Table of Contents

TORCHVISION.TRANSFORMS

Transforms are common image transformations. They can be chained together using `Compose`. Additionally, there is the `torchvision.transforms.functional` module. Functional transforms give fine-grained control over the transformations. This is useful if you have to build a more complex transformation pipeline (e.g. in the case of segmentation tasks).

CLASS `torchvision.transforms.Compose(transforms)` [SOURCE]

Composes several transforms together.

Parameters

transforms (list of `Transform` objects) – list of transforms to compose.

Example

```
>>> transforms.Compose([
>>>     transforms.CenterCrop(10),
>>>     transforms.ToTensor(),
>>> ])
```

Transforms on PIL Image

CLASS `torchvision.transforms.CenterCrop(size)` [SOURCE]

Crops the given PIL Image at the center.

Parameters

size (*sequence or int*) – Desired output size of the crop. If size is an int instead of sequence like (h, w), a square crop (size, size) is made.

CLASS `torchvision.transforms.ColorJitter(brightness=0, contrast=0, saturation=0, hue=0)` [SOURCE]

Randomly change the brightness, contrast and saturation of an image.

Parameters

- **brightness** (*float or tuple of python:float (min, max)*) – How much to jitter brightness. brightness_factor is chosen uniformly from [max(0, 1 - brightness), 1 + brightness] or the given [min, max]. Should be non negative numbers.
- **contrast** (*float or tuple of python:float (min, max)*) – How much to jitter contrast. contrast_factor is chosen uniformly from [max(0, 1 - contrast), 1 + contrast] or the given [min, max]. Should be non negative numbers.
- **saturation** (*float or tuple of python:float (min, max)*) – How much to jitter saturation. saturation_factor is chosen uniformly from [max(0, 1 - saturation), 1 + saturation] or the given [min, max]. Should be non negative numbers.
- **hue** (*float or tuple of python:float (min, max)*) – How much to jitter hue. hue_factor is chosen uniformly from [-hue, hue] or the given [min, max]. Should have 0<= hue <= 0.5 or -0.5 <= min <= max <= 0.5.

CLASS `torchvision.transforms.FiveCrop(size)` [SOURCE]

Crop the given PIL Image into four corners and the central crop

• NOTE

This transform returns a tuple of images and there may be a mismatch in the number of inputs and targets your Dataset returns. See below for an example of how to deal with this.

Parameters

size (*sequence or int*) – Desired output size of the crop. If size is an `int` instead of sequence like (h, w), a square crop of size (size, size) is made.

Example

```
>>> transform = Compose([
>>>    FiveCrop(size), # this is a list of PIL Images
>>>    Lambda(lambda crops: torch.stack([ToTensor()(crop) for crop in crops])) # returns a 4D tensor
>>> ])
>>> #In your test loop you can do the following:
>>> input, target = batch # input is a 5d tensor, target is 2d
>>> bs, ncrops, c, h, w = input.size()
>>> result = model(input.view(-1, c, h, w)) # fuse batch size and ncrops
>>> result_avg = result.view(bs, ncrops, -1).mean(1) # avg over crops
```

CLASS `torchvision.transforms.Grayscale(num_output_channels=1)` [SOURCE]

Convert image to grayscale.

Parameters

num_output_channels (*int*) – (1 or 3) number of channels desired for output image

Returns

Grayscale version of the input. - If num_output_channels == 1 : returned image is single channel - If num_output_channels == 3 : returned image is 3 channel with r == g == b

Return type

PIL Image

CLASS `torchvision.transforms.Pad(padding, fill=0, padding_mode='constant')` [SOURCE]

Pad the given PIL Image on all sides with the given "pad" value.

