ImageDataGenerator

ImageDataGeneratorクラス

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

リアルタイムにデータ拡張しながら，テンソル画像データのバッチを生成します．また，このジェネレータは，データを無限にループするので，無限にバッチを生成します．

引数

- **featurewise_center**: 真理値．データセット全体で，入力の平均を0にします．
- **samplewise_center**: 真理値．各サンプルの平均を0にします．
- **featurewise_std_normalization**: 真理値．入力をデータセットの標準偏差で正規化します．
- **samplewise_std_normalization**: 真理値．各入力をその標準偏差で正規化します．
- **zca_epsilon**: ZCA白色化のイプシロン．デフォルトは1e-6．
- **zca_whitening**: 真理値．ZCA白色化を適用します．
- **rotation_range**: 整数．画像をランダムに回転する回転範囲．
- **width_shift_range**: 浮動小数点数（横幅に対する割合）．ランダムに水平シフトする範囲．
- **height_shift_range**: 浮動小数点数（縦幅に対する割合）．ランダムに垂直シフトする範囲．
- **shear_range**: 浮動小数点数．シアー強度（反時計回りのシアー角度）．
- **zoom_range**: 浮動小数点数または[lower，upper]．ランダムにズームする範囲．浮動小数点数が与えられた場合，`[lower, upper] = [1-zoom_range, 1+zoom_range]`です．
- **channel_shift_range**: 浮動小数点数．ランダムにチャンネルをシフトする範囲．
- **fill_mode**: {"constant", "nearest", "reflect", "wrap"}のいずれか．デフォルトは 'nearest'です．指定されたモードに応じて，入力画像の境界周りを埋めます．
  - "constant": `kkkkkkkk|abcd|kkkkkkkk` (`cval=k`)
  - "nearest": `aaaaaaaa|abcd|dddddddd`
  - "reflect": `abcddcba|abcd|dcbaabcd`
  - "wrap": `abcdabcd|abcd|abcdabcd`
- **cval**: 浮動小数点数または整数． `fill_mode = "constant"`のときに境界周辺で利用される値．
- **horizontal_flip**: 真理値．水平方向に入力をランダムに反転します．
- **vertical_flip**: 真理値．垂直方向に入力をランダムに反転します．
- **rescale**: 画素値のリスケーリング係数．デフォルトはNone．Noneか0ならば，適用しない．それ以外であれば，(他の変換を行う前に) 与えられた値をデータに積算する．
- **preprocessing_function**: 各入力に適用される関数です．この関数は他の変更が行われる前に実行されます．この関数は3次元のNumpyテンソルを引数にとり，同じshapeのテンソルを出力するように定義する必要があります．
- **data_format**: {"channels_first", "channels_last"}のどちらか． `"channels_last"`の場合，入力のshapeは`(samples, height, width, channels)`となり，"channels_first"の場合

は `(samples, channels, height, width)` となります．デフォルトはKerasの設定ファイル `~/.keras/keras.json` の `image_data_format` の値です．一度も値を変更していなければ，"channels_last"になります．

- **validation_split**: 浮動小数点数．検証のために予約しておく画像の割合（厳密には0から1の間）です．

例

`.flow(x, y)` の使用例:

```
(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
```

`.flow_from_directory(directory)` の使用例:

```
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)
```

