Augmentation

Augmentation transforms generate different results every time they are called.

Base class

`RandomTransform`

```
class torchio.transforms.augmentation.RandomTransform(**kwargs)  [source]
```

Bases: `torchio.transforms.transform.Transform`

Base class for stochastic augmentation transforms.

PARAMETERS

****kwargs** – See `Transform` for additional keyword arguments.

Composition

`Compose`

```
class torchio.transforms.Compose(transforms:
    Sequence[torchio.transforms.transform.Transform], **kwargs)  [source]
```

Bases: `torchio.transforms.transform.Transform`

Compose several transforms together.

PARAMETERS

- **transforms** – Sequence of instances of `Transform`.
- ****kwargs** – See `Transform` for additional keyword arguments.

`OneOf`

```
class torchio.transforms.OneOf(transforms:
    Union[Dict[torchio.transforms.transform.Transform, float],
    Sequence[torchio.transforms.transform.Transform]], **kwargs)  [source]
```

Bases: `torchio.transforms.augmentation.random_transform.RandomTransform`

Apply only one of the given transforms.

PARAMETERS

- **transforms** – Dictionary with instances of `Transform` as keys and probabilities as values. Probabilities are normalized so they sum to one. If a sequence is given, the same probability will be assigned to each transform.
- ****kwargs** – See `Transform` for additional keyword arguments.

EXAMPLE

```
>>> import torchio as tio
>>> colin = tio.datasets.Colin27()
>>> transforms_dict = {
...     tio.RandomAffine(): 0.75,
...     tio.RandomElasticDeformation(): 0.25,
... }  # Using 3 and 1 as probabilities would have the same effect
>>> transform = tio.OneOf(transforms_dict)
>>> transformed = transform(colin)
```

Spatial

`RandomFlip`

```
class torchio.transforms.RandomFlip(axes: Union[int, Tuple[int, ...]] = 0,
    flip_probability: float = 0.5, **kwargs)  [source]
```

Bases: `torchio.transforms.augmentation.random_transform.RandomTransform`, `torchio.transforms.spatial_transform.SpatialTransform`

CONTENTS

Base class
RandomTransform
Composition
Compose
OneOf
Spatial
RandomFlip
RandomAffine
RandomElasticDeformation
RandomAnisotropy
Intensity
RandomMotion
RandomGhosting
RandomSpike
RandomBiasField
RandomBlur
RandomNoise
RandomSwap
RandomLabelsToImage
RandomGamma

Reverse the order of elements in an image along the given axes.

PARAMETERS

- **axes** – Index or tuple of indices of the spatial dimensions along which the image might be flipped. If they are integers, they must be in `(0, 1, 2)`. Anatomical labels may also be used, such as `'Left'`, `'Right'`, `'Anterior'`, `'Posterior'`, `'Inferior'`, `'Superior'`, `'Height'` and `'Width'`, `'AP'` (antero-posterior), `'lr'` (lateral), `'w'` (width) or `'i'` (inferior). Only the first letter of the string will be used. If the image is 2D, `'Height'` and `'Width'` may be used.
- **flip_probability** – Probability that the image will be flipped. This is computed on a per-axis basis.
- ****kwargs** – See `Transform` for additional keyword arguments.

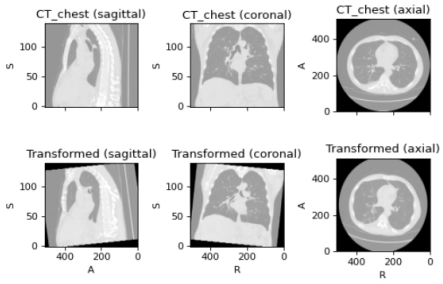
EXAMPLE

```
>>> import torchio as tio
>>> fpg = tio.datasets.FPG()
>>> flip = tio.RandomFlip(axes=('LR',))  # flip along lateral axis only
```

> **Tip**
>
> It is handy to specify the axes as anatomical labels when the image orientation is not known.

