
Tormentor

Release 0.1.1

Anguelos Nicolaou

Sep 20, 2022

CONTENTS:

1	tormentor	1
1.1	Sampling Fields	1
1.2	Abstract Augmentation Types	2
1.3	Spatial Augmentations	5
1.4	Color Augmentations	7
1.5	Document Augmentations	9
1.6	Composite Augmentation Types	10
2	Indices and tables	13
	Index	15

TORMENTOR

1.1 Sampling Fields

`tormentor.create_sampling_field(width: int, height: int, batch_size: int = 1, device: device = 'cpu') → Tuple[FloatTensor, FloatTensor]`

Creates a SamplingField.

A SamplingField is a tuple of 3D tensors of the same size. Sampling fields are augmentable by all augmentations although many augmentations (Non-spatial) have no effect on them. They can be used to resample images, pointclouds, masks. When sampling, for both axes, the input image is interpreted to lie on the region $[-1, 1]$. The output image when resampling will have the width and height of the sampling field. A sampling field can also refer to a single image rather than a batch in which case the tensors are 2D. The first dimension is the batch size. The second dimension is the width of the output image after sampling. The third dimension is the height of the output image after sampling. The created sampling fields are normalised in the range $[-1, 1]$ regardless of their size. Although not enforced, it is expected that augmentations are homomorphisms. Sampling fields are expected to operate identically on all channels and don't have a channel dimension.

Parameters

- **width** – The sampling fields width.
- **height** – The sampling fields height.
- **batch_size** – If 0, the sampling field refers to a single image. Otherwise the first dimension of the tensors. Created sampling fields are simply repeated over the batch dimension. Default value is 1.
- **device** – the device on which the sampling field will be created.

Returns

A tuple of 3D or 2D tensors with values ranged in $[-1, 1]$

`tormentor.apply_sampling_field(input_img: Tensor, coords: Tuple[FloatTensor, FloatTensor])`

Resamples one or more images by applying sampling fields.

Bilinear interpolation is employed.

Parameters

- **input_img** – A 4D float tensor [batch x channel x height x width] or a 3D tensor [channel x height x width]. Containing the image or batch from which the image is sampled.
- **coords** – A sampling field with 3D [batch x out_height x out_width] or 2D [out_height x out_width]. The dimensions of the sampling fields must be one less than the input_img.

Returns

A tensor of as many dimensions [batch x channel x out_height x out_width] or [channel x out_height x out_width]] as the input.

1.2 Abstract Augmentation Types

class tormentor.**DeterministicImageAugmentation**(*aug_id=None, seed=None*)

Deterministic augmentation functor and its factory.

This class is the base class realising an augmentation. In order to create a `create_persistent` augmentation the `forward_sample_img` method and the factory class-function have to be defined. If a constructor is defined it must call `super().__init__(**kwargs)`.

The `**kwargs` on the constructor define all needed variables for generating random augmentations. The factory method returns a lambda that instantiates augmentations. Any parameter about the augmentations randomness should be passed by the factory to the constructor and used inside `forward_sample_img`'s definition.

augment_image(*image_tensor: FloatTensor*)

Augments an image or a batch of images.

This method enforces determinism for image data. Only the batch dimension is guaranteed to be preserved. Channels, width, and height dimensions can differ on the outputs. This method should be treated as final and should not be redefined in subclasses. All subclasses should implement `forward_image` instead. Images can have any number of channels although some augmentations eg. those manipulating the color-space might expect specific channel counts.

Parameters

image_tensor – a float tensor of [batch x channels x height x width] or [channels x height x width].

Returns

An image or a batch of tensors sized [batch x channels x height x width] or [channels x height x width]

augment_image(*image_tensor: FloatTensor*)

Augments an image or a batch of images.