Parameters

- **padding** (*int or tuple*) – Padding on each border. If a single int is provided this is used to pad all borders. If tuple of length 2 is provided this is the padding on left/right and top/bottom respectively. If a tuple of length 4 is provided this is the padding for the left, top, right and bottom borders respectively.
- **fill** (*int or tuple*) – Pixel fill value for constant fill. Default is 0. If a tuple of length 3, it is used to fill R, G, B channels respectively. This value is only used when the padding_mode is constant
- **padding_mode** (*str*) –

  Type of padding. Should be: constant, edge, reflect or symmetric. Default is constant.
  - constant: pads with a constant value, this value is specified with fill
  - edge: pads with the last value at the edge of the image
  - reflect: pads with reflection of image without repeating the last value on the edge

    For example, padding [1, 2, 3, 4] with 2 elements on both sides in reflect mode will result in [3, 2, 1, 2, 3, 4, 3, 2]

  - symmetric: pads with reflection of image repeating the last value on the edge

    For example, padding [1, 2, 3, 4] with 2 elements on both sides in symmetric mode will result in [2, 1, 1, 2, 3, 4, 4, 3]

CLASS `torchvision.transforms.RandomAffine(degrees, translate=None, scale=None, shear=None, resample=False, fillcolor=0)` [SOURCE]

Random affine transformation of the image keeping center invariant

Parameters

- **degrees** (*sequence or float or int*) – Range of degrees to select from. If degrees is a number instead of sequence like (min, max), the range of degrees will be (-degrees, +degrees). Set to 0 to deactivate rotations.
- **translate** (*tuple, optional*) – tuple of maximum absolute fraction for horizontal and vertical translations. For example translate=(a, b), then horizontal shift is randomly sampled in the range -img_width * a < dx < img_width * a and vertical shift is randomly sampled in the range -img_height * b < dy < img_height * b. Will not translate by default.
- **scale** (*tuple, optional*) – scaling factor interval, e.g (a, b), then scale is randomly sampled from the range a <= scale <= b. Will keep original scale by default.
- **shear** (*sequence or float or int, optional*) – Range of degrees to select from. If shear is a number, a shear parallel to the x axis in the range (-shear, +shear) will be apllied. Else if shear is a tuple or list of 2 values a shear parallel to the x axis in the range (shear[0], shear[1]) will be applied. Else if shear is a tuple or list of 4 values, a x-axis shear in (shear[0], shear[1]) and y-axis shear in (shear[2], shear[3]) will be applied. Will not apply shear by default
- **resample** ({*PIL.Image.NEAREST, PIL.Image.BILINEAR, PIL.Image.BICUBIC*}, *optional*) – An optional resampling filter. See filters for more information. If omitted, or if the image has mode "1" or "P", it is set to PIL.Image.NEAREST.
- **fillcolor** (*tuple or int*) – Optional fill color (Tuple for RGB Image And int for grayscale) for the area outside the transform in the output image.(Pillow>=5.0.0)

CLASS `torchvision.transforms.RandomApply(transforms, p=0.5)` [SOURCE]

Apply randomly a list of transformations with a given probability

Parameters

- **transforms** (*list or tuple*) – list of transformations
- **p** (*float*) – probability

CLASS `torchvision.transforms.RandomChoice(transforms)` [SOURCE]

Apply single transformation randomly picked from a list

CLASS `torchvision.transforms.RandomCrop(size, padding=None, pad_if_needed=False, fill=0, padding_mode='constant')` [SOURCE]

Crop the given PIL Image at a random location.

Parameters

- **size** (*sequence or int*) – Desired output size of the crop. If size is an int instead of sequence like (h, w), a square crop (size, size) is made.
- **padding** (*int or sequence, optional*) – Optional padding on each border of the image. Default is None, i.e no padding. If a sequence of length 4 is provided, it is used to pad left, top, right, bottom borders respectively. If a sequence of length 2 is provided, it is used to pad left/right, top/bottom borders, respectively.
- **pad_if_needed** (*boolean*) – It will pad the image if smaller than the desired size to avoid raising an exception. Since cropping is done after padding, the padding seems to be done at a random offset.
- **fill** – Pixel fill value for constant fill. Default is 0. If a tuple of length 3, it is used to fill R, G, B channels respectively. This value is only used when the padding_mode is constant
- **padding_mode** –