`RandomAffine`

(Source code, png, hires.png, pdf)




```
class torchio.transforms.RandomAffine(scales: Union[float, Tuple[float, float],
    Tuple[float, float, float], Tuple[float, float, float, float, float, float]] =
    0.1, degrees: Union[float, Tuple[float, float], Tuple[float, float, float],
    Tuple[float, float, float, float, float, float]] = 10, translation: Union[float,
    Tuple[float, float], Tuple[float, float, float], Tuple[float, float, float,
    float, float, float]] = 0, isotropic: bool = False, center: str = 'image',
    default_pad_value: Union[str, float] = 'minimum', image_interpolation: str =
    'linear', label_interpolation: str = 'nearest', check_shape: bool = True,
    **kwargs)                                                              [source]
```

Bases: `torchio.transforms.augmentation.random_transform.RandomTransform`, `torchio.transforms.spatial_transform.SpatialTransform`

Apply a random affine transformation and resample the image.

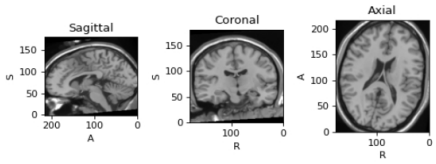
PARAMETERS

- **scales** – Tuple $(a_1, b_1, a_2, b_2, a_3, b_3)$ defining the scaling ranges. The scaling values along each dimension are (s_1, s_2, s_3), where $s_i \sim \mathcal{U}(a_i, b_i)$. If two values (a, b) are provided, then $s_i \sim \mathcal{U}(a, b)$. If only one value x is provided, then $s_i \sim \mathcal{U}(1 - x, 1 + x)$. If three values (x_1, x_2, x_3) are provided, then $s_i \sim \mathcal{U}(1 - x_i, 1 + x_i)$. For example, using `scales=(0.5, 0.5)` will zoom out the image, making the objects inside look twice as small while preserving the physical size and position of the image bounds.
- **degrees** – Tuple $(a_1, b_1, a_2, b_2, a_3, b_3)$ defining the rotation ranges in degrees. Rotation angles around each axis are $(\theta_1, \theta_2, \theta_3)$, where $\theta_i \sim \mathcal{U}(a_i, b_i)$. If two values (a, b) are provided, then $\theta_i \sim \mathcal{U}(a, b)$. If only one value x is provided, then $\theta_i \sim \mathcal{U}(-x, x)$. If three values (x_1, x_2, x_3) are provided, then $\theta_i \sim \mathcal{U}(-x_i, x_i)$.
- **translation** – Tuple $(a_1, b_1, a_2, b_2, a_3, b_3)$ defining the translation ranges in mm. Translation along each axis is (t_1, t_2, t_3), where $t_i \sim \mathcal{U}(a_i, b_i)$. If two values (a, b) are provided, then $t_i \sim \mathcal{U}(a, b)$. If only one value x is provided, then $t_i \sim \mathcal{U}(-x, x)$. If three values (x_1, x_2, x_3) are provided, then $t_i \sim \mathcal{U}(-x_i, x_i)$. For example, if the image is in RAS+ orientation (e.g., after applying `ToCanonical`) and the translation is $(10, 20, 30)$, the sample will move 10 mm to the right, 20 mm to the front, and 30 mm upwards. If the image was in, e.g., PIR+ orientation, the sample will move 10 mm to the back, 20 mm downwards, and 30 mm to the right.
- **isotropic** – If `True`, the scaling factor along all dimensions is the same, i.e. $s_1 = s_2 = s_3$.
- **center** – If `'image'`, rotations and scaling will be performed around the image center. If `'origin'`, rotations and scaling will be performed around the origin in world coordinates.
- **default_pad_value** – As the image is rotated, some values near the borders will be undefined. If `'minimum'`, the fill value will be the image minimum. If `'mean'`, the fill value is the mean of the border values. If `'otsu'`, the fill value is the mean of the values at the border that lie under an Otsu threshold. If it is a number, that value will be used.
- **image_interpolation** – See Interpolation.
- **label_interpolation** – See Interpolation.
- **check_shape** – If `True` an error will be raised if the images are in different physical spaces. If `False`, `center` should probably not be `'image'` but `'center'`.
- ****kwargs** – See `Transform` for additional keyword arguments.

