医療とAI・ビッグデータ応用

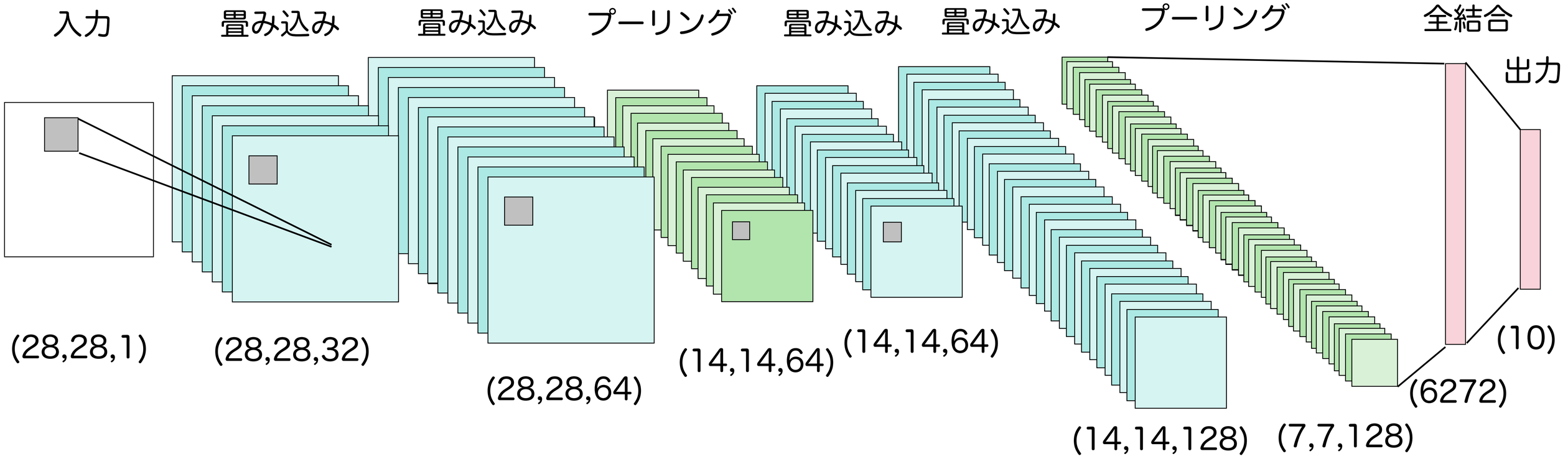
データ拡張

本スライドは、自由にお使いください。
使用した場合は、このQRコードからアンケートに回答をお願いします。

統合教育機構
須藤毅顕

1つ前の復習

畳み込みで細かく画像のパターンを抽出する
プーリングで情報を極力残しつつサイズを小さくする
最後はMLP同様に全結合で10種類の確率を出力する



データ拡張：Data augmentation

データ拡張：学習用のデータを加工して擬似的にデータ量を増やす手法
精度向上や過学習の防止が目的

いいモデルを作ることと同じぐらいデータ量が多いことは重要
データが少ないと正しく学習出来ない(過学習の原因にもなる)
医療データなど、実際には多くのデータが手に入らないことも多い

kerasにはデータ拡張のための関数が用意されている

fashion_mnistの読み込み

```
from keras.datasets import fashion_mnist
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
```

今回はkerasというライブラリに用意されている画像セットを使います

(あらかじめ配列になっています)

MNIST：0~9の文字画像のデータ

FASHION-MNIST：白黒の洋服の画像データ

0 : T-shirt/top、1 : Trouser、2 : Pullover、3 : Dress、4 : Coat、5 : Sandal
6 : Shirt、7 : Sneaker、8 : Bag、9 : Ankle boot

CIFAR10

0 : airplane、1 : automobile、2 : bird、3 : cat、4 : deer、5 : dog
6 : frog、7 : horse、8 : ship、9 : truck

Keras: 深層学習用のライブラリ(入門でも使用)

fashion_mnistの読み込み

```
from keras.datasets import fashion_mnist
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
```

CNNの時と同じように前処理

```
x_train = x_train.reshape(x_train.shape[0],28,28,1)/255
x_test = x_test.reshape(x_test.shape[0],28,28,1)/255
from keras.utils import to_categorical
y_train = to_categorical(y_train,10)
y_test = to_categorical(y_test,10)
```

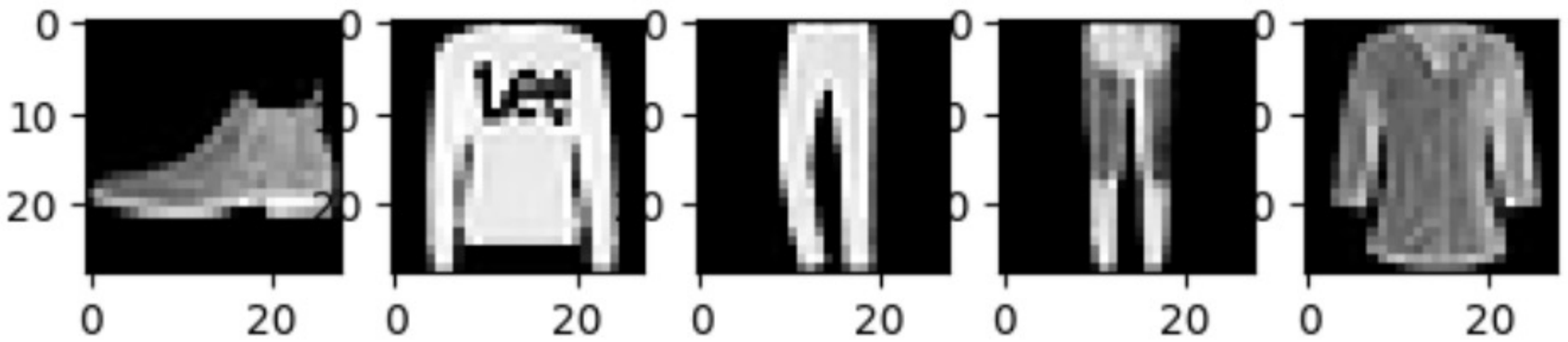
```
x_train = x_train.reshape(x_train.shape[0],28,28,1)/255
```

