Image data loading

`image_dataset_from_directory` function

[source]

```
keras.utils.image_dataset_from_directory(
    directory,
    labels="inferred",
    label_mode="int",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(256, 256),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False,
    pad_to_aspect_ratio=False,
    data_format=None,
    verbose=True,
)
```

Generates a `tf.data.Dataset` from image files in a directory.

If your directory structure is:

```
main_directory/
...class_a/
......a_image_1.jpg
......a_image_2.jpg
...class_b/
......b_image_1.jpg
......b_image_2.jpg
```

Then calling `image_dataset_from_directory(main_directory, labels='inferred')` will return a `tf.data.Dataset` that yields batches of images from the subdirectories `class_a` and `class_b`, together with labels 0 and 1 (0 corresponding to `class_a` and 1 corresponding to `class_b`).

Supported image formats: `.jpeg`, `.jpg`, `.png`, `.bmp`, `.gif`. Animated gifs are truncated to the first frame.

Arguments

- **directory**: Directory where the data is located. If `labels` is `"inferred"`, it should contain subdirectories, each containing images for a class. Otherwise, the directory structure is ignored.
- **labels**: Either `"inferred"` (labels are generated from the directory structure), `None` (no labels), or a list/tuple of integer labels of the same size as the number of image files found in the directory. Labels should be sorted according to the alphanumeric order of the image file paths (obtained via `os.walk(directory)` in Python).
- **label_mode**: String describing the encoding of `labels`. Options are:
  - `"int"`: means that the labels are encoded as integers (e.g. for `sparse_categorical_crossentropy` loss).
  - `"categorical"` means that the labels are encoded as a categorical vector (e.g. for `categorical_crossentropy` loss).
  - `"binary"` means that the labels (there can be only 2) are encoded as `float32` scalars with values 0 or 1 (e.g. for `binary_crossentropy`).
  - `None` (no labels).
- **class_names**: Only valid if `labels` is `"inferred"`. This is the explicit list of class names (must match names of subdirectories). Used to control the order of the classes (otherwise alphanumerical order is used).
- **color_mode**: One of `"grayscale"`, `"rgb"`, `"rgba"`. Defaults to `"rgb"`. Whether the images will be converted to have 1, 3, or 4 channels.

- **batch_size**: Size of the batches of data. Defaults to 32. If `None`, the data will not be batched (the dataset will yield individual samples).
- **image_size**: Size to resize images to after they are read from disk, specified as `(height, width)`. Defaults to `(256, 256)`. Since the pipeline processes batches of images that must all have the same size, this must be provided.
- **shuffle**: Whether to shuffle the data. Defaults to `True`. If set to `False`, sorts the data in alphanumeric order.
- **seed**: Optional random seed for shuffling and transformations.
- **validation_split**: Optional float between 0 and 1, fraction of data to reserve for validation.
- **subset**: Subset of the data to return. One of `"training"`, `"validation"`, or `"both"`. Only used if `validation_split` is set. When `subset="both"`, the utility returns a tuple of two datasets (the training and validation datasets respectively).
- **interpolation**: String, the interpolation method used when resizing images. Defaults to `"bilinear"`. Supports `"bilinear"`, `"nearest"`, `"bicubic"`, `"area"`, `"lanczos3"`, `"lanczos5"`, `"gaussian"`, `"mitchellcubic"`.
- **follow_links**: Whether to visit subdirectories pointed to by symlinks. Defaults to `False`.
- **crop_to_aspect_ratio**: If `True`, resize the images without aspect ratio distortion. When the original aspect ratio differs from the target aspect ratio, the output image will be cropped so as to return the largest possible window in the image (of size `image_size`) that matches the target aspect ratio. By default (`crop_to_aspect_ratio=False`), aspect ratio may not be preserved.
- **pad_to_aspect_ratio**: If `True`, resize the images without aspect ratio distortion. When the original aspect ratio differs from the target aspect ratio, the output image will be padded so as to return the largest possible window in the image (of size `image_size`) that matches the target aspect ratio. By default (`pad_to_aspect_ratio=False`), aspect ratio may not be preserved.
- **data_format**: If None uses keras.config.image_data_format() otherwise either 'channel_last' or 'channel_first'.
- **verbose**: Whether to display number information on classes and number of files found. Defaults to `True`.

Returns

A `tf.data.Dataset` object.

- If `label_mode` is `None`, it yields `float32` tensors of shape `(batch_size, image_size[0], image_size[1], num_channels)`, encoding images (see below for rules regarding `num_channels`).
- Otherwise, it yields a tuple `(images, labels)`, where `images` has shape `(batch_size, image_size[0], image_size[1], num_channels)`, and `labels` follows the format described below.

