# 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,
    smart_resize=False,
)
```

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, 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.
- **smart_resize**: If True, the resizing function used will be `tf.keras.preprocessing.image.smart_resize`, which preserves the aspect ratio of the original image by using a mixture of resizing and cropping. If False (default), the resizing function is `tf.image.resize`, which does not preserve aspect ratio.

**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 `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 = 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.

---

## `ImageDataGenerator` class

```
tf.keras.preprocessing.image.ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    zca_epsilon=1e-06,
    rotation_range=0,
    width_shift_range=0.0,
    height_shift_range=0.0,
    brightness_range=None,
    shear_range=0.0,
    zoom_range=0.0,
    channel_shift_range=0.0,
    fill_mode="nearest",
    cval=0.0,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=None,
    preprocessing_function=None,
    data_format=None,
    validation_split=0.0,
    dtype=None,
)
```

Generate batches of tensor image data with real-time data augmentation.

The data will be looped over (in batches).

**Arguments**

- **featurewise_center**: Boolean. Set input mean to 0 over the dataset, feature-wise.
- **samplewise_center**: Boolean. Set each sample mean to 0.
- **featurewise_std_normalization**: Boolean. Divide inputs by std of the dataset, feature-wise.
- **samplewise_std_normalization**: Boolean. Divide each input by its std.
- **zca_epsilon**: epsilon for ZCA whitening. Default is 1e-6.
- **zca_whitening**: Boolean. Apply ZCA whitening.
- **rotation_range**: Int. Degree range for random rotations.
- **width_shift_range**: Float, 1-D array-like or int - float: fraction of total width, if < 1, or pixels if >= 1. - 1-D array-like: random elements from the array. - int: integer number of pixels from interval `(-width_shift_range, +width_shift_range)` - With `width_shift_range=2` possible values are integers `[-1, 0, +1]`, same as with `width_shift_range=[-1, 0, +1]`, while with `width_shift_range=1.0` possible values are floats in the interval [-1.0, +1.0).
- **height_shift_range**: Float, 1-D array-like or int - float: fraction of total height, if < 1, or pixels if >= 1. - 1-D array-like: random elements from the array. - int: integer number of pixels from interval `(-height_shift_range, +height_shift_range)` - With `height_shift_range=2` possible values are integers `[-1, 0, +1]`, same as with `height_shift_range=[-1, 0, +1]`, while with `height_shift_range=1.0` possible values are floats in the interval [-1.0, +1.0).
- **brightness_range**: Tuple or list of two floats. Range for picking a brightness shift value from.
- **shear_range**: Float. Shear Intensity (Shear angle in counter-clockwise direction in degrees)

- **zoom_range**: Float or [lower, upper]. Range for random zoom. If a float, `[lower, upper] = [1-zoom_range, 1+zoom_range]`.
- **channel_shift_range**: Float. Range for random channel shifts.
- **fill_mode**: One of {"constant", "nearest", "reflect" or "wrap"}. Default is 'nearest'. Points outside the boundaries of the input are filled according to the given mode: - 'constant': kkkkkkkk|abcd|kkkkkkkk (cval=k) - 'nearest': aaaaaaaa|abcd|dddddddd - 'reflect': abcddcba|abcd|dcbaabcd - 'wrap': abcdabcd|abcd|abcdabcd
- **cval**: Float or Int. Value used for points outside the boundaries when `fill_mode = "constant"`.
- **horizontal_flip**: Boolean. Randomly flip inputs horizontally.
- **vertical_flip**: Boolean. Randomly flip inputs vertically.
- **rescale**: rescaling factor. Defaults to None. If None or 0, no rescaling is applied, otherwise we multiply the data by the value provided (after applying all other transformations).
- **preprocessing_function**: function that will be applied on each input. The function will run after the image is resized and augmented. The function should take one argument: one image (Numpy tensor with rank 3), and should output a Numpy tensor with the same shape.
- **data_format**: Image data format, either "channels_first" or "channels_last". "channels_last" mode means that the images should have shape `(samples, height, width, channels)`, "channels_first" mode means that the images should have shape `(samples, channels, height, width)`. It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels_last".
- **validation_split**: Float. Fraction of images reserved for validation (strictly between 0 and 1).
- **dtype**: Dtype to use for the generated arrays.

**Raises**

- **ValueError**: If the value of the argument, `data_format` is other than `"channels_last"` or `"channels_first"`.
- **ValueError**: If the value of the argument, `validation_split` > 1 or `validation_split` < 0.

**Examples**

Example of using `.flow(x, y)`:

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
y_train = utils.to_categorical(y_train, num_classes)
y_test = utils.to_categorical(y_test, num_classes)
datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    validation_split=0.2)
# compute quantities required for featurewise normalization
# (std, mean, and principal components if ZCA whitening is applied)
datagen.fit(x_train)
# fits the model on batches with real-time data augmentation:
model.fit(datagen.flow(x_train, y_train, batch_size=32,
         subset='training'),
         validation_data=datagen.flow(x_train, y_train,
         batch_size=8, subset='validation'),
         steps_per_epoch=len(x_train) / 32, epochs=epochs)
# here's a more "manual" example
for e in range(epochs):
    print('Epoch', e)
    batches = 0
    for x_batch, y_batch in datagen.flow(x_train, y_train, batch_size=32):
        model.fit(x_batch, y_batch)
        batches += 1
        if batches >= len(x_train) / 32:
            # we need to break the loop by hand because
            # the generator loops indefinitely
            break
```