EXAMPLE

```
>>> import torchio as tio
>>> image = tio.datasets.Colin27().t1
>>> transform = tio.RandomAffine(
...     scales=(0.9, 1.2),
...     degrees=15,
... )
>>> transformed = transform(image)
```

(Source code, png, hires.png, pdf)



`RandomElasticDeformation`

```
class torchio.transforms.RandomElasticDeformation(num_control_points: Union[int,
    Tuple[int, int, int]] = 7, max_displacement: Union[float, Tuple[float, float,
    float]] = 7.5, locked_borders: int = 2, image_interpolation: str = 'linear',
    label_interpolation: str = 'nearest', **kwargs)                    [source]
```

Bases: `torchio.transforms.augmentation.random_transform.RandomTransform`, `torchio.transforms.spatial_transform.SpatialTransform`

Apply dense random elastic deformation.

A random displacement is assigned to a coarse grid of control points around and inside the image. The displacement at each voxel is interpolated from the coarse grid using cubic B-splines.

The 'Deformable Registration' topic on ScienceDirect contains useful articles explaining interpolation of displacement fields using cubic B-splines.

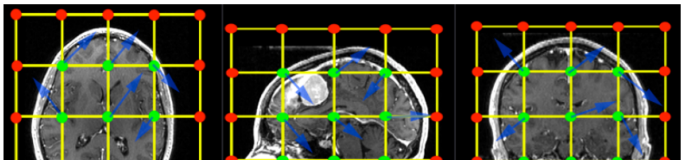
> **Warning**
>
> This transform is slow as it requires expensive computations. If your images are large you might want to use `RandomAffine` instead.

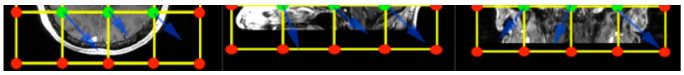
PARAMETERS

- **num_control_points** – Number of control points along each dimension of the coarse grid (n_x, n_y, n_z). If a single value n is passed, then $n_x = n_y = n_z = n$. Smaller numbers generate smoother deformations. The minimum number of control points is `4` as this transform uses cubic B-splines to interpolate displacement.
- **max_displacement** – Maximum displacement along each dimension at each control point (D_x, D_y, D_z). The displacement along dimension i at each control point is $d_i \sim \mathcal{U}(0, D_i)$. If a single value D is passed, then $D_x = D_y = D_z = D$. Note that the total maximum displacement would actually be $D_{max} = \sqrt{D_x^2 + D_y^2 + D_z^2}$.
- **locked_borders** – If `0`, all displacement vectors are kept. If `1`, displacement of control points at the border of the coarse grid will be set to `0`. If `2`, displacement of control points at the border of the image (red dots in the image below) will also be set to `0`.
- **image_interpolation** – See Interpolation. Note that this is the interpolation used to compute voxel intensities when resampling using the dense displacement field. The value of the dense displacement at each voxel is always interpolated with cubic B-splines from the values at the control points of the coarse grid.
- **label_interpolation** – See Interpolation.
- ****kwargs** – See `Transform` for additional keyword arguments.

This gist can also be used to better understand the meaning of the parameters.

This is an example from the 3D Slicer registration FAQ.





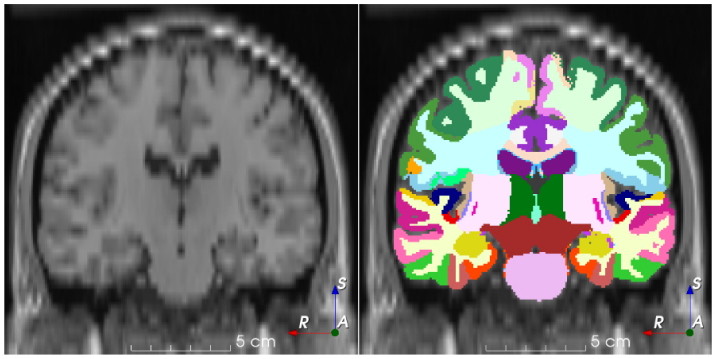
To generate a similar grid of control points with TorchIO, the transform can be instantiated as follows:

```
>>> from torchio import RandomElasticDeformation
>>> transform = RandomElasticDeformation(
...     num_control_points=(7, 7, 7),  # or just 7
...     locked_borders=2,
... )
```

Note that control points outside the image bounds are not showed in the example image (they would also be red as we set `locked_borders` to `2`).

