Package 'torchvision'

July 18, 2025

Title Models, Datasets and Transformations for Images

Version 0.7.0

Description Provides access to datasets, models and preprocessing facilities for deep learning with images. Integrates seamlessly with the 'torch' package and it's 'API' borrows heavily from 'PyTorch' vision package.

License MIT + file LICENSE

Encoding UTF-8

URL https://torchvision.mlverse.org, https://github.com/mlverse/torchvision

RoxygenNote 7.3.2

Imports torch (>= 0.5.0), fs, rlang, rappdirs, utils, jpeg, tiff, magrittr, png, abind, jsonlite, withr, cli, glue, zeallot

Suggests magick, testthat, coro, R.matlab

BugReports https://github.com/mlverse/torchvision/issues

NeedsCompilation no

Author Daniel Falbel [aut, cre], Christophe Regouby [ctb], Akanksha Koshti [ctb], Derrick Richard [ctb], RStudio [cph]

Maintainer Daniel Falbel <daniel@posit.co>

Repository CRAN

Date/Publication 2025-07-18 16:20:02 UTC

Contents

`base_loader` *Base loader*

Description

Loads an image using `jpeg`, `png` or `tiff` packages depending on the file extension.

Usage

```
base_loader(path)
```

Arguments

`path`	path to the image to load from

batched_nms *Batched Non-maximum Suppression (NMS)*

Description

Performs non-maximum suppression in a batched fashion. Each index value correspond to a category, and NMS will not be applied between elements of different categories.

Usage

```
batched_nms(boxes, scores, idxs, iou_threshold)
```

Arguments

`boxes`	(Tensor[N, 4]): boxes where NMS will be performed. They are expected to be in $(x_{min}, y_{min}, x_{max}, y_{max})$ format with • $0 \leq x_{min} < x_{max}$ and • $0 \leq y_{min} < y_{max}$.
`scores`	(Tensor[N]): scores for each one of the boxes
`idxs`	(Tensor[N]): indices of the categories for each one of the boxes.
`iou_threshold`	(float): discards all overlapping boxes with IoU > `iou_threshold`

Value

keep (Tensor): int64 tensor with the indices of the elements that have been kept by NMS, sorted in decreasing order of scores

box_area *Box Area*

Description

Computes the area of a set of bounding boxes, which are specified by its $(x_{min}, y_{min}, x_{max}, y_{max})$ coordinates.

Usage

```
box_area(boxes)
```

Arguments

`boxes`	(Tensor[N, 4]): boxes for which the area will be computed. They are expected to be in $(x_{min}, y_{min}, x_{max}, y_{max})$ format with • $0 \leq x_{min} < x_{max}$ and • $0 \leq y_{min} < y_{max}$.

Value

area (Tensor[N]): area for each box

`box_convert` *Box Convert*

Description

Converts boxes from given in_fmt to out_fmt.

Usage

```
box_convert(boxes, in_fmt, out_fmt)
```

Arguments

`boxes`	(Tensor[N, 4]): boxes which will be converted.
`in_fmt`	(str): Input format of given boxes. Supported formats are ['xyxy', 'xywh', 'cxcywh'].
`out_fmt`	(str): Output format of given boxes. Supported formats are ['xyxy', 'xywh', 'cxcywh']

Details

Supported in_fmt and out_fmt are:

- 'xyxy': boxes are represented via corners,
  - x_{min}, y_{min} being top left and
  - x_{max}, y_{max} being bottom right.
- 'xywh' : boxes are represented via corner, width and height,
  - x_{min}, y_{min} being top left,
  - w, h being width and height.
- 'cxcywh' : boxes are represented via centre, width and height,
  - c_x, c_y being center of box,
  - w, h being width and height.

Value

boxes (Tensor[N, 4]): Boxes into converted format.

box_cxcywh_to_xyxy *box_cxcywh_to_xyxy*

Description

Converts bounding boxes from (c_x, c_y, w, h) format to $(x_{min}, y_{min}, x_{max}, y_{max})$ format. (c_x, c_y) refers to center of bounding box (w, h) are width and height of bounding box

Usage

```
box_cxcywh_to_xyxy(boxes)
```

Arguments

boxes	(Tensor[N, 4]): boxes in (c_x, c_y, w, h) format which will be converted.

Value

boxes (Tensor(N, 4)): boxes in $(x_{min}, y_{min}, x_{max}, y_{max})$ format.

box_iou *Box IoU*

Description

Return intersection-over-union (Jaccard index) of boxes. Both sets of boxes are expected to be in $(x_{min}, y_{min}, x_{max}, y_{max})$ format with $0 \leq x_{min} < x_{max}$ and $0 \leq y_{min} < y_{max}$.

Usage

```
box_iou(boxes1, boxes2)
```

Arguments

boxes1	(Tensor[N, 4])
boxes2	(Tensor[M, 4])

Value

iou (Tensor[N, M]): the NxM matrix containing the pairwise IoU values for every element in boxes1 and boxes2

box_xywh_to_xyxy *box_xywh_to_xyxy*

Description

Converts bounding boxes from (x, y, w, h) format to $(x_{min}, y_{min}, x_{max}, y_{max})$ format. (x, y) refers to top left of bouding box. (w, h) refers to width and height of box.

Usage

```
box_xywh_to_xyxy(boxes)
```

Arguments

boxes	(Tensor[N, 4]): boxes in (x, y, w, h) which will be converted.

Value

boxes (Tensor[N, 4]): boxes in $(x_{min}, y_{min}, x_{max}, y_{max})$ format.

box_xyxy_to_cxcywh *box_xyxy_to_cxcywh*

Description

Converts bounding boxes from $(x_{min}, y_{min}, x_{max}, y_{max})$ format to (c_x, c_y, w, h) format. (x1, y1) refer to top left of bounding box (x2, y2) refer to bottom right of bounding box

Usage

```
box_xyxy_to_cxcywh(boxes)
```

Arguments

boxes	(Tensor[N, 4]): boxes in $(x_{min}, y_{min}, x_{max}, y_{max})$ format which will be converted.

Value

boxes (Tensor(N, 4)): boxes in (c_x, c_y, w, h) format.

box_xyxy_to_xywh *box_xyxy_to_xywh*

Description

Converts bounding boxes from $(x_{min}, y_{min}, x_{max}, y_{max})$ format to (x, y, w, h) format. (x1, y1) refer to top left of bounding box (x2, y2) refer to bottom right of bounding box

Usage

```
box_xyxy_to_xywh(boxes)
```

Arguments

boxes	(Tensor[N, 4]): boxes in $(x_{min}, y_{min}, x_{max}, y_{max})$ which will be converted.

Value

boxes (Tensor[N, 4]): boxes in (x, y, w, h) format.

caltech_dataset *Caltech Datasets*

Description

Caltech Datasets

Loads the Caltech-256 Object Category Dataset for image classification. It consists of 30,607 images across 256 distinct object categories. Each category has at least 80 images, with variability in image size.

Usage

```
caltech101_dataset(
  root = tempdir(),
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)

caltech256_dataset(
  root = tempdir(),
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)
```

Arguments

root	Character. Root directory for dataset storage. The dataset will be stored under root/caltech256.
transform	Optional function to transform input images after loading. Default is NULL.
target_transform	Optional function to transform labels. Default is NULL.
download	Logical. Whether to download the dataset if not found locally. Default is FALSE.

Details

The Caltech-101 and Caltech-256 collections are **classification** datasets made of color images with varying sizes. They cover 101 and 256 object categories respectively and are commonly used for evaluating visual recognition models.

The Caltech-101 dataset contains around 9,000 images spread over 101 object categories plus a background class. Images have varying sizes.

Caltech-256 extends this to about 30,000 images across 256 categories.

Value

An object of class caltech101_dataset, which behaves like a torch dataset. Each element is a named list with:

- x: A H x W x 3 integer array representing an RGB image.
- y: An Integer representing the label.

An object of class caltech256_dataset, which behaves like a torch dataset. Each element is a named list with:

- x: A H x W x 3 integer array representing an RGB image.
- y: An Integer representing the label.

See Also

Other classification_dataset: cifar10_dataset(), eurosat_dataset(), fer_dataset(), fgvc_aircraft_dataset(), flowers102_dataset(), mnist_dataset(), oxfordiiitpet_dataset(), tiny_imagenet_dataset()

Examples

```
## Not run:
caltech101 <- caltech101_dataset(download = TRUE)

first_item <- caltech101[1]
first_item$x  # Image array
first_item$y  # Integer label

## End(Not run)
```

`cifar10_dataset` *CIFAR datasets*

Description

The CIFAR datasets are benchmark **classification** datasets composed of 60,000 RGB thumbnail images of size 32x32 pixels. The CIFAR10 variant contains 10 classes while CIFAR100 provides 100 classes. Images are split into 50,000 training samples and 10,000 test samples.

Downloads and prepares the CIFAR100 dataset.

Usage

```
cifar10_dataset(
  root = tempdir(),
  train = TRUE,
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)

cifar100_dataset(
  root = tempdir(),
  train = TRUE,
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)
```

Arguments

`root`	(string): Root directory of dataset where directory `cifar-10-batches-bin` exists or will be saved to if download is set to TRUE.
`train`	Logical. If TRUE, use the training set; otherwise, use the test set. Not applicable to all datasets.
`transform`	Optional. A function that takes an image and returns a transformed version (e.g., normalization, cropping).
`target_transform`	Optional. A function that transforms the label.
`download`	Logical. If TRUE, downloads the dataset to `root/`. If the dataset is already present, download is skipped.

Details

Downloads and prepares the CIFAR archives.