Example of using `.flow_from_directory(directory)`:

```
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
        'data/train',
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')
validation_generator = test_datagen.flow_from_directory(
        'data/validation',
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')
model.fit(
        train_generator,
        steps_per_epoch=2000,
        epochs=50,
        validation_data=validation_generator,
        validation_steps=800)
```

Example of transforming images and masks together.

```
# we create two instances with the same arguments
data_gen_args = dict(featurewise_center=True,
                     featurewise_std_normalization=True,
                     rotation_range=90,
                     width_shift_range=0.1,
                     height_shift_range=0.1,
                     zoom_range=0.2)
image_datagen = ImageDataGenerator(**data_gen_args)
mask_datagen = ImageDataGenerator(**data_gen_args)
# Provide the same seed and keyword arguments to the fit and flow methods
seed = 1
image_datagen.fit(images, augment=True, seed=seed)
mask_datagen.fit(masks, augment=True, seed=seed)
image_generator = image_datagen.flow_from_directory(
    'data/images',
    class_mode=None,
    seed=seed)
mask_generator = mask_datagen.flow_from_directory(
    'data/masks',
    class_mode=None,
    seed=seed)
# combine generators into one which yields image and masks
train_generator = zip(image_generator, mask_generator)
model.fit(
    train_generator,
    steps_per_epoch=2000,
    epochs=50)
```

---

## `flow` method

```
ImageDataGenerator.flow(
    x,
    y=None,
    batch_size=32,
    shuffle=True,
    sample_weight=None,
    seed=None,
    save_to_dir=None,
    save_prefix="",
    save_format="png",
    subset=None,
)
```

Takes data & label arrays, generates batches of augmented data.

**Arguments**

- **x**: Input data. Numpy array of rank 4 or a tuple. If tuple, the first element should contain the images and the second element another numpy array or a list of numpy arrays that gets passed to the output without any modifications. Can be used to feed the model miscellaneous data

along with the images. In case of grayscale data, the channels axis of the image array should have value 1, in case of RGB data, it should have value 3, and in case of RGBA data, it should have value 4.
- **y**: Labels.
- **batch_size**: Int (default: 32).
- **shuffle**: Boolean (default: True).
- **sample_weight**: Sample weights.
- **seed**: Int (default: None).
- **save_to_dir**: None or str (default: None). This allows you to optionally specify a directory to which to save the augmented pictures being generated (useful for visualizing what you are doing).
- **save_prefix**: Str (default: `''`). Prefix to use for filenames of saved pictures (only relevant if `save_to_dir` is set).
- **save_format**: one of "png", "jpeg", "bmp", "pdf", "ppm", "gif", "tif", "jpg" (only relevant if `save_to_dir` is set). Default: "png".
- **subset**: Subset of data (`"training"` or `"validation"`) if `validation_split` is set in `ImageDataGenerator`.