画像とマスクに対して，同時に変更を加える例.

```
# 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メソッド

fit

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

与えられたサンプルデータに基づいて，データに依存する統計量を計算します．
featurewise_center，featurewise_std_normalization，または，zca_whiteningが指定されたときに必要です．

引数

- **x**: サンプルデータ．4次元データである必要があります．グレースケールデータではチャネルを1に，RGBデータではチャネルを3にしてください．
- **augment**: 真理値（デフォルト: False）．ランダムにサンプルを拡張するかどうか．
- **rounds**: 整数（デフォルト: 1）．augumentが与えられたときに，利用するデータに対して何回データ拡張を行うか．
- **seed**: 整数（デフォルト: None）．乱数シード．

---

flow

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

numpyデータとラベルの配列を受け取り，拡張/正規化したデータのバッチを生成します．

引数

- **x**: データ．4次元データである必要があります．グレースケールデータではチャネルを1に，RGBデータではチャネルを3にしてください．
- **y**: ラベル．
- **batch_size**: 整数（デフォルト: 32）．
- **shuffle**: 真理値（デフォルト: True）．
- **save_to_dir**: Noneまたは文字列（デフォルト: None）．生成された拡張画像を保存するディレクトリを指定できます（行ったことの可視化に有用です）．
- **save_prefix**: 文字列（デフォルト `''`）．画像を保存する際にファイル名に付けるプリフィックス（`set_to_dir` に引数が与えられた時のみ有効）．
- **save_format**: "png"または"jpeg"（`set_to_dir` に引数が与えられた時のみ有効）．デフォルトは"png".

戻り値

`x` が画像データのNumpy配列で `y` がそれに対応したラベルのNumpy配列である `(x, y)` から生成されるイテレータです．

---

flow_from_directory

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

ディレクトリへのパスを受け取り，拡張/正規化したデータのバッチを生成します．

引数 - **directory**: ディレクトリへのパス．クラスごとに1つのサブディレクトリを含み，サブディレクトリはPNGかJPGかBMPかPPMかTIF形式の画像を含まなければいけません．詳細はこのスクリプトを見てください． - **target_size**: 整数のタプル `(height, width)`．デフォルトは `(256, 256)`．この値に全画像はリサイズされます． - **color_mode**: "grayscale"か"rbg"の一方．デフォルトは"rgb"．画像を1か3チャンネルの画像に変換するかどうか． - **classes**: クラスサブディレクトリのリスト．（例えば，`['dogs', 'cats']`）．デフォルトはNone．与えられなければ，クラスのリスト自動的に推論されます（そして，ラベルのインデックスと対応づいたクラスの順序はalphanumericになります）．クラス名からクラスインデックスへのマッピングを含む辞書は `class_indeces` 属性を用いて取得できます． - **class_mode**: "categorical"か"binary"か"sparse"か"input"か"None"のいずれか1つ．デフォルトは"categorical"．返すラベルの配列のshapeを決定します："categorical"は2次元のone-hotにエンコード化されたラベル，"binary"は1次元の2値ラベル，"sparse"は1次元の整数ラベル，"input"は入力画像と同じ画像になります（主にオートエンコーダで用いられます）．Noneであれば，ラベルを返しません（ジェネレーターは画像のバッチのみ生成するため，`model.predict_generator()` や `model.evaluate_generator()` などを使う際に有用）．class_modeがNoneの場合，正常に動作させるためには `directory` のサブディレクトリにデータが存在する必要があることに注意してください． - **batch_size**: データのバッチのサイズ（デフォルト: 32）． - **shuffle**: データをシャッフルするかどうか（デフォルト: True）． - **seed**: シャッフルや変換のためのオプションの乱数シード． - **save_to_dir**: Noneまたは文字列（デフォルト: None）．生成された拡張画像を保存するディレクトリを指定できます（行ったことの可視化に有用です）． - **save_prefix**: 文字列．画像を保存する際にファイル名に付けるプリフィックス（ `set_to_dir` に引数が与えられた時のみ有効）． - **save_format**: "png"または"jpeg"（ `set_to_dir` に引数が与えられた時のみ有効）．デフォルトは"png"． - **follow_links**: サブディレクトリ内のシンボリックリンクに従うかどうか．デフォルトはFalse．

戻り値

`x` が画像データのNumpy配列で `y` がそれに対応したラベルのNumpy配列である `(x, y)` から生成されるDirectoryIteratorです．

---

random_transform

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

単一の画像のテンソルをランダムに拡張します．

引数

- **x**: 単一の画像である3次元テンソル.
- **seed**: ランダムシード.

戻り値

入力画像と同じshapeの入力画像をランダムに変換したもの.

---

standardize

```
standardize(x)
```

入力のバッチに対して正規化を適用します.

引数

- **x**: 正規化対象の入力のバッチ.

戻り値

正規化された入力.