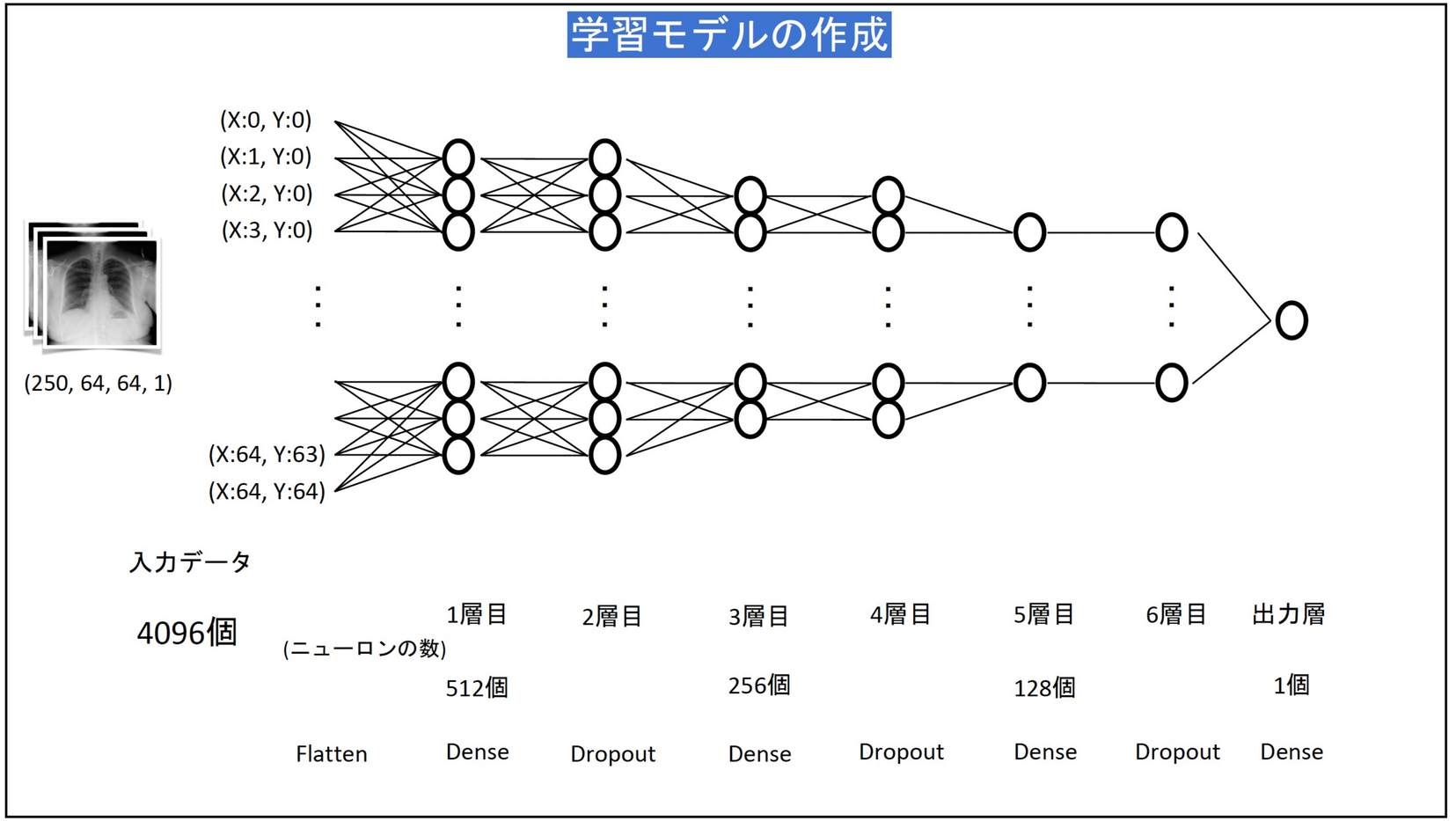
医療とAI・ビッグデータ応用
CNN

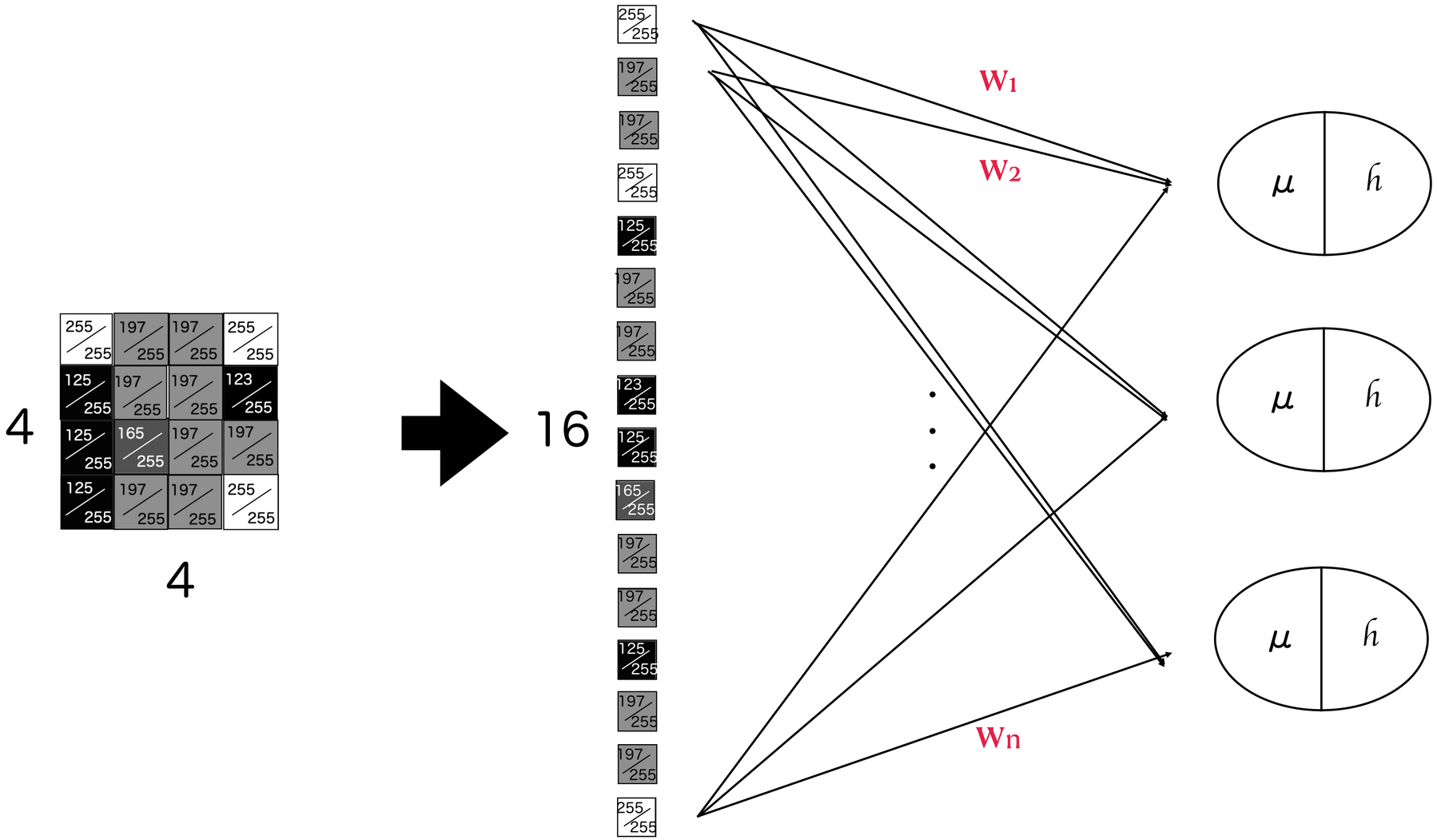
統合教育機構
須藤毅顕

前回の深層学習はMLP



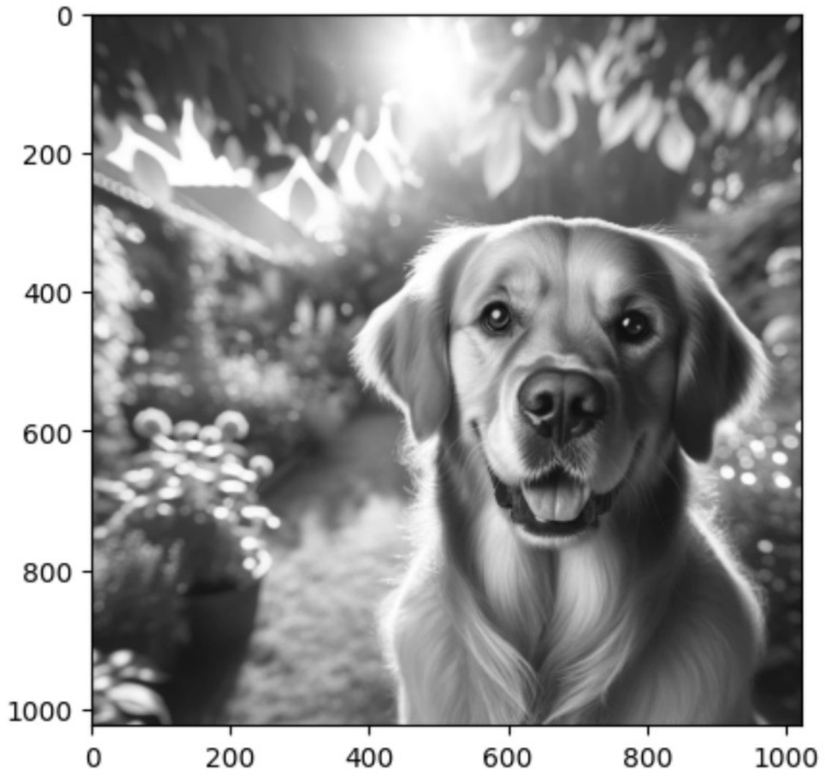
これよりも高い精度が出せるニューラルネットワークである、
CNN(Convolutional Neural Network)に取り組みます。

MLPの問題点1　MLPでは画像サイズを１次元にして入力する→画像サイズ分の重みが存在

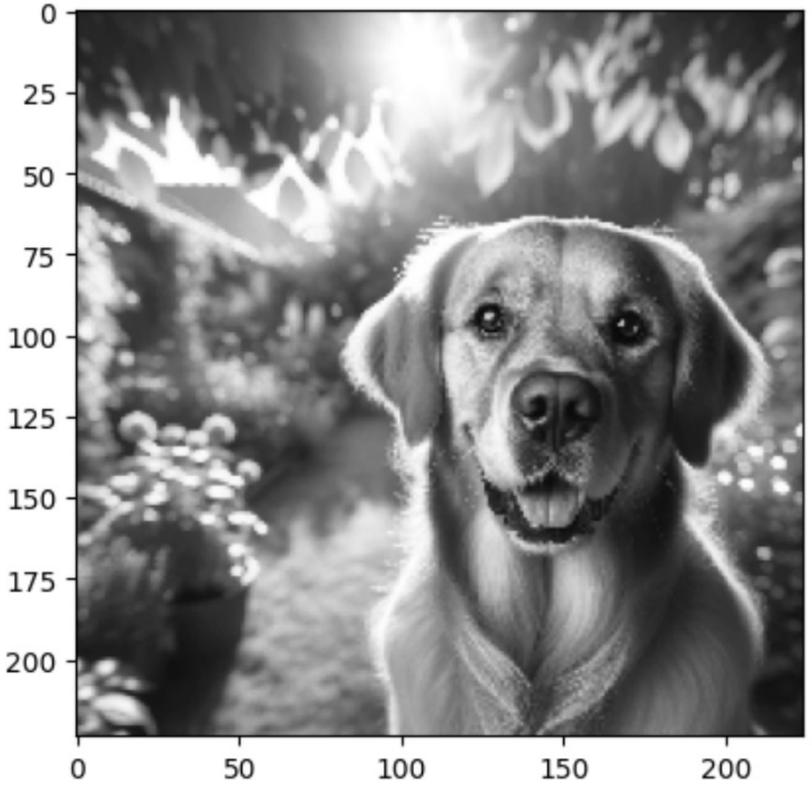


サイズが大きいほど調整する重みが増えてしまう

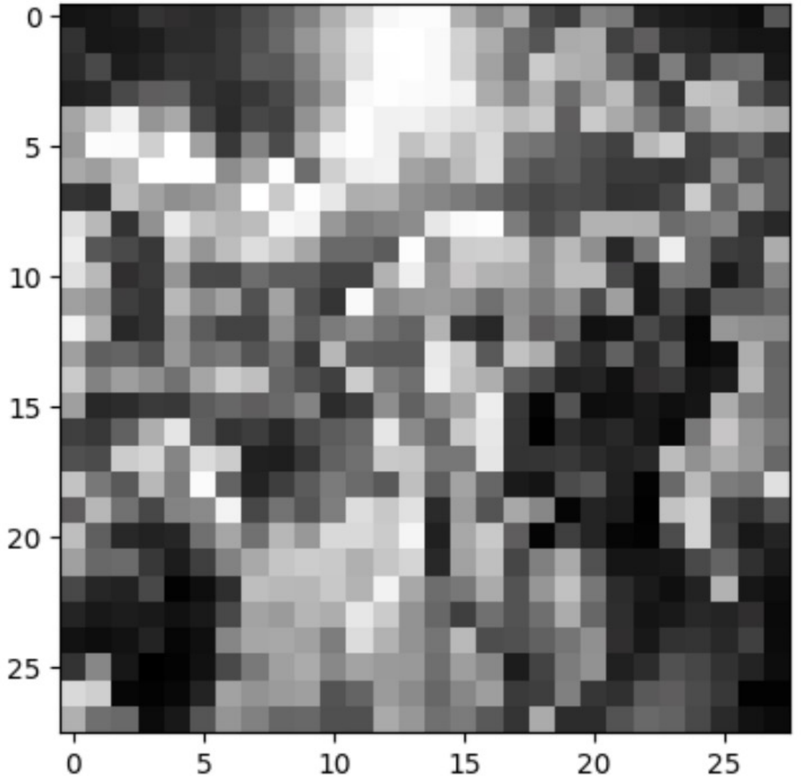
MLPの問題点1　MLPでは画像サイズを１次元にして入力する→画像サイズ分の重みが存在