=

```
x_train = x_train.reshape(x_train.shape[0],28,28,1)
x_train = x_train/255
```

画像を5枚表示

```
import matplotlib.pyplot as plt
for i in range(0,5):
    plt.subplot(1,5,i+1)
    plt.imshow(x_test[i],'gray')
plt.show()
```



255で割っているので各ピクセルは0~255が0~1になっているが、plt.imshow()は
データの最小値と最大値を用いて比率に応じたグレースケール(カラーでも)に変換して表示してくれる

データ拡張の関数 ImageDataGenerator

```
from keras.preprocessing.image import ImageDataGenerator

datagen  = ImageDataGenerator(
        rotation_range = 90
)
g = datagen.flow(x_train, y_train, batch_size=16, shuffle=False)
```

(変数1) = ImageDataGenerator(拡張方法)

rotation_range=90　← ランダムに90度以内の回転をする

(変数2) = (変数1).flow(学習用データ, 学習用ラベル, batch_size, shuffle)

16枚ずつ拡張データを生成する(シャッフルしない)

データ拡張の関数 ImageDataGenerator

len(g)

3750　　(60000/16)

gは要素が3750個ある

g[0]

```
(array([[[[0.        ],          ...,
          [0.        ],          [0.        ],
          [0.        ],          [0.        ],
          ...,                   [0.        ]]]], dtype=float32),
          [0.20739903],   array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
          [0.08286502],          [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
          [0.        ]],         [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
                                 [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
         [[0.        ],          [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
          [0.        ],          [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
          [0.        ],          [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
          ...,                   [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
          [0.        ],          [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
          [0.        ],          [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
          [0.        ]],         [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
                                 [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
         [[0.        ],          [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
          [0.        ],          [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
          [0.        ],          [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
          ...,                   [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]], dtype=float32))
          [0.        ],
```

g[0]は要素が2つ
g[0][0]が16個の画像の配列
g[0][1]が16個の正解ラベル

データ拡張の関数 ImageDataGenerator

g[0][0].shape　(16,28,28,1)

g[0][0]　16枚の画像の配列データ

```
array([[[[0.00000000e+00],
         [0.00000000e+00],
         [0.00000000e+00],
         ...,
         [8.09912235e-02],
         [0.00000000e+00],
         [0.00000000e+00]],

        [[0.00000000e+00],
         [0.00000000e+00],
         [0.00000000e+00],
         ...,
         [0.00000000e+00],
         [0.00000000e+00],
         [0.00000000e+00]],

        [[0.00000000e+00],
         [0.00000000e+00],
         [0.00000000e+00],
         ...,
         [0.00000000e+00],
         [0.00000000e+00],
         [8.65990371e-02]],

        ...,

        [[0.00000000e+00],
         [0.00000000e+00],
         [0.00000000e+00],
         ...,
         [5.16636908e-01],
         [5.30860007e-01],
         [4.98084217e-01]],

        [[0.00000000e+00],
         [0.00000000e+00],
         [0.00000000e+00],
         ...,
         [3.96649212e-01],
         [5.12116015e-01],
         [5.26339054e-01]],

        [[0.00000000e+00],
         [0.00000000e+00],
         [0.00000000e+00],
         ...,
         [2.12643251e-01],
         [3.53700489e-01],
         [4.88819718e-01]]]], dtype=float32)
```

g[0][1]

```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]],
```

16枚の画像の正解ラベル
(one-hot encoding)

データ拡張の関数 ImageDataGenerator

g[0][0].shape　(16,28,28,1)

これがg[3749]まである(計60000枚)

g[0][0]　16枚の画像の配列データ

```
array([[[[0.00000000e+00],
         [0.00000000e+00],
         [0.00000000e+00],
         ...,
         [8.09912235e-02],
         [0.00000000e+00],
         [0.00000000e+00]],

        [[0.00000000e+00],
         [0.00000000e+00],
         [0.00000000e+00],
         ...,
         [0.00000000e+00],
         [0.00000000e+00],
         [0.00000000e+00]],

        [[0.00000000e+00],
         [0.00000000e+00],
         [0.00000000e+00],
         ...,
         [0.00000000e+00],
         [0.00000000e+00],
         [8.65990371e-02]],

        ...,

        ...,

        [[0.00000000e+00],
         [0.00000000e+00],
         [0.00000000e+00],
         ...,
         [5.16636908e-01],
         [5.30860007e-01],
         [4.98084217e-01]],

        [[0.00000000e+00],
         [0.00000000e+00],
         [0.00000000e+00],
         ...,
         [3.96649212e-01],
         [5.12116015e-01],
         [5.26339054e-01]],

        [[0.00000000e+00],
         [0.00000000e+00],
         [0.00000000e+00],
         ...,
         [2.12643251e-01],
         [3.53700489e-01],
         [4.88819718e-01]]]], dtype=float32)
```

g[0][1]

```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]],
```

16枚の画像の正解ラベル
(one-hot encoding)

