Package 'torchvision'

May 8, 2026

Title Models, Datasets and Transformations for Images

Version 0.9.0

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

License MIT + file LICENSE

Encoding UTF-8

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

RoxygenNote 7.3.3

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

Suggests arrow, magick, prettyunits, testthat, coro, R.matlab, xml2, knitr, rmarkdown, torchvisionlib

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

Collate 'collection-catalog.R' 'folder-dataset.R' 'collection-rf100-doc.R' 'collection-rf100-biology.R' 'collection-rf100-damage.R' 'collection-rf100-infrared.R' 'collection-rf100-medical.R' 'collection-rf100-underwater.R' 'conditions.R' 'dataset-caltech.R' 'dataset-cifar.R' 'dataset-coco.R' 'dataset-eurosat.R' 'dataset-fer.R' 'dataset-fgvc.R' 'dataset-flickr.R' 'dataset-flowers.R' 'dataset-imagenet.R' 'dataset-lfw.R' 'dataset-mnist.R' 'dataset-oxfordiiitpet.R' 'dataset-pascal.R' 'dataset-places365.R' 'dataset-plankton.R' 'dataset-rf100-peixos.R' 'dataset-vggface2.R' 'extension.R' 'globals.R' 'models-alexnet.R' 'models-convnext.R' 'models-convnext_detection.R' 'models-convnext_segmentation.R' 'models-deeplabv3.R' 'models-efficientnet.R' 'models-efficientnetv2.R' 'models-facenet.R' 'models-faster_rcnn.R' 'models-fcn.R' 'models-inception.R'

'models-mask_rcnn.R' 'models-maxvit.R' 'models-mobilenetv2.R'
'models-mobilenetv3.R' 'models-mobilenetv3_large.R'
'models-resnet.R' 'models-vgg.R' 'models-vit.R'
'ops-box_convert.R' 'ops-boxes.R' 'transforms-array.R'
'transforms-defaults.R' 'transforms-generics.R'
'transforms-magick.R' 'transforms-segmentation.R'
'transforms-tensor.R' 'utils.R' 'vision_utils.R'

Depends R (>= 3.5)

LazyData true

VignetteBuilder knitr

NeedsCompilation no

Author Tomasz Kalinowski [ctb, cre],
Daniel Falbel [aut, cph],
Christophe Regouby [ctb],
Akanksha Koshti [ctb],
Derrick Richard [ctb],
ANAMASGARD [ctb],
Chandraveer Singh [ctb],
Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>)

Maintainer Tomasz Kalinowski <tomasz@posit.co>

Repository CRAN

Date/Publication 2026-04-22 14:20:02 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 or URL to load the image from.

Value

an channel-last array of image values with dim Height x Width x 3

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_classes *Caltech Class Labels*

Description

Utilities for resolving Caltech class identifiers to their corresponding human readable labels.

Usage

```
caltech_classes(class_id = 1:257)
```

Arguments

class_id	Integer vector of 1-based class identifiers.

Value

A character vector with 257 entries representing the Caltech 257 class labels.

See Also

Other class_resolution: coco_classes(), imagenet_classes(), pascal_voc_classes()

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(), image_folder_dataset(), lfw_dataset, mnist_dataset(), oxfordiiitpet_dataset(), places365_dataset(), tiny_imagenet_dataset(), vggface2_dataset(), whoi_plankton_dataset(), whoi_small_coralnet_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(), image_folder_dataset(), lfw_dataset, mnist_dataset(), oxfordiiitpet_dataset(), places365_dataset(), tiny_imagenet_dataset(), vggface2_dataset(), whoi_plankton_dataset(), whoi_small_coralnet_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_classes` *MS COCO Class Labels*

Description

Utilities for resolving COCO 90 class identifiers to their corresponding human readable labels. The labels are retrieved from pytorch/vision source to be compliant with torchvision pretrained models.

Usage

```
coco_classes(class_id = 1:90)
```

Arguments

`class_id`	Integer vector of 1-based class identifiers.

Value

A character vector with the COCO class names

See Also

Other class_resolution: `caltech_classes()`, `imagenet_classes()`, `pascal_voc_classes()`

`coco_detection_dataset` *COCO Detection Dataset*

Description

Loads the MS COCO dataset for object detection tasks only.

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

Files are downloaded to a `coco` subdirectory in the torch cache directory for better organization.

Value

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

- `x`: a `(C, H, W)` array representing the image.
- `y$boxes`: a `(N, 4)` `torch_tensor` of bounding boxes in the format $(x_{min}, y_{min}, x_{max}, y_{max})$.
- `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.

The returned object has S3 class `"image_with_bounding_box"` to enable automatic dispatch by visualization functions such as `draw_bounding_boxes()`.

For instance segmentation tasks, use `coco_segmentation_dataset` instead.

See Also

`coco_segmentation_dataset` for instance segmentation tasks

Other detection_dataset: `pascal_voc_datasets`, `rf100_biology_collection()`, `rf100_damage_collection()`, `rf100_document_collection()`, `rf100_infrared_collection()`, `rf100_medical_collection()`, `rf100_underwater_collection()`

Examples

```
## Not run:
# Load dataset for object detection
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)

## End(Not run)
```

`coco_segmentation_dataset` *COCO Segmentation Dataset*

Description

Loads the MS COCO dataset for instance segmentation tasks.

Usage

```
coco_segmentation_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. Use target_transform_coco_masks to convert polygon annotations to binary masks.

Details

The returned image x is in CHW format (channels, height, width), matching the torch convention. The dataset y offers instance segmentation annotations including labels, crowd indicators, and segmentation masks from the official COCO annotations.

Files are downloaded to a coco subdirectory in the torch cache directory for better organization.

Value

An object of class coco_segmentation_dataset. Each item is a list:

- x: a (C, H, W) array representing the image.
- y$labels: an integer torch_tensor with the class label for each object.
- y$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 (when using target_transform_coco_masks).

The returned object has S3 class "image_with_segmentation_mask" to enable automatic dispatch by visualization functions such as draw_segmentation_masks().

For object detection tasks without segmentation, use coco_detection_dataset instead.

See Also

coco_detection_dataset for object detection tasks

Other segmentation_dataset: oxfordiiitpet_segmentation_dataset(), pascal_voc_datasets, rf100_peixos_segmentation_dataset()

Examples

```
## Not run:
# Load dataset for instance segmentation
ds <- coco_segmentation_dataset(
  train = FALSE,
  year = "2017",
```

```
  download = TRUE,
  target_transform = target_transform_coco_masks
)

item <- ds[1]

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

## End(Not run)
```

`collection_catalog` *Dataset Collection Catalog*

Description

A comprehensive catalog of all collections RF100 (RoboFlow 100) and EMNIST datasets available in torchvision. This data frame contains metadata about each dataset including descriptions, sizes, available splits, and collection information.

Usage

```
collection_catalog
```

Format

A data frame with datasets as rows and 17 columns:

collection Collection name (biology, medical, infrared, damage, underwater, document, mnist)

dataset Dataset identifier used in collection functions

description Brief description of the dataset and its purpose

task Machine learning task type (currently all "object_detection")

num_classes Number of different object classes

num_images Total images across all splits

image_width Typical image width in pixels

image_height Typical image height in pixels

train_size_mb Size of training split in megabytes

test_size_mb Size of test split in megabytes

valid_size_mb Size of validation split in megabytes

total_size_mb Total size across all splits in megabytes

has_train Is training split available

has_test Is test split available

has_valid Is validation split available

function_name R function name to load this dataset's collection

roboflow_url URL to the collection on RoboFlow Universe

See Also

search_collection(), get_collection_catalog()

Examples

```
## Not run:
# View the complete catalog
data(collection_catalog)
View(collection_catalog)

# See all biology datasets
subset(collection_catalog, collection == "biology")

# Find large datasets (> 100 MB)
subset(collection_catalog, total_size_mb > 100)

## 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,
  color = 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: $0 \leq x_{min} < x_{max} < Height$ and $0 \leq y_{min} < y_{max} < Width$.
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.
color	Deprecated alias for `colors`.
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`	List of integer pairs `c(i, j)` specifying which keypoints to connect with a line, e.g. `list(c(1, 2), c(2, 3))`. `NULL` (default) draws no connecting lines.
`colors`	character vector containing the colors of the keypoints or single color for all keypoints. The color can be represented as strings e.g. "red" or "#FF00FF". By default, rainbow 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

```
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 = "val",
  download = FALSE,
  transform = NULL,
  target_transform = NULL
)

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

eurosat100_dataset(
  root = tempdir(),
  split = "val",
  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	One of "train", "val", or "test". Default is "val".
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(), image_folder_dataset(), lfw_dataset, mnist_dataset(), oxfordiiitpet_dataset(), places365_dataset(), tiny_imagenet_dataset(), vggface2_dataset(), whoi_plankton_dataset(), whoi_small_coralnet_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(), image_folder_dataset(), lfw_dataset, mnist_dataset(), oxfordiiitpet_dataset(), places365_dataset(), tiny_imagenet_dataset(), vggface2_dataset(), whoi_plankton_dataset(), whoi_small_coralnet_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(), image_folder_dataset(), lfw_dataset, mnist_dataset(), oxfordiiitpet_dataset(), places365_dataset(), tiny_imagenet_dataset(), vggface2_dataset(), whoi_plankton_dataset(), whoi_small_coralnet_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()`, `image_folder_dataset()`, `lfw_dataset`, `mnist_dataset()`, `oxfordiiitpet_dataset()`, `places365_dataset()`, `tiny_imagenet_dataset()`, `vggface2_dataset()`, `whoi_plankton_dataset()`, `whoi_small_coralnet_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

---

get_collection_catalog

Get Complete Collection Catalog

Description

Returns the complete catalog of datasets in collections with their metadata. This is a convenience function that loads and returns the collection_catalog data.

Usage

```
get_collection_catalog()
```

Value

A data frame with all datasets and their metadata.

See Also

search_collection(), collection_catalog

Examples

```
## Not run:
# Get complete catalog
catalog <- get_collection_catalog()

# View in RStudio
View(catalog)

# Summary statistics
summary(catalog$total_size_mb)
table(catalog$collection)

# Find smallest dataset
```

```
catalog[which.min(catalog$total_size_mb), ]

# Find largest dataset
catalog[which.max(catalog$total_size_mb), ]

## End(Not run)
```

`imagenet_classes` *ImageNet Class Labels*

Description

Utilities for resolving ImageNet-1k class identifiers to their corresponding human readable labels. The labels are retrieved from the same source used by PyTorch's reference implementation.

Usage

```
imagenet_classes(class_id = 1:1000)

imagenet_1k_classes(class_id = 1:1000)

imagenet_21k_df(class_id = 1:21843)

imagenet_21k_classes(class_id)
```

Arguments

`class_id`	Integer vector of 1-based class identifiers.

Value

A character vector with 1000 entries representing the ImageNet-1k class labels.

A data.frame containing columns `id` and `label` representing the ImageNet-21k class identifiers and labels. By default, returns all 21.8K rows.

A character vector with the labels associated with `class_id`.

See Also

Other class_resolution: `caltech_classes()`, `coco_classes()`, `pascal_voc_classes()`

Other class_resolution: `caltech_classes()`, `coco_classes()`, `pascal_voc_classes()`

Other class_resolution: `caltech_classes()`, `coco_classes()`, `pascal_voc_classes()`

Other class_resolution: `caltech_classes()`, `coco_classes()`, `pascal_voc_classes()`

`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
```

See Also

Other classification_dataset: caltech_dataset, cifar10_dataset(), eurosat_dataset(), fer_dataset(), fgvc_aircraft_dataset(), flowers102_dataset(), lfw_dataset, mnist_dataset(), oxfordiiitpet_dataset(), places365_dataset(), tiny_imagenet_dataset(), vggface2_dataset(), whoi_plankton_dataset(), whoi_small_coralnet_dataset()

`lfw_dataset` *LFW Datasets*

Description

Labelled Faces in the Wild (LFW) Datasets

Usage

```
lfw_people_dataset(
  root = tempdir(),
  transform = NULL,
  split = "original",
  target_transform = NULL,
  download = FALSE
)

lfw_pairs_dataset(
  root = tempdir(),
  train = TRUE,
  transform = NULL,
  split = "original",
  target_transform = NULL,
  download = FALSE
)
```

Arguments

`root`	Root directory for dataset storage. The dataset will be stored under `root/lfw_people` or `root/lfw_pairs`.
`transform`	Optional. A function that takes an image and returns a transformed version (e.g., normalization, cropping).
`split`	Which version of the dataset to use. One of `"original"` or `"funneled"`. Defaults to `"original"`.
`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.
`train`	For `lfw_pairs_dataset`, whether to load the training (`pairsDevTrain.txt`) or test (`pairsDevTest.txt`) split.

Details

The LFW dataset collection provides facial images for evaluating face recognition systems. It includes two variants:

- `lfw_people_dataset`: A **multi-class classification** dataset where each image is labelled by person identity.
- `lfw_pairs_dataset`: A **face verification** dataset containing image pairs with binary labels (same or different person).