1024×1024



224×224

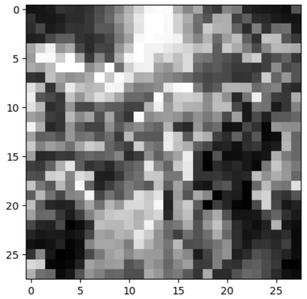


28×28

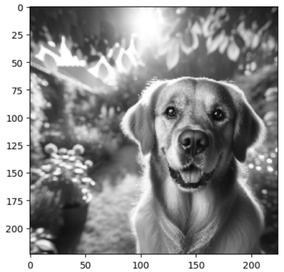
28×28では犬と分からない
→1024か224を入力したい

MLPの問題点1

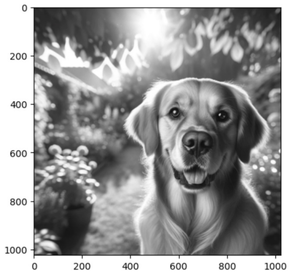
MLPでは画像サイズを１次元にして入力する→画像サイズ分の重みが存在



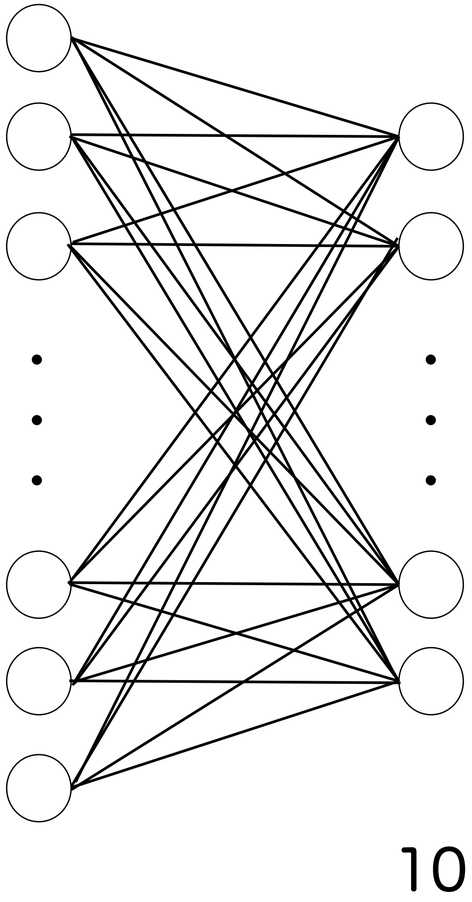
28×28



224×224



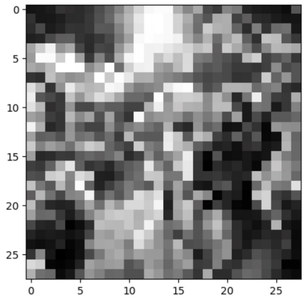
1024×1024



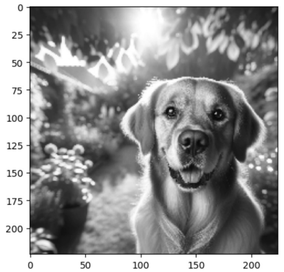
入力サイズが28×28
入力層と1つ目の中間層の間の重み
784×10 = 7850

MLPの問題点1

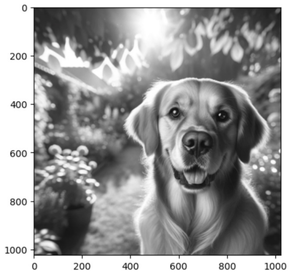
MLPでは画像サイズを１次元にして入力する→画像サイズ分の重みが存在



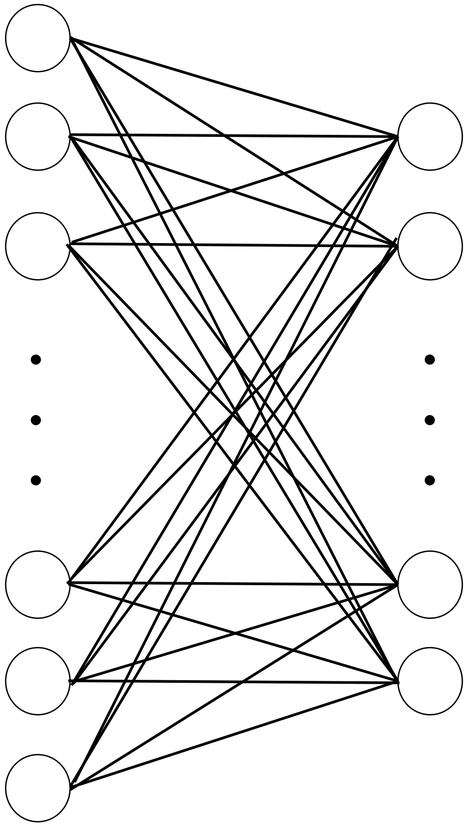
28×28



224×224



1024×1024



50176

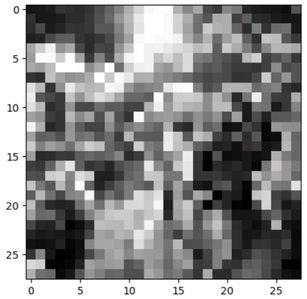
10

入力サイズが28×28
入力層と1つ目の中間層の間の重み
784×10 = 7850

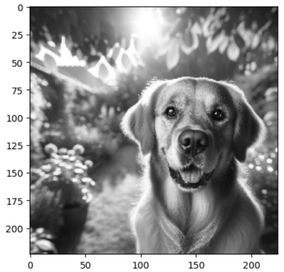
入力サイズが224×224
入力層と1つ目の中間層の間の重み
50176×10 = 501760

MLPの問題点1

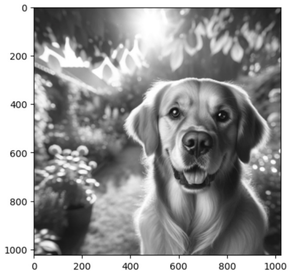
MLPでは画像サイズを１次元にして入力する→画像サイズ分の重みが存在



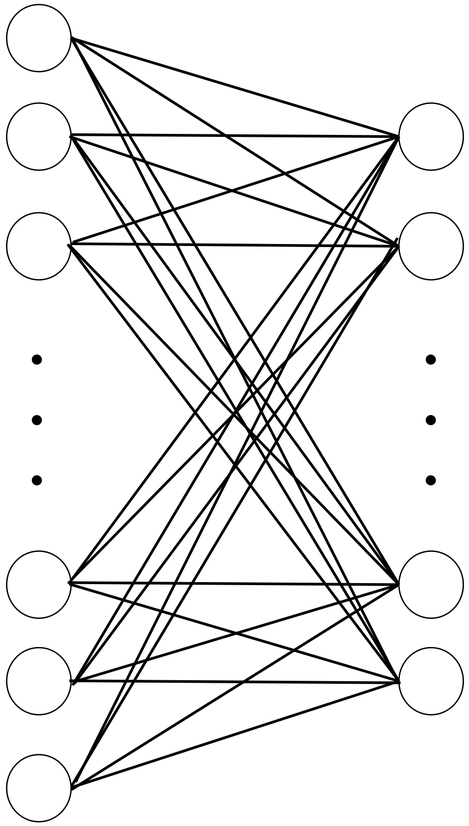
28×28



224×224



1024×1024



1048576

10

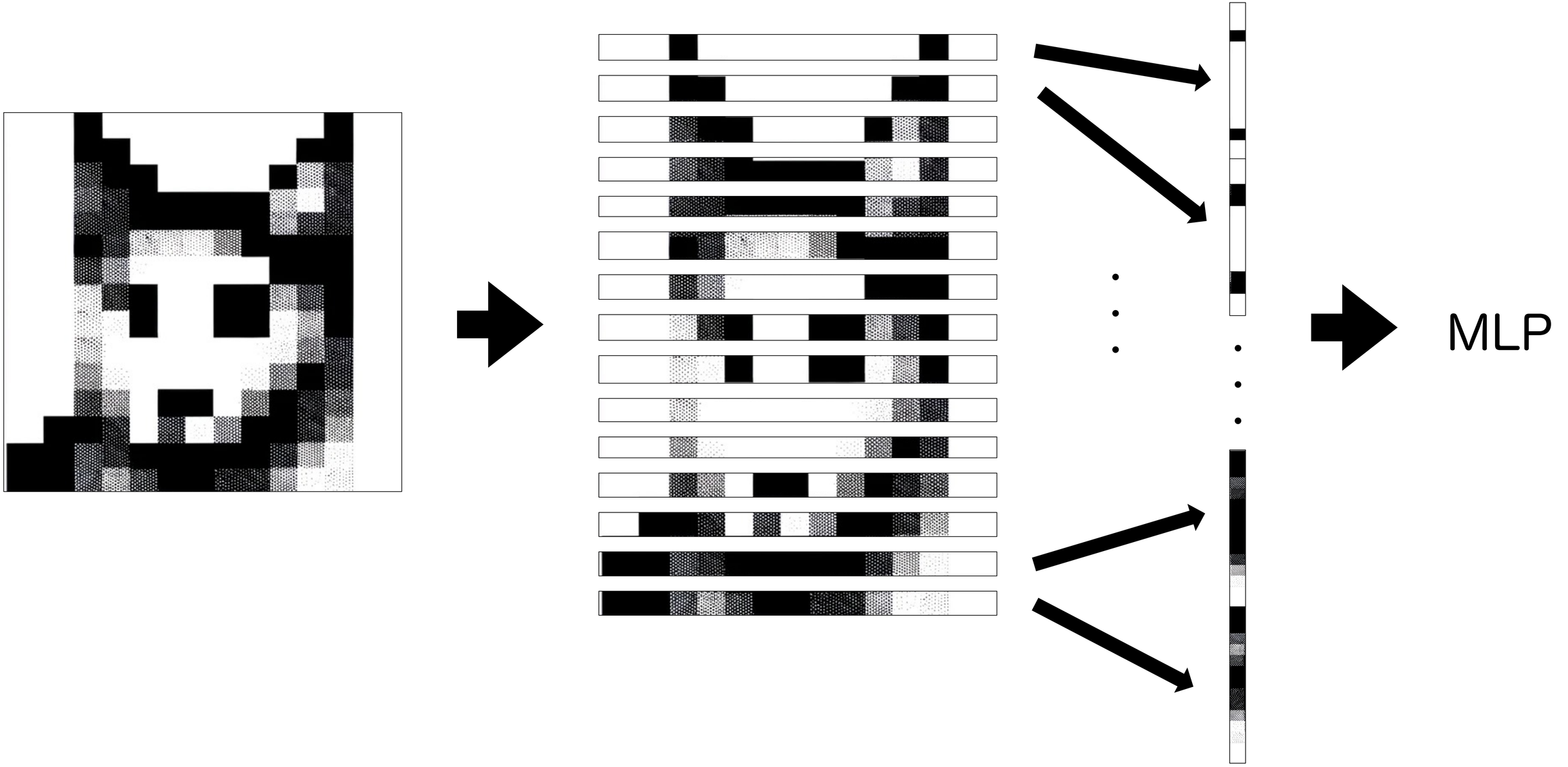
入力サイズが28×28
入力層と1つ目の中間層の間の重み
784×10 = 7850

入力サイズが224×224
入力層と1つ目の中間層の間の重み
50176×10 = 501760

入力サイズが1024×1024
入力層と1つ目の中間層の間の重み
1048576×10 = 10485760

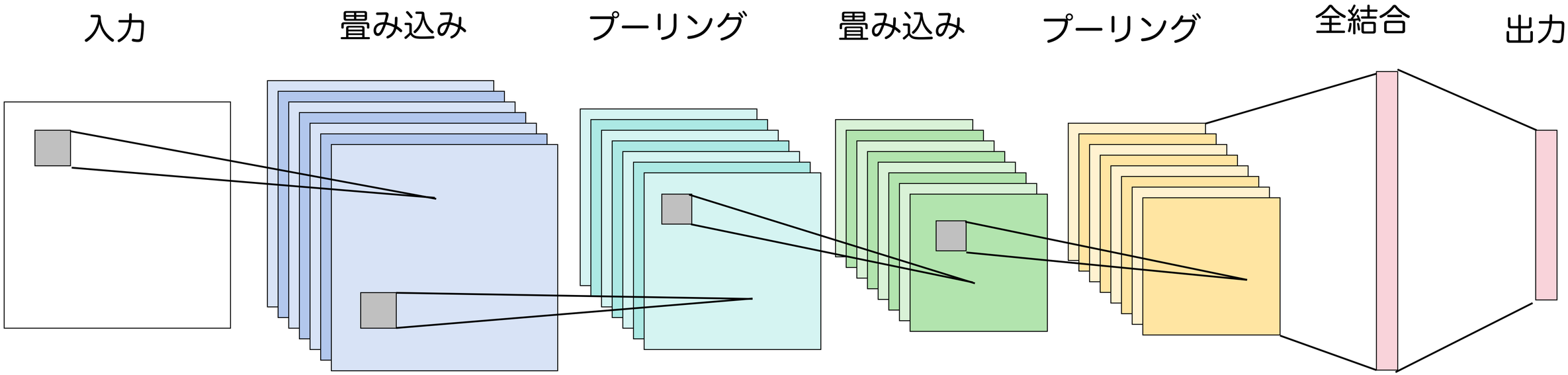
サイズが大きいほど調整する重みが増えてしまう

MLPの問題点2　MLPでは画像サイズを１次元にして入力する→画像の2次元の特徴を失う



CNN：Convolutional Neural Network

畳み込み層とプーリング層が繰り返されるニューラルネットワーク



一次元にせず、そのまま2次元の配列のまま入力する
重みの数は入力サイズと関係がない(カーネルに依存する)

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

print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,
```

CNNの場合は1次元にしない
(縦, 横, 色の数)
の3次元で入力する
白黒なら1、カラーなら3

```
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)

print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(60000, 28, 28,1)
(60000,)
(10000, 28, 28,1)
(10000,
```

MLPの時は、
(60000, 784)
(60000,)
(10000, 784)
(10000,)

モデルの作成

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

model = Sequential()
model.add(Conv2D(filters=32,kernel_size=3,strides=1,
                 padding='same',input_shape=(28,28,1),activation='relu'))
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_2 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
flatten_2 (Flatten)          (None, 25088)             0
_________________________________________________________________
dropout_2 (Dropout)          (None, 25088)             0
_________________________________________________________________
dense_8 (Dense)              (None, 10)                250890
=================================================================
Total params: 251,210
Trainable params: 251,210
Non-trainable params: 0
_________________________________________________________________
```