> ⚠ Warning
>
> Image folding may occur if the maximum displacement is larger than half the coarse grid spacing. The grid spacing can be computed using the image bounds in physical space [1] and the number of control points:
>
> ```
> >>> import numpy as np
> >>> import torchio as tio
> >>> image = tio.datasets.Slicer().MRHead.as_sitk()
> >>> image.GetSize()  # in voxels
> (256, 256, 130)
> >>> image.GetSpacing()  # in mm
> (1.0, 1.0, 1.2999954223632812)
> >>> bounds = np.array(image.GetSize()) * np.array(image.GetSpacing())
> >>> bounds  # mm
> array([256.        , 256.        , 168.99940491])
> >>> num_control_points = np.array((7, 7, 6))
> >>> grid_spacing = bounds / (num_control_points - 2)
> >>> grid_spacing
> array([51.2       , 51.2       , 42.24985123])
> >>> potential_folding = grid_spacing / 2
> >>> potential_folding  # mm
> array([25.6       , 25.6       , 21.12492561])
> ```
>
> Using a `max_displacement` larger than the computed `potential_folding` will raise a `RuntimeWarning`.
>
> [1]: Technically, 2ϵ should be added to the image bounds, where $\epsilon = 2^{-3}$ according to ITK source code.

`RandomAnisotropy`




```
class torchio.transforms.RandomAnisotropy(axes: Union[int, Tuple[int, ...]] = (0, 1,
    2), downsampling: Union[float, Tuple[float, float]] = (1.5, 5),
    image_interpolation: str = 'linear', scalars_only: bool = True, **kwargs) [source]
```

Bases: `torchio.transforms.augmentation.random_transform.RandomTransform`

Downsample an image along an axis and upsample to initial space.

This transform simulates an image that has been acquired using anisotropic spacing and resampled back to its original spacing.

Similar to the work by Billot et al.: Partial Volume Segmentation of Brain MRI Scans of any Resolution and Contrast.

PARAMETERS

- **axes** – Axis or tuple of axes along which the image will be downsampled.
- **downsampling** – Downsampling factor $m > 1$. If a tuple (a, b) is provided then $m \sim \mathcal{U}(a, b)$.
- **image_interpolation** – Image interpolation used to upsample the image back to its initial spacing. Downsampling is performed using nearest neighbor interpolation. See Interpolation for supported interpolation types.
- **scalars_only** – Apply only to instances of `torchio.ScalarImage`. This is useful when the segmentation quality needs to be kept, as in Billot et al..
- ****kwargs** – See `Transform` for additional keyword arguments.

EXAMPLE

```
>>> import torchio as tio
>>> transform = tio.RandomAnisotropy(axes=1, downsampling=2)
>>> transform = tio.RandomAnisotropy(
...     axes=(0, 1, 2),
...     downsampling=(2, 5),
... )  # Multiply spacing of one of the 3 axes by a factor randomly chosen in [2, 5]
>>> colin = tio.datasets.Colin27()
>>> transformed = transform(colin)
```

Intensity

RandomMotion

```
class torchio.transforms.RandomMotion(degrees: float = 10, translation: float = 10,
    num_transforms: int = 2, image_interpolation: str = 'linear', **kwargs)  [source]
```

Bases: `torchio.transforms.augmentation.random_transform.RandomTransform`, `torchio.transforms.intensity_transform.IntensityTransform`, `torchio.transforms.fourier.FourierTransform`

Add random MRI motion artifact.

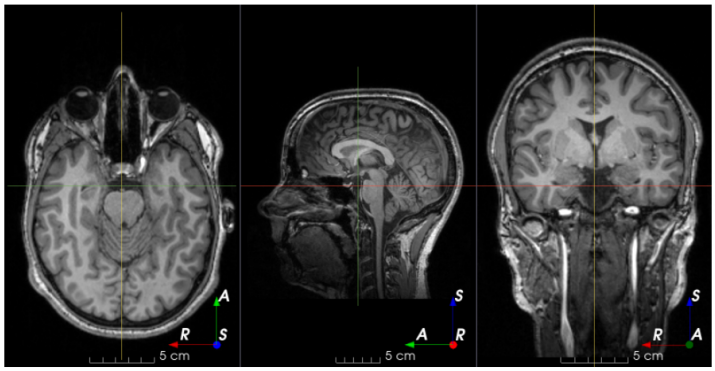
Magnetic resonance images suffer from motion artifacts when the subject moves during image acquisition. This transform follows Shaw et al., 2019 to simulate motion artifacts for data augmentation.

