ANTsPyNet

Release 0.0.1

Nick Tustison, Nick Cullen, Brian Avants

Apr 17, 2024

CONTENTS:

CHAPTER

ONE

ARCHITECTURES

1.1 Autoencoder

antspynet.architectures.**create_autoencoder_model**(*number_of_units_per_layer*, *activation='relu'*, *initializer='glorot_uniform'*)

2-D implementation of the Vgg deep learning architecture.

Builds an autoencoder based on the specified array definining the number of units in the encoding branch. Ported to Keras R from the Keras python implementation here:

https://github.com/XifengGuo/DEC-keras

Parameters

- **number_of_units_per_layer** (*tuple*) – A tuple defining the number of units in the encoding branch.
- **activation** (*string*) – Activation type for the dense layers
- **initializer** (*string*) – Initializer type for the dense layers

Returns
An encoder and autoencoder Keras model.

Return type
Keras model

Example

```
>>> model = create_autoencoder_model((784, 500, 500, 2000, 10))
>>> model.summary()
```

antspynet.architectures.**create_convolutional_autoencoder_model_2d**(*input_image_size*, *number_of_filters_per_layer=(32, 64, 128, 10)*, *convolution_kernel_size=(5, 5)*, *deconvolution_kernel_size=(5, 5)*)

Function for creating a 2-D symmetric convolutional autoencoder model.

Builds an autoencoder based on the specified array definining the number of units in the encoding branch. Ported from the Keras python implementation here:

https://github.com/XifengGuo/DEC-keras

Parameters

- **input_image_size** (*tuple*) – A tuple defining the shape of the 2-D input image
- **number_of_units_per_layer** (*tuple*) – A tuple defining the number of units in the encoding branch.
- **convolution_kernel_size** (*tuple or scalar*) – Kernel size for convolution
- **deconvolution_kernel_size** (*tuple or scalar*) – Kernel size for deconvolution

Returns
A convolutional encoder and autoencoder Keras model.

Return type
Keras models

Example

```
>>> autoencoder, encoder = create_convolutional_autoencoder_model_2d((128, 128, 3))
>>> autoencoder.summary()
>>> encoder.summary()
```

antspynet.architectures.**create_convolutional_autoencoder_model_3d**(*input_image_size*, *number_of_filters_per_layer=(32, 64, 128, 10)*, *convolution_kernel_size=(5, 5, 5)*, *deconvolution_kernel_size=(5, 5, 5)*)

Function for creating a 3-D symmetric convolutional autoencoder model.

Builds an autoencoder based on the specified array definining the number of units in the encoding branch. Ported from the Keras python implementation here:

https://github.com/XifengGuo/DEC-keras

Parameters

- **input_image_size** (*tuple*) – A tuple defining the shape of the 3-D input image
- **number_of_units_per_layer** (*tuple*) – A tuple defining the number of units in the encoding branch.
- **convolution_kernel_size** (*tuple or scalar*) – Kernel size for convolution
- **deconvolution_kernel_size** (*tuple or scalar*) – Kernel size for deconvolution

Returns
A convolutional encoder and autoencoder Keras model.

Return type
Keras models

Example

```
>>> autoencoder, encoder = create_convolutional_autoencoder_model_3d((128, 128, 128,
↪ 3))
>>> autoencoder.summary()
>>> encoder.summary()
```

1.2 Image classification/regression

antspynet.architectures.**create_alexnet_model_2d**(*input_image_size*, *number_of_outputs=1000*, *number_of_dense_units=4096*, *dropout_rate=0.0*, *mode='classification'*)

2-D implementation of the AlexNet deep learning architecture.

Creates a keras model of the AlexNet deep learning architecture for image recognition based on the paper

A. Krizhevsky, and I. Sutskever, and G. Hinton. ImageNet Classification with Deep Convolutional Neural Networks.

available here:

http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

This particular implementation was influenced by the following python implementation:

https://github.com/duggalrahul/AlexNet-Experiments-Keras/ https://github.com/lunardog/convnets-keras/

Parameters

- **input_image_size** (*tuple of length 3*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue). The batch size (i.e., number of training images) is not specified a priori.
- **number_of_outputs** (*integer*) – Number of output units in the final layer.
- **number_of_dense_units** (*integer*) – Number of dense units.
- **dropout_rate** (*scalar*) – Optional regularization parameter between [0, 1]. Default = 0.0.
- **mode** (*string*) – 'classification' or 'regression'. Default = 'classification'.

Returns
A 2-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_alexnet_model_2d((128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_alexnet_model_3d**(*input_image_size*, *number_of_outputs=1000*, *number_of_dense_units=4096*, *dropout_rate=0.0*, *mode='classification'*)

3-D implementation of the AlexNet deep learning architecture.

Creates a keras model of the AlexNet deep learning architecture for image recognition based on the paper

A. Krizhevsky, and I. Sutskever, and G. Hinton. ImageNet Classification with Deep Convolutional Neural Networks.

available here:

http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

This particular implementation was influenced by the following python implementation:

https://github.com/duggalrahul/AlexNet-Experiments-Keras/ https://github.com/lunardog/convnets-keras/

Parameters

- **input_image_size** (*tuple of length 4*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue). The batch size (i.e., number of training images) is not specified a priori.
- **number_of_outputs** (*integer*) – Number of output units in the final layer.
- **number_of_dense_units** (*integer*) – Number of dense units.
- **dropout_rate** (*scalar*) – Optional regularization parameter between [0, 1]. Default = 0.0.
- **mode** (*string*) – 'classification' or 'regression'. Default = 'classification'.

Returns
A 3-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_alexnet_model_3d((128, 128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_densenet_model_2d**(*input_image_size*, *number_of_outputs=1000*, *number_of_filters=16*, *depth=7*, *number_of_dense_blocks=1*, *growth_rate=12*, *dropout_rate=0.2*, *weight_decay=0.0001*, *mode='classification'*)

2-D implementation of the Wide ResNet deep learning architecture.

Creates a keras model of the DenseNet deep learning architecture for image recognition based on the paper

G. Huang, Z. Liu, K. Weinberger, and L. van der Maaten. Densely Connected Convolutional Networks Networks available here:

https://arxiv.org/abs/1608.06993

This particular implementation was influenced by the following python implementation:

https://github.com/tdeboissiere/DeepLearningImplementations/blob/master/DenseNet/densenet.py

Parameters

- **input_image_size** (*tuple of length 3*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **number_of_outputs** (*integer*) – Number of classification labels.
- **number_of_filters** (*integer*) – Number of filters.
- **depth** (*integer*) – Number of layers—must be equal to 3 * N + 4 where N is an integer (default = 7).
- **number_of_dense_blocks** (*integer*) – Number of dense blocks number of dense blocks to add to the end (default = 1).
- **growth_rate** (*integer*) – Number of filters to add for each dense block layer (default = 12).
- **dropout_rate** (*scalar*) – Per drop out layer rate (default = 0.2).
- **weight_decay** (*scalar*) – Weight decay (default = 1e-4).
- **mode** (*string*) – 'classification' or 'regression'. Default = 'classification'.

Returns
A 2-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_densenet_model_2d((128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_densenet_model_3d**(*input_image_size*, *number_of_outputs=1000*, *number_of_filters=16*, *depth=7*, *number_of_dense_blocks=1*, *growth_rate=12*, *dropout_rate=0.2*, *weight_decay=0.0001*, *mode='classification'*)

2-D implementation of the Wide ResNet deep learning architecture.

Creates a keras model of the DenseNet deep learning architecture for image recognition based on the paper

G. Huang, Z. Liu, K. Weinberger, and L. van der Maaten. Densely Connected Convolutional Networks Networks available here:

https://arxiv.org/abs/1608.06993

This particular implementation was influenced by the following python implementation:

https://github.com/tdeboissiere/DeepLearningImplementations/blob/master/DenseNet/densenet.py

Parameters

- **input_image_size** (*tuple of length 4*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **number_of_outputs** (*integer*) – Number of classification labels.
- **number_of_filters** (*integer*) – Number of filters.
- **depth** (*integer*) – Number of layers—must be equal to 3 * N + 4 where N is an integer (default = 7).
- **number_of_dense_blocks** (*integer*) – Number of dense blocks number of dense blocks to add to the end (default = 1).
- **growth_rate** (*integer*) – Number of filters to add for each dense block layer (default = 12).
- **dropout_rate** (*scalar*) – Per drop out layer rate (default = 0.2).
- **weight_decay** (*scalar*) – Weight decay (default = 1e-4).
- **mode** (*string*) – 'classification' or 'regression'. Default = 'classification'.

Returns
A 3-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_densenet_model_3d((128, 128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_resnet_model_2d**(*input_image_size*, *input_scalars_size=0*, *number_of_outputs=1000*, *layers=(1, 2, 3, 4)*, *residual_block_schedule=(3, 4, 6, 3)*, *lowest_resolution=64*, *cardinality=1*, *squeeze_and_excite=False*, *mode='classification'*)

2-D implementation of the ResNet deep learning architecture.

Creates a keras model of the ResNet deep learning architecture for image classification. The paper is available here:

https://arxiv.org/abs/1512.03385

This particular implementation was influenced by the following python implementation:

https://gist.github.com/mjdietzx/0cb95922aac14d446a6530f87b3a04ce

Parameters

- **input_image_size** (*tuple of length 3*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).

- **input_scalars_size** (*integer*) – Optional integer specifying the size of the input vector for scalars that get concatenated to the fully connected layer at the end of the network.
- **number_of_outputs** (*integer*) – Number of units in final layer.
- **layers** (*tuple*) – A tuple determining the number of 'filters' defined at for each layer.
- **residual_block_schedule** (*tuple*) – A tuple defining the how many residual blocks repeats for each layer.
- **lowest_resolution** (*integer*) – Number of filters at the initial layer.
- **cardinality** (*integer*) – perform ResNet (cardinality = 1) or ResNeX (cardinality is not 1 but, instead, powers of 2—try '32').
- **squeeze_and_excite** (*boolean*) – add the squeeze-and-excite block variant.
- **mode** (*string*) – 'classification', 'sigmoid' or 'regression'. Default = 'classification'.

Returns
A 2-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_resnet_model_2d((128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_resnet_model_3d**(*input_image_size*, *input_scalars_size=0*, *number_of_outputs=1000*, *layers=(1, 2, 3, 4)*, *residual_block_schedule=(3, 4, 6, 3)*, *lowest_resolution=64*, *cardinality=1*, *squeeze_and_excite=False*, *mode='classification'*)

3-D implementation of the ResNet deep learning architecture.

Creates a keras model of the ResNet deep learning architecture for image classification. The paper is available here:

https://arxiv.org/abs/1512.03385

This particular implementation was influenced by the following python implementation:

https://gist.github.com/mjdietzx/0cb95922aac14d446a6530f87b3a04ce

Parameters

- **input_image_size** (*tuple of length 4*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **input_scalars_size** (*integer*) – Optional integer specifying the size of the input vector for scalars that get concatenated to the fully connected layer at the end of the network.
- **number_of_outputs** (*integer*) – Number of units in final layer.
- **layers** (*tuple*) – A tuple determining the number of 'filters' defined at for each layer.
- **residual_block_schedule** (*tuple*) – A tuple defining the how many residual blocks repeats for each layer.

- **lowest_resolution** (*integer*) – Number of filters at the initial layer.
- **cardinality** (*integer*) – perform ResNet (cardinality = 1) or ResNeX (cardinality is not 1 but, instead, powers of 2—try '32').
- **squeeze_and_excite** (*boolean*) – add the squeeze-and-excite block variant.
- **mode** (*string*) – 'classification', 'sigmoid' or 'regression'. Default = 'classification'.

Returns
A 3-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_resnet_model_3d((128, 128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_simple_classification_with_spatial_transformer_network_model_2d**(*input_image_size=(128, 128, 1)*, *resampled_size=(30, 30)*, *number_of_outputs=2*)

2-D implementation of the spatial transformer network.

Creates a keras model of the spatial transformer network:

https://arxiv.org/abs/1506.02025

based on the following python Keras model:

https://github.com/oarriaga/STN.keras/blob/master/src/models/STN.py

@param inputImageSize Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue). The batch size (i.e., number of training images) is not specified a priori. @param resampledSize resampled size of the transformed input images. @param numberOfClassificationLabels Number of classes.

Parameters

- **input_image_size** (*tuple of length 3*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **resampled_size** (*tuple of length 2*) – Resampled size of the transformed input images.
- **number_of_outputs** (*integer*) – Number of units in final layer.

Returns
A 2-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_simple_classification_with_spatial_transformer_network_model_
↪2d((128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_simple_classification_with_spatial_transformer_network_model_3d**(*input_image_size*, *resampled_size=(30, 30, 30)*, *number_of_outputs=1*)

3-D implementation of the spatial transformer network.

Creates a keras model of the spatial transformer network:

https://arxiv.org/abs/1506.02025

based on the following python Keras model:

https://github.com/oarriaga/STN.keras/blob/master/src/models/STN.py

@param inputImageSize Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue). The batch size (i.e., number of training images) is not specified a priori. @param resampledSize resampled size of the transformed input images. @param numberOfClassificationLabels Number of classes.

Parameters

- **input_image_size** (*tuple of length 4*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **resampled_size** (*tuple of length 3*) – Resampled size of the transformed input images.
- **number_of_outputs** (*integer*) – Number of units in final layer.