データ拡張の関数 ImageDataGenerator

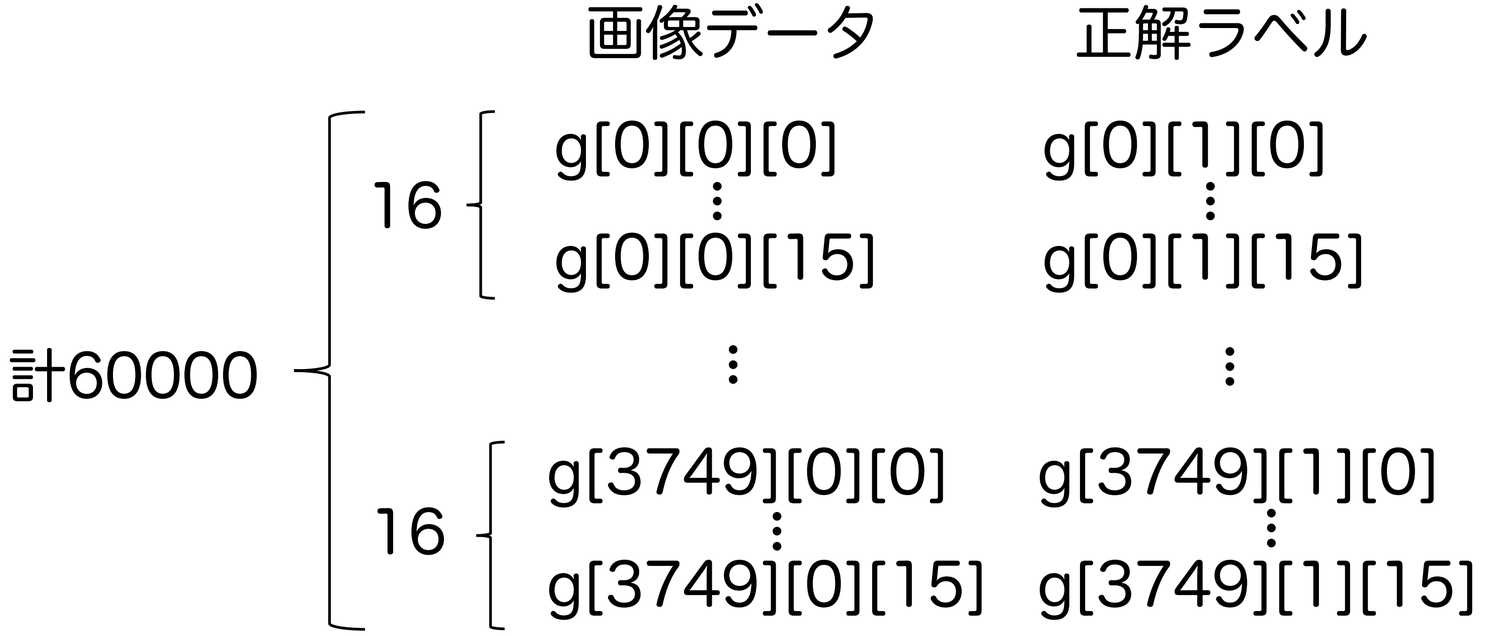
g[0][0][0] 1枚目の画像の配列データ

```
array([[[0.00000000e+00],
        [0.00000000e+00],
        [7.24799000e-03],
        [6.12542732e-04],
        [0.00000000e+00],
        [1.72699317e-01],
        [2.85143852e-01],
        [3.76469344e-01],
        [4.49917577e-02],
        [1.07536756e-03],
        [0.00000000e+00],
        [0.00000000e+00],
        [0.00000000e+00],
        [0.00000000e+00],
        [0.00000000e+00],
        [0.00000000e+00],
        [0.00000000e+00],
        [0.00000000e+00],
```

(28,28,1)

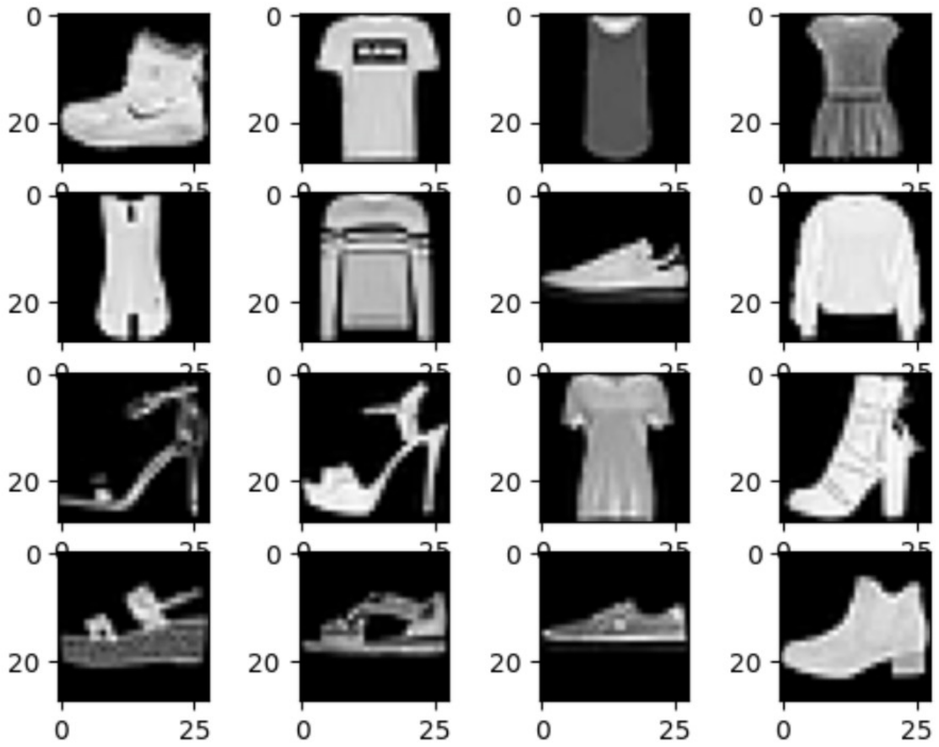
g[0][1][0] 1枚目の画像の正解ラベル (one-hot encoding)

array([0., 0., 0., 0., 0., 0., 0., 0., 0., 1.], dtype=float32)