This R implementation of the LFW dataset is based on the `fetch_lfw_people()` and `fetch_lfw_pairs()` functions from the `scikit-learn` library, but deviates in a few key aspects due to dataset availability and R API conventions:

- The `color` and `resize` arguments from Python are not directly exposed. Instead, all images are RGB with a fixed size of 250x250.
- The `split` argument in Python (e.g., `train`, `test`, `10fold`) is simplified to a `train` boolean flag in R. The `10fold` split is not supported, as the original protocol files are unavailable or incompatible with clean separation of image-label pairs.
- The `split` parameter in R controls which version of the dataset to use: `"original"` (unaligned) or `"funneled"` (aligned using funneling). The funneled version contains geometrically normalized face images, offering better alignment and typically improved performance for face recognition models.
- The dataset is downloaded from Figshare, which hosts the same files referenced in `scikit-learn`'s dataset utilities.
- `lfw_people_dataset`: 13,233 images across multiple identities (using either `"original"` or `"funneled"` splits)
- `lfw_pairs_dataset`:
  - Training split (`train = TRUE`): 2,200 image pairs
  - Test split (`train = FALSE`): 1,000 image pairs

Value

A torch dataset object `lfw_people_dataset` or `lfw_pairs_dataset`. Each element is a named list with:

- x:
  - For `lfw_people_dataset`: a H x W x 3 numeric array representing a single RGB image.
  - For `lfw_pairs_dataset`: a list of two H x W x 3 numeric arrays representing a pair of RGB images.
- y:
  - For `lfw_people_dataset`: an integer index from 1 to the number of identities in the dataset.
  - For `lfw_pairs_dataset`: 1 if the pair shows the same person, 2 if different people.

See Also

Other classification_dataset: caltech_dataset, cifar10_dataset(), eurosat_dataset(), fer_dataset(), fgvc_aircraft_dataset(), flowers102_dataset(), image_folder_dataset(), mnist_dataset(), oxfordiiitpet_dataset(), places365_dataset(), tiny_imagenet_dataset(), vggface2_dataset(), whoi_plankton_dataset(), whoi_small_coralnet_dataset()

Examples

```
## Not run:
# Load data for LFW People Dataset
lfw <- lfw_people_dataset(download = TRUE)
first_item <- lfw[1]
first_item$x  # RGB image
first_item$y  # Label index
lfw$classes[first_item$y]  # person's name (e.g., "Aaron_Eckhart")

# Load training data for LFW Pairs Dataset
lfw <- lfw_pairs_dataset(download = TRUE, train = TRUE)
first_item <- lfw[1]
first_item$x  # List of 2 RGB Images
first_item$x[[1]]  # RGB Image
first_item$x[[2]]  # RGB Image
first_item$y  # Label index
lfw$classes[first_item$y]  # Class Name (e.g., "Same" or "Different")

# Load test data for LFW Pairs Dataset
lfw <- lfw_pairs_dataset(download = TRUE, train = FALSE)
first_item <- lfw[1]
first_item$x  # List of 2 RGB Images
first_item$x[[1]]  # RGB Image
first_item$x[[2]]  # RGB Image
first_item$y  # Label index
lfw$classes[first_item$y]  # Class Name (e.g., "Same" or "Different")

## End(Not run)
```

---

list_collection_datasets

List Datasets in a Collection

Description

List all available datasets within a specific RF100 collection.

Usage

```
list_collection_datasets(collection)
```

Arguments

collection	Collection name. One of: "biology", "medical", "infrared", "damage", "underwater", "document".

Value

Character vector of dataset names in the collection.

See Also

search_collection(), get_collection_catalog()

Examples

```
## Not run:
# List all biology datasets
list_collection_datasets("biology")

# List all medical datasets
list_collection_datasets("medical")

## End(Not run)
```

---

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 or URL to load the image from.

Value

an magick-image object as result of image_read()

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_collection(
  root = tempdir(),
  split = "test",
  dataset = "balanced",
  transform = NULL,
```

```
  target_transform = NULL,
  download = FALSE
)

emnist_dataset(kind, ...)
```

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.
dataset	to select within c("byclass", "bymerge", "balanced", "letters", "digits", "mnist") representing the subset of emnist collection made of a set of classes. You can look at dataset attribute $classes to see the actual classes.
kind	the dataset in emnist_collection.
...	the other emnist_collection parameters.

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**: A collection of letters and digits with multiple datasets and 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_collection(): EMNIST collection with digits and letters arranged in multiple datasets.
- emnist_dataset(): Deprecated. Please use emnist_collection.

Supported `datasets` **for** `emnist_collection()`

- `"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()`, `image_folder_dataset()`, `lfw_dataset`, `oxfordiiitpet_dataset()`, `places365_dataset()`, `tiny_imagenet_dataset()`, `vggface2_dataset()`, `whoi_plankton_dataset()`, `whoi_small_coralnet_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_collection(dataset = "balanced", split = "test", download = TRUE)
item <- emnist[1]
item$x
item$y

kmnist <- kmnist_dataset(download = TRUE, train = FALSE)
fmnist <- fashion_mnist_dataset(download = TRUE, train = 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 classification_model: model_convnext, model_efficientnet, model_efficientnet_v2, model_facenet, model_inception_v3(), model_maxvit(), model_mobilenet_v2(), model_mobilenet_v3, model_resnet, model_vgg, model_vit

model_convnext *ConvNeXt Implementation*

Description

Implements the ConvNeXt architecture from ConvNeXt: A ConvNet for the 2020s

Usage

```
model_convnext_tiny_1k(
  pretrained = FALSE,
  progress = TRUE,
  channels = 3,
  num_classes = 1000,
  ...
)

model_convnext_tiny_22k(
  pretrained = FALSE,
  progress = TRUE,
  channels = 3,
```

```
  num_classes = 21841,
  ...
)

model_convnext_small_22k(
  pretrained = FALSE,
  progress = TRUE,
  channels = 3,
  num_classes = 21841,
  ...
)

model_convnext_small_22k1k(
  pretrained = FALSE,
  progress = TRUE,
  channels = 3,
  num_classes = 21841,
  ...
)

model_convnext_base_1k(
  pretrained = FALSE,
  progress = TRUE,
  channels = 3,
  num_classes = 1000,
  ...
)

model_convnext_base_22k(
  pretrained = FALSE,
  progress = TRUE,
  channels = 3,
  num_classes = 21841,
  ...
)

model_convnext_large_1k(
  pretrained = FALSE,
  progress = TRUE,
  channels = 3,
  num_classes = 1000,
  ...
)

model_convnext_large_22k(
  pretrained = FALSE,
  progress = TRUE,
  channels = 3,
```

```
  num_classes = 21841,
  ...
)
```

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.
channels	The number of channels in the input image. Default: 3.
num_classes	number of output classes (default: 1000).
...	Other parameters passed to the model implementation.

Functions

- `model_convnext_tiny_1k()`: ConvNeXt Tiny model trained on Imagenet 1k.
- `model_convnext_tiny_22k()`: ConvNeXt Tiny model trained on Imagenet 22k.
- `model_convnext_small_22k()`: ConvNeXt Small model trained on Imagenet 22k.
- `model_convnext_small_22k1k()`: ConvNeXt Small model pretrained on Imagenet 1k and fine-tuned on Imagenet 22k classes.
- `model_convnext_base_1k()`: ConvNeXt Base model trained on Imagenet 1k.
- `model_convnext_base_22k()`: ConvNeXt Base model trained on Imagenet 22k.
- `model_convnext_large_1k()`: ConvNeXt Large model trained on Imagenet 1k.
- `model_convnext_large_22k()`: ConvNeXt Large model trained on Imagenet 22k.

Variants

Model Summary and Performance for pretrained weights:

Model	Top-1 Acc	Params	GFLOPS	File Size	`num_classes`	image size
convnext_tiny_1k	82.1%	28M	4.5	109 MB	1000	224 x 224
convnext_tiny_22k	82.9%	29M	4.5	170 MB	21841	224 x 224
convnext_small_22k	84.6%	50M	8.7	252 MB	21841	224 x 224
convnext_small_22k1k	84.6%	50M	8.7	252 MB	21841	224 x 224
convnext_base_1k	85.1%	89M	15.4	338 MB	1000	224 x 224
convnext_base_22k	85.8%	89M	15.4	420 MB	21841	224 x 224
convnext_large_1k	84.3%	198M	34.4	750 MB	1000	224 x 224
convnext_large_22k	86.6%	198M	34.4	880 MB	21841	224 x 224

See Also

Other classification_model: `model_alexnet()`, `model_efficientnet`, `model_efficientnet_v2`, `model_facenet`, `model_inception_v3()`, `model_maxvit()`, `model_mobilenet_v2()`, `model_mobilenet_v3`, `model_resnet`, `model_vgg`, `model_vit`

Examples

```
## Not run:
# 1. Download sample image (dog)
norm_mean <- c(0.485, 0.456, 0.406) # ImageNet normalization constants, see
# https://pytorch.org/vision/stable/models.html
norm_std  <- c(0.229, 0.224, 0.225)
img_url <- "https://en.wikipedia.org/wiki/Special:FilePath/Felis_catus-cat_on_snow.jpg"
img <- base_loader(img_url)

# 2. Convert to tensor (RGB only), resize and normalize
input <- img %>%
 transform_to_tensor() %>%
 transform_resize(c(224, 224)) %>%
 transform_normalize(norm_mean, norm_std)
batch <- input$unsqueeze(1)

# 3. Load pretrained models
model_small <- convnext_tiny_1k(pretrained = TRUE, root = tempdir())
model_small$eval()

# 4. Forward pass
output_s <- model_small(batch)

# 5. Show Top-5 predictions
topk <- output_s$topk(k = 5, dim = 2)
indices <- as.integer(topk[[2]][1, ])
scores <- as.numeric(topk[[1]][1, ])
glue::glue("{seq_along(indices)}. {imagenet_classes(indices)} ({round(scores, 2)}%)")

## End(Not run)
```

---

model_convnext_detection

ConvNeXt Detection Models (Faster R-CNN style)

Description

Object detection models combining a ConvNeXt backbone with a Feature Pyramid Network (FPN) and the Faster R-CNN detection head. The architecture mirrors `model_fasterrcnn_resnet50_fpn()`, with the ResNet backbone replaced by ConvNeXt variants. The design follows the paper A ConvNet for the 2020s.

Available Models:

- `model_convnext_tiny_detection()`
- `model_convnext_small_detection()`
- `model_convnext_base_detection()`

Backbone Performance (ImageNet-1k):

Accuracy metrics reflect backbone classification performance only. Detection head weights are randomly initialized and must be fine-tuned on task-specific labelled data before meaningful predictions are produced.

Model	Top-1 Acc	Top-5 Acc	Params	GFLOPS	File Size	Backbone Weights
model_convnext_tiny_detection	82.5%	96.1%	28.6M	4.46	109 MB	IMAGENET1K_V1
model_convnext_small_detection	83.6%	96.7%	50.2M	8.68	192 MB	IMAGENET1K_V1 (2
model_convnext_base_detection	84.1%	96.9%	88.6M	15.36	338 MB	IMAGENET1K_V1

FPN Channel Configuration:

Each ConvNeXt variant produces four feature maps (C2–C5) fed into the FPN. Channel widths differ between Tiny/Small and Base:

Variant	FPN in_channels	FPN out_channels
Tiny	c(96, 192, 384, 768)	256
Small	c(96, 192, 384, 768)	256
Base	c(128, 256, 512, 1024)	256

Weights Selection:

- All variants use `IMAGENET1K_V1` backbone weights by default (supervised ImageNet-1k).
- The Small variant backbone (`model_convnext_small_22k`) was additionally pretrained on ImageNet-22k prior to fine-tuning on ImageNet-1k.
- Detection head weights are **randomly initialized** — bounding-box predictions are meaningless without fine-tuning on labelled detection data.
- Set `pretrained_backbone = TRUE` to load ImageNet backbone weights.

Usage

```
model_convnext_tiny_detection(
  num_classes = 91,
  pretrained_backbone = FALSE,
  ...
)

model_convnext_small_detection(
  num_classes = 91,
  pretrained_backbone = FALSE,
  ...
)

model_convnext_base_detection(
  num_classes = 91,
  pretrained_backbone = FALSE,
  ...
)
```

Arguments

num_classes	Number of output classes excluding background (default: 90 for COCO).
pretrained_backbone	Logical. If TRUE, loads ImageNet-pretrained ConvNeXt backbone weights. Default: FALSE.
...	Other arguments (unused).

Functions

- model_convnext_tiny_detection(): ConvNeXt Tiny with FPN detection head
- model_convnext_small_detection(): ConvNeXt Small with FPN detection head
- model_convnext_base_detection(): ConvNeXt Base with FPN detection head

Note

Detection head weights are randomly initialized. Predicted bounding boxes will be arbitrary until the detection head is trained on labelled data. Only the backbone benefits from pretrained_backbone = TRUE.

See Also

Other object_detection_model: model_facenet, model_fasterrcnn, model_maskrcnn

Examples