Value

A torch::dataset object. Each item is a list with:

- x: a 32x32x3 integer array
- y: the class label

See Also

Other classification_dataset: caltech_dataset, eurosat_dataset(), fer_dataset(), fgvc_aircraft_dataset(), flowers102_dataset(), mnist_dataset(), oxfordiiitpet_dataset(), tiny_imagenet_dataset()

Examples

```
## Not run:
ds <- cifar10_dataset(root = tempdir(), download = TRUE)
item <- ds[1]
item$x
item$y

## End(Not run)
```

---

clip_boxes_to_image *Clip Boxes to Image*

Description

Clip boxes so that they lie inside an image of size size.

Usage

```
clip_boxes_to_image(boxes, size)
```

Arguments

boxes	(Tensor[N, 4]): boxes in $(x_{min}, y_{min}, x_{max}, y_{max})$ format with • $0 \leq x_{min} < x_{max}$ and • $0 \leq y_{min} < y_{max}$.
size	(Tuple[height, width]): size of the image

Value

clipped_boxes (Tensor[N, 4])

`coco_caption_dataset` *COCO Caption Dataset*

Description

Loads the MS COCO dataset for image captioning.

Usage

```
coco_caption_dataset(
  root = tempdir(),
  train = TRUE,
  year = c("2014"),
  download = FALSE,
  transform = NULL,
  target_transform = NULL
)
```

Arguments

`root`	Root directory where the dataset is stored or will be downloaded to.
`train`	Logical. If TRUE, loads the training split; otherwise, loads the validation split.
`year`	Character. Dataset version year. One of `"2014"`.
`download`	Logical. If TRUE, downloads the dataset if it's not already present in the `root` directory.
`transform`	Optional transform function applied to the image.
`target_transform`	Optional transform function applied to the target (labels, boxes, etc.).

Value

An object of class `coco_caption_dataset`. Each item is a list:

- `x`: an `(H, W, C)` numeric array containing the RGB image.
- `y`: a character string with the image caption.

See Also

Other caption_dataset: `flickr_caption_dataset`

Examples

```
## Not run:
ds <- coco_caption_dataset(
  train = FALSE,
  download = TRUE
)
example <- ds[1]

# Access image and caption
x <- example$x
y <- example$y

# Prepare image for plotting
image_array <- as.numeric(x)
dim(image_array) <- dim(x)

plot(as.raster(image_array))
title(main = y, col.main = "black")

## End(Not run)
```

`coco_detection_dataset` *COCO Detection Dataset*

Description

Loads the MS COCO dataset for object detection and segmentation.

Usage

```
coco_detection_dataset(
  root = tempdir(),
  train = TRUE,
  year = c("2017", "2014"),
  download = FALSE,
  transform = NULL,
  target_transform = NULL
)
```

Arguments

`root`	Root directory where the dataset is stored or will be downloaded to.
`train`	Logical. If TRUE, loads the training split; otherwise, loads the validation split.
`year`	Character. Dataset version year. One of `"2014"` or `"2017"`.
`download`	Logical. If TRUE, downloads the dataset if it's not already present in the `root` directory.

`transform` Optional transform function applied to the image.

`target_transform` Optional transform function applied to the target (labels, boxes, etc.).

Details

The returned image is in CHW format (channels, height, width), matching the torch convention. The dataset y offers object detection annotations such as bounding boxes, labels, areas, crowd indicators, and segmentation masks from the official COCO annotations.

Value

An object of class `coco_detection_dataset`. Each item is a list:

- `x`: a `(C, H, W)` `torch_tensor` representing the image.
- `y$boxes`: a `(N, 4)` `torch_tensor` of bounding boxes in the format `c(x_min, y_min, x_max, y_max)`.
- `y$labels`: an integer `torch_tensor` with the class label for each object.
- `y$area`: a float `torch_tensor` indicating the area of each object.
- `y$iscrowd`: a boolean `torch_tensor`, where `TRUE` marks the object as part of a crowd.
- `y$segmentation`: a list of segmentation polygons for each object.
- `y$masks`: a `(N, H, W)` boolean `torch_tensor` containing binary segmentation masks.

The returned object has S3 classes `"image_with_bounding_box"` and `"image_with_segmentation_mask"` to enable automatic dispatch by visualization functions such as `draw_bounding_boxes()` and `draw_segmentation_masks()`

Examples

```
## Not run:
ds <- coco_detection_dataset(
  train = FALSE,
  year = "2017",
  download = TRUE
)

item <- ds[1]

# Visualize bounding boxes
boxed <- draw_bounding_boxes(item)
tensor_image_browse(boxed)

# Visualize segmentation masks (if present)
masked <- draw_segmentation_masks(item)
tensor_image_browse(masked)

## End(Not run)
```

`draw_bounding_boxes` *Draws bounding boxes on image.*

Description

Draws bounding boxes on top of one image tensor

Usage

```
draw_bounding_boxes(x, ...)

## Default S3 method:
draw_bounding_boxes(x, ...)

## S3 method for class 'torch_tensor'
draw_bounding_boxes(
  x,
  boxes,
  labels = NULL,
  colors = NULL,
  fill = FALSE,
  width = 1,
  font = c("serif", "plain"),
  font_size = 10,
  ...
)

## S3 method for class 'image_with_bounding_box'
draw_bounding_boxes(x, ...)
```

Arguments

`x`	Tensor of shape (C x H x W) and dtype `uint8` or dtype `float`. In case of dtype float, values are assumed to be in range $[0, 1]$. C value for channel can only be 1 (grayscale) or 3 (RGB).
`...`	Additional arguments passed to methods.
`boxes`	Tensor of size (N, 4) containing N bounding boxes in c(x_{min}, y_{min}, x_{max}, y_{max}). format. Note that the boxes coordinates are absolute with respect to the image. In other words: $0 \leq x_{min} < x_{max} < W$ and $0 \leq y_{min} < y_{max} < W$.
`labels`	character vector containing the labels of bounding boxes.
`colors`	character vector containing the colors of the boxes or single color for all boxes. The color can be represented as strings e.g. "red" or "#FF00FF". By default, viridis colors are generated for boxes.
`fill`	If `TRUE` fills the bounding box with specified color.
`width`	Width of text shift to the bounding box.

font	NULL for the current font family, or a character vector of length 2 for Hershey vector fonts.
font_size	The requested font size in points.

Value

torch_tensor of size (C, H, W) of dtype uint8: Image Tensor with bounding boxes plotted.

See Also

Other image display: draw_keypoints(), draw_segmentation_masks(), tensor_image_browse(), tensor_image_display(), vision_make_grid()

Examples

```
if (torch::torch_is_installed()) {
## Not run:
image_tensor <- torch::torch_randint(170, 250, size = c(3, 360, 360))$to(torch::torch_uint8())
x <- torch::torch_randint(low = 1, high = 160, size = c(12,1))
y <- torch::torch_randint(low = 1, high = 260, size = c(12,1))
boxes <- torch::torch_cat(c(x, y, x + 20, y +  10), dim = 2)
bboxed <- draw_bounding_boxes(image_tensor, boxes, colors = "black", fill = TRUE)
tensor_image_browse(bboxed)

## End(Not run)
}
```

---

draw_keypoints *Draws Keypoints*

Description

Draws Keypoints, an object describing a body part (like rightArm or leftShoulder), on given RGB tensor image.

Usage

```
draw_keypoints(
  image,
  keypoints,
  connectivity = NULL,
  colors = NULL,
  radius = 2,
  width = 3
)
```

Arguments

`image`	Tensor of shape (3 x H x W) and dtype `uint8` or dtype `float`. In case of dtype float, values are assumed to be in range $[0, 1]$.
`keypoints`	Tensor of shape (N, K, 2) the K keypoints location for each of the N detected poses instance,
`connectivity`	Vector of pair of keypoints to be connected (currently unavailable)
`colors`	character vector containing the colors of the boxes or single color for all boxes. The color can be represented as strings e.g. "red" or "#FF00FF". By default, viridis colors are generated for keypoints
`radius`	radius of the plotted keypoint.
`width`	width of line connecting keypoints.

Value

Image Tensor of dtype uint8 with keypoints drawn.

See Also

Other image display: `draw_bounding_boxes()`, `draw_segmentation_masks()`, `tensor_image_browse()`, `tensor_image_display()`, `vision_make_grid()`

Examples

```
if (torch::torch_is_installed()) {
## Not run:
image <- torch::torch_randint(190, 255, size = c(3, 360, 360))$to(torch::torch_uint8())
keypoints <- torch::torch_randint(low = 60, high = 300, size = c(4, 5, 2))
keypoint_image <- draw_keypoints(image, keypoints)
tensor_image_browse(keypoint_image)

## End(Not run)
}
```

`draw_segmentation_masks` *Draw segmentation masks*

Description

Draw segmentation masks with their respective colors on top of a given RGB tensor image

Usage

```
draw_segmentation_masks(x, ...)

## Default S3 method:
draw_segmentation_masks(x, ...)

## S3 method for class 'torch_tensor'
draw_segmentation_masks(x, masks, alpha = 0.8, colors = NULL, ...)

## S3 method for class 'image_with_segmentation_mask'
draw_segmentation_masks(x, alpha = 0.5, colors = NULL, ...)
```

Arguments

x	Tensor of shape (C x H x W) and dtype `uint8` or dtype `float`. In case of dtype float, values are assumed to be in range $[0, 1]$. C value for channel can only be 1 (grayscale) or 3 (RGB).
...	Additional arguments passed to methods.
masks	torch_tensor of shape (num_masks, H, W) or (H, W) and dtype bool.
alpha	number between 0 and 1 denoting the transparency of the masks.
colors	character vector containing the colors of the boxes or single color for all boxes. The color can be represented as strings e.g. "red" or "#FF00FF". By default, viridis colors are generated for masks

Value

torch_tensor of shape (3, H, W) and dtype uint8 of the image with segmentation masks drawn on top.