PARAMETERS

- **degrees** – Tuple (a, b) defining the rotation range in degrees of the simulated movements. The rotation angles around each axis are $(\theta_1, \theta_2, \theta_3)$, where $\theta_i \sim \mathcal{U}(a, b)$. If only one value d is provided, $\theta_i \sim \mathcal{U}(-d, d)$. Larger values generate more distorted images.
- **translation** – Tuple (a, b) defining the translation in mm of the simulated movements. The translations along each axis are (t_1, t_2, t_3), where $t_i \sim \mathcal{U}(a, b)$. If only one value t is provided, $t_i \sim \mathcal{U}(-t, t)$. Larger values generate more distorted images.
- **num_transforms** – Number of simulated movements. Larger values generate more distorted images.
- **image_interpolation** – See Interpolation.
- ****kwargs** – See `Transform` for additional keyword arguments.

> ⚠ Warning
>
> Large numbers of movements lead to longer execution times for 3D images.

RandomGhosting



```
class torchio.transforms.RandomGhosting(num_ghosts: Union[int, Tuple[int, int]] = (4,
    10), axes: Union[int, Tuple[int, ...]] = (0, 1, 2), intensity: Union[float,
    Tuple[float, float]] = (0.5, 1), restore: float = 0.02, **kwargs)  [source]
```

Bases: `torchio.transforms.augmentation.random_transform.RandomTransform`, `torchio.transforms.intensity_transform.IntensityTransform`

Add random MRI ghosting artifact.

Discrete "ghost" artifacts may occur along the phase-encode direction whenever the position or signal intensity of imaged structures within the field-of-view vary or move in a regular (periodic) fashion. Pulsatile flow of blood or CSF, cardiac motion, and respiratory motion are the most important patient-related causes of ghost artifacts in clinical MR imaging (from mriquestions.com).

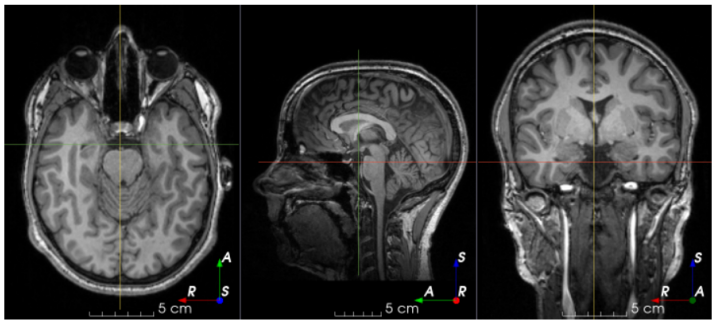
PARAMETERS

- **num_ghosts** – Number of 'ghosts' n in the image. If `num_ghosts` is a tuple (a, b), then $n \sim \mathcal{U}(a, b) \cap \mathbb{N}$. If only one value d is provided, $n \sim \mathcal{U}(0, d) \cap \mathbb{N}$.
- **axes** – Axis along which the ghosts will be created. If `axes` is a tuple, the axis will be randomly chosen from the passed values. Anatomical labels may also be used (see `RandomFlip`).
- **intensity** – Positive number representing the artifact strength s with respect to the maximum of the k-space. If `0`, the ghosts will not be visible. If a tuple (a, b) is provided then $s \sim \mathcal{U}(a, b)$. If only one value d is provided, $s \sim \mathcal{U}(0, d)$.
- **restore** – Number between `0` and `1` indicating how much of the k-space center should be restored after removing the planes that generate the artifact.

- ****kwargs** – See `Transform` for additional keyword arguments.

> **Note**
>
> The execution time of this transform does not depend on the number of ghosts.

`RandomSpike`




```
class torchio.transforms.RandomSpike(num_spikes: Union[int, Tuple[int, int]] = 1,
    intensity: Union[float, Tuple[float, float]] = (1, 3), **kwargs)          [source]
```

Bases: `torchio.transforms.augmentation.random_transform.RandomTransform`, `torchio.transforms.intensity_transform.IntensityTransform`, `torchio.transforms.fourier.FourierTransform`

Add random MRI spike artifacts.