Rules regarding labels format:

- if `label_mode` is `"int"`, the labels are an `int32` tensor of shape `(batch_size,)`.
- if `label_mode` is `"binary"`, the labels are a `float32` tensor of 1s and 0s of shape `(batch_size, 1)`.
- if `label_mode` is `"categorical"`, the labels are a `float32` tensor of shape `(batch_size, num_classes)`, representing a one-hot encoding of the class index.

Rules regarding number of channels in the yielded images:

- if `color_mode` is `"grayscale"`, there's 1 channel in the image tensors.
- if `color_mode` is `"rgb"`, there are 3 channels in the image tensors.
- if `color_mode` is `"rgba"`, there are 4 channels in the image tensors.

`load_img` function [source]

```
keras.utils.load_img(
    path,
    color_mode="rgb",
    target_size=None,
    interpolation="nearest",
    keep_aspect_ratio=False,
)
```

Loads an image into PIL format.

Example

```
image = keras.utils.load_img(image_path)
input_arr = keras.utils.img_to_array(image)
input_arr = np.array([input_arr])  # Convert single image to a batch.
predictions = model.predict(input_arr)
```

Arguments

- **path**: Path to image file.
- **color_mode**: One of `"grayscale"`, `"rgb"`, `"rgba"`. Default: `"rgb"`. The desired image format.
- **target_size**: Either `None` (default to original size) or tuple of ints `(img_height, img_width)`.
- **interpolation**: Interpolation method used to resample the image if the target size is different from that of the loaded image. Supported methods are `"nearest"`, `"bilinear"`, and `"bicubic"`. If PIL version 1.1.3 or newer is installed, `"lanczos"` is also supported. If PIL version 3.4.0 or newer is installed, `"box"` and `"hamming"` are also supported. By default, `"nearest"` is used.
- **keep_aspect_ratio**: Boolean, whether to resize images to a target size without aspect ratio distortion. The image is cropped in the center with target aspect ratio before resizing.

Returns

A PIL Image instance.

`img_to_array` function [source]

```
keras.utils.img_to_array(img, data_format=None, dtype=None)
```

Converts a PIL Image instance to a NumPy array.

Example

```
from PIL import Image
img_data = np.random.random(size=(100, 100, 3))
img = keras.utils.array_to_img(img_data)
array = keras.utils.image.img_to_array(img)
```

Arguments

- **img**: Input PIL Image instance.
- **data_format**: Image data format, can be either `"channels_first"` or `"channels_last"`. Defaults to `None`, in which case the global setting `keras.backend.image_data_format()` is used (unless you changed it, it defaults to `"channels_last"`).
- **dtype**: Dtype to use. `None` means the global setting `keras.backend.floatx()` is used (unless you changed it, it defaults to `"float32"`).

Returns

A 3D NumPy array.

`save_img` function [source]

```
keras.utils.save_img(
    path, x, data_format=None, file_format=None, scale=True, **kwargs
)
```

Saves an image stored as a NumPy array to a path or file object.

Arguments

- **path**: Path or file object.
- **x**: NumPy array.
- **data_format**: Image data format, either `"channels_first"` or `"channels_last"`.
- **file_format**: Optional file format override. If omitted, the format to use is determined from the filename extension. If a file object was used instead of a filename, this parameter should always be used.
- **scale**: Whether to rescale image values to be within `[0, 255]`.
- ****kwargs**: Additional keyword arguments passed to `PIL.Image.save()`.

`array_to_img` function

[source]

```
keras.utils.array_to_img(x, data_format=None, scale=True, dtype=None)
```

Converts a 3D NumPy array to a PIL Image instance.

Example

```
from PIL import Image
img = np.random.random(size=(100, 100, 3))
pil_img = keras.utils.array_to_img(img)
```

Arguments

- **x**: Input data, in any form that can be converted to a NumPy array.
- **data_format**: Image data format, can be either `"channels_first"` or `"channels_last"`. Defaults to `None`, in which case the global setting `keras.backend.image_data_format()` is used (unless you changed it, it defaults to `"channels_last"`).
- **scale**: Whether to rescale the image such that minimum and maximum values are 0 and 255 respectively. Defaults to `True`.
- **dtype**: Dtype to use. `None` means the global setting `keras.backend.floatx()` is used (unless you changed it, it defaults to `"float32"`). Defaults to `None`.

Returns

A PIL Image instance.