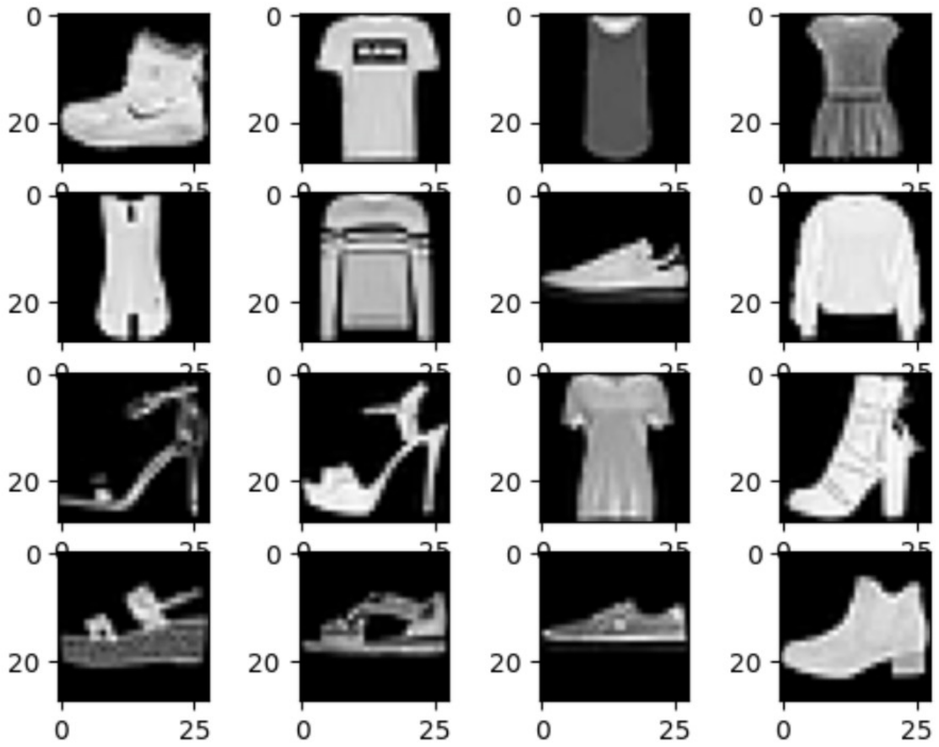
画像を16枚表示

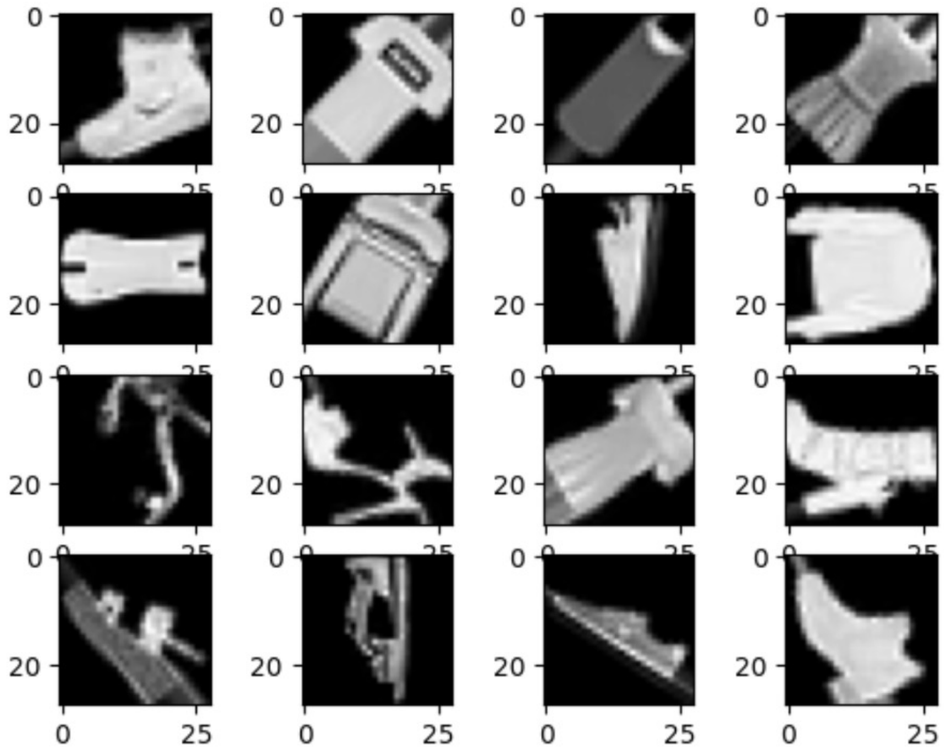
```
import matplotlib.pyplot as plt
for i in range(0,16):
    plt.subplot(4,4,i+1)
    plt.imshow(x_train[i],'gray')
plt.show()
```



増幅した画像を16枚表示

```
import matplotlib.pyplot as plt
for i in range(0,16):
    plt.subplot(4,4,i+1)
    plt.imshow(g[0][0][i],'gray')
plt.show()
```