Returns
A 3-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_simple_classification_with_spatial_transformer_network_model_
↪3d((128, 128, 128, 1))
>>> model.summary()
```

1.3 Image super-resolution

antspynet.architectures.**create_deep_back_projection_network_model_2d**(*input_image_size*, *number_of_outputs=1*, *number_of_base_filters=64*, *number_of_feature_filters=256*, *number_of_back_projection_stages=7*, *convolution_kernel_size=(12, 12)*, *strides=(8, 8)*, *last_convolution=(3, 3)*, *number_of_loss_functions=1*)

2-D implementation of the deep back-projection network.

Creates a keras model of the deep back-project network for image super resolution. More information is provided at the authors' website:

https://www.toyota-ti.ac.jp/Lab/Denshi/iim/members/muhammad.haris/projects/DBPN.html

with the paper available here:

https://arxiv.org/abs/1803.02735

This particular implementation was influenced by the following keras (python) implementation:

https://github.com/rajatkb/DBPN-Keras

with help from the original author's Caffe and Pytorch implementations:

https://github.com/alterzero/DBPN-caffe https://github.com/alterzero/DBPN-Pytorch

Parameters

- **input_image_size** (*tuple of length 3*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **number_of_outputs** (*integer*) – Number of outputs (e.g., 3 for RGB images).
- **number_of_feature_filters** (*integer*) – Number of feature filters.
- **number_of_base_filters** (*integer*) – Number of base filters.
- **number_of_back_projection_stages** (*integer*) – Number of up-down-projection stages. This number includes the final up block.
- **convolution_kernel_size** (*tuple of length 2*) – Kernel size for certain convolutional layers. The strides are dependent on the scale factor discussed in original paper. Factors used in the original implementation are as follows: 2x –> convolution_kernel_size=(6, 6), 4x –> convolution_kernel_size=(8, 8), 8x –> convolution_kernel_size=(12, 12). We default to 8x parameters.
- **strides** (*tuple of length 2*) – Strides for certain convolutional layers. This and the convolution_kernel_size are dependent on the scale factor discussed in original paper. Factors used in the original implementation are as follows: 2x –> strides = (2, 2), 4x –> strides = (4, 4), 8x –> strides = (8, 8). We default to 8x parameters.

- **last_convolution** (*tuple of length 2*) – The kernel size for the last convolutional layer.
- **number_of_loss_functions** (*integer*) – The number of data targets, e.g. 2 for 2 targets

Returns
A 2-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_deep_back_projection_network_model_2d((128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_deep_back_projection_network_model_3d**(*input_image_size*, *number_of_outputs=1*, *number_of_base_filters=64*, *number_of_feature_filters=256*, *number_of_back_projection_stages=7*, *convolution_kernel_size=(12, 12, 12)*, *strides=(8, 8, 8)*, *last_convolution=(3, 3, 3)*, *number_of_loss_functions=1*)

3-D implementation of the deep back-projection network.

Creates a keras model of the deep back-project network for image super resolution. More information is provided at the authors' website:

https://www.toyota-ti.ac.jp/Lab/Denshi/iim/members/muhammad.haris/projects/DBPN.html

with the paper available here:

https://arxiv.org/abs/1803.02735

This particular implementation was influenced by the following keras (python) implementation:

https://github.com/rajatkb/DBPN-Keras

with help from the original author's Caffe and Pytorch implementations:

https://github.com/alterzero/DBPN-caffe https://github.com/alterzero/DBPN-Pytorch

Parameters

- **input_image_size** (*tuple of length 4*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **number_of_outputs** (*integer*) – Number of outputs (e.g., 3 for RGB images).
- **number_of_feature_filters** (*integer*) – Number of feature filters.
- **number_of_base_filters** (*integer*) – Number of base filters.

- **number_of_back_projection_stages** (*integer*) – Number of up-down-projection stages. This number includes the final up block.
- **convolution_kernel_size** (*tuple of length 3*) – Kernel size for certain convolutional layers. The strides are dependent on the scale factor discussed in original paper. Factors used in the original implementation are as follows: 2x –> convolution_kernel_size=(6, 6, 6), 4x –> convolution_kernel_size=(8, 8, 8), 8x –> convolution_kernel_size=(12, 12, 12). We default to 8x parameters.
- **strides** (*tuple of length 3*) – Strides for certain convolutional layers. This and the convolution_kernel_size are dependent on the scale factor discussed in original paper. Factors used in the original implementation are as follows: 2x –> strides = (2, 2, 2), 4x –> strides = (4, 4, 4), 8x –> strides = (8, 8, 8). We default to 8x parameters.
- **last_convolution** (*tuple of length 3*) – The kernel size for the last convolutional layer.
- **number_of_loss_functions** (*integer*) – The number of data targets, e.g. 2 for 2 targets

Returns
A 3-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_deep_back_projection_network_model_3d((128, 128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_deep_denoise_super_resolution_model_2d**(*input_image_size*, *layers=2*, *lowest_resolution=64*, *convolution_kernel_size=(3, 3)*, *pool_size=(2, 2)*, *strides=(2, 2)*)

2-D implementation of the denoising autoencoder image super resolution deep learning architecture.

Parameters

- **input_image_size** (*tuple of length 3*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **layers** (*integer*) – Number of architecture layers.
- **lowest_resolution** (*integer*) – Number of filters at the beginning and end of the architecture.
- **convolution_kernel_size** (*2-d tuple*) – specifies the kernel size during the encoding path.
- **pool_size** (*2-d tuple*) – Defines the region for each pooling layer.
- **strides** (*2-d tuple*) – Defines the stride length in each direction.

Returns
A 2-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_deep_denoise_super_resolution_model_2d((128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_deep_denoise_super_resolution_model_3d**(*input_image_size*, *layers=2*, *lowest_resolution=64*, *convolution_kernel_size=(3, 3, 3)*, *pool_size=(2, 2, 2)*, *strides=(2, 2, 2)*)

3-D implementation of the denoising autoencoder image super resolution deep learning architecture.

Parameters

- **input_image_size** (*tuple of length 4*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **layers** (*integer*) – Number of architecture layers.
- **lowest_resolution** (*integer*) – Number of filters at the beginning and end of the architecture.
- **convolution_kernel_size** (*3-d tuple*) – specifies the kernel size during the encoding path.
- **pool_size** (*3-d tuple*) – Defines the region for each pooling layer.
- **strides** (*3-d tuple*) – Defines the stride length in each direction.

Returns
A 3-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_deep_denoise_super_resolution_model_3d((128, 128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_denoising_auto_encoder_super_resolution_model_2d**(*input_image_size*, *convolution_kernel_sizes=[(3, 3), (5, 5)]*, *number_of_encoding_layers=2*, *number_of_filters=64*)

2-D implementation of the denoising autoencoder image super resolution deep learning architecture.

Parameters

- **input_image_size** (*tuple of length 3*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **convolution_kernel_sizes** (*list of 2-d tuples*) – specifies the kernel size at each convolution layer. Default values are the same as given in the original paper. The length of kernel size list must be 1 greater than the tuple length of the number of filters.
- **number_of_encoding_layers** (*integer*) – The number of encoding layers.
- **number_of_filters** (*integer*) – The number of filters for each encoding layer.

Returns
A 2-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_denoising_auto_encoder_super_resolution_model_2d((128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_denoising_auto_encoder_super_resolution_model_3d**(*input_image_size*, *convolution_kernel_sizes=[(3, 3, 3), (5, 5, 5)]*, *number_of_encoding_layers=2*, *number_of_filters=64*)

2-D implementation of the denoising autoencoder image super resolution deep learning architecture.

Parameters

- **input_image_size** (*tuple of length 3*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **convolution_kernel_sizes** (*list of 3-d tuples*) – specifies the kernel size at each convolution layer. Default values are the same as given in the original paper. The length of kernel size list must be 1 greater than the tuple length of the number of filters.
- **number_of_encoding_layers** (*integer*) – The number of encoding layers.
- **number_of_filters** (*integer*) – The number of filters for each encoding layer.

Returns
A 3-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_denoising_auto_encoder_super_resolution_model_3d((128, 128, 128,␣
↪1))
>>> model.summary()
```

antspynet.architectures.**create_expanded_super_resolution_model_2d**(*input_image_size*, *convolution_kernel_sizes=[(9, 9), (1, 1), (3, 3), (5, 5), (5, 5)]*, *number_of_filters=(64, 32, 32, 32)*)

2-D implementation of the expanded image super resolution deep learning architecture.

Creates a keras model of the image super resolution deep learning framework. based on the paper available here:

https://arxiv.org/pdf/1501.00092

This particular implementation is based on the following python implementation:

https://github.com/titu1994/Image-Super-Resolution

Parameters

- **input_image_size** (*tuple of length 3*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **convolution_kernel_sizes** (*list of 2-d tuples*) – specifies the kernel size at each convolution layer. Default values are the same as given in the original paper. The length of kernel size list must be 1 greater than the tuple length of the number of filters.
- **number_of_filters** (*tuple*) – Contains the number of filters for each convolutional layer. Default values are the same as given in the original paper.

Returns
A 2-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_expanded_super_resolution_model_2d((128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_expanded_super_resolution_model_3d**(*input_image_size*, *convolution_kernel_sizes=[(9, 9, 9), (1, 1, 1), (3, 3, 3), (5, 5, 5), (5, 5, 5)]*, *number_of_filters=(64, 32, 32, 32)*)

3-D implementation of the expanded image super resolution deep learning architecture.

Creates a keras model of the image super resolution deep learning framework. based on the paper available here:

https://arxiv.org/pdf/1501.00092

This particular implementation is based on the following python implementation:

https://github.com/titu1994/Image-Super-Resolution

Parameters

- **input_image_size** (*tuple of length 3*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **convolution_kernel_sizes** (*list of 3-d tuples*) – specifies the kernel size at each convolution layer. Default values are the same as given in the original paper. The length of kernel size list must be 1 greater than the tuple length of the number of filters.
- **number_of_filters** (*tuple*) – Contains the number of filters for each convolutional layer. Default values are the same as given in the original paper.

Returns
A 2-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_expanded_super_resolution_model_3d((128, 128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_image_super_resolution_model_2d**(*input_image_size*, *convolution_kernel_sizes=[(9, 9), (1, 1), (5, 5)]*, *number_of_filters=(64, 32)*)

2-D implementation of the image super resolution deep learning architecture.

Creates a keras model of the image super resolution deep learning framework. based on the paper available here:

https://arxiv.org/pdf/1501.00092

This particular implementation is based on the following python implementation:

https://github.com/titu1994/Image-Super-Resolution

Parameters

- **input_image_size** (*tuple of length 3*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **convolution_kernel_sizes** (*list of 2-d tuples*) – specifies the kernel size at each convolution layer. Default values are the same as given in the original paper. The length of kernel size list must be 1 greater than the tuple length of the number of filters.
- **number_of_filters** (*tuple*) – Contains the number of filters for each convolutional layer. Default values are the same as given in the original paper.

Returns
A 2-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_image_super_resolution_model_2d((128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_image_super_resolution_model_3d**(*input_image_size*, *convolution_kernel_sizes=[(9, 9, 9), (1, 1, 1), (5, 5, 5)]*, *number_of_filters=(64, 32)*)

3-D implementation of the image super resolution deep learning architecture.

Creates a keras model of the image super resolution deep learning framework. based on the paper available here:

https://arxiv.org/pdf/1501.00092

This particular implementation is based on the following python implementation:

https://github.com/titu1994/Image-Super-Resolution

Parameters

- **input_image_size** (*tuple of length 4*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **convolution_kernel_sizes** (*list of 3-d tuples*) – specifies the kernel size at each convolution layer. Default values are the same as given in the original paper. The length of kernel size list must be 1 greater than the tuple length of the number of filters.
- **number_of_filters** (*tuple*) – Contains the number of filters for each convolutional layer. Default values are the same as given in the original paper.

Returns
A 3-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_image_super_resolution_model_3d((128, 128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_resnet_super_resolution_model_2d**(*input_image_size*, *convolution_kernel_size=(3, 3)*, *number_of_filters=64*, *number_of_residual_blocks=5*, *number_of_resnet_blocks=1*)

2-D implementation of the ResNet image super resolution architecture.

Creates a keras model of the expanded image super resolution deep learning framework based on the following python implementation:

https://github.com/titu1994/Image-Super-Resolution

Parameters

- **input_image_size** (*tuple of length 3*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **convolution_kernel_size** (*2-d tuple*) – Specifies the kernel size
- **number_of_filters** (*integer*) – The number of filters for each encoding layer.
- **number_of_residual_blocks** (*integer*) – Number of residual blocks.
- **number_of_resnet_blocks** (*integer*) – Number of resnet blocks. Each block will double the upsampling amount.

Returns
A 2-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_resnet_super_resolution_model_2d((128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_resnet_super_resolution_model_3d**(*input_image_size*, *convolution_kernel_size=(3, 3, 3)*, *number_of_filters=64*, *number_of_residual_blocks=5*, *number_of_resnet_blocks=1*)

3-D implementation of the ResNet image super resolution architecture.

Creates a keras model of the expanded image super resolution deep learning framework based on the following python implementation:

https://github.com/titu1994/Image-Super-Resolution

Parameters

- **input_image_size** (*tuple of length 4*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **convolution_kernel_size** (*3-d tuple*) – Specifies the kernel size
- **number_of_filters** (*integer*) – The number of filters for each encoding layer.
- **number_of_residual_blocks** (*integer*) – Number of residual blocks.
- **number_of_resnet_blocks** (*integer*) – Number of resnet blocks. Each block will double the upsampling amount.

Returns
A 3-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_resnet_super_resolution_model_3d((128, 128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_vgg_model_2d**(*input_image_size*, *number_of_outputs=1000*, *layers=(1, 2, 3, 4, 4)*, *lowest_resolution=64*, *convolution_kernel_size=(3, 3)*, *pool_size=(2, 2)*, *strides=(2, 2)*, *number_of_dense_units=4096*, *dropout_rate=0.0*, *style=19*, *mode='classification'*)

2-D implementation of the Vgg deep learning architecture.

Creates a keras model of the Vgg deep learning architecture for image recognition based on the paper

K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition

available here:

https://arxiv.org/abs/1409.1556

This particular implementation was influenced by the following python implementation:

https://gist.github.com/baraldilorenzo/8d096f48a1be4a2d660d

Parameters

- **input_image_size** (*tuple of length 3*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **number_of_outputs** (*integer*) – Number of units in the final dense layer.
- **layers** (*tuple*) – A tuple determining the number of 'filters' defined at for each layer.
- **lowest_resolution** (*integer*) – Number of filters at the beginning.
- **convolution_kernel_size** (*tuple*) – 2-d vector definining the kernel size during the encoding path
- **pool_size** (*tuple*) – 2-d vector defining the region for each pooling layer.
- **strides** (*tuple*) – 2-d vector describing the stride length in each direction.
- **number_of_dense_units** (*integer*) – Number of units in the last layers.
- **dropout_rate** (*scalar*) – Between 0 and 1 to use between dense layers.
- **style** (*integer*) – '16' or '19' for VGG16 or VGG19, respectively.
- **mode** (*string*) – 'classification' or 'regression'. Default = 'classification'.

Returns
A 2-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_vgg_model_2d((128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_vgg_model_3d**(*input_image_size*, *number_of_outputs=1000*, *layers=(1, 2, 3, 4, 4)*, *lowest_resolution=64*, *convolution_kernel_size=(3, 3, 3)*, *pool_size=(2, 2, 2)*, *strides=(2, 2, 2)*, *number_of_dense_units=4096*, *dropout_rate=0.0*, *style=19*, *mode='classification'*)

3-D implementation of the Vgg deep learning architecture.