```
## Not run:
library(magrittr)
norm_mean <- c(0.485, 0.456, 0.406) # ImageNet normalization constants
norm_std  <- c(0.229, 0.224, 0.225)

url <- paste0("https://upload.wikimedia.org/wikipedia/commons/thumb/",
              "e/ea/Morsan_Normande_vache.jpg/120px-Morsan_Normande_vache.jpg")
img <- base_loader(url) %>%
  transform_to_tensor() %>%
  transform_resize(c(520, 520))

input <- img %>% transform_normalize(norm_mean, norm_std)
batch <- input$unsqueeze(1)    # Add batch dimension: (1, 3, H, W)

# ConvNeXt Tiny detection
model <- model_convnext_tiny_detection(pretrained_backbone = TRUE)
model$eval()
# Please wait 2 mins + on CPU
pred     <- model(batch)$detections[[1]]
num_boxes <- as.integer(pred$boxes$size()[1])
topk     <- pred$scores$topk(k = 5)[[2]]
boxes    <- pred$boxes[topk, ]
labels   <- imagenet_classes(as.integer(pred$labels[topk]))

# `draw_bounding_box()` may fail if bbox values are not consistent.
```

```
if (num_boxes > 0) {
  boxed <- draw_bounding_boxes(img, boxes, labels = labels)
  tensor_image_browse(boxed)
}

## End(Not run)
```

---

model_convnext_segmentation

ConvNeXt Segmentation Models

Description

Semantic segmentation models that use a ConvNeXt backbone with either an FCN (Fully Convolutional Network) head or a UPerNet (Unified Perceptual Parsing Network) head.

These models follow the architecture patterns from mmsegmentation and can be used for semantic segmentation tasks.

Usage

```
model_convnext_tiny_fcn(
  num_classes = 21,
  aux_loss = FALSE,
  pretrained_backbone = FALSE,
  ...
)

model_convnext_small_fcn(
  num_classes = 21,
  aux_loss = FALSE,
  pretrained_backbone = FALSE,
  ...
)

model_convnext_base_fcn(
  num_classes = 21,
  aux_loss = FALSE,
  pretrained_backbone = FALSE,
  ...
)

model_convnext_tiny_upernet(
  num_classes = 21,
  aux_loss = FALSE,
  pretrained = FALSE,
  pretrained_backbone = FALSE,
```

```
  pool_scales = c(1, 2, 3, 6),
  ...
)

model_convnext_small_upernet(
  num_classes = 21,
  aux_loss = FALSE,
  pretrained = FALSE,
  pretrained_backbone = FALSE,
  pool_scales = c(1, 2, 3, 6),
  ...
)

model_convnext_base_upernet(
  num_classes = 21,
  aux_loss = FALSE,
  pretrained = FALSE,
  pretrained_backbone = FALSE,
  pool_scales = c(1, 2, 3, 6),
  ...
)
```

Arguments

num_classes	Number of output segmentation classes. Default: 21 (PASCAL VOC).
aux_loss	If TRUE, includes an auxiliary classifier branch. Default: FALSE.
pretrained_backbone	If TRUE, loads ImageNet pretrained weights for the ConvNeXt backbone. Default: FALSE.
...	Additional arguments passed to the backbone.
pretrained	If TRUE, loads convnext pretrained weights of backbone and segmentation heads.
pool_scales	Numeric vector. Pooling scales used in the Pyramid Pooling Module for UPerNet models. Default: c(1, 2, 3, 6).

Value

An nn_module representing the segmentation model.

Functions

- model_convnext_tiny_fcn(): ConvNeXt Tiny with FCN head
- model_convnext_small_fcn(): ConvNeXt Small with FCN head
- model_convnext_base_fcn(): ConvNeXt Base with FCN head
- model_convnext_tiny_upernet(): ConvNeXt Tiny with UPerNet head
- model_convnext_small_upernet(): ConvNeXt Small with UPerNet head
- model_convnext_base_upernet(): ConvNeXt Base with UPerNet head

Available FCN Models

- `model_convnext_tiny_fcn()`
- `model_convnext_small_fcn()`
- `model_convnext_base_fcn()`

Available UPerNet Models

- `model_convnext_tiny_upernet()`
- `model_convnext_small_upernet()`
- `model_convnext_base_upernet()`

See Also

Other semantic_segmentation_model: `model_deeplabv3`, `model_fcn_resnet`

Examples

```
## Not run:
library(magrittr)
norm_mean <- c(0.485, 0.456, 0.406) # ImageNet normalization constants
norm_std  <- c(0.229, 0.224, 0.225)

# Use a publicly available image
wmc <- "https://upload.wikimedia.org/wikipedia/commons/thumb/"
url <- "e/ea/Morsan_Normande_vache.jpg/120px-Morsan_Normande_vache.jpg"
img <- base_loader(paste0(wmc, url))

input <- img %>%
  transform_to_tensor() %>%
  transform_resize(c(520, 520)) %>%
  transform_normalize(norm_mean, norm_std)
batch <- input$unsqueeze(1)

# ConvNeXt Tiny FCN segmentation
model <- model_convnext_tiny_fcn(num_classes = 21, pretrained_backbone = TRUE)
model$eval()
output <- model(batch)

# Visualize result
segmented <- draw_segmentation_masks(input, output$out$squeeze(1))
tensor_image_display(segmented)

# ConvNeXt Tiny UPerNet segmentation
model <- model_convnext_tiny_upernet(num_classes = 21, pretrained_backbone = TRUE)
model$eval()
output <- model(batch)

# Visualize result
segmented <- draw_segmentation_masks(input, output$out$squeeze(1))
tensor_image_display(segmented)
```

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

`model_deeplabv3` *DeepLabV3 Semantic Segmentation Models*

Description

Semantic segmentation models implementing the DeepLabV3 architecture from Rethinking Atrous Convolution for Semantic Image Segmentation. These models use Atrous Spatial Pyramid Pooling (ASPP) to capture multi-scale context, and are available with ResNet-50 and ResNet-101 backbones.

Available Models:

- `model_deeplabv3_resnet50()`
- `model_deeplabv3_resnet101()`

Model Variants and Performance (COCO val2017, VOC labels):

All models are trained on a 20-class subset of COCO that corresponds to Pascal VOC categories, plus background (21 classes total).

Model	mIoU	Pixel Acc	Params	GFLOPS	File Size	Weights Used
model_deeplabv3_resnet50	66.4%	92.4%	42.0M	178.72	161 MB	COCO_WITH_VOC_LABELS_V1
model_deeplabv3_resnet101	67.4%	92.4%	61.0M	258.74	233 MB	COCO_WITH_VOC_LABELS_V1

Weights Selection:

- All models use `COCO_WITH_VOC_LABELS_V1` weights, trained on COCO with the 20 Pascal VOC categories (+ background = 21 classes).
- Backbone weights default to `IMAGENET1K_V1` (supervised ImageNet-1k) when `pretrained = FALSE` and `pretrained_backbone = TRUE`.
- When `pretrained = TRUE`, backbone weights are overridden by the full segmentation model weights and `pretrained_backbone` is ignored.
- The auxiliary classifier branch (`aux_loss`) is automatically enabled when loading pretrained weights; set explicitly when training from scratch.

Input Format:

Models expect input tensors of shape `(batch_size, 3, H, W)`, normalized with ImageNet mean `c(0.485, 0.456, 0.406)` and std `c(0.229, 0.224, 0.225)`. Training resolution is 520x520.

Output Format:

Returns a named list with:

- `$out` — main segmentation logits, shape `(batch, num_classes, H, W)`
- `$aux` — auxiliary logits from an intermediate backbone layer (only when `aux_loss = TRUE`)

Usage

```
model_deeplabv3_resnet50(
  pretrained = FALSE,
  progress = TRUE,
  num_classes = 21,
  aux_loss = NULL,
  pretrained_backbone = FALSE,
  ...
)

model_deeplabv3_resnet101(
  pretrained = FALSE,
  progress = TRUE,
  num_classes = 21,
  aux_loss = NULL,
  pretrained_backbone = FALSE,
  ...
)
```

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.
num_classes	Integer. Number of output segmentation classes including background. Default: 21 (Pascal VOC). Set to NULL to infer from pretrained weights.
aux_loss	Logical or NULL. If TRUE, adds an auxiliary FCN classifier head at an intermediate backbone layer, used as a secondary loss during training. If NULL (default), inferred from pretrained weights.
pretrained_backbone	Logical. If TRUE and pretrained = FALSE, loads IMAGENET1K_V1 weights for the backbone only. Ignored when pretrained = TRUE. Default: TRUE.
...	Other parameters passed to the resnet model.

Functions

- model_deeplabv3_resnet50(): DeepLabV3 with ResNet-50 backbone
- model_deeplabv3_resnet101(): DeepLabV3 with ResNet-101 backbone

See Also

Other semantic_segmentation_model: model_convnext_segmentation, model_fcn_resnet

Examples

```
## Not run:
library(magrittr)
norm_mean <- c(0.485, 0.456, 0.406)
```

```
norm_std  <- c(0.229, 0.224, 0.225)

url <- paste0("https://upload.wikimedia.org/wikipedia/commons/thumb/",
           "e/ea/Morsan_Normande_vache.jpg/120px-Morsan_Normande_vache.jpg")
img <- base_loader(url)

input <- img %>%
  transform_to_tensor() %>%
  transform_resize(c(520, 520)) %>%
  transform_normalize(norm_mean, norm_std)
batch <- input$unsqueeze(1)    # Add batch dimension: (1, 3, H, W)

# --- ResNet-50 backbone ---
model <- model_deeplabv3_resnet50(pretrained = TRUE)
model$eval()
output <- model(batch)

segmented <- draw_segmentation_masks(input, output$out$squeeze(1))
tensor_image_browse(segmented)

# Show most frequent class
mask_id <- output$out$argmax(dim = 2)  # (1, H, W)
class_contingency_with_background <- mask_id$view(-1)$bincount()
class_contingency_with_background[1] <- 0L # we clean the counter for background class id 1
top_class_index <- class_contingency_with_background$argmax()$item()
cli::cli_inform("Majority class {.pkg ResNet-50}: {.emph {pascal_voc_classes(top_class_index)}}")

# --- ResNet-101 backbone ---
model <- model_deeplabv3_resnet101(pretrained = TRUE)
model$eval()
output <- model(batch)

segmented <- draw_segmentation_masks(input, output$out$squeeze(1))
tensor_image_browse(segmented)

# Show most frequent class
mask_id <- output$out$argmax(dim = 2)  # (1, H, W)
class_contingency_with_background <- mask_id$view(-1)$bincount()
class_contingency_with_background[1] <- 0L # we clean the counter for background class id 1
top_class_index <- class_contingency_with_background$argmax()$item()
cli::cli_inform("Majority class {.pkg ResNet-50}: {.emph {pascal_voc_classes(top_class_index)}}")

## End(Not run)
```

`model_efficientnet` *EfficientNet Models*

Description

Constructs EfficientNet model architectures as described in *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. These models are designed for image classification tasks and provide a balance between accuracy and computational efficiency through compound scaling.

Usage

```
model_efficientnet_b0(pretrained = FALSE, progress = TRUE, ...)

model_efficientnet_b1(pretrained = FALSE, progress = TRUE, ...)

model_efficientnet_b2(pretrained = FALSE, progress = TRUE, ...)

model_efficientnet_b3(pretrained = FALSE, progress = TRUE, ...)

model_efficientnet_b4(pretrained = FALSE, progress = TRUE, ...)

model_efficientnet_b5(pretrained = FALSE, progress = TRUE, ...)

model_efficientnet_b6(pretrained = FALSE, progress = TRUE, ...)

model_efficientnet_b7(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, such as `num_classes` to change the output dimension.

Functions

- `model_efficientnet_b0()`: EfficientNet B0 model
- `model_efficientnet_b1()`: EfficientNet B1 model
- `model_efficientnet_b2()`: EfficientNet B2 model
- `model_efficientnet_b3()`: EfficientNet B3 model
- `model_efficientnet_b4()`: EfficientNet B4 model
- `model_efficientnet_b5()`: EfficientNet B5 model
- `model_efficientnet_b6()`: EfficientNet B6 model
- `model_efficientnet_b7()`: EfficientNet B7 model

Task

Image classification with 1000 output classes by default (ImageNet).

Input Format

The models expect input tensors of shape `(batch_size, 3, H, W)`, where H and W should typically be 224 for B0 and scaled versions for B1–B7 (e.g., B7 uses 600x600).

Variants and Scaling

Model	Width	Depth	Resolution	Params (M)	GFLOPs	Top-1 Acc.
B0	1.0	1.0	224	5.3	0.39	77.1
B1	1.0	1.1	240	7.8	0.70	79.1
B2	1.1	1.2	260	9.2	1.00	80.1
B3	1.2	1.4	300	12.0	1.80	81.6
B4	1.4	1.8	380	19.0	4.20	82.9
B5	1.6	2.2	456	30.0	9.90	83.6
B6	1.8	2.6	528	43.0	19.0	84.0
B7	2.0	3.1	600	66.0	37.0	84.3

See Also

Other classification_model: `model_alexnet()`, `model_convnext`, `model_efficientnet_v2`, `model_facenet`, `model_inception_v3()`, `model_maxvit()`, `model_mobilenet_v2()`, `model_mobilenet_v3`, `model_resnet`, `model_vgg`, `model_vit`

Examples

```
## Not run:
model <- model_efficientnet_b0()
image_batch <- torch::torch_randn(1, 3, 224, 224)
output <- model(image_batch)
imagenet_classes(which.max(as.numeric(output)))

## End(Not run)

## Not run:
# Example of using EfficientNet-B5 with its native image size
model <- model_efficientnet_b5()
image_batch <- torch::torch_randn(1, 3, 456, 456)
output <- model(image_batch)
imagenet_classes(which.max(as.numeric(output)))

## End(Not run)
```

`model_efficientnet_v2` *EfficientNetV2 Models*

Description

Constructs EfficientNetV2 model architectures as described in *EfficientNetV2: Smaller Models and Faster Training*.