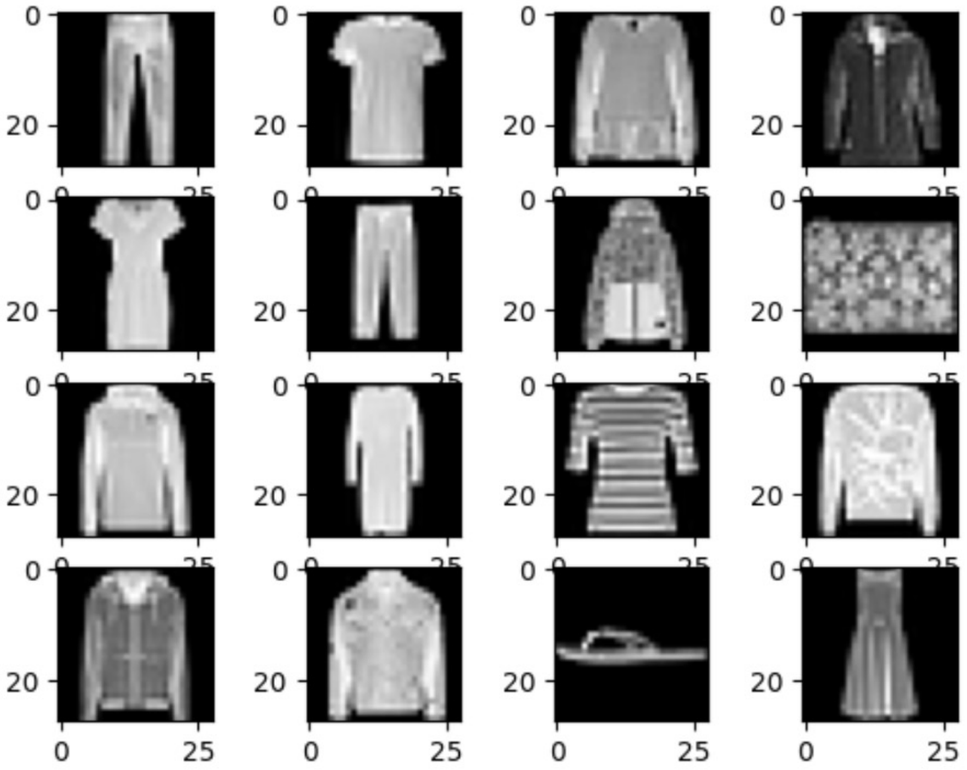
データ拡張の関数 ImageDataGenerator

```
from keras.preprocessing.image import ImageDataGenerator

datagen  = ImageDataGenerator(
        vertical_fllip = True
)
g = datagen.flow(x_train, y_train, batch_size=16, shuffle=False)
```

vertical_flip = True ← ランダムに上下反転

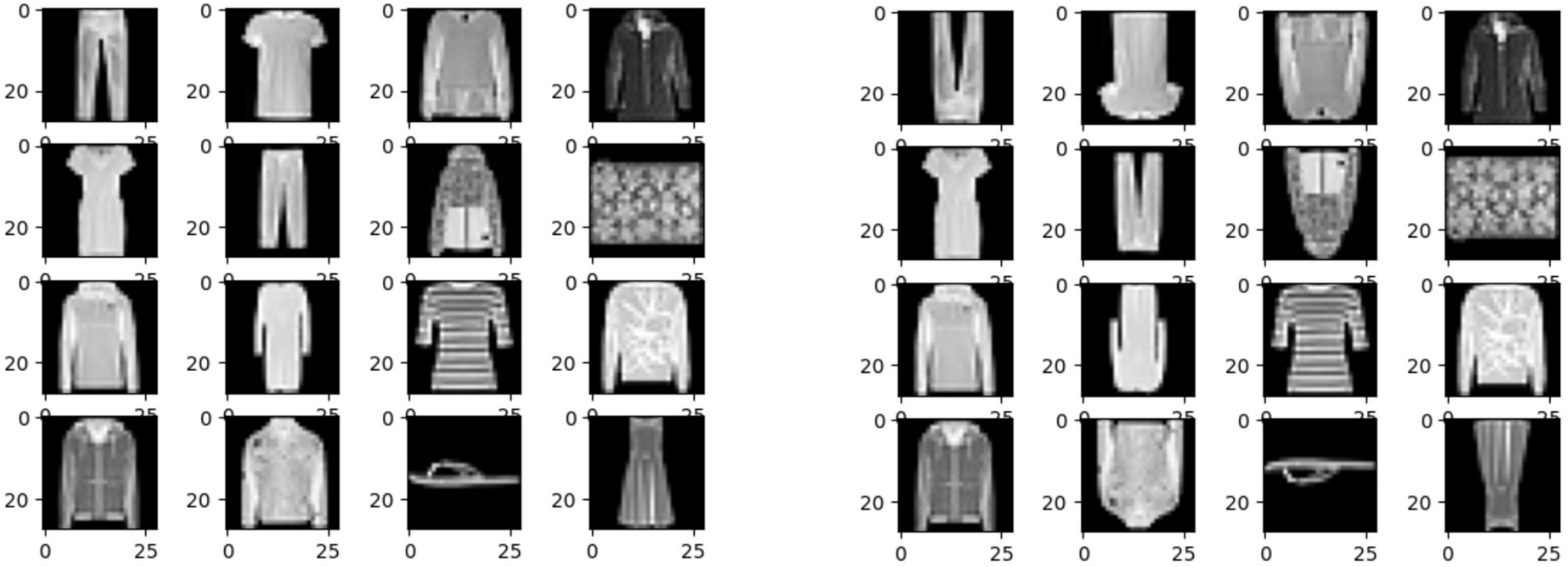
画像を16枚表示(次の16枚)

```
import matplotlib.pyplot as plt
for i in range(16,32):
    plt.subplot(4,4,i-15)
    plt.imshow(x_train[i],'gray')
plt.show()
```



増幅した画像を16枚表示(次の16枚)

```
import matplotlib.pyplot as plt
for i in range(16,32):
    plt.subplot(4,4,i-15)
    plt.imshow(g[1][0][i-16],'gray')
plt.show()
```



