Transform#

An abstract class of a Transform. A transform is callable that processes data. It could be stateful and may modify data in place, the implementation should be aware of: thread safety when mutating its own states. When used from a multi-process context, transform's instance variables are read-only. thread-unsafe transforms should inherit monai.transforms.ThreadUnsafe. data content unused by this transform may still be used in the subsequent transforms in a composed transform. storing too much information in data may cause some memory issue or IPC sync issue, especially in the multi-processing environment of PyTorch DataLoader. See Also monai.transforms.Compose data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method.

MapTransform#

A subclass of monai.transforms.Transform with an assumption that the data input of self.__call__ is a MutableMapping such as dict. The keys parameter will be used to get and set the actual data item to transform. That is, the callable of this transform should follow the pattern: ValueError – When keys is an empty iterable. TypeError – When keys type is not in Union[Hashable, Iterable[Hashable]]. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. An updated dictionary version of data by applying the transform. This function is to be called after every self.__call__(data), update data[key_transforms] and data[key_meta_dict] using the content from MetaTensor data[key], for MetaTensor backward compatibility 0.9.0. Get the first available key of self.keys in the input data dictionary. If no available key, return an empty tuple (). data (Dict[Hashable, Any]) – data that the transform will be applied to. Iterate across keys and optionally extra iterables. If key is missing, exception is raised if allow_missing_keys==False (default). If allow_missing_keys==True, key is skipped. data (Mapping[Hashable, Any]) – data that the transform will be applied to extra_iterables (Optional[Iterable]) – anything else to be iterated through Generator

RandomizableTrait#

An interface to indicate that the transform has the capability to perform randomized transforms to the data that it is called upon. This interface can be extended from by people adapting transforms to the MONAI framework as well as by implementors of MONAI transforms.

LazyTrait#

An interface to indicate that the transform has the capability to execute using MONAI's lazy resampling feature. In order to do this, the implementing class needs to be able to describe its operation as an affine matrix or grid with accompanying metadata. This interface can be extended from by people adapting transforms to the MONAI framework as well as by implementors of MONAI transforms. Get whether lazy_evaluation is enabled for this transform instance. :returns: True if the transform is operating in a lazy fashion, False if not.

MultiSampleTrait#

An interface to indicate that the transform has the capability to return multiple samples given an input, such as when performing random crops of a sample. This interface can be extended from by people adapting transforms to the MONAI framework as well as by implementors of MONAI transforms.

Randomizable#

An interface for handling random state locally, currently based on a class variable R, which is an instance of np.random.RandomState. This provides the flexibility of component-specific determinism without affecting the global states. It is recommended to use this API with monai.data.DataLoader for deterministic behaviour of the preprocessing pipelines. This API is not thread-safe. Additionally, deepcopying instance of this class often causes insufficient randomness as the random states will be duplicated. Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. Randomizable a Randomizable instance.

LazyTransform#

An implementation of functionality for lazy transforms that can be subclassed by array and dictionary transforms to simplify implementation of new lazy transforms. Get whether lazy_evaluation is enabled for this transform instance. :returns: True if the transform is operating in a lazy fashion, False if not.

RandomizableTransform#

An interface for handling random state locally, currently based on a class variable R, which is an instance of np.random.RandomState. This class introduces a randomized flag _do_transform, is mainly for randomized data augmentation transforms. For example: Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None

Compose#

Compose provides the ability to chain a series of callables together in a sequential manner. Each transform in the sequence must take a single argument and return a single value. Compose can be used in two ways: With a series of transforms that accept and return a single ndarray / tensor / tensor-like parameter. With a series of transforms that accept and return a dictionary that contains one or more parameters. Such transforms must have pass-through semantics that unused values in the dictionary must be copied to the return dictionary. It is required that the dictionary is copied between input and output of each transform. If some transform takes a data item dictionary as input, and returns a sequence of data items in the transform chain, all following transforms will be applied to each item of this list if map_items is True (the default). If map_items is False, the returned sequence is passed whole to the next callable in the chain. For example: A Compose([transformA, transformB, transformC], map_items=True)(data_dict) could achieve the following patch-based transformation on the data_dict input: transformA normalizes the intensity of ‘img’ field in the data_dict. transformB crops out image patches from the ‘img’ and ‘seg’ of data_dict, and return a list of three patch samples: transformC then randomly rotates or flips ‘img’ and ‘seg’ of each dictionary item in the list returned by transformB. The composed transforms will be set the same global random seed if user called set_determinism(). When using the pass-through dictionary operation, you can make use of monai.transforms.adaptors.adaptor to wrap transforms that don’t conform to the requirements. This approach allows you to use transforms from otherwise incompatible libraries with minimal additional work. Note In many cases, Compose is not the best way to create pre-processing pipelines. Pre-processing is often not a strictly sequential series of operations, and much of the complexity arises when a not-sequential set of functions must be called as if it were a sequence. Example: images and labels Images typically require some kind of normalization that labels do not. Both are then typically augmented through the use of random rotations, flips, and deformations. Compose can be used with a series of transforms that take a dictionary that contains ‘image’ and ‘label’ entries. This might require wrapping torchvision transforms before passing them to compose. Alternatively, one can create a class with a __call__ function that calls your pre-processing functions taking into account that not all of them are called on the labels. transforms (Union[Sequence[Callable], Callable, None]) – sequence of callables. map_items (bool) – whether to apply transform to each item in the input data if data is a list or tuple. defaults to True. unpack_items (bool) – whether to unpack input data with * as parameters for the callable function of transform. defaults to False. log_stats (bool) – whether to log the detailed information of data and applied transform when error happened, for NumPy array and PyTorch Tensor, log the data shape and value range, for other metadata, log the values directly. default to False. Call self as a function. Return a Composition with a simple list of transforms, as opposed to any nested Compositions. e.g., t1 =

Compose([x, x, x, x, Compose([Compose([x, x]), x, x])]).flatten() will result in the equivalent of t1 = Compose([x, x, x, x, x, x, x, x]). Inverse of __call__. NotImplementedError – When the subclass does not override this method. Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. Compose a Randomizable instance.

InvertibleTransform#

Classes for invertible transforms. This class exists so that an invert method can be implemented. This allows, for example, images to be cropped, rotated, padded, etc., during training and inference, and after be returned to their original size before saving to file for comparison in an external viewer. When the inverse method is called: the inverse is called on each key individually, which allows for different parameters being passed to each label (e.g., different interpolation for image and label). the inverse transforms are applied in a last-in-first-out order. As the inverse is applied, its entry is removed from the list detailing the applied transformations. That is to say that during the forward pass, the list of applied transforms grows, and then during the inverse it shrinks back down to an empty list. We currently check that the id() of the transform is the same in the forward and inverse directions. This is a useful check to ensure that the inverses are being processed in the correct order. Note to developers: When converting a transform to an invertible transform, you need to: Inherit from this class. In __call__, add a call to push_transform. Any extra information that might be needed for the inverse can be included with the dictionary extra_info. This dictionary should have the same keys regardless of whether do_transform was True or False and can only contain objects that are accepted in pytorch data loader's collate function (e.g., None is not allowed). Implement an inverse method. Make sure that after performing the inverse, pop_transform is called. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Any This function is to be called before every self.inverse(data), update each MetaTensor data[key] using data[key_transforms] and data[key_meta_dict], for MetaTensor backward compatibility 0.9.0.

TraceableTransform#

Maintains a stack of applied transforms to data. A MetaTensor (this is the preferred data type). this case, a key must be supplied (this dictionary-based approach is deprecated). If data is of type MetaTensor, then the applied transform will be added to data.applied_operations. If data[key] is a MetaTensor, the applied transform will be added to data[key].applied_operations. trace_key. If, for example, the key is image, then the transform will be appended to image_transforms (this dictionary-based approach is deprecated). data is MetaTensor data is dictionary, data[key] is MetaTensor data is dictionary, data[key] is not MetaTensor (this is a deprecated approach). The __call__ method of this transform class must be implemented so that the transformation information is stored during the data transformation. The information in the stack of applied transforms must be compatible with the default collate, by only storing strings, numbers and arrays. tracing could be enabled by self.set_tracing or setting MONAI_TRACE_TRANSFORM when initializing the class. Check transforms are of same instance. None Get most recent transform for the stack. data – dictionary of data or MetaTensor. key (Optional[Hashable]) – if data is a dictionary, data[key] will be modified. check (bool) – if true, check that self is the same type as the most recently-applied transform. pop (bool) – if true, remove the transform as it is returned. Dictionary of most recently applied transform - RuntimeError – data is neither MetaTensor nor dictionary Return a dictionary with the relevant information pertaining to an applied transform. data – input data. Can be dictionary or MetaTensor. We can use shape to determine the original size of the object (unless that has been given explicitly, see orig_size). key (Optional[Hashable]) – if data is a dictionary, data[key] will be modified. extra_info (Optional[dict]) – if desired, any extra information pertaining to the applied transform can be stored in this dictionary. These are often needed for computing the inverse transformation. orig_size (Optional[Tuple]) – sometimes during the inverse it is useful to know what the size of the original image was, in which case it can be supplied here. dict Dictionary of data pertaining to the applied transformation. Return and pop the most recent transform. data – dictionary of data or MetaTensor key (Optional[Hashable]) – if data is a dictionary, data[key] will be modified check (bool) – if true, check that self is the same type as the most recently-applied transform. Dictionary of most recently applied transform - RuntimeError – data is neither MetaTensor nor dictionary Push to a stack of applied transforms. data – dictionary of data or MetaTensor. key (Optional[Hashable]) – if data is a dictionary, data[key] will be modified. extra_info (Optional[dict]) – if desired, any extra information pertaining to the applied transform can be stored in this dictionary. These are often needed for computing the inverse transformation. orig_size (Optional[Tuple]) – sometimes during the inverse it is useful to know what the size of the original image was, in which case it can be supplied here. None None, but data has been updated to store the applied transformation. Set whether to trace transforms. None The key to store the stack of applied transforms. Temporarily set the tracing status of a transform with a context manager.

BatchInverseTransform#

Perform inverse on a batch of data. This is useful if you have inferred a batch of images and want to invert them all. transform (InvertibleTransform) – a callable data transform on input data. loader (DataLoader) – data loader used to run transforms and generate the batch of data. collate_fn (Optional[Callable]) – how to collate data after inverse transformations. default won't do any collation, so the output will be a list of size batch size. num_workers (Optional[int]) – number of workers when run data loader for inverse transforms, default to 0 as only run 1 iteration and multi-processing may be even slower. if the transforms are really slow, set num_workers for multi-processing. if set to None, use the num_workers of the transform data loader. detach (bool) – whether to detach the tensors. Scalars tensors will be detached into number types instead of torch tensors. pad_batch (bool) – when the items in a batch indicate different batch size, whether to pad all the sequences to the longest. If False, the batch size will be the length of the shortest sequence. fill_value – the value to fill the padded sequences when pad_batch=True.

Decollated#

Decollate a batch of data. If input is a dictionary, it also supports to only decollate specified keys. Note that unlike most MapTransforms, it will delete the other keys that are not specified. if keys=None, it will decollate all the data in the input. It replicates the scalar values to every item of the decollated list. keys (Union[Collection[Hashable], Hashable, None]) – keys of the corresponding items to decollate, note that it will delete other keys not specified. if None, will decollate all the keys. see also: monai.transforms.compose.MapTransform. detach (bool) – whether to detach the tensors. Scalars tensors will be detached into number types instead of torch tensors. pad_batch (bool) – when the items in a batch indicate different batch size, whether to pad all the sequences to the longest. If False, the batch size will be the length of the shortest sequence. fill_value – the value to fill the padded sequences when pad_batch=True. allow_missing_keys (bool) – don't raise exception if key is missing.

OneOf#

OneOf provides the ability to randomly choose one transform out of a list of callables with pre-defined probabilities for each. transforms (Union[Sequence[Callable], Callable, None]) – sequence of callables. weights (Union[Sequence[float], float, None]) – probabilities corresponding to each callable in transforms. Probabilities are normalized to sum to one. map_items (bool) – whether to apply transform to each item in the input data if data is a list or tuple. defaults to True. unpack_items (bool) – whether to unpack input data with * as parameters for the callable function of transform. defaults to False. log_stats (bool) – whether to log the detailed information of data and applied transform when error happened, for NumPy array and PyTorch Tensor, log the data shape and value range, for other metadata, log the values directly. default to False. Return a Composition with a simple list of transforms, as opposed to any nested Compositions. e.g., t1 = Compose([x, x, x, x, Compose([Compose([x, x]), x, x])]).flatten() will result in the equivalent of t1 = Compose([x, x, x, x, x, x, x, x]). Inverse of __call__. NotImplementedError – When the subclass does not override this method.

RandomOrder#

RandomOrder provides the ability to apply a list of transformations in random order. transforms (Union[Sequence[Callable], Callable, None]) – sequence of callables. map_items (bool) – whether to apply transform to each item in the input data if data is a list or tuple. defaults to True. unpack_items (bool) – whether to unpack input data with * as parameters for the callable function of transform. defaults to False. log_stats (bool) – whether to log the detailed information of data and applied transform when error happened, for NumPy array and PyTorch Tensor, log the data shape and value range, for other metadata, log the values directly. default to False. Inverse of __call__. NotImplementedError – When the subclass does not override this method.

Crop and Pad#

Same as MONAI's list_data_collate, except any tensors are centrally padded to match the shape of the biggest tensor in each dimension. This transform is useful if some of the applied transforms generate batch data of different sizes. This can be used on both list and dictionary data. Note that in the case of the dictionary data, it may add the transform information to the list of invertible transforms if input batch have different spatial shape, so need to call static method: inverse before inverting other transforms. Note that normally, a user won't explicitly use the __call__ method. Rather this would be passed to the DataLoader. This means that __call__ handles data as it comes out of a DataLoader, containing batch dimension. However, the inverse operates on dictionaries containing images of shape C,H,W,[D]. This asymmetry is necessary so that we can pass the inverse through multiprocessing. method (str) – padding method (see monai.transforms.SpatialPad) mode (str) – padding mode (see monai.transforms.SpatialPad) kwargs – other arguments for the np.pad or torch.pad function. note that np.pad treats channel dimension as the first dimension. batch (Any) – batch of data to pad-collate Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, ndarray] Perform padding for a given an amount of padding in each dimension. torch.nn.functional.pad is used unless the mode or kwargs are not available in torch, in which case np.pad will be used. to_pad (Optional[List[Tuple[int, int]]]) – the amount to pad in each dimension (including the channel) [(low_H, high_H), (low_W, high_W), …]. if None, must provide in the __call__ at runtime. mode (str) – available modes: (Numpy) {"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} (PyTorch) {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "constant". See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html requires pytorch >= 1.10 for best compatibility. kwargs – other arguments for the np.pad or torch.pad function. note that np.pad treats channel dimension as the first dimension. img (Tensor) – data to be transformed, assuming img is channel-first and padding doesn't apply to the channel dim. to_pad (Optional[List[Tuple[int, int]]]) – the amount to be padded in each dimension [(low_H, high_H), (low_W, high_W), …]. default to self.to_pad. mode (Optional[str]) – available modes: (Numpy) {"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} (PyTorch) {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "constant". See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html kwargs – other arguments for the np.pad or torch.pad function. note that np.pad treats channel dimension as the first dimension. Tensor dynamically compute the pad width according to the spatial shape. the output is the amount of padding for all dimensions including the channel. spatial_shape (Sequence[int]) – spatial shape of the original image. List[Tuple[int, int]] Inverse of __call__. NotImplementedError – When the subclass does not override this method. MetaTensor Performs padding to the data, symmetric for all sides or all on one side for each dimension. spatial_size (Union[Sequence[int], int, Tuple[Union[Tuple[int, …], int], …]]) – the spatial size of output data after padding, if a dimension of the input data size is larger than the pad size, will not pad that dimension. If its components have non-positive values, the corresponding size of input image will be used (no padding). for example: if the spatial size of input data is [30, 30, 30] and spatial_size=[32, 25, -1], the spatial size of output data will be [32, 30, 30]. method (str) – {"symmetric", "end"} Pad image symmetrically on every side or only pad at the end sides. Defaults to "symmetric". mode (str) – available modes for numpy array:{"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "constant". See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html kwargs – other arguments for the np.pad or torch.pad function. note that np.pad treats channel dimension as the first dimension. dynamically compute the pad width according to the spatial shape. spatial_shape (Sequence[int]) – spatial shape of the original image. List[Tuple[int, int]] Pad the input data by adding specified borders to every dimension. spatial_border (Union[Sequence[int], int]) – specified size for every spatial border. Any -ve values will be set to 0. It can be 3 shapes: single int number, pad all the borders with the same size. length equals the length of image shape, pad every spatial dimension separately. for example, image shape(CHW) is [1, 4, 4], spatial_border is [2, 1], pad every border of H dim with 2, pad every border of W dim with 1, result shape is [1, 8, 6]. length equals 2 x (length of image shape), pad every border of every dimension separately. for example, image shape(CHW) is [1, 4, 4], spatial_border is [1, 2, 3, 4], pad top of H dim with 1, pad bottom of H dim with 2, pad left of W dim with 3, pad right of W dim with 4. the result shape is [1, 7, 11]. specified size for every spatial border. Any -ve values will be set to 0. It can be 3 shapes: single int number, pad all the borders with the same size. length equals the length of image shape, pad every spatial dimension separately. for example, image shape(CHW) is [1, 4, 4], spatial_border is [2, 1], pad every border of H dim with 2, pad every border of W dim with 1, result shape is [1, 8, 6]. length equals 2 x (length of image shape), pad every border of every dimension separately. for example, image shape(CHW) is [1, 4, 4], spatial_border is [1, 2, 3, 4], pad top of H dim with 1, pad bottom of H dim with 2, pad left of W dim with 3, pad right of W dim with 4. the result shape is [1, 7, 11]. mode (str) – available modes for numpy array:{"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "constant". See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html kwargs – other arguments for the np.pad or torch.pad function. note that np.pad treats channel dimension as the first dimension. dynamically compute the pad width according to the spatial shape. the output is the amount of padding for all dimensions including the channel. spatial_shape (Sequence[int]) – spatial shape of the original image. List[Tuple[int, int]] Pad the input data, so that the spatial sizes are divisible by k. k (Union[Sequence[int], int]) – the target k for each spatial dimension. if k is negative or 0, the original size is preserved. if k is an int, the same k be applied to all the input spatial dimensions. mode (str) – available modes for numpy array:{"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "constant". See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html method (str) – {"symmetric", "end"} Pad image symmetrically on every side or only pad at the end sides. Defaults to "symmetric". kwargs – other arguments for the np.pad or torch.pad function. note that np.pad treats channel dimension as the first dimension. See also monai.transforms.SpatialPad dynamically compute the pad width according to the spatial shape. the output is the amount of padding for all dimensions including the channel. spatial_shape (Sequence[int]) – spatial shape of the original image. List[Tuple[int, int]] Perform crop operations on the input image. Apply the transform to img, assuming img is channel-first and slicing doesn’t apply to the channel dim. Tensor Compute the crop slices based on specified center & size or start & end or slices. roi_center (Union[Sequence[int], ndarray, Tensor, None]) – voxel coordinates for center of the crop ROI. roi_size (Union[Sequence[int], ndarray, Tensor, None]) – size of the crop ROI, if a dimension of ROI size is larger than image size, will not crop that dimension of the image. roi_start (Union[Sequence[int], ndarray, Tensor, None]) – voxel coordinates for start of the crop ROI. roi_end (Union[Sequence[int], ndarray, Tensor, None]) – voxel coordinates for end of the crop ROI, if a coordinate is out of image, use the end coordinate of image. roi_slices (Optional[Sequence[slice]]) – list of slices for each of the spatial dimensions. Inverse of __call__. NotImplementedError – When the subclass does not override this method. MetaTensor General purpose cropper to produce sub-volume region of interest (ROI). If a dimension of the expected ROI size is larger than the input image size, will not crop that dimension. So the cropped result may be smaller than the expected ROI, and the cropped results of several images may not have exactly the same shape. It can support to crop ND spatial (channel-first) data. a list of slices for each spatial dimension (allows for use of negative indexing and None) a spatial center and size the start and end coordinates of the ROI Apply the transform to img, assuming img is channel-first and slicing doesn't apply to the channel dim. Tensor roi_center (Union[Sequence[int], ndarray, Tensor, None]) – voxel coordinates for center of the crop ROI. roi_size (Union[Sequence[int], ndarray, Tensor, None]) – size of the crop ROI, if a dimension of ROI size is larger than image size, will not crop that dimension of the image. roi_start (Union[Sequence[int], ndarray, Tensor, None]) – voxel coordinates for start of the crop ROI. roi_end (Union[Sequence[int], ndarray, Tensor, None]) – voxel coordinates for end of the crop ROI, if a coordinate is out of image, use the end coordinate of image. roi_slices (Optional[Sequence[slice]]) – list of slices for each of the spatial dimensions. Crop at the center of image with specified ROI size. If a dimension of the expected ROI size is larger than the input image size, will not crop that dimension. So the cropped result may be smaller than the expected ROI, and the cropped results of several images may not have exactly the same shape. roi_size (Union[Sequence[int], int]) – the spatial size of the crop region e.g. [224,224,128] if a dimension of ROI size is larger than image size, will not crop that dimension of the image. If its components have non-positive values, the corresponding size of input image will be used. for example: if the spatial size of input data is [40, 40, 40] and roi_size=[32, 64, -1], the spatial size of output data will be [32, 40, 40]. Apply the transform to img, assuming img is channel-first and slicing doesn't apply to the channel dim. Tensor Compute the crop slices based on specified center & size or start & end or slices. roi_center – voxel coordinates for center of the crop ROI. roi_size – size of the crop ROI, if a dimension of ROI size is larger than image size, will not crop that dimension of the image. roi_start – voxel coordinates for start of the crop ROI. roi_end – voxel coordinates for end of the crop ROI, if a coordinate is out of image, use the end coordinate of image. roi_slices – list of slices for each of the spatial dimensions. Crop image with random size or specific size ROI. It can crop at a random position as center or at the image center. And allows to set the minimum and maximum size to limit the randomly generated ROI. Note: even random_size=False, if a dimension of the expected ROI size is larger than the input image size, will not crop that dimension. So the cropped result may be smaller than the expected ROI, and the cropped results of several images may not have exactly the same shape. roi_size (Union[Sequence[int], int]) – if random_size is True, it specifies the minimum crop region. if random_size is False, it specifies the expected ROI size to crop. e.g. [224, 224, 128] if a dimension of ROI size is larger than image size, will not crop that dimension of the image. If its components have non-positive values, the corresponding size of input image will be used. for example: if the spatial size of input data is [40, 40, 40] and roi_size=[32, 64, -1], the spatial size of output data will be [32, 40, 40]. max_roi_size (Union[Sequence[int], int, None]) – if random_size is True and roi_size specifies the min crop region size, max_roi_size can specify the max crop region size. if None, defaults to the input image size. if its components have non-positive values, the corresponding size of input image will be used. random_center (bool) – crop at random position as center or the image center. random_size (bool) – crop with random size or specific size ROI. if True, the actual size is sampled from randint(roi_size, max_roi_size + 1). Apply the transform to img, assuming img is channel-first and slicing doesn't apply to the channel dim. Tensor Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None Crop image with random size or specific size ROI to generate a list of N samples. It can crop at a random position as center or at the image center. And allows to set the minimum size to limit the randomly generated ROI. It will return a list of cropped images. Note: even random_size=False, if a dimension of the expected ROI size is larger than the input image size, will not crop that dimension. So the cropped result may be smaller than the expected ROI, and the cropped results of several images may not have exactly the same shape. roi_size (Union[Sequence[int], int]) – if random_size is True, it specifies the minimum crop region. if random_size is False, it specifies the expected ROI size to crop. e.g. [224, 224, 128] if a dimension of ROI size is larger than image size, will not crop that dimension of the image. If its components have non-positive values, the corresponding size of input image will be used. for example: if the spatial size of input data is [40, 40, 40] and roi_size=[32, 64, -1], the spatial size of output data will be [32, 40, 40]. num_samples (int) – number of samples (crop regions) to take in the returned list. max_roi_size (Union[Sequence[int], int, None]) – if random_size is True and roi_size specifies the min crop region size, max_roi_size can specify the max crop region size. if None, defaults to the input image size. if its components have non-positive values, the corresponding size of input image will be used. random_center (bool) – crop at random position as center or the image center. random_size (bool) – crop with random size or specific size ROI. The actual size is sampled from randint(roi_size, img_size). ValueError – When num_samples is nonpositive. Apply the transform to img, assuming img is channel-first and cropping doesn't change the channel dim. List[Tensor] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandSpatialCropSamples a Randomizable instance. Crop an image using a bounding box. The bounding box is generated by selecting foreground using select_fn at channels channel_indices. margin is added in each spatial dimension of the bounding box. The typical usage is to help training and evaluation if the valid part is small in the whole medical image. Users can define arbitrary function to select expected foreground from the whole image or specified channels. And it can also add margin to every dim of the bounding box of foreground object. For example: Apply the transform to img, assuming img is channel-first and slicing doesn't change the channel dim. select_fn (Callable) – function to select expected foreground, default is to select values > 0. channel_indices (Union[Iterable[int], int, None]) – if defined, select foreground only on the specified channels of image. if None, select foreground on the whole image. margin (Union[Sequence[int], int]) – add margin value to spatial dims of the bounding box, if only 1 value provided, use it for all dims. allow_smaller (bool) – when computing box size with margin, whether allow the image size to be smaller than box size, default to

True. if the margined size is larger than image size, will pad with specified mode. return_coords (bool) – whether return the coordinates of spatial bounding box for foreground. k_divisible (Union[Sequence[int], int]) – make each spatial dimension to be divisible by k, default to 1. if k_divisible is an int, the same k be applied to all the input spatial dimensions. mode (str) – available modes for numpy array:{"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "constant". See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html pad_kwargs – other arguments for the np.pad or torch.pad function. note that np.pad treats channel dimension as the first dimension. Compute the start points and end points of bounding box to crop. And adjust bounding box coords to be divisible by k. Crop and pad based on the bounding box. Inverse of __call__. NotImplementedError – When the subclass does not override this method. MetaTensor Samples a list of num_samples image patches according to the provided weight_map. spatial_size (Union[Sequence[int], int]) – the spatial size of the image patch e.g. [224, 224, 128]. If its components have non-positive values, the corresponding size of img will be used. num_samples (int) – number of samples (image patches) to take in the returned list. weight_map (Union[ndarray, Tensor, None]) – weight map used to generate patch samples. The weights must be non-negative. Each element denotes a sampling weight of the spatial location. 0 indicates no sampling. It should be a single-channel array in shape, for example, (1, spatial_dim_0, spatial_dim_1, …). img (Tensor) – input image to sample patches from. assuming img is a channel-first array. weight_map (Union[ndarray, Tensor, None]) – weight map used to generate patch samples. The weights must be non-negative. Each element denotes a sampling weight of the spatial location. 0 indicates no sampling. It should be a single-channel array in shape, for example, (1, spatial_dim_0, spatial_dim_1, …) randomize (bool) – whether to execute random operations, default to True. List[Tensor] A list of image patches Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None Crop random fixed sized regions with the center being a foreground or background voxel based on the Pos Neg Ratio. And will return a list of arrays for all the cropped images. For example, crop two (3 x 3) arrays from (5 x 5) array with pos/neg=1: If a dimension of the expected spatial size is larger than the input image size, will not crop that dimension. So the cropped result may be smaller than expected size, and the cropped results of several images may not have exactly same shape. And if the crop ROI is partly out of the image, will automatically adjust the crop center to ensure the valid crop ROI. spatial_size (Union[Sequence[int], int]) – the spatial size of the crop region e.g. [224, 224, 128]. if a dimension of ROI size is larger than image size, will not crop that dimension of the image. if its components have non-positive values, the corresponding size of label will be used. for example: if the spatial size of input data is [40, 40, 40] and spatial_size=[32, 64, -1], the spatial size of output data will be [32, 40, 40]. label (Optional[Tensor]) – the label image that is used for finding foreground/background, if None, must set at self.__call__. Non-zero indicates foreground, zero indicates background. pos (float) – used with neg together to calculate the ratio pos / (pos + neg) for the probability to pick a foreground voxel as a center rather than a background voxel. neg (float) – used with pos together to calculate the ratio pos / (pos + neg) for the probability to pick a foreground voxel as a center rather than a background voxel. num_samples (int) – number of samples (crop regions) to take in each list. image (Optional[Tensor]) – optional image data to help select valid area, can be same as img or another image array. if not None, use label == 0 & image > image_threshold to select the negative sample (background) center. So the crop center will only come from the valid image areas. image_threshold (float) – if enabled image, use image > image_threshold to determine the valid image content areas. fg_indices (Union[ndarray, Tensor, None]) – if provided pre-computed foreground indices of label, will ignore above image and image_threshold, and randomly select crop centers based on them, need to provide fg_indices and bg_indices together, expect to be 1 dim array of spatial indices after flattening. a typical usage is to call FgBgToIndices transform first and cache the results. bg_indices (Union[ndarray, Tensor, None]) – if provided pre-computed background indices of label, will ignore above image and image_threshold, and randomly select crop centers based on them, need to provide fg_indices and bg_indices together, expect to be 1 dim array of spatial indices after flattening. a typical usage is to call FgBgToIndices transform first and cache the results. allow_smaller (bool) – if False, an exception will be raised if the image is smaller than the requested ROI in any dimension. If True, any smaller dimensions will be set to match the cropped size (i.e., no cropping in that dimension). ValueError – When pos or neg are negative. ValueError – When pos=0 and neg=0. Incompatible values. img (Tensor) – input data to crop samples from based on the pos/neg ratio of label and image. Assumes img is a channel-first array. label (Optional[Tensor]) – the label image that is used for finding foreground/background, if None, use self.label. image (Optional[Tensor]) – optional image data to help select valid area, can be same as img or another image array. use label == 0 & image > image_threshold to select the negative sample(background) center. so the crop center will only exist on valid image area. if None, use self.image. fg_indices (Union[ndarray, Tensor, None]) – foreground indices to randomly select crop centers, need to provide fg_indices and bg_indices together. bg_indices (Union[ndarray, Tensor, None]) – background indices to randomly select crop centers, need to provide fg_indices and bg_indices together. randomize (bool) – whether to execute the random operations, default to True. List[Tensor] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None Crop random fixed sized regions with the center being a class based on the specified ratios of every class. The label data can be One-Hot format array or Argmax data. And will return a list of arrays for all the cropped images. For example, crop two (3 x 3) arrays from (5 x 5) array with ratios=[1, 2, 3, 1]: If a dimension of the expected spatial size is larger than the input image size, will not crop that dimension. So the cropped result may be smaller than expected size, and the cropped results of several images may not have exactly same shape. And if the crop ROI is partly out of the image, will automatically adjust the crop center to ensure the valid crop ROI. spatial_size (Union[Sequence[int], int]) – the spatial size of the crop region e.g. [224, 224, 128]. if a dimension of ROI size is larger than image size, will not crop that dimension of the image. if its components have non-positive values, the corresponding size of label will be used. for example: if the spatial size of input data is [40, 40, 40] and spatial_size=[32, 64, -1], the spatial size of output data will be [32, 40, 40]. ratios (Optional[List[Union[float, int]]]) – specified ratios of every class in the label to generate crop centers, including background class. if None, every class will have the same ratio to generate crop centers. label (Optional[Tensor]) – the label image that is used for finding every classes, if None, must set at self.__call__. num_classes (Optional[int]) – number of classes for argmax label, not necessary for One-Hot label. num_samples (int) – number of samples (crop regions) to take in each list. image

(Optional[Tensor]) – if image is not None, only return the indices of every class that are within the valid region of the image (image > image_threshold). image_threshold (float) – if enabled image, use image > image_threshold to determine the valid image content area and select class indices only in this area. indices (Optional[List[Union[ndarray, Tensor]]]) – if provided pre-computed indices of every class, will ignore above image and image_threshold, and randomly select crop centers based on them, expect to be 1 dim array of spatial indices after flattening. a typical usage is to call ClassesToIndices transform first and cache the results for better performance. allow_smaller (bool) – if False, an exception will be raised if the image is smaller than the requested ROI in any dimension. If True, any smaller dimensions will remain unchanged. img (Tensor) – input data to crop samples from based on the ratios of every class, assumes img is a channel-first array. label (Optional[Tensor]) – the label image that is used for finding indices of every class, if None, use self.label. image (Optional[Tensor]) – optional image data to help select valid area, can be same as img or another image array. use image > image_threshold to select the centers only in valid region. if None, use self.image. indices (Optional[List[Union[ndarray, Tensor]]]) – list of indices for every class in the image, used to randomly select crop centers. randomize (bool) – whether to execute the random operations, default to True. List[Tensor] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None Resize an image to a target spatial size by either centrally cropping the image or padding it evenly with a user-specified mode. When the dimension is smaller than the target size, do symmetric padding along that dim. When the dimension is larger than the target size, do central cropping along that dim. spatial_size (Union[Sequence[int], int]) – the spatial size of output data after padding or crop. If has non-positive values, the corresponding size of input image will be used (no padding). method (str) – {"symmetric", "end"} Pad image symmetrically on every side or only pad at the end sides. Defaults to "symmetric". mode (str) – available modes for numpy array:{"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "constant". See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html pad_kwargs – other arguments for the np.pad or torch.pad function. note that np.pad treats channel dimension as the first dimension. img (Tensor) – data to pad or crop, assuming img is channel-first and padding or cropping doesn't apply to the channel dim. mode (Optional[str]) – available modes for numpy array:{"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "constant". See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html kwargs – other arguments for the np.pad or torch.pad function. note that np.pad treats channel dimension as the first dimension. Tensor Inverse of __call__. NotImplementedError – When the subclass does not override this method. MetaTensor Compute coordinates of axis-aligned bounding rectangles from input image img. The output format of the coordinates is (shape is [channel, 2 * spatial dims]): 2nd_spatial_dim_start, 2nd_spatial_dim_end, …, Nth_spatial_dim_start, Nth_spatial_dim_end], … [1st_spatial_dim_start, 1st_spatial_dim_end, 2nd_spatial_dim_start, 2nd_spatial_dim_end, …, Nth_spatial_dim_start, Nth_spatial_dim_end]] The bounding boxes edges are aligned with the input image edges. This function returns [0, 0, …] if there’s no positive intensity. select_fn (Callable) – function to select expected foreground, default is to select values > 0. See also: monai.transforms.utils.generate_spatial_bounding_box. ndarray Subclass of monai.transforms.RandSpatialCrop. Crop image with random size or specific size ROI. It can crop at a random position as center or at the image center. And allows to set the minimum and maximum scale of image size to limit the randomly generated ROI. roi_scale (Union[Sequence[float], float]) – if random_size is True, it specifies the minimum crop size: roi_scale * image spatial size. if random_size is False, it specifies the expected scale of image size to crop. e.g. [0.3, 0.4, 0.5]. If its components have non-positive values, will use 1.0 instead, which means the input image size. max_roi_scale (Union[Sequence[float], float, None]) – if random_size is True and roi_scale specifies the min crop region size, max_roi_scale can specify the max crop region size: max_roi_scale * image spatial size. if None, defaults to the input image size. if its components have non-positive values, will use 1.0 instead, which means the input image size. random_center (bool) – crop at random position as center or the image center. random_size (bool) – crop with random size or specified size ROI by roi_scale * image spatial size. if True, the actual size is sampled from randint(roi_scale * image spatial size, max_roi_scale * image spatial size + 1). Apply the transform to img, assuming img is channel-first and slicing doesn’t apply to the channel dim. Tensor Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None Crop at the center of image with specified scale of ROI size. roi_scale (Union[Sequence[float], float]) – specifies the expected scale of image size to crop. e.g. [0.3, 0.4, 0.5] or a number for all dims. If its components have non-positive values, will use 1.0 instead, which means the input image size. Apply the transform to img, assuming img is channel-first and slicing doesn’t apply to the channel dim. Tensor

Intensity#

Add Gaussian noise to image. prob (float) – Probability to add Gaussian noise. mean (float) – Mean or “centre” of the distribution. std (float) – Standard deviation (spread) of distribution. dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. Apply the transform to img. Union[ndarray, Tensor] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Shift intensity uniformly for the entire image with specified offset. offset (float) – offset value to shift the intensity of image. safe (bool) – if True, then do safe dtype convert when intensity overflow. default to False. E.g., [256, -12] -> [array(0), array(244)]. If True, then [256, -12] -> [array(255), array(0)]. Apply the transform to img. Union[ndarray, Tensor] Randomly shift intensity with randomly picked offset. Apply the transform to img. img (Union[ndarray, Tensor]) – input image to shift intensity. factor (Optional[float]) – a factor to multiply the random offset, then shift. can be some image specific value at runtime, like: max(img), etc. Union[ndarray, Tensor] offsets (Union[Tuple[float, float], float]) – offset range to randomly shift. if single number, offset value is picked from (-offsets, offsets). safe (bool) – if True, then do safe dtype convert when intensity overflow. default to False.