Usage

```
model_efficientnet_v2_s(pretrained = FALSE, progress = TRUE, ...)

model_efficientnet_v2_m(pretrained = FALSE, progress = TRUE, ...)

model_efficientnet_v2_l(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, such as `num_classes` to change the output dimension.

Functions

- `model_efficientnet_v2_s()`: EfficientNetV2-S model
- `model_efficientnet_v2_m()`: EfficientNetV2-M model
- `model_efficientnet_v2_l()`: EfficientNetV2-L model

Task

Image classification with 1000 output classes by default (ImageNet).

Input Format

The models expect input tensors of shape `(batch_size, 3, H, W)`. Typical values for `H` and `W` are 384 for V2-S, 480 for V2-M, and 512 for V2-L.

Variants

Model	Resolution	Params (M)	GFLOPs	Top-1 Acc.
V2-S	384	24	8.4	83.9
V2-M	480	55	24	85.1
V2-L	512	119	55	85.7

See Also

model_efficientnet

Other classification_model: model_alexnet(), model_convnext, model_efficientnet, model_facenet, model_inception_v3(), model_maxvit(), model_mobilenet_v2(), model_mobilenet_v3, model_resnet, model_vgg, model_vit

Examples

```
## Not run:
model <- model_efficientnet_v2_s()
input <- torch::torch_randn(1, 3, 224, 224)
output <- model(input)

# Show Top-5 predictions
topk <- output$topk(k = 5, dim = 2)
indices <- as.integer(topk[[2]][1, ])
scores <- as.numeric(topk[[1]][1, ])
glue::glue("{seq_along(indices)}. {imagenet_classes(indices)} ({round(scores, 2)}%)")

## End(Not run)
```

---

model_facenet *MTCNN Face Detection Networks*

Description

These models implement the three-stage Multi-task Cascaded Convolutional Networks (MTCNN) architecture from the paper Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks.

Usage

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

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

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

model_mtcnn(pretrained = TRUE, progress = TRUE, ...)

model_facenet_inception_resnet_v1(
  pretrained = NULL,
  classify = FALSE,
  num_classes = 10,
  dropout_prob = 0.6,
  ...
)
```

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.
classify	Logical, whether to include the classification head. Default is FALSE.
num_classes	Integer, number of output classes for classification. Default is 10.
dropout_prob	Numeric, dropout probability applied before classification. Default is 0.6.

Details

MTCNN detects faces and facial landmarks in an image through a coarse-to-fine pipeline:

- **PNet** (Proposal Network): Generates candidate face bounding boxes at multiple scales.
- **RNet** (Refine Network): Refines candidate boxes, rejecting false positives.
- **ONet** (Output Network): Produces final bounding boxes and 5-point facial landmarks.

Model Variants:

Model	Input Size	Parameters	File Size	Outputs	Notes
PNet	~12×12+	~3k	30 kB	2-class face prob + bbox reg	Fully conv, sliding window s
RNet	24×24	~30k	400 kB	2-class face prob + bbox reg	Dense layers, higher recall
ONet	48×48	~100k	2 MB	2-class prob + bbox + 5-point	Landmark detection stage

Inception-ResNet-v1 is a convolutional neural network architecture combining Inception modules with residual connections, designed for face recognition tasks. The model achieves high accuracy on standard face verification benchmarks such as LFW (Labeled Faces in the Wild).

Model Variants and Performance (LFW accuracy):

Weights	LFW Accuracy	File Size
CASIA-Webface	99.05%	111 MB
VGGFace2	99.65%	107 MB

- The CASIA-Webface pretrained weights provide strong baseline accuracy.
- The VGGFace2 pretrained weights achieve higher accuracy, benefiting from a larger, more diverse dataset.

Value

`model_mtcnn()` returns a named list with three elements:

- `boxes`: A tensor of shape `(N, 4)` with bounding box coordinates `[x1, y1, x2, y2]`.
- `landmarks`: A tensor of shape `(N, 10)` with (x, y) coordinates of 5 facial landmarks: left eye, right eye, nose, left mouth corner, right mouth corner.
- `cls`: A tensor of shape `(N, 2)` with face classification probabilities (face / non-face). The `cls` head has two classes:

  - 1: Non-face probability (background)
  - 2: Face probability — use this value for thresholding detections

(Here, N is the number of detected faces in the input image.)

model_facenet_inception_resnet_v1() returns a tensor output depending on the classify argument:

- When classify = FALSE (default): A tensor of shape (N, 512), where each row is a normalized embedding vector (L2 norm = 1). These 512-dimensional FaceNet embeddings can be compared using cosine similarity or Euclidean distance for face verification and clustering.
- When classify = TRUE: A tensor of shape (N, num_classes) containing class logits.

Functions

- model_facenet_pnet(): PNet (Proposal Network) — small fully-convolutional network for candidate face box generation.
- model_facenet_rnet(): RNet (Refine Network) — medium CNN with dense layers for refining and rejecting false positives.
- model_facenet_onet(): ONet (Output Network) — deeper CNN that outputs final bounding boxes and 5 facial landmark points.
- model_mtcnn(): MTCNN (Multi-task Cascaded Convolutional Networks) — face detection and alignment using a cascade of three neural networks
- model_facenet_inception_resnet_v1(): Inception-ResNet-v1 — high-accuracy face recognition model combining Inception modules with residual connections, pretrained on VGGFace2 and CASIA-Webface datasets

See Also

Other object_detection_model: model_convnext_detection, model_fasterrcnn, model_maskrcnn

Other object_detection_model: model_convnext_detection, model_fasterrcnn, model_maskrcnn

Other classification_model: model_alexnet(), model_convnext, model_efficientnet, model_efficientnet_v2, model_inception_v3(), model_maxvit(), model_mobilenet_v2(), model_mobilenet_v3, model_resnet, model_vgg, model_vit

Examples

```
## Not run:
# Example usage of PNet
model_pnet <- model_facenet_pnet(pretrained = TRUE)
model_pnet$eval()
input_pnet <- torch_randn(1, 3, 224, 224)
output_pnet <- model_pnet(input_pnet)
output_pnet

# Example usage of RNet
model_rnet <- model_facenet_rnet(pretrained = TRUE)
model_rnet$eval()
input_rnet <- torch_randn(1, 3, 24, 24)
```

```
output_rnet <- model_rnet(input_rnet)
output_rnet

# Example usage of ONet
model_onet <- model_facenet_onet(pretrained = TRUE)
model_onet$eval()
input_onet <- torch_randn(1, 3, 48, 48)
output_onet <- model_onet(input_onet)
output_onet

# Example usage of MTCNN
mtcnn <- model_mtcnn(pretrained = TRUE)
mtcnn$eval()
image_tensor <- torch_randn(c(1, 3, 224, 224))
out <- mtcnn(image_tensor)
out

# Load an image from the web
url <- paste0("https://upload.wikimedia.org/wikipedia/commons",
              "/b/b4/Catherine_Bell_200101233d_hr_%28cropped%29.jpg")
tmp_file <- tempfile(fileext = ".jpg")
download.file(url, tmp_file, mode = "wb")
img <- jpeg::readJPEG(tmp_file)

# Convert to torch tensor [C, H, W] normalized
input <- transform_to_tensor(img)  # [C, H, W]
batch <- input$unsqueeze(1)   # [1, C, H, W]

# Load pretrained model
model <- model_facenet_inception_resnet_v1(pretrained = "vggface2")
model$eval()
output <- model(batch)
output

# Example usage of Inception-ResNet-v1 with CASIA-Webface Weights
model <- model_facenet_inception_resnet_v1(pretrained = "casia-webface")
model$eval()
output <- model(batch)
output

## End(Not run)
```

`model_fasterrcnn` *Faster R-CNN Models*

Description

Construct Faster R-CNN model variants for object-detection task.

Usage

```
model_fasterrcnn_resnet50_fpn(
  pretrained = FALSE,
  progress = TRUE,
  num_classes = 90,
  score_thresh = 0.05,
  nms_thresh = 0.5,
  detections_per_img = 100,
  ...
)

model_fasterrcnn_resnet50_fpn_v2(
  pretrained = FALSE,
  progress = TRUE,
  num_classes = 90,
  score_thresh = 0.05,
  nms_thresh = 0.5,
  detections_per_img = 100,
  ...
)

model_fasterrcnn_mobilenet_v3_large_fpn(
  pretrained = FALSE,
  progress = TRUE,
  num_classes = 90,
  score_thresh = 0.05,
  nms_thresh = 0.5,
  detections_per_img = 100,
  ...
)

model_fasterrcnn_mobilenet_v3_large_320_fpn(
  pretrained = FALSE,
  progress = TRUE,
  num_classes = 90,
  score_thresh = 0.05,
  nms_thresh = 0.5,
  detections_per_img = 100,
  ...
)
```

Arguments

pretrained	Logical. If TRUE, loads pretrained weights from local file.
progress	Logical. Show progress bar during download (unused).
num_classes	Number of output classes excluding background (default: 90 for COCO).
score_thresh	Numeric. Minimum score threshold for detections (default: 0.05).

`nms_thresh` Numeric. Non-Maximum Suppression (NMS) IoU threshold for removing overlapping boxes (default: 0.5).

`detections_per_img` Integer. Maximum number of detections per image (default: 100).

`...` Other arguments (unused).

Value

A `fasterrcnn_model` nn_module.

Functions

- `model_fasterrcnn_resnet50_fpn()`: Faster R-CNN with ResNet-50 FPN
- `model_fasterrcnn_resnet50_fpn_v2()`: Faster R-CNN with ResNet-50 FPN V2
- `model_fasterrcnn_mobilenet_v3_large_fpn()`: Faster R-CNN with MobileNet V3 Large FPN
- `model_fasterrcnn_mobilenet_v3_large_320_fpn()`: Faster R-CNN with MobileNet V3 Large 320 FPN

Task

Object detection over images with bounding boxes and class labels.

Input Format

Input images should be `torch_tensors` of shape `(batch_size, 3, H, W)` where `H` and `W` are typically around 800.

Available Models

- `model_fasterrcnn_resnet50_fpn()`
- `model_fasterrcnn_resnet50_fpn_v2()`
- `model_fasterrcnn_mobilenet_v3_large_fpn()`
- `model_fasterrcnn_mobilenet_v3_large_320_fpn()`

See Also

Other object_detection_model: `model_convnext_detection`, `model_facenet`, `model_maskrcnn`

Examples

```
## Not run:
library(magrittr)
# ImageNet normalization constants, see https://pytorch.org/vision/stable/models.html
norm_mean <- c(0.485, 0.456, 0.406)
norm_std  <- c(0.229, 0.224, 0.225)
# Use a publicly available image of an animal
url <- paste0("https://upload.wikimedia.org/wikipedia/commons/thumb/",
```

```
      "e/ea/Morsan_Normande_vache.jpg/120px-Morsan_Normande_vache.jpg")
image <- magick_loader(url) %>%
  transform_to_tensor() %>%
  transform_resize(c(520, 520))
# ResNet backbone requires image normalization
input <- image  %>%
  transform_normalize(norm_mean, norm_std)
batch_normalized <- input$unsqueeze(1)    # Add batch dimension (1, 3, H, W)

# ResNet-50 FPN V2
model <- model_fasterrcnn_resnet50_fpn_v2(pretrained = TRUE, , detections_per_img = 5 )
model$eval()
torch::with_no_grad({pred <- model(batch_normalized)$detections[[1]]})
labels <- coco_classes(as.integer(pred$labels))

# Visualize boxes
labels <- coco_classes(as.integer(pred$labels))
boxed <- draw_bounding_boxes(image, pred$boxes, labels = labels)
tensor_image_browse(boxed)

# MobileNet V3 Large 320 FPN
batch <- image$unsqueeze(1)    # Add batch dimension (1, 3, H, W)
model <- model_fasterrcnn_mobilenet_v3_large_320_fpn(
  pretrained = TRUE, score_thresh = 0.02, nms_thresh = 0.8, detections_per_img = 5
)
model$eval()
torch::with_no_grad({pred <- model(batch)$detections[[1]]})

# Visualize boxes
labels <- coco_classes(as.integer(pred$labels))
boxed <- draw_bounding_boxes(image, pred$boxes, labels = labels)
tensor_image_browse(boxed)

## End(Not run)
```

`model_fcn_resnet` *Fully Convolutional Network for Semantic Segmentation*

Description

Constructs an FCN (Fully Convolutional Network) model for semantic image segmentation, based on a ResNet backbone as described in Fully Convolutional Networks for Semantic Segmentation.