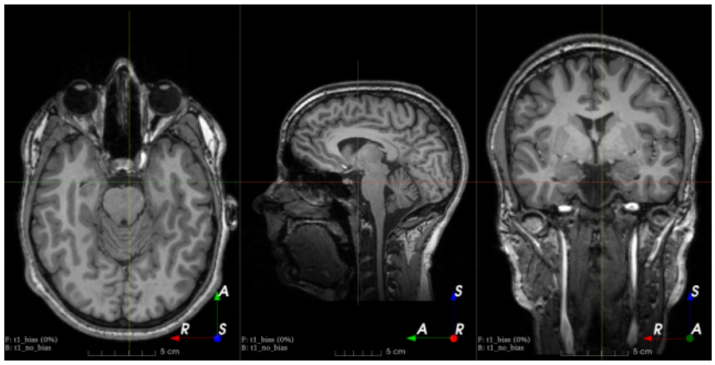
Also known as Herringbone artifact, crisscross artifact or corduroy artifact, it creates stripes in different directions in image space due to spikes in k-space.

PARAMETERS

- **num_spikes** – Number of spikes n present in k-space. If a tuple (a, b) is provided, then $n \sim \mathcal{U}(a, b) \cap \mathbb{N}$. If only one value d is provided, $n \sim \mathcal{U}(0, d) \cap \mathbb{N}$. Larger values generate more distorted images.
- **intensity** – Ratio r between the spike intensity and the maximum of the spectrum. If a tuple (a, b) is provided, then $r \sim \mathcal{U}(a, b)$. If only one value d is provided, $r \sim \mathcal{U}(-d, d)$. Larger values generate more distorted images.
- ****kwargs** – See `Transform` for additional keyword arguments.

> **Note**
>
> The execution time of this transform does not depend on the number of spikes.

`RandomBiasField`




```
class torchio.transforms.RandomBiasField(coefficients: Union[float, Tuple[float,
    float]] = 0.5, order: int = 3, **kwargs)                                  [source]
```

Bases: `torchio.transforms.augmentation.random_transform.RandomTransform`, `torchio.transforms.intensity_transform.IntensityTransform`

Add random MRI bias field artifact.

MRI magnetic field inhomogeneity creates intensity variations of very low frequency across the whole image.

The bias field is modeled as a linear combination of polynomial basis functions, as in K. Van Leemput et al., 1999, *Automated model-based tissue classification of MR images of the brain*.

It was implemented in NiftyNet by Carole Sudre and used in Sudre et al., 2017, Longitudinal segmentation of age-related white matter hyperintensities.

PARAMETERS

- **coefficients** – Maximum magnitude n of polynomial coefficients. If a tuple (a, b) is specified, then $n \sim \mathcal{U}(a, b)$.
- **order** – Order of the basis polynomial functions.
- ****kwargs** – See `Transform` for additional keyword arguments.

`RandomBlur`

```
class torchio.transforms.RandomBlur(std: Union[float, Tuple[float, float]] = (0, 2),
    **kwargs)                                                                 [source]
```

Bases: `torchio.transforms.augmentation.random_transform.RandomTransform`

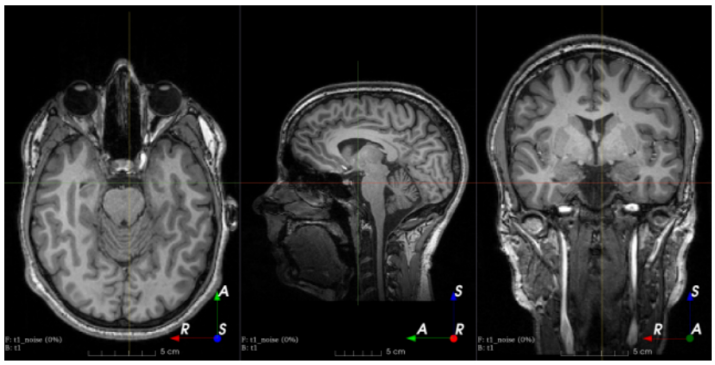
Bases: `torchio.transforms.augmentation.random_transform.RandomTransform`, `torchio.transforms.intensity_transform.IntensityTransform`

Blur an image using a random-sized Gaussian filter.