Creates a keras model of the Vgg deep learning architecture for image recognition based on the paper

K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition

available here:

https://arxiv.org/abs/1409.1556

This particular implementation was influenced by the following python implementation:

https://gist.github.com/baraldilorenzo/8d096f48a1be4a2d660d

Parameters

- **input_image_size** (*tuple of length 4*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **number_of_outputs** (*integer*) – Number of units in the final dense layer.
- **layers** (*tuple*) – A tuple determining the number of 'filters' defined at for each layer.
- **lowest_resolution** (*integer*) – Number of filters at the beginning.
- **convolution_kernel_size** (*tuple*) – 3-d vector defining the kernel size during the encoding path
- **pool_size** (*tuple*) – 3-d vector defining the region for each pooling layer.
- **strides** (*tuple*) – 3-d vector describing the stride length in each direction.
- **number_of_dense_units** (*integer*) – Number of units in the last layers.
- **dropout_rate** (*scalar*) – Between 0 and 1 to use between dense layers.
- **style** (*integer*) – '16' or '19' for VGG16 or VGG19, respectively.
- **mode** (*string*) – 'classification' or 'regression'. Default = 'classification'.

Returns
A 3-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_vgg_model_3d((128, 128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_wide_resnet_model_2d**(*input_image_size*, *number_of_outputs=1000*, *depth=2*, *width=1*, *residual_block_schedule=(16, 32, 64)*, *pool_size=(8, 8)*, *dropout_rate=0.0*, *weight_decay=0.0005*, *mode='classification'*)

2-D implementation of the Wide ResNet deep learning architecture.

Creates a keras model of the Wide ResNet deep learning architecture for image classification/regression. The paper is available here:

https://arxiv.org/abs/1512.03385

This particular implementation was influenced by the following python implementation:

https://github.com/titu1994/Wide-Residual-Networks

Parameters

- **input_image_size** (*tuple of length 3*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **number_of_outputs** (*integer*) – Number of units in the final dense layer.
- **depth** (*integer*) – Determines the depth of the network. Related to the actual number of layers by number_of_layers = depth * 6 + 4. Default = 2 (such that number_of_layers = 16).
- **width** (*integer*) – Determines the width of the network. Default = 1.
- **residual_block_schedule** (*tuple*) – Determines the number of filters per convolutional block. Default = (16, 32, 64).
- **pool_size** (*tuple*) – Pool size for final average pooling layer. Default = (8, 8).
- **dropout_rate** (*scalar*) – Float between 0 and 1 to use between dense layers.
- **weight_decay** (*scalar*) – Weighting parameter for regularization of the kernel weights of the convolution layers. Default = 0.0005.
- **mode** (*string*) – 'classification' or 'regression'. Default = 'classification'.

Returns
A 2-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_wide_resnet_model_2d((128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_wide_resnet_model_3d**(*input_image_size*, *number_of_outputs=1000*, *depth=2*, *width=1*, *residual_block_schedule=(16, 32, 64)*, *pool_size=(8, 8, 8)*, *dropout_rate=0.0*, *weight_decay=0.0005*, *mode='classification'*)

3-D implementation of the Wide ResNet deep learning architecture.

Creates a keras model of the Wide ResNet deep learning architecture for image classification/regression. The paper is available here:

https://arxiv.org/abs/1512.03385

This particular implementation was influenced by the following python implementation:

https://github.com/titu1994/Wide-Residual-Networks

Parameters

- **input_image_size** (*tuple of length 4*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **number_of_outputs** (*integer*) – Number of units in the final dense layer.
- **depth** (*integer*) – Determines the depth of the network. Related to the actual number of layers by number_of_layers = depth * 6 + 4. Default = 2 (such that number_of_layers = 16).
- **width** (*integer*) – Determines the width of the network. Default = 1.
- **residual_block_schedule** (*tuple*) – Determines the number of filters per convolutional block. Default = (16, 32, 64).
- **pool_size** (*tuple*) – Pool size for final average pooling layer. Default = (8, 8, 8).
- **dropout_rate** (*scalar*) – Float between 0 and 1 to use between dense layers.
- **weight_decay** (*scalar*) – Weighting parameter for regularization of the kernel weights of the convolution layers. Default = 0.0005.
- **mode** (*string*) – 'classification' or 'regression'. Default = 'classification'.

Returns
A 3-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_wide_resnet_model_3d((128, 128, 128, 1))
>>> model.summary()
```

1.4 Image voxelwise segmentation

antspynet.architectures.**create_unet_model_2d**(*input_image_size*, *number_of_outputs=2*, *scalar_output_size=0*, *scalar_output_activation='relu'*, *number_of_layers=4*, *number_of_filters_at_base_layer=32*, *number_of_filters=None*, *convolution_kernel_size=(3, 3)*, *deconvolution_kernel_size=(2, 2)*, *pool_size=(2, 2)*, *strides=(2, 2)*, *dropout_rate=0.0*, *weight_decay=0.0*, *mode='classification'*, *additional_options=None*)

2-D implementation of the U-net deep learning architecture.

Creates a keras model of the U-net deep learning architecture for image segmentation and regression. More information is provided at the authors' website:

https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

with the paper available here:

https://arxiv.org/abs/1505.04597

This particular implementation was influenced by the following python implementation:

https://github.com/joelthelion/ultrasound-nerve-segmentation

Parameters

- **input_image_size** (*tuple of length 3*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **number_of_outputs** (*integer*) – Meaning depends on the mode. For *classification* this is the number of segmentation labels. For *regression* this is the number of outputs.
- **scalar_output_size** (*integer*) – If greater than 0, a global average pooling from each encoding layer is concatenated to a dense layer as a secondary output.
- **scalar_output_activation** (*string*) – Activation for nonzero output scalar.
- **number_of_layers** (*integer*) – number of encoding/decoding layers.
- **number_of_filters_at_base_layer** (*integer*) – number of filters at the beginning and end of the *U*. Doubles at each descending/ascending layer.
- **number_of_filters** (*tuple*) – tuple explicitly setting the number of filters at each layer. One can either set this or number_of_layers and number_of_filters_at_base_layer. Default = None.
- **convolution_kernel_size** (*tuple of length 2*) – Defines the kernel size during the encoding.
- **deconvolution_kernel_size** (*tuple of length 2*) – Defines the kernel size during the decoding.

- **pool_size** (*tuple of length 2*) – Defines the region for each pooling layer.
- **strides** (*tuple of length 2*) – Strides for the convolutional layers.
- **dropout_rate** (*scalar*) – Float between 0 and 1 to use between dense layers.
- **weight_decay** (*scalar*) – Weighting parameter for L2 regularization of the kernel weights of the convolution layers. Default = 0.0.
- **mode** (*string*) – *classification*, *regression*, or *sigmoid*. Default = *classification*.
- **additional_options** (*string or tuple of strings*) –

  specific configuration add-ons/tweaks:

  – "attentionGating" – attention-unet variant in https://pubmed.ncbi.nlm.nih.gov/33288961/

  – "nnUnetActivationStyle" – U-net activation explained in https://pubmed.ncbi.nlm.nih.gov/33288961/

  – "initialConvolutionKernelSize[X]" – Set the first two convolutional layer kernel sizes to X.

Returns
A 2-D keras model defining the U-net network.

Return type
Keras model

Example

```
>>> model = create_unet_model_2d((128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_unet_model_3d**(*input_image_size*, *number_of_outputs=2*, *scalar_output_size=0*, *scalar_output_activation='relu'*, *number_of_layers=4*, *number_of_filters_at_base_layer=32*, *number_of_filters=None*, *convolution_kernel_size=(3, 3, 3)*, *deconvolution_kernel_size=(2, 2, 2)*, *pool_size=(2, 2, 2)*, *strides=(2, 2, 2)*, *dropout_rate=0.0*, *weight_decay=0.0*, *mode='classification'*, *additional_options=None*)

3-D implementation of the U-net deep learning architecture.

Creates a keras model of the U-net deep learning architecture for image segmentation and regression. More information is provided at the authors' website:

https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

with the paper available here:

https://arxiv.org/abs/1505.04597

This particular implementation was influenced by the following python implementation:

https://github.com/joelthelion/ultrasound-nerve-segmentation

Parameters

- **input_image_size** (*tuple of length 4*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **number_of_outputs** (*integer*) – Meaning depends on the mode. For *classification* this is the number of segmentation labels. For *regression* this is the number of outputs.
- **scalar_output_size** (*integer*) – If greater than 0, a global average pooling from each encoding layer is concatenated to a dense layer as a secondary output.
- **scalar_output_activation** (*string*) – Activation for nonzero output scalar.
- **number_of_layers** (*integer*) – number of encoding/decoding layers.
- **number_of_filters_at_base_layer** (*integer*) – number of filters at the beginning and end of the *U*. Doubles at each descending/ascending layer.
- **number_of_filters** (*tuple*) – tuple explicitly setting the number of filters at each layer. One can either set this or number_of_layers and number_of_filters_at_base_layer. Default = None.
- **convolution_kernel_size** (*tuple of length 3*) – Defines the kernel size during the encoding.
- **deconvolution_kernel_size** (*tuple of length 3*) – Defines the kernel size during the decoding.
- **pool_size** (*tuple of length 3*) – Defines the region for each pooling layer.
- **strides** (*tuple of length 3*) – Strides for the convolutional layers.
- **dropout_rate** (*scalar*) – Float between 0 and 1 to use between dense layers.
- **weight_decay** (*scalar*) – Weighting parameter for L2 regularization of the kernel weights of the convolution layers. Default = 0.0.
- **mode** (*string*) – *classification regression*, or *sigmoid*. Default = *classification*.
- **additional_options** (*string or tuple of strings*) –

  specific configuration add-ons/tweaks:

  - "attentionGating" – attention-unet variant in https://pubmed.ncbi.nlm.nih.gov/33288961/
  - "nnUnetActivationStyle" – U-net activation explained in https://pubmed.ncbi.nlm.nih.gov/33288961/
  - "initialConvolutionalKernelSize[X]" – Set the first two convolutional layer kernel sizes to X.

Returns
A 3-D keras model defining the U-net network.

Return type
Keras model

Example

```
>>> model = create_unet_model_3d((128, 128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_resunet_model_2d**(*input_image_size*, *number_of_outputs=1*, *number_of_filters_at_base_layer=32*, *bottle_neck_block_depth_schedule=(3, 4)*, *convolution_kernel_size=(3, 3)*, *deconvolution_kernel_size=(2, 2)*, *dropout_rate=0.0*, *weight_decay=0.0*, *mode='classification'*)

2-D implementation of the Resnet + U-net deep learning architecture.

Creates a keras model of the U-net + ResNet deep learning architecture for image segmentation and regression with the paper available here:

https://arxiv.org/abs/1608.04117

This particular implementation was ported from the following python implementation:

https://github.com/veugene/fcn_maker/

Parameters

- **input_image_size** (*tuple of length 3*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue). The batch size (i.e., number of training images) is not specified a priori.
- **number_of_outputs** (*integer*) – Meaning depends on the mode. For 'classification' this is the number of segmentation labels. For 'regression' this is the number of outputs.
- **number_of_filters_at_base_layer** (*integer*) – Number of filters at the beginning and end of the 'U'. Doubles at each descending/ascending layer.
- **bottle_neck_block_depth_schedule** (*tuple*) – Tuple that provides the encoding layer schedule for the number of bottleneck blocks per long skip connection.
- **convolution_kernel_size** (*tuple of length 2*) – 2-d vector defining the kernel size during the encoding path
- **deconvolution_kernel_size** (*tuple of length 2*) – 2-d vector defining the kernel size during the decoding
- **dropout_rate** (*scalar*) – Float between 0 and 1 to use between dense layers.
- **weight_decay** (*scalar*) – Weighting parameter for L2 regularization of the kernel weights of the convolution layers. Default = 0.0.
- **mode** (*string*) – 'classification' or 'regression'. Default = 'classification'.

Returns
A 2-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_resunet_model_2d((128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_resunet_model_3d**(*input_image_size*, *number_of_outputs=1*, *number_of_filters_at_base_layer=32*, *bottle_neck_block_depth_schedule=(3, 4)*, *convolution_kernel_size=(3, 3, 3)*, *deconvolution_kernel_size=(2, 2, 2)*, *dropout_rate=0.0*, *weight_decay=0.0*, *mode='classification'*)

3-D implementation of the Resnet + U-net deep learning architecture.

Creates a keras model of the U-net + ResNet deep learning architecture for image segmentation and regression with the paper available here:

https://arxiv.org/abs/1608.04117

This particular implementation was ported from the following python implementation:

https://github.com/veugene/fcn_maker/

Parameters

- **input_image_size** (*tuple of length 4*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue). The batch size (i.e., number of training images) is not specified a priori.
- **number_of_outputs** (*integer*) – Meaning depends on the mode. For 'classification' this is the number of segmentation labels. For 'regression' this is the number of outputs.
- **number_of_filters_at_base_layer** (*integer*) – Number of filters at the beginning and end of the 'U'. Doubles at each descending/ascending layer.
- **bottle_neck_block_depth_schedule** (*tuple*) – Tuple that provides the encoding layer schedule for the number of bottleneck blocks per long skip connection.
- **convolution_kernel_size** (*tuple of length 3*) – 3-d vector defining the kernel size during the encoding path
- **deconvolution_kernel_size** (*tuple of length 3*) – 3-d vector defining the kernel size during the decoding
- **dropout_rate** (*scalar*) – Float between 0 and 1 to use between dense layers.
- **weight_decay** (*scalar*) – Weighting parameter for L2 regularization of the kernel weights of the convolution layers. Default = 0.0.
- **mode** (*string*) – 'classification' or 'regression'. Default = 'classification'.

Returns
A 3-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_resunet_model_3d((128, 128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_denseunet_model_2d**(*input_image_size*, *number_of_outputs=1*, *number_of_layers_per_dense_block=(6, 12, 36, 24)*, *growth_rate=48*, *initial_number_of_filters=96*, *reduction_rate=0.0*, *depth=7*, *dropout_rate=0.0*, *weight_decay=0.0001*, *mode='classification'*)

2-D implementation of the dense U-net deep learning architecture.

Creates a keras model of the dense U-net deep learning architecture for image segmentation

X. Li, H. Chen, X. Qi, Q. Dou, C.-W. Fu, P.-A. Heng. H-DenseUNet: Hybrid Densely Connected UNet for Liver and Tumor Segmentation from CT Volumes

available here:

https://arxiv.org/pdf/1709.07330.pdf

with the author's implementation available at:

https://github.com/xmengli999/H-DenseUNet

Parameters

- **input_image_size** (*tuple of length 3*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue). The batch size (i.e., number of training images) is not specified a priori.
- **number_of_outputs** (*integer*) – Meaning depends on the mode. For 'classification' this is the number of segmentation labels. For 'regression' this is the number of outputs.
- **number_of_layers_per_dense_blocks** (*tuple*) – Number of dense blocks per layer.
- **growth_rate** (*integer*) – Number of filters to add for each dense block layer (default = 48).
- **initial_number_of_filters** (*integer*) – Number of filters at the beginning (default = 96).
- **reduction_rate** (*scalar*) – Reduction factor of transition blocks.
- **depth** (*integer*) – Number of layers—must be equal to 3 * N + 4 where N is an integer (default = 7).
- **dropout_rate** (*scalar*) – Float between 0 and 1 to use between dense layers.
- **weight_decay** (*scalar*) – Weighting parameter for L2 regularization of the kernel weights of the convolution layers (default = 1e-4).

Returns
A 2-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_denseunet_model_2d((128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_denseunet_model_3d**(*input_image_size*, *number_of_outputs=1*, *number_of_layers_per_dense_block=(6, 12, 36, 24)*, *growth_rate=48*, *initial_number_of_filters=96*, *reduction_rate=0.0*, *depth=7*, *dropout_rate=0.0*, *weight_decay=0.0001*, *mode='classification'*)

2-D implementation of the dense U-net deep learning architecture.

Creates a keras model of the dense U-net deep learning architecture for image segmentation

X. Li, H. Chen, X. Qi, Q. Dou, C.-W. Fu, P.-A. Heng. H-DenseUNet: Hybrid Densely Connected UNet for Liver and Tumor Segmentation from CT Volumes

available here:

https://arxiv.org/pdf/1709.07330.pdf

with the author's implementation available at:

https://github.com/xmengli999/H-DenseUNet

Parameters

- **input_image_size** (*tuple of length 4*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue). The batch size (i.e., number of training images) is not specified a priori.
- **number_of_outputs** (*integer*) – Meaning depends on the mode. For 'classification' this is the number of segmentation labels. For 'regression' this is the number of outputs.
- **number_of_layers_per_dense_blocks** (*tuple*) – Number of dense blocks per layer.
- **growth_rate** (*integer*) – Number of filters to add for each dense block layer (default = 48).
- **initial_number_of_filters** (*integer*) – Number of filters at the beginning (default = 96).
- **reduction_rate** (*scalar*) – Reduction factor of transition blocks.
- **depth** (*integer*) – Number of layers—must be equal to 3 * N + 4 where N is an integer (default = 7).
- **dropout_rate** (*scalar*) – Float between 0 and 1 to use between dense layers.
- **weight_decay** (*scalar*) – Weighting parameter for L2 regularization of the kernel weights of the convolution layers (default = 1e-4).

Returns
A 3-D Keras model defining the network.

Return type
Keras model

Example

