: 1) input downsample and 2) output upsample to original size. The model downsamples the input image by 2 (if use_downsample=True). The downsampled image is downsampled by [1, 2, 4, 8] times (num_depths=4) and used as input to the DiNTS search space (TopologySearch) or the DiNTS instance (TopologyInstance). TopologyInstance is the final searched model. The initialization requires the searched architecture codes. TopologySearch is a multi-path topology and cell operation search space. The architecture codes will be initialized as one. TopologyConstruction is the parent class which constructs the instance and search space. To meet the requirements of the

structure, the input size for each spatial dimension should be: divisible by 2 ** (num_depths + 1). dints_space – DiNTS search space. The value should be instance of TopologyInstance or TopologySearch. in_channels (int) – number of input image channels. num_classes (int) – number of output segmentation classes. act_name (Union[Tuple, str]) – activation name, default to 'RELU'. norm_name (Union[Tuple, str]) – normalization used in convolution blocks. Default to InstanceNorm. spatial_dims (int) – spatial 2D or 3D inputs. use_downsample (bool) – use downsample in the stem. If False, the search space will be in resolution [1, 1/2, 1/4, 1/8], if True, the search space will be in resolution [1/2, 1/4, 1/8, 1/16]. node_a – node activation numpy matrix. Its shape is (num_depths, num_blocks + 1). +1 for multi-resolution inputs. In model searching stage, node_a can be None. In deployment stage, node_a cannot be None. Prediction based on dynamic arch_code. x (Tensor) – input tensor.

TopologyConstruction for DiNTS#

The base class for TopologyInstance and TopologySearch. arch_code (Optional[list]) – [arch_code_a, arch_code_c], numpy arrays. The architecture codes defining the model. For example, for a num_depths=4, num_blocks=12 search space: arch_code_a is a 12x10 (10 paths) binary matrix representing if a path is activated. arch_code_c is a 12x10x5 (5 operations) binary matrix representing if a cell operation is used. arch_code in __init__() is used for creating the network and remove unused network blocks. If None, all paths and cells operations will be used, and must be in the searching stage (is_search=True). [arch_code_a, arch_code_c], numpy arrays. The architecture codes defining the model. For example, for a num_depths=4, num_blocks=12 search space: arch_code_a is a 12x10 (10 paths) binary matrix representing if a path is activated. arch_code_c is a 12x10x5 (5 operations) binary matrix representing if a cell operation is used. arch_code in __init__() is used for creating the network and remove unused network blocks. If None, all paths and cells operations will be used, and must be in the searching stage (is_search=True). channel_mul (float) – adjust intermediate channel number, default is 1. cell – operation of each node. num_blocks (int) – number of blocks (depth in the horizontal direction) of the DiNTS search space. num_depths (int) – number of image resolutions of the DiNTS search space: 1, 1/2, 1/4 … in each dimension. use_downsample (bool) – use downsample in the stem. If False, the search space will be in resolution [1, 1/2, 1/4, 1/8], if True, the search space will be in resolution [1/2, 1/4, 1/8, 1/16]. device (str) – 'cpu', 'cuda', or device ID. filter_nums: default to 32. Double the number of channels after downsample. topology related variables: arch_code2in: path activation to its incoming node index (resolution). For depth = 4, arch_code2in = [0, 1, 0, 1, 2, 1, 2, 3, 2, 3]. The first path outputs from node 0 (top resolution), the second path outputs from node 1 (second resolution in the search space), the third path outputs from node 0, etc. arch_code2ops: path activation to operations of upsample 1, keep 0, downsample -1. For depth = 4, arch_code2ops = [0, 1, -1, 0, 1, -1, 0, 1, -1, 0]. The first path does not change resolution, the second path perform upsample, the third perform downsample, etc. arch_code2out: path activation to its output node index. For depth = 4, arch_code2out = [0, 0, 1, 1, 1, 2, 2, 2, 3, 3], the first and second paths connects to node 0 (top resolution), the 3,4,5 paths connects to node 1, etc. This function to be implemented by the architecture instances or search spaces.