PARAMETERS

- **std** – Tuple $(a_1, b_1, a_2, b_2, a_3, b_3)$ representing the ranges (in mm) of the standard deviations $(\sigma_1, \sigma_2, \sigma_3)$ of the Gaussian kernels used to blur the image along each axis, where $\sigma_i \sim \mathcal{U}(a_i, b_i)$. If two values (a, b) are provided, then $\sigma_i \sim \mathcal{U}(a, b)$. If only one value x is provided, then $\sigma_i \sim \mathcal{U}(0, x)$. If three values (x_1, x_2, x_3) are provided, then $\sigma_i \sim \mathcal{U}(0, x_i)$.
- ****kwargs** – See `Transform` for additional keyword arguments.

`RandomNoise`



```
class torchio.transforms.RandomNoise(mean: Union[float, Tuple[float, float]] = 0,
    std: Union[float, Tuple[float, float]] = (0, 0.25), **kwargs)  [source]
```

Bases: `torchio.transforms.augmentation.random_transform.RandomTransform`, `torchio.transforms.intensity_transform.IntensityTransform`

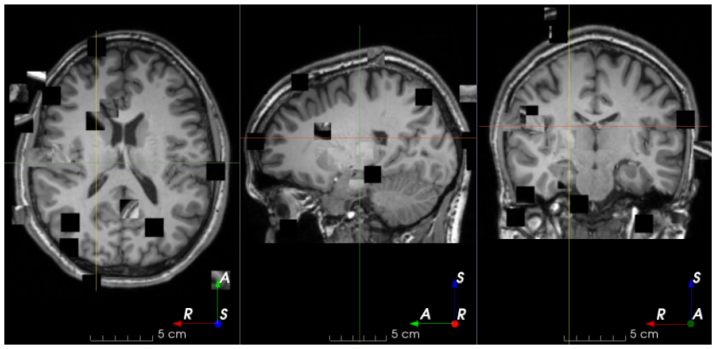
Add Gaussian noise with random parameters.

Add noise sampled from a normal distribution with random parameters.

PARAMETERS

- **mean** – Mean μ of the Gaussian distribution from which the noise is sampled. If two values (a, b) are provided, then $\mu \sim \mathcal{U}(a, b)$. If only one value d is provided, $\mu \sim \mathcal{U}(-d, d)$.
- **std** – Standard deviation σ of the Gaussian distribution from which the noise is sampled. If two values (a, b) are provided, then $\sigma \sim \mathcal{U}(a, b)$. If only one value d is provided, $\sigma \sim \mathcal{U}(0, d)$.
- ****kwargs** – See `Transform` for additional keyword arguments.

`RandomSwap`




```
class torchio.transforms.RandomSwap(patch_size: Union[int, Tuple[int, int, int]] =
    15, num_iterations: int = 100, **kwargs)  [source]
```

Bases: `torchio.transforms.augmentation.random_transform.RandomTransform`, `torchio.transforms.intensity_transform.IntensityTransform`

Randomly swap patches within an image.

This is typically used in context restoration for self-supervised learning.

PARAMETERS

- **patch_size** – Tuple of integers (w, h, d) to swap patches of size $w \times h \times d$. If a single number n is provided, $w = h = d = n$.
- **num_iterations** – Number of times that two patches will be swapped.
- ****kwargs** – See `Transform` for additional keyword arguments.

`RandomLabelsToImage`

```
class torchio.transforms.RandomLabelsToImage(label_key: Optional[str] = None,
    used_labels: Optional[Sequence[int]] = None, image_key: str =
    'image_from_labels', mean: Optional[Sequence[Union[float, Tuple[float, float]]]]
    = None, std: Optional[Sequence[Union[float, Tuple[float, float]]]] = None,
    default_mean: Union[float, Tuple[float, float]] = (0.1, 0.9), default_std:
    Union[float, Tuple[float, float]] = (0.01, 0.1), discretize: bool = False,
    **kwargs)  [source]
```

Bases: `torchio.transforms.augmentation.random_transform.RandomTransform`, `torchio.transforms.intensity_transform.IntensityTransform`

Randomly generate an image from a segmentation.

Based on the works by Billot et al.: A Learning Strategy for Contrast-agnostic MRI Segmentation and Partial Volume Segmentation of Brain MRI Scans of any Resolution and Contrast.