Usage

```
model_fcn_resnet50(
  pretrained = FALSE,
  progress = TRUE,
  num_classes = 21,
```

```
  aux_loss = NULL,
  pretrained_backbone = TRUE,
  ...
)

model_fcn_resnet101(
  pretrained = FALSE,
  progress = TRUE,
  num_classes = 21,
  aux_loss = NULL,
  pretrained_backbone = 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.
num_classes	Number of output classes. Default: 21.
aux_loss	If TRUE, includes the auxiliary classifier. If NULL, defaults to TRUE when pretrained = TRUE.
pretrained_backbone	If TRUE, uses a backbone pre-trained on ImageNet.
...	Additional arguments passed to the backbone implementation.

Details

The 21 output classes follow the PASCAL VOC convention: background, aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, dining table, dog, horse, motorbike, person, potted plant, sheep, sofa, train, tv/monitor.

Pretrained weights require num_classes = 21.

Value

An nn_module representing the FCN model.

See Also

Other semantic_segmentation_model: model_convnext_segmentation, model_deeplabv3

Examples

```
## Not run:
library(magrittr)
norm_mean <- c(0.485, 0.456, 0.406) # ImageNet normalization constants, see
# https://pytorch.org/vision/stable/models.html
norm_std  <- c(0.229, 0.224, 0.225)
img_url <- "https://en.wikipedia.org/wiki/Special:FilePath/Felis_catus-cat_on_snow.jpg"
```

```
img <- base_loader(img_url)

input <- img %>%
 transform_to_tensor() %>%
 transform_resize(c(520, 520)) %>%
 transform_normalize(norm_mean, norm_std)
batch <- input$unsqueeze(1)

model <- model_fcn_resnet50(pretrained = TRUE)
model$eval()
output <- model(batch)

# visualize the result
# `draw_segmentation_masks()` turns the torch_float output into a boolean mask internaly:
segmented <- draw_segmentation_masks(input, output$out$squeeze(1))
tensor_image_display(segmented)


model <- model_fcn_resnet101(pretrained = TRUE)
model$eval()
output <- model(batch)

# visualize the result
segmented <- draw_segmentation_masks(input, output$out$squeeze(1))
tensor_image_display(segmented)

## End(Not run)
```

`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 classification_model: `model_alexnet()`, `model_convnext`, `model_efficientnet`, `model_efficientnet_v2`, `model_facenet`, `model_maxvit()`, `model_mobilenet_v2()`, `model_mobilenet_v3`, `model_resnet`, `model_vgg`, `model_vit`

`model_maskrcnn` *Mask R-CNN Models*

Description

Construct Mask R-CNN model variants for instance segmentation task. Mask R-CNN extends Faster R-CNN by adding a mask prediction branch that outputs segmentation masks for each detected object.

Usage

```
model_maskrcnn_resnet50_fpn(
  pretrained = FALSE,
  progress = TRUE,
  num_classes = 90,
  score_thresh = 0.05,
  nms_thresh = 0.5,
  detections_per_img = 100,
  ...
)

model_maskrcnn_resnet50_fpn_v2(
  pretrained = FALSE,
  progress = TRUE,
  num_classes = 90,
  score_thresh = 0.05,
  nms_thresh = 0.5,
  detections_per_img = 100,
  ...
)
```

Arguments

`pretrained`	Logical. If TRUE, loads pretrained weights from local file.
`progress`	Logical. Show progress bar during download (unused).
`num_classes`	Number of output classes excluding background (default: 90 for COCO).

score_thresh	Numeric. Minimum score threshold for detections (default: 0.05).
nms_thresh	Numeric. Non-Maximum Suppression (NMS) IoU threshold for removing overlapping boxes (default: 0.5).
detections_per_img	Integer. Maximum number of detections per image (default: 100).
...	Other arguments (unused).

Value

A maskrcnn_model nn_module.

Functions

- model_maskrcnn_resnet50_fpn(): Mask R-CNN with ResNet-50 FPN
- model_maskrcnn_resnet50_fpn_v2(): Mask R-CNN with ResNet-50 FPN V2

Task

Instance segmentation over images with bounding boxes, class labels, and segmentation masks.

Input Format

Input images should be torch_tensors of shape (batch_size, 3, H, W) where H and W are typically around 800.

Output Format

Returns a list with:

- features: Feature maps from the backbone
- detections: List containing:
  - boxes: Bounding boxes (N, 4)
  - labels: Class labels (N)
  - scores: Confidence scores (N)
  - masks: Segmentation masks (N, 28, 28)

Available Models

- model_maskrcnn_resnet50_fpn()
- model_maskrcnn_resnet50_fpn_v2()

See Also

Other object_detection_model: model_convnext_detection, model_facenet, model_fasterrcnn

Examples

```
## Not run:
library(magrittr)
# ImageNet normalization constants, see https://pytorch.org/vision/stable/models.html
norm_mean <- c(0.485, 0.456, 0.406)
norm_std  <- c(0.229, 0.224, 0.225)

# Load an image
url <- paste0("https://upload.wikimedia.org/wikipedia/commons/thumb/",
              "e/ea/Morsan_Normande_vache.jpg/120px-Morsan_Normande_vache.jpg")
img <- base_loader(url)

input <- img %>%
  transform_to_tensor() %>%
  transform_resize(c(800, 800)) %>%
  transform_normalize(norm_mean, norm_std)
batch <- input$unsqueeze(1)

# Mask R-CNN ResNet-50 FPN
model <- model_maskrcnn_resnet50_fpn(pretrained = TRUE, , detections_per_img = 5)
model$eval()

torch::with_no_grad({pred <- model(batch)$detections[[1]]})

# Visualize boxes
labels <- coco_classes(as.integer(pred$labels))
boxed <- draw_bounding_boxes(image, pred$boxes, labels = labels)
tensor_image_browse(boxed)

## End(Not run)
```

`model_maxvit` *MaxViT Model*

Description

Implementation of the MaxViT architecture described in MaxViT: Multi-Axis Vision Transformer. The model performs image classification and by default returns logits for 1000 ImageNet classes.

Usage

```
model_maxvit(pretrained = FALSE, progress = TRUE, num_classes = 1000, ...)
```

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.
`num_classes`	(integer) Number of output classes.
`...`	Additional parameters passed to the model initializer.

See Also

Other classification_model: model_alexnet(), model_convnext, model_efficientnet, model_efficientnet_v2, model_facenet, model_inception_v3(), model_mobilenet_v2(), model_mobilenet_v3, model_resnet, model_vgg, model_vit

Examples

```
## Not run:
library(magrittr)
# 1. Load the basketball image
img_url <- "https://upload.wikimedia.org/wikipedia/commons/7/7a/Basketball.png"
img <- base_loader(img_url)

# 2. Define normalization (ImageNet)
norm_mean <- c(0.485, 0.456, 0.406)
norm_std <- c(0.229, 0.224, 0.225)

# 3. Preprocess: convert to tensor, resize, Normalize
input <- img %>%
  transform_to_tensor() %>%
  transform_resize(c(400, 400)) %>%
  transform_normalize(norm_mean, norm_std)
batch <- input$unsqueeze(1)    # Add batch dimension (1, 3, H, W)

# 4. Display the image before normalization
tensor_image_browse(input)

# 5. Load MaxViT model
model <- model_maxvit(pretrained = TRUE)
model$eval()

# 6. Run inference
output <- model(batch)
topk <- output$topk(k = 5, dim = 2)
indices <- as.integer(topk[[2]][1, ])
scores <- as.numeric(topk[[1]][1, ])

# 7. Show Top-5 predictions
glue::glue("{seq_along(indices)}. {imagenet_classes(indices)} ({round(scores, 2)}%)")

## End(Not run)
```

`model_mobilenet_v2` *MobileNetV2 Model*

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 classification_model: model_alexnet(), model_convnext, model_efficientnet, model_efficientnet_v2, model_facenet, model_inception_v3(), model_maxvit(), model_mobilenet_v3, model_resnet, model_vgg, model_vit

---

model_mobilenet_v3 *MobileNetV3 Model*

Description

MobileNetV3 is a state-of-the-art lightweight convolutional neural network architecture designed for mobile and embedded vision applications. This implementation follows the design and optimizations presented in the original paper:MobileNetV3: Searching for MobileNetV3

This function mirrors torchvision::quantization::mobilenet_v3_large and loads quantized weights when pretrained is TRUE.

Usage

```
model_mobilenet_v3_large(
  pretrained = FALSE,
  progress = TRUE,
  num_classes = 1000,
  width_mult = 1
)

model_mobilenet_v3_small(
  pretrained = FALSE,
  progress = TRUE,
  num_classes = 1000,
  width_mult = 1
)

model_mobilenet_v3_large_quantized(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.
num_classes	number of output classes (default: 1000).
width_mult	width multiplier for model scaling (default: 1.0).
...	Other parameters passed to the model implementation.

Details

The model includes two variants:

- model_mobilenet_v3_large()
- model_mobilenet_v3_small()

Both variants utilize efficient blocks such as inverted residuals, squeeze-and-excitation (SE) modules, and hard-swish activations for improved accuracy and efficiency.

Model Summary and Performance for pretrained weights:

Model	Top-1 Acc	Top-5 Acc	Params	GFLOPS	File Size	Notes
MobileNetV3 Large	74.04%	91.34%	5.48M	0.22	21.1 MB	Trained from scratch, simple
MobileNetV3 Small	67.67%	87.40%	2.54M	0.06	9.8 MB	Improved recipe over origina

Functions

- model_mobilenet_v3_large(): MobileNetV3 Large model with about 5.5 million parameters.
- model_mobilenet_v3_small(): MobileNetV3 Small model with about 2.5 million parameters.

See Also

Other classification_model: model_alexnet(), model_convnext, model_efficientnet, model_efficientnet_v2, model_facenet, model_inception_v3(), model_maxvit(), model_mobilenet_v2(), model_resnet, model_vgg, model_vit

Other classification_model: model_alexnet(), model_convnext, model_efficientnet, model_efficientnet_v2, model_facenet, model_inception_v3(), model_maxvit(), model_mobilenet_v2(), model_resnet, model_vgg, model_vit

Examples

```
## Not run:
# 1. Download sample image (dog)
norm_mean <- c(0.485, 0.456, 0.406) # ImageNet normalization constants, see
# https://pytorch.org/vision/stable/models.html
norm_std  <- c(0.229, 0.224, 0.225)
img_url <- "https://en.wikipedia.org/wiki/Special:FilePath/Felis_catus-cat_on_snow.jpg"
img <- base_loader(img_url)
```

```
# 2. Convert to tensor (RGB only), resize and normalize
input <- img %>%
 transform_to_tensor() %>%
 transform_resize(c(224, 224)) %>%
 transform_normalize(norm_mean, norm_std)
batch <- input$unsqueeze(1)

# 3. Load pretrained models
model_small <- model_mobilenet_v3_small(pretrained = TRUE)
model_small$eval()

# 4. Forward pass
output_s <- model_small(batch)

# 5. Top-5 printing helper
topk <- output_s$topk(k = 5, dim = 2)
indices <- as.integer(topk[[2]][1, ])
scores <- as.numeric(topk[[1]][1, ])

# 6. Show Top-5 predictions
glue::glue("{seq_along(indices)}. {imagenet_classes(indices)} ({round(scores, 2)}%)")

# 7. Same with large model
model_large <- model_mobilenet_v3_large(pretrained = TRUE)
model_large$eval()
output_l <- model_large(input)
topk <- output_l$topk(k = 5, dim = 2)
indices <- as.integer(topk[[2]][1, ])
scores <- as.numeric(topk[[1]][1, ])
glue::glue("{seq_along(indices)}. {imagenet_classes(indices)} ({round(scores, 2)}%)")

## End(Not run)
```

`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 classification_model: `model_alexnet()`, `model_convnext`, `model_efficientnet`, `model_efficientnet_v2`, `model_facenet`, `model_inception_v3()`, `model_maxvit()`, `model_mobilenet_v2()`, `model_mobilenet_v3`, `model_vgg`, `model_vit`

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 classification_model: model_alexnet(), model_convnext, model_efficientnet, model_efficientnet_v2, model_facenet, model_inception_v3(), model_maxvit(), model_mobilenet_v2(), model_mobilenet_v3, model_resnet, model_vit

---

model_vit *Vision Transformer Implementation*

Description

Vision Transformer (ViT) models implement the architecture proposed in the paper An Image is Worth 16x16 Words. These models are designed for image classification tasks and operate by treating image patches as tokens in a Transformer model.