TopologyInstance for DiNTS#

Instance of the final searched architecture. Only used in re-training/inference stage. Initialize DiNTS topology search space of neural architectures. x (List[Tensor]) – input tensor. List[Tensor]

TopologySearch for DiNTS#

DiNTS topology search space of neural architectures. Examples: Class method overview: get_prob_a(): convert learnable architecture weights to path activation probabilities. get_ram_cost_usage(): get estimated ram cost. get_topology_entropy(): get topology entropy loss in searching stage. decode(): get final binarized architecture code. gen_mtx(): generate variables needed for topology search. tidx: index used to convert path activation matrix T = (depth,depth) in transfer_mtx to path activation arch_code (1,3*depth-2), for depth = 4, tidx = [0, 1, 4, 5, 6, 9, 10, 11, 14, 15], A tidx (10 binary values) represents the path activation. transfer_mtx: feasible path activation matrix (denoted as T) given a node activation pattern. It is used to convert path activation pattern (1, paths) to node activation (1, nodes) node_act_list: all node activation [2^num_depths-1, depth]. For depth = 4, there are 15 node activation patterns, each of length 4. For example, [1,1,0,0] means nodes 0, 1 are activated (with input paths). all_connect: All possible path activations. For depth = 4, all_connection has 1024 vectors of length 10 (10 paths). The return value will exclude path activation of all 0. Initialize DiNTS topology search space of neural architectures. Decode network log_alpha_a/log_alpha_c using dijkstra shortest path algorithm. [node_a, arch_code_a, arch_code_c, arch_code_a_max] is decoded when using self.decode(). For example, for a num_depths=4, num_blocks=12 search space: node_a is a 4x13 binary matrix representing if a feature node is activated (13 because of multi-resolution inputs). arch_code_a is a 12x10 (10 paths) binary matrix representing if a path is activated. arch_code_c is a 12x10x5 (5 operations) binary matrix representing if a cell operation is used. arch_code with maximum probability Prediction based on dynamic arch_code. x – a list of num_depths input tensors as a multi-resolution input. tensor is of shape BCHW[D] where C must match self.filter_nums. Generate elements needed in decoding and topology. It is used to convert path activation pattern (1, paths) to node activation (1, nodes) patterns, each of length 4. For example, [1,1,0,0] means nodes 0, 1 are activated (with input paths). all_connect: All possible path activations. For depth = 4, all_connection has 1024 vectors of length 10 (10 paths). The return value will exclude path activation of all 0. Get final path and child model probabilities from architecture weights log_alpha_a. This is used in forward pass, getting training loss, and final decoding. child (bool) – return child probability (used in decoding) the path activation probability of size:[number of blocks, number of paths in each block]. For 12 blocks, 4 depths search space, the size is [12,10] probs_a: The probability of all child models (size 1023x10). Each child model is a path activation pattern(1D vector of length 10 for 10 paths). In total 1023 child models (2^10 -1) [number of blocks, number of paths in each block]. For 12 blocks, 4 depths search space, the size is [12,10] (1D vector of length 10 for 10 paths). In total 1023 child models (2^10 -1) arch_code_prob_a Get estimated output tensor size to approximate RAM consumption. in_size – input image shape (4D/5D, [BCHW[D]]) at the highest resolution level. full (bool) – full ram cost usage with all probability of 1. Get topology entropy loss at searching stage. probs – path activation probabilities

ComplexUnet#

This variant of U-Net handles complex-value input/output. It can be used as a model to learn sensitivity maps in multi-coil MRI data. It is built based on monai.networks.nets.BasicUNet by default but the user can input their convolutional model as well. ComplexUnet also applies default normalization to the input which makes it more stable to train. The data being a (complex) 2-channel tensor is a requirement for using this model. Modified and adopted from: facebookresearch/fastMRI spatial_dims (int) – number of spatial dimensions. features (Sequence[int]) – six integers as numbers of features. denotes number of channels in each layer. act (Union[str, tuple]) – activation type and arguments. Defaults to LeakyReLU. norm (Union[str, tuple]) – feature normalization type and arguments. Defaults to instance norm. bias (bool) – whether to have a bias term in convolution blocks. Defaults to True. dropout (Union[float, tuple]) – dropout ratio. Defaults to 0.0. upsample (str) – upsampling mode, available options are "deconv", "pixelshuffle", "nontrainable". pad_factor (int) – an integer denoting the number which each padded dimension will be divisible to. For example, 16 means each dimension will be divisible by 16 after padding conv_net (Optional[Module]) – the learning model used inside the ComplexUnet. The default is monai.networks.nets.basic_unet. The only requirement on the model is to have 2 as input and output number of channels. x (Tensor) – input of shape (B,C,H,W,2) for 2D data or (B,C,H,W,D,2) for 3D data Tensor output of shape (B,C,H,W,2) for 2D data or (B,C,H,W,D,2) for 3D data