See Also

Other image display: `draw_bounding_boxes()`, `draw_keypoints()`, `tensor_image_browse()`, `tensor_image_display()`, `vision_make_grid()`

Examples

```
if (torch::torch_is_installed()) {
image_tensor <- torch::torch_randint(170, 250, size = c(3, 360, 360))$to(torch::torch_uint8())
mask <- torch::torch_tril(torch::torch_ones(c(360, 360)))$to(torch::torch_bool())
masked_image <- draw_segmentation_masks(image_tensor, mask, alpha = 0.2)
tensor_image_browse(masked_image)
}
```

`eurosat_dataset` *EuroSAT datasets*

Description

A collection of Sentinel-2 satellite images for land-use **classification**. The standard version contains 27,000 RGB thumbnails (64x64) across 10 classes. Variants include the full 13 spectral bands and a small 100-image subset useful for demos.

Downloads and prepares the EuroSAT dataset with 13 spectral bands.

A subset of 100 images with 13 spectral bands useful for workshops and demos.

Usage

```
eurosat_dataset(
  root = tempdir(),
  split = "train",
  download = FALSE,
  transform = NULL,
  target_transform = NULL
)

eurosat_all_bands_dataset(
  root = tempdir(),
  split = "train",
  download = FALSE,
  transform = NULL,
  target_transform = NULL
)

eurosat100_dataset(
  root = tempdir(),
  split = "train",
  download = FALSE,
  transform = NULL,
  target_transform = NULL
)
```

Arguments

`root`	(Optional) Character. The root directory where the dataset will be stored. if empty, will use the default `rappdirs::user_cache_dir("torch")`.
`split`	Character. Must be one of `train`, `val`, or `test`.
`download`	Logical. If TRUE, downloads the dataset to `root/`. If the dataset is already present, download is skipped.
`transform`	Optional. A function that takes an image and returns a transformed version (e.g., normalization, cropping).

target_transform
: Optional. A function that transforms the label.

Details

`eurosat_dataset()` provides a total of 27,000 RGB labeled images.

`eurosat_all_bands_dataset()` provides a total of 27,000 labeled images with 13 spectral channel bands.

`eurosat100_dataset()` provides a subset of 100 labeled images with 13 spectral channel bands.

Value

A `torch::dataset` object. Each item is a list with:

- `x`: a 64x64 image tensor with 3 (RGB) or 13 (all bands) channels
- `y`: the class label

See Also

Other classification_dataset: `caltech_dataset`, `cifar10_dataset()`, `fer_dataset()`, `fgvc_aircraft_dataset()`, `flowers102_dataset()`, `mnist_dataset()`, `oxfordiiitpet_dataset()`, `tiny_imagenet_dataset()`

Examples

```
## Not run:
# Initialize the dataset
ds <- eurosat100_dataset(split = "train", download = TRUE)

# Access the first item
head <- ds[1]
print(head$x) # Image
print(head$y) # Label

## End(Not run)
```

---

fer_dataset *FER-2013 Facial Expression Dataset*

Description

Loads the FER-2013 dataset for facial expression recognition. The dataset contains grayscale images (48x48) of human faces, each labeled with one of seven emotion categories: `"Angry"`, `"Disgust"`, `"Fear"`, `"Happy"`, `"Sad"`, `"Surprise"`, and `"Neutral"`.

Usage

```
fer_dataset(
  root = tempdir(),
  train = TRUE,
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)
```

Arguments

root	(string, optional): Root directory for dataset storage, the dataset will be stored under root/fer2013.
train	Logical. If TRUE, use the training set; otherwise, use the test set. Not applicable to all datasets.
transform	Optional. A function that takes an image and returns a transformed version (e.g., normalization, cropping).
target_transform	Optional. A function that transforms the label.
download	Logical. If TRUE, downloads the dataset to root/. If the dataset is already present, download is skipped.

Details

The dataset is split into:

- "Train": training images labeled as "Training" in the original CSV.
- "Test": includes both "PublicTest" and "PrivateTest" entries.

Value

A torch dataset of class fer_dataset. Each element is a named list:

- x: a 48x48 grayscale array
- y: an integer from 1 to 7 indicating the class index

See Also

Other classification_dataset: caltech_dataset, cifar10_dataset(), eurosat_dataset(), fgvc_aircraft_dataset(), flowers102_dataset(), mnist_dataset(), oxfordiiitpet_dataset(), tiny_imagenet_dataset()

Examples

```
## Not run:
fer <- fer_dataset(train = TRUE, download = TRUE)
first_item <- fer[1]
first_item$x  # 48x48 grayscale array
first_item$y  # 4
```

```
fer$classes[first_item$y]  # "Happy"

## End(Not run)
```

`fgvc_aircraft_dataset` *FGVC Aircraft Dataset*

Description

The FGVC-Aircraft dataset supports the following official splits:

- `"train"`: training subset with labels.
- `"val"`: validation subset with labels.
- `"trainval"`: combined training and validation set with labels.
- `"test"`: test set with labels (used for evaluation).

Usage

```
fgvc_aircraft_dataset(
  root = tempdir(),
  split = "train",
  annotation_level = "variant",
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)
```

Arguments

`root`	Character. Root directory for dataset storage. The dataset will be stored under `root/fgvc-aircraft-2013b`.
`split`	Character. One of `"train"`, `"val"`, `"trainval"`, or `"test"`. Default is `"train"`.
`annotation_level`	Character. Level of annotation to use for classification. Default is `"variant"`. One of `"variant"`, `"family"`, `"manufacturer"`, or `"all"`. See *Details*.
`transform`	Optional function to transform input images after loading. Default is `NULL`.
`target_transform`	Optional function to transform labels. Default is `NULL`.
`download`	Logical. Whether to download the dataset if not found locally. Default is `FALSE`.

Details

The annotation_level determines the granularity of labels used for classification and supports four values:

- "variant": the most fine-grained level, e.g., "Boeing 737-700". There are 100 visually distinguishable variants.
- "family": a mid-level grouping, e.g., "Boeing 737", which includes multiple variants. There are 70 distinct families.
- "manufacturer": the coarsest level, e.g., "Boeing", grouping multiple families under a single manufacturer. There are 30 manufacturers.
- "all": multi-label format that returns all three levels as a vector of class indices c(manufacturer_idx, family_idx, variant_idx).

These levels form a strict hierarchy: each "manufacturer" consists of multiple "families", and each "family" contains several "variants". Not all combinations of levels are valid — for example, a "variant" always belongs to exactly one "family", and a "family" to exactly one "manufacturer".

When annotation_level = "all" is used, the $classes field is a named list with three components:

- classes$manufacturer: a character vector of manufacturer names
- classes$family: a character vector of family names
- classes$variant: a character vector of variant names

Value

An object of class fgvc_aircraft_dataset, which behaves like a torch-style dataset. Each element is a named list with:

- x: an array of shape (H, W, C) with pixel values in the range (0, 255). Please note that images have varying sizes.
- y: for single-level annotation ("variant", "family", "manufacturer"): an integer class label. for multi-level annotation ("all"): a vector of three integers c(manufacturer_idx, family_idx, variant_idx).

See Also

Other classification_dataset: caltech_dataset, cifar10_dataset(), eurosat_dataset(), fer_dataset(), flowers102_dataset(), mnist_dataset(), oxfordiiitpet_dataset(), tiny_imagenet_dataset()

Examples

```
## Not run:
# Single-label classification
fgvc <- fgvc_aircraft_dataset(transform = transform_to_tensor, download = TRUE)

# Create a custom collate function to resize images and prepare batches
resize_collate_fn <- function(batch) {
```

```
  xs <- lapply(batch, function(item) {
    torchvision::transform_resize(item$x, c(768, 1024))
  })
  xs <- torch::torch_stack(xs)
 ys <- torch::torch_tensor(sapply(batch, function(item) item$y), dtype = torch::torch_long())
  list(x = xs, y = ys)
}
dl <- torch::dataloader(dataset = fgvc, batch_size = 2, collate_fn = resize_collate_fn)
batch <- dataloader_next(dataloader_make_iter(dl))
batch$x  # batched image tensors with shape (2, 3, 768, 1024)
batch$y  # class labels as integer tensor of shape 2

# Multi-label classification
fgvc <- fgvc_aircraft_dataset(split = "test", annotation_level = "all")
item <- fgvc[1]
item$x  # a double vector representing the image
item$y  # an integer vector of length 3: manufacturer, family, and variant indices
fgvc$classes$manufacturer[item$y[1]]  # e.g., "Boeing"
fgvc$classes$family[item$y[2]]        # e.g., "Boeing 707"
fgvc$classes$variant[item$y[3]]       # e.g., "707-320"

## End(Not run)
```

`flickr_caption_dataset` *Flickr Caption Datasets*

Description

Flickr8k Dataset

Usage

```
flickr8k_caption_dataset(
  root = tempdir(),
  train = TRUE,
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)

flickr30k_caption_dataset(
  root = tempdir(),
  train = TRUE,
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)
```

Arguments

root	Character. Root directory where the dataset will be stored under root/flickr30k.
train	: If TRUE, loads the training set. If FALSE, loads the test set. Default is TRUE.
transform	Optional function to transform input images after loading. Default is NULL.
target_transform	Optional function to transform labels. Default is NULL.
download	Logical. Whether to download the dataset if not found locally. Default is FALSE.