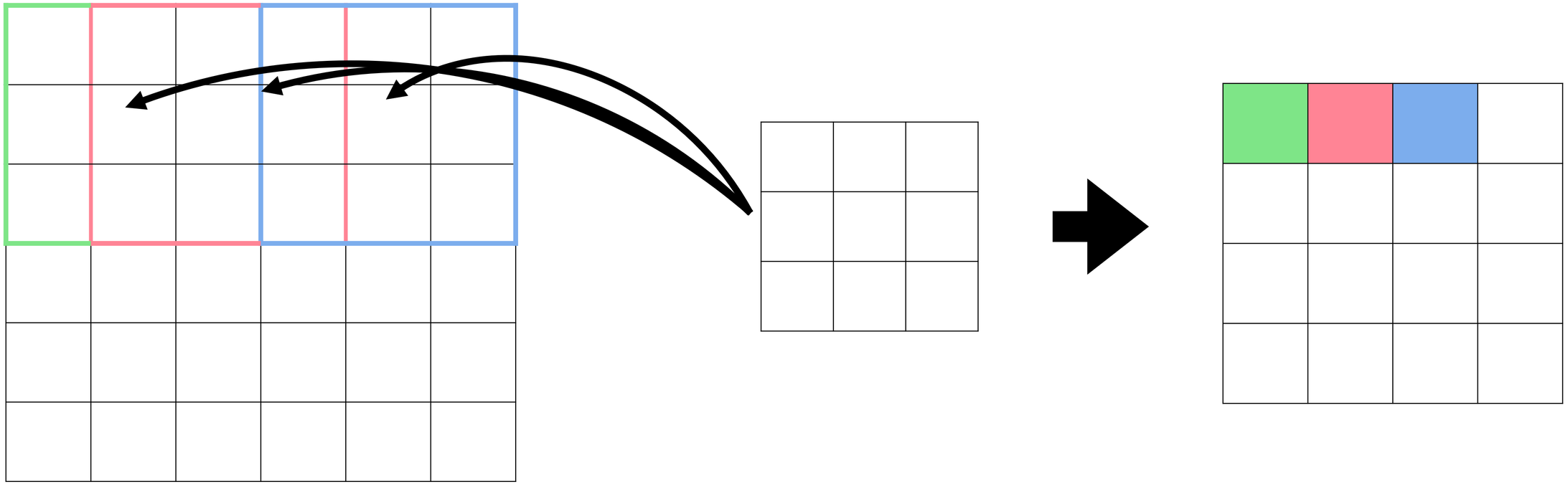
学習の実行

```
result = model.fit(x_train, y_train, epochs = 50, batch_size = 64, validation_split=0.2)
```

```
Epoch 47/50
750/750 [==============================] - 3s 4ms/step - loss: 0.1280 - accuracy: 0.9518 - val_loss: 0.3274 - val_accuracy: 0.9071
Epoch 48/50
750/750 [==============================] - 3s 4ms/step - loss: 0.1218 - accuracy: 0.9540 - val_loss: 0.3313 - val_accuracy: 0.9069
Epoch 49/50
750/750 [==============================] - 3s 4ms/step - loss: 0.1228 - accuracy: 0.9542 - val_loss: 0.3338 - val_accuracy: 0.9082
Epoch 50/50
750/750 [==============================] - 3s 4ms/step - loss: 0.1214 - accuracy: 0.9555 - val_loss: 0.3431 - val_accuracy: 0.9066
```

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法



入力層　　カーネル　　出力層

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法

0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

入力層

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法

0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

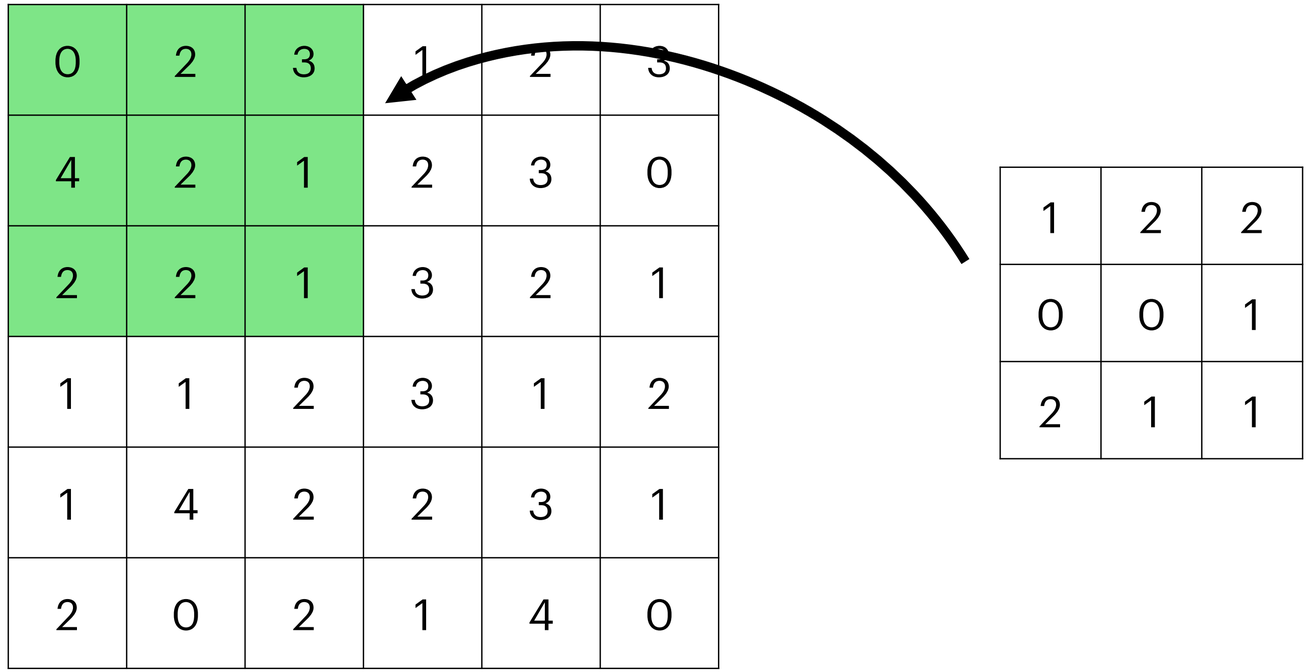
入力層

1	2	2
0	0	1
2	1	1

カーネル

畳み込み層

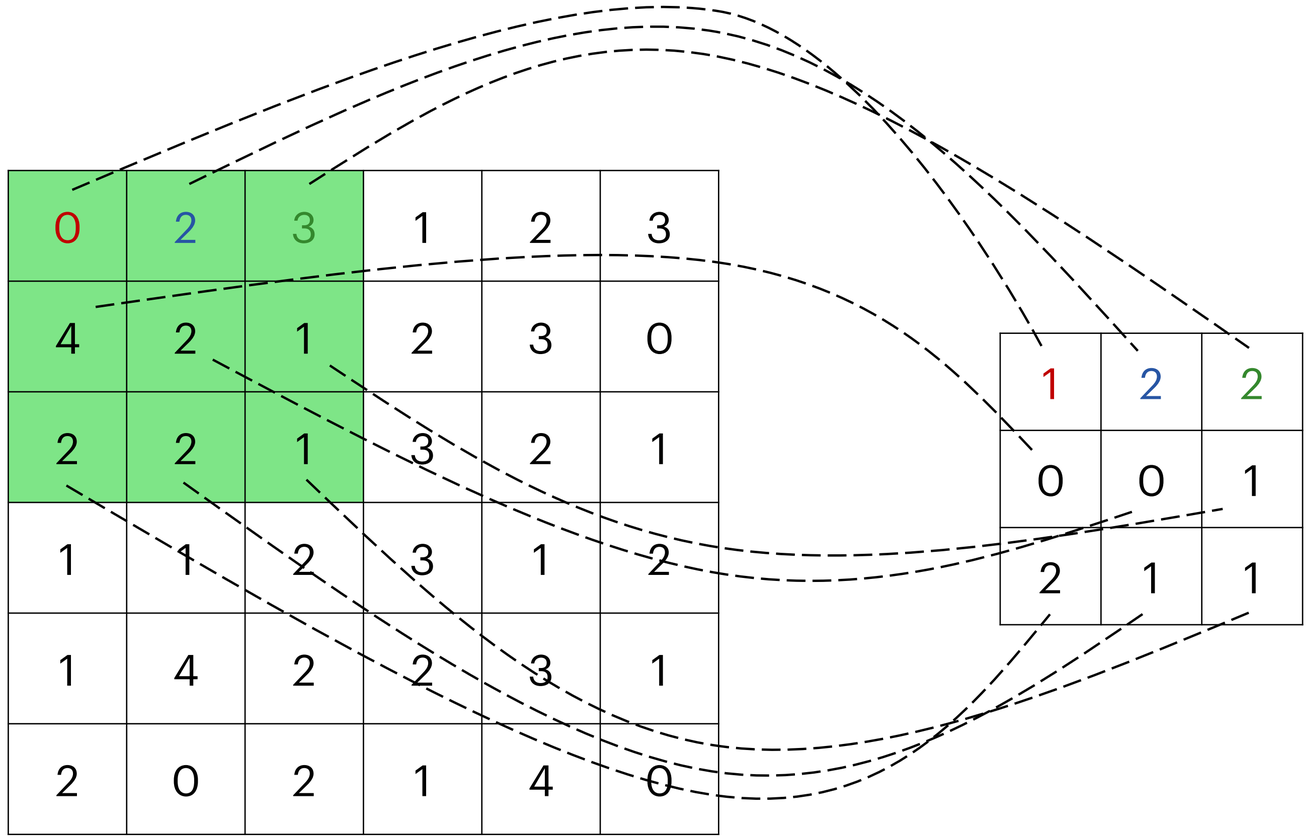
入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法



入力層　　カーネル

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法



入力層　　カーネル

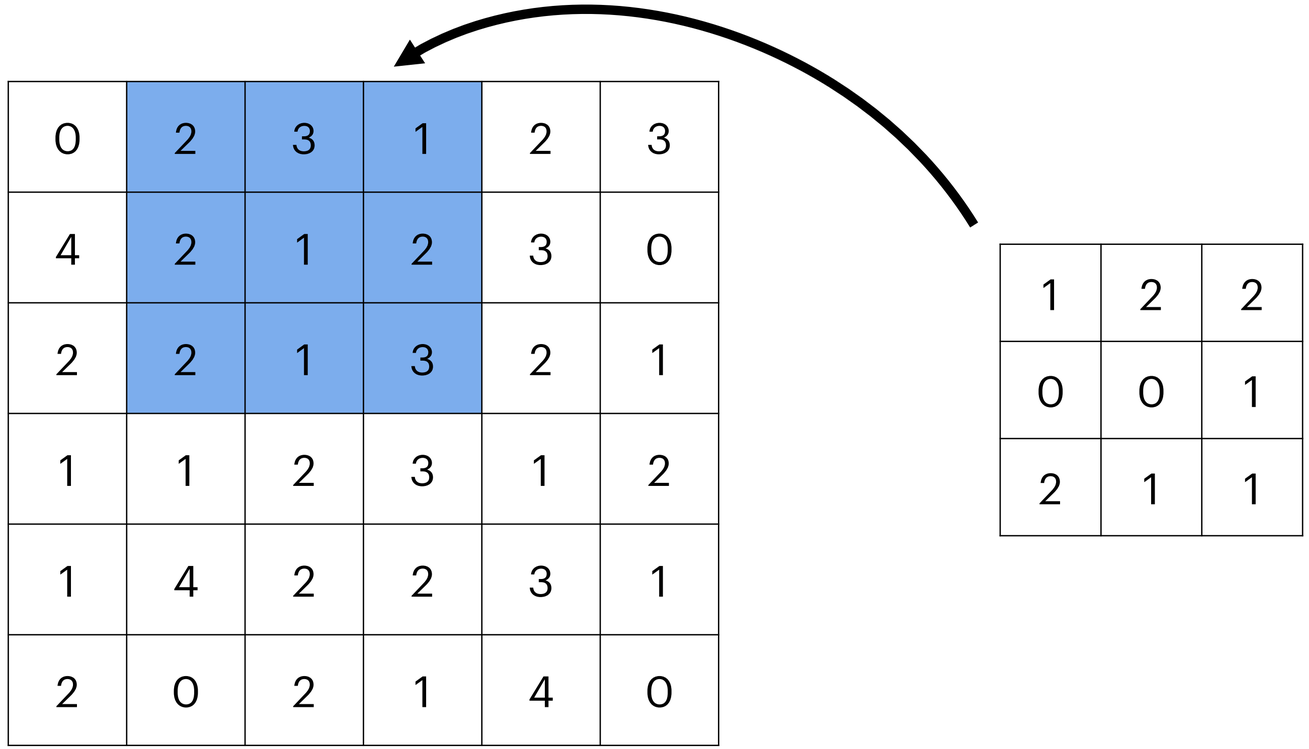
0×1	2×2	3×2
4×0	2×0	1×1
2×2	2×1	1×1

全ての積和を計算する

0×1+2×2+3×2+4×0+2×0
+1×1+2×2+2×1+1×1 = 18

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法



0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

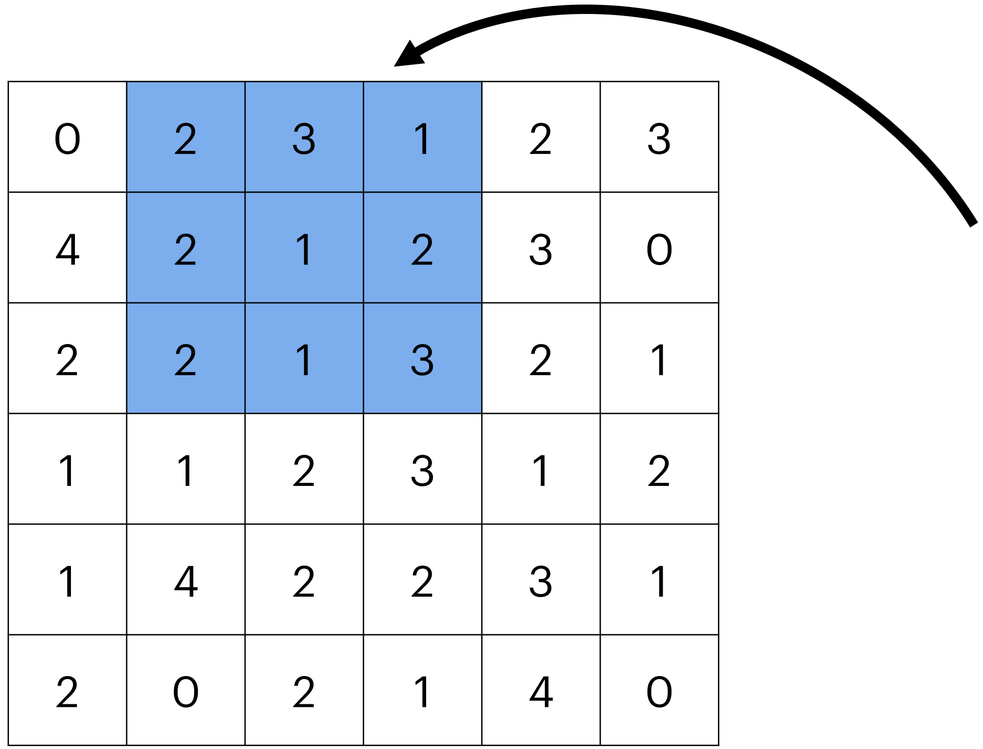
入力層

1	2	2
0	0	1
2	1	1

カーネル

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法



0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

入力層

1	2	2
0	0	1
2	1	1

カーネル

2×1	3×2	1×2
2×0	1×0	3×1
2×2	1×1	3×1

全ての積和を計算する

2×1+3×2+1×2+2×0+1×0
+3×1+2×2+1×1+3×1 = 22

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法

0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

入力層 (6 × 6)

1	2	2
0	0	1
2	1	1

カーネル (3× 3)

18	22		

特徴マップ

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法

0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

入力層 (6 × 6)

1	2	2
0	0	1
2	1	1

カーネル (3× 3)