データ拡張の関数 ImageDataGenerator

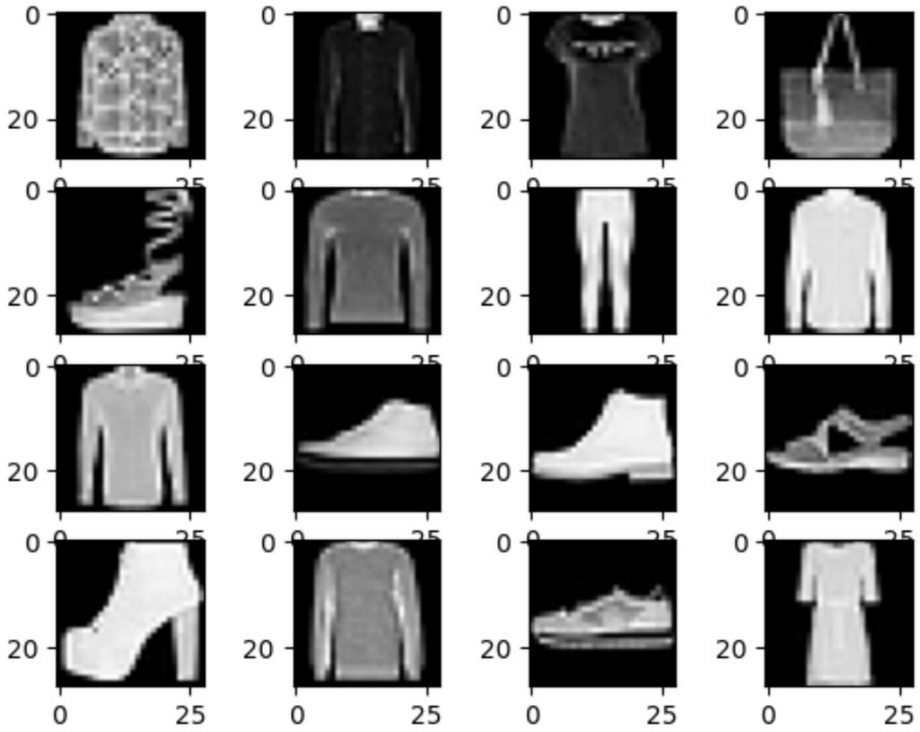
```
from keras.preprocessing.image import ImageDataGenerator

datagen  = ImageDataGenerator(
        rotation_range = 90,
        vertical_fllip = True
)
g = datagen.flow(x_train, y_train, batch_size=16, shuffle=True)
```

ランダムに90以内の回転と上下反転を実施

順番もシャッフル

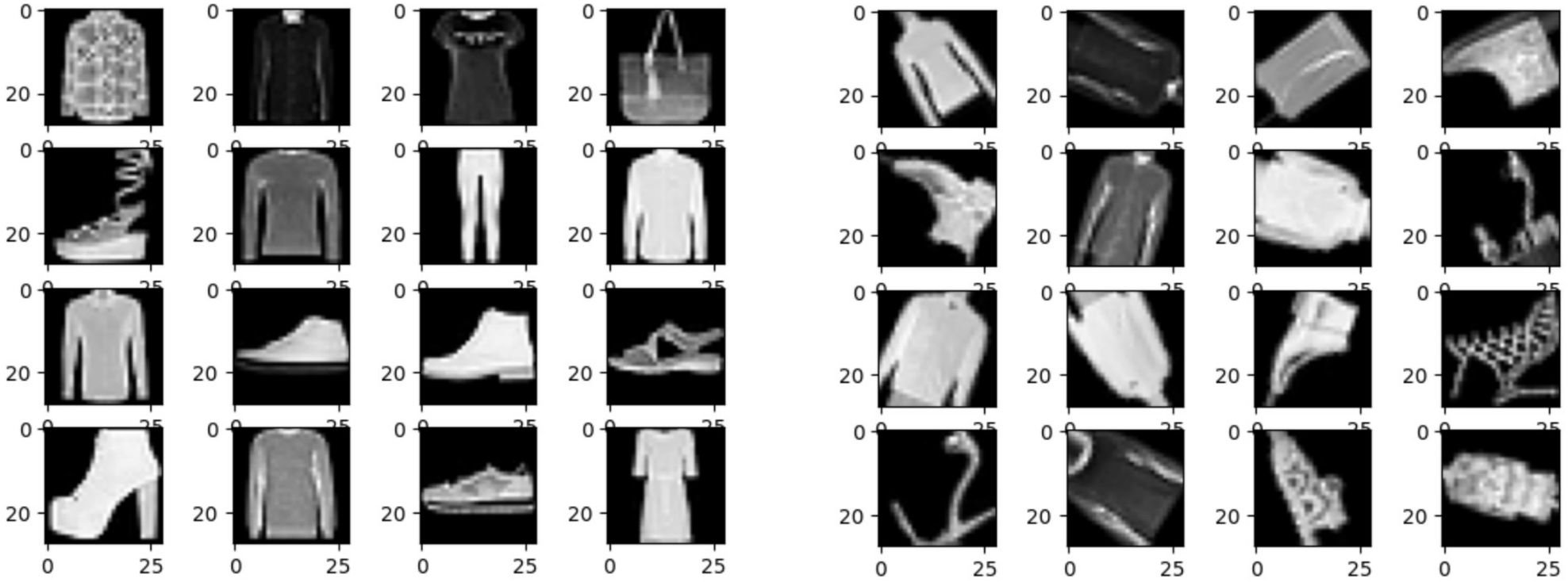
画像を16枚表示(次の16枚)

```
import matplotlib.pyplot as plt
for i in range(32,48):
    plt.subplot(4,4,i-31)
    plt.imshow(x_train[i],'gray')
plt.show()
```



増幅した画像を16枚表示(次の16枚)

```
import matplotlib.pyplot as plt
for i in range(32,48):
    plt.subplot(4,4,i-31)
    plt.imshow(g[2][0][i-32],’gray’)
plt.show()
```