Details

The Flickr8k and Flickr30k collections are **image captionning** datasets composed of 8,000 and 30,000 color images respectively, each paired with five human-annotated captions. The images are in RGB format with varying spatial resolutions, and these datasets are widely used for training and evaluating vision-language models.

Value

A torch dataset of class flickr8k_caption_dataset. Each element is a named list:

- x: a H x W x 3 integer array representing an RGB image.
- y: a character vector containing all five captions associated with the image.

A torch dataset of class flickr30k_caption_dataset. Each element is a named list:

- x: a H x W x 3 integer array representing an RGB image.
- y: a character vector containing all five captions associated with the image.

See Also

Other caption_dataset: coco_caption_dataset()

Examples

```
## Not run:
# Load the Flickr8k caption dataset
flickr8k <- flickr8k_caption_dataset(download = TRUE)

# Access the first item
first_item <- flickr8k[1]
first_item$x  # image array with shape {3, H, W}
first_item$y  # character vector containing five captions.

# Load the Flickr30k caption dataset
flickr30k <- flickr30k_caption_dataset(download = TRUE)

# Access the first item
first_item <- flickr30k[1]
first_item$x  # image array with shape {3, H, W}
first_item$y  # character vector containing five captions.
```

```
## End(Not run)
```

`flowers102_dataset` *Oxford Flowers 102 Dataset*

Description

Loads the Oxford 102 Category Flower Dataset. This dataset consists of 102 flower categories, with between 40 and 258 images per class. Images in this dataset are of variable sizes.

Usage

```
flowers102_dataset(
  root = tempdir(),
  split = "train",
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)
```

Arguments

`root`	Root directory for dataset storage. The dataset will be stored under `root/flowers102`.
`split`	One of `"train"`, `"val"`, or `"test"`. Default is `"train"`.
`transform`	Optional function to transform input images after loading. Default is `NULL`.
`target_transform`	Optional function to transform labels. Default is `NULL`.
`download`	Logical. Whether to download the dataset if not found locally. Default is `FALSE`.

Details

This is a **classification** dataset where the goal is to assign each image to one of the 102 flower categories.

The dataset is split into:

- `"train"`: training subset with labels.
- `"val"`: validation subset with labels.
- `"test"`: test subset with labels (used for evaluation).

Value

An object of class `flowers102_dataset`, which behaves like a torch dataset. Each element is a named list:

- `x`: a W x H x 3 numeric array representing an RGB image.
- `y`: an integer label indicating the class index.

See Also

Other classification_dataset: caltech_dataset, cifar10_dataset(), eurosat_dataset(), fer_dataset(), fgvc_aircraft_dataset(), mnist_dataset(), oxfordiiitpet_dataset(), tiny_imagenet_dataset()

Examples

```
## Not run:
# Load the dataset with inline transforms
flowers <- flowers102_dataset(
  split = "train",
  download = TRUE,
  transform = . %>% transform_to_tensor() %>% transform_resize(c(224, 224))
)

# Create a dataloader
dl <- dataloader(
  dataset = flowers,
  batch_size = 4
)

# Access a batch
batch <- dataloader_next(dataloader_make_iter(dl))
batch$x  # Tensor of shape (4, 3, 224, 224)
batch$y  # Tensor of shape (4,) with numeric class labels

## End(Not run)
```

---

generalized_box_iou *Generalized Box IoU*

Description

Return generalized intersection-over-union (Jaccard index) of boxes. Both sets of boxes are expected to be in $(x_{min}, y_{min}, x_{max}, y_{max})$ format with $0 \leq x_{min} < x_{max}$ and $0 \leq y_{min} < y_{max}$.

Usage

```
generalized_box_iou(boxes1, boxes2)
```

Arguments

boxes1	(Tensor[N, 4])
boxes2	(Tensor[M, 4])

Details

Implementation adapted from https://github.com/facebookresearch/detr/blob/master/util/box_ops.py

Value

generalized_iou (Tensor[N, M]): the NxM matrix containing the pairwise generalized_IoU values for every element in boxes1 and boxes2

`image_folder_dataset` *Create an image folder dataset*

Description

A generic data loader for images stored in folders. See `Details` for more information.

Usage

```
image_folder_dataset(
  root,
  transform = NULL,
  target_transform = NULL,
  loader = NULL,
  is_valid_file = NULL
)
```

Arguments

`root`	Root directory path.
`transform`	A function/transform that takes in an PIL image and returns a transformed version. E.g, `transform_random_crop()`.
`target_transform`	A function/transform that takes in the target and transforms it.
`loader`	A function to load an image given its path.
`is_valid_file`	A function that takes path of an Image file and check if the file is a valid file (used to check of corrupt files)

Details

This function assumes that the images for each class are contained in subdirectories of `root`. The names of these subdirectories are stored in the `classes` attribute of the returned object.

An example folder structure might look as follows:

```
root/dog/xxx.png
root/dog/xxy.png
root/dog/xxz.png

root/cat/123.png
root/cat/nsdf3.png
root/cat/asd932_.png
```

magick_loader *Load an Image using ImageMagick*

Description

Load an image located at `path` using the `{magick}` package.

Usage

```
magick_loader(path)
```

Arguments

`path`	path to the image to load from.

mnist_dataset *MNIST and Derived Datasets*

Description

Prepares various MNIST-style image classification datasets and optionally downloads them. Images are thumbnails images of 28 x 28 pixels of grayscale values encoded as integer.

Usage

```
mnist_dataset(
  root = tempdir(),
  train = TRUE,
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)

kmnist_dataset(
  root = tempdir(),
  train = TRUE,
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)

qmnist_dataset(
  root = tempdir(),
  split = "train",
  transform = NULL,
```

```
  target_transform = NULL,
  download = FALSE
)

fashion_mnist_dataset(
  root = tempdir(),
  train = TRUE,
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)

emnist_dataset(
  root = tempdir(),
  split = "balanced",
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)
```

Arguments

root	Root directory for dataset storage. The dataset will be stored under root/<dataset-name>. Defaults to tempdir().
train	Logical. If TRUE, use the training set; otherwise, use the test set. Not applicable to all datasets.
transform	Optional. A function that takes an image and returns a transformed version (e.g., normalization, cropping).
target_transform	Optional. A function that transforms the label.
download	Logical. If TRUE, downloads the dataset to root/. If the dataset is already present, download is skipped.
split	Character. Used in emnist_dataset() and qmnist_dataset() to specify the subset. See individual descriptions for valid values.

Details

- **MNIST**: Original handwritten digit dataset.
- **Fashion-MNIST**: Clothing item images for classification.
- **Kuzushiji-MNIST**: Japanese cursive character dataset.
- **QMNIST**: Extended MNIST with high-precision NIST data.
- **EMNIST**: Letters and digits with multiple label splits.

Value

A torch dataset object, where each items is a list of x (image) and y (label).

Functions

- `kmnist_dataset()`: Kuzushiji-MNIST cursive Japanese character dataset.
- `qmnist_dataset()`: Extended MNIST dataset with high-precision test data (QMNIST).
- `fashion_mnist_dataset()`: Fashion-MNIST clothing image dataset.
- `emnist_dataset()`: EMNIST dataset with digits and letters and multiple split modes.

Supported Splits for `emnist_dataset()`

- `"byclass"`: 62 classes (digits + uppercase + lowercase)
- `"bymerge"`: 47 classes (merged uppercase and lowercase)
- `"balanced"`: 47 classes, balanced digits and letters
- `"letters"`: 26 uppercase letters
- `"digits"`: 10 digit classes
- `"mnist"`: Standard MNIST digit classes

Supported Splits for `qmnist_dataset()`

- `"train"`: 60,000 training samples (MNIST-compatible)
- `"test"`: Extended test set
- `"nist"`: Full NIST digit set

See Also

Other classification_dataset: `caltech_dataset`, `cifar10_dataset()`, `eurosat_dataset()`, `fer_dataset()`, `fgvc_aircraft_dataset()`, `flowers102_dataset()`, `oxfordiiitpet_dataset()`, `tiny_imagenet_dataset()`

Examples

```
## Not run:
ds <- mnist_dataset(download = TRUE)
item <- ds[1]
item$x  # image
item$y  # label

qmnist <- qmnist_dataset(split = "train", download = TRUE)
item <- qmnist[1]
item$x
item$y

emnist <- emnist_dataset(split = "balanced", download = TRUE)
item <- emnist[1]
item$x
item$y

kmnist <- kmnist_dataset(download = TRUE)
fmnist <- fashion_mnist_dataset(download = TRUE)

## End(Not run)
```

model_alexnet *AlexNet Model Architecture*

Description

AlexNet model architecture from the One weird trick... paper.

Usage

```
model_alexnet(pretrained = FALSE, progress = TRUE, ...)
```

Arguments

pretrained	(bool): If TRUE, returns a model pre-trained on ImageNet.
progress	(bool): If TRUE, displays a progress bar of the download to stderr.
...	other parameters passed to the model intializer. currently only num_classes is used.

See Also

Other models: model_inception_v3(), model_mobilenet_v2(), model_resnet, model_vgg

model_inception_v3 *Inception v3 model*

Description

Architecture from Rethinking the Inception Architecture for Computer Vision The required minimum input size of the model is 75x75.

Usage

```
model_inception_v3(pretrained = FALSE, progress = TRUE, ...)
```

Arguments

pretrained	(bool): If TRUE, returns a model pre-trained on ImageNet
progress	(bool): If TRUE, displays a progress bar of the download to stderr
...	Used to pass keyword arguments to the Inception module: • aux_logits (bool): If TRUE, add an auxiliary branch that can improve training. Default: *TRUE* • transform_input (bool): If TRUE, preprocess the input according to the method with which it was trained on ImageNet. Default: *FALSE*

Note

Important: In contrast to the other models the inception_v3 expects tensors with a size of N x 3 x 299 x 299, so ensure your images are sized accordingly.