E.g., [256, -12] -> [array(0), array(244)]. If True, then [256, -12] -> [array(255), array(0)]. prob (float) – probability of shift. Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Shift intensity for the image with a factor and the standard deviation of the image by: v = v + factor * std(v). This transform can focus on only non-zero values or the entire image, and can also calculate the std on each channel separately. factor (float) – factor shift by v = v + factor * std(v). nonzero (bool) – whether only count non-zero values. channel_wise (bool) – if True, calculate on each channel separately. Please ensure that the first dimension represents the channel of the image if True. dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. Apply the transform to img. Union[ndarray, Tensor] Shift intensity for the image with a factor and the standard deviation of the image by: v = v + factor * std(v) where the factor is randomly picked. Apply the transform to img. Union[ndarray, Tensor] factors (Union[Tuple[float, float], float]) – if tuple, the randomly picked range is (min(factors), max(factors)). If single number, the range is (-factors, factors). prob (float) – probability of std shift. nonzero (bool) – whether only count non-zero values. channel_wise (bool) – if True, calculate on each channel separately. dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Random bias field augmentation for MR images. The bias field is considered as a linear combination of smoothly varying basis (polynomial) functions, as described in Automated Model-Based Tissue Classification of MR Images of the Brain. This implementation adapted from NiftyNet. Referred to Longitudinal segmentation of age-related white matter hyperintensities. degree (int) – degree of freedom of the polynomials. The value should be no less than 1. Defaults to 3. coeff_range (Tuple[float, float]) – range of the random coefficients. Defaults to (0.0, 0.1). dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. prob (float) – probability to do random bias field. Apply the transform to img. Union[ndarray, Tensor] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Scale the intensity of input image to the given value range (minv, maxv). If minv and maxv not provided, use factor to scale image by v = v * (1 + factor). Apply the transform to img. ValueError – When self.minv=None or self.maxv=None and self.factor=None. Incompatible values. Union[ndarray, Tensor] minv (Optional[float]) – minimum value of output data. maxv (Optional[float]) – maximum value of output data. factor (Optional[float]) – factor scale by v = v * (1 + factor). In order to use this parameter, please set both minv and maxv into None. channel_wise (bool) – if True, scale on each channel separately. Please ensure that the first dimension represents the channel of the image if True. dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. Randomly scale the intensity of input image by v = v * (1 + factor) where the factor is randomly picked. Apply the transform to img. Union[ndarray, Tensor] factors (Union[Tuple[float, float], float]) – factor range to randomly scale by v = v * (1 + factor). if single number, factor value is picked from (-factors, factors). prob (float) – probability of scale. dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Normalize input based on the subtrahend and divisor: (img - subtrahend) / divisor. Use calculated mean or std value of the input image if no subtrahend or divisor provided. This transform can normalize only non-zero values or entire image, and can also calculate mean and std on each channel separately. When channel_wise is True, the first dimension of subtrahend and divisor should be the number of image channels if they are not None. subtrahend (Union[Sequence, ndarray, Tensor, None]) – the amount to subtract by (usually the mean). divisor (Union[Sequence, ndarray, Tensor, None]) – the amount to divide by (usually the standard deviation). nonzero (bool) – whether only normalize non-zero values. channel_wise (bool) – if True, calculate on each channel separately, otherwise, calculate on the entire image directly. default to False. dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. Apply the transform to img, assuming img is a channel-first array if self.channel_wise is True, Union[ndarray, Tensor] Filter the intensity values of whole image to below threshold or above threshold. And fill the remaining parts of the image to the cval value. threshold (float) – the threshold to filter intensity values. above (bool) – filter values above the threshold or below the threshold, default is True. cval (float) – value to fill the remaining parts of the image, default is 0. Apply the transform to img. Union[ndarray, Tensor] Apply specific intensity scaling to the whole numpy array. Scaling from [a_min, a_max] to [b_min, b_max] with clip option. When b_min or b_max are None, scaled_array * (b_max - b_min) + b_min will be skipped. If clip=True, when b_min/b_max is None, the clipping is not performed on the corresponding edge. a_min (float) – intensity original range min. a_max (float) – intensity original range max. b_min (Optional[float]) – intensity target range min. b_max (Optional[float]) – intensity target range max. clip (bool) – whether to perform clip after scaling. dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. Apply the transform to img. Union[ndarray, Tensor] Apply range scaling to a numpy array based on the intensity distribution of the input. By default this transform will scale from [lower_intensity_percentile, upper_intensity_percentile] to [b_min, b_max], where {lower,upper}_intensity_percentile are the intensity values at the corresponding percentiles of img. The relative parameter can also be set to scale from [lower_intensity_percentile, upper_intensity_percentile] to the lower and upper percentiles of the output range [b_min, b_max]. For example: See also monai.transforms.ScaleIntensityRange lower (float) – lower intensity percentile. upper (float) – upper intensity percentile. b_min (Optional[float]) – intensity target range min. b_max (Optional[float]) – intensity target range max. clip (bool) – whether to perform clip after scaling. relative (bool) – whether to scale to the corresponding percentiles of [b_min, b_max]. channel_wise (bool) – if True, compute intensity percentile and normalize every channel separately. default to False. dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. Apply the transform to img. Union[ndarray, Tensor] Changes image intensity by gamma. Each pixel/voxel intensity is updated as: gamma (float) – gamma value to adjust the contrast as function. Apply the transform to img. Union[ndarray, Tensor] Randomly changes image intensity by gamma. Each pixel/voxel intensity is updated as: prob (float) – Probability of adjustment. gamma (Union[Sequence[float], float]) – Range of gamma values. If single number, value is picked from (0.5, gamma), default is (0.5, 4.5). Apply the transform to img. Union[ndarray, Tensor] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Mask the intensity values of input image with the specified mask data.

Mask data must have the same spatial size as the input image, and all the intensity values of input image corresponding to the selected values in the mask data will keep the original value, others will be set to 0. mask_data (Union[ndarray, Tensor, None]) – if mask_data is single channel, apply to every channel of input image. if multiple channels, the number of channels must match the input data. the intensity values of input image corresponding to the selected values in the mask data will keep the original value, others will be set to 0. if None, must specify the mask_data at runtime. select_fn (Callable) – function to select valid values of the mask_data, default is to select values > 0. mask_data (Union[ndarray, Tensor, None]) – if mask data is single channel, apply to every channel of input image. if multiple channels, the channel number must match input data. mask_data will be converted to bool values by mask_data > 0 before applying transform to input image. - ValueError – When both mask_data and self.mask_data are None. - ValueError – When mask_data and img channels differ and mask_data is not single channel. Union[ndarray, Tensor] Smooth the input data along the given axis using a Savitzky-Golay filter. window_length (int) – Length of the filter window, must be a positive odd integer. order (int) – Order of the polynomial to fit to each window, must be less than window_length. axis (int) – Optional axis along which to apply the filter kernel. Default 1 (first spatial dimension). mode (str) – Optional padding mode, passed to convolution class. 'zeros', 'reflect', 'replicate' or 'circular'. Default: 'zeros'. See torch.nn.Conv1d() for more information. img (Union[ndarray, Tensor]) – array containing input data. Must be real and in shape [channels, spatial1, spatial2, …]. Union[ndarray, Tensor] array containing smoothed result. Apply median filter to the input data based on specified radius parameter. A default value radius=1 is provided for reference. See also: monai.networks.layers.median_filter() radius (Union[Sequence[int], int]) – if a list of values, must match the count of spatial dimensions of input data, and apply every value in the list to 1 spatial dimension. if only 1 value provided, use it for all spatial dimensions. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. ~NdarrayTensor Apply Gaussian smooth to the input data based on specified sigma parameter. A default value sigma=1.0 is provided for reference. sigma (Union[Sequence[float], float]) – if a list of values, must match the count of spatial dimensions of input data, and apply every value in the list to 1 spatial dimension. if only 1 value provided, use it for all spatial dimensions. approx (str) – discrete Gaussian kernel type, available options are “erf”, “sampled”, and “scalespace”. see also monai.networks.layers.GaussianFilter(). data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. ~NdarrayTensor Apply Gaussian smooth to the input data based on randomly selected sigma parameters. sigma_x (Tuple[float, float]) – randomly select sigma value for the first spatial dimension. sigma_y (Tuple[float, float]) – randomly select sigma value for the second spatial dimension if have. sigma_z (Tuple[float, float]) – randomly select sigma value for the third spatial dimension if have. prob (float) – probability of Gaussian smooth. approx (str) – discrete Gaussian kernel type, available options are “erf”, “sampled”, and “scalespace”. see also monai.networks.layers.GaussianFilter(). data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Union[ndarray, Tensor] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Sharpen images using the Gaussian Blur filter. Referring to: http://scipy-lectures.org/advanced/image_processing/auto_examples/plot_sharpen.html. The algorithm is shown as below A set of default values sigma1=3.0, sigma2=1.0 and alpha=30.0 is provide for reference. sigma1 (Union[Sequence[float], float]) – sigma parameter for the first gaussian kernel. if a list of values, must match the count of spatial dimensions of input data, and apply every value in the list to 1 spatial dimension. if only 1 value provided, use it for all spatial dimensions. sigma2 (Union[Sequence[float], float]) – sigma parameter for the second gaussian kernel. if a list of values, must match the count of spatial dimensions of input data, and apply every value in the list to 1 spatial dimension. if only 1 value provided, use it for all spatial dimensions. alpha (float) – weight parameter to compute the final result. approx (str) – discrete Gaussian kernel type, available options are “erf”, “sampled”, and “scalespace”. see also monai.networks.layers.GaussianFilter(). data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. ~NdarrayTensor Sharpen images using the Gaussian Blur filter based on randomly selected sigma1, sigma2 and alpha. The algorithm is monai.transforms.GaussianSharpen. sigma1_x (Tuple[float, float]) – randomly select sigma value for the first spatial dimension of first gaussian kernel. sigma1_y (Tuple[float, float]) – randomly select sigma value for the second spatial dimension(if have) of first gaussian kernel. sigma1_z (Tuple[float, float]) – randomly select sigma value for the third spatial dimension(if have) of first gaussian kernel. sigma2_x (Union[Tuple[float, float], float]) – randomly select sigma value for the first spatial dimension of second gaussian kernel. if only 1 value X provided, it must be smaller than sigma1_x and randomly select from [X, sigma1_x]. sigma2_y (Union[Tuple[float, float], float]) – randomly select sigma value for the second spatial dimension(if have) of second gaussian kernel. if only 1 value Y provided, it must be smaller than sigma1_y and randomly select from [Y, sigma1_y]. sigma2_z (Union[Tuple[float, float], float]) – randomly select sigma value for the third spatial dimension(if have) of second gaussian kernel. if only 1 value Z provided, it must be smaller than sigma1_z and randomly select from [Z, sigma1_z]. alpha (Tuple[float, float]) – randomly select weight parameter to compute the final result. approx (str) – discrete Gaussian kernel type, available options are "erf", "sampled", and "scalespace". see also monai.networks.layers.GaussianFilter(). prob (float) – probability of Gaussian sharpen. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Union[ndarray, Tensor] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Apply random nonlinear transform to the image's intensity histogram. num_control_points (Union[Tuple[int, int], int]) – number of control points governing the nonlinear intensity mapping. a smaller number of control points allows for larger intensity shifts. if two values provided, number of control points selecting from range (min_value, max_value). prob (float) – probability of histogram shift. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Union[ndarray, Tensor] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Find the envelope of the input data along the requested axis using a Hilbert transform. axis (int) – Axis along which to detect the envelope. Default 1, i.e. the first spatial dimension. n (Optional[int]) – FFT size. Default img.shape[axis]. Input will be zero-padded or truncated to this size along dimension axis. – img (Union[ndarray, Tensor]) – numpy.ndarray containing input data. Must be real and in shape [channels, spatial1, spatial2, …]. np.ndarray containing envelope of data in img along the specified axis. The transform applies Gibbs noise to 2D/3D MRI images. Gibbs artifacts are one of the common type of type artifacts appearing in MRI scans. The transform is applied to all the channels in the data. For general information on Gibbs artifacts, please refer to: An Image-based Approach to Understanding the Physics of MR Artifacts. The AAPM/RSNA Physics Tutorial for Residents alpha (float) – Parametrizes the intensity of the Gibbs noise filter applied. Takes values in the interval [0,1] with alpha = 0 acting as the identity mapping. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Union[ndarray, Tensor] Naturalistic image augmentation via Gibbs artifacts. The transform randomly applies Gibbs noise to 2D/3D MRI images. Gibbs artifacts are one of the common type of type artifacts appearing in MRI scans. The transform is applied to all the channels in the data. For general information on Gibbs artifacts, please refer to: https://pubs.rsna.org/doi/full/10.1148/rg.313105115 https://pubs.rsna.org/doi/full/10.1148/radiographics.22.4.g02jl14949 prob (float) – probability of applying the transform. alpha (Sequence(float)) – Parametrizes the intensity of the Gibbs noise filter applied. Takes values in the interval [0,1] with alpha = 0 acting as the identity mapping. If a length-2 list is given as [a,b] then the value of alpha will be sampled uniformly from the interval [a,b]. 0 <= a <= b <= 1. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Set random variable to apply the transform. Get alpha from uniform distribution. None Apply localized spikes in k-space at the given locations and intensities. Spike (Herringbone) artifact is a type of data acquisition artifact which may occur during MRI scans. For general information on spike artifacts, please refer to: AAPM/RSNA physics tutorial for residents: fundamental physics of MR imaging. Body MRI artifacts in clinical practice: A physicist’s and radiologist’s perspective. loc (Union[Tuple, Sequence[Tuple]]) – spatial location for the spikes. For images with 3D spatial dimensions, the user can provide (C, X, Y, Z) to fix which channel C is affected, or (X, Y, Z) to place the same spike in all channels. For 2D cases, the user can provide (C, X, Y) or (X, Y). k_intensity (Union[Sequence[float], float, None]) – value for the log-intensity of the k-space version of the image. If one location is passed to loc or the channel is not specified, then this argument should receive a float. If loc is given a sequence of locations, then this argument should receive a sequence of intensities. This value should be tested as it is data-dependent. The default values are the 2.5 the mean of the log-intensity for each channel. Example When working with 4D data, KSpaceSpikeNoise(loc = ((3,60,64,32), (64,60,32)), k_intensity = (13,14)) will place a spike at [3, 60, 64, 32] with log-intensity = 13, and one spike per channel located respectively at [: , 64, 60, 32] with log-intensity = 14. img (Union[ndarray, Tensor]) – image with dimensions (C, H, W) or (C, H, W, D) Union[ndarray, Tensor] Naturalistic data augmentation via spike artifacts. The transform applies localized spikes in k-space, and it is the random version of monai.transforms.KSpaceSpikeNoise. Spike (Herringbone) artifact is a type of data acquisition artifact which may occur during MRI scans. For general information on spike artifacts, please refer to: AAPM/RSNA physics tutorial for residents: fundamental physics of MR imaging. Body MRI artifacts in clinical practice: A physicist's and radiologist's perspective. prob (float) – probability of applying the transform, either on all channels at once, or channel-wise if channel_wise = True. intensity_range (Optional[Sequence[Union[Sequence[float], float]]]) – pass a tuple (a, b) to sample the log-intensity from the interval (a, b) uniformly for all channels. Or pass sequence of intervals ((a0, b0), (a1, b1), …) to sample for each respective channel. In the second case, the number of 2-tuples must match the number of channels. Default ranges is (0.95x, 1.10x) where x is the mean log-intensity for each channel. channel_wise (bool) – treat each channel independently. True by default. Example To apply k-space spikes randomly with probability 0.5, and log-intensity sampled from the interval [11, 12] for each channel independently, one uses RandKSpaceSpikeNoise(prob=0.5, intensity_range=(11, 12), channel_wise=True) Apply transform to img. Assumes data is in channel-first form. img (Union[ndarray, Tensor]) – image with dimensions (C, H, W) or (C, H, W, D) Helper method to sample both the location and intensity of the spikes. When not working channel wise (channel_wise=False) it use the random variable self._do_transform to decide whether to sample a location and intensity. When working channel wise, the method randomly samples a location and intensity for each channel depending on self._do_transform. None Add Rician noise to image. Rician noise in MRI is the result of performing a magnitude operation on complex data with Gaussian noise of the same variance in both channels, as described in Noise in Magnitude Magnetic Resonance Images. This transform is adapted from DIPY. See also: The rician distribution of noisy mri data. prob (float) – Probability to add Rician noise. mean (Union[Sequence[float], float]) – Mean or "centre" of the Gaussian distributions sampled to make up the Rician noise. std (Union[Sequence[float], float]) – Standard deviation (spread) of the Gaussian distributions sampled to make up the Rician noise. channel_wise (bool) – If True, treats each channel of the image separately. relative (bool) – If True, the spread of the sampled Gaussian distributions will be std times the standard deviation of the image or channel's intensity histogram. sample_std (bool) – If True, sample the spread of the Gaussian distributions uniformly from 0 to std. dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. Apply the transform to img. Union[ndarray, Tensor] Randomly select coarse regions in the image, then execute transform operations for the regions. It's the base class of all kinds of region transforms. Refer to papers: https://arxiv.org/abs/1708.04552 holes (int) – number of regions to dropout, if max_holes is not None, use this arg as the minimum number to randomly select the expected number of regions. spatial_size (Union[Sequence[int], int]) – spatial size of the regions to dropout, if max_spatial_size is not None, use this arg as the minimum spatial size to randomly select size for every region. if some components of the spatial_size are non-positive values, the transform will use the corresponding components of input img size. For example, spatial_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. max_holes (Optional[int]) – if not None, define the maximum number to randomly select the expected number of regions. max_spatial_size (Union[Sequence[int], int, None]) – if not None, define the maximum spatial size to randomly select size for every region. if some components of the max_spatial_size are non-positive values, the transform will use the corresponding components of input img size. For example, max_spatial_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. prob (float) – probability of applying the transform. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Union[ndarray, Tensor] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Randomly coarse dropout regions in the image, then fill in the rectangular regions with specified value. Or keep the rectangular regions and fill in the other areas with specified value. Refer to papers: https://arxiv.org/abs/1708.04552, https://arxiv.org/pdf/1604.07379 And other implementation: https://albumentations.ai/docs/api_reference/augmentations/transforms/#albumentations.augmentations.transforms.CoarseDropout. holes (int) – number of regions to dropout, if max_holes is not None, use this arg as the minimum number to randomly select the expected number of regions. spatial_size (Union[Sequence[int], int]) – spatial size of the regions to dropout, if max_spatial_size is not None, use this arg as the minimum spatial size to randomly select size for every region. if some components of the spatial_size are non-positive values, the transform will use the corresponding components of input img size. For example, spatial_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. dropout_holes (bool) – if True, dropout the regions of holes and fill value, if False, keep the holes and dropout the outside and fill value. default to True. fill_value (Union[Tuple[float, float], float, None]) – target value to fill the dropout regions, if providing a number, will use it as constant value to fill all the regions. if providing a tuple for the min and max, will randomly select value for every pixel / voxel from the range [min, max). if None, will compute the min and max value of input image then randomly select value to fill, default to None. max_holes (Optional[int]) – if not None, define the maximum number to randomly select the expected number of regions. max_spatial_size (Union[Sequence[int], int, None]) – if not None, define the maximum spatial size to randomly select size for every region. if some components of the max_spatial_size are non-positive values, the transform will use the corresponding components of input img size. For example, max_spatial_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. prob (float) – probability of applying the transform. Randomly select regions in the image, then shuffle the pixels within every region. It shuffles every channel separately. Refer to paper: Kang, Guoliang, et al. “Patchshuffle regularization.” arXiv preprint arXiv:1707.07103 (2017). https://arxiv.org/abs/1707.07103 holes (int) – number of regions to dropout, if max_holes is not None, use this arg as the minimum number to randomly select the expected number of regions. spatial_size (Union[Sequence[int], int]) – spatial size of the regions to dropout, if max_spatial_size is not None, use this arg as the minimum spatial size to randomly select size for every region. if some components of the spatial_size are non-positive values, the transform will use the corresponding components of input img size. For example, spatial_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. max_holes (Optional[int]) – if not None, define the maximum number to randomly select the expected number of regions. max_spatial_size (Union[Sequence[int], int, None]) – if not None, define the maximum spatial size to randomly select size for every region. if some components of the max_spatial_size are non-positive values, the transform will use the corresponding components of input img size. For example, max_spatial_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. prob (float) – probability of applying the transform. Apply the histogram normalization to input image. Refer to: facebookresearch/CovidPrognosis. num_bins (int) – number of the bins to use in histogram, default to 256. for more details: https://numpy.org/doc/stable/reference/generated/numpy.histogram.html. min (int) – the min value to normalize input image, default to 0. max (int) – the max value to normalize input image, default to 255. mask (Union[ndarray, Tensor, None]) – if provided, must be ndarray of bools or 0s and 1s, and same shape as image. only points at which mask==True are used for the equalization. can also provide the mask along with img at runtime. dtype (Union[dtype, type, str, None]) – data type of the output, if None, same as input image. default to float32. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Union[ndarray, Tensor] Creates a binary mask that defines the foreground based on thresholds in RGB or HSV color space. This transform receives an RGB (or grayscale) image where by default it is assumed that the foreground has low values (dark) while the background has high values (white). Otherwise, set invert argument to True. threshold (Union[Dict, Callable, str, float, int]) – an int or a float number that defines the threshold that values less than that are foreground. It also can be a callable that receives each dimension of the image and calculate the threshold, or a string that defines such callable from skimage.filter.threshold_…. For the list of available threshold functions, please refer to https://scikit-image.org/docs/stable/api/skimage.filters.html Moreover, a dictionary can be passed that defines such thresholds for each channel, like {“R”: 100, “G”: “otsu”, “B”: skimage.filter.threshold_mean} hsv_threshold (Union[Dict, Callable, str, float, int, None]) – similar to threshold but HSV color space (“H”, “S”, and “V”). Unlike RBG, in HSV, value greater than hsv_threshold are considered foreground. invert (bool) – invert the intensity range of the input image, so that the dtype maximum is now the dtype minimum, and vice-versa. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Compute horizontal and vertical maps from an instance mask It generates normalized horizontal and vertical distances to the center of mass of each region. Input data with the size of [1xHxW[xD]], which channel dim will temporarily removed for calculating coordinates. dtype (Union[dtype, type, str, None]) – the data type of output Tensor. Defaults to “float32”. A torch.Tensor with the size of [2xHxW[xD]], which is stack horizontal and vertical maps data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method.

IO#

Load image file or files from provided path based on reader. If reader is not specified, this class automatically chooses readers based on the supported suffixes and in the following order: User-specified reader at runtime when calling this loader. User-specified reader in the constructor of LoadImage. Readers from the last to the first in the registered list. Current default readers: (nii, nii.gz -> NibabelReader), (png, jpg, bmp -> PILReader), (npz, npy -> NumpyReader), (nrrd -> NrrdReader), (DICOM file -> ITKReader). Please note that for png, jpg, bmp, and other 2D formats, readers often swap axis 0 and 1 after loading the array because the HW definition for non-medical specific file formats is different from other common medical packages. See also tutorial: Project-MONAI/tutorials Load image file and metadata from the given filename(s). If reader is not specified, this class automatically chooses readers based on the reversed order of registered readers self.readers. filename (Union[Sequence[Union[str, PathLike]], str, PathLike]) – path file or file-like object or a list of files. will save the filename to meta_data with key filename_or_obj. if provided a list of files, use the filename of first file to save, and will stack them together as multi-channels data. if provided directory path instead of file path, will treat it as DICOM images series and read. reader (Optional[ImageReader]) – runtime reader to load image file and metadata. reader – reader to load image file and metadata - if reader is None, a default set of SUPPORTED_READERS will be used. - if reader is a string, it’s treated as a class name or dotted path (such as "monai.data.ITKReader"), the supported built-in reader classes are "ITKReader", "NibabelReader", "NumpyReader", "PydicomReader". a reader instance will be constructed with the *args and **kwargs parameters. - if reader is a reader class/instance, it will be registered to this loader accordingly. image_only (bool) – if True return only the image MetaTensor, otherwise return image and header dict. dtype (Union[dtype, type, str, None]) – if not None convert the loaded image to this data type. ensure_channel_first (bool) – if True and loaded both image array and metadata, automatically convert the image array shape to channel first. default to False. simple_keys (bool) – whether to remove redundant metadata keys, default to False for backward compatibility. prune_meta_pattern (Optional[str]) – combined with prune_meta_sep, a regular expression used to match and prune keys in the metadata (nested dictionary), default to None, no key deletion. prune_meta_sep (str) – combined with prune_meta_pattern, used to match and prune keys in the metadata (nested dictionary). default is “.”, see also monai.transforms.DeleteItemsd. e.g. prune_meta_pattern=".*_code$", prune_meta_sep=" " removes meta keys that ends with "_code". args – additional parameters for reader if providing a reader name. kwargs – additional parameters for reader if providing a reader name. Note The transform returns a MetaTensor, unless set_track_meta(False) has been used, in which case, a torch.Tensor will be returned. If reader is specified, the loader will attempt to use the specified readers and the default supported readers. This might introduce overheads when handling the exceptions of trying the incompatible loaders. In this case, it is therefore recommended setting the most appropriate reader as the last item of the reader parameter. Register image reader to load image file and metadata. reader (ImageReader) – reader instance to be registered with this loader. Save the image (in the form of torch tensor or numpy ndarray) and metadata dictionary into files. The name of saved file will be {input_image_name}_{output_postfix}{output_ext}, where the input_image_name is extracted from the provided metadata dictionary. If no metadata provided, a running index starting from 0 will be used as the filename prefix. output_dir (Union[str, PathLike]) – output image directory. output_postfix (str) – a string appended to all output file names, default to trans. output_ext (str) – output file extension name. output_dtype (Union[dtype, type, str, None]) – data type (if not None) for saving data. Defaults to np.float32. resample (bool) – whether to resample image (if needed) before saving the data array, based on the spatial_shape (and original_affine) from metadata. mode (str) – This option is used when resample=True. Defaults to "nearest". Depending on the writers, the possible options are {"bilinear", "nearest", "bicubic"}. See also: https://pytorch.org/docs/stable/nn.functional.html#grid-sample {"nearest", "linear", "bilinear", "bicubic", "trilinear", "area"}. See also: https://pytorch.org/docs/stable/nn.functional.html#interpolate This option is used when resample=True. Defaults to "nearest". Depending on the writers, the possible options are {"bilinear", "nearest", "bicubic"}. See also: https://pytorch.org/docs/stable/nn.functional.html#grid-sample {"nearest", "linear", "bilinear", "bicubic", "trilinear", "area"}. See also: https://pytorch.org/docs/stable/nn.functional.html#interpolate padding_mode (str) – This option is used when resample = True. Defaults to "border". Possible options are {"zeros", "border", "reflection"} See also: https://pytorch.org/docs/stable/nn.functional.html#grid-sample scale (Optional[int]) – {255, 65535} postprocess data by clipping to [0, 1] and scaling [0, 255] (uint8) or [0, 65535] (uint16). Default is None (no scaling). dtype (Union[dtype, type, str, None]) – data type during resampling computation. Defaults to np.float64 for best precision. if None, use the data type of input data. To set the output data type, use output_dtype. squeeze_end_dims (bool) – if True, any trailing singleton dimensions will be removed (after the channel has been moved to the end). So if input is (C,H,W,D), this will be altered to (H,W,D,C), and then if C==1, it will be saved as (H,W,D). If D is also 1, it will be saved as (H,W). If false, image will always be saved as (H,W,D,C). data_root_dir (Union[str, PathLike]) – if not empty, it specifies the beginning parts of the input file’s absolute path. It’s used to compute input_file_rel_path, the relative path to the file from data_root_dir to preserve folder structure when saving in case there are files in different folders with the same file names. For example, with the following inputs: input_file_name: /foo/bar/test1/image.nii output_postfix: seg output_ext: .nii.gz output_dir: /output data_root_dir: /foo/bar The output will be: /output/test1/image/image_seg.nii.gz if not empty, it specifies the beginning parts of the input file’s absolute path. It’s used to compute input_file_rel_path, the relative path to the file from data_root_dir to preserve folder structure when saving in case there are files in different folders with the same file names. For example, with the following inputs: input_file_name: /foo/bar/test1/image.nii output_postfix: seg output_ext: .nii.gz output_dir: /output data_root_dir: /foo/bar The output will be: /output/test1/image/image_seg.nii.gz separate_folder (bool) – whether to save every file in a separate folder. For example: for the input filename image.nii, postfix seg and folder_path output, if separate_folder=True, it will be saved as: output/image/image_seg.nii, if False, saving as output/image_seg.nii. Default to True. print_log (bool) – whether to print logs when saving. Default to True. output_format (str) – an optional string of filename extension to specify the output image writer. see also: monai.data.image_writer.SUPPORTED_WRITERS. writer (Union[Type[ImageWriter], str, None]) – a customised monai.data.ImageWriter subclass to save data arrays. if None, use the default writer from monai.data.image_writer according to output_ext. if it's a string, it's treated as a class name or dotted path (such as "monai.data.ITKWriter"); the supported built-in writer classes are "NibabelWriter", "ITKWriter", "PILWriter". channel_dim (Optional[int]) – the index of the channel dimension. Default to 0. None to indicate no channel dimension. output_name_formatter – a callable function (returning a kwargs dict) to format the output file name. see also: monai.data.folder_layout.default_name_formatter(). img (Union[Tensor, ndarray]) – target data content that save into file. The image should be channel-first, shape: [C,H,W,[D]]. meta_data (Optional[Dict]) – key-value pairs of metadata corresponding to the data. Set the options for the underlying writer by updating the self.*_kwargs dictionaries. The arguments correspond to the following usage: writer = ImageWriter(**init_kwargs) writer.set_data_array(array, **data_kwargs) writer.set_metadata(meta_data, **meta_kwargs) writer.write(filename, **write_kwargs)

NVIDIA Tool Extension (NVTX)#

Pushes a range onto a stack of nested range span. Stores zero-based depth of the range that is started. msg (str) – ASCII message to associate with range Pushes a range onto a stack of nested range span (for randomizable transforms). Stores zero-based depth of the range that is started. msg (str) – ASCII message to associate with range Pops a range off of a stack of nested range spans. Stores zero-based depth of the range that is ended. Pops a range off of a stack of nested range spans (for randomizable transforms). Stores zero-based depth of the range that is ended. Mark an instantaneous event that occurred at some point. msg (str) – ASCII message to associate with the event. Mark an instantaneous event that occurred at some point (for randomizable transforms). msg (str) – ASCII message to associate with the event.

Post-processing#

Activation operations, typically Sigmoid or Softmax. sigmoid (bool) – whether to execute sigmoid function on model output before transform. Defaults to False. softmax (bool) – whether to execute softmax function on model output before transform. Defaults to False. other (Optional[Callable]) – callable function to execute other activation layers, for example: other = lambda x: torch.tanh(x). Defaults to None. kwargs – additional parameters to torch.softmax (used when softmax=True). Defaults to dim=0, unrecognized parameters will be ignored. TypeError – When other is not an Optional[Callable]. sigmoid (Optional[bool]) – whether to execute sigmoid function on model output before transform. Defaults to self.sigmoid. softmax (Optional[bool]) –

whether to execute softmax function on model output before transform. Defaults to self.softmax. other (Optional[Callable]) – callable function to execute other activation layers, for example: other = torch.tanh. Defaults to self.other. ValueError – When sigmoid=True and softmax=True. Incompatible values. TypeError – When other is not an Optional[Callable]. ValueError – When self.other=None and other=None. Incompatible values. Union[ndarray, Tensor] Convert the input tensor/array into discrete values, possible operations are: argmax. threshold input value to binary values. convert input value to One-Hot format (set to_one_hot=N, N is the number of classes). round the value to the closest integer. argmax (bool) – whether to execute argmax function on input data before transform. Defaults to False. to_onehot (Optional[int]) – if not None, convert input data into the one-hot format with specified number of classes. Defaults to None. threshold (Optional[float]) – if not None, threshold the float values to int number 0 or 1 with specified threshold. Defaults to None. rounding (Optional[str]) – if not None, round the data according to the specified option, available options: [“torchrounding”]. kwargs – additional parameters to torch.argmax, monai.networks.one_hot. currently dim, keepdim, dtype are supported, unrecognized parameters will be ignored. These default to 0, True, torch.float respectively. Example img (Union[ndarray, Tensor]) – the input tensor data to convert, if no channel dimension when converting to One-Hot, will automatically add it. argmax (Optional[bool]) – whether to execute argmax function on input data before transform. Defaults to self.argmax. to_onehot (Optional[int]) – if not None, convert input data into the one-hot format with specified number of classes. Defaults to self.to_onehot. threshold (Optional[float]) – if not None, threshold the float values to int number 0 or 1 with specified threshold value. Defaults to self.threshold. rounding (Optional[str]) – if not None, round the data according to the specified option, available options: [“torchrounding”]. Union[ndarray, Tensor] Keeps only the largest connected component in the image. This transform can be used as a post-processing step to clean up over-segment areas in model output. 1) For not OneHot format data, the values correspond to expected labels, 0 will be treated as background and the over-segment pixels will be set to 0. 2) For OneHot format data, the values should be 0, 1 on each labels, the over-segment pixels will be set to 0 in its channel. For example: Use with applied_labels=[1], is_onehot=False, connectivity=1: Use with applied_labels=[1, 2], is_onehot=False, independent=False, connectivity=1: Use with applied_labels=[1, 2], is_onehot=False, independent=True, connectivity=1: Use with applied_labels=[1, 2], is_onehot=False, independent=False, connectivity=2: img (Union[ndarray, Tensor]) – shape must be (C, spatial_dim1[, spatial_dim2, …]). Union[ndarray, Tensor] An array with shape (C, spatial_dim1[, spatial_dim2, …]). applied_labels (Union[Sequence[int], int, None]) – Labels for applying the connected component analysis on. If given, voxels whose value is in this list will be analyzed. If None, all non-zero values will be analyzed. is_onehot (Optional[bool]) – if True, treat the input data as OneHot format data, otherwise, not OneHot format data. default to None, which treats multi-channel data as OneHot and single channel data as not OneHot. independent (bool) – whether to treat applied_labels as a union of foreground labels. If True, the connected component analysis will be performed on each foreground label independently and return the intersection of the largest components. If False, the analysis will be performed on the union of foreground labels. default is True. connectivity (Optional[int]) – Maximum number of orthogonal hops to consider a pixel/voxel as a neighbor. Accepted values are ranging from 1 to input.ndim. If None, a full connectivity of input.ndim is used. for more details: https://scikit-image.org/docs/dev/api/skimage.measure.html#skimage.measure.label. num_components (int) – The number of largest components to preserve. Use skimage.morphology.remove_small_objects to remove small objects from images. See:

https://scikit-image.org/docs/dev/api/skimage.morphology.html#remove-small-objects. Data should be one-hotted. min_size (int) – objects smaller than this size (in pixel) are removed. connectivity (int) – Maximum number of orthogonal hops to consider a pixel/voxel as a neighbor. Accepted values are ranging from 1 to input.ndim. If None, a full connectivity of input.ndim is used. For more details refer to linked scikit-image documentation. independent_channels (bool) – Whether or not to consider channels as independent. If true, then conjoining islands from different labels will be removed if they are below the threshold. If false, the overall size islands made from all non-background voxels will be used. img (Union[ndarray, Tensor]) – shape must be (C, spatial_dim1[, spatial_dim2, …]). Data should be one-hotted. Union[ndarray, Tensor] An array with shape (C, spatial_dim1[, spatial_dim2, …]). This transform filters out labels and can be used as a processing step to view only certain labels. The list of applied labels defines which labels will be kept. Note All labels which do not match the applied_labels are set to the background label (0). For example: Use LabelFilter with applied_labels=[1, 5, 9]: Filter the image on the applied_labels. img (Union[ndarray, Tensor]) – Pytorch tensor or numpy array of any shape. NotImplementedError – The provided image was not a Pytorch Tensor or numpy array. Union[ndarray, Tensor] Pytorch tensor or numpy array of the same shape as the input. Initialize the LabelFilter class with the labels to filter on. applied_labels (Union[Iterable[int], int]) – Label(s) to filter on. This transform fills holes in the image and can be used to remove artifacts inside segments. An enclosed hole is defined as a background pixel/voxel which is only enclosed by a single class. The definition of enclosed can be defined with the connectivity parameter: It is possible to define for which labels the hole filling should be applied. The input image is assumed to be a PyTorch Tensor or numpy array with shape [C, spatial_dim1[, spatial_dim2, …]]. If C = 1, then the values correspond to expected labels. If C > 1, then a one-hot-encoding is expected where the index of C matches the label indexing. Note The label 0 will be treated as background and the enclosed holes will be set to the neighboring class label. The performance of this method heavily depends on the number of labels. It is a bit faster if the list of applied_labels is provided. Limiting the number of applied_labels results in a big decrease in processing time. For example: Use FillHoles with default parameters: The hole in label 1 is fully enclosed and therefore filled with label 1. The background label near label 2 and 3 is not fully enclosed and therefore not filled. Fill the holes in the provided image. Note The value 0 is assumed as background label. img (Union[ndarray, Tensor]) – Pytorch Tensor or numpy array of shape [C, spatial_dim1[, spatial_dim2, …]]. NotImplementedError – The provided image was not a Pytorch Tensor or numpy array. Union[ndarray, Tensor] Pytorch Tensor or numpy array of shape [C, spatial_dim1[, spatial_dim2, …]]. Initialize the connectivity and limit the labels for which holes are filled. applied_labels (Union[Iterable[int], int, None]) – Labels for which to fill holes. Defaults to None, that is filling holes for all labels. connectivity (Optional[int]) – Maximum number of orthogonal hops to consider a pixel/voxel as a neighbor. Accepted values are ranging from 1 to input.ndim. Defaults to a full connectivity of input.ndim. Return the contour of binary input images that only compose of 0 and 1, with Laplacian kernel set as default for edge detection. Typical usage is to plot the edge of label or segmentation output. kernel_type (str) – the method applied to do edge detection, default is “Laplace”. NotImplementedError – When kernel_type is not “Laplace”. img (Union[ndarray, Tensor]) – torch tensor data to extract the contour, with shape: [channels, height, width[, depth]] ValueError – When image ndim is not one of [3, 4]. it’s the binary classification result of whether a pixel is edge or not. in order to keep the original shape of mask image, we use padding as default. the edge detection is just approximate because it defects inherent to Laplace kernel, ideally the edge should be thin enough, but now it has a thickness. it’s the binary classification result of whether a pixel is edge or not. in order to keep the original shape of mask image, we use padding as default. the edge detection is just approximate because it defects inherent to Laplace kernel, ideally the edge should be thin enough, but now it has a thickness. A torch tensor with the same shape as img, note Execute mean ensemble on the input data. The input data can be a list or tuple of PyTorch Tensor with shape: [C[, H, W, D]], Or a single PyTorch Tensor with shape: [E, C[, H, W, D]], the E dimension represents the output data from different models. Typically, the input data is model output of segmentation task or classification task. And it also can support to add weights for the input data. weights (Union[Sequence[float], ndarray, Tensor, None]) – can be a list or tuple of numbers for input data with shape: [E, C, H, W[, D]]. or a Numpy ndarray or a PyTorch Tensor data. the weights will be added to input data from highest dimension, for example: 1. if the weights only has 1 dimension, it will be added to the E dimension of input data. 2. if the weights has 2 dimensions, it will be added to E and C dimensions. it’s a typical practice to add weights for different classes: to ensemble 3 segmentation model outputs, every output has 4 channels(classes), so the input data shape can be: [3, 4, H, W, D]. and add different weights for different classes, so the weights shape can be: [3, 4]. for example: weights = [[1, 2, 3, 4], [4, 3, 2, 1], [1, 1, 1, 1]]. Call self as a function. Union[ndarray, Tensor] Performs probability based non-maximum suppression (NMS) on the probabilities map via iteratively selecting the coordinate with highest probability and then move it as well as its surrounding values. The remove range is determined by the parameter box_size. If multiple coordinates have the same highest probability, only one of them will be selected. spatial_dims (int) – number of spatial dimensions of the input probabilities map. Defaults to 2. sigma (Union[Sequence[float], float, Sequence[Tensor], Tensor]) – the standard deviation for gaussian filter. It could be a single value, or spatial_dims number of values. Defaults to 0.0. prob_threshold (float) – the probability threshold, the function will stop searching if the highest probability is no larger than the threshold. The value should be no less than 0.0. Defaults to 0.5. box_size (Union[int, Sequence[int]]) – the box size (in pixel) to be removed around the pixel with the maximum probability. It can be an integer that defines the size of a square or cube, or a list containing different values for each dimensions. Defaults to 48. a list of selected lists, where inner lists contain probability and coordinates. For example, for 3D input, the inner lists are in the form of [probability, x, y, z]. ValueError – When prob_threshold is less than 0.0. ValueError – When box_size is a list or tuple, and its length is not equal to spatial_dims. ValueError – When box_size has a less than 1 value. Calculate Sobel gradients of a grayscale image with the shape of CxH[xWxDx…] or BxH[xWxDx…]. kernel_size (int) – the size of the Sobel kernel. Defaults to 3. spatial_axes (Union[Sequence[int], int, None]) – the axes that define the direction of the gradient to be calculated. It calculate the gradient along each of the provide axis. By default it calculate the gradient for all spatial axes. normalize_kernels (bool) – if normalize the Sobel kernel to provide proper gradients. Defaults to True. normalize_gradients (bool) – if normalize the output gradient to 0 and 1. Defaults to False. padding_mode (str) – the padding mode of the image when convolving with Sobel kernels. Defaults to “reflect”. Acceptable values are 'zeros', 'reflect', 'replicate' or 'circular'. See torch.nn.Conv1d() for more information. dtype (dtype) – kernel data type (torch.dtype). Defaults to torch.float32. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Tensor Execute vote ensemble on the input data. The input data can be a list or tuple of PyTorch Tensor with shape: [C[, H, W, D]], Or a single PyTorch Tensor with shape: [E[, C, H, W, D]], the E dimension represents the output data from different models. Typically, the input data is model output of segmentation task or classification task. Note This vote transform expects the input data is discrete values. It can be multiple channels data in One-Hot format or single channel data. It will vote to select the most common data between items. The output data has the same shape as every item of the input data. num_classes (Optional[int]) – if the input is single channel data instead of One-Hot, we can't get class number from channel, need to explicitly specify the number of classes to vote. Call self as a function. Union[ndarray, Tensor]

