ImageDataGenerator

```
keras.preprocessing.image.ImageDataGenerator(featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    rotation_range=0.,
    width_shift_range=0.,
    height_shift_range=0.,
    shear_range=0.,
    zoom_range=0.,
    channel_shift_range=0.,
    fill_mode='nearest',
    cval=0.,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=None,
    dim_ordering=K.image_dim_ordering())
```

Generate batches of tensor image data with real-time data augmentation. The data will be looped over (in batches) indefinitely.

- **Arguments**:
    - **featurewise_center**: Boolean. Set input mean to 0 over the dataset.
    - **samplewise_center**: Boolean. Set each sample mean to 0.
    - **featurewise_std_normalization**: Boolean. Divide inputs by std of the dataset.
    - **samplewise_std_normalization**: Boolean. Divide each input by its std.
    - **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 as radians)
    - **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"}. Points outside the boundaries of the input are filled according to the given mode.
    - **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).

- **dim_ordering**: One of {"th", "tf"}. "tf" mode means that the images should have shape `(samples, width, height, channels)`, "th" mode means that the images should have shape `(samples, channels, width, height)`. It defaults to the `image_dim_ordering` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "th".

- **Methods**:

  - **fit(X)**: 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.
      - **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.
  - **flow(X, y)**: Takes numpy data & label arrays, and generates batches of augmented/normalized data. Yields batches indefinitely, in an infinite loop.
    - **Arguments**:
      - **X**: data.
      - **y**: labels.
      - **batch_size**: int (default: 32).
      - **shuffle**: boolean (defaut: False).
      - **save_to_dir**: None or str (default: None). This allows you to optimally 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: "jpeg".
    - **_yields**: Tuples of `(x, y)` where `x` is a numpy array of image data and `y` is a numpy array of corresponding labels. The generator loops indefinitely.
  - **flow_from_directory(directory)**: Takes the path to a directory, and generates batches of augmented/normalized data. Yields batches indefinitely, in an infinite loop.
    - **Arguments**:
      - __directory: path to the target directory. It should contain one subdirectory per class, and the subdirectories should contain PNG or JPG images. See this script for more details.
      - **target_size**: tuple of integers, 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 (and the order of the classes, which will map to the label indices, will be alphanumeric).
      - **class_mode**: one of "categorical", "binary", "sparse" 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. 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.).

- **batch_size**: size of the batches of data (default: 32).
- **shuffle**: whether to shuffle the data (default: True)
- **seed**: optional random seed for shuffling.
- **save_to_dir**: None or str (default: None). This allows you to optimally 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: "jpeg".

- **Examples**:

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

```
(X_train, y_train), (X_test, y_test) = cifar10.load_data(test_split=0.1)
Y_train = np_utils.to_categorical(y_train, nb_classes)
Y_test = np_utils.to_categorical(y_test, nb_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),
                    samples_per_epoch=len(X_train), nb_epoch=nb_epoch)

# here's a more "manual" example
for e in range(nb_epoch):
    print 'Epoch', e
    batches = 0
    for X_batch, Y_batch in datagen.flow(X_train, Y_train, batch_size=32):
        loss = model.train(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,
        samples_per_epoch=2000,
        nb_epoch=50,
        validation_data=validation_generator,
        nb_val_samples=800)
```