See Also

Other models: `model_alexnet()`, `model_mobilenet_v2()`, `model_resnet`, `model_vgg`

`model_mobilenet_v2` *Constructs a MobileNetV2 architecture from* R*hrefhttps://arxiv.org/abs/1801.04381MobileNetV2: Inverted Residuals and Linear Bottlenecks.*

Description

Constructs a MobileNetV2 architecture from MobileNetV2: Inverted Residuals and Linear Bottlenecks.

Usage

```
model_mobilenet_v2(pretrained = FALSE, progress = TRUE, ...)
```

Arguments

`pretrained`	(bool): If TRUE, returns a model pre-trained on ImageNet.
`progress`	(bool): If TRUE, displays a progress bar of the download to stderr.
`...`	Other parameters passed to the model implementation.

See Also

Other models: `model_alexnet()`, `model_inception_v3()`, `model_resnet`, `model_vgg`

`model_resnet` *ResNet implementation*

Description

ResNet models implementation from Deep Residual Learning for Image Recognition and later related papers (see Functions)

Usage

```
model_resnet18(pretrained = FALSE, progress = TRUE, ...)

model_resnet34(pretrained = FALSE, progress = TRUE, ...)

model_resnet50(pretrained = FALSE, progress = TRUE, ...)

model_resnet101(pretrained = FALSE, progress = TRUE, ...)

model_resnet152(pretrained = FALSE, progress = TRUE, ...)

model_resnext50_32x4d(pretrained = FALSE, progress = TRUE, ...)

model_resnext101_32x8d(pretrained = FALSE, progress = TRUE, ...)

model_wide_resnet50_2(pretrained = FALSE, progress = TRUE, ...)

model_wide_resnet101_2(pretrained = FALSE, progress = TRUE, ...)
```

Arguments

`pretrained`	(bool): If TRUE, returns a model pre-trained on ImageNet.
`progress`	(bool): If TRUE, displays a progress bar of the download to stderr.
`...`	Other parameters passed to the resnet model.

Functions

- `model_resnet18()`: ResNet 18-layer model
- `model_resnet34()`: ResNet 34-layer model
- `model_resnet50()`: ResNet 50-layer model
- `model_resnet101()`: ResNet 101-layer model
- `model_resnet152()`: ResNet 152-layer model
- `model_resnext50_32x4d()`: ResNeXt-50 32x4d model from "Aggregated Residual Transformation for Deep Neural Networks" with 32 groups having each a width of 4.
- `model_resnext101_32x8d()`: ResNeXt-101 32x8d model from "Aggregated Residual Transformation for Deep Neural Networks" with 32 groups having each a width of 8.
- `model_wide_resnet50_2()`: Wide ResNet-50-2 model from "Wide Residual Networks" with width per group of 128.
- `model_wide_resnet101_2()`: Wide ResNet-101-2 model from "Wide Residual Networks" with width per group of 128.

See Also

Other models: `model_alexnet()`, `model_inception_v3()`, `model_mobilenet_v2()`, `model_vgg`

model_vgg *VGG implementation*

Description

VGG models implementations based on Very Deep Convolutional Networks For Large-Scale Image Recognition

Usage

```
model_vgg11(pretrained = FALSE, progress = TRUE, ...)

model_vgg11_bn(pretrained = FALSE, progress = TRUE, ...)

model_vgg13(pretrained = FALSE, progress = TRUE, ...)

model_vgg13_bn(pretrained = FALSE, progress = TRUE, ...)

model_vgg16(pretrained = FALSE, progress = TRUE, ...)

model_vgg16_bn(pretrained = FALSE, progress = TRUE, ...)

model_vgg19(pretrained = FALSE, progress = TRUE, ...)

model_vgg19_bn(pretrained = FALSE, progress = TRUE, ...)
```

Arguments

pretrained	(bool): If TRUE, returns a model pre-trained on ImageNet
progress	(bool): If TRUE, displays a progress bar of the download to stderr
...	other parameters passed to the VGG model implementation.

Functions

- model_vgg11(): VGG 11-layer model (configuration "A")
- model_vgg11_bn(): VGG 11-layer model (configuration "A") with batch normalization
- model_vgg13(): VGG 13-layer model (configuration "B")
- model_vgg13_bn(): VGG 13-layer model (configuration "B") with batch normalization
- model_vgg16(): VGG 13-layer model (configuration "D")
- model_vgg16_bn(): VGG 13-layer model (configuration "D") with batch normalization
- model_vgg19(): VGG 19-layer model (configuration "E")
- model_vgg19_bn(): VGG 19-layer model (configuration "E") with batch normalization

See Also

Other models: model_alexnet(), model_inception_v3(), model_mobilenet_v2(), model_resnet

nms *Non-maximum Suppression (NMS)*

Description

Performs non-maximum suppression (NMS) on the boxes according to their intersection-over-union (IoU). NMS iteratively removes lower scoring boxes which have an IoU greater than iou_threshold with another (higher scoring) box.

Usage

```
nms(boxes, scores, iou_threshold)
```

Arguments

boxes	(Tensor[N, 4])): boxes to perform NMS on. They are expected to be in $(x_{min}, y_{min}, x_{max}, y_{max})$ format with • $0 \leq x_{min} < x_{max}$ and • $0 \leq y_{min} < y_{max}$.
scores	(Tensor[N]): scores for each one of the boxes
iou_threshold	(float): discards all overlapping boxes with IoU > iou_threshold

Details

If multiple boxes have the exact same score and satisfy the IoU criterion with respect to a reference box, the selected box is not guaranteed to be the same between CPU and GPU. This is similar to the behavior of argsort in torch when repeated values are present.

Current algorithm has a time complexity of O(n^2) and runs in native R. It may be improve in the future by a Rcpp implementation or through alternative algorithm

Value

keep (Tensor): int64 tensor with the indices of the elements that have been kept by NMS, sorted in decreasing order of scores.

oxfordiiitpet_dataset *Oxford-IIIT Pet Classification Datasets*

Description

Oxford-IIIT Pet Datasets

Usage

```
oxfordiiitpet_dataset(
  root = tempdir(),
  train = TRUE,
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)

oxfordiiitpet_binary_dataset(
  root = tempdir(),
  train = TRUE,
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)
```

Arguments

root	Character. Root directory where the dataset is stored or will be downloaded to. Files are placed under root/oxfordiiitpet.
train	Logical. If TRUE, use the training set; otherwise, use the test set. Not applicable to all datasets.
transform	Optional. A function that takes an image and returns a transformed version (e.g., normalization, cropping).
target_transform	Optional. A function that transforms the label.
download	Logical. If TRUE, downloads the dataset to root/. If the dataset is already present, download is skipped.

Details

The Oxford-IIIT Pet collection is a **classification** dataset consisting of high-quality images of 37 cat and dog breeds. It includes two variants:

- oxfordiiitpet_dataset: Multi-class classification across 37 pet breeds.
- oxfordiiitpet_binary_dataset: Binary classification distinguishing cats vs dogs.

The Oxford-IIIT Pet dataset contains over 7,000 images across 37 categories, with roughly 200 images per class. Each image is labeled with its breed and species (cat/dog).

Value

A torch dataset object oxfordiiitpet_dataset or oxfordiiitpet_binary_dataset. Each element is a named list with:

- x: A H x W x 3 integer array representing an RGB image.
- y: An integer label:

- For `oxfordiiitpet_dataset`: a value from 1–37 representing the breed.
- For `oxfordiiitpet_binary_dataset`: 1 for Cat, 2 for Dog.

See Also

Other classification_dataset: `caltech_dataset`, `cifar10_dataset()`, `eurosat_dataset()`, `fer_dataset()`, `fgvc_aircraft_dataset()`, `flowers102_dataset()`, `mnist_dataset()`, `tiny_imagenet_dataset()`

Examples

```
## Not run:
# Multi-class version
oxford <- oxfordiiitpet_dataset(download = TRUE)
first_item <- oxford[1]
first_item$x  # RGB image
first_item$y  # Label in 1-37
oxford$classes[first_item$y]  # Breed name

# Binary version
oxford_bin <- oxfordiiitpet_binary_dataset(download = TRUE)
first_item <- oxford_bin[1]
first_item$x  # RGB image
first_item$y  # 1 for Cat, 2 for Dog
oxford_bin$classes[first_item$y]  # "Cat" or "Dog"

## End(Not run)
```

---

oxfordiiitpet_segmentation_dataset

Oxford-IIIT Pet Segmentation Dataset

Description

The Oxford-IIIT Pet Dataset is a **segmentation** dataset consisting of color images of 37 pet breeds (cats and dogs). Each image is annotated with a pixel-level trimap segmentation mask, identifying pet, background, and outline regions. It is commonly used for evaluating models on object segmentation tasks.

Usage

```
oxfordiiitpet_segmentation_dataset(
  root = tempdir(),
  train = TRUE,
  target_type = "category",
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)
```

Arguments

root	Character. Root directory where the dataset is stored or will be downloaded to. Files are placed under root/oxfordiiitpet.
train	Logical. If TRUE, use the training set; otherwise, use the test set. Not applicable to all datasets.
target_type	Character. One of "category" or "binary-category" (default: "category").
transform	Optional. A function that takes an image and returns a transformed version (e.g., normalization, cropping).
target_transform	Optional. A function that transforms the label.
download	Logical. If TRUE, downloads the dataset to root/. If the dataset is already present, download is skipped.