Signal#

Randomly drop a portion of a signal signal (Union[ndarray, Tensor]) – input 1 dimension signal to be dropped Union[ndarray, Tensor] boundaries (Sequence[float]) – list defining lower and upper boundaries for the signal drop, default (lower and upper values need to be positive) – [0.0, 1.0] Apply a random rescaling on a signal signal (Union[ndarray, Tensor]) – input 1 dimension signal to be scaled Union[ndarray, Tensor] boundaries (Sequence[float]) – list defining lower and upper boundaries for the signal scaling, default : [-1.0, 1.0] Apply a random shift on a signal signal (Union[ndarray, Tensor]) – input 1 dimension signal to be shifted Union[ndarray, Tensor] mode (Optional[str]) – define how the extension of the input array is done beyond its boundaries, see for more details : https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.shift.html. filling (Optional[float]) – value to fill past edges of input if mode is 'constant'. Default is 0.0. see for mode details : https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.shift.html. boundaries (Sequence[float]) – list defining lower and upper boundaries for the signal shift, default : [-1.0, 1.0] Add a random sinusoidal signal to the input signal signal (Union[ndarray, Tensor]) – input 1 dimension signal to which sinusoidal signal will be added Union[ndarray, Tensor] boundaries (Sequence[float]) – list defining lower and upper boundaries for the sinusoidal magnitude, lower and upper values need to be positive ,default : [0.1, 0.3] frequencies (Sequence[float]) – list defining lower and upper frequencies for sinusoidal signal generation ,default : [0.001, 0.02] Add a random square pulse signal to the input signal signal (Union[ndarray, Tensor]) – input 1 dimension signal to which square pulse will be added Union[ndarray, Tensor] boundaries (Sequence[float]) – list defining lower and upper boundaries for the square pulse magnitude, lower and upper values need to be positive , default : [0.01, 0.2] frequencies (Sequence[float]) – list defining lower and upper frequencies for the square pulse signal generation , default : [0.001, 0.02] Add a random gaussian noise to the input signal signal (Union[ndarray, Tensor]) – input 1 dimension signal to which gaussian noise will be added Union[ndarray, Tensor] boundaries (Sequence[float]) – list defining lower and upper boundaries for the signal magnitude, default : [0.001,0.02] Add a random partial sinusoidal signal to the input signal signal (Union[ndarray, Tensor]) – input 1 dimension signal to which a partial sinusoidal signal added (will be) – Union[ndarray, Tensor] boundaries (Sequence[float]) – list defining lower and upper boundaries for the sinusoidal magnitude, lower and upper values need to be positive , default : [0.1, 0.3] frequencies (Sequence[float]) – list defining lower and upper frequencies for sinusoidal signal generation , default : [0.001, 0.02] fraction (Sequence[float]) – list defining lower and upper boundaries for partial signal generation default : [0.01, 0.2] Add a random partial square pulse to a signal signal (Union[ndarray, Tensor]) – input 1 dimension signal to which a partial square pulse will be added Union[ndarray, Tensor] boundaries (Sequence[float]) – list defining lower and upper boundaries for the square pulse magnitude, lower and upper values need to be positive , default : [0.01, 0.2] frequencies (Sequence[float]) – list defining lower and upper frequencies for square pulse signal generation example : [0.001, 0.02] fraction (Sequence[float]) – list defining lower and upper boundaries for partial square pulse generation default: [0.01, 0.2] replace empty part of a signal (NaN) signal (Union[ndarray, Tensor]) – signal to be filled Union[ndarray, Tensor] replacement (float) – value to replace nan items in signal Remove a frequency from a signal signal (ndarray) – signal to be frequency removed Any frequency (Optional[float]) – frequency to be removed from the signal quality_factor (Optional[float]) – quality factor for notch filter see : https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.iirnotch.html sampling_freq (Optional[float]) – sampling frequency of the input signal Generate continuous wavelet transform of a signal signal (ndarray) – signal for which to generate continuous wavelet transform Any type (str) – mother wavelet type. Available options are: {"mexh", "morl", "cmorB-C", , "gausP"} see – https://pywavelets.readthedocs.io/en/latest/ref/cwt.html length (float) – expected length, default 125.0 frequency (float) – signal frequency, default 500.0

Spatial#

Resample input image from the orientation/spacing defined by src_affine affine matrix into the ones specified by dst_affine affine matrix. Internally this transform computes the affine transform matrix from src_affine to dst_affine, by xform = linalg.solve(src_affine, dst_affine), and call monai.transforms.Affine with xform. img (Tensor) – input image to be resampled. It currently supports channel-first arrays with at most three spatial dimensions. dst_affine (Optional[Tensor]) – destination affine matrix. Defaults to None, which means the same as img.affine. the shape should be (r+1, r+1) where r is the spatial rank of img. when dst_affine and spatial_size are None, the input will be returned without resampling, but the data type will be float32. spatial_size (Union[Sequence[int], Tensor, int, None]) – output image spatial size. if spatial_size and self.spatial_size are not defined, the transform will compute a spatial size automatically containing the previous field of view. if spatial_size is -1 are the transform will use the corresponding input img size. mode (Union[str, int, None]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to self.mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it’s an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html padding_mode (Optional[str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to self.padding_mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html align_corners (Optional[bool]) – Geometrically, we consider the pixels of the input as squares rather than points. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html

Defaults to None, effectively using the value of self.align_corners. dtype (Union[dtype, type, str, None]) – data type for resampling computation. Defaults to self.dtype or np.float64 (for best precision). If None, use the data type of input data. To be compatible with other modules, the output data type is always float32. The spatial rank is determined by the smallest among img.ndim -1, len(src_affine) - 1, and 3. When both monai.config.USE_COMPILED and align_corners are set to True, MONAI's resampling implementation will be used. Set dst_affine and spatial_size to None to turn off the resampling step. Tensor mode (Union[str, int]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it's an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html padding_mode (str) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "border". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {'reflect', 'grid-mirror', 'constant', 'grid-constant', 'nearest', 'mirror', 'grid-wrap', 'wrap'}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html dtype (Union[dtype, type, str, None]) – data type for resampling computation. Defaults to float64 for best precision. If None, use the data type of input data. To be compatible with other modules, the output data type is always float32. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Tensor Resample an image to match given metadata. The affine matrix will be aligned, and the size of the output image will match. img (Tensor) – input image to be resampled to match dst_meta. It currently supports channel-first arrays with at most three spatial dimensions. src_meta (Optional[Dict]) – Dictionary containing the source affine matrix in the form {'affine':src_affine}. If affine is not specified, an identity matrix is assumed. Defaults to None. See also: https://docs.monai.io/en/stable/transforms.html#spatialresample dst_meta (Optional[Dict]) – Dictionary containing the target affine matrix and target spatial shape in the form {'affine':src_affine, 'spatial_shape':spatial_size}. If affine is not specified, src_affine is assumed. If spatial_shape is not specified, spatial size is automatically computed, containing the previous field of view. Defaults to None. See also: https://docs.monai.io/en/stable/transforms.html#spatialresample mode (Union[str, int, None]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it's an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html padding_mode (Optional[str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "border". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {'reflect', 'grid-mirror', 'constant', 'grid-constant', 'nearest', 'mirror', 'grid-wrap', 'wrap'}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html align_corners (Optional[bool]) – Geometrically, we consider the pixels of the input as squares rather than points. Defaults to None, effectively using the value of self.align_corners. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html dtype (Union[dtype, type, str, None]) – data type for resampling computation. Defaults to self.dtype or np.float64 (for best precision). If None, use the data type of input data. To be compatible with other modules, the output data type is always float32. RuntimeError – When src_meta is missing. RuntimeError – When dst_meta is missing. ValueError – When the affine matrix of the source image is not invertible. Tensor Resampled input tensor or MetaTensor. Resample input image into the specified pixdim. data_array (Tensor) – in shape (num_channels, H[, W, …]). mode (Union[str, int, None]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to "self.mode". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it's an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html padding_mode (Optional[str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "self.padding_mode". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {'reflect', 'grid-mirror', 'constant', 'grid-constant', 'nearest', 'mirror', 'grid-wrap', 'wrap'}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html align_corners (Optional[bool]) – Geometrically, we consider the pixels of the input as squares rather than points. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html Defaults to None, effectively using the value of self.align_corners. dtype (Union[dtype, type, str, None]) – data type for resampling computation. Defaults to self.dtype. If None, use the data type of input data. To be compatible with other modules, the output data type is always float32. scale_extent (Optional[bool]) – whether the scale is computed based on the spacing or the full extent of voxels, The option is ignored if output spatial size is specified when calling this transform. See also: monai.data.utils.compute_shape_offset(). When this is True, align_corners should be True because compute_shape_offset already provides the corner alignment shift/scaling. output_spatial_shape (Union[Sequence[int], ndarray, int, None]) – specify the shape of the output data_array. This is typically useful for the inverse of Spacingd where sometimes we could not compute the exact shape due to the quantization error with the affine. ValueError – When data_array has no spatial dimensions. ValueError – When pixdim is nonpositive. Tensor data tensor or MetaTensor (resampled into self.pixdim). pixdim (Union[Sequence[float], float, ndarray]) – output voxel spacing. if providing a single number, will use it for the first dimension. items of the pixdim sequence map to the spatial dimensions of input image, if length of pixdim sequence is longer than image spatial dimensions, will ignore the longer part, if shorter, will pad with the last value. For example, for 3D image if pixdim is [1.0, 2.0] it will be padded to [1.0, 2.0, 2.0] if the components of the pixdim are non-positive values, the transform will use the corresponding components of the original pixdim, which is computed from the affine matrix of input image. diagonal (bool) – whether to resample the input to have a diagonal affine matrix. If True, the input data is resampled to the following affine: np.diag((pixdim_0, pixdim_1, ..., pixdim_n, 1)) This effectively resets the volume to the world coordinate system (RAS+ in nibabel). The original orientation, rotation, shearing are not preserved. If False, this transform preserves the axes orientation, orthogonal rotation and translation components from the original affine. This option will not flip/swap axes of the original data. whether to resample the input to have a diagonal affine matrix. If True, the input data is resampled to the following affine: This effectively resets the volume to the world coordinate system (RAS+ in nibabel). The original orientation, rotation, shearing are not preserved. If False, this transform preserves the axes orientation, orthogonal rotation and translation components from the original affine.

This option will not flip/swap axes of the original data. mode (Union[str, int]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it’s an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html padding_mode (str) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "border". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html align_corners (bool) – Geometrically, we consider the pixels of the input as squares rather than points. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html dtype (Union[dtype, type, str, None]) – data type for resampling computation. Defaults to float64 for best precision. If None, use the data type of input data. To be compatible with other modules, the output data type is always float32. scale_extent (bool) – whether the scale is computed based on the spacing or the full extent of voxels, default False. The option is ignored if output spatial size is specified when calling this transform. See also: monai.data.utils.compute_shape_offset(). When this is True, align_corners should be True because compute_shape_offset already provides the corner alignment shift/scaling. recompute_affine (bool) – whether to recompute affine based on the output shape. The affine computed analytically does not reflect the potential quantization errors in terms of the output shape. Set this flag to True to recompute the output affine based on the actual pixdim. Default to False. min_pixdim (Union[Sequence[float], float, ndarray, None]) – minimal input spacing to be resampled. If provided, input image with a larger spacing than this value will be kept in its original spacing (not be resampled to pixdim). Set it to None to use the value of pixdim. Default to None. max_pixdim (Union[Sequence[float], float, ndarray, None]) – maximal input spacing to be resampled. If provided, input image with a smaller spacing than this value will be kept in its original spacing (not be resampled to pixdim). Set it to None to use the value of pixdim. Default to None. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Tensor Change the input image’s orientation into the specified based on axcodes. If input type is MetaTensor, original affine is extracted with data_array.affine. If input type is torch.Tensor, original affine is assumed to be identity. data_array (Tensor) – in shape (num_channels, H[, W, …]). ValueError – When data_array has no spatial dimensions. ValueError – When axcodes spatiality differs from data_array. Tensor data_array [reoriented in self.axcodes]. Output type will be MetaTensorunless get_track_meta() == False, in which case it will be torch.Tensor. unless get_track_meta() == False, in which case it will be torch.Tensor. axcodes (Optional[str]) – N elements sequence for spatial ND input’s orientation. e.g. axcodes=’RAS’ represents 3D orientation: (Left, Right), (Posterior, Anterior), (Inferior, Superior). default orientation labels options are: ‘L’ and ‘R’ for the first dimension, ‘P’ and ‘A’ for the second, ‘I’ and ‘S’ for the third. as_closest_canonical (bool) – if True, load the image as closest to canonical axis format. labels (Optional[Sequence[Tuple[str, str]]]) – optional, None or sequence of (2,) sequences (2,) sequences are labels for (beginning, end) of output axis. Defaults to (('L', 'R'), ('P', 'A'), ('I', 'S')). ValueError – When axcodes=None and as_closest_canonical=True. Incompatible values. See Also: nibabel.orientations.ornt2axcodes. Inverse of __call__. NotImplementedError – When the subclass does not override this method.

Tensor Randomly rotate the input arrays. range_x (Union[Tuple[float, float], float]) – Range of rotation angle in radians in the plane defined by the first and second axes. If single number, angle is uniformly sampled from (-range_x, range_x). range_y (Union[Tuple[float, float], float]) – Range of rotation angle in radians in the plane defined by the first and third axes. If single number, angle is uniformly sampled from (-range_y, range_y). only work for 3D data. range_z (Union[Tuple[float, float], float]) – Range of rotation angle in radians in the plane defined by the second and third axes. If single number, angle is uniformly sampled from (-range_z, range_z). only work for 3D data. prob (float) – Probability of rotation. keep_size (bool) – If it is False, the output shape is adapted so that the input array is contained completely in the output. If it is True, the output shape is the same as the input. Default is True. mode (str) – {"bilinear", "nearest"} Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html padding_mode (str) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "border". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html align_corners (bool) – Defaults to False. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html dtype (Union[dtype, type, str, None, dtype]) – data type for resampling computation. Defaults to float32. If None, use the data type of input data. To be compatible with other modules, the output data type is always float32. img (Tensor) – channel first array, must have shape 2D: (nchannels, H, W), or 3D: (nchannels, H, W, D). mode (Optional[str]) – {"bilinear", "nearest"} Interpolation mode to calculate output values. Defaults to self.mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html padding_mode (Optional[str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to self.padding_mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html align_corners (Optional[bool]) – Defaults to self.align_corners. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html dtype (Union[dtype, type, str, None, dtype]) – data type for resampling computation. Defaults to self.dtype. If None, use the data type of input data. To be compatible with other modules, the output data type is always float32. randomize (bool) – whether to execute randomize() function first, default to True. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Tensor Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Randomly flips the image along axes. Preserves shape. See numpy.flip for additional details. https://docs.scipy.org/doc/numpy/reference/generated/numpy.flip.html prob (float) – Probability of flipping. spatial_axis (Union[Sequence[int], int, None]) – Spatial axes along which to flip over. Default is None. img (Tensor) – channel first array, must have shape: (num_channels, H[, W, …,]), randomize (bool) – whether to execute randomize() function first, default to True. Tensor Inverse of __call__. NotImplementedError – When the subclass does not override this method. Tensor Randomly select a spatial axis and flip along it. See numpy.flip for additional details. https://docs.scipy.org/doc/numpy/reference/generated/numpy.flip.html prob (float) – Probability of flipping. img (Tensor) – channel first array, must have shape: (num_channels, H[, W, …,]) randomize (bool) – whether to execute randomize() function first, default to True. Tensor Inverse of __call__. NotImplementedError – When the subclass does not override this method. Tensor Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Randomly zooms input arrays with given probability within given zoom range. prob (float) – Probability of zooming. min_zoom (Union[Sequence[float], float]) – Min zoom factor. Can be float or sequence same size as image. If a float, select a random factor from [min_zoom, max_zoom] then apply to all spatial dims to keep the original spatial shape ratio. If a sequence, min_zoom should contain one value for each spatial axis. If 2 values provided for 3D data, use the first value for both H & W dims to keep the same zoom ratio. max_zoom (Union[Sequence[float], float]) – Max zoom factor. Can be float or sequence same size as image. If a float, select a random factor from [min_zoom, max_zoom] then apply to all spatial dims to keep the original spatial shape ratio. If a sequence, max_zoom should contain one value for each spatial axis. If 2 values provided for 3D data, use the first value for both H & W dims to keep the same zoom ratio. mode (str) – {"nearest", "nearest-exact", "linear", "bilinear", "bicubic", "trilinear", "area"} The interpolation mode. Defaults to "area". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html padding_mode (str) – available modes for numpy array:{"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "constant". The mode to pad data after zooming. See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html align_corners (Optional[bool]) – This only has an effect when mode is 'linear', 'bilinear', 'bicubic' or 'trilinear'. Default: None. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html keep_size (bool) – Should keep original size (pad if needed), default is True. kwargs – other arguments for the np.pad or torch.pad function. note that np.pad treats channel dimension as the first dimension. img (Tensor) – channel first array, must have shape 2D: (nchannels, H, W), or 3D: (nchannels, H, W, D). mode (Optional[str]) – {"nearest", "nearest-exact", "linear", "bilinear", "bicubic", "trilinear", "area"}, the interpolation mode. Defaults to self.mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html padding_mode (Optional[str]) – available modes for numpy array:{"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "constant". The mode to pad data after zooming. See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html align_corners (Optional[bool]) – This only has an effect when mode is 'linear', 'bilinear', 'bicubic' or 'trilinear'. Defaults to self.align_corners. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html randomize (bool) – whether to execute randomize() function first, default to True. Tensor Inverse of __call__. NotImplementedError – When the subclass does not override this method. Tensor Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Transform img given the affine parameters. A tutorial is available: Project-MONAI/tutorials. img (Tensor) – shape must be (num_channels, H, W[, D]), spatial_size (Union[Sequence[int], int, None]) – output image spatial size. if spatial_size and self.spatial_size are not defined, or smaller than 1, the transform will use the spatial size of img. if img has two spatial dimensions, spatial_size should have 2 elements [h, w]. if img has three spatial dimensions, spatial_size should have 3 elements [h, w, d]. mode (Union[str, int, None]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to self.mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it's an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html padding_mode (Optional[str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to self.padding_mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {'reflect', 'grid-mirror', 'constant', 'grid-constant', 'nearest', 'mirror', 'grid-wrap', 'wrap'}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html Union[Tensor, Tuple[Tensor, Union[ndarray, Tensor]]] The affine transformations are applied in rotate, shear, translate, scale order. rotate_params (Union[Sequence[float], float, None]) – a rotation angle in radians, a scalar for 2D image, a tuple of 3 floats for 3D. Defaults to no rotation. shear_params (Union[Sequence[float], float, None]) – shearing factors for affine matrix, take a 3D affine as example: [[1.0, params[0], params[1], 0.0], [params[2], 1.0, params[3], 0.0], [params[4], params[5], 1.0, 0.0], [0.0, 0.0, 0.0, 1.0],] a tuple of 2 floats for 2D, a tuple of 6 floats for 3D. Defaults to no shearing. shearing factors for affine matrix, take a 3D affine as example: translate_params (Union[Sequence[float], float, None]) – a tuple of 2 floats for 2D, a tuple of 3 floats for 3D. Translation is in pixel/voxel relative to the center of the input image. Defaults to no translation. scale_params (Union[Sequence[float], float, None]) – scale factor for every spatial dims. a tuple of 2 floats for 2D, a tuple of 3 floats for 3D. Defaults to 1.0. affine (Union[ndarray, Tensor, None]) – If applied, ignore the params (rotate_params, etc.) and use the supplied matrix. Should be square with each side = num of image spatial dimensions + 1. spatial_size (Union[Sequence[int], int, None]) – output image spatial size. if spatial_size and self.spatial_size are not defined, or smaller than 1, the transform will use the spatial size of img. if some components of the spatial_size are non-positive values, the transform will use the corresponding components of img size. For example, spatial_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. mode (Union[str, int]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it's an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html padding_mode (str) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "reflection". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {'reflect', 'grid-mirror', 'constant', 'grid-constant', 'nearest', 'mirror', 'grid-wrap', 'wrap'}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html normalized (bool) – indicating whether the provided affine is defined to include a normalization transform converting the coordinates from [-(size-1)/2, (size-1)/2] (defined in create_grid) to [0, size - 1] or [-1, 1] in order to be compatible with the underlying resampling API. If normalized=False, additional coordinate normalization will be applied before resampling. See also:

monai.networks.utils.normalize_transform(). device (Optional[device]) – device on which the tensor will be allocated. dtype (Union[dtype, type, str, None]) – data type for resampling computation. Defaults to float32. If None, use the data type of input data. To be compatible with other modules, the output data type is always float32. image_only (bool) – if True return only the image volume, otherwise return (image, affine). Deprecated since version 0.8.1: norm_coords is deprecated, please use normalized instead (the new flag is a negation, i.e., norm_coords == not normalized). Inverse of __call__. NotImplementedError – When the subclass does not override this method. Tensor img (Tensor) – shape must be (num_channels, H, W[, D]). grid (Optional[Tensor]) – shape must be (3, H, W) for 2D or (4, H, W, D) for 3D. if norm_coords is True, the grid values must be in [-(size-1)/2, (size-1)/2]. if USE_COMPILED=True and norm_coords=False, grid values must be in [0, size-1]. if USE_COMPILED=False and norm_coords=False, grid values must be in [-1, 1]. mode (Union[str, int, None]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to self.mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When USE_COMPILED is True, this argument uses "nearest", "bilinear", "bicubic" to indicate 0, 1, 3 order interpolations. See also: https://docs.monai.io/en/stable/networks.html#grid-pull (experimental). When it’s an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html padding_mode (Optional[str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to self.padding_mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When USE_COMPILED is True, this argument uses an integer to represent the padding mode. See also: https://docs.monai.io/en/stable/networks.html#grid-pull (experimental). When mode is an integer, using numpy/cupy backends, this argument accepts {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage .map_coordinates.html dtype (Union[dtype, type, str, None]) – data type for resampling computation. Defaults to self.dtype. To be compatible with other modules, the output data type is always float32. See also monai.config.USE_COMPILED Tensor computes output image using values from img, locations from grid using pytorch. supports spatially 2D or 3D (num_channels, H, W[, D]). mode (Union[str, int]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When USE_COMPILED is True, this argument uses "nearest", "bilinear", "bicubic" to indicate 0, 1, 3 order interpolations. See also: https://docs.monai.io/en/stable/networks.html#grid-pull (experimental). When it’s an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html padding_mode (str) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "border". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When USE_COMPILED is True, this argument uses an integer to represent the padding mode. See also: https://docs.monai.io/en/stable/networks.html#grid-pull (experimental). When mode is an integer, using numpy/cupy backends, this argument accepts {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage .map_coordinates.html norm_coords (bool) – whether to normalize the coordinates from [-(size-1)/2, (size-1)/2] to [0, size - 1] (for monai/csrc implementation) or [-1, 1] (for torch grid_sample implementation) to be compatible with the underlying resampling API. device (Optional[device]) – device on which the tensor will be allocated. dtype (Union[dtype, type, str, None]) – data type for resampling computation. Defaults to float64 for best precision. If None, use the data type of input data. To be compatible with other modules, the output data type is always float32. Random affine transform. A tutorial is available: Project-MONAI/tutorials. img (Tensor) – shape must be (num_channels, H, W[, D]), spatial_size (Union[Sequence[int], int, None]) – output image spatial size. if spatial_size and self.spatial_size are not defined, or smaller than 1, the transform will use the spatial size of img. if img has two spatial dimensions, spatial_size should have 2 elements [h, w]. if img has three spatial dimensions, spatial_size should have 3 elements [h, w, d]. mode (Union[str, int, None]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to self.mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it's an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html padding_mode (Optional[str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to self.padding_mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {'reflect', 'grid-mirror', 'constant', 'grid-constant', 'nearest', 'mirror', 'grid-wrap', 'wrap'}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html randomize (bool) – whether to execute randomize() function first, default to True. grid – precomputed grid to be used (mainly to accelerate RandAffined). Tensor prob (float) – probability of returning a randomized affine grid. defaults to 0.1, with 10% chance returns a randomized grid. rotate_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – angle range in radians. If element i is a pair of (min, max) values, then uniform[-rotate_range[i][0], rotate_range[i][1]) will be used to generate the rotation parameter for the i`th spatial dimension. If not, `uniform[-rotate_range[i], rotate_range[i]) will be used. This can be altered on a per-dimension basis. E.g., ((0,3), 1, …): for dim0, rotation will be in range [0, 3], and for dim1 [-1, 1] will be used. Setting a single value will use [-x, x] for dim0 and nothing for the remaining dimensions. shear_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – shear range with format matching rotate_range, it defines the range to randomly select shearing factors(a tuple of 2 floats for 2D, a tuple of 6 floats for 3D) for affine matrix, take a 3D affine as example: [[1.0, params[0], params[1], 0.0], [params[2], 1.0, params[3], 0.0], [params[4], params[5], 1.0, 0.0], [0.0, 0.0, 0.0, 1.0],] shear range with format matching rotate_range, it defines the range to randomly select shearing factors(a tuple of 2 floats for 2D, a tuple of 6 floats for 3D) for affine matrix, take a 3D affine as example: translate_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – translate range with format matching rotate_range, it defines the range to randomly select pixel/voxel to translate for every spatial dims. scale_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – scaling range with format matching rotate_range. it defines the range to randomly select the scale factor to translate for every spatial dims. A value of 1.0 is added to the result. This allows 0 to correspond to no change (i.e., a scaling of 1.0). spatial_size (Union[Sequence[int], int, None]) – output image spatial size. if spatial_size and self.spatial_size are not defined, or smaller than 1, the transform will use the spatial size of img. if some components of the spatial_size are non-positive values, the transform will use the corresponding components of img size. For example, spatial_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. mode (Union[str, int]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to bilinear. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it’s an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html padding_mode (str) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to reflection. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates .html cache_grid (bool) – whether to cache the identity sampling grid. If the spatial size is not dynamically defined by input image, enabling this option could accelerate the transform. device (Optional[device]) – device on which the tensor will be allocated. See also RandAffineGrid for the random affine parameters configurations. Affine for the affine transformation parameters configurations. Return a cached or new identity grid depends on the availability. spatial_size (Sequence[int]) – non-dynamic spatial size Inverse of __call__. NotImplementedError – When the subclass does not override this method. Tensor Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandAffine a Randomizable instance. Generate random deformation grid. spatial_size (Sequence[int]) – spatial size of the grid. Tensor spacing (Union[Sequence[float], float]) – spacing of the grid in 2D or 3D. e.g., spacing=(1, 1) indicates pixel-wise deformation in 2D, spacing=(1, 1, 1) indicates voxel-wise deformation in 3D, spacing=(2, 2) indicates deformation field defined on every other pixel in 2D. magnitude_range (Tuple[float, float]) – the random offsets will be generated from uniform[magnitude[0], magnitude[1]). device (Optional[device]) – device to store the output grid data. Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None Affine transforms on the coordinates. rotate_params (Union[Sequence[float], float, None]) – a rotation angle in radians, a scalar for 2D image, a tuple of 3 floats for 3D. Defaults to no rotation. shear_params (Union[Sequence[float], float, None]) – shearing factors for affine matrix, take a 3D affine as example: [[1.0, params[0], params[1], 0.0], [params[2], 1.0, params[3], 0.0], [params[4], params[5], 1.0, 0.0], [0.0, 0.0, 0.0, 1.0],] a tuple of 2 floats for 2D, a tuple of 6 floats for 3D. Defaults to no shearing. shearing factors for affine matrix, take a 3D affine as example: translate_params (Union[Sequence[float], float, None]) – a tuple of 2 floats for 2D, a tuple of 3 floats for 3D. Translation is in pixel/voxel relative to the center of the input image. Defaults to no translation. scale_params (Union[Sequence[float], float, None]) – scale factor for every spatial dims. a tuple of 2 floats for 2D, a tuple of 3 floats for 3D. Defaults to 1.0. dtype (Union[dtype, type, str, None]) – data type for the grid computation. Defaults to float32. If None, use the data type of input data (if grid is provided). device (Optional[device]) – device on which the tensor will be allocated, if a new grid is generated. affine (Union[ndarray, Tensor, None]) – If applied, ignore the params (rotate_params, etc.) and use the supplied matrix. Should be square with each side = num of image spatial dimensions + 1. The grid can be initialized with a spatial_size parameter, or provided directly as grid. Therefore, either spatial_size or grid must be provided. When initialising from spatial_size, the backend “torch” will be used. spatial_size (Optional[Sequence[int]]) – output grid size. grid (Optional[Tensor]) – grid to be transformed. Shape must be (3, H, W) for 2D or (4, H, W, D) for 3D. ValueError – When grid=None and spatial_size=None. Incompatible values. Tuple[Tensor, Tensor] Generate randomised affine grid. spatial_size (Optional[Sequence[int]]) – output grid size. grid (Union[ndarray, Tensor, None]) – grid to be transformed. Shape must be (3, H, W) for 2D or (4, H, W, D) for 3D. randomize (bool) – boolean as to whether the grid parameters governing the grid should be randomized. Tensor a 2D (3xHxW) or 3D (4xHxWxD) grid. rotate_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – angle range in radians. If element i is a pair of (min, max) values, then uniform[-rotate_range[i][0], rotate_range[i][1]) will be used to generate the rotation parameter for the i`th spatial dimension. If not, `uniform[-rotate_range[i], rotate_range[i]) will be used. This can be altered on a per-dimension basis. E.g., ((0,3), 1, ...): for dim0, rotation will be in range [0, 3], and for dim1 [-1, 1] will be used. Setting a single value will use [-x, x] for dim0 and nothing for the remaining dimensions. shear_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – shear range with format matching rotate_range, it defines the range to randomly select shearing factors(a tuple of 2 floats for 2D, a tuple of 6 floats for 3D) for affine matrix, take a 3D affine as example: [[1.0, params[0], params[1], 0.0], [params[2], 1.0, params[3], 0.0], [params[4], params[5], 1.0, 0.0], [0.0, 0.0, 0.0, 1.0],] shear range with format matching rotate_range, it defines the range to randomly select shearing factors(a tuple of 2 floats for 2D, a tuple of 6 floats for 3D) for affine matrix, take a 3D affine as example: translate_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – translate range with format matching rotate_range, it defines the range to randomly select voxels to translate for every spatial dims. scale_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – scaling range with format matching rotate_range. it defines the range to randomly select the scale factor to translate for every spatial dims. A value of 1.0 is added to the result. This allows 0 to correspond to no change (i.e., a scaling of 1.0). device (Optional[device]) – device to store the output grid data. See also monai.transforms.utils.create_rotate() monai.transforms.utils.create_shear() monai.transforms.utils.create_translate() monai.transforms.utils.create_scale() Get the most recently applied transformation matrix Optional[Tensor] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None img (Tensor) – shape must be (num_channels, H, W[, D]). distort_steps (Optional[Sequence[Sequence]]) – This argument is a list of tuples, where each tuple contains the distort steps of the corresponding dimensions (in the order of H, W[, D]). The length of each tuple equals to num_cells + 1. Each value in the tuple represents the distort step of the related cell. mode (Optional[str]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to self.mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it’s an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html padding_mode (Optional[str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to self.padding_mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {'reflect', 'grid-mirror', 'constant', 'grid-constant', 'nearest', 'mirror', 'grid-wrap', 'wrap'}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates .html Tensor Grid distortion transform. Refer to: albumentations-team/albumentations num_cells (Union[Tuple[int], int]) – number of grid cells on each dimension. distort_steps (Sequence[Sequence[float]]) – This argument is a list of tuples, where each tuple contains the distort steps of the corresponding dimensions (in the order of H, W[, D]). The length of each tuple equals to num_cells + 1. Each value in the tuple represents the distort step of the related cell. mode (Union[str, int]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it's an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html padding_mode (str) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "border". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {'reflect', 'grid-mirror', 'constant', 'grid-constant', 'nearest', 'mirror', 'grid-wrap', 'wrap'}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates .html device (Optional[device]) – device on which the tensor will be allocated. img (Tensor) – shape must be (num_channels, H, W[, D]). mode (Optional[str]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to self.mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it's an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html padding_mode (Optional[str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to self.padding_mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {'reflect', 'grid-mirror', 'constant', 'grid-constant', 'nearest', 'mirror', 'grid-wrap', 'wrap'}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates .html randomize (bool) – whether to shuffle the random factors using randomize(), default to True. Tensor Random grid distortion transform. Refer to: albumentations-team/albumentations num_cells (Union[Tuple[int], int]) – number of grid cells on each dimension. prob (float) – probability of returning a randomized grid distortion transform. Defaults to 0.1. distort_limit (Union[Tuple[float, float], float]) – range to randomly distort. If single number, distort_limit is picked from (-distort_limit, distort_limit). Defaults to (-0.03, 0.03). mode (Union[str, int]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it's an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html padding_mode (str) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "border". See also:

https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates .html device (Optional[device]) – device on which the tensor will be allocated. Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Random elastic deformation and affine in 2D. A tutorial is available: Project-MONAI/tutorials. img (Tensor) – shape must be (num_channels, H, W), spatial_size (Union[Tuple[int, int], int, None]) – specifying output image spatial size [h, w]. if spatial_size and self.spatial_size are not defined, or smaller than 1, the transform will use the spatial size of img. mode (Union[str, int, None]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to self.mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it’s an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy. org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html padding_mode (Optional[str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to self.padding_mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates .html randomize (bool) – whether to execute randomize() function first, default to True. Tensor spacing (Union[Tuple[float, float], float]) – distance in between the control points. magnitude_range (Tuple[float, float]) – the random offsets will be generated from uniform[magnitude[0], magnitude[1]). prob (float) – probability of returning a randomized elastic transform. defaults to 0.1, with 10% chance returns a randomized elastic transform, otherwise returns a spatial_size centered area extracted from the input image. rotate_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – angle range in radians. If element i is a pair of (min, max) values, then uniform[-rotate_range[i][0], rotate_range[i][1]) will be used to generate the rotation parameter for the i`th spatial dimension. If not, `uniform[-rotate_range[i], rotate_range[i]) will be used. This can be altered on a per-dimension basis. E.g., ((0,3), 1, …): for dim0, rotation will be in range [0, 3], and for dim1 [-1, 1] will be used. Setting a single value will use [-x, x] for dim0 and nothing for the remaining dimensions. shear_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – shear range with format matching rotate_range, it defines the range to randomly select shearing factors(a tuple of 2 floats for 2D) for affine matrix, take a 2D affine as example: [[1.0, params[0], 0.0], [params[1], 1.0, 0.0], [0.0, 0.0, 1.0],] shear range with format matching rotate_range, it defines the range to randomly select shearing factors(a tuple of 2 floats for 2D) for affine matrix, take a 2D affine as example: translate_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – translate range with format matching rotate_range, it defines the range to randomly select pixel to translate for every spatial dims. scale_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – scaling range with format matching rotate_range. it defines the range to randomly select the scale factor to translate for every spatial dims. A value of 1.0 is added to the result. This allows 0 to correspond to no change (i.e., a scaling of 1.0). spatial_size (Union[Tuple[int, int], int, None]) – specifying output image spatial size [h, w]. if spatial_size and self.spatial_size are not defined, or smaller than 1, the transform will use the spatial

size of img. if some components of the spatial_size are non-positive values, the transform will use the corresponding components of img size. For example, spatial_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. mode (Union[str, int]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to "bilinear". See also:
https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it’s an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html
padding_mode (str) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "reflection". See also:
https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html device (Optional[device]) – device on which the tensor will be allocated. See also RandAffineGrid for the random affine parameters configurations. Affine for the affine transformation parameters configurations. Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. Rand2DElastic a Randomizable instance. Random elastic deformation and affine in 3D. A tutorial is available: Project-MONAI/tutorials. img (Tensor) – shape must be (num_channels, H, W, D), spatial_size (Union[Tuple[int, int, int], int, None]) – specifying spatial 3D output image spatial size [h, w, d]. if spatial_size and self.spatial_size are not defined, or smaller than 1, the transform will use the spatial size of img. mode (Union[str, int, None]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to self.mode. See also:
https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it’s an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html
padding_mode (Optional[str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to self.padding_mode. See also:
https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html randomize (bool) – whether to execute randomize() function first, default to True. Tensor sigma_range (Tuple[float, float]) – a Gaussian kernel with standard deviation sampled from uniform[sigma_range[0], sigma_range[1]) will be used to smooth the random offset grid. magnitude_range (Tuple[float, float]) – the random offsets on the grid will be generated from uniform[magnitude[0], magnitude[1]). prob (float) – probability of returning a randomized elastic transform. defaults to 0.1, with 10% chance returns a randomized elastic transform, otherwise returns a spatial_size centered area extracted from the input image. rotate_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – angle range in