18	22	19	

特徴マップ

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法

0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

入力層 (6 × 6)

1	2	2
0	0	1
2	1	1

カーネル (3× 3)

18	22	19	20

特徴マップ

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法

0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

入力層 (6 × 6)

1	2	2
0	0	1
2	1	1

カーネル (3× 3)

18	22	19	20
16			

特徴マップ

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法

0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

入力層 (6 × 6)

1	2	2
0	0	1
2	1	1

カーネル (3× 3)

18	22	19	20
16	18		

特徴マップ

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法

0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

入力層 (6 × 6)

1	2	2
0	0	1
2	1	1

カーネル (3× 3)

18	22	19	20
16	18	21	

特徴マップ

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法

0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

入力層 (6 × 6)

1	2	2
0	0	1
2	1	1

カーネル (3× 3)

18	22	19	20
16	18	21	18

特徴マップ

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法

0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

入力層 (6 × 6)

1	2	2
0	0	1
2	1	1

カーネル (3× 3)

18	22	19	20
16	18	21	18
18			

特徴マップ

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法

0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

入力層 (6 × 6)

1	2	2
0	0	1
2	1	1

カーネル (3× 3)

18	22	19	20
16	18	21	18
18	25		

特徴マップ

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法

0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

入力層 (6 × 6)

1	2	2
0	0	1
2	1	1

カーネル (3× 3)

18	22	19	20
16	18	21	18
18	25	21	

特徴マップ

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法

0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

入力層 (6 × 6)

1	2	2
0	0	1
2	1	1

カーネル (3× 3)

18	22	19	20
16	18	21	18
18	25	21	19

特徴マップ

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法

0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

入力層 (6 × 6)

1	2	2
0	0	1
2	1	1

カーネル (3× 3)

18	22	19	20
16	18	21	18
18	25	21	19
15			

特徴マップ

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法

0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

入力層 (6 × 6)

1	2	2
0	0	1
2	1	1

カーネル (3× 3)

18	22	19	20
16	18	21	18
18	25	21	19
15	17		

特徴マップ

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法

0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

入力層 (6 × 6)

1	2	2
0	0	1
2	1	1

カーネル (3× 3)

18	22	19	20
16	18	21	18
18	25	21	19
15	17	22	

特徴マップ

畳み込み層

入力データに対してカーネルと呼ばれる小さな行列をスライドさせながら学習させる手法

0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

入力層 (6 × 6)

1	2	2
0	0	1
2	1	1

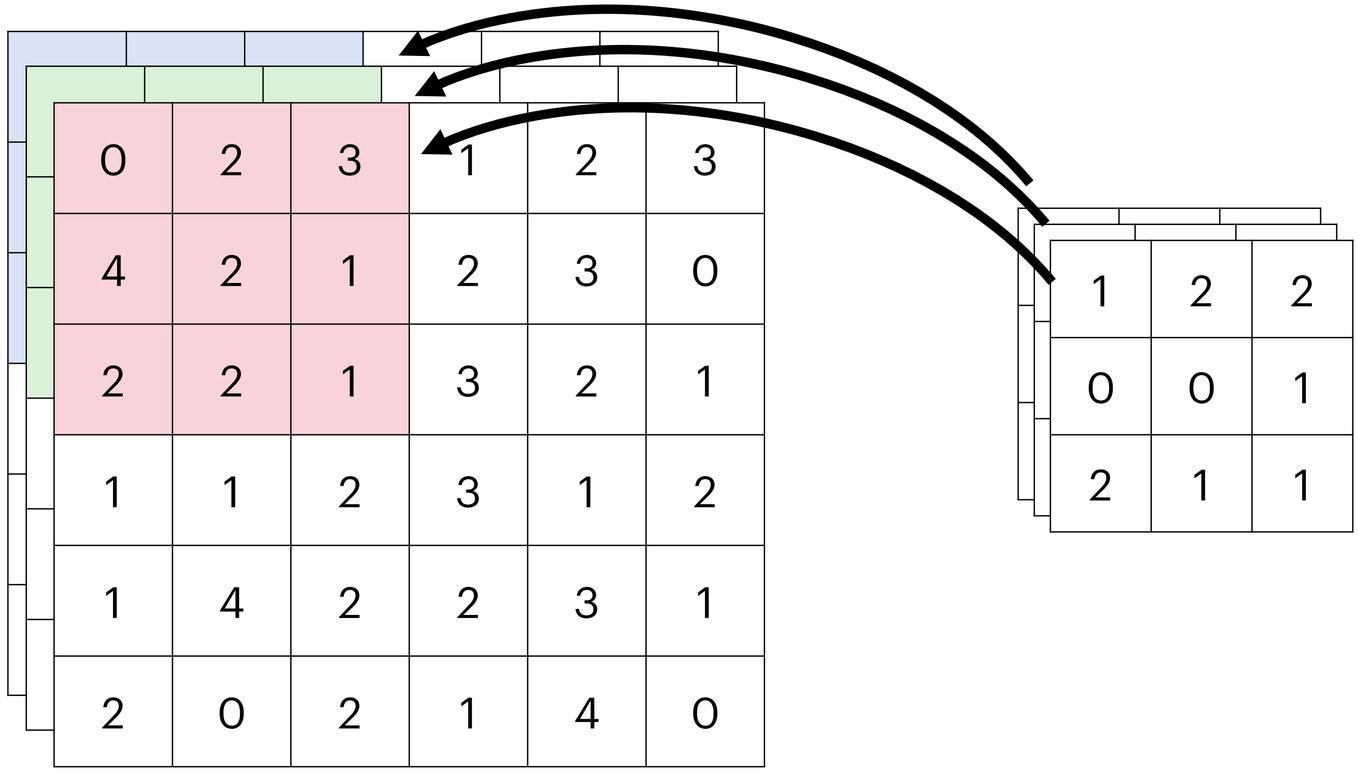
カーネル (3× 3)

18	22	19	20
16	18	21	18
18	25	21	19
15	17	22	16

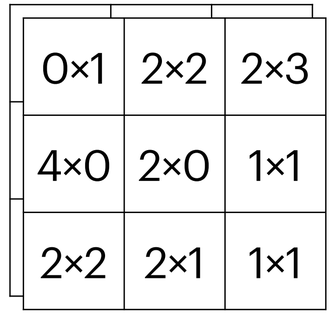
特徴マップ

入力層が3次元の場合

３次元(カラー)の場合は、カーネルも3つになる



入力層　　カーネル



全ての和を計算する

0×1+2×2+2×3+4×0+2×0
+1×1+2×2+2×1+1×1

+ (緑の積和) + (青の積和)

カラー(RGB)は赤、緑、青が0~255

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

filters = 出力する特徴マップの数

input_shape = MLPと同様に入力層の形

kernel_size = カーネルの大きさ

strides = カーネルをずらす幅

padding = データの端をどう扱うか

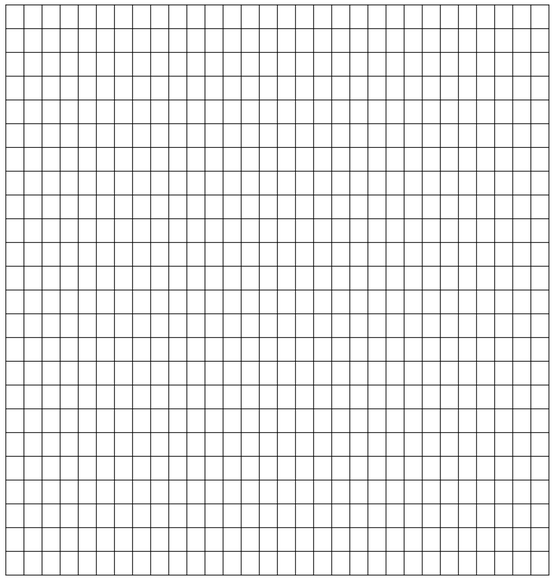
activation = 活性化関数

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

filters = 出力する特徴マップの数

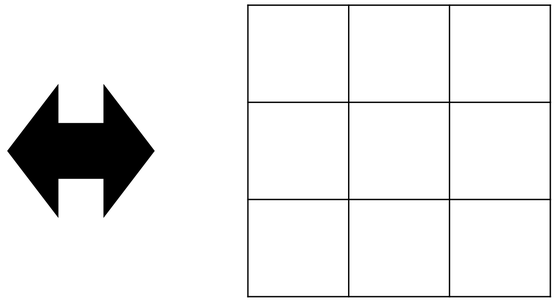
input_shape = MLPと同様に入力層の形

kernel_size = カーネルの大きさ

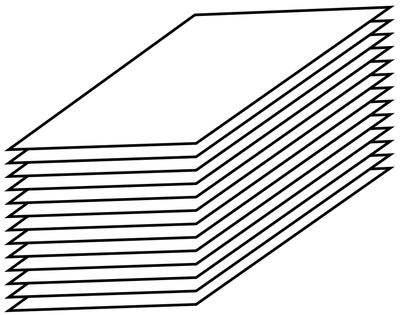


(28, 28, 1)

カラーなら(28, 28, 3)



3 × 3



32

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

strides = カーネルをずらす幅

strides = 1

0	2	3	1	2
4	2	1	2	3
2	2	1	3	2
1	1	2	3	1
1	4	2	2	3

⟷

1	2	2
0	0	1
2	1	1

strides = 2

0	2	3	1	2
4	2	1	2	3
2	2	1	3	2
1	1	2	3	1
1	4	2	2	3

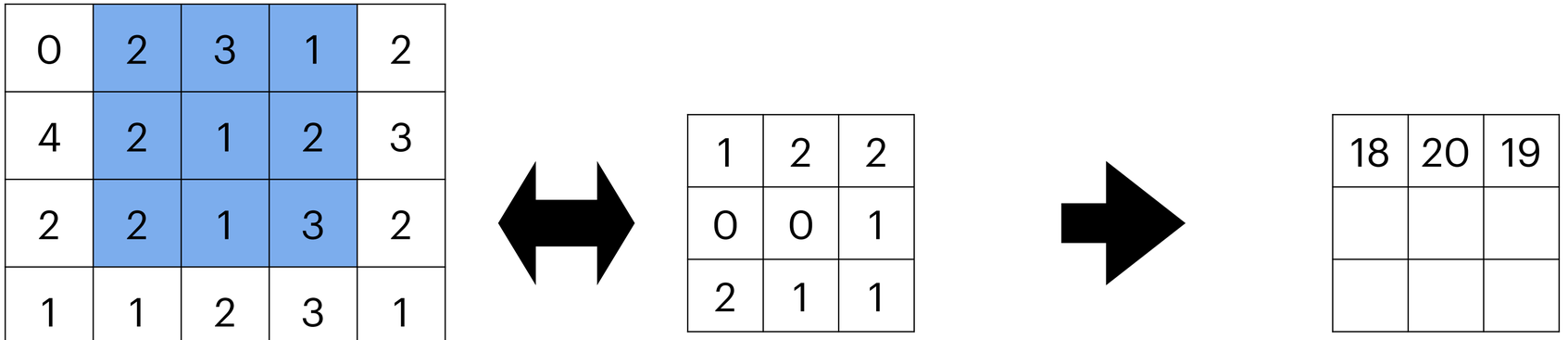
⟷

1	2	2
0	0	1
2	1	1

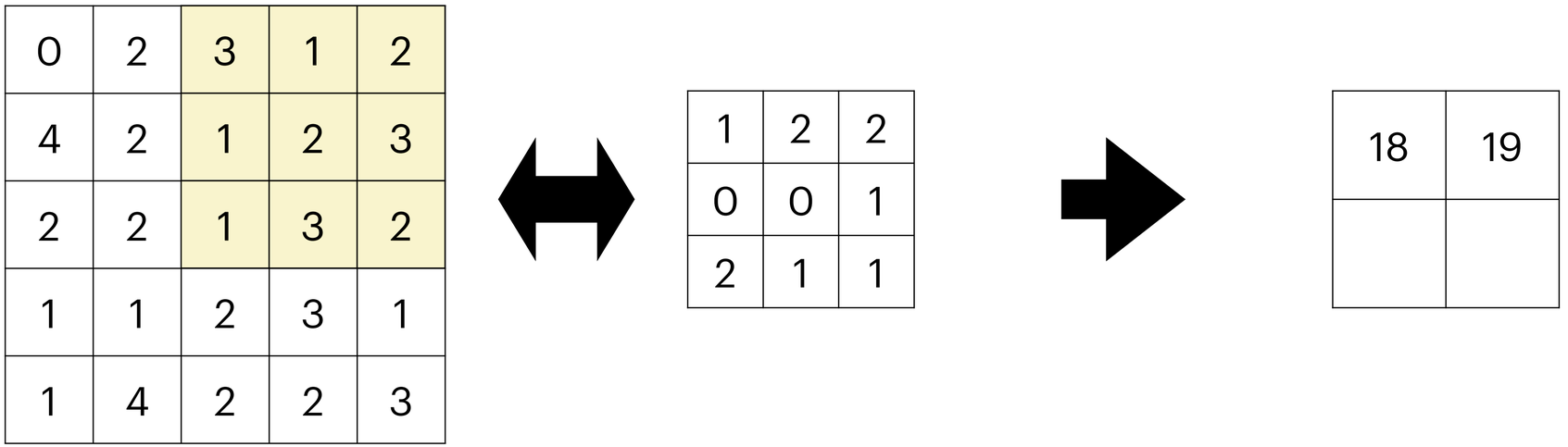
```
model.add(Conv2D(filters=32,kernel_size=3,strides=1,
                 padding='same',input_shape=(28,28,1),activation='relu'))
```

strides = カーネルをずらす幅

strides = 1



strides = 2

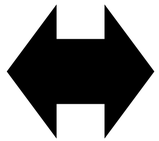


```
model.add(Conv2D(filters=32,kernel_size=3,strides=1,
          padding='same',input_shape=(28,28,1),activation='relu'))
```

padding = データの端をどう扱うか

0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

入力層 (6 × 6)



1	2	2
0	0	1
2	1	1

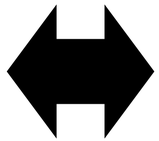
カーネル (3× 3)

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

padding = データの端をどう扱うか

0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

入力層 (6 × 6)



1	2	2
0	0	1
2	1	1

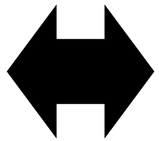
カーネル (3× 3)

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

padding = データの端をどう扱うか

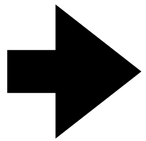
0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

入力層 (6 × 6)



1	2	2
0	0	1
2	1	1

カーネル (3× 3)



18	22	19	20
16	18	21	18
18	25	21	19
15	17	22	16

特徴マップ(4×4)

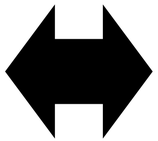
そのままだと特徴マップのサイズは入力層より小さくなる

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

入力データの周りを0で埋めてサイズを同じにする

0	0	0	0	0	0	0	0
0	0	2	3	1	2	3	0
0	4	2	1	2	3	0	0
0	2	2	1	3	2	1	0
0	1	1	2	3	1	2	0
0	1	4	2	2	3	1	0
0	2	0	2	1	4	0	0
0	0	0	0	0	0	0	0