Value

A torch dataset object oxfordiiitpet_dataset. Each item is a named list:

- x: a H x W x 3 integer array representing an RGB image.
- y$masks: a boolean tensor of shape (3, H, W), representing the segmentation trimap as one-hot masks.
- y$label: an integer representing the class label, depending on the target_type:
  - "category": an integer in 1–37 indicating the pet breed.
  - "binary-category": 1 for Cat, 2 for Dog.

Examples

```
## Not run:
# Load the Oxford-IIIT Pet dataset with basic tensor transform
oxfordiiitpet <- oxfordiiitpet_segmentation_dataset(
   transform = transform_to_tensor,
   download = TRUE
)

# Retrieve the image tensor, segmentation mask and label
first_item <- oxfordiiitpet[1]
first_item$x  # RGB image tensor of shape (3, H, W)
first_item$y$masks   # (3, H, W) bool tensor: pet, background, outline
first_item$y$label  # Integer label (1-37 or 1-2 depending on target_type)
oxfordiiitpet$classes[first_item$y$label] # Class name of the label

# Visualize
overlay <- draw_segmentation_masks(first_item)
tensor_image_browse(overlay)

## End(Not run)
```

remove_small_boxes *Remove Small Boxes*

Description

Remove boxes which contains at least one side smaller than min_size.

Usage

```
remove_small_boxes(boxes, min_size)
```

Arguments

boxes	(Tensor[N, 4]): boxes in $(x_{min}, y_{min}, x_{max}, y_{max})$ format with • $0 \leq x_{min} < x_{max}$ and • $0 \leq y_{min} < y_{max}$.
min_size	(float): minimum size

Value

keep (Tensor[K]): indices of the boxes that have both sides larger than min_size

tensor_image_browse *Display image tensor*

Description

Display image tensor into browser

Usage

```
tensor_image_browse(image, browser = getOption("browser"))
```

Arguments

image	`torch_tensor()` of shape (1, W, H) for grayscale image or (3, W, H) for color image to display
browser	argument passed to browseURL

See Also

Other image display: `draw_bounding_boxes()`, `draw_keypoints()`, `draw_segmentation_masks()`, `tensor_image_display()`, `vision_make_grid()`

tensor_image_display *Display image tensor*

Description

Display image tensor onto the X11 device

Usage

```
tensor_image_display(image, animate = TRUE)
```

Arguments

image	torch_tensor() of shape (1, W, H) for grayscale image or (3, W, H) for color image to display
animate	support animations in the X11 display

See Also

Other image display: draw_bounding_boxes(), draw_keypoints(), draw_segmentation_masks(), tensor_image_browse(), vision_make_grid()

tiny_imagenet_dataset *Tiny ImageNet dataset*

Description

Prepares the Tiny ImageNet dataset and optionally downloads it.

Usage

```
tiny_imagenet_dataset(root, split = "train", download = FALSE, ...)
```

Arguments

root	directory path to download the dataset.
split	dataset split, train, validation or test.
download	whether to download or not the dataset.
...	other arguments passed to image_folder_dataset().

See Also

Other classification_dataset: caltech_dataset, cifar10_dataset(), eurosat_dataset(), fer_dataset(), fgvc_aircraft_dataset(), flowers102_dataset(), mnist_dataset(), oxfordiiitpet_dataset()

transform_adjust_brightness

Adjust the brightness of an image

Description

Adjust the brightness of an image

Usage

```
transform_adjust_brightness(img, brightness_factor)
```

Arguments

img	A magick-image, array or torch_tensor.
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.

See Also

Other transforms: transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

transform_adjust_contrast

Adjust the contrast of an image

Description

Adjust the contrast of an image

Usage

```
transform_adjust_contrast(img, contrast_factor)
```

Arguments

img	A magick-image, array or torch_tensor.
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.

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

---

transform_adjust_gamma

Adjust the gamma of an RGB image

Description

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

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

Usage

```
transform_adjust_gamma(img, gamma, gain = 1)
```

Arguments

img	A magick-image, array or torch_tensor.
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.

Details

See Gamma Correction for more details.

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

---

transform_adjust_hue *Adjust the hue of an image*

Description

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.

Usage

```
transform_adjust_hue(img, hue_factor)
```

Arguments

img	A magick-image, array or torch_tensor.
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.

Details

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.

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(),

transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

`transform_adjust_saturation` *Adjust the color saturation of an image*

Description

Adjust the color saturation of an image

Usage

```
transform_adjust_saturation(img, saturation_factor)
```

Arguments

`img`	A `magick-image`, `array` or `torch_tensor`.
`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.

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

`transform_affine` *Apply affine transformation on an image keeping image center invariant*

Description

Apply affine transformation on an image keeping image center invariant

Usage

```
transform_affine(
  img,
  angle,
  translate,
  scale,
  shear,
  resample = 0,
  fillcolor = NULL
)
```

Arguments

`img`	A `magick-image`, `array` or `torch_tensor`.
`angle`	(float or int): rotation angle value in degrees, counter-clockwise.
`translate`	(sequence of int) – horizontal and vertical translations (post-rotation translation)
`scale`	(float) – overall scale
`shear`	(float or sequence) – shear angle value in degrees between -180 to 180, clockwise direction. If a sequence is specified, the first value corresponds to a shear parallel to the x-axis, while the second value corresponds to a shear parallel to the y-axis.
`resample`	(int, optional): An optional resampling filter. See interpolation modes.
`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). This option is not supported for Tensor input. Fill value for the area outside the transform in the output image is always 0.

See Also

Other transforms: `transform_adjust_brightness()`, `transform_adjust_contrast()`, `transform_adjust_gamma()`, `transform_adjust_hue()`, `transform_adjust_saturation()`, `transform_center_crop()`, `transform_color_jitter()`, `transform_convert_image_dtype()`, `transform_crop()`, `transform_five_crop()`, `transform_grayscale()`, `transform_hflip()`, `transform_linear_transformation()`, `transform_normalize()`, `transform_pad()`, `transform_perspective()`, `transform_random_affine()`, `transform_random_apply()`, `transform_random_choice()`, `transform_random_crop()`, `transform_random_erasing()`, `transform_random_grayscale()`, `transform_random_horizontal_flip()`, `transform_random_order()`, `transform_random_perspective()`,

transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

`transform_center_crop` *Crops the given image at the center*

Description

The image can be a Magick Image or a torch Tensor, in which case it is expected to have `[..., H, W]` shape, where ... means an arbitrary number of leading dimensions.

Usage

```
transform_center_crop(img, size)
```

Arguments

`img`	A `magick-image`, `array` or `torch_tensor`.
`size`	(sequence or int): Desired output size of the crop. If size is an int instead of sequence like c(h, w), a square crop (size, size) is made. If provided a tuple or list of length 1, it will be interpreted as `c(size, size)`.

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

`transform_color_jitter`
Randomly change the brightness, contrast and saturation of an image

---

Description

Randomly change the brightness, contrast and saturation of an image

Usage

```
transform_color_jitter(
  img,
  brightness = 0,
  contrast = 0,
  saturation = 0,
  hue = 0
)
```

Arguments

img	A magick-image, array or torch_tensor.
brightness	(float or tuple of 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 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 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 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.

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

---

transform_convert_image_dtype

Convert a tensor image to the given dtype *and scale the values accordingly*

Description

Convert a tensor image to the given dtype and scale the values accordingly

Usage

```
transform_convert_image_dtype(img, dtype = torch::torch_float())
```

Arguments

img	A magick-image, array or torch_tensor.
dtype	(torch.dtype): Desired data type of the output.

Note

When converting from a smaller to a larger integer dtype the maximum values are **not** mapped exactly. If converted back and forth, this mismatch has no effect.

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

---

transform_crop *Crop the given image at specified location and output size*

Description

Crop the given image at specified location and output size

Usage

```
transform_crop(img, top, left, height, width)
```

Arguments

img	A magick-image, array or torch_tensor.
top	(int): Vertical component of the top left corner of the crop box.
left	(int): Horizontal component of the top left corner of the crop box.
height	(int): Height of the crop box.
width	(int): Width of the crop box.

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayso transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

---

transform_five_crop *Crop image into four corners and a central crop*

Description

Crop the given image into four corners and the central crop. This transform returns a tuple of images and there may be a mismatch in the number of inputs and targets your Dataset returns.

Usage

```
transform_five_crop(img, size)
```

Arguments

img	A magick-image, array or torch_tensor.
size	(sequence or int): Desired output size. If size is a sequence like c(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).

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

transform_grayscale *Convert image to grayscale*

Description

Convert image to grayscale

Usage

```
transform_grayscale(img, num_output_channels)
```

Arguments

img	A magick-image, array or torch_tensor.
num_output_channels	(int): (1 or 3) number of channels desired for output image

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

transform_hflip *Horizontally flip a PIL Image or Tensor*

Description

Horizontally flip a PIL Image or Tensor

Usage

```
transform_hflip(img)
```

Arguments

img	A magick-image, array or torch_tensor.

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

---

transform_linear_transformation

Transform a tensor image with a square transformation matrix and a mean_vector computed offline

---

Description

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.

Usage

```
transform_linear_transformation(img, transformation_matrix, mean_vector)
```

Arguments

`img`	A `magick-image`, `array` or `torch_tensor`.
`transformation_matrix`	(Tensor): tensor `[D x D]`, D = C x H x W.
`mean_vector`	(Tensor): tensor D, D = C x H x W.

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`.