他にも色んなバリエーションがある

- 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．

https://keras.io/ja/preprocessing/image/

kerasの公式ホームページより

モデルの作成

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Conv2D, Flatten, MaxPooling2D

model = Sequential()
model.add(Conv2D(filters=32,kernel_size=3, strides=1,
                 padding='same',input_shape=(28,28,1),activation='relu'))
model.add(Conv2D(filters=64,kernel_size=3, strides=1,
                 padding='same', activation='relu'))

model.add(MaxPooling2D(pool_size=2))

model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(10,activation='softmax'))

model.compile(loss='categorical_crossentropy',
                              optimizer='Adam',metrics=['accuracy'])
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 28, 28, 64)        18496
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 64)        0
_________________________________________________________________
flatten_4 (Flatten)          (None, 12544)             0
_________________________________________________________________
dropout_4 (Dropout)          (None, 12544)             0
_________________________________________________________________
dense_10 (Dense)             (None, 10)                125450
=================================================================
Total params: 144,266
Trainable params: 144,266
Non-trainable params: 0
_________________________________________________________________
```

学習の仕方(まだ実行しない)

```
datagen  = ImageDataGenerator(
        rotation_range = 90
)
g = datagen.flow(x_train, y_train, batch_size=64)
result = model.fit(g, epochs = 50, validation_data=(x_val, y_val),
                                                steps_per_epoch=x_train.shape[0]//64)
```

64枚ずつ拡張した画像を学習させる

学習の仕方(まだ実行しない)

```
datagen  = ImageDataGenerator(
        rotation_range = 90
)
g = datagen.flow(x_train, y_train, batch_size=64)
result = model.fit(g, epochs = 50, validation_data=(x_val, y_val),
                                                steps_per_epoch=x_train.shape[0]//64)
```

64枚ずつ拡張した画像を学習させる

通常のmodel.fit()はvalidation_splitでtrainを分けていたが、
今回はvalidation_data=(x_val, y_val)であらかじめデータを用意する

学習の仕方(前処理)

```
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2)
print(x_train.shape)
print(x_val.shape)
print(y_train.shape)
print(y_val.shape)
```

```
(48000, 28, 28, 1)
(12000, 28, 28, 1)
(48000, 10)
(12000, 10)
```

train_test_split(学習用データ,学習用ラベル, test_size=割合)
で指定した割合にデータを分割する

学習の仕方(実行)