**Returns**

An `Iterator` yielding tuples of `(x, y)` where `x` is a numpy array of image data (in the case of a single image input) or a list of numpy arrays (in the case with additional inputs) and `y` is a numpy array of corresponding labels. If 'sample_weight' is not None, the yielded tuples are of the form `(x, y, sample_weight)`. If `y` is None, only the numpy array `x` is returned.

**Raises**

- **ValueError**: If the Value of the argument, `subset` is other than "training" or "validation".

---

## `flow_from_dataframe` method

```
ImageDataGenerator.flow_from_dataframe(
    dataframe,
    directory=None,
    x_col="filename",
    y_col="class",
    weight_col=None,
    target_size=(256, 256),
    color_mode="rgb",
    classes=None,
    class_mode="categorical",
    batch_size=32,
    shuffle=True,
    seed=None,
    save_to_dir=None,
    save_prefix="",
    save_format="png",
    subset=None,
    interpolation="nearest",
    validate_filenames=True,
    **kwargs
)
```

Takes the dataframe and the path to a directory + generates batches.

The generated batches contain augmented/normalized data.

**A simple tutorial can be found** here.

**Arguments**

- **dataframe**: Pandas dataframe containing the filepaths relative to `directory` (or absolute paths if `directory` is None) of the images in a string column. It should include other column/s depending on the `class_mode`: - if `class_mode` is `"categorical"` (default value) it must include the `y_col` column with the class/es of each image. Values in column can be string/list/tuple if a single class or list/tuple if multiple classes. - if `class_mode` is `"binary"` or `"sparse"` it must include the given `y_col` column with class values as strings. - if `class_mode` is `"raw"` or `"multi_output"` it should contain the columns specified in `y_col`. - if `class_mode` is `"input"` or `None` no extra column is needed.
- **directory**: string, path to the directory to read images from. If `None`, data in `x_col` column should be absolute paths.
- **x_col**: string, column in `dataframe` that contains the filenames (or absolute paths if `directory` is `None`).
- **y_col**: string or list, column/s in `dataframe` that has the target data.
- **weight_col**: string, column in `dataframe` that contains the sample weights. Default: `None`.

- **target_size**: tuple of integers `(height, width)`, default: `(256, 256)`. The dimensions to which all images found will be resized.
- **color_mode**: one of "grayscale", "rgb", "rgba". Default: "rgb". Whether the images will be converted to have 1 or 3 color channels.
- **classes**: optional list of classes (e.g. `['dogs', 'cats']`). Default is None. If not provided, the list of classes will be automatically inferred from the `y_col`, which will map to the label indices, will be alphanumeric). The dictionary containing the mapping from class names to class indices can be obtained via the attribute `class_indices`.
- **class_mode**: one of "binary", "categorical", "input", "multi_output", "raw", sparse" or None. Default: "categorical". Mode for yielding the targets: - `"binary"`: 1D numpy array of binary labels, - `"categorical"`: 2D numpy array of one-hot encoded labels. Supports multi-label output. - `"input"`: images identical to input images (mainly used to work with autoencoders), - `"multi_output"`: list with the values of the different columns, - `"raw"`: numpy array of values in `y_col` column(s), - `"sparse"`: 1D numpy array of integer labels, - `None`, no targets are returned (the generator will only yield batches of image data, which is useful to use in `model.predict()`).
- **batch_size**: size of the batches of data (default: 32).
- **shuffle**: whether to shuffle the data (default: True)
- **seed**: optional random seed for shuffling and transformations.
- **save_to_dir**: None or str (default: None). This allows you to optionally specify a directory to which to save the augmented pictures being generated (useful for visualizing what you are doing).
- **save_prefix**: str. Prefix to use for filenames of saved pictures (only relevant if `save_to_dir` is set).
- **save_format**: one of "png", "jpeg", "bmp", "pdf", "ppm", "gif", "tif", "jpg" (only relevant if `save_to_dir` is set). Default: "png".
- **subset**: Subset of data (`"training"` or `"validation"`) if `validation_split` is set in `ImageDataGenerator`.
- **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.
- **validate_filenames**: Boolean, whether to validate image filenames in `x_col`. If `True`, invalid images will be ignored. Disabling this option can lead to speed-up in the execution of this function. Defaults to `True`.
- ****kwargs**: legacy arguments for raising deprecation warnings.