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_normalize(), transform_pad(), transform_perspective(

transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

`transform_normalize` *Normalize a tensor image with mean and standard deviation*

Description

Given mean: `(mean[1],...,mean[n])` and std: `(std[1],..,std[n])` for `n` channels, this transform will normalize each channel of the input `torch_tensor` i.e., `output[channel] = (input[channel] - mean[channel]) / std[channel]`

Usage

```
transform_normalize(img, mean, std, inplace = FALSE)
```

Arguments

`img`	A `magick-image`, `array` or `torch_tensor`.
`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.

Note

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

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

transform_pad *Pad the given image on all sides with the given "pad" value*

Description

The image can be a Magick Image or a torch Tensor, in which case it is expected to have `[..., H, W]` shape, where ... means an arbitrary number of leading dimensions.

Usage

```
transform_pad(img, padding, fill = 0, padding_mode = "constant")
```

Arguments

img	A `magick-image`, `array` or `torch_tensor`.
padding	(int or tuple or list): 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, right, top and bottom borders respectively.
fill	(int or str 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. Only int value is supported for Tensors.
padding_mode	Type of padding. Should be: constant, edge, reflect or symmetric. Default is constant. Mode symmetric is not yet supported for Tensor inputs. • 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]`

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

`transform_perspective` *Perspective transformation of an image*

Description

Perspective transformation of an image

Usage

```
transform_perspective(
  img,
  startpoints,
  endpoints,
  interpolation = 2,
  fill = NULL
)
```

Arguments

`img`	A `magick-image`, `array` or `torch_tensor`.
`startpoints`	(list of list of ints): List containing four lists of two integers corresponding to four corners `[top-left, top-right, bottom-right, bottom-left]` of the original image.
`endpoints`	(list of list of ints): List containing four lists of two integers corresponding to four corners `[top-left, top-right, bottom-right, bottom-left]` of the transformed image.
`interpolation`	(int, optional) Desired interpolation. An integer `0` = `nearest`, `2` = `bilinear`, and `3` = `bicubic` or a name from `magick::filter_types()`.
`fill`	(int or str 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. Only int value is supported for Tensors.

See Also

Other transforms: `transform_adjust_brightness()`, `transform_adjust_contrast()`, `transform_adjust_gamma()`, `transform_adjust_hue()`, `transform_adjust_saturation()`, `transform_affine()`, `transform_center_crop()`, `transform_color_jitter()`, `transform_convert_image_dtype()`, `transform_crop()`, `transform_five_crop()`, `transform_grayscale()`, `transform_hflip()`, `transform_linear_transformation()`, `transform_normalize()`, `transform_pad()`, `transform_random_affine()`, `transform_random_apply()`, `transform_random_choice()`, `transform_random_crop()`, `transform_random_erasing()`, `transform_random_grayscale()`, `transform_random_horizontal_flip()`, `transform_random_order()`, `transform_random_perspective()`, `transform_random_resized_crop()`, `transform_random_rotation()`, `transform_random_vertical_flip()`, `transform_resize()`, `transform_resized_crop()`, `transform_rgb_to_grayscale()`, `transform_rotate()`, `transform_ten_crop()`, `transform_to_tensor()`, `transform_vflip()`

`transform_random_affine` *Random affine transformation of the image keeping center invariant*

Description

Random affine transformation of the image keeping center invariant

Usage

```
transform_random_affine(
  img,
  degrees,
  translate = NULL,
  scale = NULL,
  shear = NULL,
  resample = 0,
  fillcolor = 0
)
```

Arguments

`img`	A `magick-image`, `array` or `torch_tensor`.
`degrees`	(sequence or float or int): Range of degrees to select from. If degrees is a number instead of sequence like c(min, max), the range of degrees will be (-degrees, +degrees).
`translate`	(tuple, optional): tuple of maximum absolute fraction for horizontal and vertical translations. For example `translate=c(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 c(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 applied. Else if shear is a tuple or list of 2 values a shear parallel to the x axis in the range `(shear[1], shear[2])` will be applied. Else if shear is a tuple or list of 4 values, a x-axis shear in `(shear[1], shear[2])` and y-axis shear in `(shear[3], shear[4])` will be applied. Will not apply shear by default.
`resample`	(int, optional): An optional resampling filter. See interpolation modes.
`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). This option is not supported for Tensor input. Fill value for the area outside the transform in the output image is always 0.

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

---

transform_random_apply

Apply a list of transformations randomly with a given probability

---

Description

Apply a list of transformations randomly with a given probability

Usage

```
transform_random_apply(img, transforms, p = 0.5)
```

Arguments

img	A magick-image, array or torch_tensor.
transforms	(list or tuple): list of transformations.
p	(float): probability.

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

transform_random_choice

Apply single transformation randomly picked from a list

Description

Apply single transformation randomly picked from a list

Usage

```
transform_random_choice(img, transforms)
```

Arguments

img	A magick-image, array or torch_tensor.
transforms	(list or tuple): list of transformations.

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

transform_random_crop *Crop the given image at a random location*

Description

The image can be a Magick Image or a Tensor, in which case it is expected to have [..., H, W] shape, where ... means an arbitrary number of leading dimensions.

Usage

```
transform_random_crop(
  img,
  size,
  padding = NULL,
  pad_if_needed = FALSE,
```

```
  fill = 0,
  padding_mode = "constant"
)
```

Arguments

img	A magick-image, array or torch_tensor.
size	(sequence or int): Desired output size. If size is a sequence like c(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).
padding	(int or tuple or list): 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, right, top and 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	(int or str 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. Only int value is supported for Tensors.
padding_mode	Type of padding. Should be: constant, edge, reflect or symmetric. Default is constant. Mode symmetric is not yet supported for Tensor inputs. • 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]`

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

`transform_random_erasing` *Randomly selects a rectangular region in an image and erases its pixel values*

Description

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

Usage

```
transform_random_erasing(
  img,
  p = 0.5,
  scale = c(0.02, 0.33),
  ratio = c(0.3, 3.3),
  value = 0,
  inplace = FALSE
)
```

Arguments

`img`	A `magick-image`, `array` or `torch_tensor`.
`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.

See Also

Other transforms: `transform_adjust_brightness()`, `transform_adjust_contrast()`, `transform_adjust_gamma()`, `transform_adjust_hue()`, `transform_adjust_saturation()`, `transform_affine()`, `transform_center_crop()`, `transform_color_jitter()`, `transform_convert_image_dtype()`, `transform_crop()`, `transform_five_crop()`, `transform_grayscale()`, `transform_hflip()`, `transform_linear_transformation()`, `transform_normalize()`, `transform_pad()`, `transform_perspective()`, `transform_random_affine()`, `transform_random_apply()`, `transform_random_choice()`, `transform_random_crop()`, `transform_random_grayscale()`, `transform_random_horizontal_flip()`, `transform_random_order()`, `transform_random_perspective()`, `transform_random_resized_crop()`, `transform_random_rotation()`, `transform_random_vertical_flip()`, `transform_resize()`, `transform_resized_crop()`, `transform_rgb_to_grayscale()`, `transform_rotate()`, `transform_ten_crop()`, `transform_to_tensor()`, `transform_vflip()`

transform_random_grayscale

Randomly convert image to grayscale with a given probability

Description

Convert image to grayscale with a probability of p.

Usage

```
transform_random_grayscale(img, p = 0.1)
```

Arguments

img	A magick-image, array or torch_tensor.
p	(float): probability that image should be converted to grayscale (default 0.1).

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_horizo transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

transform_random_horizontal_flip

Horizontally flip an image randomly with a given probability

Description

Horizontally flip an image randomly with a given probability. The image can be a Magick Image or a torch Tensor, in which case it is expected to have [..., H, W] shape, where ... means an arbitrary number of leading dimensions

Usage

```
transform_random_horizontal_flip(img, p = 0.5)
```

Arguments

img	A magick-image, array or torch_tensor.
p	(float): probability of the image being flipped. Default value is 0.5

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayso transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

---

transform_random_order

Apply a list of transformations in a random order

Description

Apply a list of transformations in a random order

Usage

```
transform_random_order(img, transforms)
```

Arguments

img	A magick-image, array or torch_tensor.
transforms	(list or tuple): list of transformations.

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayso transform_random_horizontal_flip(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

transform_random_perspective *Random perspective transformation of an image with a given probability*

Description

Performs a random perspective transformation of the given image with a given probability

Usage

```
transform_random_perspective(
  img,
  distortion_scale = 0.5,
  p = 0.5,
  interpolation = 2,
  fill = 0
)
```

Arguments

img	A magick-image, array or torch_tensor.
distortion_scale	(float): argument to control the degree of distortion and ranges from 0 to 1. Default is 0.5.
p	(float): probability of the image being transformed. Default is 0.5.
interpolation	(int, optional) Desired interpolation. An integer 0 = nearest, 2 = bilinear, and 3 = bicubic or a name from magick::filter_types().
fill	(int or str 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. Only int value is supported for Tensors.

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayso transform_random_horizontal_flip(), transform_random_order(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

`transform_random_resized_crop` *Crop image to random size and aspect ratio*

Description

Crop the given image to a random size and aspect ratio. The image can be a Magick Image or a Tensor, in which case it is expected to have `[..., H, W]` shape, where ... means an arbitrary number of leading dimensions

Usage

```
transform_random_resized_crop(
  img,
  size,
  scale = c(0.08, 1),
  ratio = c(3/4, 4/3),
  interpolation = 2
)
```

Arguments

`img`	A `magick-image`, `array` or `torch_tensor`.
`size`	(sequence or int): Desired output size. If size is a sequence like c(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).
`scale`	(tuple of float): range of size of the origin size cropped
`ratio`	(tuple of float): range of aspect ratio of the origin aspect ratio cropped.
`interpolation`	(int, optional) Desired interpolation. An integer `0` = `nearest`, `2` = `bilinear`, and `3` = `bicubic` or a name from `magick::filter_types()`.