```
>>> model = create_denseunet_model_3d((128, 128, 128, 1))
>>> model.summary()
```

antspynet.architectures.**create_nobrainer_unet_model_3d**(*input_image_size*)

Implementation of the "NoBrainer" U-net architecture

Creates a keras model of the U-net deep learning architecture for image segmentation available at:

https://github.com/neuronets/nobrainer/

Parameters

input_image_size (*tuple of length 4*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).

Returns

A 3-D keras model defining the U-net network.

Return type

Keras model

Example

```
>>> model = create_nobrainer_unet_model_3d((None, None, None, 1))
>>> model.summary()
```

antspynet.architectures.**create_hippmapp3r_unet_model_3d**(*input_image_size*, *do_first_network=True*, *data_format='channels_last'*)

Implementation of the "HippMapp3r" U-net architecture

Creates a keras model implementation of the u-net architecture described here:

https://onlinelibrary.wiley.com/doi/pdf/10.1002/hbm.24811

with the implementation available here:

https://github.com/mgoubran/HippMapp3r

Parameters

- **input_image_size** (*tuple of length 4*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **do_first_network** (*boolean*) – Boolean dictating if the model built should be the first (initial) network or second (refinement) network.
- **data_format** (*string*) – One of "channels_first" or "channels_last". We do this for this specific architecture as the original weights were saved in "channels_first" format.

Returns

A 3-D keras model defining the U-net network.

Return type

Keras model

Example

```
>>> shape_initial_stage = (160, 160, 128)
>>> model_initial_stage = antspynet.create_hippmapp3r_unet_model_3d((*shape_initial_
↪stage, 1), True)
>>> model_initial_stage.load_weights(antspynet.get_pretrained_network(
↪"hippMapp3rInitial"))
>>> shape_refine_stage = (112, 112, 64)
>>> model_refine_stage = antspynet.create_hippmapp3r_unet_model_3d((*shape_refine_
↪stage, 1), False)
>>> model_refine_stage.load_weights(antspynet.get_pretrained_network(
↪"hippMapp3rRefine"))
```

antspynet.architectures.**create_sysu_media_unet_model_2d**(*input_image_size*, *anatomy='wmh'*)

Implementation of the sysu_media U-net architecture

Creates a keras model implementation of the u-net architecture in the 2017 MICCAI WMH challenge by the sysu_medial team described here:

https://pubmed.ncbi.nlm.nih.gov/30125711/

with the original implementation available here:

https://github.com/hongweilibran/wmh_ibbmTum

Parameters

- **input_image_size** (*tuple of length 4*) – This will be (200, 200, 2) for t1/flair input and (200, 200, 1)} for flair-only input.
- **anatomy** (*string*) – "wmh" or "claustrum"

Returns
A 2-D keras model defining the U-net network.

Return type
Keras model

Example

```
>>> image_size = (200, 200)
>>> model = antspynet.create_sysu_media_unet_model_2d((*image_size, 1))
```

antspynet.**create_hypothalamus_unet_model_3d**(*input_image_size*)

Implementation of the U-net architecture for hypothalamus segmentation described in

https://pubmed.ncbi.nlm.nih.gov/32853816/

and ported from the original implementation:

https://github.com/BBillot/hypothalamus_seg

The network has is characterized by the following parameters:

- 3 resolution levels: 24 —> 48 —> 96 filters
- convolution: kernel size: (3, 3, 3), activation: 'elu',
- pool size: (2, 2, 2)

Returns
A 3-D keras model defining the U-net network.

Return type
Keras model

Example

```
>>> model = create_hypothalamus_unet_model_3d((160, 160, 160, 1))
```

1.5 Custom

antspynet.architectures.**create_simple_fully_convolutional_network_model_3d**(*input_image_size*, *number_of_filters_per_layer=(32, 64, 128, 256, 256, 64)*, *number_of_bins=40*, *dropout_rate=0.5*)

Implementation of the "SCFN" architecture for Brain/Gender prediction

Creates a keras model implementation of the Simple Fully Convolutional Network model from the FMRIB group:

https://github.com/ha-ha-ha-han/UKBiobank_deep_pretrain

Parameters

- **input_image_size** (*tuple of length 4*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **number_of_filters_per_layer** (*array*) – number of filters for the convolutional layers.
- **number_of_bins** (*integer*) – number of bins for final softmax output.
- **dropout_rate** (*float between 0 and 1*) – Optional dropout rate before final convolution layer.

Returns
A 3-D keras model.

Return type
Keras model

Example

```
>>> model = create_simple_fully_convolutional_network_model_3d((None, None, None,␣
↪1))
>>> model.summary()
```

1.6 Generative adverserial networks

class antspynet.architectures.**VanillaGanModel**(*input_image_size*, *latent_dimension=100*)

Vanilla GAN model.

Parameters

- **input_image_size** (*tuple*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **latent_dimension** (*integer*) –

Returns
A Keras model defining the network.

Return type
Keras model

class antspynet.architectures.**DeepConvolutionalGanModel**(*input_image_size*, *latent_dimension=100*)

GAN model using CNNs

Deep convolutional generative adverserial network from the paper:

https://arxiv.org/abs/1511.06434

and ported from the Keras (python) implementation:

https://github.com/eriklindernoren/Keras-GAN/blob/master/dcgan/dcgan.py

Parameters

- **input_image_size** (*tuple*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **latent_dimension** (*integer*) –

Returns
A Keras model defining the network.

Return type
Keras model

class antspynet.architectures.**WassersteinGanModel**(*input_image_size*, *latent_dimension=100*, *number_of_critic_iterations=5*, *clip_value=0.01*)

Wasserstein GAN model

Wasserstein generative adverserial network from the paper:

https://arxiv.org/abs/1701.07875

and ported from the Keras implementation:

https://github.com/eriklindernoren/Keras-GAN/blob/master/srgan/srgan.py

Parameters

- **input_image_size** (*tuple*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).

- **latent_dimension** (*integer*) – Default = 100.
- **number_of_critic_iterations** (*integer*) – Default = 5.
- **clip_value** (*float*) – Default = 0.01.

Returns
A Keras model defining the network.

Return type
Keras model

class antspynet.architectures.**ImprovedWassersteinGanModel**(*input_image_size*, *latent_dimension=100*, *number_of_critic_iterations=5*)

Improved Wasserstein GAN model

Improved Wasserstein generative adverserial network from the paper:

https://arxiv.org/abs/1704.00028

and ported from the Keras implementation:

https://github.com/eriklindernoren/Keras-GAN/blob/master/wgan_gp/wgan_gp.py

Parameters

- **input_image_size** (*tuple*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **latent_dimension** (*integer*) – Default = 100.
- **number_of_critic_iterations** (*integer*) – Default = 5.

Returns
A Keras model defining the network.

Return type
Keras model

class antspynet.architectures.**CycleGanModel**(*input_image_size*, *lambda_cycle_loss_weight=10.0*, *lambda_identity_loss_weight=1.0*, *number_of_filters_at_base_layer=(32, 64)*)

Cycle GAN model

Cycle generative adverserial network from the paper:

https://arxiv.org/pdf/1703.10593

and ported from the Keras (python) implementation:

https://github.com/eriklindernoren/Keras-GAN/blob/master/cyclegan/cyclegan.py

Parameters

- **input_image_size** (*tuple*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **latent_dimension** (*integer*) –

Returns
A Keras model defining the network.

Return type
Keras model

class antspynet.architectures.**SuperResolutionGanModel**(*low_resolution_image_size*, *scale_factor=2*, *use_image_net_weights=True*, *number_of_residual_blocks=16*, *number_of_filters_at_base_layer=(64, 64)*)

Super resolution GAN model

Super resolution generative adverserial network from the paper:

https://arxiv.org/abs/1609.04802

and ported from the Keras implementation:

https://github.com/eriklindernoren/Keras-GAN/blob/master/wgan/wgan.py

Parameters

- **input_image_size** (*tuple*) – Used for specifying the input tensor shape. The shape (or dimension) of that tensor is the image dimensions followed by the number of channels (e.g., red, green, and blue).
- **low_resolution_image_size** (*tuple*) – Size of the input image.
- **scale_factor** (*integer*) – Upsampling factor for the output super-resolution image.
- **use_image_net_weights** (*boolean*) – Determines whether or not one uses the image-net weights. Only valid for 2-D images.
- **number_of_residual_blocks** (*16*) – Number of residual blocks used in constructing the generator.
- **number_of_filters_at_base_layer** (*tuple of length 2*) – Number of filters at the base layer for the generator and discriminator, respectively.

Returns
A Keras model defining the network.

Return type
Keras model

CHAPTER

TWO

UTILITIES

2.1 Custom metrics

antspynet.utilities.**binary_dice_coefficient**(*smoothing_factor=0.0*)

Binary dice segmentation loss.

Note: Assumption is that y_true is *not* a one-hot representation of the segmentation batch. For use with e.g., sigmoid activation.

Parameters

smoothing_factor (*float*) – Used to smooth value during optimization

Return type

Loss value (negative Dice coefficient).

antspynet.utilities.**multilabel_dice_coefficient**(*dimensionality=3*, *smoothing_factor=0.0*)

Multi-label dice segmentation loss

Note: Assumption is that y_true is a one-hot representation of the segmentation batch. The background (label 0) should be included but is not used in the calculation. For use with e.g., softmax activation.

Parameters

- **dimensionality** (*dimensionality*) – Image dimension
- **smoothing_factor** (*float*) – Image dimension

Return type

Loss value (negative Dice coefficient).

Example

```
>>> import ants
>>> import antspynet
>>> import tensorflow as tf
>>> import numpy as np
>>>
>>> r16 = ants.image_read(ants.get_ants_data("r16"))
>>> r16_seg = ants.kmeans_segmentation(r16, 3)['segmentation']
>>> r16_array = np.expand_dims(r16_seg.numpy(), axis=0)
>>> r16_tensor = tf.convert_to_tensor(antspynet.encode_unet(r16_array, (0, 1, 2,␣
↪3)))
>>>
```

(continues on next page)

(continued from previous page)

