**IMPORTING THE IMAGE DATA GENERATOR**

**LIBRARY**

# **Dataset preprocessing**

Keras dataset preprocessing utilities, located at [tf.keras.preprocessing](https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing), help you go from raw data on disk to a [tf.data.Dataset](https://www.tensorflow.org/api_docs/python/tf/data/Dataset) object that can be used to train a model.

Here's a quick example: let's say you have 10 folders, each containing 10,000 images from a different category, and you want to train a classifier that maps an image to its category. Your training data folder would look like this:

training\_data/

...class\_a/

......a\_image\_1.jpg

......a\_image\_2.jpg

...class\_b/

......b\_image\_1.jpg

......b\_image\_2.jpg

etc.

You may also have a validation data folder validation\_data/ structured in the same way.

You could simply do:

from tensorflow import keras

from tensorflow.keras.preprocessing import image\_dataset\_from\_directory

train\_ds = image\_dataset\_from\_directory(

directory='training\_data/',

labels='inferred',

label\_mode='categorical',

batch\_size=32,

image\_size=(256, 256))

validation\_ds = image\_dataset\_from\_directory(

directory='validation\_data/',

labels='inferred',

label\_mode='categorical',

batch\_size=32,

image\_size=(256, 256))

model = keras.applications.Xception(weights=None, input\_shape=(256, 256, 3), classes=10)

model.compile(optimizer='rmsprop', loss='categorical\_crossentropy')

model.fit(train\_ds, epochs=10, validation\_data=validation\_ds)

## Available dataset preprocessing utilities

### [Image data preprocessing](https://keras.io/api/preprocessing/image)

* [image\_dataset\_from\_directory function](https://keras.io/api/preprocessing/image/#image_dataset_from_directory-function)
* [load\_img function](https://keras.io/api/preprocessing/image/#load_img-function)
* [img\_to\_array function](https://keras.io/api/preprocessing/image/#img_to_array-function)

### [Timeseries data preprocessing](https://keras.io/api/preprocessing/timeseries)

* [timeseries\_dataset\_from\_array function](https://keras.io/api/preprocessing/timeseries/#timeseries_dataset_from_array-function)
* [pad\_sequences function](https://keras.io/api/preprocessing/timeseries/#pad_sequences-function)

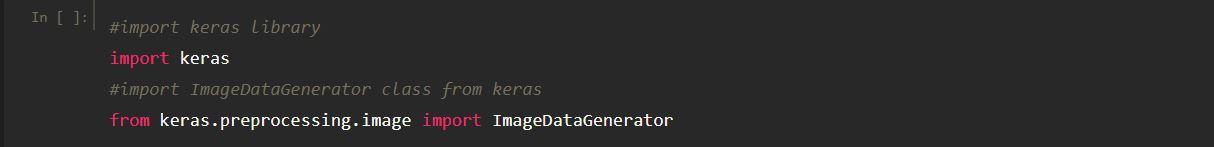
### [Text data preprocessing](https://keras.io/api/preprocessing/text)

* [text\_dataset\_from\_directory function](https://keras.io/api/preprocessing/text/#text_dataset_from_directory-function)

**IMAGE DATA GENERATOR LIBRARY**

The first step is usually importing the libraries that will be needed in the program.

Import Keras library from that library import the ImageDataGenerator Library to your Python script:



# Image data preprocessing

### image\_dataset\_from\_directory function

tf.keras.preprocessing.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,

\*\*kwargs

)

Generates a [tf.data.Dataset](https://www.tensorflow.org/api_docs/python/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](https://www.tensorflow.org/api_docs/python/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, 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**: - '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 explict 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". Default: "rgb". Whether the images will be converted to have 1, 3, or 4 channels.
* **batch\_size**: Size of the batches of data. Default: 32.
* **image\_size**: Size to resize images to after they are read from disk. 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. Default: 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**: One of "training" or "validation". Only used if validation\_split is set.
* **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 visits 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.
* **\*\*kwargs**: Legacy keyword arguments.

**Returns**

A [tf.data.Dataset](https://www.tensorflow.org/api_docs/python/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 categorial, 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 channel in the image tensors. - if color\_mode is rgba, there are 4 channel in the image tensors.

**load\_img function**

tf.keras.preprocessing.image.load\_img(

path, grayscale=False, color\_mode="rgb", target\_size=None, interpolation="nearest"

)

Loads an image into PIL format.

**Usage:**

image = tf.keras.preprocessing.image.load\_img(image\_path)

input\_arr = tf.keras.preprocessing.image.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.
* **grayscale**: DEPRECATED use color\_mode="grayscale".
* **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.

**Returns**

A PIL Image instance.

**Raises**

* **ImportError**: if PIL is not available.
* **ValueError**: if interpolation method is not supported.

### img\_to\_array function

tf.keras.preprocessing.image.img\_to\_array(img, data\_format=None, dtype=None)

Converts a PIL Image instance to a Numpy array.

**Usage:**

from PIL import Image

img\_data = np.random.random(size=(100, 100, 3))

img = tf.keras.preprocessing.image.array\_to\_img(img\_data)

array = tf.keras.preprocessing.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 tf.keras.backend.image\_data\_format() is used (unless you changed it, it defaults to "channels\_last").
* **dtype**: Dtype to use. Default to None, in which case the global setting tf.keras.backend.floatx() is used (unless you changed it, it defaults to "float32")

**Returns**

A 3D Numpy array.

**Raises**

* **ValueError**: if invalid img or data\_format is passed.