Details

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.

See Also

Other transforms: `transform_adjust_brightness()`, `transform_adjust_contrast()`, `transform_adjust_gamma()`, `transform_adjust_hue()`, `transform_adjust_saturation()`, `transform_affine()`, `transform_center_crop()`, `transform_color_jitter()`, `transform_convert_image_dtype()`, `transform_crop()`, `transform_five_crop()`, `transform_grayscale()`, `transform_hflip()`, `transform_linear_transformation()`, `transform_normalize()`, `transform_pad()`, `transform_perspective()`, `transform_random_affine()`, `transform_random_apply()`, `transform_random_choice()`, `transform_random_crop()`, `transform_random_erasing()`, `transform_random_grays`

transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

`transform_random_rotation` *Rotate the image by angle*

Description

Rotate the image by angle

Usage

```
transform_random_rotation(
  img,
  degrees,
  resample = 0,
  expand = FALSE,
  center = NULL,
  fill = NULL
)
```

Arguments

`img`	A `magick-image`, `array` or `torch_tensor`.
`degrees`	(sequence or float or int): Range of degrees to select from. If degrees is a number instead of sequence like c(min, max), the range of degrees will be (-degrees, +degrees).
`resample`	(int, optional): An optional resampling filter. See interpolation modes.
`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`	(list or tuple, optional): Optional center of rotation, c(x, y). Origin is the upper left corner. Default is the center of the image.
`fill`	(n-tuple or int or float): Pixel fill value for area outside the rotated image. If int or float, the value is used for all bands respectively. Defaults to 0 for all bands. This option is only available for Pillow>=5.2.0. This option is not supported for Tensor input. Fill value for the area outside the transform in the output image is always 0.

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayso transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

---

transform_random_vertical_flip

Vertically flip an image randomly with a given probability

Description

The image can be a PIL Image or a torch Tensor, in which case it is expected to have `[..., H, W]` shape, where ... means an arbitrary number of leading dimensions

Usage

```
transform_random_vertical_flip(img, p = 0.5)
```

Arguments

img	A magick-image, array or torch_tensor.
p	(float): probability of the image being flipped. Default value is 0.5

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayso transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

transform_resize *Resize the input image to the given size*

Description

The image can be a Magic Image or a torch Tensor, in which case it is expected to have `[..., H, W]` shape, where ... means an arbitrary number of leading dimensions

Usage

```
transform_resize(img, size, interpolation = 2)
```

Arguments

`img`	A `magick-image`, `array` or `torch_tensor`.
`size`	(sequence or int): Desired output size. If size is a sequence like c(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. An integer `0` = `nearest`, `2` = `bilinear`, and `3` = `bicubic` or a name from `magick::filter_types()`.

See Also

Other transforms: `transform_adjust_brightness()`, `transform_adjust_contrast()`, `transform_adjust_gamma()`, `transform_adjust_hue()`, `transform_adjust_saturation()`, `transform_affine()`, `transform_center_crop()`, `transform_color_jitter()`, `transform_convert_image_dtype()`, `transform_crop()`, `transform_five_crop()`, `transform_grayscale()`, `transform_hflip()`, `transform_linear_transformation()`, `transform_normalize()`, `transform_pad()`, `transform_perspective()`, `transform_random_affine()`, `transform_random_apply()`, `transform_random_choice()`, `transform_random_crop()`, `transform_random_erasing()`, `transform_random_grayscale()`, `transform_random_horizontal_flip()`, `transform_random_order()`, `transform_random_perspective()`, `transform_random_resized_crop()`, `transform_random_rotation()`, `transform_random_vertical_flip()`, `transform_resized_crop()`, `transform_rgb_to_grayscale()`, `transform_rotate()`, `transform_ten_crop()`, `transform_to_tensor()`, `transform_vflip()`

transform_resized_crop

Crop an image and resize it to a desired size

Description

Crop an image and resize it to a desired size

Usage

```
transform_resized_crop(img, top, left, height, width, size, interpolation = 2)
```

Arguments

img	A magick-image, array or torch_tensor.
top	(int): Vertical component of the top left corner of the crop box.
left	(int): Horizontal component of the top left corner of the crop box.
height	(int): Height of the crop box.
width	(int): Width of the crop box.
size	(sequence or int): Desired output size. If size is a sequence like c(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. An integer 0 = nearest, 2 = bilinear, and 3 = bicubic or a name from magick::filter_types().

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grays transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

---

transform_rgb_to_grayscale

Convert RGB Image Tensor to Grayscale

Description

For RGB to Grayscale conversion, ITU-R 601-2 luma transform is performed which is L = R * 0.2989 + G * 0.5870 + B * 0.1140

Usage

```
transform_rgb_to_grayscale(img)
```

Arguments

img	A magick-image, array or torch_tensor.

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rotate(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

---

transform_rotate *Angular rotation of an image*

Description

Angular rotation of an image

Usage

```
transform_rotate(
  img,
  angle,
  resample = 0,
  expand = FALSE,
  center = NULL,
  fill = NULL
)
```

Arguments

img	A magick-image, array or torch_tensor.
angle	(float or int): rotation angle value in degrees, counter-clockwise.
resample	(int, optional): An optional resampling filter. See interpolation modes.
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	(list or tuple, optional): Optional center of rotation, c(x, y). Origin is the upper left corner. Default is the center of the image.
fill	(n-tuple or int or float): Pixel fill value for area outside the rotated image. If int or float, the value is used for all bands respectively. Defaults to 0 for all bands. This option is only available for Pillow>=5.2.0. This option is not supported for Tensor input. Fill value for the area outside the transform in the output image is always 0.

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayso transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_ten_crop(), transform_to_tensor(), transform_vflip()

---

transform_ten_crop *Crop an image and the flipped image each into four corners and a central crop*

Description

Crop the given image into four corners and the central crop, plus the flipped version of these (horizontal flipping is used by default). This transform returns a tuple of images and there may be a mismatch in the number of inputs and targets your Dataset returns.

Usage

```
transform_ten_crop(img, size, vertical_flip = FALSE)
```

Arguments

img	A magick-image, array or torch_tensor.
size	(sequence or int): Desired output size. If size is a sequence like c(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).
vertical_flip	(bool): Use vertical flipping instead of horizontal

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grayso transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_to_tensor(), transform_vflip()

transform_to_tensor *Convert an image to a tensor*

Description

Converts a Magick Image or array (H x W x C) in the range `[0, 255]` to a `torch_tensor` of shape (C x H x W) in the range `[0.0, 1.0]`. In the other cases, tensors are returned without scaling.

Usage

```
transform_to_tensor(img)
```

Arguments

`img`	A `magick-image`, `array` or `torch_tensor`.

Note

Because the input image is scaled to `[0.0, 1.0]`, this transformation should not be used when transforming target image masks.

See Also

Other transforms: `transform_adjust_brightness()`, `transform_adjust_contrast()`, `transform_adjust_gamma()`, `transform_adjust_hue()`, `transform_adjust_saturation()`, `transform_affine()`, `transform_center_crop()`, `transform_color_jitter()`, `transform_convert_image_dtype()`, `transform_crop()`, `transform_five_crop()`, `transform_grayscale()`, `transform_hflip()`, `transform_linear_transformation()`, `transform_normalize()`, `transform_pad()`, `transform_perspective()`, `transform_random_affine()`, `transform_random_apply()`, `transform_random_choice()`, `transform_random_crop()`, `transform_random_erasing()`, `transform_random_grayso`
`transform_random_horizontal_flip()`, `transform_random_order()`, `transform_random_perspective()`, `transform_random_resized_crop()`, `transform_random_rotation()`, `transform_random_vertical_flip()`, `transform_resize()`, `transform_resized_crop()`, `transform_rgb_to_grayscale()`, `transform_rotate()`, `transform_ten_crop()`, `transform_vflip()`

transform_vflip *Vertically flip a PIL Image or Tensor*

Description

Vertically flip a PIL Image or Tensor

Usage

```
transform_vflip(img)
```

Arguments

img	A magick-image, array or torch_tensor.

See Also

Other transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_color_jitter(), transform_convert_image_dtype(), transform_crop(), transform_five_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_random_affine(), transform_random_apply(), transform_random_choice(), transform_random_crop(), transform_random_erasing(), transform_random_grays transform_random_horizontal_flip(), transform_random_order(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip(), transform_resize(), transform_resized_crop(), transform_rgb_to_grayscale(), transform_rotate(), transform_ten_crop(), transform_to_tensor()

---

vision_make_grid *A simplified version of torchvision.utils.make_grid*

Description

Arranges a batch B of (image) tensors in a grid, with optional padding between images. Expects a 4d mini-batch tensor of shape (B x C x H x W).

Usage

```
vision_make_grid(
  tensor,
  scale = TRUE,
  num_rows = 8,
  padding = 2,
  pad_value = 0
)
```

Arguments

tensor	tensor of shape (B x C x H x W) to arrange in grid.
scale	whether to normalize (min-max-scale) the input tensor.
num_rows	number of rows making up the grid (default 8).
padding	amount of padding between batch images (default 2).
pad_value	pixel value to use for padding.

Value

a 3d torch_tensor of shape $\approx (C, num\_rows \times H, num\_cols \times W)$ of all images arranged in a grid.

See Also

Other image display: draw_bounding_boxes(), draw_keypoints(), draw_segmentation_masks(), tensor_image_browse(), tensor_image_display()

Index