```
>>> r64 = ants.image_read(ants.get_ants_data("r64"))
>>> r64_seg = ants.kmeans_segmentation(r64, 3)['segmentation']
>>> r64_array = np.expand_dims(r64_seg.numpy(), axis=0)
>>> r64_tensor = tf.convert_to_tensor(antspynet.encode_unet(r64_array, (0, 1, 2,␣
↪3)))
>>>
>>> dice_loss = antspynet.multilabel_dice_coefficient(dimensionality=2)
>>> loss_value = dice_loss(r16_tensor, r64_tensor).numpy()
>>> # Compare with...
>>> ants.label_overlap_measures(r16_seg, r64_seg)
```

antspynet.utilities.**peak_signal_to_noise_ratio**(*y_true*, *y_pred*)

antspynet.utilities.**pearson_correlation_coefficient**(*y_true*, *y_pred*)

antspynet.utilities.**categorical_focal_loss**(*gamma=2.0*, *alpha=0.25*)

antspynet.utilities.**weighted_categorical_crossentropy**(*weights*)

antspynet.**maximum_mean_discrepancy**(*sigma=1.0*)

2.2 Custom normalization layers

class antspynet.utilities.**InstanceNormalization**(**args*, ***kwargs*)

Instance normalization layer.

Normalize the activations of the previous layer at each step, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.

Taken from

https://github.com/keras-team/keras-contrib/blob/master/keras_contrib/layers/normalization/instancenormalization.py

Parameters

- **axis** (*integer*) – Integer specifying which axis should be normalized, typically the feature axis. For example, after a Conv2D layer with *channels_first*, set axis = 1. Setting *axis=-1L* will normalize all values in each instance of the batch. Axis 0 is the batch dimension for tensorflow backend so we throw an error if *axis = 0*.
- **epsilon** (*float*) – Small float added to variance to avoid dividing by zero.
- **center** (If True, add offset of *beta* to normalized tensor.) – If False, *beta* is ignored.
- **scale** (If True, multiply by *gamma*.) – If False, *gamma* is not used. When the next layer is linear (also e.g., *nn.relu*), this can be disabled since the scaling will be done by the next layer.
- **beta_initializer** (*string*) – Initializer for the beta weight.
- **gamma_initializer** (*string*) – Initializer for the gamma weight.
- **beta_regularizer** (*string*) – Optional regularizer for the beta weight.
- **gamma_regularizer** (*string*) – Optional regularizer for the gamma weight.
- **beta_constraint** (*string*) – Optional constraint for the beta weight.

- **gamma_constraint** (*string*) – Optional constraint for the gamma weight.

Return type
Keras layer

2.3 Custom activation layers

class antspynet.utilities.**LogSoftmax**(**args*, ***kwargs*)

Log Softmax activation function.

Input shape:
Arbitrary. Use the keyword argument *input_shape* (tuple of integers, does not include the samples axis) when using this layer as the first layer in a model.

Output shape:
Same shape as the input.

Parameters
axis – Integer, axis along which the softmax normalization is applied.

2.4 Resample tensor layer

class antspynet.utilities.**ResampleTensorLayer2D**(**args*, ***kwargs*)

Tensor resampling layer (2D).

Parameters

- **shape** (*tuple*) – Specifies the output shape of the resampled tensor.
- **interpolation_type** (*string*) – One of 'nearest_neighbor', 'linear', or 'cubic'.

Returns
A keras layer

Return type
Keras layer

class antspynet.utilities.**ResampleTensorLayer3D**(**args*, ***kwargs*)

Tensor resampling layer (3D).

Parameters

- **shape** (*tuple*) – Specifies the output shape of the resampled tensor.
- **interpolation_type** (*string*) – One of 'nearest_neighbor', 'linear', or 'cubic'.

Returns
A keras layer

Return type
Keras layer

2.5 Mixture density networks

2.6 Attention

class antspynet.utilities.**AttentionLayer2D**(**args*, ***kwargs*)

Attention layer (2-D) from the self attention GAN

taken from the following python implementation

https://stackoverflow.com/questions/50819931/self-attention-gan-in-keras

based on the following paper:

https://arxiv.org/abs/1805.08318

Parameters
number_of_channels (*integer*) – Number of channels

Returns
A keras layer

Return type
Layer

class antspynet.utilities.**AttentionLayer3D**(**args*, ***kwargs*)

Attention layer (3-D) from the self attention GAN

taken from the following python implementation

https://stackoverflow.com/questions/50819931/self-attention-gan-in-keras

based on the following paper:

https://arxiv.org/abs/1805.08318

Parameters
number_of_channels (*integer*) – Number of channels

Returns
A keras layer

Return type
Layer

Example

```
>>> input_shape = (100, 100, 3)
>>> input = Input(shape=input_shape)
>>> number_of_filters = 64
>>> outputs = Conv2D(filters=number_of_filters, kernel_size=2)(input)
>>> outputs = AttentionLayer2D(number_of_channels=number_of_filters)(outputs)
>>> model = Model(inputs=input, outputs=outputs)
```

2.7 Clustering

class antspynet.utilities.**DeepEmbeddedClustering**(**args*, ***kwargs*)

Deep embedded lustering layer.

Parameters

- **number_of_clusters** (*integer*) – Specifies which axis to normalize.
- **initial_cluster_weights** (*list*) – Initial clustering weights.
- **alpha** (*scalar*) – Parameter.

Returns
A keras layer

Return type
Keras layer

class antspynet.utilities.**DeepEmbeddedClusteringModel**(*number_of_units_per_layer=None*, *number_of_clusters=10*, *alpha=1.0*, *initializer='glorot_uniform'*, *convolutional=False*, *input_image_size=None*)

Deep embedded clustering with and without convolutions.

Parameters

- **number_of_units_per_layer** (*integer*) – Autoencoder number of units per layer.
- **number_of_clusters** (*integer*) – Number of clusters.
- **alpha** (*scalar*) – Parameter
- **initializer** (*string*) – Initializer for autoencoder.

Returns
A keras clustering model.

Return type
Keras model

2.8 Image patch

antspynet.utilities.**extract_image_patches**(*image*, *patch_size*, *max_number_of_patches='all'*, *stride_length=1*, *mask_image=None*, *random_seed=None*, *return_as_array=False*, *randomize=True*)

Extract 2-D or 3-D image patches.

Parameters

- **image** (*ANTsImage*) – Input image with one or more components.
- **patch_size** (*n-D tuple (depending on dimensionality).*) – Width, height, and depth (if 3-D) of patches.
- **max_number_of_patches** (*integer or string*) – Maximum number of patches returned. If "all" is specified, then all patches in sequence (defined by the stride_length are extracted.

- **stride_length** (*integer or n-D tuple*) – Defines the sequential patch overlap for max_number_of_patches = "all". Can be a image-dimensional vector or a scalar.
- **mask_image** (*ANTsImage (optional)*) – Optional image specifying the sampling region for the patches when max_number_of_patches does not equal "all". The way we constrain patch selection using a mask is by forcing each returned patch to have a masked voxel at its center.
- **random_seed** (*integer (optional)*) – Seed value that allows reproducible patch extraction across runs.
- **return_as_array** (*boolean*) – Specifies the return type of the function. If False (default) the return type is a list where each element is a single patch. Otherwise the return type is an array of size dim(number_of_patches, patch_size).
- **randomize** (*boolean*) – Boolean controlling whether we randomize indices when masking.

Return type
A list (or array) of patches.

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image_patches = extract_image_patches(image, patch_size=(32, 32))
```

antspynet.utilities.**reconstruct_image_from_patches**(*patches*, *domain_image*, *stride_length=1*, *domain_image_is_mask=False*)

Reconstruct image from a list of patches.

Parameters

- **patches** (*list or array of patches*) – List or array of patches defining an image. Patches are assumed to have the same format as returned by extract_image_patches.
- **domain_image** (*ANTs image*) – Image or mask to define the geometric information of the reconstructed image. If this is a mask image, the reconstruction will only use patches in the mask.
- **stride_length** (*integer or n-D tuple*) – Defines the sequential patch overlap for max_number_of_patches = "all". Can be a image-dimensional vector or a scalar.
- **domain_image_is_mask** (*boolean*) – Boolean specifying whether the domain image is a mask used to limit the region of reconstruction from the patches.

Return type
An ANTs image.

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image_patches = extract_image_patches(image, patch_size=(16, 16), stride_
↪length=4)
>>> reconstructed_image = reconstruct_image_from_patches(image_patches, image,␣
↪stride_length=4)
```

2.9 Super-resolution

antspynet.utilities.**mse**(*x*, *y=None*)

Mean square error of a single image or between two images.

Parameters

- **x** (*input image*) – ants input image
- **y** (*input image*) – ants input image

Return type
Value.

Example

```
>>> r16 = ants.image_read(ants.get_data("r16"))
>>> r64 = ants.image_read(ants.get_data("r64"))
>>> value = mse(r16, r64)
```

antspynet.utilities.**mae**(*x*, *y=None*)

Mean absolute error of a single image or between two images.

Parameters

- **x** (*input image*) – ants input image
- **y** (*input image*) – ants input image

Return type
Value

Example

```
>>> r16 = ants.image_read(ants.get_data("r16"))
>>> r64 = ants.image_read(ants.get_data("r64"))
>>> value = mae(r16, r64)
```

antspynet.utilities.**psnr**(*x*, *y*)

Peak signal-to-noise ratio between two images.

Parameters

- **x** (*input image*) – ants input image

- **y** (*input image*) – ants input image

Return type
Value

Example

```
>>> r16 = ants.image_read(ants.get_data("r16"))
>>> r64 = ants.image_read(ants.get_data("r64"))
>>> value = psnr(r16, r64)
```

antspynet.utilities.**ssim**(*x*, *y*, *K=(0.01, 0.03)*)

Structural similarity index (SSI) between two images.

Implementation of the SSI quantity for two images proposed in

Z. Wang, A.C. Bovik, H.R. Sheikh, E.P. Simoncelli. "Image quality assessment: from error visibility to structural similarity". IEEE TIP. 13 (4): 600–612.

Parameters

- **x** (*input image*) – ants input image
- **y** (*input image*) – ants input image
- **K** (*tuple of length 2*) – tuple which contain SSI parameters meant to stabilize the formula in case of weak denominators.

Return type
Value

Example

```
>>> r16 = ants.image_read(ants.get_data("r16"))
>>> r64 = ants.image_read(ants.get_data("r64"))
>>> value = psnr(r16, r64)
```

antspynet.utilities.**gmsd**(*x*, *y*)

Gradient magnitude similarity deviation

A fast and simple metric that correlates to perceptual quality.

Parameters

- **x** (*input image*) – ants input image
- **y** (*input image*) – ants input image

Return type
Value

Example

```
>>> r16 = ants.image_read(ants.get_data("r16"))
>>> r64 = ants.image_read(ants.get_data("r64"))
>>> value = gmsd(r16, r64)
```

antspynet.utilities.**apply_super_resolution_model_to_image**(*image*, *model*, *target_range=(-127.5, 127.5)*, *batch_size=32*, *regression_order=None*, *verbose=False*)

Apply a pretrained deep back projection model for super resolution. Helper function for applying a pretrained deep back projection model. Apply a patch-wise trained network to perform super-resolution. Can be applied to variable sized inputs. Warning: This function may be better used on CPU unless the GPU can accommodate the full image size. Warning 2: The global intensity range (min to max) of the output will match the input where the range is taken over all channels.

Parameters

- **image** (*ANTs image*) – input image.
- **model** (*keras object or string*) – pretrained keras model or filename.
- **target_range** (*2-element tuple*) – a tuple or array defining the (min, max) of the input image (e.g., -127.5, 127.5). Output images will be scaled back to original intensity. This range should match the mapping used in the training of the network.
- **batch_size** (*integer*) – Batch size used for the prediction call.
- **regression_order** (*integer*) – If specified, Apply the function regression_match_image with poly_order=regression_order.
- **verbose** (*boolean*) – If True, show status messages.

Return type

Super-resolution image upscaled to resolution specified by the network.

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image_sr = apply_super_resolution_model_to_image(image, get_pretrained_network(
↪"dbpn4x"))
```

2.10 Spatial transformer network

class antspynet.utilities.**SpatialTransformer2D**(**args*, ***kwargs*)

Custom layer for the spatial transfomer network.

Parameters

- **inputs** (*list of size 2*) – The first element are the images and the second element are the weights.
- **resampled_size** (*tuple of length 2*) – Size of the resampled output images.
- **transform_type** (*string*) – Transform type (default = 'affine').

- **interpolator_type** (*string*) – Interpolator type (default = 'linear').

Returns
A 2-D keras layer

Return type
Keras layer

class antspynet.utilities.**SpatialTransformer3D**(**args*, ***kwargs*)

Custom layer for the spatial transfomer network.

Parameters

- **inputs** (*list of size 2*) – The first element are the images and the second element are the weights.
- **resampled_size** (*tuple of length 3*) – Size of the resampled output images.
- **transform_type** (*string*) – Transform type (default = 'affine').
- **interpolator_type** (*string*) – Interpolator type (default = 'linear').

Returns
A 3-D keras layer

Return type
Keras layer

2.11 Applications

antspynet.utilities.**brain_extraction**(*image*, *modality*, *antsxnet_cache_directory=None*, *verbose=False*)

Perform brain extraction using U-net and ANTs-based training data. "NoBrainer" is also possible where brain extraction uses U-net and FreeSurfer training data ported from the

https://github.com/neuronets/nobrainer-models

Parameters

- **image** (*ANTsImage*) – input image (or list of images for multi-modal scenarios).
- **modality** (*string*) –

  Modality image type. Options include:

  – "t1": T1-weighted MRI—ANTs-trained. Previous versions are specified as "t1.v0", "t1.v1".

  – "t1nobrainer": T1-weighted MRI—FreeSurfer-trained: h/t Satra Ghosh and Jakub Kaczmarzyk.

  – **"t1combined": Brian's combination of "t1" and "t1nobrainer". One can also specify**
  "t1combined[X]" where X is the morphological radius. X = 12 by default.

  – "flair": FLAIR MRI. Previous versions are specified as "flair.v0".

  – "t2": T2 MRI. Previous versions are specified as "t2.v0".

  – "t2star": T2Star MRI.

  – "bold": 3-D mean BOLD MRI. Previous versions are specified as "bold.v0".

  – "fa": fractional anisotropy. Previous versions are specified as "fa.v0".

- ”t1t2infant”: Combined T1-w/T2-w infant MRI h/t Martin Styner.
- ”t1infant”: T1-w infant MRI h/t Martin Styner.
- ”t2infant”: T2-w infant MRI h/t Martin Styner.

- **antsxnet_cache_directory** (*string*) – Destination directory for storing the downloaded template and model weights. Since these can be reused, if is None, these data will be downloaded to a ~/.keras/ANTsXNet/.
- **verbose** (*boolean*) – Print progress to the screen.

Return type
ANTs probability brain mask image.

Example

```
>>> probability_brain_mask = brain_extraction(brain_image, modality="t1")
```

antspynet.utilities.**cortical_thickness**(*t1*, *antsxnet_cache_directory=None*, *verbose=False*)

Perform KellyKapowski cortical thickness using deep_atropos for segmentation. Description concerning implementaiton and evaluation:

https://www.medrxiv.org/content/10.1101/2020.10.19.20215392v1

Parameters

- **t1** (*ANTsImage*) – input 3-D unprocessed T1-weighted brain image.
- **antsxnet_cache_directory** (*string*) – Destination directory for storing the downloaded template and model weights. Since these can be reused, if is None, these data will be downloaded to a ~/.keras/ANTsXNet/.
- **verbose** (*boolean*) – Print progress to the screen.

Return type
Cortical thickness image and segmentation probability images.