radians. If element i is a pair of (min, max) values, then uniform[-rotate_range[i][0], rotate_range[i][1]) will be used to generate the rotation parameter for the i`th spatial dimension. If not, `uniform[-rotate_range[i], rotate_range[i]) will be used. This can be altered on a per-dimension basis. E.g., ((0,3), 1, …): for dim0, rotation will be in range [0, 3], and for dim1 [-1, 1] will be used. Setting a single value will use [-x, x] for dim0 and nothing for the remaining dimensions. shear_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – shear range with format matching rotate_range, it defines the range to randomly select shearing factors(a tuple of 6 floats for 3D) for affine matrix, take a 3D affine as example: [[1.0, params[0], params[1], 0.0], [params[2], 1.0, params[3], 0.0], [params[4], params[5], 1.0, 0.0], [0.0, 0.0, 0.0, 1.0],] shear range with format matching rotate_range, it defines the range to randomly select shearing factors(a tuple of 6 floats for 3D) for affine matrix, take a 3D affine as example: translate_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – translate range with format matching rotate_range, it defines the range to randomly select voxel to translate for every spatial dims. scale_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – scaling range with format matching rotate_range. it defines the range to randomly select the scale factor to translate for every spatial dims. A value of 1.0 is added to the result. This allows 0 to correspond to no change (i.e., a scaling of 1.0). spatial_size (Union[Tuple[int, int, int], int, None]) – specifying output image spatial size [h, w, d]. if spatial_size and self.spatial_size are not defined, or smaller than 1, the transform will use the spatial size of img. if some components of the spatial_size are non-positive values, the transform will use the corresponding components of img size. For example, spatial_size=(32, 32, -1) will be adapted to (32, 32, 64) if the third spatial dimension size of img is 64. mode (Union[str, int]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it’s an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html padding_mode (str) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "reflection". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html device (Optional[device]) – device on which the tensor will be allocated. See also RandAffineGrid for the random affine parameters configurations. Affine for the affine transformation parameters configurations. Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. Rand3DElastic a Randomizable instance. Rotate an array by 90 degrees in the plane specified by axes. See torch.rot90 for additional details: https://pytorch.org/docs/stable/generated/torch.rot90.html#torch-rot90. img (Tensor) – channel first array, must have shape: (num_channels, H[, W, …,]), Tensor k (int) – number of times to rotate by 90 degrees. spatial_axes (Tuple[int, int]) – 2 int numbers, defines the plane to rotate with 2 spatial axes. Default: (0, 1), this is the first two axis in spatial dimensions. If axis is negative it counts from the last to the first axis. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Tensor With probability prob, input arrays are rotated by 90 degrees in the plane specified by spatial_axes. img (Tensor) – channel first array, must have shape: (num_channels, H[, W, …,]), randomize (bool) – whether to execute randomize() function first, default to True. Tensor prob (float) – probability of rotating. (Default 0.1, with 10% probability it returns a rotated array) max_k (int) – number of rotations will be sampled from np.random.randint(max_k) + 1, (Default 3). spatial_axes (Tuple[int, int]) – 2 int numbers, defines the plane to rotate with 2 spatial axes. Default: (0, 1), this is the first two axis in spatial dimensions. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Tensor Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Reverses the order of elements along the given spatial axis. Preserves shape. See torch.flip documentation for additional details: https://pytorch.org/docs/stable/generated/torch.flip.html spatial_axis (Union[Sequence[int], int, None]) – spatial axes along which to flip over. Default is None. The default axis=None will flip over all of the axes of the input array. If axis is negative it counts from the last to the first axis. If axis is a tuple of ints, flipping is performed on all of the axes specified in the tuple. img (Tensor) – channel first array, must have shape: (num_channels, H[, W, …,]) Tensor Inverse of __call__. NotImplementedError – When the subclass does not override this method. Tensor Resize the input image to given spatial size (with scaling, not cropping/padding). Implemented using torch.nn.functional.interpolate. spatial_size (Union[Sequence[int], int]) – expected shape of spatial dimensions after resize operation. if some components of the spatial_size are non-positive values, the transform will use the corresponding components of img size. For example, spatial_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. size_mode (str) – should be “all” or “longest”, if “all”, will use spatial_size for all the spatial dims, if “longest”, rescale the image so that only the longest side is equal to specified spatial_size, which must be an int number in this case, keeping the aspect ratio of the initial image, refer to: https://albumentations.ai/docs/api_reference/augmentations/geometric/resize/#albumentations.augmentations.geometric.resize.LongestMaxSize. mode (str) – {"nearest", "nearest-exact", "linear", "bilinear", "bicubic", "trilinear", "area"} The interpolation mode. Defaults to "area". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html align_corners (Optional[bool]) – This only has an effect when mode is ‘linear’, ‘bilinear’, ‘bicubic’ or ‘trilinear’. Default: None. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html anti_aliasing (bool) – bool Whether to apply a Gaussian filter to smooth the image prior to downsampling. It is crucial to filter when downsampling the image to avoid aliasing artifacts. See also skimage.transform.resize anti_aliasing_sigma (Union[Sequence[float], float, None]) – {float, tuple of floats}, optional Standard deviation for Gaussian filtering used when anti-aliasing. By default, this value is chosen as (s - 1) / 2 where s is the downsampling factor, where s > 1. For the up-size case, s < 1, no anti-aliasing is performed prior to rescaling. img (Tensor) – channel first array, must have shape: (num_channels, H[, W, …,]). mode (Optional[str]) – {"nearest", "nearest-exact", "linear", "bilinear", "bicubic", "trilinear", "area"} The interpolation mode. Defaults to self.mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html align_corners (Optional[bool]) – This only has an effect when mode is ‘linear’, ‘bilinear’, ‘bicubic’ or ‘trilinear’. Defaults to self.align_corners. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html anti_aliasing (Optional[bool]) – bool, optional Whether to apply a Gaussian filter to smooth the image prior to downsampling. It is crucial to filter when downsampling the image to avoid aliasing artifacts. See also skimage.transform.resize anti_aliasing_sigma (Union[Sequence[float], float, None]) – {float, tuple of floats}, optional Standard deviation for Gaussian filtering used when anti-aliasing. By default, this value is chosen as (s - 1) / 2 where s is the downsampling factor, where s > 1. For the up-size case, s < 1, no anti-aliasing is performed prior to rescaling. ValueError – When self.spatial_size length is less than img spatial dimensions. Tensor Inverse of __call__. NotImplementedError – When the subclass does not override this method. Tensor Rotates an input image by given angle using monai.networks.layers.AffineTransform. angle (Union[Sequence[float], float]) – Rotation angle(s) in radians. should a float for 2D, three floats for 3D. keep_size (bool) – If it is True, the output shape is kept the same as the input. If it is False, the output shape is adapted so that the input array is contained completely in the output. Default is True. mode (str) – {"bilinear", "nearest"} Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html padding_mode (str) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "border". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html align_corners (bool) – Defaults to False. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html dtype (Union[dtype, type, str, None, dtype]) – data type for resampling computation. Defaults to float32. If None, use the data type of input data. To be compatible with other modules, the output data type is always float32. img (Tensor) – channel first array, must have shape: [chns, H, W] or [chns, H, W, D]. mode (Optional[str]) – {"bilinear", "nearest"} Interpolation mode to calculate output values. Defaults to self.mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html padding_mode (Optional[str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to self.padding_mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html align_corners: Defaults to self.align_corners. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html align_corners (Optional[bool]) – Defaults to self.align_corners. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html dtype (Union[dtype, type, str, None, dtype]) – data type for resampling computation. Defaults to self.dtype. If None, use the data type of input data. To be compatible with other modules, the output data type is always float32. ValueError – When img spatially is not one of [2D, 3D]. Tensor Inverse of __call__. NotImplementedError – When the subclass does not override this method. Tensor Zooms an ND image using torch.nn.functional.interpolate. For details, please see https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html. Different from monai.transforms.resize, this transform takes scaling factors as input, and provides an option of preserving the input spatial size. zoom (Union[Sequence[float], float]) – The zoom factor along the spatial axes. If a float, zoom is the same for each spatial axis. If a sequence, zoom should contain one value for each spatial axis. mode (str) – {"nearest", "nearest-exact", "linear", "bilinear", "bicubic", "trilinear", "area"} The interpolation mode. Defaults to "area". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html padding_mode (str) – available modes for numpy array:{"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "edge". The mode to pad data after zooming. See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html align_corners (Optional[bool]) – This only has an effect when mode is 'linear', 'bilinear', 'bicubic' or 'trilinear'. Default: None. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html keep_size (bool) – Should keep original size (padding/slicing if needed), default is True. kwargs – other arguments for the np.pad or torch.pad function. note that np.pad treats channel dimension as the first dimension. img (Tensor) – channel first array, must have shape: (num_channels, H[, W, …,]). mode (Optional[str]) – {"nearest", "nearest-exact", "linear", "bilinear", "bicubic", "trilinear", "area"} The interpolation mode. Defaults to self.mode. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html padding_mode (Optional[str]) – available modes for numpy array:{"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "edge". The mode to pad data after zooming. See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html align_corners (Optional[bool]) – This only has an effect when mode is 'linear', 'bilinear', 'bicubic' or 'trilinear'. Defaults to self.align_corners. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html Tensor Inverse of __call__. NotImplementedError – When the subclass does not override this method. Tensor Extract all the patches sweeping the entire image in a row-major sliding-window manner with possible overlaps. It can sort the patches and return all or a subset of them. patch_size (Sequence[int]) – size of patches to generate slices for, 0 or None selects whole dimension offset (Optional[Sequence[int]]) – offset of starting position in the array, default is 0 for each dimension. num_patches (Optional[int]) – number of patches to return. Defaults to None, which returns all the available patches. If the required patches are more than the available patches, padding will be applied. overlap (Union[Sequence[float], float]) – the amount of overlap of neighboring patches in each dimension (a value between 0.0 and 1.0). If only one float number is given, it will be applied to all dimensions. Defaults to 0.0. sort_fn (Optional[str]) – when num_patches is provided, it determines if keep patches with highest values ("max"), lowest values ("min"), or in their default order (None). Default to None. threshold (Optional[float]) – a value to keep only the patches whose sum of intensities are less than the threshold. Defaults to no filtering. pad_mode (str) – refer to NumpyPadMode and PytorchPadMode. If None, no padding will be applied. Defaults to "constant". pad_kwargs – other arguments for the np.pad or torch.pad function. A MetaTensor consisting of a batch of all the patches with associated metadata MetaTensor data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Sort the patches based on the sum of their intensity, and just keep self.num_patches of them. :type image_np: ndarray :param image_np: a numpy.ndarray representing a stack of patches :type locations: ndarray :param locations: a numpy.ndarray representing the stack of location of each patch Filter the patches and their locations according to a threshold :type image_np: ndarray :param image_np: a numpy.ndarray representing a stack of patches :type locations: ndarray :param locations: a numpy.ndarray representing the stack of location of each patch Extract all the patches sweeping the entire image in a row-major sliding-window manner with possible overlaps, and with random offset for the minimal corner of the image, (0,0) for 2D and (0,0,0) for 3D. It can sort the patches and return all or a subset of them. patch_size (Sequence[int]) – size of patches to generate slices for, 0 or None selects whole dimension min_offset (Union[Sequence[int], int, None]) – the minimum range of offset to be selected randomly. Defaults to 0. max_offset (Union[Sequence[int], int, None]) – the maximum range of offset to be selected randomly. Defaults to image size modulo patch size. num_patches (Optional[int]) – number of patches to return. Defaults to None, which returns all the available patches. overlap (Union[Sequence[float], float]) – the amount of overlap of neighboring patches in each dimension (a value between 0.0 and 1.0). If only one float number is given, it will be applied to all dimensions. Defaults to 0.0. sort_fn (Optional[str]) – when num_patches is provided, it determines if keep patches with highest values (“max”), lowest values (“min”), or in their default order (None). Default to None. threshold (Optional[float]) – a value to keep only the patches whose sum of intensities are less than the threshold. Defaults to no filtering. pad_mode (str) – refer to NumpyPadMode and PytorchPadMode. If None, no padding will be applied. Defaults to "constant". pad_kwargs – other arguments for the np.pad or torch.pad function. A MetaTensor consisting of a batch of all the patches with associated metadata MetaTensor data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. Split the image into patches based on the provided grid in 2D. grid (Tuple[int, int]) – a tuple define the shape of the grid upon which the image is split. Defaults to (2, 2) size (Union[Tuple[int, int], int, None]) – a tuple or an integer that defines the output patch sizes. If it’s an integer, the value will be repeated for each dimension. The default is None, where the patch size will be inferred from the grid shape. Example Given an image (torch.Tensor or numpy.ndarray) with size of (3, 10, 10) and a grid of (2, 2), it will return a Tensor or array with the size of (4, 3, 5, 5). Here, if the size is provided, the returned shape will be (4, 3, size, size) Note: This transform currently support only image with two spatial dimensions. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a

Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. List[Union[ndarray, Tensor]]

Smooth Field#

Randomly adjust the contrast of input images by calculating a randomized smooth field for each invocation. This uses SmoothField internally to define the adjustment over the image. If pad is greater than 0 the edges of the input volume of that width will be mostly unchanged. Contrast is changed by raising input values by the power of the smooth field so the range of values given by gamma should be chosen with this in mind. For example, a minimum value of 0 in gamma will produce white areas so this should be avoided. After the contrast is adjusted the values of the result are rescaled to the range of the original input. spatial_size (Sequence[int]) – size of input array's spatial dimensions rand_size (Sequence[int]) – size of the randomized field to start from pad (int) – number of pixels/voxels along the edges of the field to pad with 1 mode (str) – interpolation mode to use when upsampling align_corners (Optional[bool]) – if True align the corners when upsampling field prob (float) – probability transform is applied gamma (Union[Sequence[float], float]) – (min, max) range for exponential field device (Optional[device]) – Pytorch device to define field on Apply the transform to img, if randomize randomizing the smooth field otherwise reusing the previous. Union[ndarray, Tensor] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandSmoothFieldAdjustContrast a Randomizable instance. Randomly adjust the intensity of input images by calculating a randomized smooth field for each invocation. This uses SmoothField internally to define the adjustment over the image. If pad is greater than 0 the edges of the input volume of that width will be mostly unchanged. Intensity is changed by multiplying the inputs by the smooth field, so the values of gamma should be chosen with this in mind. The default values of (0.1, 1.0) are sensible in that values will not be zeroed out by the field nor multiplied greater than the original value range. spatial_size (Sequence[int]) – size of input array rand_size (Sequence[int]) – size of the randomized field to start from pad (int) – number of pixels/voxels along the edges of the field to pad with 1 mode (str) – interpolation mode to use when upsampling align_corners (Optional[bool]) – if True align the corners when upsampling field prob (float) – probability transform is applied gamma (Union[Sequence[float], float]) – (min, max) range of intensity multipliers device (Optional[device]) – Pytorch device to define field on Apply the transform to img, if randomize randomizing the smooth field otherwise reusing the previous. Union[ndarray, Tensor] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandSmoothFieldAdjustIntensity a Randomizable instance. Deform an image using a random smooth field and Pytorch's grid_sample. The amount of deformation is given by def_range in fractions of the size of the image. The size of each dimension of the input image is always defined as 2 regardless of actual image voxel dimensions, that is the coordinates in every dimension range from -1 to 1. A value of 0.1 means pixels/voxels can be moved by up to 5% of the image's size. spatial_size (Sequence[int]) – input array size to which deformation grid is interpolated rand_size (Sequence[int]) – size of the randomized field to start from pad (int) – number of pixels/voxels along the edges of the field to pad with 0 field_mode (str) – interpolation mode to use when upsampling the deformation field align_corners (Optional[bool]) – if True align the corners when upsampling field prob (float) – probability transform is applied def_range (Union[Sequence[float], float]) – value of the deformation range in image size fractions, single min/max value or min/max pair grid_dtype – type for the deformation grid calculated from the field grid_mode (str) – interpolation mode used for sampling input using deformation grid grid_padding_mode (str) – padding mode used for sampling input using deformation grid grid_align_corners (Optional[bool]) – if True align the corners when sampling the deformation grid device (Optional[device]) – Pytorch device to define field on data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Union[ndarray, Tensor] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. Randomizable a Randomizable instance.

MRI Transforms#

A basic class for under-sampling mask setup. It provides common features for under-sampling mask generators. For example, RandomMaskFunc and EquispacedMaskFunc (two mask transform objects defined right after this module) both inherit MaskFunc to properly setup properties like the acceleration factor. This is an extra instance to allow for defining new mask generators. For creating other mask transforms, define a new class and simply override __call__. See an example of this in monai.apps.reconstruction.transforms.array.RandomKspacemask. kspace (Union[ndarray, Tensor]) – The input k-space data. The shape is (…,num_coils,H,W,2) for complex 2D inputs and (…,num_coils,H,W,D) for real 3D data. center_fractions (Sequence[float]) – Fraction of low-frequency columns to be retained. If multiple values are provided, then one of these numbers is chosen uniformly each time. accelerations (Sequence[float]) – Amount of under-sampling. This should have the same length as center_fractions. If multiple values are provided, then one of these is chosen uniformly each time. spatial_dims (int) – Number of spatial dims (e.g., it's 2 for a 2D data; it's also 2 for pseudo-3D datasets like the fastMRI dataset). The last spatial dim is selected for sampling. For the fastMRI dataset, k-space has the form (…,num_slices,num_coils,H,W) and sampling is done along W. For a general 3D data with the shape (…,num_coils,H,W,D), sampling is done along D. is_complex (bool) – if True, then the last dimension will be reserved for real/imaginary parts. If multiple values are provided for center_fractions and accelerations, this function selects one value uniformly for each training/test sample. Sequence[float] A tuple containing(1) center_fraction: chosen fraction of center kspace lines to exclude from under-sampling (2) acceleration: chosen acceleration factor (1) center_fraction: chosen fraction of center kspace lines to exclude from under-sampling (2) acceleration: chosen acceleration factor This k-space mask transform under-samples the k-space according to a random sampling pattern. Precisely, it uniformly selects a subset of columns from the input k-space data. If the k-space data has N columns, the mask picks out: 1. N_low_freqs = (N * center_fraction) columns in the center corresponding to low-frequencies 2. The other columns are selected uniformly at random with a probability equal to: prob = (N / acceleration - N_low_freqs) / (N - N_low_freqs). This ensures that the expected number of columns selected is equal to (N / acceleration) It is possible to use multiple center_fractions and accelerations, in which case one possible (center_fraction, acceleration) is chosen uniformly at random each time the transform is called. Example If accelerations = [4, 8] and center_fractions = [0.08, 0.04], then there is a 50% probability that 4-fold acceleration with 8% center fraction is selected and a 50% probability that 8-fold acceleration with 4% center fraction is selected. facebookresearch/fastMRI kspace (Union[ndarray, Tensor]) – The input k-space data. The shape is (…,num_coils,H,W,2) for complex 2D inputs and (…,num_coils,H,W,D) for real 3D data. The last spatial dim is selected for sampling. For the fastMRI dataset, k-space has the form (…,num_slices,num_coils,H,W) and sampling is done along W. For a general 3D data with the shape (…,num_coils,H,W,D), sampling is done along D. Sequence[Tensor] A tuple containing the under-sampled kspace absolute value of the inverse fourier of the under-sampled kspace the under-sampled kspace absolute value of the inverse fourier of the under-sampled kspace This k-space mask transform under-samples the k-space according to an equi-distant sampling pattern. Precisely, it selects an equi-distant subset of columns from the input k-space data. If the k-space data has N columns, the mask picks out: 1. N_low_freqs = (N * center_fraction) columns in the center corresponding to low-frequencies 2. The other columns are selected with equal spacing at a proportion that reaches the desired acceleration rate taking into consideration the number of low frequencies. This ensures that the expected number of columns selected is equal to (N / acceleration) It is possible to use multiple center_fractions and accelerations, in which case one possible (center_fraction, acceleration) is chosen uniformly at random each time the EquispacedMaskFunc object is called. Example If accelerations = [4, 8] and center_fractions = [0.08, 0.04], then there is a 50% probability that 4-fold acceleration with 8% center fraction is selected and a 50% probability that 8-fold acceleration with 4% center fraction is selected. facebookresearch/fastMRI kspace (Union[ndarray, Tensor]) – The input k-space data. The shape is (…,num_coils,H,W,2) for complex 2D inputs and (…,num_coils,H,W,D) for real 3D data. The last spatial dim is selected for sampling. For the fastMRI multi-coil dataset, k-space has the form (…,num_slices,num_coils,H,W) and sampling is done along W. For a general 3D data with the shape (…,num_coils,H,W,D), sampling is done along D. Sequence[Tensor] A tuple containing the under-sampled kspace absolute value of the inverse fourier of the under-sampled kspace the under-sampled kspace absolute value of the inverse fourier of the under-sampled kspace

Utility#

Do nothing to the data. As the output value is same as input, it can be used as a testing tool to verify the transform chain, Compose or transform adaptor, etc. Apply the transform to img. Union[ndarray, Tensor] Change the channel dimension of the image to the first dimension. Most of the image transformations in monai.transforms assume the input image is in the channel-first format, which has the shape (num_channels, spatial_dim_1[, spatial_dim_2, …]). This transform could be used to convert, for example, a channel-last image array in shape (spatial_dim_1[, spatial_dim_2, …], num_channels) into the channel-first format, so that the multidimensional image array can be correctly interpreted by the other transforms. channel_dim (int) – which dimension of input image is the channel, default is the last dimension. Apply the transform to img. Union[ndarray, Tensor] Change the channel dimension of the image to the last dimension. Some of other 3rd party transforms assume the input image is in the channel-last format with shape (spatial_dim_1[, spatial_dim_2, …], num_channels). This transform could be used to convert, for example, a channel-first image array in shape (num_channels, spatial_dim_1[, spatial_dim_2, …]) into the channel-last format, so that MONAI transforms can construct a chain with other 3rd party transforms together. channel_dim (int) – which dimension of input image is the channel, default is the first dimension. Apply the transform to img. Union[ndarray, Tensor] Adds a 1-length channel dimension to the input image. Most of the image transformations in monai.transforms assumes the input image is in the channel-first format, which has the shape (num_channels, spatial_dim_1[, spatial_dim_2, …]). This transform could be used, for example, to convert a (spatial_dim_1[, spatial_dim_2, …]) spatial image into the channel-first format so that the multidimensional image array can be correctly interpreted by the other transforms. Apply the transform to img. Union[ndarray, Tensor] Adjust or add the channel dimension of input data to ensure channel_first shape. This extracts the original_channel_dim info from provided meta_data dictionary or MetaTensor input. This value should state which dimension is the channel dimension so that it can be moved forward, or contain “no_channel” to state no dimension is the channel and so a 1-size first dimension is to be added. strict_check (bool) – whether to raise an error when the meta information is insufficient. channel_dim (Union[str, int, None]) – This argument can be used to specify the original channel dimension (integer) of the input array. It overrides the original_channel_dim from provided MetaTensor input. If the input array doesn’t have a channel dim, this value should be 'no_channel'. If this is set to None, this class relies on img or meta_dict to provide the channel dimension. Apply the transform to img. Tensor Repeat channel data to construct expected input shape for models. The repeats count includes the origin data, for example: RepeatChannel(repeats=2)([[1, 2], [3, 4]]) generates: [[1, 2], [1, 2], [3, 4], [3, 4]] repeats (int) – the number of repetitions for each element. Apply the transform to img, assuming img is a “channel-first” array. Union[ndarray, Tensor] Given an image of size X along a certain dimension, return a list of length X containing images. Useful for converting 3D images into a stack of 2D images, splitting multichannel inputs into single channels, for example. Note: torch.split/np.split is used, so the outputs are views of the input (shallow copy). dim (int) – dimension on which to split keepdim (bool) – if True, output will have singleton in the split dimension. If False, this dimension will be squeezed. update_meta – whether to update the MetaObj in each split result. Apply the transform to img. List[Tensor] Split Numpy array or PyTorch Tensor data according to the channel dim. It can help applying different following transforms to different channels. Note: torch.split/np.split is used, so the outputs are views of the input (shallow copy). channel_dim (int) – which dimension of input image is the channel, default to 0. Cast the Numpy data to specified numpy data type, or cast the PyTorch Tensor to specified PyTorch data type. Apply the transform to img, assuming img is a numpy array or PyTorch Tensor. dtype (Union[dtype, type, str, None, dtype]) – convert image to this data type, default is self.dtype. TypeError – When img type is not in Union[numpy.ndarray, torch.Tensor]. Union[ndarray, Tensor] dtype – convert image to this data type, default is np.float32. Converts the input image to a tensor without applying any other transformations. Input data can be PyTorch Tensor, numpy array, list, dictionary, int, float, bool, str, etc. Will convert Tensor, Numpy array, float, int, bool to Tensor, strings and objects keep the original. For dictionary, list or tuple, convert every item to a Tensor if applicable and wrap_sequence=False. dtype (Optional[dtype]) – target data type to when converting to Tensor. device (Optional[device]) – target device to put the converted Tensor data. wrap_sequence (bool) – if False, then lists will recursively call this function, default to True. E.g., if False, [1, 2] -> [tensor(1), tensor(2)], if True, then [1, 2] -> tensor([1, 2]). track_meta (Optional[bool]) – whether to convert to MetaTensor or regular tensor, default to None, use the return value of get_track_meta. Apply the transform to img and make it contiguous. Converts the input data to numpy array, can support list or tuple of numbers and PyTorch Tensor. dtype (Union[dtype, type, str, None]) – target data type when converting to numpy array. wrap_sequence (bool) – if False, then lists will recursively call this function, default to True. E.g., if False, [1, 2] -> [array(1), array(2)], if True, then [1, 2] -> array([1, 2]). Apply the transform to img and make it contiguous. Converts the input data to CuPy array, can support list or tuple of numbers, NumPy and PyTorch Tensor. dtype (Optional[dtype]) – data type specifier. It is inferred from the input by default. if not None, must be an argument of numpy.dtype, for more details: https://docs.cupy.dev/en/stable/reference/generated/cupy.array.html. wrap_sequence (bool) – if False, then lists will recursively call this function, default to True. E.g., if False, [1, 2] -> [array(1), array(2)], if True, then [1, 2] -> array([1, 2]). Create a CuPy array from data and make it contiguous Transposes the input image based on the given indices dimension ordering. Apply the transform to img. Union[ndarray, Tensor] Squeeze a unitary dimension. img (Union[ndarray, Tensor]) – numpy arrays with required dimension dim removed Union[ndarray, Tensor] dim (Optional[int]) – dimension to be squeezed. Default = 0 “None” works when the input is numpy array. update_meta – whether to update the meta info if the input is a metatensor. Default is True. TypeError – When dim is not an Optional[int]. Utility transform to show the statistics of data for debug or analysis. It can be inserted into any place of a transform chain and check results of previous transforms. It support both numpy.ndarray and torch.tensor as input data, so it can be used in pre-processing and post-processing. It gets logger from logging.getLogger(name), we can setup a logger outside first with the same name. If the log level of logging.RootLogger is higher than INFO, will add a separate StreamHandler log handler with INFO level and record to stdout. Apply the transform to img, optionally take arguments similar to the class constructor. Union[ndarray, Tensor] prefix (str) – will be printed in format: “{prefix} statistics”. data_type (bool) – whether to show the type of input data. data_shape (bool) – whether to show the shape of input data.

value_range (bool) – whether to show the value range of input data. data_value (bool) – whether to show the raw value of input data. a typical example is to print some properties of Nifti image: affine, pixdim, etc. additional_info (Optional[Callable]) – user can define callable function to extract additional info from input data. name (str) – identifier of logging.logger to use, defaulting to "DataStats". TypeError – When additional_info is not an Optional[Callable]. This is a pass through transform to be used for testing purposes. It allows adding fake behaviors that are useful for testing purposes to simulate how large datasets behave without needing to test on large data sets. For example, simulating slow NFS data transfers, or slow network transfers in testing by adding explicit timing delays. Testing of small test data can lead to incomplete understanding of real world issues, and may lead to sub-optimal design choices. img (Union[ndarray, Tensor]) – data remain unchanged throughout this transform. delay_time (Optional[float]) – The minimum amount of time, in fractions of seconds, to accomplish this delay task. Union[ndarray, Tensor] delay_time (float) – The minimum amount of time, in fractions of seconds, to accomplish this delay task. Apply a user-defined lambda as a transform. For example: func (Optional[Callable]) – Lambda/function to be applied. inv_func (Callable) – Lambda/function of inverse operation, default to lambda x: x. TypeError – When func is not an Optional[Callable]. Apply self.func to img. func (Optional[Callable]) – Lambda/function to be applied. Defaults to self.func. TypeError – When func is not an Optional[Callable]. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Randomizable version monai.transforms.Lambda, the input func may contain random logic, or randomly execute the function based on prob. func (Optional[Callable]) – Lambda/function to be applied. prob (float) – probability of executing the random function, default to 1.0, with 100% probability to execute. inv_func (Callable) – Lambda/function of inverse operation, default to lambda x: x. For more details, please check monai.transforms.Lambda. Apply self.func to img. func (Optional[Callable]) – Lambda/function to be applied. Defaults to self.func. TypeError – When func is not an Optional[Callable]. Inverse of __call__. NotImplementedError – When the subclass does not override this method. RemoveRepeatedChannel data to undo RepeatChannel The repeats count specifies the deletion of the origin data, for example: RemoveRepeatedChannel(repeats=2)([[1, 2], [1, 2], [3, 4], [3, 4]]) generates: [[1, 2], [3, 4]] repeats (int) – the number of repetitions to be deleted for each element. Apply the transform to img, assuming img is a "channel-first" array. Union[ndarray, Tensor] Convert labels to mask for other tasks. A typical usage is to convert segmentation labels to mask data to pre-process images and then feed the images into classification network. It can support single channel labels or One-Hot labels with specified select_labels. For example, users can select label value = [2, 3] to construct mask data, or select the second and the third channels of labels to construct mask data. The output mask data can be a multiple channels binary data or a single channel binary data that merges all the channels. select_labels (Union[Sequence[int], int]) – labels to generate mask from. for 1 channel label, the select_labels is the expected label values, like: [1, 2, 3]. for One-Hot format label, the select_labels is the expected channel indices. merge_channels (bool) – whether to use np.any() to merge the result on channel dim. if yes, will return a single channel mask with binary data. select_labels (Union[Sequence[int], int, None]) – labels to generate mask from. for 1 channel label, the select_labels is the expected label values, like: [1, 2, 3]. for One-Hot format label, the select_labels is the expected channel indices. merge_channels (bool) – whether to use np.any() to merge the result on channel dim. if yes, will return a single channel mask with binary data. Union[ndarray, Tensor] Compute foreground and background of the input label data, return the indices. If no output_shape specified, output data will be 1 dim indices after flattening. This transform can help pre-compute foreground and background regions for other transforms. A typical usage is to randomly select foreground and background to crop. The main logic is based on monai.transforms.utils.map_binary_to_indices. image_threshold (float) – if enabled image at runtime, use image > image_threshold to determine the valid image content area and select background only in this area. output_shape (Optional[Sequence[int]]) – expected shape of output indices. if not None, unravel indices to specified shape. label (Union[ndarray, Tensor]) – input data to compute foreground and background indices. image (Union[ndarray, Tensor, None]) – if image is not None, use label = 0 & image > image_threshold to define background. so the output items will not map to all the voxels in the label. output_shape (Optional[Sequence[int]]) – expected shape of output indices. if None, use self.output_shape instead. Tuple[Union[ndarray, Tensor], Union[ndarray, Tensor]] label (Union[ndarray, Tensor]) – input data to compute the indices of every class. image (Union[ndarray, Tensor, None]) – if image is not None, use image > image_threshold to define valid region, and only select the indices within the valid region. output_shape (Optional[Sequence[int]]) – expected shape of output indices. if None, use self.output_shape instead. List[Union[ndarray, Tensor]] Compute indices of every class of the input label data, return a list of indices. If no output_shape specified, output data will be 1 dim indices after flattening. This transform can help pre-compute indices of the class regions for other transforms. A typical usage is to randomly select indices of classes to crop. The main logic is based on monai.transforms.utils.map_classes_to_indices. num_classes (Optional[int]) – number of classes for argmax label, not necessary for One-Hot label. image_threshold (float) – if enabled image at runtime, use image > image_threshold to determine the valid image content area and select only the indices of classes in this area. output_shape (Optional[Sequence[int]]) – expected shape of output indices. if not None, unravel indices to specified shape. Convert labels to multi channels based on brats18 classes: label 1 is the necrotic and non-enhancing tumor core label 2 is the peritumoral edema label 4 is the GD-enhancing tumor The possible classes are TC (Tumor core), WT (Whole tumor) and ET (Enhancing tumor). data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Union[ndarray, Tensor] Add extreme points of label to the image as a new channel. This transform generates extreme point from label and applies a gaussian filter. The pixel values in points image are rescaled to range [rescale_min, rescale_max] and added as a new channel to input image. The algorithm is described in Roth et al., Going to Extremes: Weakly Supervised Medical Image Segmentation https://arxiv.org/abs/2009.11988. This transform only supports single channel labels (1, spatial_dim1, [spatial_dim2, …]). The background index is ignored when calculating extreme points. background (int) – Class index of background label, defaults to 0. pert (float) – Random perturbation amount to add to the points, defaults to 0.0. ValueError – When no label image provided. ValueError – When label image is not single channel. img (Union[ndarray, Tensor]) – the image that we want to add new channel to. label (Union[ndarray, Tensor, None]) – label image to get extreme points from. Shape must be (1, spatial_dim1, [, spatial_dim2, …]). Doesn’t support one-hot labels. sigma (Union[Sequence[float], float, Sequence[Tensor], Tensor]) – if a list of values, must match the count of spatial dimensions of input data, and apply every value in the list to 1 spatial dimension. if only 1 value provided, use it for all spatial dimensions. rescale_min (float) – minimum value of output data. rescale_max (float) – maximum value of output data. Union[ndarray, Tensor] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None This is a wrapper transform for PyTorch TorchVision transform based on the specified transform name and args. As most of the TorchVision transforms only work for PIL image and PyTorch Tensor, this transform expects input data to be PyTorch Tensor, users can easily call ToTensor transform to convert a Numpy array to Tensor. img (Union[ndarray, Tensor]) – PyTorch Tensor data for the TorchVision transform. name (str) – The transform name in TorchVision package. args – parameters for the TorchVision transform. kwargs – parameters for the TorchVision transform. Utility to map label values to another set of values. For example, map [3, 2, 1] to [0, 1, 2], [1, 2, 3] -> [0.5, 1.5, 2.5], [“label3”, “label2”, “label1”] -> [0, 1, 2], [3.5, 2.5, 1.5] -> [“label0”, “label1”, “label2”], etc. The label data must be numpy array or array-like data and the output data will be numpy array. Call self as a function. orig_labels (Sequence) – original labels that map to others. target_labels (Sequence) – expected label values, 1: 1 map to the orig_labels. dtype (Union[dtype, type, str, None]) – convert the output data to dtype, default to float32. Ensure the input data to be a PyTorch Tensor or numpy array, support: numpy array, PyTorch Tensor, float, int, bool, string and object keep the original. If passing a dictionary, list or tuple, still return dictionary, list or tuple will recursively convert every item to the expected data type if wrap_sequence=False. data_type (str) – target data type to convert, should be “tensor” or “numpy”. dtype (Union[dtype, type, str, None, dtype]) – target data content type to convert, for example: np.float32, torch.float, etc. device (Optional[device]) – for Tensor data type, specify the target device. wrap_sequence (bool) – if False, then lists will recursively call this function, default to True. E.g., if False, [1, 2] -> [tensor(1), tensor(2)], if True, then [1, 2] -> tensor([1, 2]). track_meta (Optional[bool]) – if True convert to MetaTensor, otherwise to Pytorch Tensor, if None behave according to return value of py:func:monai.data.meta_obj.get_track_meta. data (Union[ndarray, Tensor]) – input data can be PyTorch Tensor, numpy array, list, dictionary, int, float, bool, str, etc. will ensure Tensor, Numpy array, float, int, bool as Tensors or numpy arrays, strings and objects keep the original. for dictionary, list or tuple, ensure every item as expected type if applicable and wrap_sequence=False. Compute statistics for the intensity values of input image and store into the metadata dictionary. For example: if ops=[lambda x: np.mean(x), “max”] and key_prefix=”orig”, may generate below stats: {“orig_custom_0”: 1.5, “orig_max”: 3.0}. ops (Sequence[Union[str, Callable]]) – expected operations to compute statistics for the intensity. if a string, will map to the predefined operations, supported: [“mean”, “median”, “max”, “min”, “std”] mapping to np.nanmean, np.nanmedian, np.nanmax, np.nanmin, np.nanstd. if a callable function, will execute the function on input image. key_prefix (str) – the prefix to combine with ops name to generate the key to store the results in the metadata dictionary. if some ops are callable functions, will use “{key_prefix}_custom_{index}” as the key, where index counts from 0. channel_wise (bool) – whether to compute statistics for every channel of input image separately. if True, return a list of values for every operation, default to False. Compute statistics for the intensity of input image. img (Union[ndarray, Tensor]) – input image to compute intensity stats. meta_data (Optional[Dict]) – metadata dictionary to store the statistics data, if None, will create an empty dictionary. mask (Optional[ndarray]) – if not None, mask the image to extract only the interested area to compute statistics. mask must have the same shape as input img. Tuple[Union[ndarray, Tensor], Dict] Move PyTorch Tensor to the specified device. It can help cache data into GPU and execute following logic on GPU directly. Note If moving data to GPU device in the multi-processing workers of DataLoader, may got below CUDA error: “RuntimeError: Cannot re-initialize CUDA in forked subprocess. To use CUDA with multiprocessing, you must use the ‘spawn’ start method.” So usually suggest to set num_workers=0 in the DataLoader or ThreadDataLoader. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. device (Union[device, str]) – target device to move the Tensor, for example: “cuda:1”. kwargs – other args for the PyTorch Tensor.to() API, for more details: https://pytorch.org/docs/stable/generated/torch.Tensor.to.html. Wrap a non-randomized cuCIM transform, defined based on the transform name and args. For randomized transforms use monai.transforms.RandCuCIM. name (str) – the transform name in CuCIM package args – parameters for the CuCIM transform kwargs – parameters for the CuCIM transform Note CuCIM transform only work with CuPy arrays, so this transform expects input data to be cupy.ndarray. Users can call ToCuPy transform to convert a numpy array or torch tensor to cupy array. data – a CuPy array (cupy.ndarray) for the cuCIM transform cupy.ndarray Wrap a randomized cuCIM transform, defined based on the transform name and args For deterministic non-randomized transforms use monai.transforms.CuCIM. name (str) – the transform name in CuCIM package. args – parameters for the CuCIM transform. kwargs – parameters for the CuCIM transform. Note CuCIM transform only work with CuPy arrays, so this transform expects input data to be cupy.ndarray. Users can call ToCuPy transform to convert a numpy array or torch tensor to cupy array. If the random factor of the underlying cuCIM transform is not derived from self.R, the results may not be deterministic. See Also: monai.transforms.Randomizable. Appends additional channels encoding coordinates of the input. Useful when e.g. training using patch-based sampling, to allow feeding of the patch’s location into the network. This can be seen as a input-only version of CoordConv: Liu, R. et al. An Intriguing Failing of Convolutional Neural Networks and the CoordConv Solution, NeurIPS 2018. spatial_dims (Sequence[int]) – the spatial dimensions that are to have their coordinates encoded in a channel and appended to the input image. E.g., (0, 1, 2) represents H, W, D dims and append three channels to the input image, encoding the coordinates of the input’s three spatial dimensions. Deprecated since version 0.8.0: spatial_channels is deprecated, use spatial_dims instead. img (Union[ndarray, Tensor]) – data to be transformed, assuming img is channel first. Union[ndarray, Tensor]

Crop and Pad (Dict)#

Dictionary-based wrapper of monai.transforms.Pad. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform padder (Pad) – pad transform for the input image. mode (Union[Sequence[str], str]) – available modes for numpy array:{"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "constant". See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html It also can be a sequence of string, each element corresponds to a key in keys. allow_missing_keys (bool) – don’t raise exception if key is missing. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, MetaTensor] Dictionary-based wrapper of monai.transforms.SpatialPad. Performs padding to the data, symmetric for all sides or all on one side for each dimension. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform spatial_size (Union[Sequence[int], int]) – the spatial size of output data after padding, if a dimension of the input data size is larger than the pad size, will not pad that dimension. If its components have non-positive values, the corresponding size of input image will be used. for example: if the spatial size of input data is [30, 30, 30] and spatial_size=[32, 25, -1], the spatial size of output data will be [32, 30, 30]. method (str) – {"symmetric", "end"} Pad image symmetrically on every side or only pad at the end sides. Defaults to "symmetric". mode (Union[Sequence[str], str]) – available modes for numpy array:{"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "constant". See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html It also can be a sequence of string, each element corresponds to a key in keys. allow_missing_keys (bool) – don’t raise exception if key is missing. kwargs – other arguments for the np.pad or torch.pad function. note that np.pad treats channel dimension as the first dimension. Pad the input data by adding specified borders to every dimension. Dictionary-based wrapper of monai.transforms.BorderPad. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform spatial_border (Union[Sequence[int], int]) – specified size for every spatial border. it can be 3 shapes: single int number, pad all the borders with the same size. length equals the length of image shape, pad every spatial dimension separately. for example, image shape(CHW) is [1, 4, 4], spatial_border is [2, 1], pad every border of H dim with 2, pad every border of W dim with 1, result shape is [1, 8, 6]. length equals 2 x (length of image shape), pad every border of every dimension separately. for example, image shape(CHW) is [1, 4, 4], spatial_border is [1, 2, 3, 4], pad top of H dim with 1, pad bottom of H dim with 2, pad left of W dim with 3, pad right of W dim with 4. the result shape is [1, 7, 11]. specified size for every spatial border. it can be 3 shapes: single int number, pad all the borders with the same size. length equals the length of image shape, pad every spatial dimension separately. for example, image shape(CHW) is [1, 4, 4], spatial_border is [2, 1], pad every border of H dim with 2, pad every border of W dim with 1, result shape is [1, 8, 6]. length equals 2 x (length of image shape), pad every border of every dimension separately. for example, image shape(CHW) is [1, 4, 4], spatial_border is [1, 2, 3, 4], pad top of H dim with 1, pad bottom of H dim with 2, pad left of W dim with 3, pad right of W dim with 4. the result shape is [1, 7, 11]. mode (Union[Sequence[str], str]) – available modes for numpy array:{"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "constant". See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html It also can be a sequence of string, each element corresponds to a key in keys. allow_missing_keys (bool) – don't raise exception if key is missing. kwargs – other arguments for the np.pad or torch.pad function. note that np.pad treats channel dimension as the first dimension. Pad the input data, so that the spatial sizes are divisible by k. Dictionary-based wrapper of monai.transforms.DivisiblePad. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform k (Union[Sequence[int], int]) – the target k for each spatial dimension. if k is negative or 0, the original size is preserved. if k is an int, the same k be applied to all the input spatial dimensions. mode (Union[Sequence[str], str]) – available modes for numpy array:{"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "constant". See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html It also can be a sequence of string, each element corresponds to a key in keys. method (str) – {"symmetric", "end"} Pad image symmetrically on every side or only pad at the end sides. Defaults to "symmetric". allow_missing_keys (bool) – don't raise exception if key is missing. kwargs – other arguments for the np.pad or torch.pad function. note that np.pad treats channel dimension as the first dimension. See also monai.transforms.SpatialPad Dictionary-based wrapper of abstract class monai.transforms.Crop. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform cropper (Crop) – crop transform for the input image. allow_missing_keys (bool) – don't raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] An updated dictionary version of data by applying the transform. Inverse of __call__. NotImplementedError – When the subclass does not override this method.