  Type of padding. Should be: constant, edge, reflect or symmetric. Default is constant.
  - constant: pads with a constant value, this value is specified with fill
  - edge: pads with the last value on the edge of the image
  - reflect: pads with reflection of image (without repeating the last value on the edge)

    padding [1, 2, 3, 4] with 2 elements on both sides in reflect mode will result in [3, 2, 1, 2, 3, 4, 3, 2]

  - symmetric: pads with reflection of image (repeating the last value on the edge)

    padding [1, 2, 3, 4] with 2 elements on both sides in symmetric mode will result in [2, 1, 1, 2, 3, 4, 4, 3]

CLASS `torchvision.transforms.RandomGrayscale(p=0.1)` [SOURCE]

Randomly convert image to grayscale with a probability of p (default 0.1).

Parameters

p (*float*) – probability that image should be converted to grayscale.

Returns

Grayscale version of the input image with probability p and unchanged with probability (1-p). - If input image is 1 channel: grayscale version is 1 channel - If input image is 3 channel: grayscale version is 3 channel with r == g == b

Return type

PIL Image

CLASS `torchvision.transforms.RandomHorizontalFlip(p=0.5)` [SOURCE]

Horizontally flip the given PIL Image randomly with a given probability.

Parameters

p (*float*) – probability of the image being flipped. Default value is 0.5

CLASS `torchvision.transforms.RandomOrder(transforms)` [SOURCE]

Apply a list of transformations in a random order

CLASS `torchvision.transforms.RandomPerspective(distortion_scale=0.5, p=0.5, interpolation=3)` [SOURCE]

Performs Perspective transformation of the given PIL Image randomly with a given probability.

Parameters

- **interpolation** – Default- Image.BICUBIC
- **p** (*float*) – probability of the image being perspectively transformed. Default value is 0.5
- **distortion_scale** (*float*) – it controls the degree of distortion and ranges from 0 to 1. Default value is 0.5.

CLASS `torchvision.transforms.RandomResizedCrop(size, scale=(0.08, 1.0), ratio=(0.75, 1.3333333333333333), interpolation=2)` [SOURCE]

Crop the given PIL Image to random size and aspect ratio.

A crop of random size (default: of 0.08 to 1.0) of the original size and a random aspect ratio (default: of 3/4 to 4/3) of the original aspect ratio is made. This crop is finally resized to given size. This is popularly used to train the Inception networks.

Parameters

- **size** – expected output size of each edge
- **scale** – range of size of the origin size cropped
- **ratio** – range of aspect ratio of the origin aspect ratio cropped
- **interpolation** – Default: PIL.Image.BILINEAR

CLASS `torchvision.transforms.RandomRotation(degrees, resample=False, expand=False, center=None)` [SOURCE]

Rotate the image by angle.

Parameters

- **degrees** (*sequence or float or int*) – Range of degrees to select from. If degrees is a number instead of sequence like (min, max), the range of degrees will be (-degrees, +degrees).
- **resample** ({*PIL.Image.NEAREST, PIL.Image.BILINEAR, PIL.Image.BICUBIC*}, *optional*) – An optional resampling filter. See filters for more information. If omitted, or if the image has mode “1” or “P”, it is set to PIL.Image.NEAREST.
- **expand** (*bool, optional*) – Optional expansion flag. If true, expands the output to make it large enough to hold the entire rotated image. If false or omitted, make the output image the same size as the input image. Note that the expand flag assumes rotation around the center and no translation.
- **center** (*2-tuple, optional*) – Optional center of rotation. Origin is the upper left corner. Default is the center of the image.

CLASS `torchvision.transforms.RandomSizedCrop(*args, **kwargs)` [SOURCE]

Note: This transform is deprecated in favor of RandomResizedCrop.

CLASS `torchvision.transforms.RandomVerticalFlip(p=0.5)` [SOURCE]

Vertically flip the given PIL Image randomly with a given probability.

Parameters

p (*float*) – probability of the image being flipped. Default value is 0.5

CLASS `torchvision.transforms.Resize(size, interpolation=2)` [SOURCE]

Resize the input PIL Image to the given size.