入力層 (6 × 6)



1	2	2
0	0	1
2	1	1

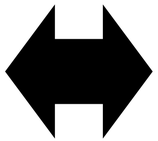
カーネル (3× 3)

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

入力データの周りを0で埋めてサイズを同じにする

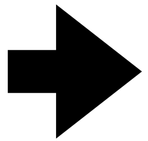
0	0	0	0	0	0	0	0
0	0	2	3	1	2	3	0
0	4	2	1	2	3	0	0
0	2	2	1	3	2	1	0
0	1	1	2	3	1	2	0
0	1	4	2	2	3	1	0
0	2	0	2	1	4	0	0
0	0	0	0	0	0	0	0

入力層 (6 × 6)



1	2	2
0	0	1
2	1	1

カーネル (3× 3)

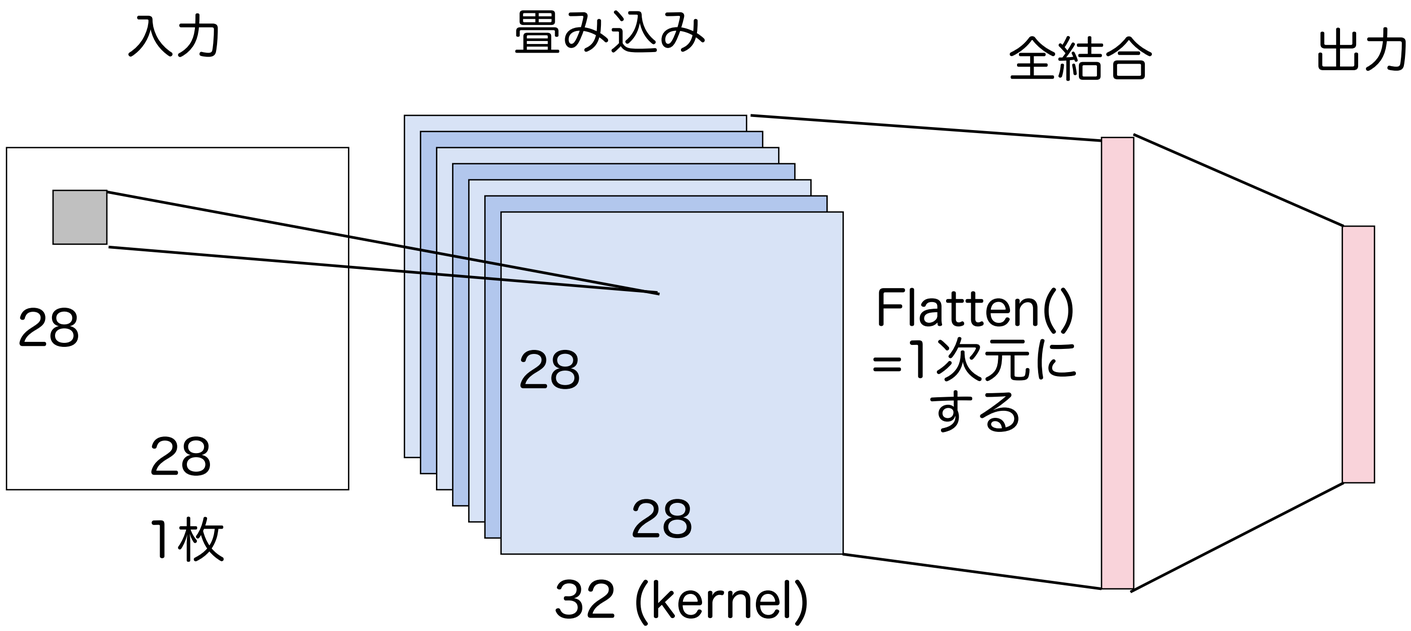


8	14	8	9	10	6
10	18	22	19	20	13
16	16	18	21	18	7
14	18	25	21	19	11
10	15	17	22	16	13
10	15	13	16	10	5

特徴マップ(6×6)

CNN

畳み込み層とプーリング層が繰り返されるニューラルネットワーク



Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 28, 28, 32)	320
flatten_2 (Flatten)	(None, 25088)	0
dropout_2 (Dropout)	(None, 25088)	0
dense_8 (Dense)	(None, 10)	250890

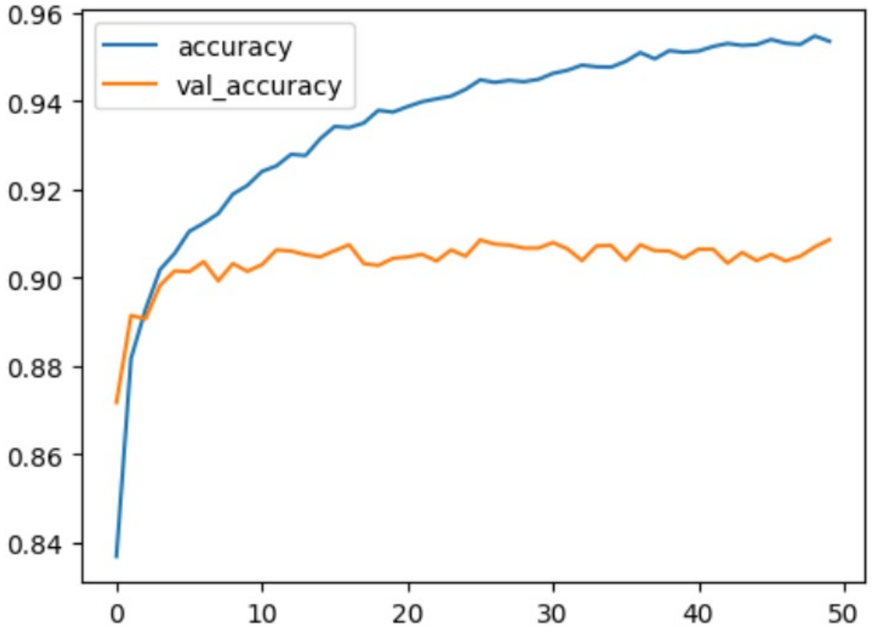
Total params: 251,210
Trainable params: 251,210
Non-trainable params: 0

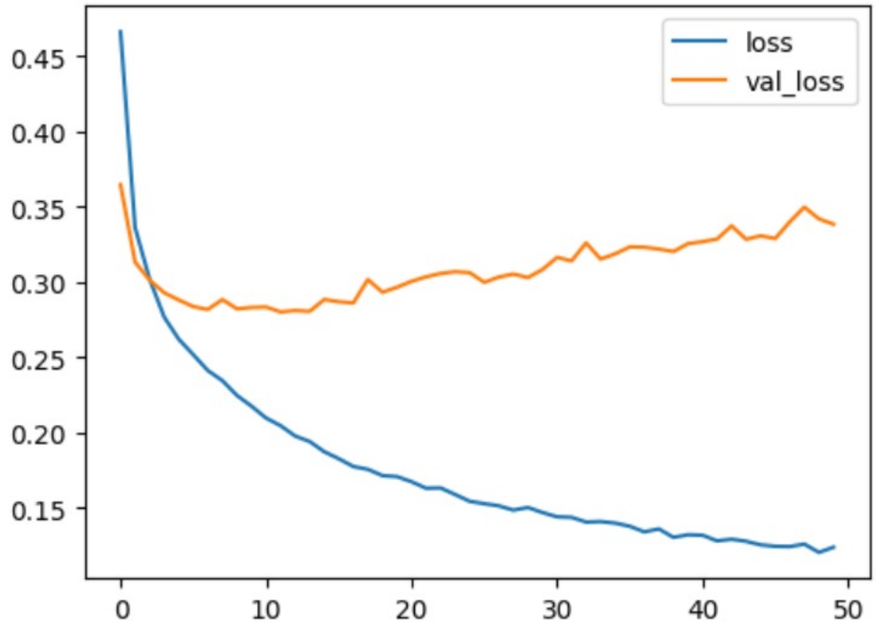
(3×3 + 1) × 32 = 320

(28×28×32 + 1) × 10 = 250890

一次元にせず、そのまま2次元の配列のまま入力する

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(result.history['accuracy'], label='accuracy')
plt.plot(result.history['val_accuracy'], label='val_accuracy')
plt.legend()
plt.subplot(1,2,2)
plt.plot(result.history['loss'], label='loss)')
plt.plot(result.history['val_loss'], label='val_loss)')
plt.legend()
plt.show()
```

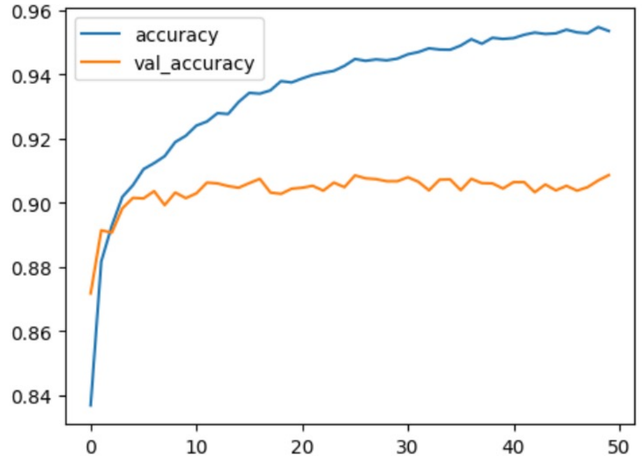


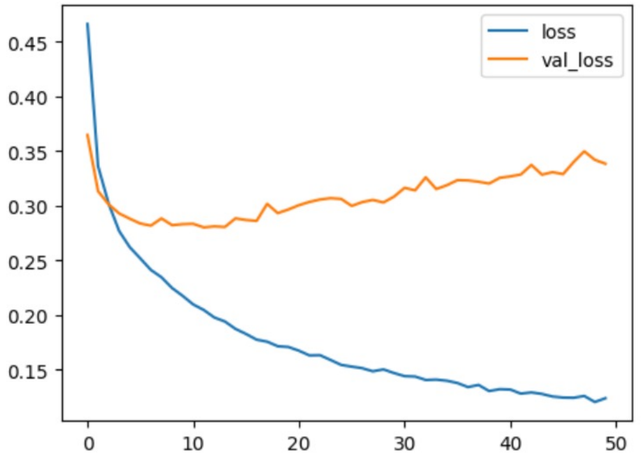


```
score = model.evaluate(x_test, y_test)
print('test loss:',score[0])
print('test accuracy:',score[1])
```

test loss: 0.34445714950561523

test accuracy: 0.90319997072219985

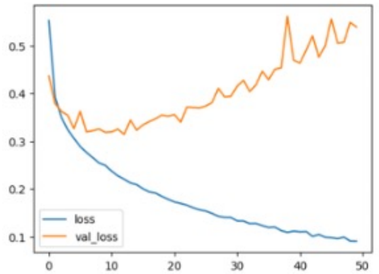


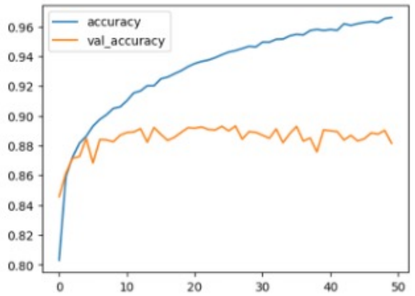


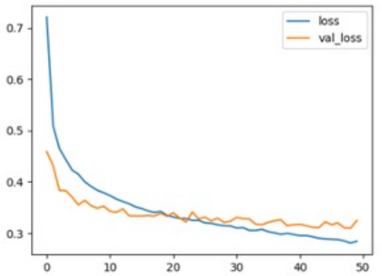
既にMLPよりも精度が良いことが分かる

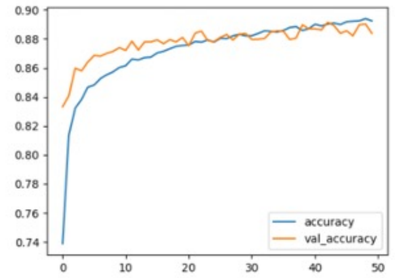
MLP（前回のスライド）

Dropoutを加えると過学習を抑制できる









Test loss: 0.5994181036949158
Test accuracy 0.8773999810218811

Test loss: 0.3330557644367218
Test accuracy: 0.8855000138282776

実際は過学習を抑えつつ精度をどれだけ上げられるかを検討する

畳み込み層の追加

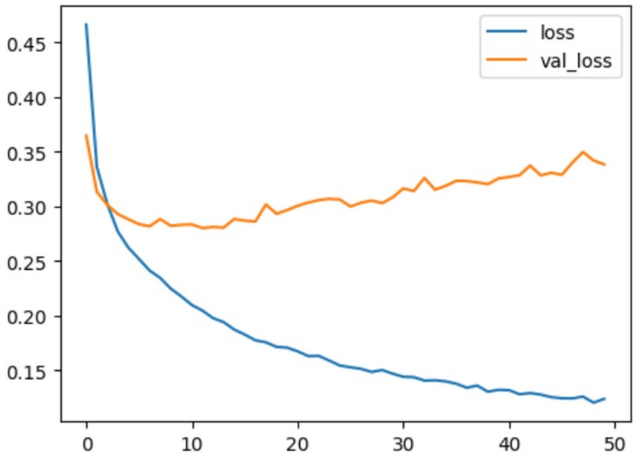
```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Conv2D, Flatten

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



model.compile(loss='categorical_crossentropy',
                              optimizer='Adam',metrics=['accuracy'])
model.summary()
result = model.fit(x_train,y_train,epochs=50,batch_size=64,validation_split=0.2)
```

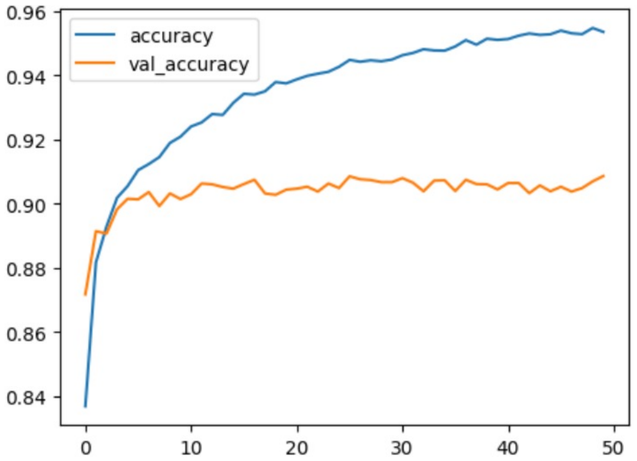
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 28, 28, 64)        18496
_________________________________________________________________
flatten_3 (Flatten)          (None, 50176)             0
_________________________________________________________________
dropout_3 (Dropout)          (None, 50176)             0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                501770
=================================================================
Total params: 520,586
Trainable params: 520,586
Non-trainable params: 0
_________________________________________________________________
```