Example

```
>>> image = ants.image_read("t1w_image.nii.gz")
>>> kk = cortical_thickness(image)
```

antspynet.utilities.**longitudinal_cortical_thickness**(*t1s*, *initial_template='oasis'*, *number_of_iterations=1*, *refinement_transform='antsRegistrationSyNQuick[a]'*, *antsxnet_cache_directory=None*, *verbose=False*)

Perform KellyKapowski cortical thickness longitudinally using code{deepAtropos} for segmentation of the derived single-subject template. It takes inspiration from the work described here:

https://pubmed.ncbi.nlm.nih.gov/31356207/

Parameters

- **t1s** (*list of ANTsImage*) – Input list of 3-D unprocessed t1-weighted brain images from a single subject.

- **initial_template** (*string or ANTsImage*) – Input image to define the orientation of the SST. Can be a string (see get_antsxnet_data) or a specified template. This allows the user to create a SST outside of this routine.
- **number_of_iterations** (*int*) – Defines the number of iterations for refining the SST.
- **refinement_transform** (*string*) – Transform for defining the refinement registration transform. See options in ants.registration.
- **antsxnet_cache_directory** (*string*) – Destination directory for storing the downloaded template and model weights. Since these can be reused, if is None, these data will be downloaded to a ~/.keras/ANTsXNet/.
- **verbose** (*boolean*) – Print progress to the screen.

Return type
Cortical thickness image and segmentation probability images.

Example

```
>>> t1s = list()
>>> t1s.append(ants.image_read("t1w_image.nii.gz"))
>>> kk = longitudinal_cortical_thickness(image)
```

antspynet.utilities.**lung_extraction**(*image*, *modality='proton'*, *antsxnet_cache_directory=None*, *verbose=False*)

Perform lung extraction.

Parameters

- **image** (*ANTsImage*) – input image
- **modality** (*string*) – Modality image type. Options include "ct", "proton", "protonLobes", "maskLobes", "ventilation", and "xray".
- **antsxnet_cache_directory** (*string*) – Destination directory for storing the downloaded template and model weights. Since these can be reused, if is None, these data will be downloaded to a ~/.keras/ANTsXNet/.
- **verbose** (*boolean*) – Print progress to the screen.

Return type
Dictionary of ANTs segmentation and probability images.

Example

```
>>> output = lung_extraction(lung_image, modality="proton")
```

antspynet.utilities.**preprocess_brain_image**(*image*, *truncate_intensity=(0.01, 0.99)*, *brain_extraction_modality=None*, *template_transform_type=None*, *template='biobank'*, *do_bias_correction=True*, *return_bias_field=False*, *do_denoising=True*, *intensity_matching_type=None*, *reference_image=None*, *intensity_normalization_type=None*, *antsxnet_cache_directory=None*, *verbose=True*)

Basic preprocessing pipeline for T1-weighted brain MRI

Standard preprocessing steps that have been previously described in various papers including the cortical thickness pipeline:

https://www.ncbi.nlm.nih.gov/pubmed/24879923

Parameters

- **image** (*ANTsImage*) – input image
- **truncate_intensity** (*2-length tuple*) – Defines the quantile threshold for truncating the image intensity
- **brain_extraction_modality** (*string or None*) – Perform brain extraction using antspynet tools. One of "t1", "t1v0", "t1nobrainer", "t1combined", "flair", "t2", "bold", "fa", "t1infant", "t2infant", or None.
- **template_transform_type** (*string*) – See details in help for ants.registration. Typically "Rigid" or "Affine".
- **template** (*ANTs image (not skull-stripped)*) – Alternatively, one can specify the default "biobank" or "croppedMni152" to download and use premade templates.
- **do_bias_correction** (*boolean*) – Perform N4 bias field correction.
- **return_bias_field** (*boolean*) – If True, return bias field as an additional output *without* bias correcting the preprocessed image.
- **do_denoising** (*boolean*) – Perform non-local means denoising.
- **intensity_matching_type** (*string*) – Either "regression" or "histogram". Only is performed if reference_image is not None.
- **reference_image** (*ANTs image*) – Reference image for intensity matching.
- **intensity_normalization_type** (*string*) – Either rescale the intensities to [0,1] (i.e., "01") or zero-mean, unit variance (i.e., "0mean"). If None normalization is not performed.
- **antsxnet_cache_directory** (*string*) – Destination directory for storing the downloaded template and model weights. Since these can be reused, if is None, these data will be downloaded to a ~/.keras/ANTsXNet/.
- **verbose** (*boolean*) – Print progress to the screen.

Returns

- *Dictionary with preprocessing information ANTs image (i.e., source_image) matched to the*
- *(reference_image).*

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> preprocessed_image = preprocess_brain_image(image, do_brain_extraction=False)
```

antspynet.utilities.**sysu_media_wmh_segmentation**(*flair*, *t1=None*, *use_ensemble=True*, *antsxnet_cache_directory=None*, *verbose=False*)

Perform WMH segmentation using the winning submission in the MICCAI 2017 challenge by the sysu_media team using FLAIR or T1/FLAIR. The MICCAI challenge is discussed in

https://pubmed.ncbi.nlm.nih.gov/30908194/

with the sysu_media's team entry is discussed in

https://pubmed.ncbi.nlm.nih.gov/30125711/

with the original implementation available here:

https://github.com/hongweilibran/wmh_ibbmTum

The original implementation used global thresholding as a quick brain extraction approach. Due to possible generalization difficulties, we leave such post-processing steps to the user. For brain or white matter masking see functions brain_extraction or deep_atropos, respectively.

Parameters

- **flair** (*ANTsImage*) – input 3-D FLAIR brain image (not skull-stripped).
- **t1** (*ANTsImage*) – input 3-D T1 brain image (not skull-stripped).
- **use_ensemble** (*boolean*) – check whether to use all 3 sets of weights.
- **antsxnet_cache_directory** (*string*) – Destination directory for storing the downloaded template and model weights. Since these can be reused, if is None, these data will be downloaded to a ~/.keras/ANTsXNet/.
- **verbose** (*boolean*) – Print progress to the screen.

Return type
WMH segmentation probability image

Example

```
>>> image = ants.image_read("flair.nii.gz")
>>> probability_mask = sysu_media_wmh_segmentation(image)
```

antspynet.utilities.**claustrum_segmentation**(*t1*, *do_preprocessing=True*, *use_ensemble=True*, *antsxnet_cache_directory=None*, *verbose=False*)

Claustrum segmentation

Described here:

https://pubmed.ncbi.nlm.nih.gov/34520080/

with the implementation available at:

https://github.com/hongweilibran/claustrum_multi_view

Parameters

- **t1** (*ANTsImage*) – input 3-D T1 brain image.
- **do_preprocessing** (*boolean*) – perform n4 bias correction.
- **use_ensemble** (*boolean*) – check whether to use all 3 sets of weights.
- **antsxnet_cache_directory** (*string*) – Destination directory for storing the downloaded template and model weights. Since these can be reused, if is None, these data will be downloaded to a ~/.keras/ANTsXNet/.
- **verbose** (*boolean*) – Print progress to the screen.

Return type
Claustrum segmentation probability image

Example

```
>>> image = ants.image_read("t1.nii.gz")
>>> probability_mask = claustrum_segmentation(image)
```

antspynet.utilities.**hypothalamus_segmentation**(*t1*, *antsxnet_cache_directory=None*, *verbose=False*)

Hypothalamus and subunits segmentation

Described here:

https://pubmed.ncbi.nlm.nih.gov/32853816/

ported from the original implementation

https://github.com/BBillot/hypothalamus_seg

Subunits labeling:

Label 1: left anterior-inferior Label 2: left anterior-superior Label 3: left posterior Label 4: left tubular inferior Label 5: left tubular superior Label 6: right anterior-inferior Label 7: right anterior-superior Label 8: right posterior Label 9: right tubular inferior Label 10: right tubular superior

Parameters

- **t1** (*ANTsImage*) – input 3-D T1 brain image.
- **antsxnet_cache_directory** (*string*) – Destination directory for storing the downloaded template and model weights. Since these can be reused, if is None, these data will be downloaded to a ~/.keras/ANTsXNet/.
- **verbose** (*boolean*) – Print progress to the screen.

Return type
Hypothalamus segmentation (and subunits) probability images

Example

```
>>> image = ants.image_read("t1.nii.gz")
>>> hypo = hypothalamus_segmentation(image)
```

antspynet.utilities.**hippmapp3r_segmentation**(*t1*, *do_preprocessing=True*, *antsxnet_cache_directory=None*, *verbose=False*)

Perform HippMapp3r (hippocampal) segmentation described in

https://www.ncbi.nlm.nih.gov/pubmed/31609046

with models and architecture ported from

https://github.com/mgoubran/HippMapp3r

Additional documentation and attribution resources found at

https://hippmapp3r.readthedocs.io/en/latest/

Preprocessing consists of:

- n4 bias correction and
- brain extraction

The input T1 should undergo the same steps. If the input T1 is the raw T1, these steps can be performed by the internal preprocessing, i.e. set do_preprocessing = True

Parameters

- **t1** (*ANTsImage*) – input image
- **do_preprocessing** (*boolean*) – See description above.
- **antsxnet_cache_directory** (*string*) – Destination directory for storing the downloaded template and model weights. Since these can be reused, if is None, these data will be downloaded to a ~/.keras/ANTsXNet/.
- **verbose** (*boolean*) – Print progress to the screen.

Return type
ANTs labeled hippocampal image.

Example

```
>>> mask = hippmapp3r_segmentation(t1)
```

antspynet.utilities.**deep_flash**(*t1*, *t2=None*, *do_preprocessing=True*, *use_rank_intensity=True*, *antsxnet_cache_directory=None*, *verbose=False*)

Hippocampal/Enthorhinal segmentation using "Deep Flash"

Perform hippocampal/entorhinal segmentation in T1 and T1/T2 images using labels from Mike Yassa's lab—https://faculty.sites.uci.edu/myassa/

https://www.nature.com/articles/s41598-024-59440-6

The labeling is as follows: Label 0 : background Label 5 : left aLEC Label 6 : right aLEC Label 7 : left pMEC Label 8 : right pMEC Label 9 : left perirhinal Label 10: right perirhinal Label 11: left parahippocampal Label 12: right parahippocampal Label 13: left DG/CA2/CA3/CA4 Label 14: right DG/CA2/CA3/CA4 Label 15: left CA1 Label 16: right CA1 Label 17: left subiculum Label 18: right subiculum

Preprocessing on the training data consisted of:

- n4 bias correction,
- affine registration to the "deep flash" template.

which is performed on the input images if do_preprocessing = True.

Parameters

- **t1** (*ANTsImage*) – raw or preprocessed 3-D T1-weighted brain image.
- **t2** (*ANTsImage*) – Optional 3-D T2-weighted brain image. If specified, it is assumed to be pre-aligned to the t1.

- **do_preprocessing** (*boolean*) – See description above.
- **use_rank_intensity** (*boolean*) – If false, use histogram matching with cropped template ROI. Otherwise, use a rank intensity transform on the cropped ROI.
- **antsxnet_cache_directory** (*string*) – Destination directory for storing the downloaded template and model weights. Since these can be reused, if is None, these data will be downloaded to a ~/.keras/ANTsXNet/.
- **verbose** (*boolean*) – Print progress to the screen.

Returns

- *List consisting of the segmentation image and probability images for*
- *each label and foreground.*

Example

```
>>> image = ants.image_read("t1.nii.gz")
>>> flash = deep_flash(image)
```

antspynet.utilities.**deep_atropos**(*t1*, *do_preprocessing=True*, *use_spatial_priors=1*, *antsxnet_cache_directory=None*, *verbose=False*)

Six-tissue segmentation.

Perform Atropos-style six tissue segmentation using deep learning.

The labeling is as follows: Label 0 : background Label 1 : CSF Label 2 : gray matter Label 3 : white matter Label 4 : deep gray matter Label 5 : brain stem Label 6 : cerebellum

Preprocessing on the training data consisted of:

- n4 bias correction,
- denoising,
- brain extraction, and
- affine registration to MNI.

The input T1 should undergo the same steps. If the input T1 is the raw T1, these steps can be performed by the internal preprocessing, i.e. set do_preprocessing = True

Parameters

- **t1** (*ANTsImage*) – raw or preprocessed 3-D T1-weighted brain image.
- **do_preprocessing** (*boolean*) – See description above.
- **use_spatial_priors** (*integer*) – Use MNI spatial tissue priors (0 or 1). Currently, only '0' (no priors) and '1' (cerebellar prior only) are the only two options. Default is 1.
- **antsxnet_cache_directory** (*string*) – Destination directory for storing the downloaded template and model weights. Since these can be reused, if is None, these data will be downloaded to a ~/.keras/ANTsXNet/.
- **verbose** (*boolean*) – Print progress to the screen.

Returns

- *List consisting of the segmentation image and probability images for*
- *each label.*

Example