CoilSensitivityModel#

This class uses a convolutional model to learn coil sensitivity maps for multi-coil MRI reconstruction. The convolutional model is monai.apps.reconstruction.networks.nets.complex_unet by default but can be specified by the user as well. Learning is done on the center of the under-sampled kspace (that region is fully sampled). The data being a (complex) 2-channel tensor is a requirement for using this model. Modified and adopted from: facebookresearch/fastMRI spatial_dims (int) – number of spatial dimensions. features (Sequence[int]) – six integers as numbers of features. denotes number of channels in each layer. act (Union[str, tuple]) – activation type and arguments. Defaults to LeakyReLU. norm (Union[str, tuple]) – feature normalization type and arguments. Defaults to instance norm. bias (bool) – whether to have a bias term in convolution blocks. Defaults to True. dropout (Union[float, tuple]) – dropout ratio. Defaults to 0.0. upsample (str) – upsampling mode, available options are "deconv", "pixelshuffle", "nontrainable". coil_dim (int) – coil dimension in the data conv_net (Optional[Module]) – the learning model used to estimate the coil sensitivity maps. default is monai.apps.reconstruction.networks.nets.complex_unet. The only requirement on the model is to have 2 as input and output number of channels. masked_kspace (Tensor) – the under-sampled kspace (which is the input measurement). Its shape is (B,C,H,W,2) for 2D data or (B,C,H,W,D,2) for 3D data. mask (Tensor) – the under-sampling mask with shape (1,1,1,W,1) for 2D data or (1,1,1,1,D,1) for 3D data. Tensor predicted coil sensitivity maps with shape (B,C,H,W,2) for 2D data or (B,C,H,W,D,2) for 3D data. Extracts the size of the fully-sampled part of the kspace. Note that when a kspace is under-sampled, a part of its center is fully sampled. This part is called the Auto Calibration Region (ACR). ACR is used for sensitivity map computation. mask (Tensor) – the under-sampling mask of shape (…, S, 1) where S denotes the sampling dimension Tuple[int, int] A tuple containing left index of the region right index of the region left index of the region right index of the region Note indices, then it means that the fully-sampled center region has size 4 starting from 8 to 12.

e2e-VarNet#

The end-to-end variational network (or simply e2e-VarNet) based on Sriram et. al., “End-to-end variational networks for accelerated MRI reconstruction”. It comprises several cascades each consisting of refinement and data consistency steps. The network takes in the under-sampled kspace and estimates the ground-truth reconstruction. Modified and adopted from: facebookresearch/fastMRI coil_sensitivity_model (Module) – A convolutional model for learning coil sensitivity maps. An example is monai.apps.reconstruction.networks.nets.coil_sensitivity_model.CoilSensitivityModel. refinement_model (Module) – A convolutional network used in the refinement step of e2e-VarNet. An example is monai.apps.reconstruction.networks.nets.complex_unet.ComplexUnet. num_cascades (int) – Number of cascades. Each cascade is a monai.apps.reconstruction.networks.blocks.varnetblock.VarNetBlock which consists of refinement and data consistency steps. spatial_dims (int) – number of spatial dimensions. masked_kspace (Tensor) – The under-sampled kspace. It’s a 2D kspace (B,C,H,W,2) with the last dimension being 2 (for real/imaginary parts) and C denoting the coil dimension. 3D data will have the shape (B,C,H,W,D,2). mask (Tensor) – The under-sampling mask with shape (1,1,1,W,1) for 2D data or (1,1,1,1,D,1) for 3D data. Tensor The reconstructed image which is the root sum of squares (rss) of the absolute valueof the inverse fourier of the predicted kspace (note that rss combines coil images into one image). of the inverse fourier of the predicted kspace (note that rss combines coil images into one image).