This method enforces determinism for image data. Only the batch dimension is guaranteed to be preserved. Channels, width, and height dimensions can differ on the outputs. This method should be treated as final and should not be redefined in subclasses. All subclasses should implement `forward_image` instead. Images can have any number of channels although some augmentations eg. those manipulating the color-space might expect specific channel counts.

Parameters

image_tensor – a float tensor of [batch x channels x height x width] or [channels x height x width].

Returns

An image or a batch of tensors sized [batch x channels x height x width] or [channels x height x width]

augment_mask(*mask_tensor: Tensor*)

Augments an mask or a batch of masks.

Masks differ from images as they are interpreted to answer a pixel-wise “where” question. Although technically they can be indistinguishable from images they are interpreted differently. A typical example would be a dense segmentation mask such containing class one-hot encoding along the channel dimension. This

method enforces determinism for mask data. Only the batch dimension is guaranteed to be preserved. Channels, width, and height dimensions can differ on the outputs. This method should be treated as final and should not be redefined in subclasses. A subclasses should implement `forward_mask` instead.

Parameters

mask_tensor – a float tensor of [batch x channels x height x width] or [channels x height x width].

Returns

An mask or a batch of tensors sized [batch x channels x height x width] or [channels x height x width]

augment_pointcloud(*pc: Union[List[Tuple[FloatTensor, FloatTensor]], Tuple[FloatTensor, FloatTensor]]*, *image_tensor: FloatTensor*, *compute_img: bool*)

Augments pointclouds over an image or a batch.

Pointclouds are defined to be in pixel coordinates in contextualised by an image or at least an image size. The pointcloud for a single image is a tuple of 1D float tensors (vectors) one with the X coordinates and one with the Y coordinates. If the image_tensor is a batch, then a list of pointclouds is associated with the batch, one for every image in the batch. Pointcloud augmentation shares a lot of the heavy computation with augmenting its reference image tensor, both are employing an augmented sampling field. This method should be treated as final and should not be redefined in subclasses. A subclasses should implement `forward_pointcloud` instead.

Parameters

- **pc** – Either a tuple of vectors with X Y coordinates, or a list of many such tuples.
- **image_tensor** – A 3D tensor [channel x height x width] or a 4D tensor [batch x channel x height x width].
- **compute_img** – If True, the reference image will be augmented and returned, if false the reference image will be returned unaugmented.

Returns

A tuple with a pointcloud or a list of pointclouds, and a 3D or 4D tensor. The image tensor is either the original image_tensor or the same exact augmentation applied the point cloud.

augment_sampling_field(*sf: Tuple[FloatTensor, FloatTensor]*)

Augments a sampling field for an image or samplingfields for batches.

Sampling fields are the way to see how augmentations move things around. A sampling field can be generated with `create_sampling_field` and be used to resample image data with `apply_sampling_field`

This method enforces determinism for image data. Only the batch dimension is guaranteed to be preserved. Channels, width, and height dimensions can differ on the outputs. This method should be treated as final and should not be redefined in subclasses. A subclasses should implement `forward_sampling_field` instead.

Parameters

sf – a tuple with 2 float tensors of the same size. Either [batch x height x width] or [height x width]

Returns

A tuple of 2 tensors sized [batch x new_height x new_width] or [new_height x new_width]

forward_bboxes(*bboxes: FloatTensor*, *image_tensor=None*, *width_height=None*) → FloatTensor

Applies a transformation on Image coordinate defined bounding boxes.

Bounding Boxes are encoded as [Left, Top, Right, Bottom]

Parameters

- **bboxes** (*torch.FloatTensor*) – A tensor with bboxes for a sample [N x 4] or a batch [S x N x 4]
- **image_tensor** (*torch.FloatTensor*) – A valid batch image tensor [S x C x H x W] or sample image tensor [C x H x W]. In both cases it only used to normalise bbox coordinates and can be omitted if width_height is specified.
- **width_height** (*int*, *int*) – Values used to normalise bbox coordinates to [-1,1] and back, should be omitted if image tensor is passed

Returns: a tensor with the bounding boxes of the transformed bounding box.

forward_img(*batch_tensor: FloatTensor*) → *FloatTensor*

Distorts a batch of one or more images.