PARAMETERS

- **label_key** – String designating the label map in the subject that will be used to generate the new image.
- **used_labels** – Sequence of integers designating the labels used to generate the new image. If categorical encoding is used, `label_channels` refers to the values of the categorical encoding. If one hot encoding or partial-volume label maps are used, `label_channels` refers to the channels of the label maps. Default uses all labels. Missing voxels will be filled with zero or with voxels from an already existing volume, see `image_key`.
- **image_key** – String designating the key to which the new volume will be saved. If this key corresponds to an already existing volume, missing voxels will be filled with the corresponding values in the original volume.
- **mean** – Sequence of means for each label. For each value v, if a tuple (a, b) is provided then $v \sim \mathcal{U}(a, b)$. If `None`, `default_mean` range will be used for every label. If not `None` and `label_channels` is not `None`, `mean` and `label_channels` must have the same length.
- **std** – Sequence of standard deviations for each label. For each value v, if a tuple (a, b) is provided then $v \sim \mathcal{U}(a, b)$. If `None`, `default_std` range will be used for every label. If not `None` and `label_channels` is not `None`, `std` and `label_channels` must have the same length.
- **default_mean** – Default mean range.
- **default_std** – Default standard deviation range.
- **discretize** – If `True`, partial-volume label maps will be discretized. Does not have any effects if not using partial-volume label maps. Discretization is done taking the class of the highest value per voxel in the different partial-volume label maps using `torch.argmax()` on the channel dimension (i.e. 0).
- ****kwargs** – See `Transform` for additional keyword arguments.

> **Tip**
>
> It is recommended to blur the new images to make the result more realistic. See `RandomBlur`.

EXAMPLE

```
>>> import torchio as tio
>>> subject = tio.datasets.ICBM2009CNonlinearSymmetric()
>>> # Using the default parameters
>>> transform = tio.RandomLabelsToImage(label_key='tissues')
>>> # Using custom mean and std
>>> transform = tio.RandomLabelsToImage(
...     label_key='tissues', mean=[0.33, 0.66, 1.], std=[0, 0, 0]
... )
>>> # Discretizing the partial volume maps and blurring the result
>>> simulation_transform = tio.RandomLabelsToImage(
...     label_key='tissues', mean=[0.33, 0.66, 1.], std=[0, 0, 0], discretize=True
... )
>>> blurring_transform = tio.RandomBlur(std=0.3)
>>> transform = tio.Compose([simulation_transform, blurring_transform])
>>> transformed = transform(subject)  # subject has a new key 'image_from_labels' with th
>>> # Filling holes of the simulated image with the original T1 image
>>> rescale_transform = tio.RescaleIntensity(
...     out_min_max=(0, 1), percentiles=(1, 99))   # Rescale intensity before filling ho
>>> simulation_transform = tio.RandomLabelsToImage(
...     label_key='tissues',
...     image_key='t1',
...     used_labels=[0, 1]
... )
>>> transform = tio.Compose([rescale_transform, simulation_transform])
>>> transformed = transform(subject)  # subject's key 't1' has been replaced with the sim
```

RandomGamma

```
class torchio.transforms.RandomGamma(log_gamma: Union[float, Tuple[float, float]] =
    (- 0.3, 0.3), **kwargs)                                                    [source]
```

Bases: `torchio.transforms.augmentation.random_transform.RandomTransform`, `torchio.transforms.intensity_transform.IntensityTransform`

Randomly change contrast of an image by raising its values to the power γ.

PARAMETERS

- **log_gamma** – Tuple (a, b) to compute the exponent $\gamma = e^{\beta}$, where $\beta \sim \mathcal{U}(a, b)$. If a single value d is provided, then $\beta \sim \mathcal{U}(-d, d)$. Negative and positive values for this argument perform gamma compression and expansion, respectively. See the Gamma correction Wikipedia entry for more information.
- ****kwargs** – See `Transform` for additional keyword arguments.

> **Note**
>
> Fractional exponentiation of negative values is generally not well-defined for non-complex numbers. If negative values are found in the input image I, the applied transform is $\text{sign}(I)|I|^{\gamma}$, instead of the usual I^{γ}. The `RescaleIntensity` transform may be used to ensure that all values are positive. This is generally not problematic, but it is recommended to visualize results on image with negative values. More information can be found on this StackExchange question.

EXAMPLE

```
>>> import torchio as tio
>>> subject = tio.datasets.FPG()
>>> transform = tio.RandomGamma(log_gamma=(-0.3, 0.3))  # gamma between 0.74 and 1.34
>>> transformed = transform(subject)
```