**Returns**

A `DataFrameIterator` yielding tuples of `(x, y)` where `x` is a numpy array containing a batch of images with shape `(batch_size, *target_size, channels)` and `y` is a numpy array of corresponding labels.

---

## `flow_from_directory` method

```
ImageDataGenerator.flow_from_directory(
    directory,
    target_size=(256, 256),
    color_mode="rgb",
    classes=None,
    class_mode="categorical",
    batch_size=32,
    shuffle=True,
    seed=None,
    save_to_dir=None,
    save_prefix="",
    save_format="png",
    follow_links=False,
    subset=None,
    interpolation="nearest",
)
```

Takes the path to a directory & generates batches of augmented data.

**Arguments**

- **directory**: string, path to the target directory. It should contain one subdirectory per class. Any PNG, JPG, BMP, PPM or TIF images inside each of the subdirectories directory tree will be included in the generator. See this script for more details.
- **target_size**: Tuple of integers `(height, width)`, defaults to `(256, 256)`. The dimensions to which all images found will be resized.
- **color_mode**: One of "grayscale", "rgb", "rgba". Default: "rgb". Whether the images will be converted to have 1, 3, or 4 channels.
- **classes**: Optional list of class subdirectories (e.g. `['dogs', 'cats']`). Default: None. If not provided, the list of classes will be automatically inferred from the subdirectory

names/structure under `directory`, where each subdirectory will be treated as a different class (and the order of the classes, which will map to the label indices, will be alphanumeric). The dictionary containing the mapping from class names to class indices can be obtained via the attribute `class_indices`.

- **class_mode**: One of "categorical", "binary", "sparse", "input", or None. Default: "categorical". Determines the type of label arrays that are returned: - "categorical" will be 2D one-hot encoded labels, - "binary" will be 1D binary labels, - "sparse" will be 1D integer labels, - "input" will be images identical to input images (mainly used to work with autoencoders). - If None, no labels are returned (the generator will only yield batches of image data, which is useful to use with `model.predict()`). Please note that in case of class_mode None, the data still needs to reside in a subdirectory of `directory` for it to work correctly.
- **batch_size**: Size of the batches of data (default: 32).
- **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.
- **save_to_dir**: None or str (default: None). This allows you to optionally specify a directory to which to save the augmented pictures being generated (useful for visualizing what you are doing).
- **save_prefix**: Str. Prefix to use for filenames of saved pictures (only relevant if `save_to_dir` is set).
- **save_format**: one of "png", "jpeg", "bmp", "pdf", "ppm", "gif", "tif", "jpg" (only relevant if `save_to_dir` is set). Default: "png".
- **follow_links**: Whether to follow symlinks inside class subdirectories (default: False).
- **subset**: Subset of data (`"training"` or `"validation"`) if `validation_split` is set in `ImageDataGenerator`.
- **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 `DirectoryIterator` yielding tuples of `(x, y)` where `x` is a numpy array containing a batch of images with shape `(batch_size, *target_size, channels)` and `y` is a numpy array of corresponding labels.

---