Parameters

- **size** (*sequence or int*) – Desired output size. If size is a sequence like (h, w), output size will be matched to this. If size is an int, smaller edge of the image will be matched to this number. i.e, if height > width, then image will be rescaled to (size * height / width, size)
- **interpolation** (*int, optional*) – Desired interpolation. Default is `PIL.Image.BILINEAR`

CLASS `torchvision.transforms.Scale(*args, **kwargs)` [SOURCE]

Note: This transform is deprecated in favor of Resize.

CLASS `torchvision.transforms.TenCrop(size, vertical_flip=False)` [SOURCE]

Crop the given PIL Image into four corners and the central crop plus the flipped version of these (horizontal flipping is used by default)

• NOTE

This transform returns a tuple of images and there may be a mismatch in the number of inputs and targets your Dataset returns. See below for an example of how to deal with this.

Parameters

- **size** (*sequence or int*) – Desired output size of the crop. If size is an int instead of sequence like (h, w), a square crop (size, size) is made.
- **vertical_flip** (*bool*) – Use vertical flipping instead of horizontal

Example

```
>>> transform = Compose([
>>>    TenCrop(size), # this is a list of PIL Images
>>>    Lambda(lambda crops: torch.stack([ToTensor()(crop) for crop in crops])) # returns a 4D tensor
>>> ])
>>> #In your test loop you can do the following:
>>> input, target = batch # input is a 5d tensor, target is 2d
>>> bs, ncrops, c, h, w = input.size()
>>> result = model(input.view(-1, c, h, w)) # fuse batch size and ncrops
>>> result_avg = result.view(bs, ncrops, -1).mean(1) # avg over crops
```

Transforms on torch.*Tensor

CLASS `torchvision.transforms.LinearTransformation(transformation_matrix, mean_vector)` [SOURCE]

Transform a tensor image with a square transformation matrix and a mean_vector computed offline. Given transformation_matrix and mean_vector, will flatten the torch.*Tensor and subtract mean_vector from it which is then followed by computing the dot product with the transformation matrix and then reshaping the tensor to its original shape.

Applications:

whitening transformation: Suppose X is a column vector zero-centered data. Then compute the data covariance matrix [D x D] with torch.mm(X.t(), X), perform SVD on this matrix and pass it as transformation_matrix.

Parameters

- **transformation_matrix** (*Tensor*) – tensor [D x D], D = C x H x W
- **mean_vector** (*Tensor*) – tensor [D], D = C x H x W

CLASS `torchvision.transforms.Normalize(mean, std, inplace=False)` [SOURCE]

Normalize a tensor image with mean and standard deviation. Given mean: `(M1,...,Mn)` and std: `(S1,..,Sn)` for `n` channels, this transform will normalize each channel of the input `torch.*Tensor` i.e. `input[channel] = (input[channel] - mean[channel]) / std[channel]`

• NOTE

This transform acts out of place, i.e., it does not mutates the input tensor.

Parameters

- **mean** (*sequence*) – Sequence of means for each channel.
- **std** (*sequence*) – Sequence of standard deviations for each channel.
- **inplace** (*bool,optional*) – Bool to make this operation in-place.

`__call__(tensor)` [SOURCE]

Parameters

tensor (*Tensor*) – Tensor image of size (C, H, W) to be normalized.

Returns

Normalized Tensor image.

Return type

Tensor

CLASS `torchvision.transforms.RandomErasing(p=0.5, scale=(0.02, 0.33), ratio=(0.3, 3.3), value=0, inplace=False)` [SOURCE]

Randomly selects a rectangle region in an image and erases its pixels.

'Random Erasing Data Augmentation' by Zhong et al. See https://arxiv.org/pdf/1708.04896.pdf

Parameters

- **p** – probability that the random erasing operation will be performed.
- **scale** – range of proportion of erased area against input image.
- **ratio** – range of aspect ratio of erased area.
- **value** – erasing value. Default is 0. If a single int, it is used to erase all pixels. If a tuple of length 3, it is used to erase R, G, B channels respectively. If a str of 'random', erasing each pixel with random values.
- **inplace** – boolean to make this transform inplace. Default set to False.