Dict[Hashable, MetaTensor] Base class for random crop transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform cropper (Crop) – random crop transform for the input image. allow_missing_keys (bool) – don't raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] An updated dictionary version of data by applying the transform. Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandCropd a Randomizable instance. Dictionary-based wrapper of monai.transforms.SpatialCrop. General purpose cropper to produce sub-volume region of interest (ROI). If a dimension of the expected ROI size is larger than the input image size, will not crop that dimension. So the cropped result may be smaller than the expected ROI, and the cropped results of several images may not have exactly the same shape. It can support to crop ND spatial (channel-first) data. a list of slices for each spatial dimension (allows for use of -ve indexing and None) a spatial center and size the start and end coordinates of the ROI keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform roi_center (Optional[Sequence[int]]) – voxel coordinates for center of the crop ROI. roi_size (Optional[Sequence[int]]) – size of the crop ROI, if a dimension of ROI size is larger than image size, will not crop that dimension of the image. roi_start (Optional[Sequence[int]]) – voxel coordinates for start of the crop ROI. roi_end (Optional[Sequence[int]]) – voxel coordinates for end of the crop ROI, if a coordinate is out of image, use the end coordinate of image. roi_slices (Optional[Sequence[slice]]) – list of slices for each of the spatial dimensions. allow_missing_keys (bool) – don't raise exception if key is missing. Dictionary-based wrapper of monai.transforms.CenterSpatialCrop. If a dimension of the expected ROI size is larger than the input image size, will not crop that dimension. So the cropped result may be smaller than the expected ROI, and the cropped results of several images may not have exactly the same shape. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.MapTransform roi_size (Union[Sequence[int], int]) – the size of the crop region e.g. [224,224,128] if a dimension of ROI size is larger than image size, will not crop that dimension of the image. If its components have non-positive values, the corresponding size of input image will be used. for example: if the spatial size of input data is [40, 40, 40] and roi_size=[32, 64, -1], the spatial size of output data will be [32, 40, 40]. allow_missing_keys (bool) – don't raise exception if key is missing.

Dictionary-based version monai.transforms.RandSpatialCrop. Crop image with random size or specific size ROI. It can crop at a random position as center or at the image center. And allows to set the minimum and maximum size to limit the randomly generated ROI. Suppose all the expected fields specified by keys have same shape. Note: even random_size=False, if a dimension of the expected ROI size is larger than the input image size, will not crop that dimension. So the cropped result may be smaller than the expected ROI, and the cropped results of several images may not have exactly the same shape. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.MapTransform roi_size (Union[Sequence[int], int]) – if random_size is True, it specifies the minimum crop region. if random_size is False, it specifies the expected ROI size to crop. e.g. [224, 224, 128] if a dimension of ROI size is larger than image size, will not crop that dimension of the image. If its components have non-positive values, the corresponding size of input image will be used. for example: if the spatial size of input data is [40, 40, 40] and roi_size=[32, 64, -1], the spatial size of output data will be [32, 40, 40]. max_roi_size (Union[Sequence[int], int, None]) – if random_size is True and roi_size specifies the min crop region size, max_roi_size can specify the max crop region size. if None, defaults to the input image size. if its components have non-positive values, the corresponding size of input image will be used. random_center (bool) – crop at random position as center or the image center. random_size (bool) – crop with random size or specific size ROI. if True, the actual size is sampled from: randint(roi_scale * image spatial size, max_roi_scale * image spatial size + 1). allow_missing_keys (bool) – don't raise exception if key is missing. Dictionary-based version monai.transforms.RandSpatialCropSamples. Crop image with random size or specific size ROI to generate a list of N samples. It can crop at a random position as center or at the image center. And allows to set the minimum size to limit the randomly generated ROI. Suppose all the expected fields specified by keys have same shape, and add patch_index to the corresponding metadata. It will return a list of dictionaries for all the cropped images. Note: even random_size=False, if a dimension of the expected ROI size is larger than the input image size, will not crop that dimension. So the cropped result may be smaller than the expected ROI, and the cropped results of several images may not have exactly the same shape. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.MapTransform roi_size (Union[Sequence[int], int]) – if random_size is True, it specifies the minimum crop region. if random_size is False, it specifies the expected ROI size to crop. e.g. [224, 224, 128] if a dimension of ROI size is larger than image size, will not crop that dimension of the image. If its components have non-positive values, the corresponding size of input image will be used. for example: if the spatial size of input data is [40, 40, 40] and roi_size=[32, 64, -1], the spatial size of output data will be [32, 40, 40]. num_samples (int) – number of samples (crop regions) to take in the returned list. max_roi_size (Union[Sequence[int], int, None]) – if random_size is True and roi_size specifies the min crop region size, max_roi_size can specify the max crop region size. if None, defaults to the input image size. if its components have non-positive values, the corresponding size of input image will be used. random_center (bool) – crop at random position as center or the image center. random_size (bool) – crop with random size or specific size ROI. The actual size is sampled from randint(roi_size, img_size). allow_missing_keys (bool) – don't raise exception if key is missing. ValueError – When num_samples is nonpositive. Call self as a function. List[Dict[Hashable, Tensor]] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data.

NotImplementedError – When the subclass does not override this method. None Dictionary-based version monai.transforms.CropForeground. Crop only the foreground object of the expected images. The typical usage is to help training and evaluation if the valid part is small in the whole medical image. The valid part can be determined by any field in the data with source_key, for example: - Select values > 0 in image field as the foreground and crop on all fields specified by keys. - Select label = 3 in label field as the foreground to crop on all fields specified by keys. - Select label > 0 in the third channel of a One-Hot label field as the foreground to crop all keys fields. Users can define arbitrary function to select expected foreground from the whole source image or specified channels. And it can also add margin to every dim of the bounding box of foreground object. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform source_key (str) – data source to generate the bounding box of foreground, can be image or label, etc. select_fn (Callable) – function to select expected foreground, default is to select values > 0. channel_indices (Union[Iterable[int], int, None]) – if defined, select foreground only on the specified channels of image. if None, select foreground on the whole image. margin (Union[Sequence[int], int]) – add margin value to spatial dims of the bounding box, if only 1 value provided, use it for all dims. allow_smaller (bool) – when computing box size with margin, whether allow the image size to be smaller than box size, default to True. if the margined size is larger than image size, will pad with specified mode. k_divisible (Union[Sequence[int], int]) – make each spatial dimension to be divisible by k, default to 1. if k_divisible is an int, the same k be applied to all the input spatial dimensions. mode (Union[Sequence[str], str]) – available modes for numpy array:{"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "constant". See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html it also can be a sequence of string, each element corresponds to a key in keys. start_coord_key (str) – key to record the start coordinate of spatial bounding box for foreground. end_coord_key (str) – key to record the end coordinate of spatial bounding box for foreground. allow_missing_keys (bool) – don’t raise exception if key is missing. pad_kwargs – other arguments for the np.pad or torch.pad function. note that np.pad treats channel dimension as the first dimension. Samples a list of num_samples image patches according to the provided weight_map. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform w_key (str) – key for the weight map. The corresponding value will be used as the sampling weights, it should be a single-channel array in size, for example, (1, spatial_dim_0, spatial_dim_1, …) spatial_size (Union[Sequence[int], int]) – the spatial size of the image patch e.g. [224, 224, 128]. If its components have non-positive values, the corresponding size of img will be used. num_samples (int) – number of samples (image patches) to take in the returned list. allow_missing_keys (bool) – don't raise exception if key is missing. See also monai.transforms.RandWeightedCrop Call self as a function. List[Dict[Hashable, Tensor]] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandWeightedCropd a Randomizable instance. Dictionary-based version monai.transforms.RandCropByPosNegLabel. Crop random fixed sized regions with the center being a foreground or background voxel based on the Pos Neg Ratio. Suppose all the expected fields specified by keys have same shape, and add patch_index to the corresponding metadata. And will return a list of dictionaries for all the cropped images. If a dimension of the expected spatial size is larger than the input image size, will not crop that dimension. So the cropped result may be smaller than the expected size, and the cropped results of several images may not have exactly the same shape. And if the crop ROI is partly out of the image, will automatically adjust the crop center to ensure the valid crop ROI. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform label_key (str) – name of key for label image, this will be used for finding foreground/background. spatial_size (Union[Sequence[int], int]) – the spatial size of the crop region e.g. [224, 224, 128]. if a dimension of ROI size is larger than image size, will not crop that dimension of the image. if its components have non-positive values, the corresponding size of data[label_key] will be used. for example: if the spatial size of input data is [40, 40, 40] and spatial_size=[32, 64, -1], the spatial size of output data will be [32, 40, 40]. pos (float) – used with neg together to calculate the ratio pos / (pos + neg) for the probability to pick a foreground voxel as a center rather than a background voxel. neg (float) – used with pos together to calculate the ratio pos / (pos + neg) for the probability to pick a foreground voxel as a center rather than a background voxel. num_samples (int) – number of samples (crop regions) to take in each list. image_key (Optional[str]) – if image_key is not None, use label == 0 & image > image_threshold to select the negative sample(background) center. so the crop center will only exist on valid image area. image_threshold (float) – if enabled image_key, use image > image_threshold to determine the valid image content area. fg_indices_key (Optional[str]) – if provided pre-computed foreground indices of label, will ignore above image_key and image_threshold, and randomly select crop centers based on them, need to provide fg_indices_key and bg_indices_key together, expect to be 1 dim array of spatial indices after flattening. a typical usage is to call FgBgToIndicesd transform first and cache the results. bg_indices_key (Optional[str]) – if provided pre-computed background indices of label, will ignore above image_key and image_threshold, and randomly select crop centers based on them, need to provide fg_indices_key and bg_indices_key together, expect to be 1 dim array of spatial indices after flattening. a typical usage is to call FgBgToIndicesd transform first and cache the results. allow_smaller (bool) – if False, an exception will be raised if the image is smaller than the requested ROI in any dimension. If True, any smaller dimensions will be set to match the cropped size (i.e., no cropping in that dimension). allow_missing_keys (bool) – don't raise exception if key is missing. ValueError – When pos or neg are negative. ValueError – When pos=0 and neg=0. Incompatible values. Call self as a function. List[Dict[Hashable, Tensor]] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandCropByPosNegLabeld a Randomizable instance. Dictionary-based version monai.transforms.RandCropByLabelClasses. Crop random fixed sized regions with the center being a class based on the specified ratios of every class. The label data can be One-Hot format array or Argmax data. And will return a list of arrays for all the cropped images. For example, crop two (3 x 3) arrays from (5 x 5) array with ratios=[1, 2, 3, 1]: If a dimension of the expected spatial size is larger than the input image size, will not crop that dimension. So the cropped result may be smaller than expected size, and the cropped results of several images may not have exactly same shape. And if the crop ROI is partly out of the image, will automatically adjust the crop center to ensure the valid crop ROI. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform label_key (str) – name of key for label image, this will be used for finding indices of every class. spatial_size (Union[Sequence[int], int]) – the spatial size of the crop region e.g. [224, 224, 128]. if a dimension of ROI size is larger than image size, will not crop that dimension of the image. if its components have non-positive values, the corresponding size of label will be used. for example: if the spatial size of input data is [40, 40, 40] and spatial_size=[32, 64, -1], the spatial size of output data will be [32, 40, 40]. ratios (Optional[List[Union[float, int]]]) – specified ratios of every class in the label to generate crop centers, including background class. if None, every class will have the same ratio to generate crop centers. num_classes (Optional[int]) – number of classes for argmax label, not necessary for One-Hot label. num_samples (int) – number of samples (crop regions) to take in each list. image_key (Optional[str]) – if image_key is not None, only return the indices of every class that are within the valid region of the image (image > image_threshold). image_threshold (float) – if enabled image_key, use image > image_threshold to determine the valid image content area and select class indices only in this area. indices_key (Optional[str]) – if provided pre-computed indices of every class, will ignore above image and image_threshold, and randomly select crop centers based on them, expect to be 1 dim array of spatial indices after flattening. a typical usage is to call ClassesToIndices transform first and cache the results for better performance. allow_smaller (bool) – if False, an exception will be raised if the image is smaller than the requested ROI in any dimension. If True, any smaller dimensions will remain unchanged. allow_missing_keys (bool) – don't raise exception if key is missing. Call self as a function. List[Dict[Hashable, Tensor]] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandCropByLabelClassesd a Randomizable instance. Dictionary-based wrapper of monai.transforms.ResizeWithPadOrCrop. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.MapTransform spatial_size (Union[Sequence[int], int]) – the spatial size of output data after padding or crop. If has non-positive values, the corresponding size of input image will be used (no padding). mode (Union[Sequence[str], str]) – available modes for numpy array:{"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "constant". See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html It also can be a sequence of string, each element corresponds to a key in keys. allow_missing_keys (bool) – don’t raise exception if key is missing. method (str) – {"symmetric", "end"} Pad image symmetrically on every side or only pad at the end sides. Defaults to "symmetric". pad_kwargs – other arguments for the np.pad or torch.pad function. note that np.pad treats channel dimension as the first dimension. Dictionary-based wrapper of monai.transforms.BoundingRect. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.MapTransform bbox_key_postfix (str) – the output bounding box coordinates will be written to the value of {key}_{bbox_key_postfix}. select_fn (Callable) – function to select expected foreground, default is to select values > 0. allow_missing_keys (bool) – don’t raise exception if key is missing. See also: monai.transforms.utils.generate_spatial_bounding_box. Dict[Hashable, Union[ndarray, Tensor]] Dictionary-based version monai.transforms.RandScaleCrop. Crop image with random size or specific size ROI. It can crop at a random position as center or at the image center. And allows to set the minimum and maximum scale of image size to limit the randomly generated ROI. Suppose all the expected fields specified by keys have same shape. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.MapTransform roi_scale (Union[Sequence[float], float]) – if random_size is True, it specifies the minimum crop size: roi_scale * image spatial size. if random_size is False, it specifies the expected scale of image size to crop. e.g. [0.3, 0.4, 0.5]. If its components have non-positive values, will use 1.0 instead, which means the input image size. max_roi_scale (Union[Sequence[float], float, None]) – if random_size is True and roi_scale specifies the min crop region size, max_roi_scale can specify the max crop region size: max_roi_scale * image spatial size. if None, defaults to the input image size. if its components have non-positive values, will use 1.0 instead, which means the input image size. random_center (bool) – crop at random position as center or the image center. random_size (bool) – crop with random size or specified size ROI by roi_scale * image spatial size. if True, the actual size is sampled from: randint(roi_scale * image spatial size, max_roi_scale * image spatial size + 1). allow_missing_keys (bool) – don’t raise exception if key is missing. Dictionary-based wrapper of monai.transforms.CenterScaleCrop. Note: as using the same scaled ROI to crop, all the input data specified by keys should have the same spatial shape. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.MapTransform roi_scale (Union[Sequence[float], float]) – specifies the expected scale of image size to crop. e.g. [0.3, 0.4, 0.5] or a number for all dims. If its components have non-positive values, will use 1.0 instead, which means the input image size. allow_missing_keys (bool) – don’t raise exception if key is missing.

Intensity (Dict)#

Dictionary-based version monai.transforms.RandGaussianNoise. Add Gaussian noise to image. This transform assumes all the expected fields have same shape, if want to add different noise for every field, please use this transform separately. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform prob (float) – Probability to add Gaussian noise. mean (float) – Mean or “centre” of the distribution. std (float) – Standard deviation (spread) of distribution. dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. allow_missing_keys (bool) – don’t raise exception if key is missing. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandGaussianNoised a Randomizable instance. Dictionary-based wrapper of monai.transforms.ShiftIntensity. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform offset (float) – offset value to shift the intensity of image. safe (bool) – if True, then do safe dtype convert when intensity overflow. default to False. E.g., [256, -12] -> [array(0), array(244)]. If True, then [256, -12] -> [array(255), array(0)]. factor_key (Optional[str]) – if not None, use it as the key to extract a value from the corresponding metadata dictionary of key at runtime, and multiply the offset to shift intensity. Usually, IntensityStatsd transform can pre-compute statistics of intensity values and store in the metadata. it also can be a sequence of strings, map to keys. meta_keys (Union[Collection[Hashable], Hashable, None]) – explicitly indicate the key of the corresponding metadata dictionary. used to extract the factor value is factor_key is not None. for example, for data with key image, the metadata by default is in image_meta_dict. the metadata is a dictionary object which contains: filename, original_shape, etc. it can be a sequence of string, map to the keys. if None, will try to construct meta_keys by key_{meta_key_postfix}. meta_key_postfix (str) – if meta_keys is None, use key_{postfix} to fetch the metadata according to the key data, default is meta_dict, the metadata is a dictionary object. used to extract the factor value is factor_key is not None. allow_missing_keys (bool) – don’t raise exception if key is missing. Dictionary-based version monai.transforms.RandShiftIntensity. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform offsets (Union[Tuple[float, float], float]) – offset range to randomly shift. if single number, offset value is picked from (-offsets, offsets). safe (bool) – if True, then do safe dtype convert when intensity overflow. default to False. E.g., [256, -12] -> [array(0), array(244)]. If True, then [256, -12] -> [array(255), array(0)]. factor_key (Optional[str]) – if not None, use it as the key to extract a value from the corresponding metadata dictionary of key at runtime, and multiply the random offset to shift intensity. Usually, IntensityStatsd transform can pre-compute statistics of intensity values and store in the metadata. it also can be a sequence of strings, map to keys. meta_keys (Union[Collection[Hashable], Hashable, None]) – explicitly indicate the key of the corresponding metadata dictionary. used to extract the factor value is factor_key is not None. for example, for data with key image, the metadata by default is in image_meta_dict. the metadata is a dictionary object which contains: filename, original_shape, etc. it can be a sequence of string, map to the keys. if None, will try to construct meta_keys by key_{meta_key_postfix}. meta_key_postfix (str) – if meta_keys is None, use key_{postfix} to fetch the metadata according to the key data, default is meta_dict, the metadata is a dictionary object. used to extract the factor value is factor_key is not None. prob (float) – probability of shift. (Default 0.1, with 10% probability it returns an array shifted intensity.) allow_missing_keys (bool) – don’t raise exception if key is missing. Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandShiftIntensityd a Randomizable instance. Dictionary-based wrapper of monai.transforms.StdShiftIntensity. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform factor (float) – factor shift by v = v + factor * std(v). nonzero (bool) – whether only count non-zero values. channel_wise (bool) – if True, calculate on each channel separately. Please ensure that the first dimension represents the channel of the image if True. dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. allow_missing_keys (bool) – don't raise exception if key is missing. Dictionary-based version monai.transforms.RandStdShiftIntensity. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform factors (Union[Tuple[float, float], float]) – if tuple, the randomly picked range is (min(factors), max(factors)). If single number, the range is (-factors, factors). prob (float) – probability of std shift. nonzero (bool) – whether only count non-zero values. channel_wise (bool) – if True, calculate on each channel separately. dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. allow_missing_keys (bool) – don't raise exception if key is missing. Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandStdShiftIntensityd a Randomizable instance. Dictionary-based version monai.transforms.RandBiasField. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform degree (int) – degree of freedom of the polynomials. The value should be no less than 1. Defaults to 3. coeff_range (Tuple[float, float]) – range of the random coefficients. Defaults to (0.0, 0.1). dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. prob (float) – probability to do random bias field. allow_missing_keys (bool) – don't raise exception if key is missing. Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandBiasFieldd a Randomizable instance. Dictionary-based wrapper of monai.transforms.ScaleIntensity. Scale the intensity of input image to the given value range (minv, maxv). If minv and maxv not provided, use factor to scale image by v = v * (1 + factor). data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform minv (Optional[float]) – minimum value of output data. maxv (Optional[float]) – maximum value of output data. factor (Optional[float]) – factor scale by v = v * (1 + factor). In order to use this parameter, please set both minv and maxv into None. channel_wise (bool) – if True, scale on each channel separately. Please ensure that the first dimension represents the channel of the image if True. dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. allow_missing_keys (bool) – don't raise exception if key is missing. Dictionary-based version monai.transforms.RandScaleIntensity. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform factors (Union[Tuple[float, float], float]) – factor range to randomly scale by v = v * (1 + factor). if single number, factor value is picked from (-factors, factors). prob (float) – probability of scale. (Default 0.1, with 10% probability it returns a scaled array.) dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. allow_missing_keys (bool) – don't raise exception if key is missing. Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandScaleIntensityd a Randomizable instance. Dictionary-based wrapper of monai.transforms.NormalizeIntensity. This transform can normalize only non-zero values or entire image, and can also calculate mean and std on each channel separately. keys (Union[Collection[Hashable],

Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.MapTransform subtrahend (Union[ndarray, Tensor, None]) – the amount to subtract by (usually the mean) divisor (Union[ndarray, Tensor, None]) – the amount to divide by (usually the standard deviation) nonzero (bool) – whether only normalize non-zero values. channel_wise (bool) – if True, calculate on each channel separately, otherwise, calculate on the entire image directly. default to False. dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. allow_missing_keys (bool) – don't raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Dictionary-based wrapper of monai.transforms.ThresholdIntensity. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.MapTransform threshold (float) – the threshold to filter intensity values. above (bool) – filter values above the threshold or below the threshold, default is True. cval (float) – value to fill the remaining parts of the image, default is 0. allow_missing_keys (bool) – don't raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Dictionary-based wrapper of monai.transforms.ScaleIntensityRange. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.MapTransform a_min (float) – intensity original range min. a_max (float) – intensity original range max. b_min (Optional[float]) – intensity target range min. b_max (Optional[float]) – intensity target range max. clip (bool) – whether to perform clip after scaling. dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. allow_missing_keys (bool) – don't raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method.

Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Dictionary-based version of GibbsNoise. The transform applies Gibbs noise to 2D/3D MRI images. Gibbs artifacts are one of the common type of type artifacts appearing in MRI scans. For general information on Gibbs artifacts, please refer to: https://pubs.rsna.org/doi/full/10.1148/rg.313105115 https://pubs.rsna.org/doi/full/10.1148/radiographics.22.4.g02jl14949 keys (Union[Collection[Hashable], Hashable]) – 'image', 'label', or ['image', 'label'] depending on which data you need to transform. alpha (float) – Parametrizes the intensity of the Gibbs noise filter applied. Takes values in the interval [0,1] with alpha = 0 acting as the identity mapping. allow_missing_keys (bool) – do not raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Dictionary-based version of RandGibbsNoise. Naturalistic image augmentation via Gibbs artifacts. The transform randomly applies Gibbs noise to 2D/3D MRI images. Gibbs artifacts are one of the common type of type artifacts appearing in MRI scans. The transform is applied to all the channels in the data. For general information on Gibbs artifacts, please refer to: https://pubs.rsna.org/doi/full/10.1148/rg.313105115 https://pubs.rsna.org/doi/full/10.1148/radiographics.22.4.g02jl14949 keys (Union[Collection[Hashable], Hashable]) – 'image', 'label', or ['image', 'label'] depending on which data you need to transform. prob (float) – probability of applying the transform. alpha (float, List[float]) – Parametrizes the intensity of the Gibbs noise filter applied. Takes values in the interval [0,1] with alpha = 0 acting as the identity mapping. If a length-2 list is given as [a,b] then the value of alpha will be sampled uniformly from the interval [a,b]. allow_missing_keys (bool) – do not raise exception if key is missing. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandGibbsNoised a Randomizable instance. Dictionary-based wrapper of monai.transforms.KSpaceSpikeNoise. Applies localized spikes in k-space at the given locations and intensities. Spike (Herringbone) artifact is a type of data acquisition artifact which may occur during MRI scans. For general information on spike artifacts, please refer to: AAPM/RSNA physics tutorial for residents: fundamental physics of MR imaging. Body MRI artifacts in clinical practice: A physicist’s and radiologist’s perspective. keys (Union[Collection[Hashable], Hashable]) – “image”, “label”, or [“image”, “label”] depending on which data you need to transform. loc (Union[Tuple, Sequence[Tuple]]) – spatial location for the spikes. For images with 3D spatial dimensions, the user can provide (C, X, Y, Z) to fix which channel C is affected, or (X, Y, Z) to place the same spike in all channels. For 2D cases, the user can provide (C, X, Y) or (X, Y). k_intensity (Union[Sequence[float], float, None]) – value for the log-intensity of the k-space version of the image. If one location is passed to loc or the channel is not specified, then this argument should receive a float. If loc is given a sequence of locations, then this argument should receive a sequence of intensities. This value should be tested as it is data-dependent. The default values are the 2.5 the mean of the log-intensity for each channel. allow_missing_keys (bool) – do not raise exception if key is missing. Example When working with 4D data, KSpaceSpikeNoised("image", loc = ((3,60,64,32), (64,60,32)), k_intensity = (13,14)) will place a spike at [3, 60, 64, 32] with log-intensity = 13, and one spike per channel located respectively at [: , 64, 60, 32] with log-intensity = 14. data (Mapping[Hashable, Union[ndarray, Tensor]]) – Expects image/label to have dimensions (C, H, W) or (C, H, W, D), where C is the channel. Dict[Hashable, Union[ndarray, Tensor]] Dictionary-based version of monai.transforms.RandKSpaceSpikeNoise. Naturalistic data augmentation via spike artifacts. The transform applies localized spikes in k-space. For general information on spike artifacts, please refer to: AAPM/RSNA physics tutorial for residents: fundamental physics of MR imaging. Body MRI artifacts in clinical practice: A physicist’s and radiologist’s perspective. keys (Union[Collection[Hashable], Hashable]) – “image”, “label”, or [“image”, “label”] depending on which data you need to transform. prob (float) – probability to add spike artifact to each item in the dictionary provided it is realized that the noise will be applied to the dictionary. intensity_range (Optional[Sequence[Union[Sequence[float], float]]]) – pass a tuple (a, b) to sample the log-intensity from the interval (a, b) uniformly for all channels. Or pass sequence of intervals ((a0, b0), (a1, b1), …) to sample for each respective channel. In the second case, the number of 2-tuples must match the number of channels. Default ranges is (0.95x, 1.10x) where x is the mean log-intensity for each channel. channel_wise (bool) – treat each channel independently. True by default. allow_missing_keys (bool) – do not raise exception if key is missing. Example To apply k-space spikes randomly on the image only, with probability 0.5, and log-intensity sampled from the interval [13, 15] for each channel independently, one uses RandKSpaceSpikeNoised("image", prob=0.5, intensity_ranges=(13, 15), channel_wise=True). data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandKSpaceSpikeNoised a Randomizable instance. Dictionary-based version monai.transforms.RandRicianNoise. Add Rician noise to image. This transform assumes all the expected fields have same shape, if want to add different noise for every field, please use this transform separately. keys (Union[Collection[Hashable], Hashable]) – Keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform prob (float) – Probability to add Rician noise to the dictionary. mean (Union[Sequence[float], float]) – Mean or "centre" of the Gaussian distributions sampled to make up the Rician noise. std (Union[Sequence[float], float]) – Standard deviation (spread) of the Gaussian distributions sampled to make up the Rician noise. channel_wise (bool) – If True, treats each channel of the image separately. relative (bool) – If True, the spread of the sampled Gaussian distributions will be std times the standard deviation of the image or channel's intensity histogram. sample_std (bool) – If True, sample the spread of the Gaussian distributions uniformly from 0 to std. dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. allow_missing_keys (bool) – Don't raise exception if key is missing. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandRicianNoised a Randomizable instance. Dictionary-based wrapper of monai.transforms.ScaleIntensityRangePercentiles. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.MapTransform lower (float) – lower percentile. upper (float) – upper percentile. b_min (Optional[float]) – intensity target range min. b_max (Optional[float]) – intensity target range max. clip (bool) – whether to perform clip after scaling. relative (bool) – whether to scale to the corresponding percentiles of [b_min, b_max] channel_wise (bool) – if True, compute intensity percentile and normalize every channel separately. default to False. dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. allow_missing_keys (bool) – don't raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method.

Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Dictionary-based wrapper of monai.transforms.AdjustContrast. Changes image intensity by gamma. Each pixel/voxel intensity is updated as: x = ((x - min) / intensity_range) ^ gamma * intensity_range + min keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.MapTransform gamma (float) – gamma value to adjust the contrast as function. allow_missing_keys (bool) – don't raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Dictionary-based version monai.transforms.RandAdjustContrast. Randomly changes image intensity by gamma. Each pixel/voxel intensity is updated as: x = ((x - min) / intensity_range) ^ gamma * intensity_range + min keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.MapTransform prob (float) – Probability of adjustment. gamma (Union[Tuple[float, float], float]) – Range of gamma values. If single number, value is picked from (0.5, gamma), default is (0.5, 4.5). allow_missing_keys (bool) – don't raise exception if key is missing. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandAdjustContrastd a Randomizable instance. Dictionary-based wrapper of monai.transforms.MaskIntensity. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform mask_data (Union[ndarray, Tensor, None]) – if mask data is single channel, apply to every channel of input image. if multiple channels, the channel number must match input data. the intensity values of input image corresponding to the selected values in the mask data will keep the original value, others will be set to 0. if None, will extract the mask data from input data based on mask_key. mask_key (Optional[str]) – the key to extract mask data from input dictionary, only works when mask_data is None. select_fn (Callable) – function to select valid values of the mask_data, default is to select values > 0. allow_missing_keys (bool) – don’t raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Dictionary-based wrapper of monai.transforms.SavitzkyGolaySmooth. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform window_length (int) – length of the filter window, must be a positive odd integer. order (int) – order of the polynomial to fit to each window, must be less than window_length. axis (int) – optional axis along which to apply the filter kernel. Default 1 (first spatial dimension). mode (str) – optional padding mode, passed to convolution class. 'zeros', 'reflect', 'replicate' or 'circular'. default: 'zeros'. See torch.nn.Conv1d() for more information. allow_missing_keys (bool) – don’t raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Dictionary-based wrapper of monai.transforms.MedianSmooth. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform radius (Union[Sequence[int], int]) – if a list of values, must match the count of spatial dimensions of input data, and apply every value in the list to 1 spatial dimension. if only 1 value provided, use it for all spatial dimensions. allow_missing_keys (bool) – don’t raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Dictionary-based wrapper of monai.transforms.GaussianSmooth. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform sigma (Union[Sequence[float], float]) – if a list of values, must match the count of spatial dimensions of input data, and apply every value in the list to 1 spatial dimension. if only 1 value provided, use it for all spatial dimensions. approx (str) – discrete Gaussian kernel type, available options are “erf”, “sampled”, and “scalespace”. see also monai.networks.layers.GaussianFilter(). allow_missing_keys (bool) – don’t raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Dictionary-based wrapper of monai.transforms.GaussianSmooth. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform sigma_x (Tuple[float, float]) – randomly select sigma value for the first spatial dimension. sigma_y (Tuple[float, float]) – randomly select sigma value for the second spatial dimension if have. sigma_z (Tuple[float, float]) – randomly select sigma value for the third spatial dimension if have. approx (str) – discrete Gaussian kernel type, available options are “erf”, “sampled”, and “scalespace”. see also monai.networks.layers.GaussianFilter(). prob (float) – probability of Gaussian smooth. allow_missing_keys (bool) – don’t raise exception if key is missing. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandGaussianSmoothd a Randomizable instance. Dictionary-based wrapper of monai.transforms.GaussianSharpen. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform sigma1 (Union[Sequence[float], float]) – sigma parameter for the first gaussian kernel. if a list of values, must match the count of spatial dimensions of input data, and apply every value in the list to 1 spatial dimension. if only 1 value provided, use it for all spatial dimensions. sigma2 (Union[Sequence[float], float]) – sigma parameter for the second gaussian kernel. if a list of values, must match the count of spatial dimensions of input data, and apply every value in the list to 1 spatial dimension. if only 1 value provided, use it for all spatial dimensions. alpha (float) – weight parameter to compute the final result. approx (str) – discrete Gaussian kernel type, available options are “erf”, “sampled”, and “scalespace”. see also monai.networks.layers.GaussianFilter().

allow_missing_keys (bool) – don’t raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Dictionary-based wrapper of monai.transforms.GaussianSharpen. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform sigma1_x (Tuple[float, float]) – randomly select sigma value for the first spatial dimension of first gaussian kernel. sigma1_y (Tuple[float, float]) – randomly select sigma value for the second spatial dimension(if have) of first gaussian kernel. sigma1_z (Tuple[float, float]) – randomly select sigma value for the third spatial dimension(if have) of first gaussian kernel. sigma2_x (Union[Tuple[float, float], float]) – randomly select sigma value for the first spatial dimension of second gaussian kernel. if only 1 value X provided, it must be smaller than sigma1_x and randomly select from [X, sigma1_x]. sigma2_y (Union[Tuple[float, float], float]) – randomly select sigma value for the second spatial dimension(if have) of second gaussian kernel. if only 1 value Y provided, it must be smaller than sigma1_y and randomly select from [Y, sigma1_y]. sigma2_z (Union[Tuple[float, float], float]) – randomly select sigma value for the third spatial dimension(if have) of second gaussian kernel. if only 1 value Z provided, it must be smaller than sigma1_z and randomly select from [Z, sigma1_z]. alpha (Tuple[float, float]) – randomly select weight parameter to compute the final result. approx (str) – discrete Gaussian kernel type, available options are “erf”, “sampled”, and “scalespace”. see also monai.networks.layers.GaussianFilter(). prob (float) – probability of Gaussian sharpen. allow_missing_keys (bool) – don’t raise exception if key is missing. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandGaussianSharpend a Randomizable instance. Dictionary-based version monai.transforms.RandHistogramShift. Apply random nonlinear transform the image’s intensity histogram. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.MapTransform num_control_points (Union[Tuple[int, int], int]) – number of control points governing the nonlinear intensity mapping. a smaller number of control points allows for larger intensity shifts. if two values provided, number of control points selecting from range (min_value, max_value). prob (float) – probability of histogram shift. allow_missing_keys (bool) – don't raise exception if key is missing. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandHistogramShiftd a Randomizable instance. Dictionary-based wrapper of monai.transforms.RandCoarseDropout. Expect all the data specified by keys have same spatial shape and will randomly dropout the same regions for every key, if want to dropout differently for every key, please use this transform separately. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform holes (int) – number of regions to dropout, if max_holes is not None, use this arg as the minimum number to randomly select the expected number of regions. spatial_size (Union[Sequence[int], int]) – spatial size of the regions to dropout, if max_spatial_size is not None, use this arg as the minimum spatial size to randomly select size for every region. if some components of the spatial_size are non-positive values, the transform will use the corresponding components of input img size. For example, spatial_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. dropout_holes (bool) – if True, dropout the regions of holes and fill value, if False, keep the holes and dropout the outside and fill value. default to True. fill_value (Union[Tuple[float, float], float, None]) – target value to fill the dropout regions, if providing a number, will use it as constant value to fill all the regions. if providing a tuple for the min and max, will randomly select value for every pixel / voxel from the range [min, max). if None, will compute the min and max value of input image then randomly select value to fill, default to None. max_holes (Optional[int]) – if not None, define the maximum number to randomly select the expected number of regions. max_spatial_size (Union[Sequence[int], int, None]) – if not None, define the maximum spatial size to randomly select size for every region. if some components of the max_spatial_size are non-positive values, the transform will use the corresponding components of input img size. For example, max_spatial_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. prob (float) – probability of applying the transform. allow_missing_keys (bool) – don't raise exception if key is missing. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example:

AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandCoarseDropoutd a Randomizable instance. Dictionary-based wrapper of monai.transforms.RandCoarseShuffle. Expect all the data specified by keys have same spatial shape and will randomly dropout the same regions for every key, if want to shuffle different regions for every key, please use this transform separately. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform holes (int) – number of regions to dropout, if max_holes is not None, use this arg as the minimum number to randomly select the expected number of regions. spatial_size (Union[Sequence[int], int]) – spatial size of the regions to dropout, if max_spatial_size is not None, use this arg as the minimum spatial size to randomly select size for every region. if some components of the spatial_size are non-positive values, the transform will use the corresponding components of input img size. For example, spatial_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. max_holes (Optional[int]) – if not None, define the maximum number to randomly select the expected number of regions. max_spatial_size (Union[Sequence[int], int, None]) – if not None, define the maximum spatial size to randomly select size for every region. if some components of the max_spatial_size are non-positive values, the transform will use the corresponding components of input img size. For example, max_spatial_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. prob (float) – probability of applying the transform. allow_missing_keys (bool) – don’t raise exception if key is missing. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandCoarseShuffled a Randomizable instance. Dictionary-based wrapper of monai.transforms.HistogramNormalize. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform num_bins (int) – number of the bins to use in histogram, default to 256. for more details: https://numpy.org/doc/stable/reference/generated/numpy.histogram.html. min (int) –

the min value to normalize input image, default to 255. max (int) – the max value to normalize input image, default to 255. mask (Union[ndarray, Tensor, None]) – if provided, must be ndarray of bools or 0s and 1s, and same shape as image. only points at which mask==True are used for the equalization. can also provide the mask by mask_key at runtime. mask_key (Optional[str]) – if mask is None, will try to get the mask with mask_key. dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. allow_missing_keys (bool) – do not raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Creates a binary mask that defines the foreground based on thresholds in RGB or HSV color space. This transform receives an RGB (or grayscale) image where by default it is assumed that the foreground has low values (dark) while the background is white. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. threshold (Union[Dict, Callable, str, float]) – an int or a float number that defines the threshold that values less than that are foreground. It also can be a callable that receives each dimension of the image and calculate the threshold, or a string that defines such callable from skimage.filter.threshold_…. For the list of available threshold functions, please refer to https://scikit-image.org/docs/stable/api/skimage.filters.html Moreover, a dictionary can be passed that defines such thresholds for each channel, like {“R”: 100, “G”: “otsu”, “B”: skimage.filter.threshold_mean} hsv_threshold (Union[Dict, Callable, str, float, int, None]) – similar to threshold but HSV color space (“H”, “S”, and “V”). Unlike RBG, in HSV, value greater than hsv_threshold are considered foreground. invert (bool) – invert the intensity range of the input image, so that the dtype maximum is now the dtype minimum, and vice-versa. new_key_prefix (Optional[str]) – this prefix be prepended to the key to create a new key for the output and keep the value of key intact. By default not prefix is set and the corresponding array to the key will be replaced. allow_missing_keys (bool) – do not raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Compute horizontal and vertical maps from an instance mask It generates normalized horizontal and vertical distances to the center of mass of each region. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. dtype (Union[dtype, type, str, None]) – the type of output Tensor. Defaults to “float32”. new_key_prefix (str) – this prefix be prepended to the key to create a new key for the output and keep the value of key intact. Defaults to "“_hover”, so if the input key is “mask” the output will be “hover_mask”. allow_missing_keys (bool) – do not raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform.