Parameters

batch_tensor – Images are 4D tensors of [batch_size, #channels, height, width] size.

Returns

A create_persistent 4D tensor [batch_size, #channels, height, width] with the create_persistent image.

forward_img_path_probabilities(*batch_tensor: FloatTensor*) → *FloatTensor*

Returns the probability of a specific augmentation's path being sampled

Normally only augmentation choice and cascade are not leafs in the execution graph.

Parameters

batch_tensor – A tensor in order to infer the batchsize

Returns

A 1D tensor with the probability of each batch-sample's augmentation path probability.

forward_mask(*batch_tensor: Tensor*) → *LongTensor*

Distorts a a batch of one or more masks.

Parameters

batch_tensor – Images are 4D tensors of [batch_size, #channels, height, width] size.

Returns

A create_persistent 4D tensor [batch_size, #channels, height, width] with the create_persistent image.

forward_pointcloud(*pcl: List[Tuple[FloatTensor, FloatTensor]]*, *batch_tensor: FloatTensor*, *compute_img: bool*) → *Tuple[List[Tuple[FloatTensor, FloatTensor]], FloatTensor]*

Applies a transformation on normalised coordinate points.

:param pcl, a pointcloud for every image in the batch pointclouds are given in pixel coordinates. The list must have the same size as the batch_tensor. Each pointcloud is a tuple consisting of the vectors

Parameters

- **batch_tensor** (*torch.FloatTensor*) – The images to which each of the pointclouds refers [BxCxHxW]
- **compute_img** (*bool*) – If *False* the only the pointcloud will be computed, if *True* the images in the batch_tensor will also be augmented.

Returns

augmented pointclouds and input image tensor or the augmented image tensor depending on compute_img.

Return type

PointCloudsImages

forward_sampling_field(*coords*: *Tuple*[*FloatTensor*, *FloatTensor*]) → *Tuple*[*FloatTensor*, *FloatTensor*]

Defines a spatial transform.

Parameters

Q (*coords*) – a tuple with two 3D float tensors each having a size of [Batch x Height x Width]. X and Y coordinates are in reference the range [-1, 1].

Returns

The augmented samplint field.

generate_batch_state(*batch_tensor*: *Tensor*) → *Tuple*[*Tensor*, ...]

Generates deterministic state for each augmentation.

Returns: a tuple of tensors representing the complete state of the augmentaion so that a

like_me()

Returns a new augmentation following the same distributions as self.

Returns

An instance of the same class as self.

class tormentor.**SpatialImageAugmentation**(*aug_id=None*, *seed=None*)

Parent class for augmentations that move things around.

Every class were image pixels move around rather that just change should be a descendant of this class. All classes that do not descend from this class are expected to be neutral for pointclouds, and sampling fields and should be descendants of StaticImageAugmentation.

class tormentor.**StaticImageAugmentation**(*aug_id=None*, *seed=None*)

Parent class for augmentations that don't move things around.

All classes that do descend from this are expected to be neutral for pointclouds, sampling fields although they might be erasing regions of the images so they are not guarntied to be neutral to masks. Every class were image pixels move around rather that just stay static, should be a descendant of SpatialImageAugmentation.

class tormentor.**ColorAugmentation**(*aug_id=None*, *seed=None*)

Abstract class for all augmentations manipulating the colorspace.

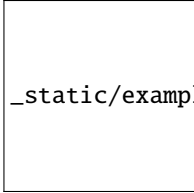
All augmentations inheriting ColorAugmentation, expect 3-channel inputs that can be interpreted as RGB in the range [0., 1.]. If the channels are neither 3 or 1, the augmentation becomes an identity. The sub-classes should only define `generate_batch_state(self, batch: torch.FloatTensor)` and classmethod `functional_image(cls, batch: torch.FloatTensor, *batch_state)`.

