Image Preprocessing

ImageDataGenerator class

[source]

```
keras.preprocessing.image.ImageDataGenerator(featurewise_center=False, samplewise_center=Fal
```

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 (fraction of total width). Range for random horizontal shifts.
- **height_shift_range**: Float (fraction of total height). Range for random vertical shifts.
- **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 (before applying any other transformation).
- **preprocessing_function**: function that will be implied 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**: One of {"channels_first", "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).

Examples

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

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
y_train = np_utils.to_categorical(y_train, num_classes)
y_test = np_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)

# 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_generator(datagen.flow(x_train, y_train, batch_size=32),
                    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_generator(
        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_generator(
    train_generator,
    steps_per_epoch=2000,
    epochs=50)
```

ImageDataGenerator methods

fit

```
fit(x, augment=False, rounds=1, seed=None)
```

Compute the internal data stats related to the data-dependent transformations, based on an array of sample data. Only required if featurewise_center or featurewise_std_normalization or zca_whitening.

Arguments

- **x**: sample data. Should have rank 4. In case of grayscale data, the channels axis should have value 1, and in case of RGB data, it should have value 3.
- **augment**: Boolean (default: False). Whether to fit on randomly augmented samples.
- **rounds**: int (default: 1). If augment, how many augmentation passes over the data to use.
- **seed**: int (default: None). Random seed.

---

flow

```
flow(x, y=None, batch_size=32, shuffle=True, seed=None, save_to_dir=None, save_prefix='', sa
```

Takes numpy data & label arrays, and generates batches of augmented/normalized data.

Arguments

- **x**: data. Should have rank 4. In case of grayscale data, the channels axis should have value 1, and in case of RGB data, it should have value 3.
- **y**: labels.
- **batch_size**: int (default: 32).
- **shuffle**: boolean (default: True).
- **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" (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 and `y` is a numpy array of corresponding labels.

---

flow_from_directory

```
flow_from_directory(directory, target_size=(256, 256), color_mode='rgb', classes=None, class
```

Takes the path to a directory, and generates batches of augmented/normalized data.

Arguments

- **directory**: 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)`, default: `(256, 256)`. The dimensions to which all images found will be resized.
- **color_mode**: one of "grayscale", "rbg". Default: "rgb". Whether the images will be converted to have 1 or 3 color 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 `model.predict_generator()`, `model.evaluate_generator()`, etc.). 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)
- **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" (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.

---

random_transform

```
random_transform(x, seed=None)
```

Randomly augment a single image tensor.

Arguments

- **x**: 3D tensor, single image.
- **seed**: random seed.

Returns

A randomly transformed version of the input (same shape).

---

standardize

```
standardize(x)
```

Apply the normalization configuration to a batch of inputs.

Arguments

- **x**: batch of inputs to be normalized.

Returns

The inputs, normalized.