Usage

```
model_vit_b_16(pretrained = FALSE, progress = TRUE, ...)

model_vit_b_32(pretrained = FALSE, progress = TRUE, ...)

model_vit_l_16(pretrained = FALSE, progress = TRUE, ...)

model_vit_l_32(pretrained = FALSE, progress = TRUE, ...)

model_vit_h_14(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.

Details

Model Variants and Performance (ImageNet-1k):

Model	Top-1 Acc	Top-5 Acc	Params	GFLOPS	File Size	Weights Used	Notes
vit_b_16	81.1%	95.3%	86.6M	17.56	346 MB	IMAGENET1K_V1	Base, 16x16 patc
vit_b_32	75.9%	92.5%	88.2M	4.41	353 MB	IMAGENET1K_V1	Base, 32x32 patc
vit_l_16	79.7%	94.6%	304.3M	61.55	1.22 GB	IMAGENET1K_V1	Large, 16x16 pa
vit_l_32	77.0%	93.1%	306.5M	15.38	1.23 GB	IMAGENET1K_V1	Large, 32x32 pa
vit_h_14	88.6%	98.7%	633.5M	1016.7	2.53 GB	IMAGENET1K_SWAG_E2E_V1	Huge, 14x14

- **TorchVision Recipe**: https://github.com/pytorch/vision/tree/main/references/classification
- **SWAG Recipe**: https://github.com/facebookresearch/SWAG

Weights Selection:

- All models use the default `IMAGENET1K_V1` weights for consistency, stability, and official support from TorchVision.
- These are supervised weights trained on ImageNet-1k.
- For `vit_h_14`, the default weight is `IMAGENET1K_SWAG_E2E_V1`, pretrained on SWAG and fine-tuned on ImageNet.

Functions

- `model_vit_b_16()`: ViT-B/16 model (Base, 16×16 patch size)
- `model_vit_b_32()`: ViT-B/32 model (Base, 32×32 patch size)
- `model_vit_l_16()`: ViT-L/16 model (Base, 16×16 patch size)
- `model_vit_l_32()`: ViT-L/32 model (Base, 32×32 patch size)
- `model_vit_h_14()`: ViT-H/14 model (Base, 14×14 patch size)

See Also

Other classification_model: `model_alexnet()`, `model_convnext`, `model_efficientnet`, `model_efficientnet_v2`, `model_facenet`, `model_inception_v3()`, `model_maxvit()`, `model_mobilenet_v2()`, `model_mobilenet_v3`, `model_resnet`, `model_vgg`

`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()`, `image_folder_dataset()`, `lfw_dataset`, `mnist_dataset()`, `places365_dataset()`, `tiny_imagenet_dataset()`, `vggface2_dataset()`, `whoi_plankton_dataset()`, `whoi_small_coralnet_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.

See Also

Other segmentation_dataset: coco_segmentation_dataset(), pascal_voc_datasets, rf100_peixos_segmentation_da

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 and label (trimap in raw format)
first_item <- oxfordiiitpet[1]
first_item$x  # RGB image tensor of shape (3, H, W)
first_item$y$trimap  # (H, W) integer tensor: 1=pet, 2=background, 3=outline
first_item$y$label  # Integer label (1-37 or 1-2 depending on target_type)
oxfordiiitpet$classes[first_item$y$label] # Class name of the label

# Load dataset with explicit segmentation mask transformation
oxfordiiitpet_masked <- oxfordiiitpet_segmentation_dataset(
   transform = transform_to_tensor,
   target_transform = target_transform_trimap_masks,
   download = TRUE
)

masked_item <- oxfordiiitpet_masked[1]
masked_item$y$masks  # (3, H, W) bool tensor: pet, background, outline

# Visualize segmentation masks
overlay <- draw_segmentation_masks(masked_item)
tensor_image_browse(overlay)

## End(Not run)
```

---

pascal_voc_classes *Pascal VOC Segmentation Dataset*

Description

The Pascal Visual Object Classes (VOC) dataset is a widely used benchmark for object detection and semantic segmentation tasks in computer vision.

Usage

```
pascal_voc_classes(class_id = 1:21)
```

Arguments

class_id	Integer vector of 1-based class identifiers. Must be within [1, 21].

Details

This dataset provides RGB images along with per-pixel class segmentation masks for 20 object categories, plus a background class. Each pixel in the mask is labeled with a class index corresponding to one of the predefined semantic categories.

The VOC dataset was released in yearly editions (2007 to 2012), with slight variations in data splits and annotation formats. Notably, only the 2007 edition includes a separate `test` split; all other years (2008–2012) provide only the `train`, `val`, and `trainval` splits.

The dataset defines 21 semantic classes: `"background"`, `"aeroplane"`, `"bicycle"`, `"bird"`, `"boat"`, `"bottle"`, `"bus"`, `"car"`, `"cat"`, `"chair"`, `"cow"`, `"dining table"`, `"dog"`, `"horse"`, `"motorbike"`, `"person"`, `"potted plant"`, `"sheep"`, `"sofa"`, `"train"`, and `"tv/monitor"`. They are available through the `classes` variable of the dataset object.

See Also

Other class_resolution: `caltech_classes()`, `coco_classes()`, `imagenet_classes()`

---

pascal_voc_datasets *Pascal VOC Datasets*

Description

This dataset is frequently used for training and evaluating semantic segmentation models, and supports tasks requiring dense, per-pixel annotations.

Usage

```
pascal_segmentation_dataset(
  root = tempdir(),
  year = "2012",
  split = "train",
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)

pascal_detection_dataset(
  root = tempdir(),
  year = "2012",
  split = "train",
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)
```

Arguments

root	Character. Root directory where the dataset will be stored under root/pascal_voc_<year>.
year	Character. VOC dataset version to use. One of "2007", "2008", "2009", "2010", "2011", or "2012". Default is "2012".
split	Character. One of "train", "val", "trainval", or "test". Determines the dataset split. Default is "train".
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 of class pascal_segmentation_dataset.

The returned list inherits class image_with_segmentation_mask, which allows generic visualization utilities to be applied.

Each element is a named list with the following structure:

- x: a H x W x 3 array representing the RGB image.
- y: A named list containing:
  - masks: A torch_tensor of dtype bool and shape (21, H, W), representing a multi-channel segmentation mask. Each of the 21 channels corresponds to a Pascal VOC classes
  - labels: An integer vector indicating the indices of the classes present in the mask.

A torch dataset of class pascal_detection_dataset.

The returned list inherits class image_with_bounding_box, which allows generic visualization utilities to be applied.

Each element is a named list:

- x: a H x W x 3 array representing the RGB image.
- y: a list with:
  - labels: a character vector with object class names.
  - boxes: a tensor of shape (N, 4) with bounding box coordinates in (xmin, ymin, xmax, ymax) format.

See Also

Other segmentation_dataset: coco_segmentation_dataset(), oxfordiiitpet_segmentation_dataset(), rf100_peixos_segmentation_dataset()

Other detection_dataset: coco_detection_dataset(), rf100_biology_collection(), rf100_damage_collection(), rf100_document_collection(), rf100_infrared_collection(), rf100_medical_collection(), rf100_underwater_collection()

Examples

```
## Not run:
# Load Pascal VOC segmentation dataset (2007 train split)
pascal_seg <- pascal_segmentation_dataset(
 transform = transform_to_tensor,
 download = TRUE,
 year = "2007"
)

# Access the first image and its mask
first_item <- pascal_seg[1]
first_item$x  # Image
first_item$y$masks  # Segmentation mask
first_item$y$labels  # Unique class labels in the mask
pascal_voc_classes(first_item$y$labels)  # Class names

# Visualise the first image and its mask
masked_img <- draw_segmentation_masks(first_item)
tensor_image_browse(masked_img)

# Load Pascal VOC detection dataset (2007 train split)
pascal_det <- pascal_detection_dataset(
 transform = transform_to_tensor,
 download = TRUE,
 year = "2007"
)

# Access the first image and its bounding boxes
first_item <- pascal_det[1]
first_item$x  # Image
first_item$y$labels  # Object labels
first_item$y$boxes  # Bounding box tensor

# Visualise the first image with bounding boxes
boxed_img <- draw_bounding_boxes(first_item)
tensor_image_browse(boxed_img)

## End(Not run)
```

---

places365_dataset *Places365 Dataset*

Description

Loads the MIT Places365 dataset for scene classification.

Usage

```
places365_dataset(
  root = tempdir(),
  split = c("train", "val", "test"),
  transform = NULL,
  target_transform = NULL,
  download = FALSE,
  loader = magick_loader
)

places365_dataset_large(
  root = tempdir(),
  split = c("train", "val", "test"),
  transform = NULL,
  target_transform = NULL,
  download = FALSE,
  loader = magick_loader
)
```

Arguments

root	Root directory for dataset storage. The dataset will be stored under root/<dataset-name>. Defaults to tempdir().
split	One of "train", "val", or "test".
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.
loader	A function to load an image given its path. Defaults to magick_loader(), which uses the {magick} package.

Details

The dataset provides three splits: "train", "val", and "test". Folder structure and image layout on disk are handled internally by the loader.

This function downloads and prepares the smaller 256x256 image version (~30 GB). For the high-resolution variant (~160 GB), use places365_dataset_large(). Note that images in the large version come in varying sizes, so resizing may be needed before batching.

The test split corresponds to the *private* evaluation set used in the Places365 challenge. Annotation files are not publicly released, so only the images are provided.

Value

A torch dataset of class places365_dataset. Each element is a named list with:

- x: the image as loaded (or transformed if `transform` is set).
- y: the integer class label. For the `test` split, no labels are available and `y` will always be `NA`.

Functions

- `places365_dataset_large()`: High resolution variant (~160 GB).

See Also

Other classification_dataset: `caltech_dataset`, `cifar10_dataset()`, `eurosat_dataset()`, `fer_dataset()`, `fgvc_aircraft_dataset()`, `flowers102_dataset()`, `image_folder_dataset()`, `lfw_dataset`, `mnist_dataset()`, `oxfordiiitpet_dataset()`, `tiny_imagenet_dataset()`, `vggface2_dataset()`, `whoi_plankton_dataset()`, `whoi_small_coralnet_dataset()`

Examples

```
## Not run:
ds <- places365_dataset(
  split = "val",
  download = TRUE,
  transform = transform_to_tensor
)
item <- ds[1]
tensor_image_browse(item$x)

# Show class index and label
label_idx <- item$y
label_name <- ds$classes[label_idx]
label_idx  # Class Index
label_name  # Name of the Label

dl <- dataloader(ds, batch_size = 2)
batch <- dataloader_next(dataloader_make_iter(dl))
batch$x

ds_large <- places365_dataset_large(
  split = "val",
  download = TRUE,
  transform = . %>% transform_to_tensor() %>% transform_resize(c(256, 256))
)
dl <- torch::dataloader(dataset = ds_large, batch_size = 2)
batch <- dataloader_next(dataloader_make_iter(dl))
batch$x

## 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

rf100_biology_collection *RoboFlow 100 Biology dataset Collection*

Description

Loads one of the RoboFlow 100 Biology datasets with bounding box annotations for object-detection task.

Usage

```
rf100_biology_collection(
  dataset,
  split = c("train", "test", "valid"),
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)
```

Arguments

dataset	Dataset to select within c("stomata_cell", "blood_cell", "parasite", "cell","bacteria", "cotton_disease","mitosis", "phage", "liver_disease", "moth").
split	the subset of the dataset to choose between c("train", "test", "valid").
transform	Optional transform function applied to the image.
target_transform	Optional transform function applied to the target.
download	Logical. If TRUE, downloads the dataset if not present at root.

Value

A torch dataset. Each element is a named list with:

- x: H x W x 3 array representing the image, auto-oriented and stretched to 640 x 640.
- y: a list containing the target with:
  - image_id: numeric identifier of the x image.
  - labels: numeric identifier of the N bounding-box object class.
  - boxes: a torch_tensor of shape (N, 4) with bounding boxes, each in $(x_{min}, y_{min}, x_{max}, y_{max})$ format.

The returned item inherits the class image_with_bounding_box so it can be visualised with helper functions such as draw_bounding_boxes().

See Also

Other detection_dataset: coco_detection_dataset(), pascal_voc_datasets, rf100_damage_collection(), rf100_document_collection(), rf100_infrared_collection(), rf100_medical_collection(), rf100_underwater_collection()

Examples

```
## Not run:
ds <- rf100_biology_collection(
  dataset = "stomata_cell",
  split = "test",
  transform = transform_to_tensor,
  download = TRUE
)
item <- ds[1]
boxed <- draw_bounding_boxes(item)
tensor_image_browse(boxed)

## End(Not run)
```

`rf100_damage_collection` *RoboFlow 100 Damages dataset Collection*

Description

Loads one of the RoboFlow 100 Damage & Risk assesment datasets with bounding box annotations for object-detection task.

Usage

```
rf100_damage_collection(
  dataset,
  split = c("train", "test", "valid"),
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)
```

Arguments

`dataset`	Dataset to select within `c("liquid_crystals", "solar_panel", "asbestos")`.
`split`	the subset of the dataset to choose between `c("train", "test", "valid")`.
`transform`	Optional transform function applied to the image.
`target_transform`	Optional transform function applied to the target.
`download`	Logical. If TRUE, downloads the dataset if not present at `root`.

Value

A torch dataset. Each element is a named list with:

- `x`: H x W x 3 array representing the image, auto-oriented and stretched to 640 x 640.
- `y`: a list containing the target with:
  - `image_id`: numeric identifier of the x image.
  - `labels`: numeric identifier of the N bounding-box object class.
  - `boxes`: a torch_tensor of shape (N, 4) with bounding boxes, each in $(x_{min}, y_{min}, x_{max}, y_{max})$ format.

The returned item inherits the class `image_with_bounding_box` so it can be visualised with helper functions such as `draw_bounding_boxes()`.

See Also

Other detection_dataset: `coco_detection_dataset()`, `pascal_voc_datasets`, `rf100_biology_collection()`, `rf100_document_collection()`, `rf100_infrared_collection()`, `rf100_medical_collection()`, `rf100_underwater_collection()`

Examples