畳み込み層の追加

test loss: 0.34445714950561523

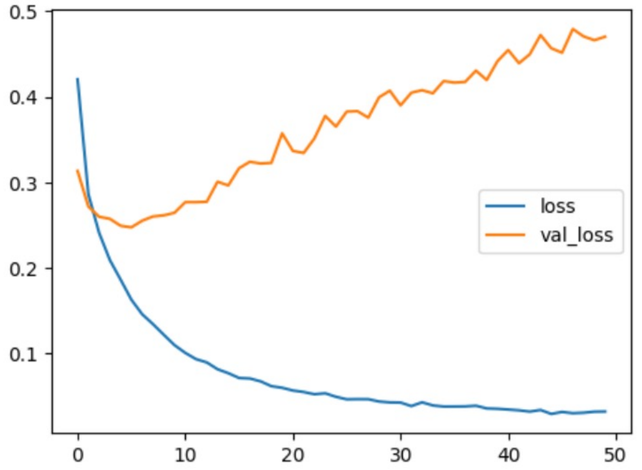
test accuracy: 0.9031999707221985

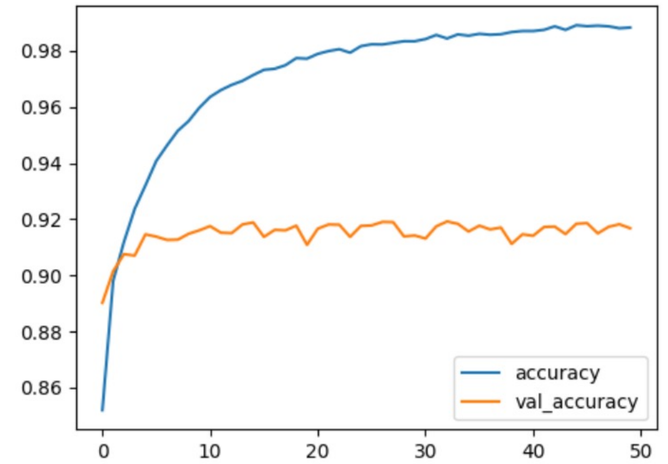




test loss: 0.4966644048690796

test accuracy: 0.9117000102996826





プーリング層の追加

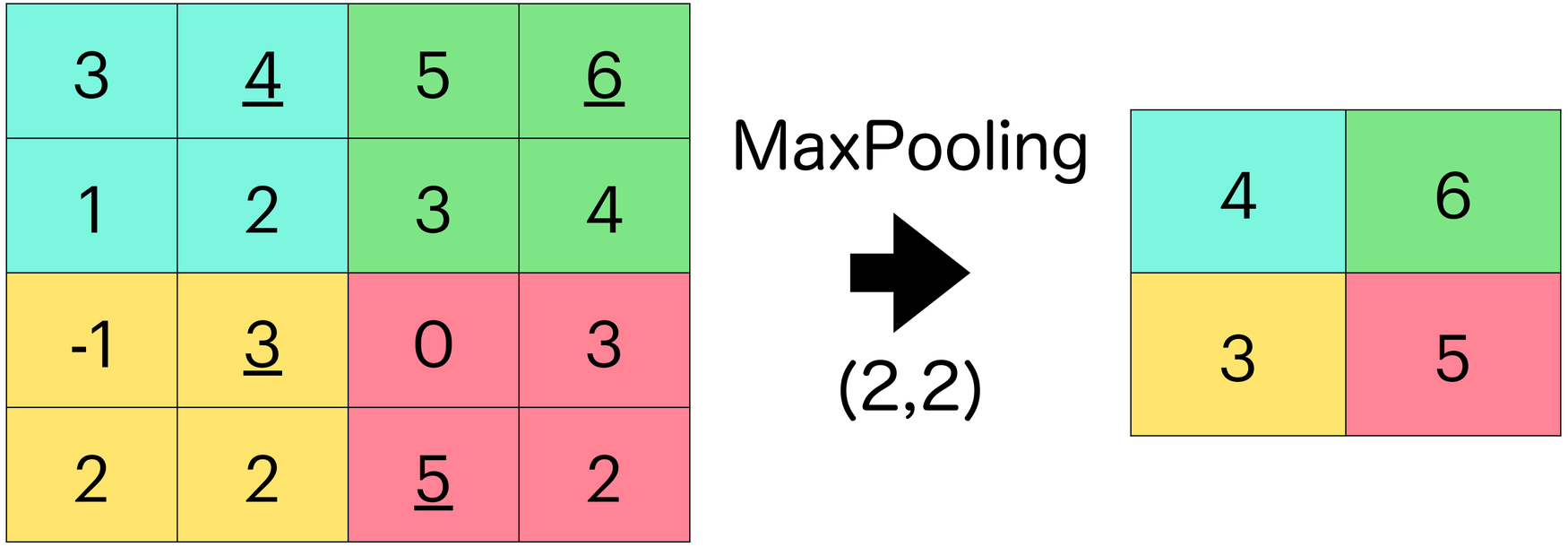
```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Conv2D, Flatten, MaxPooling2D

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()
result = model.fit(x_train,y_train,epochs=50,batch_size=64,validation_split=0.2)
```

```
_________________________________________________________________
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
_________________________________________________________________
```

プーリング層

データを縮小する方法

マックスプーリング：入力データを小さな領域に分割し、各領域の最大値をとってくることで、データを縮小する。



プーリング前の特徴マップ

プーリング後の特徴マップ

プーリング層

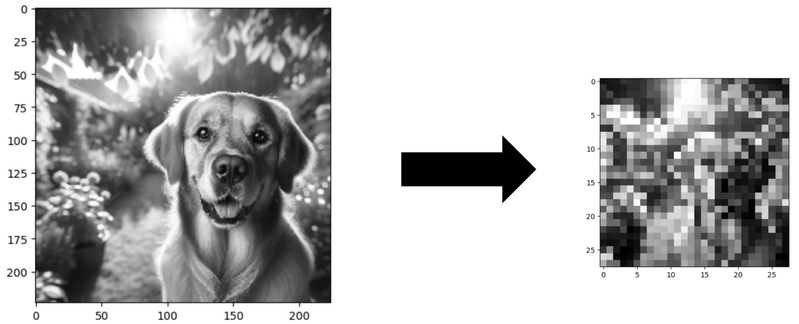
Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 28, 28, 32)	320
conv2d_4 (Conv2D)	(None, 28, 28, 64)	18496
flatten_3 (Flatten)	(None, 50176)	0
dropout_3 (Dropout)	(None, 50176)	0
dense_9 (Dense)	(None, 10)	501770

Total params: 520,586
Trainable params: 520,586
Non-trainable params: 0

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

プーリング層で(28,28)が(14,14)になっている



プーリング層

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 28, 28, 64)        18496
_________________________________________________________________
flatten_3 (Flatten)          (None, 50176)             0
_________________________________________________________________
dropout_3 (Dropout)          (None, 50176)             0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                501770
=================================================================
Total params: 520,586
Trainable params: 520,586
Non-trainable params: 0
_________________________________________________________________
```

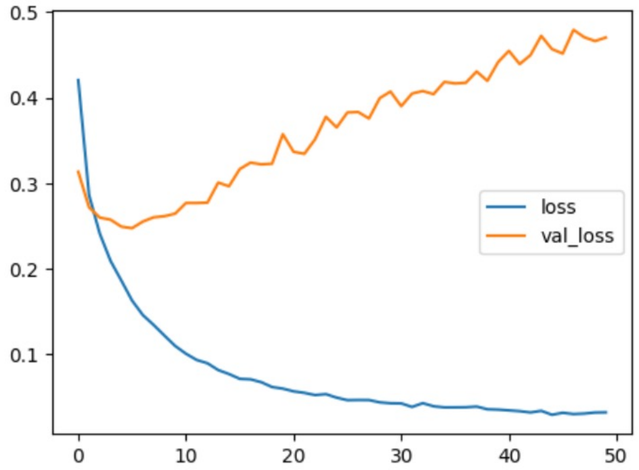
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
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
_________________________________________________________________
```

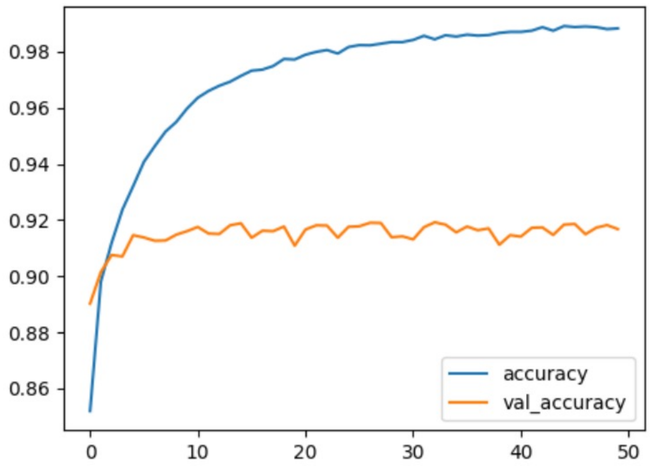
プーリング層で(28,28)が(14,14)になっている

パラメーターの数も1/4になっている

test loss: 0.4966644048690796

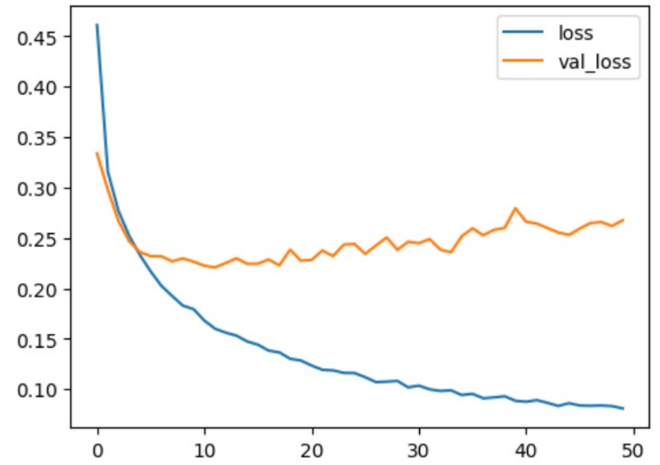
test accuracy: 0.9117000102996826

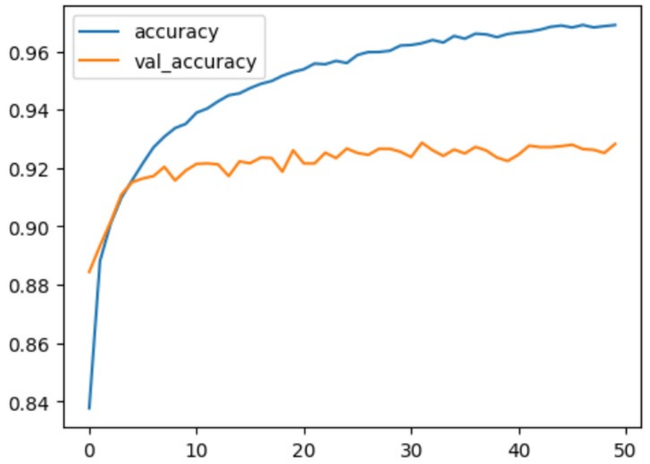




Test loss: 0.2698012888431549

Test accuracy: 0.9247000217437744





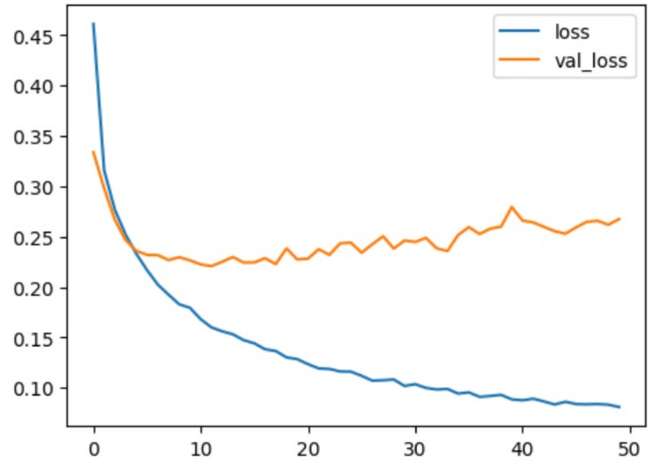
畳み込み、畳み込み、プーリング、で繰り返すことが多い

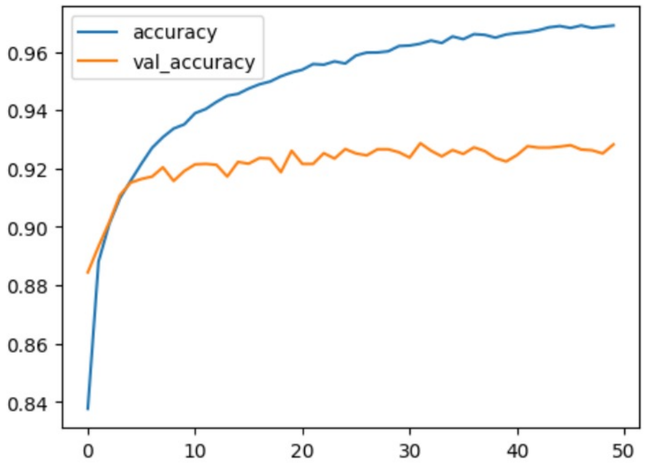
```
model = Sequential()

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(Dropout(0.5))

model.add(Conv2D(filters=64,kernel_size=3,strides=1,
                 padding='same',activation='relu'))
model.add(Conv2D(filters=128,kernel_size=3,strides=1,padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.5))

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()
```

```
result = model.fit(x_train, y_train, epochs = 50, batch_size = 64, validation_split=0.2, shuffle=True)
```