Returns

Erased Image.

Examples:

```
>>> transform = transforms.Compose([
>>> transforms.RandomHorizontalFlip(),
>>> transforms.ToTensor(),
>>> transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),
>>> transforms.RandomErasing(),
>>> ])
```

Conversion Transforms

CLASS `torchvision.transforms.ToPILImage(mode=None)` [SOURCE]

Convert a tensor or an ndarray to PIL Image.

Converts a torch.*Tensor of shape C x H x W or a numpy ndarray of shape H x W x C to a PIL Image while preserving the value range.

Parameters

mode (PIL.Image mode) –

color space and pixel depth of input data (optional). If `mode` is `None` (default) there are some assumptions made about the input data:

- If the input has 4 channels, the `mode` is assumed to be `RGBA`.
- If the input has 3 channels, the `mode` is assumed to be `RGB`.
- If the input has 2 channels, the `mode` is assumed to be `LA`.
- If the input has 1 channel, the `mode` is determined by the data type (i.e `int`, `float`, `short`).

`__call__(pic)` [SOURCE]

Parameters

pic (*Tensor or numpy.ndarray*) – Image to be converted to PIL Image.

Returns

Image converted to PIL Image.

Return type

PIL Image

CLASS `torchvision.transforms.ToTensor` [SOURCE]

Convert a `PIL Image` or `numpy.ndarray` to tensor.

Converts a PIL Image or numpy.ndarray (H x W x C) in the range [0, 255] to a torch.FloatTensor of shape (C x H x W) in the range [0.0, 1.0] if the PIL Image belongs to one of the modes (L, LA, P, I, F, RGB, YCbCr, RGBA, CMYK, 1) or if the numpy.ndarray has dtype = np.uint8

In the other cases, tensors are returned without scaling.

`__call__(pic)` [SOURCE]

Parameters

pic (*PIL Image or numpy.ndarray*) – Image to be converted to tensor.

Returns

Converted image.

Return type

Tensor

Generic Transforms

CLASS `torchvision.transforms.Lambda(lambd)` [SOURCE]

Apply a user-defined lambda as a transform.

Parameters

lambd (*function*) – Lambda/function to be used for transform.

Functional Transforms

Functional transforms give you fine-grained control of the transformation pipeline. As opposed to the transformations above, functional transforms don't contain a random number generator for their parameters. That means you have to specify/generate all parameters, but you can reuse the functional transform. For example, you can apply a functional transform to multiple images like this:

```
import torchvision.transforms.functional as TF
import random

def my_segmentation_transforms(image, segmentation):
    if random.random() > 0.5:
        angle = random.randint(-30, 30)
        image = TF.rotate(image, angle)
        segmentation = TF.rotate(segmentation, angle)
    # more transforms ...
    return image, segmentation
```

`torchvision.transforms.functional.adjust_brightness(img, brightness_factor)` [SOURCE]

Adjust brightness of an Image.

Parameters

- **img** (*PIL Image*) – PIL Image to be adjusted.
- **brightness_factor** (*float*) – How much to adjust the brightness. Can be any non negative number. 0 gives a black image, 1 gives the original image while 2 increases the brightness by a factor of 2.

Returns

Brightness adjusted image.

Return type

PIL Image

`torchvision.transforms.functional.adjust_contrast(img, contrast_factor)` [SOURCE]

Adjust contrast of an Image.

Parameters

- **img** (*PIL Image*) – PIL Image to be adjusted.
- **contrast_factor** (*float*) – How much to adjust the contrast. Can be any non negative number. 0 gives a solid gray image, 1 gives the original image while 2 increases the contrast by a factor of 2.

Returns

Contrast adjusted image.

Return type

PIL Image

`torchvision.transforms.functional.adjust_gamma(img, gamma, gain=1)` [SOURCE]

Perform gamma correction on an image.