IO (Dict)#

Dictionary-based wrapper of monai.transforms.LoadImage, It can load both image data and metadata. When loading a list of files in one key, the arrays will be stacked and a new dimension will be added as the first dimension In this case, the metadata of the first image will be used to represent the stacked result. The affine transform of all the stacked images should be same. The output metadata field will be created as meta_keys or key_{meta_key_postfix}. If reader is not specified, this class automatically chooses readers based on the supported suffixes and in the following order: User-specified reader at runtime when calling this loader. User-specified reader in the constructor of LoadImage. Readers from the last to the first in the registered list. Current default readers: (nii, nii.gz -> NibabelReader), (png, jpg, bmp -> PILReader), (npz, npy -> NumpyReader), (dcm, DICOM series and others -> ITKReader). Please note that for png, jpg, bmp, and other 2D formats, readers often swap axis 0 and 1 after loading the array because the HW definition for non-medical specific file formats is different from other common medical packages. Note If reader is specified, the loader will attempt to use the specified readers and the default supported readers. This might introduce overheads when handling the exceptions of trying the incompatible loaders. In this case, it is therefore recommended setting the most appropriate reader as the last item of the reader parameter. See also tutorial: Project-MONAI/tutorials KeyError – When not self.overwriting and key already exists in data. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform reader (Union[Type[ImageReader], str, None]) – reader to load image file and metadata - if reader is None, a default set of SUPPORTED_READERS will be used. - if reader is a string, it’s treated as a class name or dotted path (such as "monai.data.ITKReader"), the supported built-in reader classes are "ITKReader", "NibabelReader", "NumpyReader". a reader instance will be constructed with the *args and **kwargs parameters. - if reader is a reader class/instance, it will be registered to this loader accordingly. dtype (Union[dtype, type, str, None]) – if not None, convert the loaded image data to this data type. meta_keys (Union[Collection[Hashable], Hashable, None]) – explicitly indicate the key to store the corresponding metadata dictionary. the metadata is a dictionary object which contains: filename, original_shape, etc. it can be a sequence of string, map to the keys. if None, will try to construct meta_keys by key_{meta_key_postfix}. meta_key_postfix (str) – if meta_keys is None, use key_{postfix} to store the metadata of the nifti image, default is meta_dict. The metadata is a dictionary object. For example, load nifti file for image, store the metadata into image_meta_dict. overwriting (bool) – whether allow overwriting existing metadata of same key. default is False, which will raise exception if encountering existing key. image_only (bool) – if True return dictionary containing just only the image volumes, otherwise return dictionary containing image data array and header dict per input key. ensure_channel_first (bool) – if True and loaded both image array and metadata, automatically convert the image array shape to channel first. default to False. simple_keys (bool) – whether to remove redundant metadata keys, default to False for backward compatibility. prune_meta_pattern (Optional[str]) – combined with prune_meta_sep, a regular expression used to match and prune keys in the metadata (nested dictionary), default to None, no key deletion. prune_meta_sep (str) – combined with prune_meta_pattern, used to match and prune keys in the metadata (nested dictionary). default is ".", see also monai.transforms.DeleteItemsd. e.g. prune_meta_pattern=".*_code$", prune_meta_sep=" " removes meta keys that ends with "_code". allow_missing_keys (bool) – don't raise exception if key is missing. args – additional parameters for reader if providing a reader name. kwargs – additional parameters for reader if providing a reader name. Register a virtual subclass of an ABC. Returns the subclass, to allow usage as a class decorator. Dictionary-based wrapper of monai.transforms.SaveImage. Note Image should be channel-first shape: [C,H,W,[D]]. If the data is a patch of an image, the patch index will be appended to the filename. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform meta_keys (Union[Collection[Hashable], Hashable, None]) – explicitly indicate the key of the corresponding metadata dictionary. For example, for data with key image, the metadata by default is in image_meta_dict. The metadata is a dictionary contains values such as filename, original_shape. This argument can be a sequence of string, map to the keys. If None, will try to construct meta_keys by key_{meta_key_postfix}. meta_key_postfix (str) – if meta_keys is None, use key_{meta_key_postfix} to retrieve the metadict. output_dir (Union[Path, str]) – output image directory. output_postfix (str) – a string appended to all output file names, default to trans. output_ext (str) – output file extension name, available extensions: .nii.gz, .nii, .png. resample (bool) – whether to resample image (if needed) before saving the data array, based on the spatial_shape (and original_affine) from metadata. mode (str) – This option is used when resample=True. Defaults to "nearest". Depending on the writers, the possible options are: {"bilinear", "nearest", "bicubic"}. See also: https://pytorch.org/docs/stable/nn.functional.html#grid-sample {"nearest", "linear", "bilinear", "bicubic", "trilinear", "area"}. See also: https://pytorch.org/docs/stable/nn.functional.html#interpolate This option is used when resample=True. Defaults to "nearest". Depending on the writers, the possible options are: {"bilinear", "nearest", "bicubic"}. See also: https://pytorch.org/docs/stable/nn.functional.html#grid-sample {"nearest", "linear", "bilinear", "bicubic", "trilinear", "area"}. See also: https://pytorch.org/docs/stable/nn.functional.html#interpolate padding_mode (str) – This option is used when resample = True. Defaults to "border". Possible options are {"zeros", "border", "reflection"} See also: https://pytorch.org/docs/stable/nn.functional.html#grid-sample scale (Optional[int]) – {255, 65535} postprocess data by clipping to [0, 1] and scaling [0, 255] (uint8) or [0, 65535] (uint16). Default is None (no scaling). dtype (Union[dtype, type, str, None]) – data type during resampling computation. Defaults to np.float64 for best precision. if None, use the data type of input data. To set the output data type, use output_dtype. output_dtype (Union[dtype, type, str, None]) – data type for saving data. Defaults to np.float32. allow_missing_keys (bool) – don't raise exception if key is missing. squeeze_end_dims (bool) – if True, any trailing singleton dimensions will be removed (after the channel has been moved to the end). So if input is (C,H,W,D), this will be altered to (H,W,D,C), and then if C==1, it will be saved as (H,W,D). If D is also 1, it will be saved as (H,W). If false, image will always be saved as (H,W,D,C). data_root_dir (str) – if not empty, it specifies the beginning parts of the input file's absolute path. It's used to compute input_file_rel_path, the relative path to the file from data_root_dir to preserve folder structure when saving in case there are files in different folders with the same file names. For example, with the following inputs: input_file_name: /foo/bar/test1/image.nii output_postfix: seg output_ext: .nii.gz output_dir: /output data_root_dir: /foo/bar The output will be: /output/test1/image/image_seg.nii.gz if not empty, it specifies the beginning parts of the input file's absolute path. It's used to compute input_file_rel_path, the relative path to the file from data_root_dir to preserve folder structure when saving in case there are files in different folders with the same file names. For example, with the following inputs: input_file_name: /foo/bar/test1/image.nii output_postfix: seg output_ext: .nii.gz output_dir: /output data_root_dir: /foo/bar The output will be: /output/test1/image/image_seg.nii.gz separate_folder (bool) – whether to save every file in a separate folder. For example: for the input filename image.nii, postfix seg and folder_path output, if separate_folder=True, it will be saved as: output/image/image_seg.nii, if False, saving as output/image_seg.nii. Default to True. print_log (bool) – whether to print logs when saving. Default to True. output_format (str) – an optional string to specify the output image writer. see also: monai.data.image_writer.SUPPORTED_WRITERS. writer (Union[Type[ImageWriter], str, None]) – a customised monai.data.ImageWriter subclass to save data arrays. if None, use the default writer from monai.data.image_writer according to output_ext. if it's a string, it's treated as a class name or dotted path; the supported built-in writer classes are "NibabelWriter", "ITKWriter", "PILWriter". output_name_formatter – a callable function (returning a kwargs dict) to format the output file name. see also: monai.data.folder_layout.default_name_formatter(). data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. An updated dictionary version of data by applying the transform.

Post-processing (Dict)#

Dictionary-based wrapper of monai.transforms.AddActivations. Add activation layers to the input data specified by keys. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method.

Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to model output and label. See also: monai.transforms.compose.MapTransform sigmoid (Union[Sequence[bool], bool]) – whether to execute sigmoid function on model output before transform. it also can be a sequence of bool, each element corresponds to a key in keys. softmax (Union[Sequence[bool], bool]) – whether to execute softmax function on model output before transform. it also can be a sequence of bool, each element corresponds to a key in keys. other (Union[Sequence[Callable], Callable, None]) – callable function to execute other activation layers, for example: other = torch.tanh. it also can be a sequence of Callable, each element corresponds to a key in keys. allow_missing_keys (bool) – don't raise exception if key is missing. kwargs – additional parameters to torch.softmax (used when softmax=True). Defaults to dim=0, unrecognized parameters will be ignored. Dictionary-based wrapper of monai.transforms.AsDiscrete. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to model output and label. See also: monai.transforms.compose.MapTransform argmax (Union[Sequence[bool], bool]) – whether to execute argmax function on input data before transform. it also can be a sequence of bool, each element corresponds to a key in keys. to_onehot (Union[Sequence[Optional[int]], int, None]) – if not None, convert input data into the one-hot format with specified number of classes. defaults to None. it also can be a sequence, each element corresponds to a key in keys. threshold (Union[Sequence[Optional[float]], float, None]) – if not None, threshold the float values to int number 0 or 1 with specified threshold value. defaults to None. it also can be a sequence, each element corresponds to a key in keys. rounding (Union[Sequence[Optional[str]], str, None]) – if not None, round the data according to the specified option, available options: ["torchrounding"]. it also can be a sequence of str or None, each element corresponds to a key in keys. allow_missing_keys (bool) – don't raise exception if key is missing. kwargs – additional parameters to AsDiscrete. dim, keepdim, dtype are supported, unrecognized parameters will be ignored. These default to 0, True, torch.float respectively. Dictionary-based wrapper of monai.transforms.KeepLargestConnectedComponent. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform applied_labels (Union[Sequence[int], int, None]) – Labels for applying the connected component analysis on. If given, voxels whose value is in this list will be analyzed. If None, all non-zero values will be analyzed. is_onehot (Optional[bool]) – if True, treat the input data as OneHot format data, otherwise, not OneHot format data. default to None, which treats multi-channel data as OneHot and single channel data as not OneHot. independent (bool) – whether to treat applied_labels as a union of foreground labels. If True, the connected component analysis will be performed on each foreground label independently and return the intersection of the largest components. If False, the analysis will be performed on the union of foreground labels. default is True. connectivity (Optional[int]) – Maximum number of orthogonal hops to consider a pixel/voxel as a neighbor. Accepted values are ranging from 1 to input.ndim. If None, a full connectivity of input.ndim is used. for more details: https://scikit-image.org/docs/dev/api/skimage.measure.html#skimage.measure.label. num_components (int) – The number of largest components to preserve. allow_missing_keys (bool) – don't raise exception if key is missing. Dictionary-based wrapper of monai.transforms.RemoveSmallObjectsd. min_size (int) – objects smaller than this size are removed. connectivity (int) – Maximum number of orthogonal hops to consider a pixel/voxel as a neighbor. Accepted values are ranging from 1 to input.ndim. If None, a full connectivity of input.ndim is used. For more details refer to linked scikit-image documentation. independent_channels (bool) – Whether or not to consider channels as independent. If true, then conjoining islands from different labels will be removed if they are below the threshold. If false, the overall size islands made from all non-background voxels will be used. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Dictionary-based wrapper of monai.transforms.LabelFilter. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform applied_labels (Union[Sequence[int], int]) – Label(s) to filter on. allow_missing_keys (bool) – don't raise exception if key is missing. Dictionary-based wrapper of monai.transforms.FillHoles. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Initialize the connectivity and limit the labels for which holes are filled. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform applied_labels (Optional[Union[Iterable[int], int]], optional) – Labels for which to fill holes. Defaults to None, that is filling holes for all labels. connectivity (int, optional) – Maximum number of orthogonal hops to consider a pixel/voxel as a neighbor. Accepted values are ranging from 1 to input.ndim. Defaults to a full connectivity of input.ndim. allow_missing_keys (bool) – don’t raise exception if key is missing. Dictionary-based wrapper of monai.transforms.LabelToContour. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform kernel_type (str) – the method applied to do edge detection, default is “Laplace”. allow_missing_keys (bool) – don’t raise exception if key is missing. Base class of dictionary-based ensemble transforms. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be stack and execute ensemble. if only 1 key provided, suppose it’s a PyTorch Tensor with data stacked on dimension E. output_key (Optional[str]) – the key to store ensemble result in the dictionary. ensemble (Callable[[Union[Sequence[Union[ndarray, Tensor]], ndarray, Tensor]], Union[ndarray, Tensor]]) – callable method to execute ensemble on specified data. if only 1 key provided in keys, output_key can be None and use keys as default. allow_missing_keys (bool) – don’t raise exception if key is missing. TypeError – When ensemble is not callable. ValueError – When len(keys) > 1 and output_key=None. Incompatible values. Dictionary-based wrapper of monai.transforms.MeanEnsemble. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be stack and execute ensemble. if only 1 key provided, suppose it’s a PyTorch Tensor with data stacked on dimension E. output_key (Optional[str]) – the key to store ensemble result in the dictionary. if only 1 key provided in keys, output_key can be None and use keys as default. weights (Union[Sequence[float], ndarray, Tensor, None]) – can be a list or tuple of numbers for input data with shape: [E, C, H, W[, D]]. or a Numpy ndarray or a PyTorch Tensor data. the weights will be added to input data from highest dimension, for example: 1. if the weights only has 1 dimension, it will be added to the E dimension of input data. 2. if the weights has 2 dimensions, it will be added to E and C dimensions. it’s a typical practice to add weights for different classes: to ensemble 3 segmentation model outputs, every output has 4 channels(classes), so the input data shape can be: [3, 4, H, W, D]. and add different weights for different classes, so the weights shape can be: [3, 4]. for example: weights = [[1, 2, 3, 4], [4, 3, 2, 1], [1, 1, 1, 1]]. Dictionary-based wrapper of monai.transforms.VoteEnsemble. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be stack and execute ensemble. if only 1 key provided, suppose it’s a PyTorch Tensor with data stacked on dimension E. output_key (Optional[str]) – the key to store ensemble result in the dictionary. if only 1 key provided in keys, output_key can be None and use keys as default. num_classes (Optional[int]) – if the input is single channel data instead of One-Hot, we can’t get class number from channel, need to explicitly specify the number of classes to vote. Utility transform to invert the previously applied transforms. Taking the transform previously applied on orig_keys, this Invertd will apply the inverse of it to the data stored at keys. Invertd’s output will also include a copy of the metadata dictionary (originally from orig_meta_keys or the metadata of orig_keys), with the relevant fields inverted and stored at meta_keys. A typical usage is to apply the inverse of the preprocessing (transform=preprocessings) on input orig_keys=image to the model predictions keys=pred. A detailed usage example is available in the tutorial: Project-MONAI/tutorials Note The output of the inverted data and metadata will be stored at keys and meta_keys respectively. To correctly invert the transforms, the information of the previously applied transforms should be available at {orig_keys}_transforms, and the original metadata at orig_meta_keys. (meta_key_postfix is an optional string to conveniently construct “meta_keys” and/or “orig_meta_keys”.) see also: monai.transforms.TraceableTransform. The transform will not change the content in orig_keys and orig_meta_key. These keys are only used to represent the data status of key before inverting. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Any] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – the key of expected data in the dict, the inverse of transforms will be applied on it in-place. It also can be a list of keys, will apply the inverse transform respectively. transform (InvertibleTransform) – the transform applied to orig_key, its inverse will be applied on key. orig_keys (Union[Collection[Hashable], Hashable, None]) – the key of the original input data in the dict. These keys default to self.keys if not set. the transform trace information of transforms should be stored at {orig_keys}_transforms. It can also be a list of keys, each matches the keys. meta_keys (Union[Collection[Hashable], Hashable, None]) – The key to output the inverted metadata dictionary. The metadata is a dictionary optionally containing: filename, original_shape. It can be a sequence of strings, maps to keys. If None, will try to create a metadata dict with the default key: {key}_{meta_key_postfix}. orig_meta_keys (Union[Collection[Hashable], Hashable, None]) – the key of the metadata of original input data. The metadata is a dictionary optionally containing: filename, original_shape. It can be a sequence of strings, maps to the keys. If None, will try to create a metadata dict with the default key: {orig_key}_{meta_key_postfix}. This metadata dict will also be included in the inverted dict, stored in meta_keys. meta_key_postfix (str) – if orig_meta_keys is None, use {orig_key}_{meta_key_postfix} to fetch the metadata from dict, if meta_keys is None, use {key}_{meta_key_postfix}. Default: "meta_dict". nearest_interp (Union[bool, Sequence[bool]]) – whether to use nearest interpolation mode when inverting the spatial transforms, default to True. If False, use the same interpolation mode as the original transform. It also can be a list of bool, each matches to the keys data. to_tensor (Union[bool, Sequence[bool]]) – whether to convert the inverted data into PyTorch Tensor first, default to True. It also can be a list of bool, each matches to the keys data. device (Union[str, device, Sequence[Union[str, device]], None]) – if converted to Tensor, move the inverted results to target device before post_func, default to None, it also can be a list of string or torch.device, each matches to the keys data. post_func (Union[Sequence[Callable], Callable, None]) – post processing for the inverted data, should be a callable function. It also can be a list of callable, each matches to the keys data. allow_missing_keys (bool) – don’t raise exception if key is missing. Save the classification results and metadata into CSV file or other storage. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to model output, this transform only supports 1 key. See also: monai.transforms.compose.MapTransform meta_keys (Union[Collection[Hashable], Hashable, None]) – explicitly indicate the key of the corresponding metadata dictionary. for example, for data with key image, the metadata by default is in image_meta_dict. the metadata is a dictionary object which contains: filename, original_shape, etc. it can be a sequence of string, map to the keys. if None, will try to construct meta_keys by key_{meta_key_postfix}. will extract the filename of input image to save classification results. meta_key_postfix (str) – key_{postfix} was used to store the metadata in LoadImaged. so need the key to extract the metadata of input image, like filename, etc. default is meta_dict. for example, for data with key image, the metadata by default is in image_meta_dict. the metadata is a dictionary object which contains: filename, original_shape, etc. this arg only works when meta_keys=None. if no corresponding metadata, set to None. saver (Optional[CSVSaver]) – the saver instance to save classification results, if None, create a CSVSaver internally. the saver must provide save(data, meta_data) and finalize() APIs. output_dir (Union[str, PathLike]) – if saver=None, specify the directory to save the CSV file. filename (str) – if saver=None, specify the name of the saved

CSV file. delimiter (str) – the delimiter character in the saved file, default to “,” as the default output type is csv. to be consistent with: https://docs.python.org/3/library/csv.html#csv.Dialect.delimiter. overwrite (bool) – if saver=None, indicate whether to overwriting existing CSV file content, if True, will clear the file before saving. otherwise, will append new content to the CSV file. flush (bool) – if saver=None, indicate whether to write the cache data to CSV file immediately in this transform and clear the cache. default to True. If False, may need user to call saver.finalize() manually or use ClassificationSaver handler. allow_missing_keys (bool) – don’t raise exception if key is missing. If want to write content into file, may need to call finalize of saver when epoch completed. Or users can also get the cache content from saver instead of writing into file. Performs probability based non-maximum suppression (NMS) on the probabilities map via iteratively selecting the coordinate with highest probability and then move it as well as its surrounding values. The remove range is determined by the parameter box_size. If multiple coordinates have the same highest probability, only one of them will be selected. spatial_dims (int) – number of spatial dimensions of the input probabilities map. Defaults to 2. sigma (Union[Sequence[float], float, Sequence[Tensor], Tensor]) – the standard deviation for gaussian filter. It could be a single value, or spatial_dims number of values. Defaults to 0.0. prob_threshold (float) – the probability threshold, the function will stop searching if the highest probability is no larger than the threshold. The value should be no less than 0.0. Defaults to 0.5. box_size (Union[int, Sequence[int]]) – the box size (in pixel) to be removed around the pixel with the maximum probability. It can be an integer that defines the size of a square or cube, or a list containing different values for each dimensions. Defaults to 48. a list of selected lists, where inner lists contain probability and coordinates. For example, for 3D input, the inner lists are in the form of [probability, x, y, z]. ValueError – When prob_threshold is less than 0.0. ValueError – When box_size is a list or tuple, and its length is not equal to spatial_dims. ValueError – When box_size has a less than 1 value. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. An updated dictionary version of data by applying the transform. Calculate Sobel horizontal and vertical gradients of a grayscale image. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to model output. kernel_size (int) – the size of the Sobel kernel. Defaults to 3. spatial_axes (Union[Sequence[int], int, None]) – the axes that define the direction of the gradient to be calculated. It calculate the gradient along each of the provide axis. By default it calculate the gradient for all spatial axes. normalize_kernels (bool) – if normalize the Sobel kernel to provide proper gradients. Defaults to True. normalize_gradients (bool) – if normalize the output gradient to 0 and 1. Defaults to False. padding_mode (str) – the padding mode of the image when convolving with Sobel kernels. Defaults to “reflect”. Acceptable values are 'zeros', 'reflect', 'replicate' or 'circular'. See torch.nn.Conv1d() for more information. dtype (dtype) – kernel data type (torch.dtype). Defaults to torch.float32. new_key_prefix (Optional[str]) – this prefix be prepended to the key to create a new key for the output and keep the value of key intact. By default not prefix is set and the corresponding array to the key will be replaced. allow_missing_keys (bool) – don’t raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform.

Spatial (Dict)#

Dictionary-based wrapper of monai.transforms.SpatialResample. This transform assumes the data dictionary has a key for the input data’s metadata and contains src_affine and dst_affine required by SpatialResample. The key is formed by key_{meta_key_postfix}. The transform will swap src_affine and dst_affine affine (with potential data type changes) in the dictionary so that src_affine always refers to the current status of affine. See also monai.transforms.SpatialResample data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. mode (Union[Sequence[str], str]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it’s an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html It also can be a sequence, each element corresponds to a key in keys. padding_mode (Union[Sequence[str], str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "border". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html It also can be a sequence, each element corresponds to a key in keys. align_corners (Union[Sequence[bool], bool]) – Geometrically, we consider the pixels of the input as squares rather than points. See also: https://pytorch.org/docs/stable/nn.functional.html#grid-sample It also can be a sequence of bool, each element corresponds to a key in keys. dtype (Union[Sequence[Union[dtype, type, str, None]], dtype, type, str, None]) – data type for resampling computation. Defaults to float64 for best precision. If None, use the data type of input data. To be compatible with other modules, the output data type is always float32. It also can be a sequence of dtypes, each element corresponds to a key in keys. dst_keys (Union[Collection[Hashable], Hashable, None]) – the key of the corresponding dst_affine in the metadata dictionary. allow_missing_keys (bool) – don't raise exception if key is missing. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Dictionary-based wrapper of monai.transforms.ResampleToMatch. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. key_dst (str) – key of image to resample to match. mode (Union[Sequence[str], str]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it's an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html It also can be a sequence, each element corresponds to a key in keys. padding_mode (Union[Sequence[str], str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "border". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {'reflect', 'grid-mirror', 'constant', 'grid-constant', 'nearest', 'mirror', 'grid-wrap', 'wrap'}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates .html It also can be a sequence, each element corresponds to a key in keys. align_corners (Union[Sequence[bool], bool]) – Geometrically, we consider the pixels of the input as squares rather than points. See also: https://pytorch.org/docs/stable/nn.functional.html#grid-sample It also can be a sequence of bool, each element corresponds to a key in keys. dtype (Union[Sequence[Union[dtype, type, str, None]], dtype, type, str, None]) – data type for resampling computation. Defaults to float64 for best precision. If None, use the data type of input data. To be compatible with other modules, the output data type is always float32. It also can be a sequence of dtypes, each element corresponds to a key in keys. allow_missing_keys (bool) – don't raise exception if key is missing. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Dictionary-based wrapper of monai.transforms.Spacing. This transform assumes the data dictionary has a key for the input data's metadata and contains affine field. The key is formed by key_{meta_key_postfix}. After resampling the input array, this transform will write the new affine to the affine field of metadata which is formed by key_{meta_key_postfix}. See also monai.transforms.Spacing data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] An updated dictionary version of data by applying the transform. pixdim (Union[Sequence[float], float]) – output voxel spacing. if providing a single number, will use it for the first dimension. items of the pixdim sequence map to the spatial dimensions of input image, if length of pixdim sequence is longer than image spatial dimensions, will ignore the longer part, if shorter, will pad with 1.0. if the components of the pixdim are non-positive values, the transform will use the corresponding components of the original pixdim, which is computed from the affine matrix of input image. diagonal (bool) – whether to resample the input to have a diagonal affine matrix. If True, the input data is resampled to the following affine: np.diag((pixdim_0, pixdim_1, pixdim_2, 1)) This effectively resets the volume to the world coordinate system (RAS+ in nibabel). The original orientation, rotation, shearing are not preserved. If False, the axes orientation, orthogonal rotation and translations components from the original affine will be preserved in the target affine. This option will not flip/swap axes against the original ones. whether to resample the input to have a diagonal affine matrix. If True, the input data is resampled to the following affine: This effectively resets the volume to the world coordinate system (RAS+ in nibabel). The original orientation, rotation, shearing are not preserved. If False, the axes orientation, orthogonal rotation and translations components from the original affine will be preserved in the target affine. This option will not flip/swap axes against the original ones. mode (Union[Sequence[str], str]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it’s an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html It also can be a sequence, each element corresponds to a key in keys. padding_mode (Union[Sequence[str], str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "border". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates .html It also can be a sequence, each element corresponds to a key in keys. align_corners (Union[Sequence[bool], bool]) – Geometrically, we consider the pixels of the input as squares rather than points. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html It also can be a sequence of bool, each element corresponds to a key in keys. dtype (Union[Sequence[Union[dtype, type, str, None]], dtype, type, str, None]) – data type for resampling computation. Defaults to float64 for best precision. If None, use the data type of input data. To be compatible with other modules, the output data type is always float32. It also can be a sequence of dtypes, each element corresponds to a key in keys. scale_extent (bool) – whether the scale is computed based on the spacing or the full extent of voxels, default False. The option is ignored if output spatial size is specified when calling this transform. See also: monai.data.utils.compute_shape_offset(). When this is True, align_corners should be True because compute_shape_offset already provides the corner alignment shift/scaling. recompute_affine (bool) – whether to recompute affine based on the output shape. The affine computed analytically does not reflect the potential quantization errors in terms of the output shape. Set this flag to True to recompute the output affine based on the actual pixdim. Default to False. min_pixdim (Union[Sequence[float], float, None]) – minimal input spacing to be resampled. If provided, input image with a larger spacing than this value will be kept in its original spacing (not be resampled to pixdim). Set it to None to use the value of pixdim. Default to None. max_pixdim (Union[Sequence[float], float, None]) – maximal input spacing to be resampled. If provided, input image with a smaller spacing than this value will be kept in its original spacing (not be resampled to pixdim). Set it to None to use the value of pixdim. Default to None. allow_missing_keys (bool) – don't raise exception if key is missing. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] Dictionary-based wrapper of monai.transforms.Orientation. This transform assumes the channel-first input format. In the case of using this transform for normalizing the orientations of images, it should be used before any anisotropic spatial transforms. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] An updated dictionary version of data by applying the transform. axcodes (Optional[str]) – N elements sequence for spatial ND input's orientation. e.g. axcodes='RAS' represents 3D orientation: (Left, Right), (Posterior, Anterior), (Inferior, Superior). default orientation labels options are: 'L' and 'R' for the first dimension, 'P' and 'A' for the second, 'I' and 'S' for the third. as_closest_canonical (bool) – if True, load the image as closest to canonical axis format. labels (Optional[Sequence[Tuple[str, str]]]) – optional, None or sequence of (2,) sequences (2,) sequences are labels for (beginning, end) of output axis. Defaults to (('L', 'R'), ('P', 'A'), ('I', 'S')). allow_missing_keys (bool) – don't raise exception if key is missing. See also nibabel.orientations.ornt2axcodes. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Dictionary-based wrapper of monai.transforms.Flip. See numpy.flip for additional details. https://docs.scipy.org/doc/numpy/reference/generated/numpy.flip.html keys (Union[Collection[Hashable], Hashable]) – Keys to pick data for transformation. spatial_axis (Union[Sequence[int], int, None]) – Spatial axes along which to flip over. Default is None. allow_missing_keys (bool) – don't raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] An updated dictionary version of data by applying the transform. Inverse of __call__.

NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Dictionary-based version monai.transforms.RandFlip. See numpy.flip for additional details. https://docs.scipy.org/doc/numpy/reference/generated/numpy.flip.html keys (Union[Collection[Hashable], Hashable]) – Keys to pick data for transformation. prob (float) – Probability of flipping. spatial_axis (Union[Sequence[int], int, None]) – Spatial axes along which to flip over. Default is None. allow_missing_keys (bool) – don't raise exception if key is missing. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandFlipd a Randomizable instance. Dictionary-based version monai.transforms.RandAxisFlip. See numpy.flip for additional details. https://docs.scipy.org/doc/numpy/reference/generated/numpy.flip.html keys (Union[Collection[Hashable], Hashable]) – Keys to pick data for transformation. prob (float) – Probability of flipping. allow_missing_keys (bool) – don't raise exception if key is missing. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandAxisFlipd a Randomizable instance. Dictionary-based wrapper of monai.transforms.Rotate. keys (Union[Collection[Hashable], Hashable]) – Keys to pick data for transformation. angle (Union[Sequence[float], float]) – Rotation angle(s) in radians. keep_size (bool) – If it is False, the output shape is adapted so that the input array is contained completely in the output. If it is True, the output shape is the same as the input. Default is True.

mode (Union[Sequence[str], str]) – {"bilinear", "nearest"} Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html It also can be a sequence of string, each element corresponds to a key in keys. padding_mode (Union[Sequence[str], str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "border". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html It also can be a sequence of string, each element corresponds to a key in keys. align_corners (Union[Sequence[bool], bool]) – Defaults to False. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html It also can be a sequence of bool, each element corresponds to a key in keys. dtype (Union[Sequence[Union[dtype, type, str, None, dtype]], dtype, type, str, None, dtype]) – data type for resampling computation. Defaults to float32. If None, use the data type of input data. To be compatible with other modules, the output data type is always float32. It also can be a sequence of dtype or None, each element corresponds to a key in keys. allow_missing_keys (bool) – don't raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] An updated dictionary version of data by applying the transform. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Dictionary-based version monai.transforms.RandRotate Randomly rotates the input arrays. keys (Union[Collection[Hashable], Hashable]) – Keys to pick data for transformation. range_x (Union[Tuple[float, float], float]) – Range of rotation angle in radians in the plane defined by the first and second axes. If single number, angle is uniformly sampled from (-range_x, range_x). range_y (Union[Tuple[float, float], float]) – Range of rotation angle in radians in the plane defined by the first and third axes. If single number, angle is uniformly sampled from (-range_y, range_y). only work for 3D data. range_z (Union[Tuple[float, float], float]) – Range of rotation angle in radians in the plane defined by the second and third axes. If single number, angle is uniformly sampled from (-range_z, range_z). only work for 3D data. prob (float) – Probability of rotation. keep_size (bool) – If it is False, the output shape is adapted so that the input array is contained completely in the output. If it is True, the output shape is the same as the input. Default is True. mode (Union[Sequence[str], str]) – {"bilinear", "nearest"} Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html It also can be a sequence of string, each element corresponds to a key in keys. padding_mode (Union[Sequence[str], str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "border". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html It also can be a sequence of string, each element corresponds to a key in keys. align_corners (Union[Sequence[bool], bool]) – Defaults to False. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html It also can be a sequence of bool, each element corresponds to a key in keys. dtype (Union[Sequence[Union[dtype, type, str, None, dtype]], dtype, type, str, None, dtype])

– data type for resampling computation. Defaults to float64 for best precision. If None, use the data type of input data. To be compatible with other modules, the output data type is always float32. It also can be a sequence of dtype or None, each element corresponds to a key in keys. allow_missing_keys (bool) – don’t raise exception if key is missing. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandRotated a Randomizable instance. Dictionary-based wrapper of monai.transforms.Zoom. keys (Union[Collection[Hashable], Hashable]) – Keys to pick data for transformation. zoom (Union[Sequence[float], float]) – The zoom factor along the spatial axes. If a float, zoom is the same for each spatial axis. If a sequence, zoom should contain one value for each spatial axis. mode (Union[Sequence[str], str]) – {"nearest", "nearest-exact", "linear", "bilinear", "bicubic", "trilinear", "area"} The interpolation mode. Defaults to "area". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html It also can be a sequence of string, each element corresponds to a key in keys. padding_mode (Union[Sequence[str], str]) – available modes for numpy array:{"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "edge". The mode to pad data after zooming. See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html align_corners (Union[Sequence[Optional[bool]], bool, None]) – This only has an effect when mode is ‘linear’, ‘bilinear’, ‘bicubic’ or ‘trilinear’. Default: None. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html It also can be a sequence of bool or None, each element corresponds to a key in keys. keep_size (bool) – Should keep original size (pad if needed), default is True. allow_missing_keys (bool) – don’t raise exception if key is missing. kwargs – other arguments for the np.pad or torch.pad function. note that np.pad treats channel dimension as the first dimension. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] An updated dictionary version of data by applying the transform. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Dict-based version monai.transforms.RandZoom. keys (Union[Collection[Hashable], Hashable]) – Keys to pick data for transformation. prob (float) – Probability of zooming. min_zoom (Union[Sequence[float], float]) – Min zoom factor. Can be float or sequence same size as image. If a float, select a random factor from [min_zoom, max_zoom] then apply to all spatial dims to keep the original spatial shape ratio. If a sequence, min_zoom should contain one value for each spatial axis. If 2 values provided for 3D data, use the first value for both H & W dims to keep the same zoom ratio. max_zoom (Union[Sequence[float], float]) – Max zoom factor. Can be float or sequence same size as image. If a float, select a random factor from [min_zoom, max_zoom] then apply to all spatial dims to keep the original spatial shape ratio. If a sequence, max_zoom should contain one value for each spatial axis. If 2 values provided for 3D data, use the first value for both H & W dims to keep the same zoom ratio. mode (Union[Sequence[str], str]) – {"nearest", "nearest-exact", "linear", "bilinear", "bicubic", "trilinear", "area"} The interpolation mode. Defaults to "area". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html It also can be a sequence of string, each element corresponds to a key in keys. padding_mode (Union[Sequence[str], str]) – available modes for numpy array:{"constant", "edge", "linear_ramp", "maximum", "mean", "median", "minimum", "reflect", "symmetric", "wrap", "empty"} available modes for PyTorch Tensor: {"constant", "reflect", "replicate", "circular"}. One of the listed string values or a user supplied function. Defaults to "edge". The mode to pad data after zooming. See also: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html https://pytorch.org/docs/stable/generated/torch.nn.functional.pad.html align_corners (Union[Sequence[Optional[bool]], bool, None]) – This only has an effect when mode is ‘linear’, ‘bilinear’, ‘bicubic’ or ‘trilinear’. Default: None. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html It also can be a sequence of bool or None, each element corresponds to a key in keys. keep_size (bool) – Should keep original size (pad if needed), default is True. allow_missing_keys (bool) – don’t raise exception if key is missing. kwargs – other args for np.pad API, note that np.pad treats channel dimension as the first dimension. more details: https://numpy.org/doc/1.18/reference/generated/numpy.pad.html data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState].