```
## Not run:
ds <- rf100_damage_collection(
  dataset = "solar_panel",
  split = "test",
  transform = transform_to_tensor,
  download = TRUE
)
item <- ds[1]
boxed <- draw_bounding_boxes(item)
tensor_image_browse(boxed)

## End(Not run)
```

---

rf100_document_collection

RF100 Document Collection Datasets

Description

RoboFlow 100 Document dataset Collection

Usage

```
rf100_document_collection(
  dataset,
  split = c("train", "test", "valid"),
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)
```

Arguments

dataset	Dataset to select within c("tweeter_post", "tweeter_profile", "document_part","activity_dia "signature", "paper_part", "tabular_data", "paragraph", "currency", "wine_label").
split	the subset of the dataset to choose between c("train", "test", "valid").
transform	Optional transform function applied to the image.
target_transform	Optional transform function applied to the target.
download	Logical. If TRUE, downloads the dataset if not present at root.

Details

Loads one of the RoboFlow 100 Document datasets with bounding box annotations for object-detection task.

Value

A torch dataset. Each element is a named list with:

- x: H x W x 3 array representing the image, auto-oriented and stretched to 640 x 640.
- y: a list containing the target with:
  - image_id: numeric identifier of the x image.
  - labels: numeric identifier of the N bounding-box object class.
  - boxes: a torch_tensor of shape (N, 4) with bounding boxes, each in $(x_{min}, y_{min}, x_{max}, y_{max})$ format.

The returned item inherits the class image_with_bounding_box so it can be visualised with helper functions such as draw_bounding_boxes().

See Also

Other detection_dataset: coco_detection_dataset(), pascal_voc_datasets, rf100_biology_collection(), rf100_damage_collection(), rf100_infrared_collection(), rf100_medical_collection(), rf100_underwater_collection()

Examples

```
## Not run:
ds <- rf100_document_collection(
  dataset = "tweeter_post",
  split = "train",
  transform = transform_to_tensor,
  download = TRUE
)

# Retrieve a sample and inspect annotations
item <- ds[1]
item$y$labels
item$y$boxes

# Draw bounding boxes and display the image
boxed_img <- draw_bounding_boxes(item)
tensor_image_browse(boxed_img)

## End(Not run)
```

---

rf100_infrared_collection

RoboFlow 100 Infrared dataset Collection

Description

Loads one of the RoboFlow 100 Infrared datasets with per-dataset folders and train/valid/test splits.

Usage

```
rf100_infrared_collection(
  dataset,
  split = c("train", "test", "valid"),
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)
```

Arguments

dataset	Dataset to select within c("thermal_dog_and_people", "solar_panel", "thermal_cheetah", "ir_object").
split	the subset of the dataset to choose between c("train", "test", "valid").
transform	Optional transform function applied to the image.
target_transform	Optional transform function applied to the target.
download	Logical. If TRUE, downloads the dataset if not present at root.

Value

A torch dataset. Each element is a named list with:

- x: H x W x 3 array representing the image, auto-oriented and stretched to 640 x 640.
- y: a list containing the target with:
  - image_id: numeric identifier of the x image.
  - labels: numeric identifier of the N bounding-box object class.
  - boxes: a torch_tensor of shape (N, 4) with bounding boxes, each in $(x_{min}, y_{min}, x_{max}, y_{max})$ format.

The returned item inherits the class image_with_bounding_box so it can be visualised with helper functions such as draw_bounding_boxes().

See Also

Other detection_dataset: coco_detection_dataset(), pascal_voc_datasets, rf100_biology_collection(), rf100_damage_collection(), rf100_document_collection(), rf100_medical_collection(), rf100_underwater_collection()

Examples

```
## Not run:
ds <- rf100_infrared_collection(
  dataset = "thermal_dog_and_people",
  split = "test",
  transform = transform_to_tensor,
  download = TRUE
)
```

```
item <- ds[1]
boxed <- draw_bounding_boxes(item)
tensor_image_browse(boxed)

## End(Not run)
```

`rf100_medical_collection`

RoboFlow 100 Medical dataset Collection

Description

Loads one of the RoboFlow 100 Medical datasets with per-dataset folders and train/valid/test splits.

Usage

```
rf100_medical_collection(
  dataset,
  split = c("train", "test", "valid"),
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)
```

Arguments

`dataset`	Dataset to select within `c("radio_signal","rheumatology", "knee", "abdomen_mri", "brain_axial_mri","gynecology_mri", "brain_tumor", "fracture")`.
`split`	the subset of the dataset to choose between `c("train", "test", "valid")`.
`transform`	Optional transform function applied to the image.
`target_transform`	Optional transform function applied to the target.
`download`	Logical. If TRUE, downloads the dataset if not present at `root`.

Value

A torch dataset. Each element is a named list with:

- `x`: H x W x 3 array representing the image, auto-oriented and stretched to 640 x 640.
- `y`: a list containing the target with:
  - `image_id`: numeric identifier of the x image.
  - `labels`: numeric identifier of the N bounding-box object class.
  - `boxes`: a torch_tensor of shape (N, 4) with bounding boxes, each in $(x_{min}, y_{min}, x_{max}, y_{max})$ format.

The returned item inherits the class `image_with_bounding_box` so it can be visualised with helper functions such as `draw_bounding_boxes()`.

See Also

Other detection_dataset: coco_detection_dataset(), pascal_voc_datasets, rf100_biology_collection(), rf100_damage_collection(), rf100_document_collection(), rf100_infrared_collection(), rf100_underwater_collection()

Examples

```
## Not run:
ds <- rf100_medical_collection(
  dataset = "rheumatology",
  split = "test",
  transform = transform_to_tensor,
  download = TRUE
)
item <- ds[1]
boxed <- draw_bounding_boxes(item)
tensor_image_browse(boxed)

## End(Not run)
```

---

rf100_peixos_segmentation_dataset

RF100 Peixos Segmentation Dataset

Description

Loads the Roboflow 100 "peixos" dataset for semantic segmentation. "peixos" contains 3 splits of respectively 821 / 118 / 251 color images of size 640 x 640. Segmentation masks are generated on-the-fly from polygon annotations of the unique "fish" category.

Usage

```
rf100_peixos_segmentation_dataset(
  split = c("train", "test", "valid"),
  root = tempdir(),
  download = FALSE,
  transform = NULL,
  target_transform = NULL
)
```

Arguments

split	the subset of the dataset to choose between c("train", "test", "valid").
root	directory path to download the dataset.
download	Logical. If TRUE, downloads the dataset if not present at root.
transform	Optional transform function applied to the image.
target_transform	Optional transform function applied to the target.

Value

A torch dataset. Each element is a named list with:

- x: H × W × 3 array (use `transform_to_tensor()` in `transform` to get C × H × W tensor).
- y: a list with:
  - `masks`: boolean tensor of shape (1, H, W).
  - `labels`: integer vector with the class index (always 1 for "fish").

The returned item is given class `image_with_segmentation_mask` so it can be visualised with helpers like `draw_segmentation_masks()`.

See Also

Other segmentation_dataset: `coco_segmentation_dataset()`, `oxfordiiitpet_segmentation_dataset()`, `pascal_voc_datasets`

Examples

```
## Not run:
ds <- rf100_peixos_segmentation_dataset(
  split = "train",
  transform = transform_to_tensor,
  download = TRUE
)
item <- ds[1]
overlay <- draw_segmentation_masks(item)
tensor_image_browse(overlay)

## End(Not run)
```

---

rf100_underwater_collection

RoboFlow 100 Underwater dataset Collection

Description

Loads one of the underwater related RoboFlow 100 Environmental datasets: "pipes", "aquarium", "objects", or "coral". Images are provided with bounding box annotations for object-detection task.

Usage

```
rf100_underwater_collection(
  dataset,
  split = c("train", "test", "valid"),
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)
```

Arguments

dataset	Dataset to select within c("pipes", "aquarium", "objects", "coral").
split	the subset of the dataset to choose between c("train", "test", "valid").
transform	Optional transform function applied to the image.
target_transform	Optional transform function applied to the target.
download	Logical. If TRUE, downloads the dataset if not present at root.

Value

A torch dataset. Each element is a named list with:

- x: H x W x 3 array representing the image, auto-oriented and stretched to 640 x 640.
- y: a list containing the target with:
  - image_id: numeric identifier of the x image.
  - labels: numeric identifier of the N bounding-box object class.
  - boxes: a torch_tensor of shape (N, 4) with bounding boxes, each in $(x_{min}, y_{min}, x_{max}, y_{max})$ format.

The returned item inherits the class image_with_bounding_box so it can be visualised with helper functions such as draw_bounding_boxes().

See Also

Other detection_dataset: coco_detection_dataset(), pascal_voc_datasets, rf100_biology_collection(), rf100_damage_collection(), rf100_document_collection(), rf100_infrared_collection(), rf100_medical_collection()

Examples

```
## Not run:
ds <- rf100_underwater_collection(
  dataset = "aquarium",
  split = "train",
  transform = transform_to_tensor,
  download = TRUE
)

item <- ds[24]
# map label ids into their class names
item$y$labels <- ds$classes[item$y$labels]
boxed <- draw_bounding_boxes(item)
tensor_image_browse(boxed)

## End(Not run)
```

search_collection *Search Collection Catalog*

Description

Search through all Collection datasets by keywords in name or description, or filter by collection. This makes it easy to discover datasets relevant to your task without browsing each collection individually.

Usage

```
search_collection(keyword = NULL, collection = NULL)
```

Arguments

keyword	Character string to search for (case-insensitive). Searches in both dataset names and descriptions. If NULL, returns all datasets (optionally filtered by collection).
collection	Filter by collection name. One of: "biology", "medical", "infrared", "damage", "underwater", "document". If NULL, searches all collections.

Value

A data frame with matching datasets and their metadata. Returns NULL invisibly if no matches are found.

See Also

get_collection_catalog(), collection_catalog

Examples

```
## Not run:
# Find all medical datasets
search_collection(collection = "medical")

# Find datasets about cells
search_collection("cell")

# Find photovoltaic/solar datasets
search_collection("solar")
search_collection("photovoltaic")

# Find all biology datasets with "cell" in name/description
search_collection("cell", collection = "biology")

# List all available datasets
search_collection()
```

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

`target_transform_coco_masks` *Target Transform: COCO Polygon Segmentation to Masks*

Description

Converts COCO-style polygon segmentation annotations from target `$segmentation` variable into boolean mask tensors as target `$masks` variable in order to ease later-on visualisation via `draw_segmentation_mask()`. Use as `target_transform` in `coco_detection_dataset()`.

Usage

```
target_transform_coco_masks(y)
```

Arguments

`y`	list being COCO dataset target variable, with names `segmentation`, `image_height`, `image_width`.

Value

Modified `y` list with added `masks` field (N, H, W) boolean tensor, N being the number of classes.

See Also

Other target_transforms: `target_transform_trimap_masks()`

Examples

```
## Not run:
ds <- coco_detection_dataset(
  root = "data",
  target_transform = target_transform_coco_masks
)
item <- ds[1]
draw_segmentation_masks(item)

## End(Not run)
```

target_transform_trimap_masks *Target Transform: Trimap to Boolean Masks*

Description

Converts Oxford-IIIT Pet dataset target `$trimap` variable (values 1,2,3) into 3-channel boolean masks tensors as target `$masks` variable in order to ease later-on visualisation via `draw_segmentation_mask()`. Use as `target_transform` in `oxfordiiitpet_segmentation_dataset()`.

Usage

```
target_transform_trimap_masks(y)
```

Arguments

y	List containing `trimap` field (H, W) tensor with values 1, 2, 3

Details

Creates three mutually exclusive masks:

- Channel 1: Pet pixels (trimap == 1)
- Channel 2: Background pixels (trimap == 2)
- Channel 3: Outline pixels (trimap == 3)

Value

Modified y list with added `masks` field (3, H, W) boolean tensor

See Also

Other target_transforms: `target_transform_coco_masks()`

Examples

```
## Not run:
ds <- oxfordiiitpet_segmentation_dataset(
  root = "data",
  target_transform = target_transform_trimap_masks
)
item <- ds[1]
draw_segmentation_masks(item)