Test loss: 0.2698012888431549

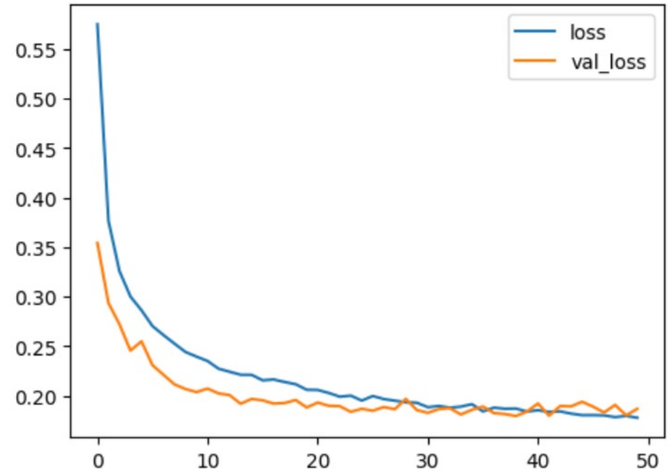
Test accuracy: 0.9247000217437744

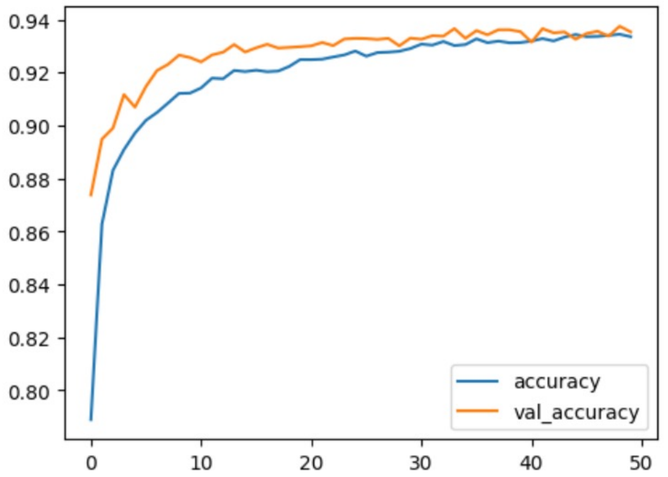




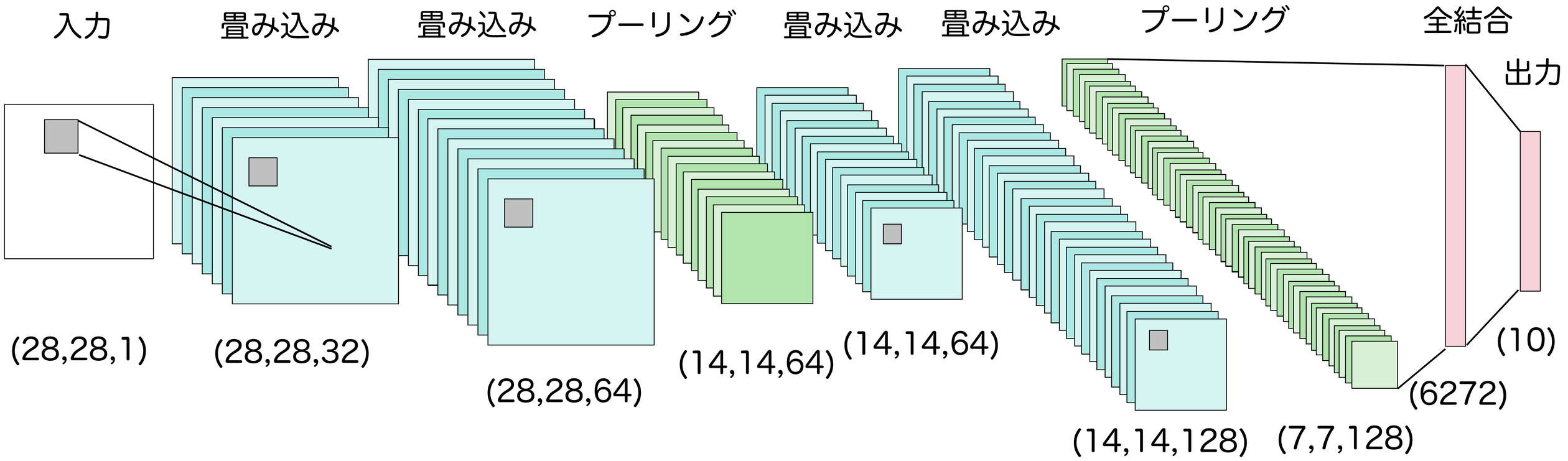
test loss: 0.20224225521087646

test accuracy: 0.92900000200271606

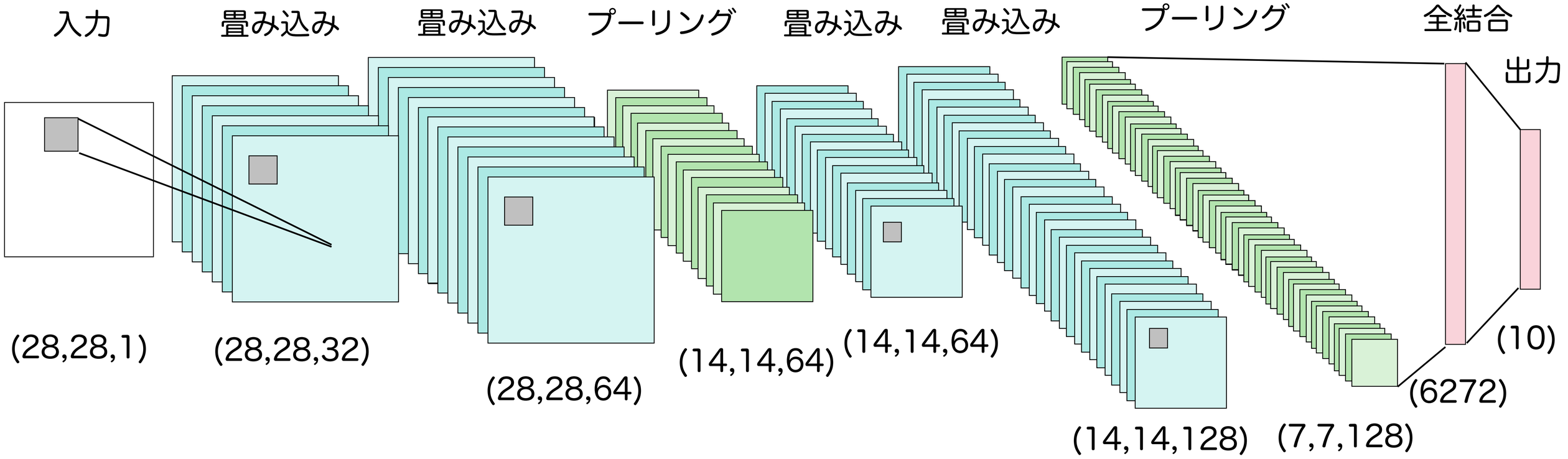




```
model.add(Conv2D(filters=32,kernel_size=3,strides=1, padding='same',input_shape=(28,28,1),activation='relu'))
model.add(Conv2D(filters=64,kernel_size=3,strides=1,padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.5))

model.add(Conv2D(filters=64,kernel_size=3,strides=1, padding='same',activation='relu'))
model.add(Conv2D(filters=128,kernel_size=3,strides=1,padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.5))

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



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



課題

・WebClassにある”kadai6.ipynb”をやってみましょう
・実行したら”学籍番号_名前_6.ipynb”という名前で保存して提出して下さい。

締め切りは2週間後の11/30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

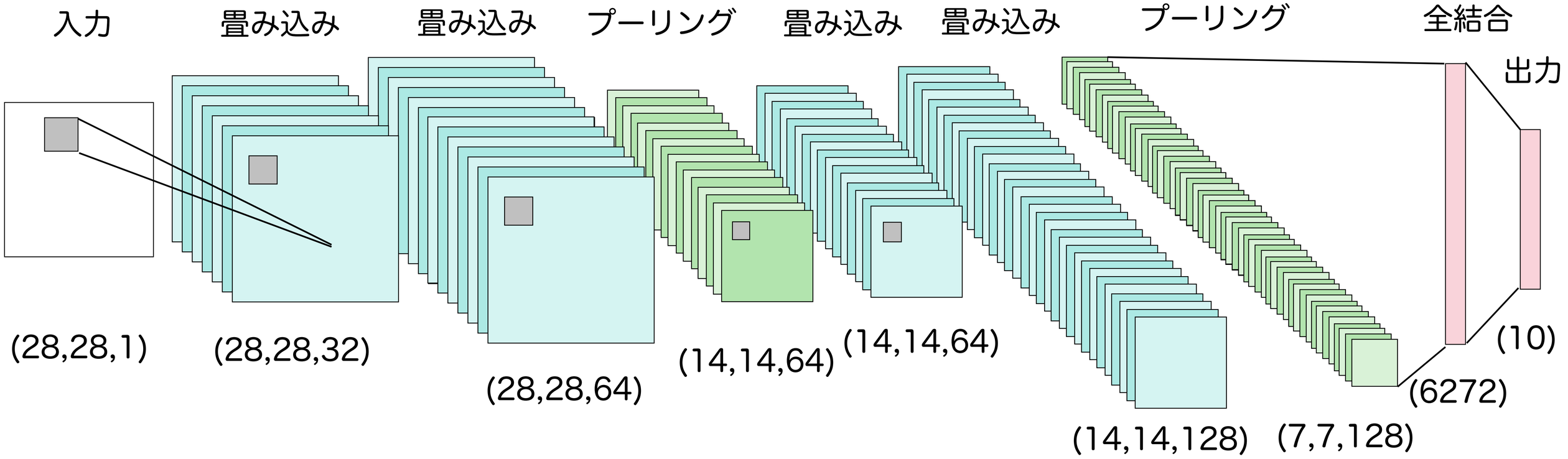
医療とAI・ビッグデータ応用

データ拡張

統合教育機構
須藤毅顕

前回の復習

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

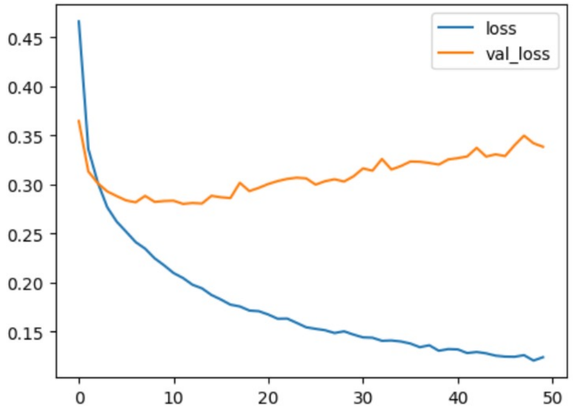


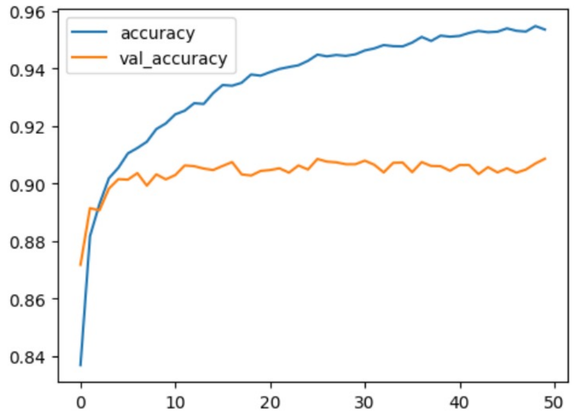
前回までの精度

畳み込み層1つ

test loss: 0.34445714950561523

test accuracy: 0.9031999707221

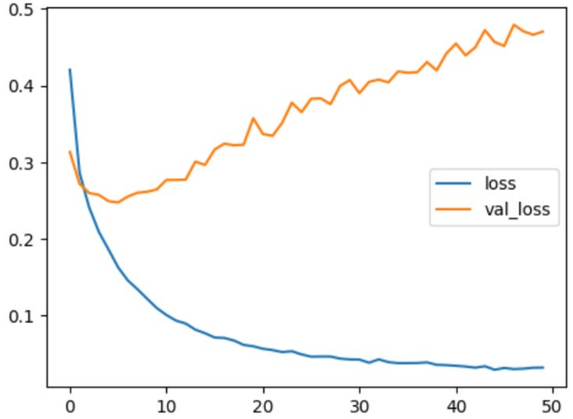


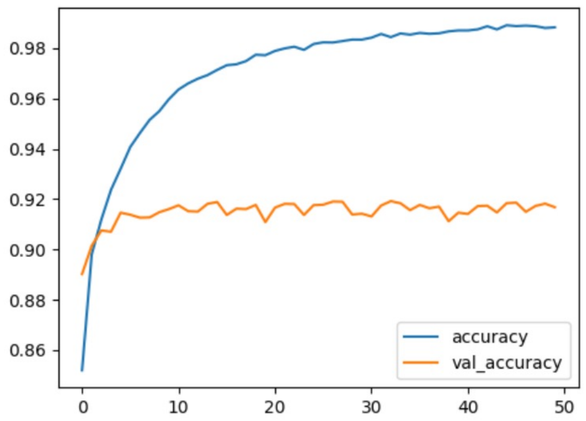


畳み込み層2つ

test loss: 0.4966644048690796

test accuracy: 0.911700010299

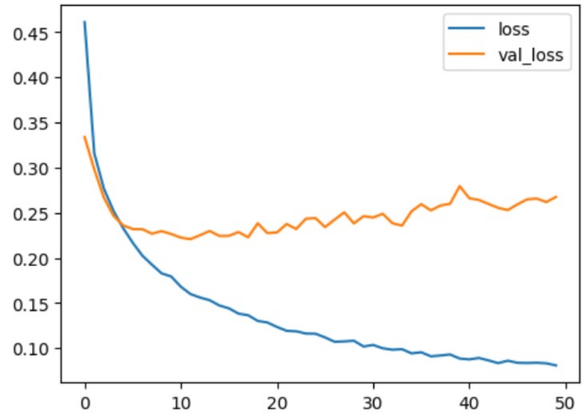


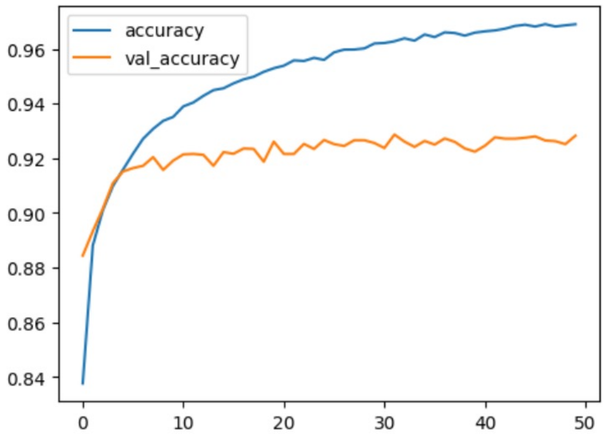


プーリング層追加

Test loss: 0.2698012888431549

Test accuracy: 0.924700021743

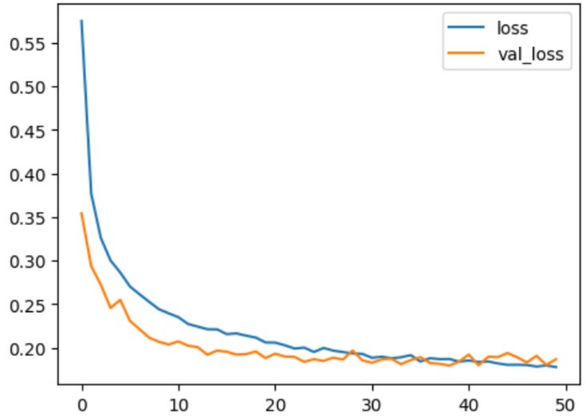


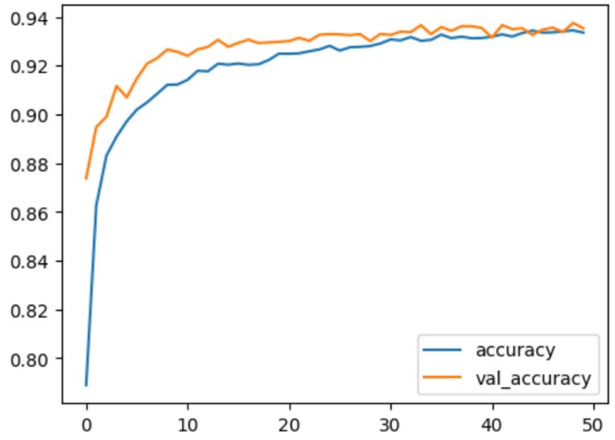


2回繰り返し

test loss: 0.20224225521087646

test accuracy: 0.92900002002716





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()
```