Also known as Power Law Transform. Intensities in RGB mode are adjusted based on the following equation:

$$I_{\text{out}} = 255 \times \text{gain} \times \left(\frac{I_{\text{in}}}{255}\right)^{\gamma}$$

See Gamma Correction for more details.

Parameters

- **img** (*PIL Image*) – PIL Image to be adjusted.
- **gamma** (*float*) – Non negative real number, same as γ in the equation. gamma larger than 1 make the shadows darker, while gamma smaller than 1 make dark regions lighter.
- **gain** (*float*) – The constant multiplier.

`torchvision.transforms.functional.adjust_hue(img, hue_factor)` [SOURCE]

Adjust hue of an image.

The image hue is adjusted by converting the image to HSV and cyclically shifting the intensities in the hue channel (H). The image is then converted back to original image mode.

hue_factor is the amount of shift in H channel and must be in the interval [*-0.5*, *0.5*].

See Hue for more details.

Parameters

- **img** (*PIL Image*) – PIL Image to be adjusted.
- **hue_factor** (*float*) – How much to shift the hue channel. Should be in [-0.5, 0.5]. 0.5 and -0.5 give complete reversal of hue channel in HSV space in positive and negative direction respectively. 0 means no shift. Therefore, both -0.5 and 0.5 will give an image with complementary colors while 0 gives the original image.

Returns

Hue adjusted image.

Return type

PIL Image

`torchvision.transforms.functional.adjust_saturation(img, saturation_factor)` [SOURCE]

Adjust color saturation of an image.

Parameters

- **img** (*PIL Image*) – PIL Image to be adjusted.
- **saturation_factor** (*float*) – How much to adjust the saturation. 0 will give a black and white image, 1 will give the original image while 2 will enhance the saturation by a factor of 2.

Returns

Saturation adjusted image.

Return type

PIL Image

`torchvision.transforms.functional.affine(img, angle, translate, scale, shear, resample=0, fillcolor=None)` [SOURCE]

Apply affine transformation on the image keeping image center invariant

Parameters

- **img** (*PIL Image*) – PIL Image to be rotated.
- **angle** (*float or int*) – rotation angle in degrees between -180 and 180, clockwise direction.
- **translate** (*list or tuple of python:integers*) – horizontal and vertical translations (post-rotation translation)
- **scale** (*float*) – overall scale
- **shear** (*float or tuple or list*) – shear angle value in degrees between -180 to 180, clockwise direction.
- **a tuple of list is specified, the first value corresponds to a shear parallel to the x axis, while** (*If*) –
- **second value corresponds to a shear parallel to the y axis.** (*the*) –
- **resample** (`PIL.Image.NEAREST` or `PIL.Image.BILINEAR` or `PIL.Image.BICUBIC` , optional) – An optional resampling filter. See filters for more information. If omitted, or if the image has mode “1” or “P”, it is set to `PIL.Image.NEAREST` .
- **fillcolor** (*int*) – Optional fill color for the area outside the transform in the output image. (Pillow>=5.0.0)

`torchvision.transforms.functional.crop(img, i, j, h, w)` [SOURCE]

Crop the given PIL Image.

Parameters

- **img** (*PIL Image*) – Image to be cropped.
- **i** (*int*) – i in (i,j) i.e coordinates of the upper left corner.
- **j** (*int*) – j in (i,j) i.e coordinates of the upper left corner.
- **h** (*int*) – Height of the cropped image.
- **w** (*int*) – Width of the cropped image.

Returns

Cropped image.

Return type

PIL Image

PIL Image

`torchvision.transforms.functional.erase(img, i, j, h, w, v, inplace=False)` [SOURCE]

Erase the input Tensor Image with given value.

Parameters

- **img** (*Tensor Image*) – Tensor image of size (C, H, W) to be erased
- **i** (*int*) – i in (i,j) i.e coordinates of the upper left corner.
- **j** (*int*) – j in (i,j) i.e coordinates of the upper left corner.
- **h** (*int*) – Height of the erased region.
- **w** (*int*) – Width of the erased region.
- **v** – Erasing value.
- **inplace** (*bool, optional*) – For in-place operations. By default is set False.