RandZoomd a Randomizable instance. Extract all the patches sweeping the entire image in a row-major sliding-window manner with possible overlaps. It can sort the patches and return all or a subset of them. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. patch_size (Sequence[int]) – size of patches to generate slices for, 0 or None selects whole dimension offset (Optional[Sequence[int]]) – starting position in the array, default is 0 for each dimension. np.random.randint(0, patch_size, 2) creates random start between 0 and patch_size for a 2D image. num_patches (Optional[int]) – number of patches to return. Defaults to None, which returns all the available patches. overlap (float) – amount of overlap between patches in each dimension. Default to 0.0. sort_fn (Optional[str]) – when num_patches is provided, it determines if keep patches with highest values (“max”), lowest values (“min”), or in their default order (None). Default to None. threshold (Optional[float]) – a value to keep only the patches whose sum of intensities are less than the threshold. Defaults to no filtering. pad_mode (str) – refer to NumpyPadMode and PytorchPadMode. If None, no padding will be applied. Defaults to "constant". allow_missing_keys (bool) – don’t raise exception if key is missing. pad_kwargs – other arguments for the np.pad or torch.pad function. a list of dictionaries, each of which contains the all the original key/value with the values for keysreplaced by the patches. It also add the following new keys: ”patch_location”: the starting location of the patch in the image, “patch_size”: size of the extracted patch “num_patches”: total number of patches in the image “offset”: the amount of offset for the patches in the image (starting position of upper left patch) replaced by the patches. It also add the following new keys: ”patch_location”: the starting location of the patch in the image, “patch_size”: size of the extracted patch “num_patches”: total number of patches in the image “offset”: the amount of offset for the patches in the image (starting position of upper left patch) data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Extract all the patches sweeping the entire image in a row-major sliding-window manner with possible overlaps, and with random offset for the minimal corner of the image, (0,0) for 2D and (0,0,0) for 3D. It can sort the patches and return all or a subset of them. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. patch_size (Sequence[int]) – size of patches to generate slices for, 0 or None selects whole dimension min_offset (Union[Sequence[int], int, None]) – the minimum range of starting position to be selected randomly. Defaults to 0. max_offset (Union[Sequence[int], int, None]) – the maximum range of starting position to be selected randomly. Defaults to image size modulo patch size. num_patches (Optional[int]) – number of patches to return. Defaults to None, which returns all the available patches. overlap (float) – the amount of overlap of neighboring patches in each dimension (a value between 0.0 and 1.0). If only one float number is given, it will be applied to all dimensions. Defaults to 0.0. sort_fn (Optional[str]) – when num_patches is provided, it determines if keep patches with highest values (“max”), lowest values (“min”), or in their default order (None). Default to None. threshold (Optional[float]) – a value to keep only the patches whose sum of intensities are less than the threshold. Defaults to no filtering. pad_mode (str) –

refer to NumpyPadMode and PytorchPadMode. If None, no padding will be applied. Defaults to "constant". allow_missing_keys (bool) – don't raise exception if key is missing. pad_kwargs – other arguments for the np.pad or torch.pad function. a list of dictionaries, each of which contains the all the original key/value with the values for keysreplaced by the patches. It also add the following new keys: "patch_location": the starting location of the patch in the image, "patch_size": size of the extracted patch "num_patches": total number of patches in the image "offset": the amount of offset for the patches in the image (starting position of the first patch) replaced by the patches. It also add the following new keys: "patch_location": the starting location of the patch in the image, "patch_size": size of the extracted patch "num_patches": total number of patches in the image "offset": the amount of offset for the patches in the image (starting position of the first patch) data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandGridPatchd a Randomizable instance. Split the image into patches based on the provided grid in 2D. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. grid (Tuple[int, int]) – a tuple define the shape of the grid upon which the image is split. Defaults to (2, 2) size (Union[int, Tuple[int, int], Dict[Hashable, Union[int, Tuple[int, int], None]], None]) – a tuple or an integer that defines the output patch sizes, or a dictionary that define it separately for each key, like {"image": 3, "mask", (2, 2)}. If it's an integer, the value will be repeated for each dimension. The default is None, where the patch size will be inferred from the grid shape. allow_missing_keys (bool) – don't raise exception if key is missing. Note: This transform currently support only image with two spatial dimensions. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. List[Dict[Hashable, Union[ndarray, Tensor]]] An updated dictionary version of data by applying the transform. Dictionary-based version monai.transforms.RandRotate90. With probability prob, input arrays are rotated by 90 degrees in the plane specified by spatial_axes. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Mapping[Hashable, Tensor] keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform prob (float) – probability of rotating. (Default 0.1, with 10% probability it returns a rotated array.) max_k (int) – number of rotations will be sampled from np.random.randint(max_k) + 1. (Default 3) spatial_axes (Tuple[int, int]) – 2 int numbers, defines the plane to rotate with 2 spatial axes. Default: (0, 1), this is the first two axis in spatial dimensions. allow_missing_keys (bool) – don’t raise exception if key is missing. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Dictionary-based wrapper of monai.transforms.Rotate90. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] An updated dictionary version of data by applying the transform. k (int) – number of times to rotate by 90 degrees. spatial_axes (Tuple[int, int]) – 2 int numbers, defines the plane to rotate with 2 spatial axes. Default: (0, 1), this is the first two axis in spatial dimensions. allow_missing_keys (bool) – don’t raise exception if key is missing. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Dictionary-based wrapper of monai.transforms.Resize. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform spatial_size (Union[Sequence[int], int]) – expected shape of spatial dimensions after resize operation. if some components of the spatial_size are non-positive values, the transform will use the corresponding components of img size. For example, spatial_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. size_mode (str) – should be “all” or “longest”, if “all”, will use spatial_size for all the spatial dims, if “longest”, rescale the image so that only the longest side is equal to specified spatial_size, which must be an int number in this case, keeping the aspect ratio of the initial image, refer to: https://albumentations.ai/docs/api_reference/augmentations/geometric/resize/#albumentations.augmentations.geometric.resize.LongestMaxSize. mode (Union[Sequence[str], str]) – {"nearest", "nearest-exact", "linear", "bilinear", "bicubic", "trilinear", "area"} The interpolation mode. Defaults to "area". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html It also can be a sequence of string, each element corresponds to a key in keys.

align_corners (Union[Sequence[Optional[bool]], bool, None]) – This only has an effect when mode is 'linear', 'bilinear', 'bicubic' or 'trilinear'. Default: None. See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.interpolate.html It also can be a sequence of bool or None, each element corresponds to a key in keys. anti_aliasing (Union[Sequence[bool], bool]) – bool Whether to apply a Gaussian filter to smooth the image prior to downsampling. It is crucial to filter when downsampling the image to avoid aliasing artifacts. See also skimage.transform.resize anti_aliasing_sigma (Union[Sequence[Union[Sequence[float], float, None]], Sequence[float], float, None]) – {float, tuple of floats}, optional Standard deviation for Gaussian filtering used when anti-aliasing. By default, this value is chosen as (s - 1) / 2 where s is the downsampling factor, where s > 1. For the up-size case, s < 1, no anti-aliasing is performed prior to rescaling. allow_missing_keys (bool) – don't raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] An updated dictionary version of data by applying the transform. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Dictionary-based wrapper of monai.transforms.Affine. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. rotate_params (Union[Sequence[float], float, None]) – a rotation angle in radians, a scalar for 2D image, a tuple of 3 floats for 3D. Defaults to no rotation. shear_params (Union[Sequence[float], float, None]) – shearing factors for affine matrix, take a 3D affine as example: [[1.0, params[0], params[1], 0.0], [params[2], 1.0, params[3], 0.0], [params[4], params[5], 1.0, 0.0], [0.0, 0.0, 0.0, 1.0],] a tuple of 2 floats for 2D, a tuple of 6 floats for 3D. Defaults to no shearing. shearing factors for affine matrix, take a 3D affine as example: translate_params (Union[Sequence[float], float, None]) – a tuple of 2 floats for 2D, a tuple of 3 floats for 3D. Translation is in pixel/voxel relative to the center of the input image. Defaults to no translation. scale_params (Union[Sequence[float], float, None]) – scale factor for every spatial dims. a tuple of 2 floats for 2D, a tuple of 3 floats for 3D. Defaults to 1.0. affine (Union[ndarray, Tensor, None]) – if applied, ignore the params (rotate_params, etc.) and use the supplied matrix. Should be square with each side = num of image spatial dimensions + 1. spatial_size (Union[Sequence[int], int, None]) – output image spatial size. if spatial_size and self.spatial_size are not defined, or smaller than 1, the transform will use the spatial size of img. if some components of the spatial_size are non-positive values, the transform will use the corresponding components of img size. For example, spatial_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. mode (Union[Sequence[str], str]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it's an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html It also can be a sequence, each element corresponds to a key in keys. padding_mode (Union[Sequence[str], str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "reflection". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {'reflect', 'grid-mirror', 'constant', 'grid-constant', 'nearest', 'mirror', 'grid-wrap', 'wrap'}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates .html It also can be a sequence, each element corresponds to a key in keys. device (Optional[device]) – device on which the tensor will be allocated. dtype (Union[dtype, type, str, None, dtype]) – data type for resampling computation. Defaults to float32. If None, use the data type of input data. To be compatible with other modules, the output data type is always float32. allow_missing_keys (bool) – don't raise exception if key is missing. See also monai.transforms.compose.MapTransform RandAffineGrid for the random affine parameters configurations. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Dictionary-based wrapper of monai.transforms.RandAffine. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. spatial_size (Union[Sequence[int], int, None]) – output image spatial size. if spatial_size and self.spatial_size are not defined, or smaller than 1, the transform will use the spatial size of img. if some components of the spatial_size are non-positive values, the transform will use the corresponding components of img size. For example, spatial_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. prob (float) – probability of returning a randomized affine grid. defaults to 0.1, with 10% chance returns a randomized grid. rotate_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – angle range in radians. If element i is a pair of (min, max) values, then uniform[-rotate_range[i][0], rotate_range[i][1]) will be used to generate the rotation parameter for the i`th spatial dimension. If not, `uniform[-rotate_range[i], rotate_range[i]) will be used. This can be altered on a per-dimension basis. E.g., ((0,3), 1, …): for dim0, rotation will be in range [0, 3], and for dim1 [-1, 1] will be used. Setting a single value will use [-x, x] for dim0 and nothing for the remaining dimensions. shear_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – shear range with format matching rotate_range, it defines the range to randomly select shearing factors(a tuple of 2 floats for 2D, a tuple of 6 floats for 3D) for affine matrix, take a 3D affine as example: [[1.0, params[0], params[1], 0.0], [params[2], 1.0, params[3], 0.0], [params[4], params[5], 1.0, 0.0], [0.0, 0.0, 0.0, 1.0],] shear range with format matching rotate_range, it defines the range to randomly select shearing factors(a tuple of 2 floats for 2D, a tuple of 6 floats for 3D) for affine matrix, take a 3D affine as example: translate_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – translate range with format matching rotate_range, it defines the range to randomly select pixel/voxel to translate for every spatial dims. scale_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – scaling range with format matching rotate_range. it defines the range to randomly select the scale factor to translate for every spatial dims. A value of 1.0 is added to the result. This allows 0 to correspond to no change (i.e., a scaling of 1.0). mode (Union[Sequence[str], str]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it's an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html It also can be a sequence, each element corresponds to a key in keys. padding_mode (Union[Sequence[str], str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "reflection". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {'reflect', 'grid-mirror', 'constant', 'grid-constant', 'nearest', 'mirror', 'grid-wrap', 'wrap'}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html It also can be a sequence, each element corresponds to a key in keys. cache_grid (bool) – whether to cache the identity sampling grid. If the spatial size is not dynamically defined by input image, enabling this option could accelerate the transform. device (Optional[device]) – device on which the tensor will be allocated. allow_missing_keys (bool) – don't raise exception if key is missing. See also monai.transforms.compose.MapTransform RandAffineGrid for the random affine parameters configurations. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandAffined a Randomizable instance. Dictionary-based wrapper of monai.transforms.Rand2DElastic. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. spacing (Union[Tuple[float, float], float]) – distance in between the control points. magnitude_range (Tuple[float, float]) – 2 int numbers, the random offsets will be generated from uniform[magnitude[0], magnitude[1]). spatial_size (Union[Tuple[int, int], int, None]) – specifying output image spatial size [h, w]. if spatial_size and self.spatial_size are not defined, or smaller than 1, the transform will use the spatial size of img. if some components of the spatial_size are non-positive values, the transform will use the corresponding components of img size. For example, spatial_size=(32, -1) will be adapted to (32, 64) if the second spatial dimension size of img is 64. prob (float) – probability of returning a randomized affine grid. defaults to 0.1, with 10% chance returns a randomized grid, otherwise returns a spatial_size centered area extracted from the input image. rotate_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – angle range in radians. If element i is a pair of (min, max) values, then uniform[-rotate_range[i][0], rotate_range[i][1]) will be used to generate the rotation parameter for the i`th spatial dimension. If not, `uniform[-rotate_range[i], rotate_range[i]) will be used. This can be altered on a per-dimension basis. E.g., ((0,3), 1, …): for dim0, rotation will be in range [0, 3], and for dim1 [-1, 1] will be used. Setting a single value will use [-x, x] for dim0 and nothing for the remaining dimensions. shear_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – shear range with format matching rotate_range, it defines the range to randomly select shearing factors(a tuple of 2 floats for 2D) for affine matrix, take a 2D affine as example: [[1.0, params[0], 0.0], [params[1], 1.0, 0.0], [0.0, 0.0, 1.0],] shear range with format matching rotate_range, it defines the range to randomly select shearing factors(a tuple of 2 floats for 2D) for affine matrix, take a 2D affine as example: translate_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – translate range with format matching rotate_range, it defines the range to randomly select pixel to translate for every spatial dims. scale_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – scaling range with format matching rotate_range. it defines the range to randomly select the scale factor to translate for every spatial dims. A value of 1.0 is added to the result. This allows 0 to correspond to no change (i.e., a scaling of 1.0). mode (Union[Sequence[str], str]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it’s an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html It also can be a sequence, each element corresponds to a key in keys. padding_mode (Union[Sequence[str], str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "reflection". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates .html It also can be a sequence, each element corresponds to a key in keys. device (Optional[device]) – device on which the tensor will be allocated. allow_missing_keys (bool) – don’t raise exception if key is missing. See also RandAffineGrid for the random affine parameters configurations. Affine for the affine transformation parameters configurations. Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. Rand2DElasticd a Randomizable instance. Dictionary-based wrapper of monai.transforms.Rand3DElastic. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. sigma_range (Tuple[float, float]) – a Gaussian kernel with standard deviation sampled from uniform[sigma_range[0], sigma_range[1]) will be used to smooth the random offset grid. magnitude_range (Tuple[float, float]) – the random offsets on the grid will be generated from uniform[magnitude[0], magnitude[1]). spatial_size (Union[Tuple[int, int, int], int, None]) – specifying output image spatial size [h, w, d]. if spatial_size and self.spatial_size are not defined, or smaller than 1, the transform will use the spatial size of img. if some components of the spatial_size are non-positive values, the transform will use the corresponding components of img size. For example, spatial_size=(32, 32, -1) will be adapted to (32, 32, 64) if the third spatial dimension size of img is 64. prob (float) – probability of returning a randomized affine grid. defaults to 0.1, with 10% chance returns a randomized grid, otherwise returns a spatial_size centered area extracted from the input image. rotate_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – angle range in radians. If element i is a pair of (min, max) values, then uniform[-rotate_range[i][0], rotate_range[i][1]) will be used to generate the rotation parameter for the i`th spatial dimension. If not, `uniform[-rotate_range[i], rotate_range[i]) will be used. This can be altered on a per-dimension basis. E.g., ((0,3), 1, …): for dim0, rotation will be in range [0, 3], and for dim1 [-1, 1] will be used. Setting a single value will use [-x, x] for dim0 and nothing for the remaining dimensions. shear_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – shear range with format matching rotate_range, it defines the range to randomly select shearing factors(a tuple of 6 floats for 3D) for affine matrix, take a 3D affine as example: [[1.0, params[0], params[1], 0.0], [params[2], 1.0, params[3], 0.0], [params[4], params[5], 1.0, 0.0], [0.0, 0.0, 0.0, 1.0],] shear range with format matching rotate_range, it defines the range to randomly select shearing factors(a tuple of 6 floats for 3D) for affine matrix, take a 3D affine as example: translate_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – translate range with format matching rotate_range, it defines the range to randomly select voxel to translate for every spatial dims. scale_range (Union[Sequence[Union[Tuple[float, float], float]], float, None]) – scaling range with format matching rotate_range. it defines the range to randomly select the scale factor to translate for every spatial dims. A value of 1.0 is added to the result. This allows 0 to correspond to no change (i.e., a scaling of 1.0). mode (Union[Sequence[str], str]) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it’s an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html It also can be a sequence, each element corresponds to a key in keys. padding_mode (Union[Sequence[str], str]) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "reflection". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates .html It also can be a sequence, each element corresponds to a key in keys. device (Optional[device]) – device on which the tensor will be allocated. allow_missing_keys (bool) – don’t raise exception if key is missing. See also RandAffineGrid for the random affine parameters configurations. Affine for the affine transformation parameters configurations. Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. Rand3DElasticd a Randomizable instance. Dictionary-based wrapper of monai.transforms.GridDistortion. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. num_cells (Union[Tuple[int], int]) – number of grid cells on each dimension. distort_steps (List[Tuple]) – This argument is a list of tuples, where each tuple contains the distort steps of the corresponding dimensions (in the order of H, W[, D]). The length of each tuple equals to num_cells + 1. Each value in the tuple represents the distort step of the related cell. mode (str) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it’s an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy. org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html It also can be a sequence, each element corresponds to a key in keys. padding_mode (str) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "border". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates .html It also can be a sequence, each element corresponds to a key in keys. device (Optional[device]) – device on which the tensor will be allocated. allow_missing_keys (bool) – don’t raise exception if key is missing. Dictionary-based wrapper of monai.transforms.RandGridDistortion. data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. num_cells (Union[Tuple[int], int]) – number of grid cells on each dimension. prob (float) – probability of returning a randomized grid distortion transform. Defaults to 0.1. distort_limit (Union[Tuple[float, float], float]) – range to randomly distort. If single number, distort_limit is picked from (-distort_limit, distort_limit). Defaults to (-0.03, 0.03). mode (str) – {"bilinear", "nearest"} or spline interpolation order 0-5 (integers). Interpolation mode to calculate output values. Defaults to "bilinear". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When it’s an integer, the numpy (cpu tensor)/cupy (cuda tensor) backends will be used and the value represents the order of the spline interpolation. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html It also can be a sequence, each element corresponds to a key in keys. padding_mode (str) – {"zeros", "border", "reflection"} Padding mode for outside grid values. Defaults to "border". See also: https://pytorch.org/docs/stable/generated/torch.nn.functional.grid_sample.html When mode is an integer, using numpy/cupy backends, this argument accepts {‘reflect’, ‘grid-mirror’, ‘constant’, ‘grid-constant’, ‘nearest’, ‘mirror’, ‘grid-wrap’, ‘wrap’}. See also: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html It also can be a sequence, each element corresponds to a key in keys. device (Optional[device]) – device on which the tensor will be allocated. allow_missing_keys (bool) – don’t raise exception if key is missing. Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandGridDistortiond a Randomizable instance.

Smooth Field (Dict)#

Dictionary version of RandSmoothFieldAdjustContrast. The field is randomized once per invocation by default so the same field is applied to every selected key. The mode parameter specifying interpolation mode for the field can be a single value or a sequence of values with one for each key in keys. keys (Union[Collection[Hashable], Hashable]) – key names to apply the augment to spatial_size (Sequence[int]) – size of input arrays, all arrays stated in keys must have same dimensions rand_size (Sequence[int]) – size of the randomized field to start from pad (int) – number of pixels/voxels along the edges of the field to pad with 0 mode (Union[Sequence[str], str]) – interpolation mode to use when upsampling align_corners (Optional[bool]) – if True align the corners when upsampling field prob (float) – probability transform is applied gamma (Union[Sequence[float], float]) – (min, max) range for exponential field device (Optional[device]) – Pytorch device to define field on data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Mapping[Hashable, Union[ndarray, Tensor]] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandSmoothFieldAdjustContrastd a Randomizable instance. Dictionary version of RandSmoothFieldAdjustIntensity. The field is randomized once per invocation by default so the same field is applied to every selected key. The mode parameter specifying interpolation mode for the field can be a single value or a sequence of values with one for each key in keys. keys (Union[Collection[Hashable], Hashable]) – key names to apply the augment to spatial_size (Sequence[int]) – size of input arrays, all arrays stated in keys must have same dimensions rand_size (Sequence[int]) – size of the randomized field to start from pad (int) – number of pixels/voxels along the edges of the field to pad with 0 mode (Union[Sequence[str], str]) – interpolation mode to use when upsampling align_corners (Optional[bool]) – if True align the corners when upsampling field prob (float) – probability transform is applied gamma (Union[Sequence[float], float]) – (min, max) range of intensity multipliers device (Optional[device]) – Pytorch device to define field on data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Mapping[Hashable, Union[ndarray, Tensor]] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandSmoothFieldAdjustIntensityd a Randomizable instance. Dictionary version of RandSmoothDeform. The field is randomized once per invocation by default so the same field is applied to every selected key. The field_mode parameter specifying interpolation mode for the field can be a single value or a sequence of values with one for each key in keys. Similarly the grid_mode parameter can be one value or one per key. keys

(Union[Collection[Hashable], Hashable]) – key names to apply the augment to spatial_size (Sequence[int]) – input array size to which deformation grid is interpolated rand_size (Sequence[int]) – size of the randomized field to start from pad (int) – number of pixels/voxels along the edges of the field to pad with 0 field_mode (Union[Sequence[str], str]) – interpolation mode to use when upsampling the deformation field align_corners (Optional[bool]) – if True align the corners when upsampling field prob (float) – probability transform is applied def_range (Union[Sequence[float], float]) – value of the deformation range in image size fractions grid_dtype – type for the deformation grid calculated from the field grid_mode (Union[Sequence[str], str]) – interpolation mode used for sampling input using deformation grid grid_padding_mode (str) – padding mode used for sampling input using deformation grid grid_align_corners (Optional[bool]) – if True align the corners when sampling the deformation grid device (Optional[device]) – Pytorch device to define field on data is an element which often comes from an iteration over an iterable, such as torch.utils.data.Dataset. This method should return an updated version of data. To simplify the input validations, most of the transforms assume that data is a Numpy ndarray, PyTorch Tensor or string, the data shape can be: string data without shape, LoadImage transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannel expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirst expects (spatial_dim_1[, spatial_dim_2, …], num_channels), the channel dimension is often not omitted even if number of channels is one. This method can optionally take additional arguments to help execute transformation operation. NotImplementedError – When the subclass does not override this method. Mapping[Hashable, Union[ndarray, Tensor]] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. None Set the random state locally, to control the randomness, the derived classes should use self.R instead of np.random to introduce random factors. seed (Optional[int]) – set the random state with an integer seed. state (Optional[RandomState]) – set the random state with a np.random.RandomState object. TypeError – When state is not an Optional[np.random.RandomState]. RandSmoothDeformd a Randomizable instance.

MRI transforms (Dict)#

Dictionary-based wrapper of monai.apps.reconstruction.transforms.array.RandomKspacemask. Other mask transforms can inherit from this class, for example: monai.apps.reconstruction.transforms.dictionary.EquispacedKspaceMaskd. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.MapTransform center_fractions (Sequence[float]) – Fraction of low-frequency columns to be retained. If multiple values are provided, then one of these numbers is chosen uniformly each time. accelerations (Sequence[float]) – Amount of under-sampling. This should have the same length as center_fractions. If multiple values are provided, then one of these is chosen uniformly each time. spatial_dims (int) – Number of spatial dims (e.g., it’s 2 for a 2D data; it’s also 2 for pseudo-3D datasets like the fastMRI dataset). The last spatial dim is selected for sampling. For the fastMRI dataset, k-space has the form (…,num_slices,num_coils,H,W) and sampling is done along W. For a general 3D data with the shape (…,num_coils,H,W,D), sampling is done along D. is_complex (bool) –

if True, then the last dimension will be reserved for real/imaginary parts. allow_missing_keys (bool) – don’t raise exception if key is missing. data (Mapping[Hashable, Union[ndarray, Tensor]]) – is a dictionary containing (key,value) pairs from the loaded dataset Dict[Hashable, Tensor] the new data dictionary Dictionary-based wrapper of monai.apps.reconstruction.transforms.array.EquispacedKspaceMask. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.MapTransform center_fractions (Sequence[float]) – Fraction of low-frequency columns to be retained. If multiple values are provided, then one of these numbers is chosen uniformly each time. accelerations (Sequence[float]) – Amount of under-sampling. This should have the same length as center_fractions. If multiple values are provided, then one of these is chosen uniformly each time. spatial_dims (int) – Number of spatial dims (e.g., it’s 2 for a 2D data; it’s also 2 for pseudo-3D datasets like the fastMRI dataset). The last spatial dim is selected for sampling. For the fastMRI dataset, k-space has the form (…,num_slices,num_coils,H,W) and sampling is done along W. For a general 3D data with the shape (…,num_coils,H,W,D), sampling is done along D. is_complex (bool) – if True, then the last dimension will be reserved for real/imaginary parts. allow_missing_keys (bool) – don’t raise exception if key is missing. data (Mapping[Hashable, Union[ndarray, Tensor]]) – is a dictionary containing (key,value) pairs from the loaded dataset Dict[Hashable, Tensor] the new data dictionary Moves keys from meta to data. It is useful when a dataset of paired samples is loaded and certain keys should be moved from meta to data. keys (Union[Collection[Hashable], Hashable]) – keys to be transferred from meta to data meta_key (str) – the meta key where all the meta-data is stored allow_missing_keys (bool) – don’t raise exception if key is missing Example When the fastMRI dataset is loaded, “kspace” is stored in the data dictionary, but the ground-truth image with the key “reconstruction_rss” is stored in the meta data. In this case, ExtractDataKeyFromMetaKeyd moves “reconstruction_rss” to data. data (Mapping[Hashable, Union[ndarray, Tensor]]) – is a dictionary containing (key,value) pairs from the loaded dataset Dict[Hashable, Tensor] the new data dictionary Dictionary-based wrapper of monai.transforms.SpatialCrop. This is similar to monai.transforms.SpatialCropd which is a general purpose cropper to produce sub-volume region of interest (ROI). Their difference is that this transform does cropping according to a reference image. If a dimension of the expected ROI size is larger than the input image size, will not crop that dimension. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform ref_key (str) – key of the item to be used to crop items of “keys” allow_missing_keys (bool) – don’t raise exception if key is missing. Example In an image reconstruction task, let keys=[“image”] and ref_key=[“target”]. Also, let data be the data dictionary. Then, ReferenceBasedSpatialCropd center-crops data[“image”] based on the spatial size of data[“target”] by calling monai.transforms.SpatialCrop. This transform can support to crop ND spatial (channel-first) data. It also supports pseudo ND spatial data (e.g., (C,H,W) is a pseudo-3D data point where C is the number of slices) data (Mapping[Hashable, Tensor]) – is a dictionary containing (key,value) pairs from the loaded dataset Dict[Hashable, Tensor] the new data dictionary Dictionary-based wrapper of monai.transforms.NormalizeIntensity. This is similar to monai.transforms.NormalizeIntensityd and can normalize non-zero values or the entire image. The difference is that this transform does normalization according to a reference image. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.MapTransform ref_key (str) – key of the item to be used to normalize items of “keys” subtrahend (Union[ndarray, Tensor, None]) – the amount to subtract by (usually the mean) divisor

(Union[ndarray, Tensor, None]) – the amount to divide by (usually the standard deviation) nonzero (bool) – whether only normalize non-zero values. channel_wise (bool) – if True, calculate on each channel separately, otherwise, calculate on the entire image directly. default to False. dtype (Union[dtype, type, str, None]) – output data type, if None, same as input image. defaults to float32. allow_missing_keys (bool) – don't raise exception if key is missing. Example In an image reconstruction task, let keys=["image", "target"] and ref_key=["image"]. Also, let data be the data dictionary. Then, ReferenceBasedNormalizeIntensityd normalizes data["target"] and data["image"] based on the mean-std of data["image"] by calling monai.transforms.NormalizeIntensity. This transform can support to normalize ND spatial (channel-first) data. It also supports pseudo ND spatial data (e.g., (C,H,W) is a pseudo-3D data point where C is the number of slices) data (Mapping[Hashable, Union[ndarray, Tensor]]) – is a dictionary containing (key,value) pairs from the loaded dataset Dict[Hashable, Union[ndarray, Tensor]] the new data dictionary

Utility (Dict)#

Dictionary-based wrapper of monai.transforms.Identity. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform allow_missing_keys (bool) – don't raise exception if key is missing. Dictionary-based wrapper of monai.transforms.AsChannelFirst. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform channel_dim (int) – which dimension of input image is the channel, default is the last dimension. allow_missing_keys (bool) – don't raise exception if key is missing. Dictionary-based wrapper of monai.transforms.AsChannelLast. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform channel_dim (int) – which dimension of input image is the channel, default is the first dimension. allow_missing_keys (bool) – don't raise exception if key is missing. Dictionary-based wrapper of monai.transforms.AddChannel. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform allow_missing_keys (bool) – don't raise exception if key is missing. Dictionary-based wrapper of monai.transforms.EnsureChannelFirst. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform strict_check (bool) – whether to raise an error when the meta information is insufficient. allow_missing_keys (bool) – don't raise exception if key is missing. channel_dim – This argument can be used to specify the original channel dimension (integer) of the input array. It overrides the original_channel_dim from provided MetaTensor input. If the input array doesn't have a channel dim, this value should be 'no_channel'. If this is set to None, this class relies on img or meta_dict to provide the channel dimension. Dictionary-based wrapper of monai.transforms.RepeatChannel. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example:

AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform repeats (int) – the number of repetitions for each element. allow_missing_keys (bool) – don't raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Union[Dict[Hashable, Tensor], List[Dict[Hashable, Tensor]]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform output_postfixes (Optional[Sequence[str]]) – the postfixes to construct keys to store split data. for example: if the key of input data is pred and split 2 classes, the output data keys will be: pred_(output_postfixes[0]), pred_(output_postfixes[1]) if None, using the index number: pred_0, pred_1, … pred_N. dim (int) – which dimension of input image is the channel, default to 0. keepdim (bool) – if True, output will have singleton in the split dimension. If False, this dimension will be squeezed. update_meta (bool) – if True, copy [key]_meta_dict for each output and update affine to reflect the cropped image list_output (bool) – it True, the output will be a list of dictionaries with the same keys as original. allow_missing_keys (bool) – don't raise exception if key is missing. Dictionary-based wrapper of monai.transforms.SplitChannel. All the input specified by keys should be split into same count of data. Dictionary-based wrapper of monai.transforms.CastToType. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform dtype (Union[Sequence[Union[dtype, type, str, None, dtype]], dtype, type, str, None, dtype]) – convert image to this data type, default is np.float32. it also can be a sequence of dtypes or torch.dtype, each element corresponds to a key in keys. allow_missing_keys (bool) – don't raise exception if key is missing. Dictionary-based wrapper of monai.transforms.ToTensor. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a

Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform dtype (Optional[dtype]) – target data content type to convert, for example: torch.float, etc. device (Optional[device]) – specify the target device to put the Tensor data. wrap_sequence (bool) – if False, then lists will recursively call this function, default to True. E.g., if False, [1, 2] -> [tensor(1), tensor(2)], if True, then [1, 2] -> tensor([1, 2]). track_meta (Optional[bool]) – if True convert to MetaTensor, otherwise to Pytorch Tensor, if None behave according to return value of py:func:monai.data.meta_obj.get_track_meta. allow_missing_keys (bool) – don't raise exception if key is missing. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] Dictionary-based wrapper of monai.transforms.ToNumpy. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Any] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform dtype (Union[dtype, type, str, None]) – target data type when converting to numpy array. wrap_sequence (bool) – if False, then lists will recursively call this function, default to True. E.g., if False, [1, 2] -> [array(1), array(2)], if True, then [1, 2] -> array([1, 2]). allow_missing_keys (bool) – don't raise exception if key is missing. Converts the input image (in the form of NumPy array or PyTorch Tensor) to PIL image Apply the transform to img. Dictionary-based wrapper of monai.transforms.ToCupy. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform dtype (Optional[dtype]) – data type specifier. It is inferred from the input by default. if not None, must be an argument of numpy.dtype, for more details: https://docs.cupy.dev/en/stable/reference/generated/cupy.array.html. wrap_sequence (bool) – if False, then lists will recursively call this function, default to True. E.g., if False, [1, 2] -> [array(1), array(2)], if True, then [1, 2] -> array([1, 2]). allow_missing_keys (bool) – don't raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Dictionary-based wrapper of monai.transforms.ToNumpy. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Any] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform allow_missing_keys (bool) – don’t raise exception if key is missing. Delete specified items from data dictionary to release memory. It will remove the key-values and copy the others to construct a new dictionary. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to delete, can be “A{sep}B{sep}C” to delete key C in nested dictionary, C can be regular expression. See also: monai.transforms.compose.MapTransform sep (str) – the separator tag to define nested dictionary keys, default to “.”. use_re (Union[Sequence[bool], bool]) – whether the specified key is a regular expression, it also can be a list of bool values, mapping them to keys. Select only specified items from data dictionary to release memory. It will copy the selected key-values and construct a new dictionary. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. An updated dictionary version of data by applying the transform. If an item is dictionary, it flatten the item by moving the sub-items (defined by sub-keys) to the top level. {“pred”: {“a”: …, “b”, … }} –> {“a”: …, “b”, … } keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be flatten sub_keys (Union[Collection[Hashable], Hashable, None]) – the sub-keys of items to be flatten. If not provided all the sub-keys are flattened. delete_keys (bool) – whether to delete the key of the items that their sub-keys are flattened. Default to True. prefix (Optional[str]) – optional prefix to be added to the sub-keys when moving to the top level. By default no prefix will be added. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. An updated dictionary version of data by applying the transform. Dictionary-based wrapper of monai.transforms.Transpose. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Any] Dictionary-based wrapper of monai.transforms.SqueezeDim. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform dim (int) – dimension to be squeezed. Default: 0 (the first dimension) update_meta (bool) – whether to update the meta info if the input is a metatensor. Default is True. allow_missing_keys (bool) – don’t raise exception if key is missing. Dictionary-based wrapper of monai.transforms.DataStats. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method.

Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform prefix (Union[Sequence[str], str]) – will be printed in format: “{prefix} statistics”. it also can be a sequence of string, each element corresponds to a key in keys. data_type (Union[Sequence[bool], bool]) – whether to show the type of input data. it also can be a sequence of bool, each element corresponds to a key in keys. data_shape (Union[Sequence[bool], bool]) – whether to show the shape of input data. it also can be a sequence of bool, each element corresponds to a key in keys. value_range (Union[Sequence[bool], bool]) – whether to show the value range of input data. it also can be a sequence of bool, each element corresponds to a key in keys. data_value (Union[Sequence[bool], bool]) – whether to show the raw value of input data. it also can be a sequence of bool, each element corresponds to a key in keys. a typical example is to print some properties of Nifti image: affine, pixdim, etc. additional_info (Union[Sequence[Callable], Callable, None]) – user can define callable function to extract additional info from input data. it also can be a sequence of string, each element corresponds to a key in keys. name (str) – identifier of logging.logger to use, defaulting to “DataStats”. allow_missing_keys (bool) – don’t raise exception if key is missing. Dictionary-based wrapper of monai.transforms.SimulateDelay. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform delay_time (Union[Sequence[float], float]) – The minimum amount of time, in fractions of seconds, to accomplish this identity task. It also can be a sequence of string, each element corresponds to a key in keys. allow_missing_keys (bool) – don’t raise exception if key is missing. Copy specified items from data dictionary and save with different key names. It can copy several items together and copy several times. KeyError – When a key in self.names already exists in data. Dict[Hashable, Union[ndarray, Tensor]] keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform times (int) – expected copy times, for example, if keys is “img”, times is 3, it will add 3 copies of “img” data to the dictionary, default to 1. names (Union[Collection[Hashable], Hashable, None]) – the names corresponding to the newly copied data, the length should match len(keys) x times. for example, if keys is [“img”, “seg”] and times is 2, names can be: [“img_1”, “seg_1”, “img_2”, “seg_2”]. if None, use “{key}_{index}” as key for copy times N, index from 0 to N-1. allow_missing_keys (bool) – don’t raise exception if key is missing. ValueError – When times is nonpositive. ValueError – When len(names) is not len(keys) * times. Incompatible values. Concatenate specified items from data dictionary together on the first dim to construct a big array. Expect all the items are numpy array or PyTorch Tensor or MetaTensor. Return the first input’s meta information when items are MetaTensor. TypeError – When items in data differ in type. TypeError – When the item type is not in Union[numpy.ndarray, torch.Tensor, MetaTensor]. Dict[Hashable, Union[ndarray, Tensor]] keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be concatenated together. See also: monai.transforms.compose.MapTransform name (str) – the name corresponding to the key to store the concatenated data. dim (int) – on which dimension to concatenate the items, default is 0. allow_missing_keys (bool) – don’t raise exception if key is missing. Dictionary-based wrapper of monai.transforms.Lambda. For example: keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform func (Union[Sequence[Callable], Callable]) – Lambda/function to be applied. It also can be a sequence of Callable, each element corresponds to a key in keys. inv_func (Union[Sequence[Callable], Callable]) – Lambda/function of inverse operation if want to invert transforms, default to lambda x: x. It also can be a sequence of Callable, each element corresponds to a key in keys. overwrite (Union[Sequence[bool], bool, Sequence[str], str]) – whether to overwrite the original data in the input dictionary with lambda function output. it can be bool or str, when setting to str, it will create a new key for the output and keep the value of key intact. default to True. it also can be a sequence of bool or str, each element corresponds to a key in keys. allow_missing_keys (bool) – don’t raise exception if key is missing. image’s original size. If need these complicated information, please write a new InvertibleTransform directly. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] An updated dictionary version of data by applying the transform. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Randomizable version monai.transforms.Lambdad, the input func may contain random logic, or randomly execute the function based on prob. so CacheDataset will not execute it and cache the results. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform func (Union[Sequence[Callable], Callable]) – Lambda/function to be applied. It also can be a sequence of Callable, each element corresponds to a key in keys. inv_func (Union[Sequence[Callable], Callable]) – Lambda/function of inverse operation if want to invert transforms, default to lambda x: x. It also can be a sequence of Callable, each element corresponds to a key in keys. overwrite (Union[Sequence[bool], bool]) – whether to overwrite the original data in the input dictionary with lambda function output. default to True. it also can be a sequence of bool, each element corresponds to a key in keys. prob (float) – probability of executing the random function, default to 1.0, with 100% probability to execute. note that all the data specified by keys will share the same random probability to execute or not. allow_missing_keys (bool) – don’t raise exception if key is missing. For more details, please check monai.transforms.Lambdad. image’s original size. If need these complicated information, please write a new InvertibleTransform directly. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. An updated dictionary version of data by applying the transform. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] Dictionary-based wrapper of monai.transforms.RemoveRepeatedChannel. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform repeats (int) – the number of repetitions for each element. allow_missing_keys (bool) – don't raise exception if key is missing. Dictionary-based wrapper of monai.transforms.LabelToMask. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform select_labels (Union[Sequence[int], int]) – labels to generate mask from. for 1 channel label, the select_labels is the expected label values, like: [1, 2, 3]. for One-Hot format label, the select_labels is the expected channel indices. merge_channels (bool) – whether to use np.any() to merge the result on channel dim. if yes, will return a single channel mask with binary data. allow_missing_keys (bool) – don't raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Dictionary-based wrapper of monai.transforms.FgBgToIndices. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform fg_postfix (str) – postfix to save the computed foreground indices in dict. for example, if computed on label and postfix = "_fg_indices", the key will be label_fg_indices. bg_postfix (str) – postfix to save the computed background indices in dict. for example, if computed on label and postfix = "_bg_indices", the key will be label_bg_indices. image_key (Optional[str]) – if image_key is not None, use label == 0 & image > image_threshold to determine the negative sample(background). so the output items will not map to all the voxels in the label. image_threshold (float) – if enabled image_key, use image > image_threshold to determine the valid image content area and select background only in this area.

output_shape (Optional[Sequence[int]]) – expected shape of output indices. if not None, unravel indices to specified shape. allow_missing_keys (bool) – don't raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Dictionary-based wrapper of monai.transforms.ClassesToIndices. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform indices_postfix (str) – postfix to save the computed indices of all classes in dict. for example, if computed on label and postfix = "_cls_indices", the key will be label_cls_indices. num_classes (Optional[int]) – number of classes for argmax label, not necessary for One-Hot label. image_key (Optional[str]) – if image_key is not None, use image > image_threshold to define valid region, and only select the indices within the valid region. image_threshold (float) – if enabled image_key, use image > image_threshold to determine the valid image content area and select only the indices of classes in this area. output_shape (Optional[Sequence[int]]) – expected shape of output indices. if not None, unravel indices to specified shape. allow_missing_keys (bool) – don't raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. An updated dictionary version of data by applying the transform. Dictionary-based wrapper of monai.transforms.ConvertToMultiChannelBasedOnBratsClasses. Convert labels to multi channels based on brats18 classes: label 1 is the necrotic and non-enhancing tumor core label 2 is the peritumoral edema label 4 is the GD-enhancing tumor The possible classes are TC (Tumor core), WT (Whole tumor) and ET (Enhancing tumor). data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Dictionary-based wrapper of monai.transforms.AddExtremePointsChannel. keys

(Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform label_key (str) – key to label source to get the extreme points. background (int) – Class index of background label, defaults to 0. pert (float) – Random perturbation amount to add to the points, defaults to 0.0. sigma (Union[Sequence[float], float, Sequence[Tensor], Tensor]) – if a list of values, must match the count of spatial dimensions of input data, and apply every value in the list to 1 spatial dimension. if only 1 value provided, use it for all spatial dimensions. rescale_min (float) – minimum value of output data. rescale_max (float) – maximum value of output data. allow_missing_keys (bool) – don't raise exception if key is missing. Call self as a function. Dict[Hashable, Union[ndarray, Tensor]] Within this method, self.R should be used, instead of np.random, to introduce random factors. all self.R calls happen here so that we have a better chance to identify errors of sync the random state. This method can generate the random factors based on properties of the input data. NotImplementedError – When the subclass does not override this method. None Dictionary-based wrapper of monai.transforms.TorchVision for non-randomized transforms. For randomized transforms of TorchVision use monai.transforms.RandTorchVisiond. Note As most of the TorchVision transforms only work for PIL image and PyTorch Tensor, this transform expects input data to be dict of PyTorch Tensors, users can easily call ToTensord transform to convert Numpy to Tensor. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform name (str) – The transform name in TorchVision package. allow_missing_keys (bool) – don't raise exception if key is missing. args – parameters for the TorchVision transform. kwargs – parameters for the TorchVision transform. Dictionary-based wrapper of monai.transforms.TorchVision for randomized transforms. For deterministic non-randomized transforms of TorchVision use monai.transforms.TorchVisiond. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform name (str) – The transform name in TorchVision package. allow_missing_keys (bool) – don't raise exception if key is missing. args – parameters for the TorchVision transform. kwargs – parameters for the TorchVision transform. Note As most of the TorchVision transforms only work for PIL image and PyTorch Tensor, this transform expects input data to be dict of PyTorch Tensors. Users should call ToTensord transform first to convert Numpy to Tensor. This class inherits the Randomizable purely to prevent any dataset caching to skip the transform computation. If the random factor of the underlying torchvision transform is not derived from self.R, the results may not be deterministic. See Also: monai.transforms.Randomizable. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Dictionary-based wrapper of monai.transforms.MapLabelValue. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform orig_labels (Sequence) – original labels that map to others. target_labels (Sequence) – expected label values, 1: 1 map to the orig_labels. dtype (Union[dtype, type, str, None]) – convert the output data to dtype, default to float32. allow_missing_keys (bool) – don’t raise exception if key is missing. Dictionary-based wrapper of monai.transforms.EnsureType. Ensure the input data to be a PyTorch Tensor or numpy array, support: numpy array, PyTorch Tensor, float, int, bool, string and object keep the original. If passing a dictionary, list or tuple, still return dictionary, list or tuple and recursively convert every item to the expected data type if wrap_sequence=False. Note: Currently, we only convert tensor data to numpy array or scalar number in the inverse operation. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform data_type (str) – target data type to convert, should be “tensor” or “numpy”. dtype (Union[dtype, type, str, None, dtype]) – target data content type to convert, for example: np.float32, torch.float, etc. device (Optional[device]) – for Tensor data type, specify the target device. wrap_sequence (bool) – if False, then lists will recursively call this function, default to True. E.g., if False, [1, 2] -> [tensor(1), tensor(2)], if True, then [1, 2] -> tensor([1, 2]). track_meta (Optional[bool]) – whether to convert to MetaTensor when data_type is “tensor”. If False, the output data type will be torch.Tensor. Default to the return value of get_track_meta. allow_missing_keys (bool) – don’t raise exception if key is missing. Dictionary-based wrapper of monai.transforms.IntensityStats. Compute statistics for the intensity values of input image and store into the metadata dictionary. For example: if ops=[lambda x: np.mean(x), “max”] and key_prefix=”orig”, may generate below stats: {“orig_custom_0”: 1.5, “orig_max”: 3.0}. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform ops (Sequence[Union[str, Callable]]) – expected operations to compute statistics for the intensity. if a string, will map to the predefined operations, supported: [“mean”, “median”, “max”, “min”, “std”] mapping to np.nanmean, np.nanmedian, np.nanmax, np.nanmin, np.nanstd. if a callable function, will execute the function on input image. key_prefix (str) – the prefix to combine with ops name to generate the key to store the results in the metadata dictionary. if some ops are callable functions, will use “{key_prefix}_custom_{index}” as the key, where index counts from 0. mask_keys (Union[Collection[Hashable], Hashable, None]) – if not None, specify the mask array for the image to extract only the interested area to compute statistics, mask must have the same shape as the image. it should be a sequence of strings or None, map to the keys. channel_wise (bool) – whether to compute statistics for every channel of input image separately. if True, return a list of values for every operation, default to False. meta_keys (Union[Collection[Hashable], Hashable, None]) – explicitly indicate the key of the corresponding metadata dictionary. used to store the computed statistics to the meta dict. for example, for data with key image, the metadata by default is in image_meta_dict. the metadata is a dictionary object which contains: filename, original_shape, etc. it can be a sequence of string, map to the keys. if None, will try to construct meta_keys by key_{meta_key_postfix}. meta_key_postfix (str) – if meta_keys is None, use key_{postfix} to fetch the metadata according to the key data, default is meta_dict, the metadata is a dictionary object. used to store the computed statistics to the meta dict. allow_missing_keys (bool) – don’t raise exception if key is missing. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannelId expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Dictionary-based wrapper of monai.transforms.ToDevice. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChannelId expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Tensor] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform device (Union[device, str]) – target device to move the Tensor, for example: “cuda:1”. allow_missing_keys (bool) – don’t raise exception if key is missing. kwargs – other args for the PyTorch Tensor.to() API, for more details: https://pytorch.org/docs/stable/generated/torch.Tensor.to.html. Dictionary-based wrapper of monai.transforms.CuCIM for non-randomized transforms. For randomized transforms of CuCIM use monai.transforms.RandCuCIMd. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform name (str) – The transform name in CuCIM package. allow_missing_keys (bool) – don't raise exception if key is missing. args – parameters for the CuCIM transform. kwargs – parameters for the CuCIM transform. Note CuCIM transforms only work with CuPy arrays, this transform expects input data to be cupy.ndarray. Users can call ToCuPy transform to convert a numpy array or torch tensor to cupy array. data – Dict[Hashable, cupy.ndarray] Dict[Hashable, cupy.ndarray] Dictionary-based wrapper of monai.transforms.CuCIM for randomized transforms. For deterministic non-randomized transforms of CuCIM use monai.transforms.CuCIMd. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform name (str) – The transform name in CuCIM package. allow_missing_keys (bool) – don't raise exception if key is missing. args – parameters for the CuCIM transform. kwargs – parameters for the CuCIM transform. Note CuCIM transform only work with CuPy arrays, so this transform expects input data to be cupy.ndarray. Users should call ToCuPy transform first to convert a numpy array or torch tensor to cupy array. This class inherits the Randomizable purely to prevent any dataset caching to skip the transform computation. If the random factor of the underlying cuCIM transform is not derived from self.R, the results may not be deterministic. See Also: monai.transforms.Randomizable. data – Dict[Hashable, cupy.ndarray] Dict[Hashable, cupy.ndarray] Dictionary-based wrapper of monai.transforms.AddCoordinateChannels. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform spatial_dims (Sequence[int]) – the spatial dimensions that are to have their coordinates encoded in a channel and appended to the input image. E.g., (0, 1, 2) represents H, W, D dims and append three channels to the input image, encoding the coordinates of the input's three spatial dimensions. allow_missing_keys (bool) – don't raise exception if key is missing. Deprecated since version 0.8.0: spatial_channels is deprecated, use spatial_dims instead. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform.

MetaTensor#

Dictionary-based transform to convert a dictionary to MetaTensor. If input is {"a": torch.Tensor, "a_meta_dict": dict, "b": …}, then output will have the form {"a": MetaTensor, "b": MetaTensor}. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] Dictionary-based transform to convert MetaTensor to a dictionary. If input is {“a”: MetaTensor, “b”: MetaTensor}, then output will have the form {“a”: torch.Tensor, “a_meta_dict”: dict, “a_transforms”: list, “b”: …}. data often comes from an iteration over an iterable, such as torch.utils.data.Dataset. To simplify the input validations, this method assumes: data is a Python dictionary, data[key] is a Numpy ndarray, PyTorch Tensor or string, where key is an element of self.keys, the data shape can be: string data without shape, LoadImaged transform expects file paths, most of the pre-/post-processing transforms expect: (num_channels, spatial_dim_1[, spatial_dim_2, ...]), except for example: AddChanneld expects (spatial_dim_1[, spatial_dim_2, …]) and AsChannelFirstd expects (spatial_dim_1[, spatial_dim_2, …], num_channels) the channel dimension is often not omitted even if number of channels is one. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]] An updated dictionary version of data by applying the transform. keys (Union[Collection[Hashable], Hashable]) – keys of the corresponding items to be transformed. See also: monai.transforms.compose.MapTransform data_type (Union[Sequence[str], str]) – target data type to convert, should be “tensor” or “numpy”. allow_missing_keys (bool) – don’t raise exception if key is missing. Inverse of __call__. NotImplementedError – When the subclass does not override this method. Dict[Hashable, Union[ndarray, Tensor]]

How to use the adaptor function#

The key to using ‘adaptor’ lies in understanding the function that want to adapt. The ‘inputs’ and ‘outputs’ parameters take either strings, lists/tuples of strings or a dictionary mapping strings, depending on call signature of the function being called. The adaptor function is written to minimise the cognitive load on the caller. There should be a minimal number of cases where the caller has to set anything on the input parameter, and for functions that return a single value, it is only necessary to name the dictionary keyword to which that value is assigned. outputs can take either a string, a list/tuple of string or a dict of string to string, depending on what the transform being adapted returns: If the transform returns a single argument, then outputs can be supplied a string that indicates what key to assign the return value to in the dictionary If the transform returns a list/tuple of values, then outputs can be supplied a list/tuple of the same length. The strings in outputs map the return value at the corresponding position to a key in the dictionary If the transform returns a dictionary of values, then outputs must be supplied a dictionary that maps keys in the function’s return dictionary to the dictionary being passed between functions Note, the caller is free to use a more complex way of specifying the outputs parameter than is required. The following are synonymous and will be treated identically: inputs can usually be omitted when using adaptor. It is only required when a the function’s parameter names do not match the names in the dictionary that is used to chain transform calls. Inputs: dictionary in: None | Name maps params in (match): None | Name list | Name maps params in (mismatch): Name maps params & **kwargs (match) : None | Name maps params & **kwargs (mismatch) : Name maps Outputs: dictionary out: None | Name maps list/tuple out: list/tuple variable out: string

Utilities#

Helper class storing Fourier mappings Applies inverse shift and fourier transform. Only the spatial dimensions are transformed. k (Union[ndarray, Tensor]) – K-space data. spatial_dims (int) – Number of spatial dimensions. Tensor in image space. x Applies fourier transform and shifts the zero-frequency component to the center of the spectrum. Only the spatial dimensions get transformed. x (Union[ndarray, Tensor]) – Image to transform. spatial_dims (int) – Number of spatial dimensions. k: K-space data. Union[ndarray, Tensor] Temporarily set all MapTransforms to not throw an error if keys are missing. After, revert to original states. transform (Union[MapTransform, Compose, Tuple[MapTransform], Tuple[Compose]]) – either MapTransform or a Compose Example: Adds hook before or after a func call. If mode is "pre", the wrapper will call hook then func. If the mode is "post", the wrapper will call func then hook. Check boundaries for Signal transforms None Compute the target spatial size which should be divisible by k. spatial_shape (Sequence[int]) – original spatial shape. k (Union[Sequence[int], int]) – the target k for each spatial dimension. if k is negative or 0, the original size is preserved. if k is an int, the same k be applied to all the input spatial dimensions. Recursively change the interpolation mode in the applied operation stacks, default to "nearest". See also: monai.transform.inverse.InvertibleTransform trans_info – applied operation stack, tracking the previously applied invertible transform. mode (str) – target interpolation mode to convert, default to "nearest" as it's usually used to save the mode output. align_corners (Optional[bool]) – target align corner value in PyTorch interpolation API, need to align with the mode. Utility to convert padding mode between numpy array and PyTorch Tensor. dst (Union[ndarray, Tensor]) – target data to convert padding mode for, should be numpy array or PyTorch Tensor. mode (Optional[str]) – current padding mode. Check and ensure the numpy array or PyTorch Tensor in data to be contiguous in memory. data (Union[ndarray, Tensor, str, bytes, Mapping, Sequence[Any]]) – input data to convert, will recursively convert the numpy array or PyTorch Tensor in dict and sequence. kwargs – if x is PyTorch Tensor, additional args for torch.contiguous, more details: https://pytorch.org/docs/stable/generated/torch.Tensor.contiguous.html#torch.Tensor.contiguous. Union[ndarray, Tensor, Mapping, Sequence[Any]] Calculate the slices to copy a sliced area of array in src_shape into array in dest_shape. The area has dimensions dims (use 0 or None to copy everything in that dimension), the source area is centered at srccenter index in src and copied into area centered at destcenter in dest. The dimensions of the copied area will be clipped to fit within the source and destination arrays so a smaller area may be copied than expected. Return value is the tuples of slice objects indexing the copied area in src, and those indexing the copy area in dest. Example Tuple[Tuple[slice, …], Tuple[slice, …]] control grid with two additional point in each direction compute a spatial_size mesh. when homogeneous=True, the output shape is (N+1, dim_size_1, dim_size_2, …, dim_size_N) when homogeneous=False, the output shape is (N, dim_size_1, dim_size_2, …, dim_size_N) spatial_size (Sequence[int]) – spatial size of the grid. spacing (Optional[Sequence[float]]) – same len as spatial_size, defaults to 1.0 (dense grid). homogeneous (bool) – whether to make homogeneous coordinates. dtype (Union[dtype, type, str, None, dtype]) – output grid data type, defaults to float. device (Optional[device]) – device to compute and store the output (when the backend is "torch"). backend – APIs to use, numpy or torch. Union[ndarray, Tensor] create a 2D or 3D rotation matrix spatial_dims (int) – {2, 3} spatial rank radians (Union[Sequence[float], float]) – rotation radians when spatial_dims == 3, the radians sequence corresponds to rotation in the 1st, 2nd, and 3rd dim respectively. device (Optional[device]) – device to compute and store the output (when the backend is “torch”). backend (str) – APIs to use, numpy or torch. ValueError – When radians is empty. ValueError – When spatial_dims is not one of [2, 3]. Union[ndarray, Tensor] create a scaling matrix spatial_dims (int) – spatial rank scaling_factor (Union[Sequence[float], float]) – scaling factors for every spatial dim, defaults to 1. device (Optional[device]) – device to compute and store the output (when the backend is “torch”). backend – APIs to use, numpy or torch. Union[ndarray, Tensor] create a shearing matrix spatial_dims (int) – spatial rank coefs (Union[Sequence[float], float]) – shearing factors, a tuple of 2 floats for 2D, a tuple of 6 floats for 3D), take a 3D affine as example: [[1.0, coefs[0], coefs[1], 0.0], [coefs[2], 1.0, coefs[3], 0.0], [coefs[4], coefs[5], 1.0, 0.0], [0.0, 0.0, 0.0, 1.0],] shearing factors, a tuple of 2 floats for 2D, a tuple of 6 floats for 3D), take a 3D affine as example: device (Optional[device]) – device to compute and store the output (when the backend is “torch”). backend – APIs to use, numpy or torch. NotImplementedError – When spatial_dims is not one of [2, 3]. Union[ndarray, Tensor] create a translation matrix spatial_dims (int) – spatial rank shift (Union[Sequence[float], float]) – translate pixel/voxel for every spatial dim, defaults to 0. device (Optional[device]) – device to compute and store the output (when the backend is “torch”). backend – APIs to use, numpy or torch. Union[ndarray, Tensor] Utility to equalize input image based on the histogram. If skimage installed, will leverage skimage.exposure.histogram, otherwise, use np.histogram instead. img (ndarray) – input image to equalize. mask (Optional[ndarray]) – if provided, must be ndarray of bools or 0s and 1s, and same shape as image. only points at which mask==True are used for the equalization. num_bins (int) – number of the bins to use in histogram, default to 256. for more details: https://numpy.org/doc/stable/reference/generated/numpy.histogram.html. min (int) – the min value to normalize input image, default to 0. max (int) – the max value to normalize input image, default to 255. ndarray Please refer to monai.transforms.AddExtremePointsChannel for the usage. Applies a gaussian filter to the extreme points image. Then the pixel values in points image are rescaled to range [rescale_min, rescale_max]. points (List[Tuple[int, …]]) – Extreme points of the object/organ. label (Union[ndarray, Tensor]) – label image to get extreme points from. Shape must be (1, spatial_dim1, [, spatial_dim2, …]). Doesn’t support one-hot labels. sigma (Union[Sequence[float], float, Sequence[Tensor], Tensor]) – if a list of values, must match the count of spatial dimensions of input data, and apply every value in the list to 1 spatial dimension. if only 1 value provided, use it for all spatial dimensions. rescale_min (float) – minimum value of output data. rescale_max (float) – maximum value of output data. Tensor Fill the holes in the provided image. The label 0 will be treated as background and the enclosed holes will be set to the neighboring class label. What is considered to be an enclosed hole is defined by the connectivity. Holes on the edge are always considered to be open (not enclosed). Note The performance of this method heavily depends on the number of labels. It is a bit faster if the list of applied_labels is provided. Limiting the number of applied_labels results in a big decrease in processing time. If the image is one-hot-encoded, then the applied_labels need to match the channel index. img_arr (ndarray) – numpy array of shape [C, spatial_dim1[, spatial_dim2, …]]. applied_labels (Optional[Iterable[int]]) – Labels for which to fill holes. Defaults to None, that is filling holes for all labels. connectivity (Optional[int]) – Maximum number of orthogonal hops to consider a pixel/voxel as a neighbor. Accepted values are ranging from 1 to input.ndim. Defaults to a full connectivity of input.ndim. ndarray numpy array of shape [C, spatial_dim1[, spatial_dim2, …]]. Generate valid sample locations based on the specified ratios of label classes. Valid: samples sitting entirely within image, expected input shape: [C, H, W, D] or [C, H, W] spatial_size (Union[Sequence[int], int]) – spatial size of the ROIs to be sampled. num_samples (int) – total sample centers to be generated. label_spatial_shape (Sequence[int]) – spatial shape of the original label data to unravel selected centers. indices (Sequence[Union[ndarray, Tensor]]) – sequence of pre-computed foreground indices of every class in 1 dimension. ratios (Optional[List[Union[float, int]]]) – ratios of every class in the label to generate crop centers, including background class. if None, every class will have the same ratio to generate crop centers. rand_state (Optional[RandomState]) – numpy randomState object to align with other modules. allow_smaller (bool) – if False, an exception will be raised if the image is smaller than the requested ROI in any dimension. If True, any smaller dimensions will be set to match the cropped size (i.e., no cropping in that dimension). List[List[int]] Generate valid sample locations based on the label with option for specifying foreground ratio Valid: samples sitting entirely within image, expected input shape: [C, H, W, D] or [C, H, W] spatial_size (Union[Sequence[int], int]) – spatial size of the ROIs to be sampled. num_samples (int) – total sample centers to be generated. pos_ratio (float) – ratio of total locations generated that have center being foreground. label_spatial_shape (Sequence[int]) – spatial shape of the original label data to unravel selected centers. fg_indices (Union[ndarray, Tensor]) – pre-computed foreground indices in 1 dimension. bg_indices (Union[ndarray, Tensor]) – pre-computed background indices in 1 dimension. rand_state (Optional[RandomState]) – numpy randomState object to align with other modules. allow_smaller (bool) – if False, an exception will be raised if the image is smaller than the requested ROI in any dimension. If True, any smaller dimensions will be set to match the cropped size (i.e., no cropping in that dimension). ValueError – When the proposed roi is larger than the image. ValueError – When the foreground and background indices lengths are 0. List[List[int]] Generate the spatial bounding box of foreground in the image with start-end positions (inclusive). Users can define arbitrary function to select expected foreground from the whole image or specified channels. And it can also add margin to every dim of the bounding box. The output format of the coordinates is: [1st_spatial_dim_start, 2nd_spatial_dim_start, …, Nth_spatial_dim_start], [1st_spatial_dim_end, 2nd_spatial_dim_end, …, Nth_spatial_dim_end] If allow_smaller, the bounding boxes edges are aligned with the input image edges. This function returns [0, 0, …], [0, 0, …] if there’s no positive intensity. img (Union[ndarray, Tensor]) – a “channel-first” image of shape (C, spatial_dim1[, spatial_dim2, …]) to generate bounding box from. select_fn (Callable) – function to select expected foreground, default is to select values > 0. channel_indices (Union[Iterable[int], int, None]) – if defined, select foreground only on the specified channels of image. if None, select foreground on the whole image. margin (Union[Sequence[int], int]) – add margin value to spatial dims of the bounding box, if only 1 value provided, use it for all dims. allow_smaller (bool) – when computing box size with margin, whether allow the image size to be smaller than box size, default to True. Tuple[List[int], List[int]] Generate extreme points from an image. These are used to generate initial segmentation for annotation models. An optional perturbation can be passed to simulate user clicks. img (Union[ndarray, Tensor]) – Image to generate extreme points from. Expected Shape is (spatial_dim1, [, spatial_dim2, ...]). rand_state (Optional[RandomState]) – np.random.RandomState object used to select random indices. background (int) – Value to be consider as background, defaults to 0. pert (float) – Random perturbation amount to add to the points, defaults to 0.0. List[Tuple[int, …]] A list of extreme points, its length is equal to 2 * spatial dimension of input image. The output format of the coordinates is: [1st_spatial_dim_min, 1st_spatial_dim_max, 2nd_spatial_dim_min, …, Nth_spatial_dim_max] A list of extreme points, its length is equal to 2 * spatial dimension of input image. The output format of the coordinates is: [1st_spatial_dim_min, 1st_spatial_dim_max, 2nd_spatial_dim_min, …, Nth_spatial_dim_max] ValueError – When the input image does not have any foreground pixel. Gets the largest connected component mask of an image. img (~NdarrayTensor) – Image to get largest connected component from. Shape is (spatial_dim1 [, spatial_dim2, …]) connectivity (Optional[int]) – Maximum number of orthogonal hops to consider a pixel/voxel as a neighbor. Accepted values are ranging from 1 to input.ndim. If None, a full connectivity of input.ndim is used. for more details: https://scikit-image.org/docs/dev/api/skimage.measure.html#skimage.measure.label. num_components (int) – The number of largest components to preserve. ~NdarrayTensor Get the number of times that the data need to be converted (e.g., numpy to torch). Conversions between different devices are also counted (e.g., CPU to GPU). transform (Compose) – composed transforms to be tested test_data (Any) – data to be used to count the number of conversions key (Optional[Hashable]) – if using dictionary transforms, this key will be used to check the number of conversions. int Get the backends of all MONAI transforms. Dictionary, where each key is a transform, and its corresponding values are a boolean list, stating whether that transform supports (1) torch.Tensor, and (2) np.ndarray as input without needing to convert. Get list of non-background labels in an image. img (Union[ndarray, Tensor]) – Image to be processed. Shape should be [C, W, H, [D]] with C=1 if not onehot else num_classes. is_onehot (bool) – Boolean as to whether input image is one-hotted. If one-hotted, only return channels with discard (Union[Iterable[int], int, None]) – Can be used to remove labels (e.g., background). Can be any value, sequence of values, or None (nothing is discarded). Set[int] Set of labels Returns the minimum and maximum indices of non-zero lines in axis 0 of img, followed by that for axis 1. Returns True if (x,y) is within the rectangle (margin, margin, maxx-margin, maxy-margin). bool Returns True if img is empty, that is its maximum value is not greater than its minimum. bool Returns a boolean version of img where the positive values are converted into True, the other values are False. Compute the foreground and background of input label data, return the indices after fattening. For example: label = np.array([[[0, 1, 1], [1, 0, 1], [1, 1, 0]]]) foreground indices = np.array([1, 2, 3, 5, 6, 7]) and background indices = np.array([0, 4, 8]) label (Union[ndarray, Tensor]) – use the label data to get the foreground/background information. image (Union[ndarray, Tensor, None]) – if image is not None, use label = 0 & image > image_threshold to define background. so the output items will not map to all the voxels in the label. image_threshold (float) – if enabled image, use image > image_threshold to determine the valid image content area and select background only in this area. Tuple[Union[ndarray, Tensor], Union[ndarray, Tensor]] Filter out indices of every class of the input label data, return the indices after fattening. It can handle both One-Hot format label and Argmax format label, must provide num_classes for Argmax label. For example: label = np.array([[[0, 1, 2], [2, 0, 1], [1, 2, 0]]]) and num_classes=3, will return a list which contains the indices of the 3 classes: [np.array([0, 4, 8]), np.array([1, 5, 6]), np.array([2, 3, 7])] label (Union[ndarray, Tensor]) – use the label data to get the indices of every class. num_classes (Optional[int]) – number of classes for argmax label, not necessary for One-Hot label. image (Union[ndarray, Tensor, None]) – if image is not None, only return the indices of every class that are within the valid region of the image (image > image_threshold). image_threshold (float) – if enabled image, use image > image_threshold to determine the valid image content area and select class indices only in this area. List[Union[ndarray, Tensor]] Utility to map the spatial axes to real axes in channel first/last shape. For example: If channel_first is True, and img has 3 spatial dims, map spatial axes to real axes as below: None -> [1, 2, 3] [0, 1] -> [1, 2] [0, -1] -> [1, -1] If channel_first is False, and img has 3 spatial dims, map spatial axes to real axes as below: None -> [0, 1, 2] [0, 1] ->

[0, 1] [0, -1] -> [0, -2] img_ndim (int) – dimension number of the target image. spatial_axes (Union[Sequence[int], int, None]) – spatial axes to be converted, default is None. The default None will convert to all the spatial axes of the image. If axis is negative it counts from the last to the first axis. If axis is a tuple of ints. channel_first (bool) – the image data is channel first or channel last, default to channel first. List[int] Prints a list of backends of all MONAI transforms. Returns True if a randomly chosen number is less than or equal to prob, by default this is a 50/50 chance. bool Use skimage.morphology.remove_small_objects to remove small objects from images. See: https://scikit-image.org/docs/dev/api/skimage.morphology.html#remove-small-objects. Data should be one-hotted. img (~NdarrayTensor) – image to process. Expected shape: C, H,W,[D]. Expected to only have singleton channel dimension, i.e., not be one-hotted. Converted to type int. min_size (int) – objects smaller than this size are removed. connectivity (int) – Maximum number of orthogonal hops to consider a pixel/voxel as a neighbor. Accepted values are ranging from 1 to input.ndim. If None, a full connectivity of input.ndim is used. For more details refer to linked scikit-image documentation. independent_channels (bool) – Whether to consider each channel independently. ~NdarrayTensor Rescale the values of numpy array arr to be from minv to maxv. If either minv or maxv is None, it returns (a - min_a) / (max_a - min_a). arr (Union[ndarray, Tensor]) – input array to rescale. minv (Optional[float]) – minimum value of target rescaled array. maxv (Optional[float]) – maxmum value of target rescaled array. dtype (Union[dtype, type, str, None, dtype]) – if not None, convert input array to dtype before computation. Union[ndarray, Tensor] Rescale the array arr to be between the minimum and maximum values of the type dtype. ndarray Rescale each array slice along the first dimension of arr independently. ndarray find MetaTensors in list or dict data and (in-place) set TraceKeys.ID to Tracekys.NONE. Resize img by cropping or expanding the image from the center. The resize_dims values are the output dimensions (or None to use original dimension of img). If a dimension is smaller than that of img then the result will be cropped and if larger padded with zeros, in both cases this is done relative to the center of img. The result is a new image with the specified dimensions and values from img copied into its center. Scale the affine matrix according to the new spatial size. affine – affine matrix to scale. spatial_size – original spatial size. new_spatial_size – new spatial size. centered (bool) – whether the scaling is with respect to the image center (True, default) or corner (False). Scaled affine matrix. Given the key, sync up between metatensor data_dict[key] and meta_dict data_dict[key_transforms/meta_dict]. t=True: the one with more applied_operations in metatensor vs meta_dict is the output, False: less is the output. Computes n_samples of random patch sampling locations, given the sampling weight map w and patch spatial_size. spatial_size (Union[int, Sequence[int]]) – length of each spatial dimension of the patch. w (Union[ndarray, Tensor]) – weight map, the weights must be non-negative. each element denotes a sampling weight of the spatial location. 0 indicates no sampling. The weight map shape is assumed (spatial_dim_0, spatial_dim_1, ..., spatial_dim_n). n_samples (int) – number of patch samples r_state (Optional[RandomState]) – a random state container List a list of n_samples N-D integers representing the spatial sampling location of patches. Returns True if the values within margin indices of the edges of img in dimensions 1 and 2 are 0. bool np.allclose with equivalent implementation for torch. bool np.any with equivalent implementation for torch. For pytorch, convert to boolean for compatibility with older versions. x (Union[ndarray, Tensor]) – input array/tensor. axis (Union[int, Sequence[int]]) – axis to perform any over. Union[ndarray, Tensor] Return a contiguous flattened array/tensor. np.ascontiguousarray with equivalent implementation for torch (contiguous). x (Union[~NdarrayTensor, ~T]) – array/tensor. kwargs – if x is PyTorch Tensor,

additional args for torch.contiguous, more details: https://pytorch.org/docs/stable/generated/torch.Tensor.contiguous.html. Union[ndarray, Tensor, ~T] np.clip with equivalent implementation for torch. Union[ndarray, Tensor] np.concatenate with equivalent implementation for torch (torch.cat). Union[ndarray, Tensor] np.cumsum with equivalent implementation for torch. a (Union[ndarray, Tensor]) – input data to compute cumsum. axis – expected axis to compute cumsum. kwargs – if a is PyTorch Tensor, additional args for torch.cumsum, more details: https://pytorch.org/docs/stable/generated/torch.cumsum.html. Union[ndarray, Tensor] np.floor_divide with equivalent implementation for torch. As of pt1.8, use torch.div(…, rounding_mode=”floor”), and before that, use torch.floor_divide. a (Union[ndarray, Tensor]) – first array/tensor b – scalar to divide by Union[ndarray, Tensor] Element-wise floor division between two arrays/tensors. np.in1d with equivalent implementation for torch. np.isfinite with equivalent implementation for torch. Union[ndarray, Tensor] np.isnan with equivalent implementation for torch. x (Union[ndarray, Tensor]) – array/tensor. Union[ndarray, Tensor] torch.max with equivalent implementation for numpy x (~NdarrayTensor) – array/tensor. ~NdarrayTensor the maximum of x. np.maximum with equivalent implementation for torch. a (Union[ndarray, Tensor]) – first array/tensor. b (Union[ndarray, Tensor]) – second array/tensor. Union[ndarray, Tensor] Element-wise maximum between two arrays/tensors. torch.mean with equivalent implementation for numpy x (~NdarrayTensor) – array/tensor. ~NdarrayTensor the mean of x torch.median with equivalent implementation for numpy x (~NdarrayTensor) – array/tensor. the median of x. ~NdarrayTensor torch.min with equivalent implementation for numpy x (~NdarrayTensor) – array/tensor. ~NdarrayTensor the minimum of x. torch.mode with equivalent implementation for numpy. x (~NdarrayTensor) – array/tensor. dim (int) – dimension along which to perform mode (referred to as axis by numpy). to_long (bool) – convert input to long before performing mode. ~NdarrayTensor moveaxis for pytorch and numpy Union[ndarray, Tensor] np.nonzero with equivalent implementation for torch. x (Union[ndarray, Tensor]) – array/tensor. Union[ndarray, Tensor] Index unravelled for given shape np.percentile with equivalent implementation for torch. Pytorch uses quantile. For more details please refer to: https://pytorch.org/docs/stable/generated/torch.quantile.html. https://numpy.org/doc/stable/reference/generated/numpy.percentile.html. x (Union[ndarray, Tensor]) – input data. q – percentile to compute (should in range 0 <= q <= 100). dim (Optional[int]) – the dim along which the percentiles are computed. default is to compute the percentile along a flattened version of the array. keepdim (bool) – whether the output data has dim retained or not. kwargs – if x is numpy array, additional args for np.percentile, more details: https://numpy.org/doc/stable/reference/generated/numpy.percentile.html. Union[ndarray, Tensor, float, int] Resulting value (scalar) np.ravel with equivalent implementation for torch. x (Union[ndarray, Tensor]) – array/tensor to ravel. Union[ndarray, Tensor] Return a contiguous flattened array/tensor. np.repeat with equivalent implementation for torch (repeat_interleave). a (Union[ndarray, Tensor]) – input data to repeat. repeats (int) – number of repetitions for each element, repeats is broadcast to fit the shape of the given axis. axis (Optional[int]) – axis along which to repeat values. kwargs – if a is PyTorch Tensor, additional args for torch.repeat_interleave, more details: https://pytorch.org/docs/stable/generated/torch.repeat_interleave.html. Union[ndarray, Tensor] np.searchsorted with equivalent implementation for torch. a (~NdarrayTensor) – numpy array or tensor, containing monotonically increasing sequence on the innermost dimension. v (Union[ndarray, Tensor]) – containing the search values. right – if False, return the first suitable location that is found, if True, return the last such index. sorter – if a is numpy array, optional array of integer indices that sort array a into ascending order. kwargs – if a is PyTorch Tensor, additional args for torch.searchsorted, more details: https://pytorch.org/docs/stable/generated/torch.searchsorted.html. ~NdarrayTensor np.stack with equivalent implementation for torch. x (Sequence[~NdarrayTensor]) – array/tensor. dim (int) – dimension along which to perform the stack (referred to as axis by numpy). ~NdarrayTensor torch.std with equivalent implementation for numpy x (~NdarrayTensor) – array/tensor. ~NdarrayTensor the standard deviation of x. torch.unique with equivalent implementation for numpy. x (~NdarrayTensor) – array/tensor. ~NdarrayTensor np.unravel_index with equivalent implementation for torch. idx – index to unravel. shape – shape of array/tensor. Union[ndarray, Tensor] Index unravelled for given shape Computing unravel coordinates from indices. idx – a sequence of indices to unravel. shape – shape of array/tensor. Union[ndarray, Tensor] Stacked indices unravelled for given shape Note that torch.where may convert y.dtype to x.dtype. Union[ndarray, Tensor] previous Model Bundle next Loss functions