1.3 Spatial Augmentations

class tormentor.**Perspective**(*aug_id=None*, *seed=None*)

Applies a perspective transformation on the data by moving the corners of an image.

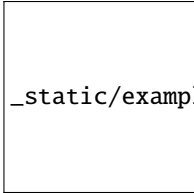
This augmentation is parametrised by two random variables `x_offset` and `y_offset` which are the multipliers of each of the image corners corners (-1, -1), (1, -1), (1, 1), and (-1, 1).



`_static/example_images/Perspective.png`

class tormentor.**Rotate**(*aug_id=None, seed=None*)

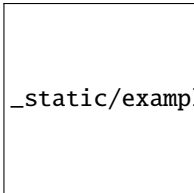
Rotates the image around the center.



`_static/example_images/Rotate.png`

class tormentor.**Zoom**(*aug_id=None, seed=None*)

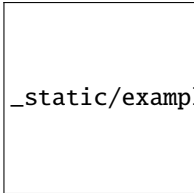
Augments by scaling images preserving their aspect ratio.



`_static/example_images/Zoom.png`

class tormentor.**Scale**(*aug_id=None, seed=None*)

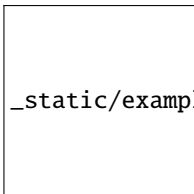
Augmentation by scaling images preserving aspect ratio.



`_static/example_images/Scale.png`

class tormentor.**Translate**(*aug_id=None, seed=None*)

Augmentation by translating images.



`_static/example_images/Translate.png`

class tormentor.**ScaleTranslate**(*aug_id=None, seed=None*)

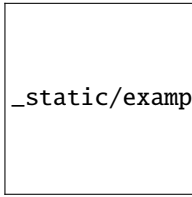
Augmentation by scaling and translating images preserving aspect ratio.



`_static/example_images/ScaleTranslate.png`

```
class tormentor.Flip(aug_id=None, seed=None)
```

Implementation of augmentation by flipping the X or Y axis.

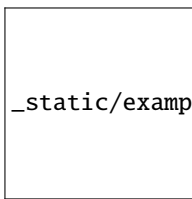


_static/example_images/Flip.png

```
class tormentor.ElasticTransform(aug_id=None, seed=None)
```

Augmentation ElasticTransform.

This augmentation does not guaranty to be a homomorphism. In order for the augmentation to behave as a homomorphism, `harmonic_smoothing` must be quite low. On the other hand, the complexity of the operation is n^2 with respect to `harmonic_smoothing` or $n \log(n)$ depending of how the gaussian filters are implemented.



_static/example_images/ElasticTransform.png

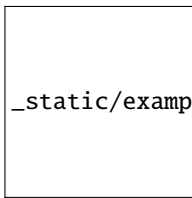
```
class tormentor.Wrap(aug_id=None, seed=None)
```

Augmentation Wrap.

This augmentation acts like many simultaneous elastic transforms with gaussian sigmas set at various harmonics.

Distributions:

`roughness`: Quantification of the local inconsistency of the distortion effect. `intensity`: Quantification of the intensity of the distortion effect.



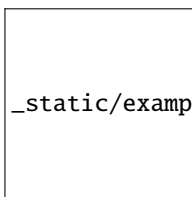
_static/example_images/Wrap.png

```
class tormentor.Shred(aug_id=None, seed=None)
```

1.4 Color Augmentations

```
class tormentor.Invert(aug_id=None, seed=None)
```

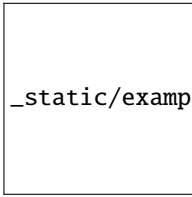
Performs color inversion in HSV colorspace for some images randomly selected.



_static/example_images/Invert.png

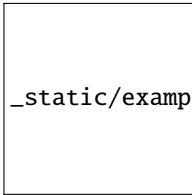
class tormentor.**Brightness**(*aug_id=None, seed=None*)

Changes the brightness of the image.