## End(Not run)
```

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(), image_folder_dataset(), lfw_dataset, mnist_dataset(), oxfordiiitpet_dataset(), places365_dataset(), vggface2_dataset(), whoi_plankton_dataset(), whoi_small_coralnet_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 unitary_transforms: transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_convert_image_dty transform_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_resize(), transform_rgb_to_grayscale(), transform_rotate(), 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 unitary_transforms: transform_adjust_brightness(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_convert_image_dty transform_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_resize(), transform_rgb_to_grayscale(), transform_rotate(), 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

Search for Gamma Correction for more details.

See Also

Other unitary_transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_convert_image_dtype(), transform_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_resize(), transform_rgb_to_grayscale(), transform_rotate(), 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]`.

Search for Hue for more details.

See Also

Other unitary_transforms: `transform_adjust_brightness()`, `transform_adjust_contrast()`, `transform_adjust_gamma()`, `transform_adjust_saturation()`, `transform_affine()`, `transform_center_crop()`, `transform_convert_image_dtype()`, `transform_crop()`, `transform_grayscale()`, `transform_hflip()`, `transform_linear_transformation()`, `transform_normalize()`, `transform_pad()`, `transform_perspective()`, `transform_resize()`, `transform_rgb_to_grayscale()`, `transform_rotate()`, `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 unitary_transforms: `transform_adjust_brightness()`, `transform_adjust_contrast()`, `transform_adjust_gamma()`, `transform_adjust_hue()`, `transform_affine()`, `transform_center_crop()`, `transform_convert_image_dtype()`, `transform_crop()`, `transform_grayscale()`, `transform_hflip()`, `transform_linear_transformation()`, `transform_normalize()`, `transform_pad()`, `transform_perspective()`, `transform_resize()`, `transform_rgb_to_grayscale()`, `transform_rotate()`, `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,
  interpolation = 0,
  fill = NULL,
  resample,
  fillcolor,
  center = 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.
`interpolation`	(int or character): Interpolation mode. Supported values are 0 / "nearest" and 2 / "bilinear". Default is 0.
`fill`	Fill color for area outside the transform. Default is NULL.
`resample`	Deprecated. Use interpolation instead.
`fillcolor`	Deprecated. Use fill instead.
`center`	Optional center of rotation, c(x, y). Default is image center.

See Also

Other unitary_transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_center_crop(transform_convert_image_dtype(), transform_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_resize(), transform_rgb_to_grayscale(), transform_rotate(), 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 unitary_transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_convert_image_dtype(), transform_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_resize(), transform_rgb_to_grayscale(), transform_rotate(), 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 random_transforms: transform_random_affine(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip()

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 unitary_transforms: transform_adjust_brightness(), transform_adjust_contrast(), transform_adjust_gamma(), transform_adjust_hue(), transform_adjust_saturation(), transform_affine(), transform_center_crop(), transform_crop(), transform_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_resize(), transform_rgb_to_grayscale(), transform_rotate(), 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 unitary_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_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_resize(), transform_rgb_to_grayscale(), transform_rotate(), 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 combining_transforms: transform_random_apply(), transform_random_choice(), transform_random_order(), transform_resized_crop(), transform_ten_crop()

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 unitary_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_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_resize(), transform_rgb_to_grayscale(), transform_rotate(), 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 unitary_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_grayscale(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_resize(), transform_rgb_to_grayscale(), transform_rotate(), 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 unitary_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_grayscale(), transform_hflip(), transform_normalize(), transform_pad(), transform_perspective(), transform_resize(), transform_rgb_to_grayscale(), transform_rotate(), 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 unitary_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_grayscale()`, `transform_hflip()`, `transform_linear_transformation()`, `transform_pad()`, `transform_perspective()`, `transform_resize()`, `transform_rgb_to_grayscale()`, `transform_rotate()`, `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 unitary_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_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_perspective(), transform_resize(), transform_rgb_to_grayscale(), transform_rotate(), 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 unitary_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_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_resize(), transform_rgb_to_grayscale(), transform_rotate(), 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,
  interpolation = 0,
  fill = 0,
  resample,
  fillcolor
)
```

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.
`interpolation`	(int or character, optional): Interpolation mode. Supported values are 0 / "nearest" and 2 / "bilinear". Default is 0.
`fill`	(tuple or int): Fill color for the area outside the transform. Default is 0. This option is not supported for Tensor input.
`resample`	Deprecated. Use interpolation instead.
`fillcolor`	Deprecated. Use fill instead.

See Also

Other random_transforms: `transform_color_jitter()`, `transform_random_crop()`, `transform_random_erasing()`, `transform_random_grayscale()`, `transform_random_horizontal_flip()`, `transform_random_perspective()`, `transform_random_resized_crop()`, `transform_random_rotation()`, `transform_random_vertical_flip()`

`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 combining_transforms: transform_five_crop(), transform_random_choice(), transform_random_order(), transform_resized_crop(), transform_ten_crop()

---

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 combining_transforms: transform_five_crop(), transform_random_apply(), transform_random_order(), transform_resized_crop(), transform_ten_crop()

---

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 random_transforms: transform_color_jitter(), transform_random_affine(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip()

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 random_transforms: transform_color_jitter(), transform_random_affine(), transform_random_crop(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip()

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 random_transforms: transform_color_jitter(), transform_random_affine(), transform_random_crop(), transform_random_erasing(), transform_random_horizontal_flip(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip()

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 random_transforms: transform_color_jitter(), transform_random_affine(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_perspective(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip()

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 combining_transforms: transform_five_crop(), transform_random_apply(), transform_random_choice(), transform_resized_crop(), transform_ten_crop()

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 random_transforms: transform_color_jitter(), transform_random_affine(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_resized_crop(), transform_random_rotation(), transform_random_vertical_flip()

---

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 random_transforms: transform_color_jitter(), transform_random_affine(), transform_random_crop(), transform_random_erasing(), transform_random_grayscale(), transform_random_horizontal_flip(), transform_random_perspective(), transform_random_rotation(), transform_random_vertical_flip()

---

transform_random_rotation

Rotate the image by angle

Description

Rotate the image by angle

Usage

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

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).
interpolation	(int, optional): Interpolation mode. 0 for nearest, 2 for bilinear. Default is 0 (nearest).

expand	(bool, optional): Optional expansion flag. If true, expands the output to make it large enough to hold the entire rotated image. If false or omitted, make the output image the same size as the input image. Note that the expand flag assumes rotation around the center and no translation.
center	(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.
resample	Deprecated. Use interpolation instead.

See Also

Other random_transforms: `transform_color_jitter()`, `transform_random_affine()`, `transform_random_crop()`, `transform_random_erasing()`, `transform_random_grayscale()`, `transform_random_horizontal_flip()`, `transform_random_perspective()`, `transform_random_resized_crop()`, `transform_random_vertical_flip()`

---

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 random_transforms: `transform_color_jitter()`, `transform_random_affine()`, `transform_random_crop()`, `transform_random_erasing()`, `transform_random_grayscale()`, `transform_random_horizontal_flip()`, `transform_random_perspective()`, `transform_random_resized_crop()`, `transform_random_rotation()`

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 unitary_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_grayscale()`, `transform_hflip()`, `transform_linear_transformation()`, `transform_normalize()`, `transform_pad()`, `transform_perspective()`, `transform_rgb_to_grayscale()`, `transform_rotate()`, `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 combining_transforms: transform_five_crop(), transform_random_apply(), transform_random_choice(), transform_random_order(), transform_ten_crop()

---

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 unitary_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_grayscale(), transform_hflip(), transform_linear_transformation(), transform_normalize(), transform_pad(), transform_perspective(), transform_resize(), transform_rotate(), transform_to_tensor(), transform_vflip()

`transform_rotate` *Angular rotation of an image*

Description

Angular rotation of an image

Usage

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

Arguments

`img`	A `magick-image`, `array` or `torch_tensor`.
`angle`	(float or int): rotation angle value in degrees, counter-clockwise.
`interpolation`	(int, optional): Interpolation mode. 0 for nearest, 2 for bilinear. Default is 0 (nearest).
`expand`	(bool, optional): Optional expansion flag. If true, expands the output to make it large enough to hold the entire rotated image. If false or omitted, make the output image the same size as the input image. Note that the expand flag assumes rotation around the center and no translation.
`center`	(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.
`resample`	Deprecated. Use interpolation instead.

See Also

Other unitary_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_grayscale()`, `transform_hflip()`, `transform_linear_transformation()`, `transform_normalize()`, `transform_pad()`, `transform_perspective()`, `transform_resize()`, `transform_rgb_to_grayscale()`, `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 combining_transforms: transform_five_crop(), transform_random_apply(), transform_random_choice(), transform_random_order(), transform_resized_crop()

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 unitary_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_grayscale()`, `transform_hflip()`, `transform_linear_transformation()`, `transform_normalize()`, `transform_pad()`, `transform_perspective()`, `transform_resize()`, `transform_rgb_to_grayscale()`, `transform_rotate()`, `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 unitary_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_grayscale()`, `transform_hflip()`, `transform_linear_transformation()`, `transform_normalize()`, `transform_pad()`, `transform_perspective()`, `transform_resize()`, `transform_rgb_to_grayscale()`, `transform_rotate()`, `transform_to_tensor()`

vggface2_dataset *VGGFace2 Dataset*

Description

The VGGFace2 dataset is a large-scale face recognition dataset containing images of celebrities from a wide range of ethnicities, professions, and ages. Each identity has multiple images with variations in context, pose, age, and illumination.

Usage

```
vggface2_dataset(
  root = tempdir(),
  split = "val",
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)
```

Arguments

root	Character. Root directory where the dataset will be stored under root/vggface2.
split	One of "train", "val", or "test". Default is "val".
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 vggface2_dataset:

- x: RGB image array.
- y: Integer label (1...N) for the identity.

ds$classes is a named list mapping integer labels to a list with:

- name: Character name of the person.
- gender: "Male" or "Female".

See Also

Other classification_dataset: caltech_dataset, cifar10_dataset(), eurosat_dataset(), fer_dataset(), fgvc_aircraft_dataset(), flowers102_dataset(), image_folder_dataset(), lfw_dataset, mnist_dataset(), oxfordiiitpet_dataset(), places365_dataset(), tiny_imagenet_dataset(), whoi_plankton_dataset(), whoi_small_coralnet_dataset()

Examples

```
## Not run:
#Load the training set
ds <- vggface2_dataset(download = TRUE)
item <- ds[1]
item$x      # image array RGB
item$y      # integer label
ds$classes[item$y]  # list(name=..., gender=...)

#Load the test set
ds <- vggface2_dataset(download = TRUE, train = FALSE)
item <- ds[1]
item$x      # image array RGB
item$y      # integer label
ds$classes[item$y]  # list(name=..., gender=...)

## End(Not run)
```

`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()`

`whoi_plankton_dataset` *WHOI Plankton Datasets*

Description

WHOI-Plankton Dataset

Usage

```
whoi_small_plankton_dataset(
  split = "val",
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)

whoi_plankton_dataset(
  split = "val",
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)
```

Arguments

`split`	One of `"train"`, `"val"`, or `"test"`. Default is `"val"`.
`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 WHOI-Plankton and WHOI-Plankton small are **image classification** datasets from the Woods Hole Oceanographic Institution (WHOI) of microscopic marine plankton. https://hdl.handle.net/10.1575/1912/7341 Images were collected in situ by automated submersible imaging-in-flow cytometry with an instrument called Imaging FlowCytobot (IFCB). They are small grayscale images of varying size. Images are classified into 100 classes, with an overview available in project Wiki page Dataset size is 957k and 58k respectively, and each provides a train / val / test split.

Value

A torch dataset with a

- `classes` attribute providing the vector of class names.

Each element is a named list:

- x: a H x W x 1 integer array representing an grayscale image.
- y: the class id of the image.

See Also

Other classification_dataset: `caltech_dataset`, `cifar10_dataset()`, `eurosat_dataset()`, `fer_dataset()`, `fgvc_aircraft_dataset()`, `flowers102_dataset()`, `image_folder_dataset()`, `lfw_dataset`, `mnist_dataset()`, `oxfordiiitpet_dataset()`, `places365_dataset()`, `tiny_imagenet_dataset()`, `vggface2_dataset()`, `whoi_small_coralnet_dataset()`

Examples

```
## Not run:
# Load the small plankton dataset and turn images into tensor images
plankton <- whoi_small_plankton_dataset(download = TRUE, transform = transform_to_tensor)

# Access the first item
first_item <- plankton[1]
first_item$x  # a tensor grayscale image with shape {1, H, W}
first_item$y  # id of the plankton class.
plankton$classes[first_item$y] # name of the plankton class

# Load the full plankton dataset
plankton <- whoi_plankton_dataset(download = TRUE)

# Access the first item
first_item <- plankton[1]
first_item$x  # grayscale image array with shape {H, W}
first_item$y  # id of the plankton class.

## End(Not run)
```

`whoi_small_coralnet_dataset` *Coralnet Dataset*

Description

Small Coralnet dataset is an image **classification dataset** of very large submarine coral reef images annotated into 3 classes and produced by CoralNet, a resource for benthic images classification.

Usage

```
whoi_small_coralnet_dataset(
  split = "val",
  transform = NULL,
  target_transform = NULL,
  download = FALSE
)
```

Arguments

`split`	One of `"train"`, `"val"`, or `"test"`. Default is `"val"`.
`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.

See Also

Other classification_dataset: `caltech_dataset`, `cifar10_dataset()`, `eurosat_dataset()`, `fer_dataset()`, `fgvc_aircraft_dataset()`, `flowers102_dataset()`, `image_folder_dataset()`, `lfw_dataset`, `mnist_dataset()`, `oxfordiiitpet_dataset()`, `places365_dataset()`, `tiny_imagenet_dataset()`, `vggface2_dataset()`, `whoi_plankton_dataset()`

Index