Returns

Erased image.

Return type

Tensor Image

`torchvision.transforms.functional.five_crop(img, size)` [SOURCE]

Crop the given PIL Image into four corners and the central crop.

• NOTE

This transform returns a tuple of images and there may be a mismatch in the number of inputs and targets your `Dataset` returns.

Parameters

size (*sequence or int*) – Desired output size of the crop. If size is an int instead of sequence like (h, w), a square crop (size, size) is made.

Returns

tuple (tl, tr, bl, br, center)

Corresponding top left, top right, bottom left, bottom right and center crop.

Return type

tuple

`torchvision.transforms.functional.hflip(img)` [SOURCE]

Horizontally flip the given PIL Image.

Parameters

img (*PIL Image*) – Image to be flipped.

Returns

Horizontall flipped image.

Return type

PIL Image

`torchvision.transforms.functional.normalize(tensor, mean, std, inplace=False)` [SOURCE]

Normalize a tensor image with mean and standard deviation.

• NOTE

This transform acts out of place by default, i.e., it does not mutates the input tensor.

See `Normalize` for more details.

Parameters

- **tensor** (*Tensor*) – Tensor image of size (C, H, W) to be normalized.
- **mean** (*sequence*) – Sequence of means for each channel.
- **std** (*sequence*) – Sequence of standard deviations for each channel.
- **inplace** (*bool,optional*) – Bool to make this operation inplace.

Returns

Normalized Tensor image.

Return type

Tensor

`torchvision.transforms.functional.pad(img, padding, fill=0, padding_mode='constant')` [SOURCE]

Pad the given PIL Image on all sides with specified padding mode and fill value.

Parameters

- **img** (*PIL Image*) – Image to be padded.
- **padding** (*int or tuple*) – Padding on each border. If a single int is provided this is used to pad all borders. If tuple of length 2 is provided this is the padding on left/right and top/bottom respectively. If a tuple of length 4 is provided this is the padding for the left, top, right and bottom borders respectively.
- **fill** – Pixel fill value for constant fill. Default is 0. If a tuple of length 3, it is used to fill R, G, B channels respectively. This value is only used when the padding_mode is constant
- **padding_mode** –

  Type of padding. Should be: constant, edge, reflect or symmetric. Default is constant.
  - constant: pads with a constant value, this value is specified with fill
  - edge: pads with the last value on the edge of the image

- reflect: pads with reflection of image (without repeating the last value on the edge)

  padding [1, 2, 3, 4] with 2 elements on both sides in reflect mode will result in [3, 2, 1, 2, 3, 4, 3, 2]

- symmetric: pads with reflection of image (repeating the last value on the edge)

  padding [1, 2, 3, 4] with 2 elements on both sides in symmetric mode will result in [2, 1, 1, 2, 3, 4, 4, 3]

Returns

Padded image.

Return type

PIL Image

`torchvision.transforms.functional.perspective(img, startpoints, endpoints, interpolation=3)` [SOURCE]

Perform perspective transform of the given PIL Image.

Parameters

- **img** (*PIL Image*) – Image to be transformed.
- **startpoints** – List containing [top-left, top-right, bottom-right, bottom-left] of the orignal image
- **endpoints** – List containing [top-left, top-right, bottom-right, bottom-left] of the transformed image
- **interpolation** – Default- Image.BICUBIC

Returns

Perspectively transformed Image.

Return type

PIL Image

`torchvision.transforms.functional.resize(img, size, interpolation=2)` [SOURCE]

Resize the input PIL Image to the given size.

Parameters

- **img** (*PIL Image*) – Image to be resized.
- **size** (*sequence or int*) – Desired output size. If size is a sequence like (h, w), the output size will be matched to this. If size is an int, the smaller edge of the image will be matched to this number maintaing the aspect ratio. i.e, if height > width, then image will be rescaled to $\left(\text{size} \times \frac{\text{height}}{\text{width}}, \text{size}\right)$
- **interpolation** (*int, optional*) – Desired interpolation. Default is `PIL.Image.BILINEAR`

Returns

Resized image.