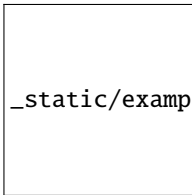
class tormentor.**Saturation**(*aug_id=None, seed=None*)

Changes the saturation of the image.



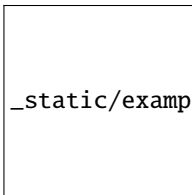
class tormentor.**Contrast**(*aug_id=None, seed=None*)

Changes the contrast of the image.



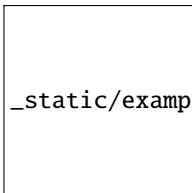
class tormentor.**Hue**(*aug_id=None, seed=None*)

Changes the Hue of the image.



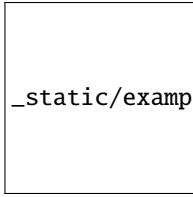
class tormentor.**ColorJitter**(*aug_id=None, seed=None*)

Changes hue, contrast, saturation, and brightness of the image.



class tormentor.**PlasmaBrightness**(*aug_id=None, seed=None*)

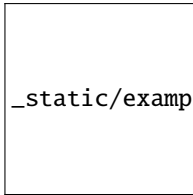
Changes the brightness of the image locally.



_static/example_images/PlasmaBrightness.png

class tormentor.PlasmaRgbBrightness(*aug_id=None, seed=None*)

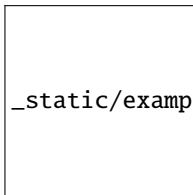
Changes the saturation of the image.



_static/example_images/Saturation.png

class tormentor.PlasmaContrast(*aug_id=None, seed=None*)

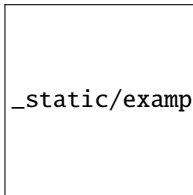
Changes the contrast of the image locally.



_static/example_images/PlasmaContrast.png

class tormentor.PlasmaShadow(*aug_id=None, seed=None*)

Lowens the brightness of the image over a random mask.



_static/example_images/PlasmaShadow.png

1.5 Document Augmentations

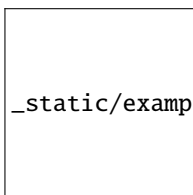
class tormentor.Wrap(*aug_id=None, seed=None*)

Augmentation Wrap.

This augmentation acts like many simultaneous elastic transforms with gaussian sigmas set at various harmonics.

Distributions:

roughness: Quantification of the local inconsistency of the distortion effect. **intensity:** Quantification of the intensity of the distortion effect.



_static/example_images/Wrap.png

```
class tormentor.Shred(aug_id=None, seed=None)
```

1.6 Composite Augmentation Types

```
class tormentor.AugmentationCascade
```

Select randomly among many augmentations.

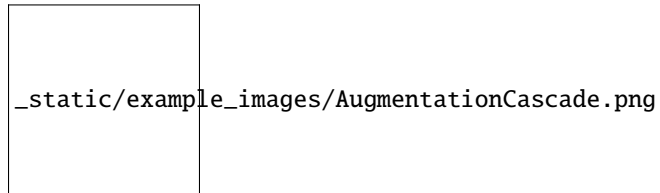


Fig. 1: Cascade of perspective augmentation followed by plasma-brightness

```
augmentation_factory = tormentor.RandomPerspective | tormentor.RandomPlasmaBrightness
```