```
>>> image = ants.image_read("t1.nii.gz")
>>> flash = deep_atropos(image)
```

antspynet.utilities.**desikan_killiany_tourville_labeling**(*t1*, *do_preprocessing=True*, *return_probability_images=False*, *do_lobar_parcellation=False*, *antsxnet_cache_directory=None*, *verbose=False*)

Cortical and deep gray matter labeling using Desikan-Killiany-Tourville

Perform DKT labeling using deep learning

The labeling is as follows:

Inner labels: Label 0: background Label 4: left lateral ventricle Label 5: left inferior lateral ventricle Label 6: left cerebellem exterior Label 7: left cerebellum white matter Label 10: left thalamus proper Label 11: left caudate Label 12: left putamen Label 13: left pallidium Label 14: 3rd ventricle Label 15: 4th ventricle Label 16: brain stem Label 17: left hippocampus Label 18: left amygdala Label 24: CSF Label 25: left lesion Label 26: left accumbens area Label 28: left ventral DC Label 30: left vessel Label 43: right lateral ventricle Label 44: right inferior lateral ventricle Label 45: right cerebellum exterior Label 46: right cerebellum white matter Label 49: right thalamus proper Label 50: right caudate Label 51: right putamen Label 52: right palladium Label 53: right hippocampus Label 54: right amygdala Label 57: right lesion Label 58: right accumbens area Label 60: right ventral DC Label 62: right vessel Label 72: 5th ventricle Label 85: optic chasm Label 91: left basal forebrain Label 92: right basal forebrain Label 630: cerebellar vermal lobules I-V Label 631: cerebellar vermal lobules VI-VII Label 632: cerebellar vermal lobules VIII-X

Outer labels: Label 1002: left caudal anterior cingulate Label 1003: left caudal middle frontal Label 1005: left cuneus Label 1006: left entorhinal Label 1007: left fusiform Label 1008: left inferior parietal Label 1009: left inferior temporal Label 1010: left isthmus cingulate Label 1011: left lateral occipital Label 1012: left lateral orbitofrontal Label 1013: left lingual Label 1014: left medial orbitofrontal Label 1015: left middle temporal Label 1016: left parahippocampal Label 1017: left paracentral Label 1018: left pars opercularis Label 1019: left pars orbitalis Label 1020: left pars triangularis Label 1021: left pericalcarine Label 1022: left postcentral Label 1023: left posterior cingulate Label 1024: left precentral Label 1025: left precuneus Label 1026: left rostral anterior cingulate Label 1027: left rostral middle frontal Label 1028: left superior frontal Label 1029: left superior parietal Label 1030: left superior temporal Label 1031: left supramarginal Label 1034: left transverse temporal Label 1035: left insula Label 2002: right caudal anterior cingulate Label 2003: right caudal middle frontal Label 2005: right cuneus Label 2006: right entorhinal Label 2007: right fusiform Label 2008: right inferior parietal Label 2009: right inferior temporal Label 2010: right isthmus cingulate Label 2011: right lateral occipital Label 2012: right lateral orbitofrontal Label 2013: right lingual Label 2014: right medial orbitofrontal Label 2015: right middle temporal Label 2016: right parahippocampal Label 2017: right paracentral Label 2018: right pars opercularis Label 2019: right pars orbitalis Label 2020: right pars triangularis Label 2021: right pericalcarine Label 2022: right postcentral Label 2023: right posterior cingulate Label 2024: right precentral Label 2025: right precuneus Label 2026: right rostral anterior cingulate Label 2027: right rostral middle frontal Label 2028: right superior frontal Label 2029: right superior parietal Label 2030: right superior temporal Label 2031: right supramarginal Label 2034: right transverse temporal Label 2035: right insula

Performing the lobar parcellation is based on the FreeSurfer division described here:

See https://surfer.nmr.mgh.harvard.edu/fswiki/CorticalParcellation

Frontal lobe: Label 1002: left caudal anterior cingulate Label 1003: left caudal middle frontal Label 1012: left lateral orbitofrontal Label 1014: left medial orbitofrontal Label 1017: left paracentral Label 1018: left pars opercularis Label 1019: left pars orbitalis Label 1020: left pars triangularis Label 1024: left precentral Label 1026: left rostral anterior cingulate Label 1027: left rostral middle frontal Label 1028: left superior frontal Label 2002: right caudal anterior cingulate Label 2003: right caudal middle frontal Label 2012: right

lateral orbitofrontal Label 2014: right medial orbitofrontal Label 2017: right paracentral Label 2018: right pars opercularis Label 2019: right pars orbitalis Label 2020: right pars triangularis Label 2024: right precentral Label 2026: right rostral anterior cingulate Label 2027: right rostral middle frontal Label 2028: right superior frontal

Parietal: Label 1008: left inferior parietal Label 1010: left isthmus cingulate Label 1022: left postcentral Label 1023: left posterior cingulate Label 1025: left precuneus Label 1029: left superior parietal Label 1031: left supramarginal Label 2008: right inferior parietal Label 2010: right isthmus cingulate Label 2022: right postcentral Label 2023: right posterior cingulate Label 2025: right precuneus Label 2029: right superior parietal Label 2031: right supramarginal

Temporal: Label 1006: left entorhinal Label 1007: left fusiform Label 1009: left inferior temporal Label 1015: left middle temporal Label 1016: left parahippocampal Label 1030: left superior temporal Label 1034: left transverse temporal Label 2006: right entorhinal Label 2007: right fusiform Label 2009: right inferior temporal Label 2015: right middle temporal Label 2016: right parahippocampal Label 2030: right superior temporal Label 2034: right transverse temporal

Occipital: Label 1005: left cuneus Label 1011: left lateral occipital Label 1013: left lingual Label 1021: left pericalcarine Label 2005: right cuneus Label 2011: right lateral occipital Label 2013: right lingual Label 2021: right pericalcarine

Other outer labels: Label 1035: left insula Label 2035: right insula

Preprocessing on the training data consisted of:

- n4 bias correction,
- denoising,
- brain extraction, and
- affine registration to MNI.

The input T1 should undergo the same steps. If the input T1 is the raw T1, these steps can be performed by the internal preprocessing, i.e. set do_preprocessing = True

Parameters

- **t1** (*ANTsImage*) – raw or preprocessed 3-D T1-weighted brain image.
- **do_preprocessing** (*boolean*) – See description above.
- **return_probability_images** (*boolean*) – Whether to return the two sets of probability images for the inner and outer labels.
- **do_lobar_parcellation** (*boolean*) – Perform lobar parcellation (also divided by hemisphere).
- **antsxnet_cache_directory** (*string*) – Destination directory for storing the downloaded template and model weights. Since these can be reused, if is None, these data will be downloaded to a ~/.keras/ANTsXNet/.
- **verbose** (*boolean*) – Print progress to the screen.

Returns

- *List consisting of the segmentation image and probability images for*
- *each label.*

Example

```
>>> image = ants.image_read("t1.nii.gz")
>>> dkt = desikan_killiany_tourville_labeling(image)
```

antspynet.utilities.**brain_age**(*t1*, *do_preprocessing=True*, *number_of_simulations=0*, *sd_affine=0.01*, *antsxnet_cache_directory=None*, *verbose=False*)

Estimate BrainAge from a T1-weighted MR image using the DeepBrainNet architecture and weights described here:

https://github.com/vishnubashyam/DeepBrainNet

and described in the following article:

https://pubmed.ncbi.nlm.nih.gov/32591831/

Preprocessing on the training data consisted of:

- n4 bias correction,
- brain extraction, and
- affine registration to MNI.

The input T1 should undergo the same steps. If the input T1 is the raw T1, these steps can be performed by the internal preprocessing, i.e. set do_preprocessing = True

Parameters

- **t1** (*ANTsImage*) – raw or preprocessed 3-D T1-weighted brain image.
- **do_preprocessing** (*boolean*) – See description above.
- **number_of_simulations** (*integer*) – Number of random affine perturbations to transform the input.
- **sd_affine** (*float*) – Define the standard deviation of the affine transformation parameter.
- **antsxnet_cache_directory** (*string*) – Destination directory for storing the downloaded template and model weights. Since these can be reused, if is None, these data will be downloaded to a ~/.keras/ANTsXNet/.
- **verbose** (*boolean*) – Print progress to the screen.

Returns

- *List consisting of the segmentation image and probability images for*
- *each label.*

Example

```
>>> image = ants.image_read("t1.nii.gz")
>>> deep = brain_age(image)
>>> print("Predicted age: ", deep['predicted_age']
```

antspynet.utilities.**mri_super_resolution**(*image*, *antsxnet_cache_directory=None*, *verbose=False*)

Perform super-resolution (2x) of MRI data using deep back projection network.

Parameters

- **image** (*ANTsImage*) – magnetic resonance image
- **antsxnet_cache_directory** (*string*) – Destination directory for storing the downloaded template and model weights. Since these can be reused, if is None, these data will be downloaded to a ~/.keras/ANTsXNet/.
- **verbose** (*boolean*) – Print progress to the screen.

Return type
The super-resolved image.

Example

```
>>> image = ants.image_read("t1.nii.gz")
>>> image_sr = mri_super_resolution(image)
```

antspynet.utilities.**tid_neural_image_assessment**(*image*, *mask=None*, *patch_size=101*, *stride_length=None*, *padding_size=0*, *dimensions_to_predict=0*, *antsxnet_cache_directory=None*, *which_model='tidsQualityAssessment'*, *image_scaling=[255, 127.5]*, *do_patch_scaling=False*, *no_reconstruction=False*, *verbose=False*)

Perform MOS-based assessment of an image.

Use a ResNet architecture to estimate image quality in 2D or 3D using subjective QC image databases described in

https://www.sciencedirect.com/science/article/pii/S0923596514001490

or

https://doi.org/10.1109/TIP.2020.2967829

where the image assessment is either "global", i.e., a single number or an image based on the specified patch size. In the 3-D case, neighboring slices are used for each estimate. Note that parameters should be kept as consistent as possible in order to enable comparison. Patch size should be roughly 1/12th to 1/4th of image size to enable locality. A global estimate can be gained by setting patch_size = "global".

Parameters

- **image** (*ANTsImage (2-D or 3-D)*) – input image.
- **mask** (*ANTsImage (2-D or 3-D)*) – optional mask for designating calculation ROI.
- **patch_size** (*integer*) – prime number of patch_size. 101 is good. Otherwise, choose "global" for a single global estimate of quality.
- **stride_length** (*integer or vector of image dimension length*) – optional value to speed up computation (typically less than patch size).
- **padding_size** (*positive or negative integer or vector of image dimension length*) – de(padding) to remove edge effects.
- **dimensions_to_predict** (*integer or vector*) – if image dimension is 3, this parameter specifies which dimensions should be used for prediction. If more than one dimension is specified, the results are averaged.

- **antsxnet_cache_directory** (*string*) – Destination directory for storing the downloaded template and model weights. Since these can be reused, if is None, these data will be downloaded to ~/.keras/ANTsXNet/.
- **which_model** (*string or tf/keras model*) – model type e.g. string tidsQualityAssessment, koniqMS, koniqMS2 or koniqMS3 where the former predicts mean opinion score (MOS) and MOS standard deviation and the latter koniq models predict mean opinion score (MOS) and sharpness. passing a user-defined model is also valid.
- **image_scaling** (*a two-vector where the first value is the multiplier and the*) – second value the subtractor so each image will be scaled as img = ants.iMath(img,"Normalize")*m - s.
- **do_patch_scaling** (*boolean controlling whether each patch is scaled or*) – (if False) only a global scaling of the image is used.
- **no_reconstruction** (*boolean reconstruction is time consuming - turn this on*) – if you just want the predicted values
- **verbose** (*boolean*) – Print progress to the screen.

Returns

- *List of QC results predicting both both human rater's mean and standard*
- *deviation of the MOS ("mean opinion scores") or sharpness depending on the*
- *selected network. Both aggregate and spatial scores are returned, the latter*
- *in the form of an image.*

Example

```
>>> image = ants.image_read(ants.get_data("r16"))
>>> mask = ants.get_mask(image)
>>> tid = tid_neural_image_assessment(image, mask=mask, patch_size=101, stride_
↪length=7)
```

antspynet.utilities.**neural_style_transfer**(*content_image*, *style_images*, *initial_combination_image=None*, *number_of_iterations=10*, *learning_rate=1.0*, *total_variation_weight=8.5e-05*, *content_weight=0.025*, *style_image_weights=1.0*, *content_layer_names=['block5_conv2']*, *style_layer_names='all'*, *content_mask=None*, *style_masks=None*, *use_shifted_activations=True*, *use_chained_inference=True*, *verbose=False*, *output_prefix=None*)

The popular neural style transfer described here:

https://arxiv.org/abs/1508.06576 and https://arxiv.org/abs/1605.04603

and taken from François Chollet's implementation

https://keras.io/examples/generative/neural_style_transfer/

and titu1994's modifications:

https://github.com/titu1994/Neural-Style-Transfer

in order to possibly modify and experiment with medical images.

Parameters

- **content_image** (*ANTsImage (1 or 3-component)*) – Content (or base) image.
- **style_images** (*ANTsImage or list of ANTsImages*) – Style (or reference) image.
- **initial_combination_image** (*ANTsImage (1 or 3-component)*) – Starting point for the optimization. Allows one to start from the output from a previous run. Otherwise, start from the content image. Note that the original paper starts with a noise image.
- **number_of_iterations** (*integer*) – Number of gradient steps taken during optimization.
- **learning_rate** (*float*) – Parameter for Adam optimization.
- **total_variation_weight** (*float*) – A penalty on the regularization term to keep the features of the output image locally coherent.
- **content_weight** (*float*) – Weight of the content layers in the optimization function.
- **style_image_weights** (*float or list of floats*) – Weights of the style term in the optimization function for each style image. Can either specify a single scalar to be used for all the images or one for each image. The style term computes the sum of the L2 norm between the Gram matrices of the different layers (using ImageNet-trained VGG) of the style and content images.
- **content_layer_names** (*list of strings*) – Names of VGG layers from which to compute the content loss.
- **style_layer_names** (*list of strings*) – Names of VGG layers from which to compute the style loss. If "all", the layers used are ['block1_conv1', 'block1_conv2', 'block2_conv1', 'block2_conv2', 'block3_conv1', 'block3_conv2', 'block3_conv3', 'block3_conv4', 'block4_conv1', 'block4_conv2', 'block4_conv3', 'block4_conv4', 'block5_conv1', 'block5_conv2', 'block5_conv3', 'block5_conv4']. This is a proposed improvement from https://arxiv.org/abs/1605.04603. In the original implementation, the layers used are: ['block1_conv1', 'block2_conv1', 'block3_conv1',

  'block4_conv1', 'block5_conv1'].
- **content_mask** (*ANTsImage*) – Specify the region for content consideration.
- **style_masks** (*ANTsImage or list of ANTsImages*) – Specify the region for style consideration.
- **use_shifted_activations** (*boolean*) – Use shifted activations in calculating the Gram matrix (improvement mentioned in https://arxiv.org/abs/1605.04603).
- **use_chained_inference** (*boolean*) – Another proposed improvement from https://arxiv.org/abs/1605.04603.
- **verbose** (*boolean*) – Print progress to the screen.
- **output_prefix** (*string*) – If specified, outputs a png image to disk at each iteration.

Return type

ANTs 3-component image.

Example

```
>>> image = neural_style_transfer(content_image, style_image)
```

antspynet.utilities.**el_bicho**(*ventilation_image*, *mask*, *use_coarse_slices_only=True*, *antsxnet_cache_directory=None*, *verbose=False*)

Perform functional lung segmentation using hyperpolarized gases.

https://pubmed.ncbi.nlm.nih.gov/30195415/

Parameters

- **ventilation_image** (*ANTsImage*) – input ventilation image.
- **mask** (*ANTsImage*) – input mask.
- **use_coarse_slices_only** (*boolean*) – If True, apply network only in the dimension of greatest slice thickness. If False, apply to all dimensions and average the results.
- **antsxnet_cache_directory** (*string*) – Destination directory for storing the downloaded template and model weights. Since these can be reused, if is None, these data will be downloaded to a ~/.keras/ANTsXNet/.
- **verbose** (*boolean*) – Print progress to the screen.

Return type

Ventilation segmentation and corresponding probability images

Example