```
datagen  = ImageDataGenerator(
        rotation_range = 90
)
g = datagen.flow(x_train, y_train, batch_size=64)
result = model.fit(g, epochs = 50, validation_data=(x_val, y_val),
                                    steps_per_epoch=x_train.shape[0]//64)
```

48000

12000

64枚ずつ拡張した画像を学習させる

通常のmodel.fit()はvalidation_splitでtrainを分けていたが、
今回はvalidation_data=(x_val, y_val)であらかじめデータを用意する

steps_per_epoch=1エポックで更新する回数
通常のmodel.fit()と違い、数を指定する必要がある

今回はx_train.shape[0]//64=750 (通常のmodel.fit()と同じ条件)

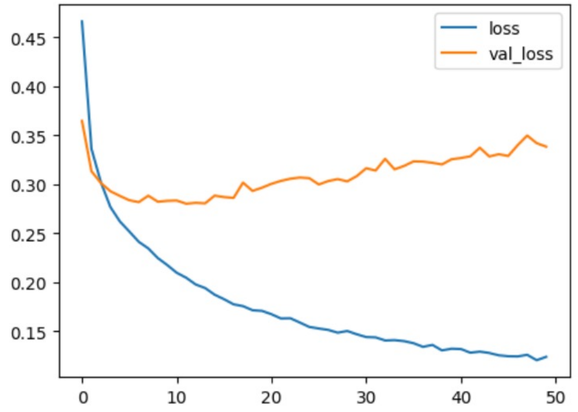
(x_train.shape[0]は48000、”//”は割り算の整数)

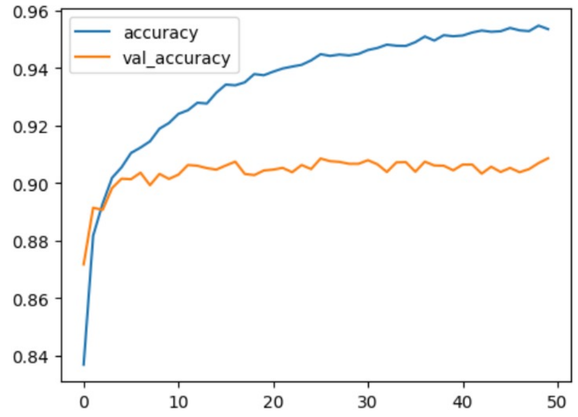
結果の比較

畳み込み層1つ

test loss: 0.34445714950561523

test accuracy: 0.9031999707221

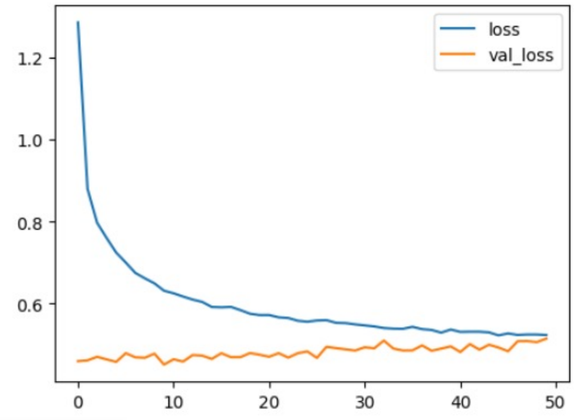


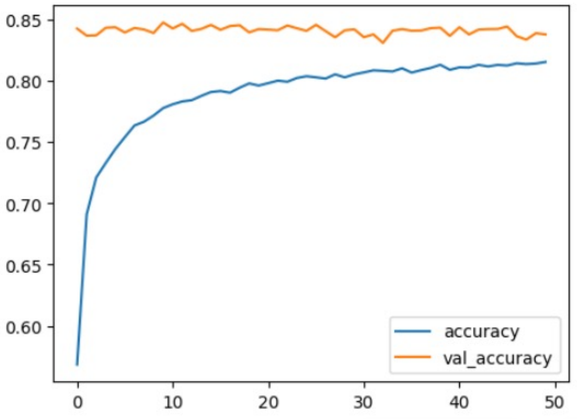


畳み込み層1つ(データ拡張)

test loss: 0.5629728436470032

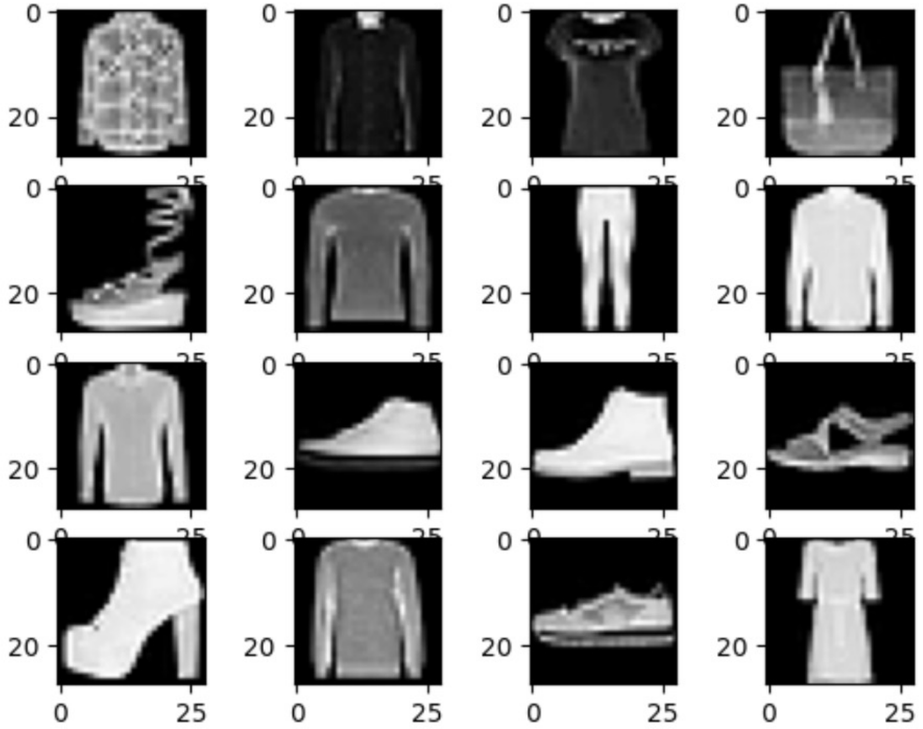
test accuracy: 0.8313000202178955



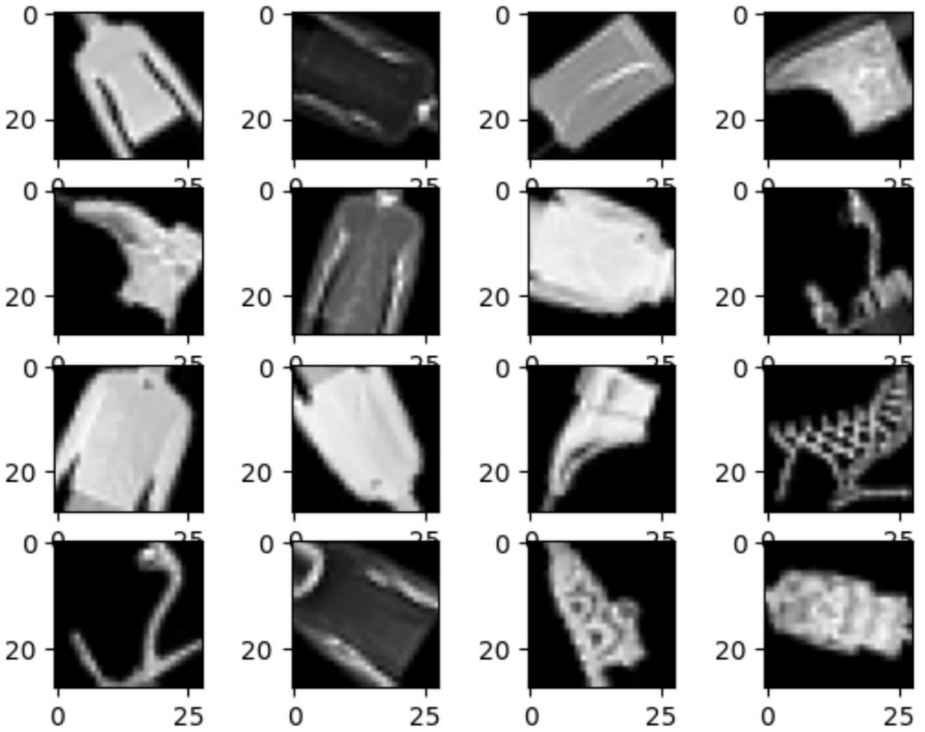


あれ？

実はfashion_mnistはあまりデータ拡張に適していない



元のデータ



回転と上下反転

元のデータが綺麗に並んでいるため、あまり効果が期待できない
・ほぼ洋服のサイズが同じで左右対称
・靴の先端は必ず左を向いている
→ 加工すると無駄にデータを学習させてしまう

どうゆう時に適しているか

cifar10
(次回の
グループ演習
で使用予定)

物体の大きさや向きが違うような画像分類では回転や反転などが有効な可能性が高い

どうゆう加工なら有効か元のデータの性質も把握しておくことが重要

課題

・WebClassにある課題8をやりましょう

締め切りは1週間後の6/6の23:59です。
締め切りを過ぎた課題は受け取らないので注意して下さい。
(1週間後に正解をアップします)