A more complete usage of AugmentationCascade and AugmentationChoice can be seen in the following listing which produces the following computation graph. In the graph AugmentationCascade can be thought of as all arrows that don't leave an AugmentationChoice

```
from tormentor import RandomColorJitter, RandomFlip, RandomWrap, \
    RandomPlasmaBrightness, RandomPerspective, \
    RandomGaussianAdditiveNoise, RandomRotate

linear_aug = (RandomFlip ^ RandomPerspective ^ RandomRotate) | RandomColorJitter
nonlinear_aug = RandomWrap | RandomPlasmaBrightness
final_augmentation = (linear_aug ^ nonlinear_aug) | RandomGaussianAdditiveNoise

epochs, batch_size, n_points, width, height = 10, 5, 20, 320, 240

for _ in range(epochs):
    image_batch = torch.rand(batch_size, 3, height, width)
    segmentation_batch = torch.rand(batch_size, 1, height, width).round()
    augmentation = final_augmentation()
    augmented_images = augmentation(image_batch)
    augmented_gt = augmentation(segmentation_batch)
    # Train and do other things
```

```
class tormentor.AugmentationChoice(aug_id=None, seed=None)
```

Select randomly among many augmentations.



Fig. 2: Random choice of perspective and plasma-brightness augmentations

```
augmentation_factory = tormentor.RandomPerspective ^ tormentor.RandomPlasmaBrightness
augmentation = augmentation_factory()
augmented_image = augmentation(image)
```


INDICES AND TABLES

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

A

[apply_sampling_field\(\)](#) (in module *tormentor*), 1
[augment_image\(\)](#) (*tormentor.DeterministicImageAugmentation* method), 2
[augment_mask\(\)](#) (*tormentor.DeterministicImageAugmentation* method), 2
[augment_pointcloud\(\)](#) (*tormentor.DeterministicImageAugmentation* method), 3
[augment_sampling_field\(\)](#) (*tormentor.DeterministicImageAugmentation* method), 3
[AugmentationCascade](#) (class in *tormentor*), 10
[AugmentationChoice](#) (class in *tormentor*), 10

B

[Brightness](#) (class in *tormentor*), 7

C

[ColorAugmentation](#) (class in *tormentor*), 5
[ColorJitter](#) (class in *tormentor*), 8
[Contrast](#) (class in *tormentor*), 8
[create_sampling_field\(\)](#) (in module *tormentor*), 1

D

[DeterministicImageAugmentation](#) (class in *tormentor*), 2

E

[ElasticTransform](#) (class in *tormentor*), 7

F

[Flip](#) (class in *tormentor*), 6
[forward_bboxes\(\)](#) (*tormentor.DeterministicImageAugmentation* method), 3
[forward_img\(\)](#) (*tormentor.DeterministicImageAugmentation* method), 4

[forward_img_path_probabilities\(\)](#) (*tormentor.DeterministicImageAugmentation* method), 4
[forward_mask\(\)](#) (*tormentor.DeterministicImageAugmentation* method), 4
[forward_pointcloud\(\)](#) (*tormentor.DeterministicImageAugmentation* method), 4
[forward_sampling_field\(\)](#) (*tormentor.DeterministicImageAugmentation* method), 5

G

[generate_batch_state\(\)](#) (*tormentor.DeterministicImageAugmentation* method), 5

H

[Hue](#) (class in *tormentor*), 8

I

[Invert](#) (class in *tormentor*), 7

L

[like_me\(\)](#) (*tormentor.DeterministicImageAugmentation* method), 5

P

[Perspective](#) (class in *tormentor*), 5
[PlasmaBrightness](#) (class in *tormentor*), 8
[PlasmaContrast](#) (class in *tormentor*), 9
[PlasmaRgbBrightness](#) (class in *tormentor*), 9
[PlasmaShadow](#) (class in *tormentor*), 9

R

[Rotate](#) (class in *tormentor*), 6

S

[Saturation](#) (class in *tormentor*), 8
[Scale](#) (class in *tormentor*), 6

ScaleTranslate (*class in tormentor*), 6

Shred (*class in tormentor*), 7, 9

SpatialImageAugmentation (*class in tormentor*), 5

StaticImageAugmentation (*class in tormentor*), 5

T

Translate (*class in tormentor*), 6

W

Wrap (*class in tormentor*), 7, 9

Z

Zoom (*class in tormentor*), 6