Return type

PIL Image

`torchvision.transforms.functional.resized_crop(img, i, j, h, w, size, interpolation=2)` [SOURCE]

Crop the given PIL Image and resize it to desired size.

Notably used in `RandomResizedCrop`.

Parameters

- **img** (*PIL Image*) – Image to be cropped.
- **i** (*int*) – i in (i,j) i.e coordinates of the upper left corner
- **j** (*int*) – j in (i,j) i.e coordinates of the upper left corner
- **h** (*int*) – Height of the cropped image.
- **w** (*int*) – Width of the cropped image.
- **size** (*sequence or int*) – Desired output size. Same semantics as `resize`.
- **interpolation** (*int, optional*) – Desired interpolation. Default is `PIL.Image.BILINEAR`.

Returns

Cropped image.

Return type

PIL Image

`torchvision.transforms.functional.rotate(img, angle, resample=False, expand=False, center=None)` [SOURCE]

Rotate the image by angle.

Parameters

- **img** (*PIL Image*) – PIL Image to be rotated.
- **angle** (*float or int*) – In degrees degrees counter clockwise order.
- **resample** (`PIL.Image.NEAREST` or `PIL.Image.BILINEAR` or `PIL.Image.BICUBIC`, optional) – An optional resampling filter. See filters for more information. If omitted, or if the image has mode “1” or “P”, it is set to `PIL.Image.NEAREST`.
- **expand** (*bool, optional*) – Optional expansion flag. If true, expands the output image to make it large enough to hold the entire rotated image. If false or omitted, make the output image the same size as the input image. Note that the expand flag assumes rotation around the center and no translation.
- **center** (*2-tuple, optional*) – Optional center of rotation. Origin is the upper left corner. Default is the center of the image.

`torchvision.transforms.functional.ten_crop(img, size, vertical_flip=False)` [SOURCE]

Crop the given PIL Image into four corners and the central crop plus the

flipped version of these (horizontal flipping is used by default).

NOTE

NOTE

This transform returns a tuple of images and there may be a mismatch in the number of inputs and targets your `Dataset` returns.

Parameters

- **size** (*sequence or int*) – Desired output size of the crop. If size is an int instead of sequence like (h, w), a square crop (size, size) is made.
- **vertical_flip** (*bool*) – Use vertical flipping instead of horizontal

Returns

tuple (tl, tr, bl, br, center, tl_flip, tr_flip, bl_flip, br_flip, center_flip)

Corresponding top left, top right, bottom left, bottom right and center crop and same for the flipped image.

Return type

tuple

`torchvision.transforms.functional.to_grayscale(img, num_output_channels=1)` [SOURCE]

Convert image to grayscale version of image.

Parameters

img (*PIL Image*) – Image to be converted to grayscale.

Returns

Grayscale version of the image.

if num_output_channels = 1 : returned image is single channel

if num_output_channels = 3 : returned image is 3 channel with r = g = b

Return type

PIL Image

`torchvision.transforms.functional.to_pil_image(pic, mode=None)` [SOURCE]

Convert a tensor or an ndarray to PIL Image.

See `ToPILImage` for more details.

Parameters

- **pic** (*Tensor or numpy.ndarray*) – Image to be converted to PIL Image.
- **mode** (PIL.Image mode) – color space and pixel depth of input data (optional).

Returns

Image converted to PIL Image.

Return type

PIL Image

`torchvision.transforms.functional.to_tensor(pic)` [SOURCE]

Convert a `PIL Image` or `numpy.ndarray` to tensor.

See `ToTensor` for more details.

Parameters

pic (*PIL Image or numpy.ndarray*) – Image to be converted to tensor.

Returns

Converted image.

Return type

Tensor

`torchvision.transforms.functional.vflip(img)` [SOURCE]

Vertically flip the given PIL Image.

Parameters

img (*PIL Image*) – Image to be flipped.

Returns

Vertically flipped image.

Return type

PIL Image

‹ Previous Next ›