```
>>> image = ants.image_read("ventilation.nii.gz")
>>> mask = ants.image_read("mask.nii.gz")
>>> lung_seg = el_bicho(image, mask, use_coarse_slices=True, verbose=False)
```

antspynet.utilities.**arterial_lesion_segmentation**(*image*, *antsxnet_cache_directory=None*, *verbose=False*)

Perform arterial lesion segmentation using U-net.

Parameters

- **image** (*ANTsImage*) – input image
- **antsxnet_cache_directory** (*string*) – Destination directory for storing the downloaded template and model weights. Since these can be reused, if is None, these data will be downloaded to a ~/.keras/ANTsXNet/.
- **verbose** (*boolean*) – Print progress to the screen.

Return type

Foreground probability image.

Example

```
>>> output = arterial_lesion_segmentation(histology_image)
```

2.12 Miscellaneous

antspynet.utilities.**get_pretrained_network**(*file_id=None*, *target_file_name=None*, *antsxnet_cache_directory=None*)

Download pretrained network/weights.

Parameters

- **string** (*antsxnet_cache_directory*) – One of the permitted file ids or pass "show" to list all valid possibilities. Note that most require internet access to download.
- **string** – Optional target filename.
- **string** – Optional target output. If not specified these data will be downloaded to the subdirectory ~/.keras/ANTsXNet/.

Return type
A filename string

Example

```
>>> model_file = get_pretrained_network('brainExtraction')
```

antspynet.utilities.**get_antsxnet_data**(*file_id=None*, *target_file_name=None*, *antsxnet_cache_directory=None*)

Download data such as prefabricated templates and spatial priors.

Parameters

- **string** (*antsxnet_cache_directory*) – One of the permitted file ids or pass "show" to list all valid possibilities. Note that most require internet access to download.
- **string** – Optional target filename.
- **string** – Optional target output. If not specified these data will be downloaded to the subdirectory ~/.keras/ANTsXNet/.

Return type
A filename string

Example

```
>>> template_file = get_antsxnet_data('biobank')
```

class antspynet.utilities.**Scale**(**args*, ***kwargs*)

Custom layer used in the Dense U-net class for normalization which learns a set of weights and biases for scaling the input data.

Parameters

- **axis** (*integer*) – Specifies which axis to normalize.
- **momentum** (*scalar*) – Value used for computation of the exponential average of the mean and standard deviation.

antspynet.utilities.**regression_match_image**(*source_image*, *reference_image*, *mask=None*, *poly_order=1*, *truncate=True*)

Image intensity normalization using linear regression.

Parameters

- **source_image** (*ANTsImage*) – Image whose intensities are matched to the reference image.
- **reference_image** (*ANTsImage*) – Defines the reference intensity function.
- **poly_order** (*integer*) – Polynomial order of fit. Default is 1 (linear fit).
- **mask** (*ANTsImage*) – Defines voxels for regression modeling.
- **truncate** (*boolean*) – Turns on/off the clipping of intensities.

Return type
ANTs image (i.e., source_image) matched to the (reference_image).

Example

```
>>> import ants
>>> source_image = ants.image_read(ants.get_ants_data('r16'))
>>> reference_image = ants.image_read(ants.get_ants_data('r64'))
>>> matched_image = regression_match_image(source_image, reference_image)
```

antspynet.utilities.**randomly_transform_image_data**(*reference_image*, *input_image_list*, *segmentation_image_list=None*, *number_of_simulations=10*, *transform_type='affine'*, *sd_affine=0.02*, *deformation_transform_type='bspline'*, *number_of_random_points=1000*, *sd_noise=10.0*, *number_of_fitting_levels=4*, *mesh_size=1*, *sd_smoothing=4.0*, *input_image_interpolator='linear'*, *segmentation_image_interpolator='nearestNeighbor'*)

Randomly transform image data (optional: with corresponding segmentations).

Apply rigid, affine and/or deformable maps to an input set of training images. The reference image domain defines the space in which this happens.

Parameters

- **reference_image** (*ANTsImage*) – Defines the spatial domain for all output images. If the input images do not match the spatial domain of the reference image, we internally resample the target to the reference image. This could have unexpected consequences. Resampling to the reference domain is performed by testing using ants.image_physical_space_consistency then calling ants.resample_image_to_target with failure.
- **input_image_list** (*list of lists of ANTsImages*) – List of lists of input images to warp. The internal list sets contain one or more images (per subject) which are assumed

to be mutually aligned. The outer list contains multiple subject lists which are randomly sampled to produce output image list.

- **segmentation_image_list** (*list of ANTsImages*) – List of segmentation images corresponding to the input image list (optional).
- **number_of_simulations** (*integer*) – Number of simulated output image sets.
- **transform_type** (*string*) – One of the following options: "translation", "rigid", "scaleShear", "affine", "deformation", "affineAndDeformation".
- **sd_affine** (*float*) – Parameter dictating deviation amount from identity for random linear transformations.
- **deformation_transform_type** (*string*) – "bspline" or "exponential".
- **number_of_random_points** (*integer*) – Number of displacement points for the deformation field.
- **sd_noise** (*float*) – Standard deviation of the displacement field.
- **number_of_fitting_levels** (*integer*) – Number of fitting levels (bspline deformation only).
- **mesh_size** (*int or n-D tuple*) – Determines fitting resolution (bspline deformation only).
- **sd_smoothing** (*float*) – Standard deviation of the Gaussian smoothing in mm (exponential field only).
- **input_image_interpolator** (*string*) – One of the following options: "nearestNeighbor", "linear", "gaussian", "bSpline".
- **segmentation_image_interpolator** (*string*) – Only "nearestNeighbor" is currently available.

Return type
list of lists of transformed images

Example

```
>>> import ants
>>> image1_list = list()
>>> image1_list.append(ants.image_read(ants.get_ants_data("r16")))
>>> image2_list = list()
>>> image2_list.append(ants.image_read(ants.get_ants_data("r64")))
>>> input_segmentations = list()
>>> input_segmentations.append(ants.threshold_image(image1, "Otsu", 3))
>>> input_segmentations.append(ants.threshold_image(image2, "Otsu", 3))
>>> input_images = list()
>>> input_images.append(image1_list)
>>> input_images.append(image2_list)
>>> data = antspynet.randomly_transform_image_data(image1,
>>>     input_images, input_segmentations, sd_affine=0.02,
>>>     transform_type = "affineAndDeformation" )
```

antspynet.utilities.**data_augmentation**(*input_image_list*, *segmentation_image_list=None*, *pointset_list=None*, *number_of_simulations=10*, *reference_image=None*, *transform_type='affineAndDeformation'*, *noise_model='additivegaussian'*, *noise_parameters=(0.0, 0.05)*, *sd_simulated_bias_field=1.0*, *sd_histogram_warping=0.05*, *sd_affine=0.05*, *sd_deformation=0.2*, *output_numpy_file_prefix=None*, *verbose=False*)

Randomly transform image data.

Given an input image list (possibly multi-modal) and an optional corresponding segmentation image list, this function will perform data augmentation with the following augmentation possibilities:

- spatial transformations
- added image noise
- simulated bias field
- histogram warping

Parameters

- **input_image_list** (*list of lists of ANTsImages*) – List of lists of input images to warp. The internal list sets contain one or more images (per subject) which are assumed to be mutually aligned. The outer list contains multiple subject lists which are randomly sampled to produce output image list.
- **segmentation_image_list** (*list of ANTsImages*) – List of segmentation images corresponding to the input image list (optional).
- **pointset_list** (*list of pointsets*) – Numpy arrays corresponding to the input image list (optional). If using this option, the transform_type must be invertible.
- **number_of_simulations** (*integer*) – Number of simulated output image sets. Default = 10.
- **reference_image** (*ANTsImage*) – Defines the spatial domain for all output images. If one is not specified, we used the first image in the input image list.
- **transform_type** (*string*) – One of the following options: "translation", "rigid", "scaleShear", "affine", "deformation", "affineAndDeformation".
- **noise_model** (*string*) – 'additivegaussian', 'saltandpepper', 'shot', and 'speckle'. Alternatively, one can specify a tuple or list of one or more of the options and one is selected at random with reasonable, randomized parameters. Note that the "speckle" model takes much longer than the others.
- **noise_parameters** (*tuple or array or float*) – 'additivegaussian': (mean, standardDeviation) 'saltandpepper': (probability, saltValue, pepperValue) 'shot': scale 'speckle': standardDeviation Note that the standard deviation, scale, and probability values are *max* values and are randomly selected in the range [0, noise_parameter]. Also, the "mean", "saltValue" and "pepperValue" are assumed to be in the intensity normalized range of [0, 1].
- **sd_simulated_bias_field** (*float*) – Characterize the standard deviation of the amplitude.
- **sd_histogram_warping** (*float*) – Determines the strength of the bias field.
- **sd_affine** (*float*) – Determines the amount of affine transformation.
- **sd_deformation** (*float*) – Determines the amount of deformable transformation.

- **output_numpy_file_prefix** (*string*) – Filename of output numpy array containing all the simulated images and segmentations.

Return type
list of lists of transformed images and/or outputs to a numpy array.

Example

```
>>> image1_list = list()
>>> image1_list.append(ants.image_read(ants.get_ants_data("r16")))
>>> image2_list = list()
>>> image2_list.append(ants.image_read(ants.get_ants_data("r64")))
>>> segmentation1 = ants.threshold_image(image1_list[0], "Otsu", 3)
>>> segmentation2 = ants.threshold_image(image2_list[0], "Otsu", 3)
>>> input_segmentations = list()
>>> input_segmentations.append(segmentation1)
>>> input_segmentations.append(segmentation2)
>>> points1 = ants.get_centroids(segmentation1)[:,0:2]
>>> points2 = ants.get_centroids(segmentation2)[:,0:2]
>>> input_points = list()
>>> input_points.append(points1)
>>> input_points.append(points2)
>>> input_images = list()
>>> input_images.append(image1_list)
>>> input_images.append(image2_list)
>>> data = data_augmentation(input_images,
                             input_segmentations,
                             input_points,
                             tranform_type="scaleShear")
```

antspynet.utilities.**histogram_warp_image_intensities**(*image*, *break_points=(0.25, 0.5, 0.75)*, *displacements=None*, *clamp_end_points=(False, False)*, *sd_displacements=0.05*, *transform_domain_size=20*)

Transform image intensities based on histogram mapping.

Apply B-spline 1-D maps to an input image for intensity warping.

Parameters

- **image** (*ANTsImage*) – Input image.
- **break_points** (*integer or tuple*) – Parametric points at which the intensity transform displacements are specified between [0, 1]. Alternatively, a single number can be given and the sequence is linearly spaced in [0, 1].
- **displacements** (*tuple*) – displacements to define intensity warping. Length must be equal to the breakPoints. Alternatively, if None random displacements are chosen (random normal: mean = 0, sd = sd_displacements).
- **sd_displacements** (*float*) – Characterize the randomness of the intensity displacement.
- **clamp_end_points** (*2-element tuple of booleans*) – Specify non-zero intensity change at the ends of the histogram.

- **transform_domain_size** (*integer*) – Defines the sampling resolution of the B-spline warping.

Return type
ANTs image

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data("r64"))
>>> transformed_image = histogram_warp_image_intensities( image )
```

antspynet.utilities.**simulate_bias_field**(*domain_image*, *number_of_points=10*, *sd_bias_field=1.0*, *number_of_fitting_levels=4*, *mesh_size=1*)

Simulate random bias field

Low frequency, spatial varying simulated random bias field using random points and B-spline fitting.

Parameters

- **domain_image** (*ANTsImage*) – Image to define the spatial domain of the bias field.
- **number_of_points** (*integer*) – Number of randomly defined points to define the bias field (default = 10).
- **sd_bias_field** (*float*) – Characterize the standard deviation of the amplitude (default = 1).
- **number_of_fitting_levels** (*integer*) – B-spline fitting parameter.

Return type
ANTs image

Example

```
>>> import ants
>>> import numpy as np
>>> image = ants.image_read(ants.get_ants_data("r64"))
>>> log_field = simulate_bias_field(image, number_of_points=10, sd_bias_field=1.0,
...     number_of_fitting_levels=2, mesh_size=10)
>>> log_field = log_field.iMath("Normalize")
>>> field_array = np.power(np.exp(log_field.numpy()), 4)
>>> image = image * ants.from_numpy(field_array, origin=image.origin,
...     spacing=image.spacing, direction=image.direction)
```

antspynet.utilities.**crop_image_center**(*image*, *crop_size*)

Crop the center of an image.

Parameters

- **image** (*ANTsImage*) – Input image
- **crop_size** (*n-D tuple*) – Width, height, depth (if 3-D), and time (if 4-D) of crop region.

Return type
ANTs image.

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> cropped_image = crop_image_center(image, crop_size=(64, 64))
```

antspynet.utilities.**pad_or_crop_image_to_size**(*image*, *size*)

Pad or crop an image to a specified size

Parameters

- **image** (*ANTsImage*) – Input image
- **size** (*tuple*) – size of output image

Return type
A cropped or padded image

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> padded_image = pad_or_crop_image_to_size(image, (333, 333))
```

antspynet.utilities.**pad_image_by_factor**(*image*, *factor*)

Pad an image based on a factor.

Pad image of size (x, y, z) to (x', y', z') where (x', y', z') is a divisible by a user-specified factor.

Parameters

- **image** (*ANTsImage*) – Input image
- **factor** (*scalar or n-D tuple*) – Padding factor

Return type
ANTs image.

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> padded_image = pad_image_by_factor(image, factor=4)
```

antspynet.utilities.**encode_unet**(*segmentations_array*, *segmentation_labels=None*)

Basic one-hot transformation of segmentations array

Parameters

- **segmentations_array** (*numpy array*) – multi-label numpy array
- **segmentation_labels** (*tuple or list*) – Note that a background label (typically 0) needs to be included.

Return type
An n-d array of shape batch_size x width x height x <depth> x number_of_segmentation_labels

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> seg = ants.kmeans_segmentation(image, 3)['segmentation']
>>> one_hot = encode_unet(seg.numpy().astype('int'))
```

antspynet.utilities.**decode_unet**(*y_predicted*, *domain_image*)

Decoding function for the u-net prediction outcome

Parameters

- **y_predicted** (*an array*) – Shape batch_size x width x height x <depth> x number_of_segmentation_labels
- **domain_image** (*ANTs image*) – Defines the geometry of the returned probability images

Return type

List of probability images.

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
```

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

D

E

G

H

I

L

M

N

P

R

S

T

V

W