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)
```

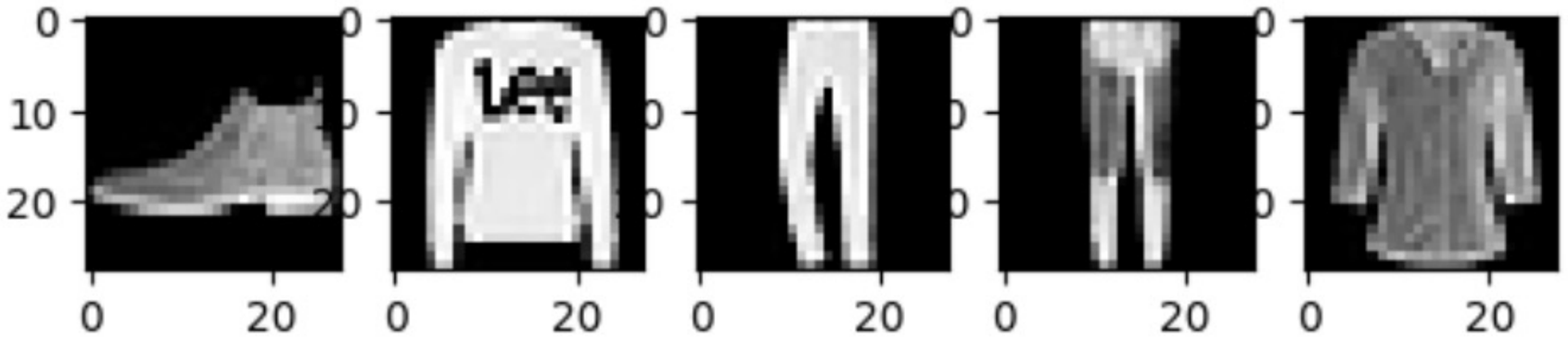
```
x_train = x_train.reshape(x_train.shape[0],28,28,1)/255
```

=

```
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()
```



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)

データ拡張の関数 ImageDataGenerator

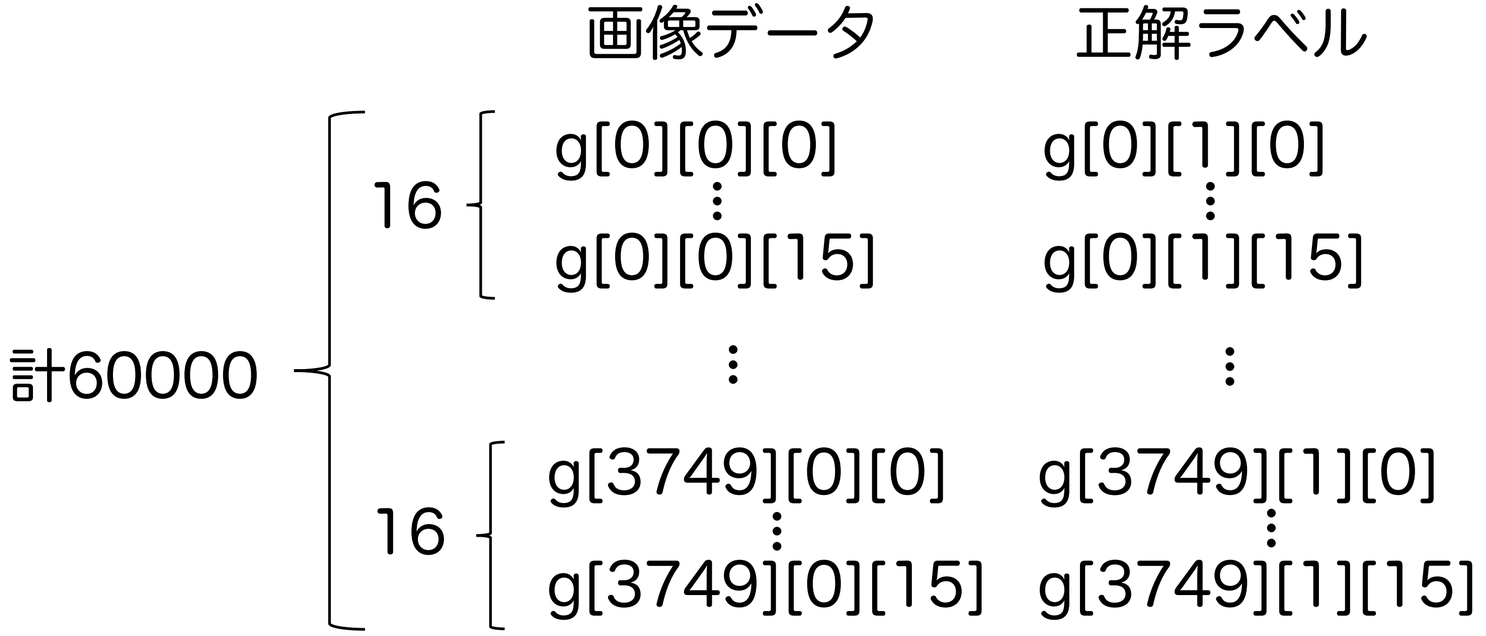
g[0][0][0] 1枚目の画像の配列データ

```
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)

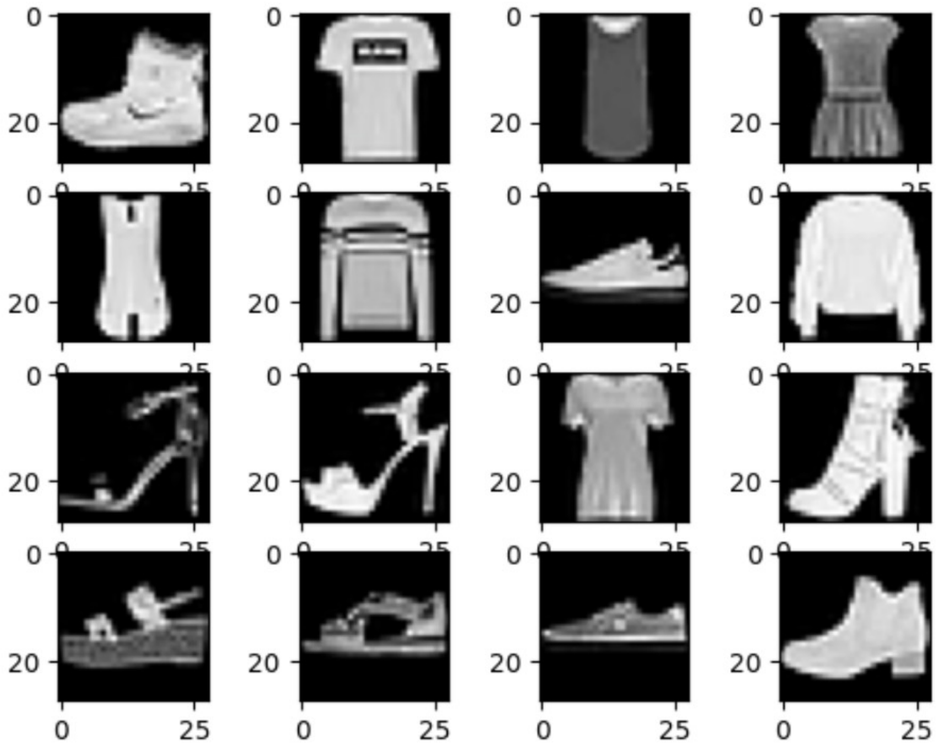
g[0][1][0] 1枚目の画像の正解ラベル (one-hot encoding)

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



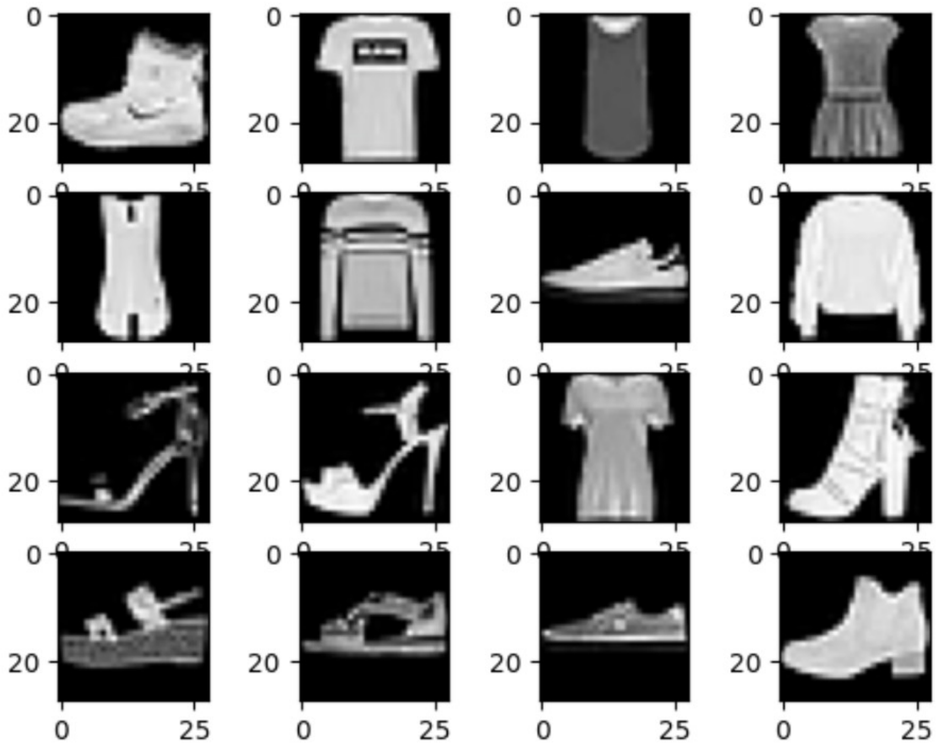
画像を16枚表示

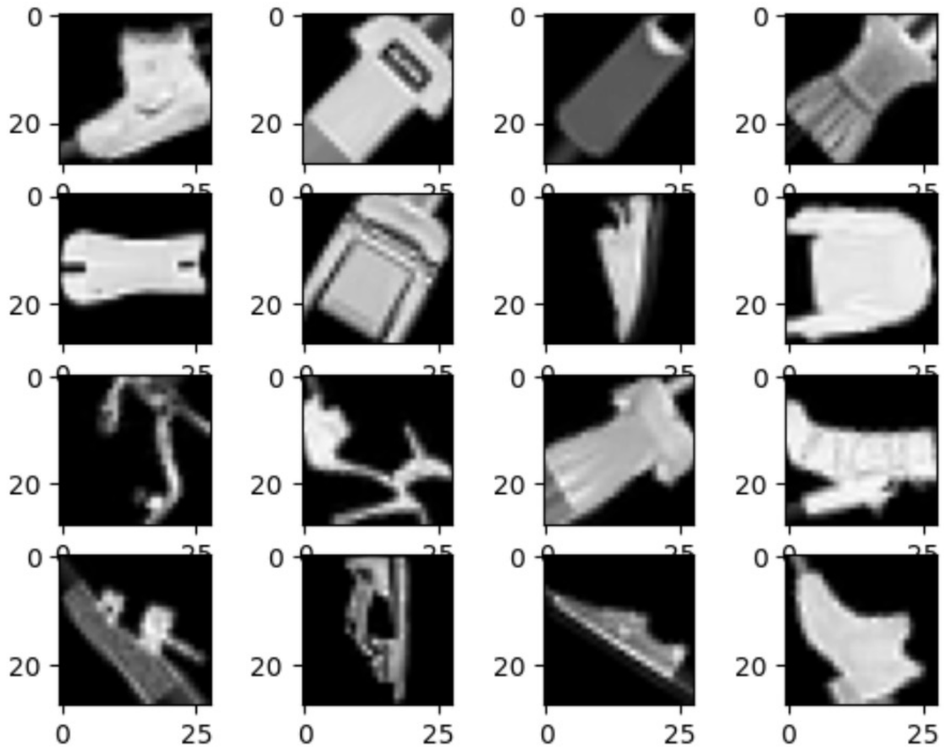
```
import matplotlib.pyplot as plt
for i in range(0,16):
    plt.subplot(4,4,i+1)
    plt.imshow(x_train[i],'gray')
plt.show()
```



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()
```





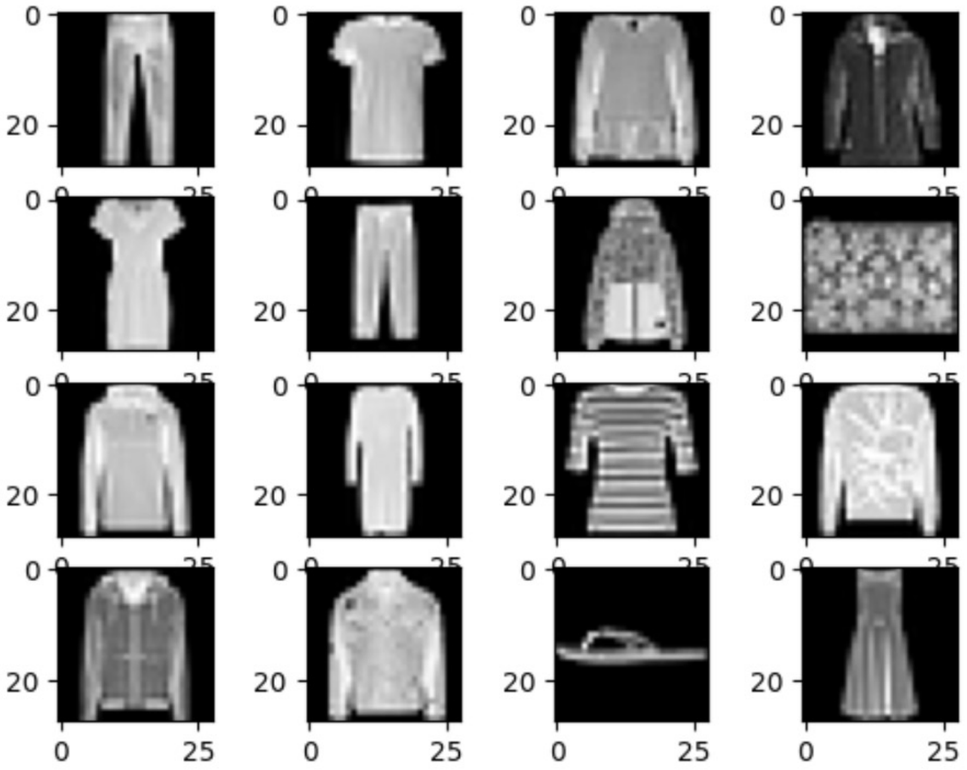
データ拡張の関数 ImageDataGenerator

```
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

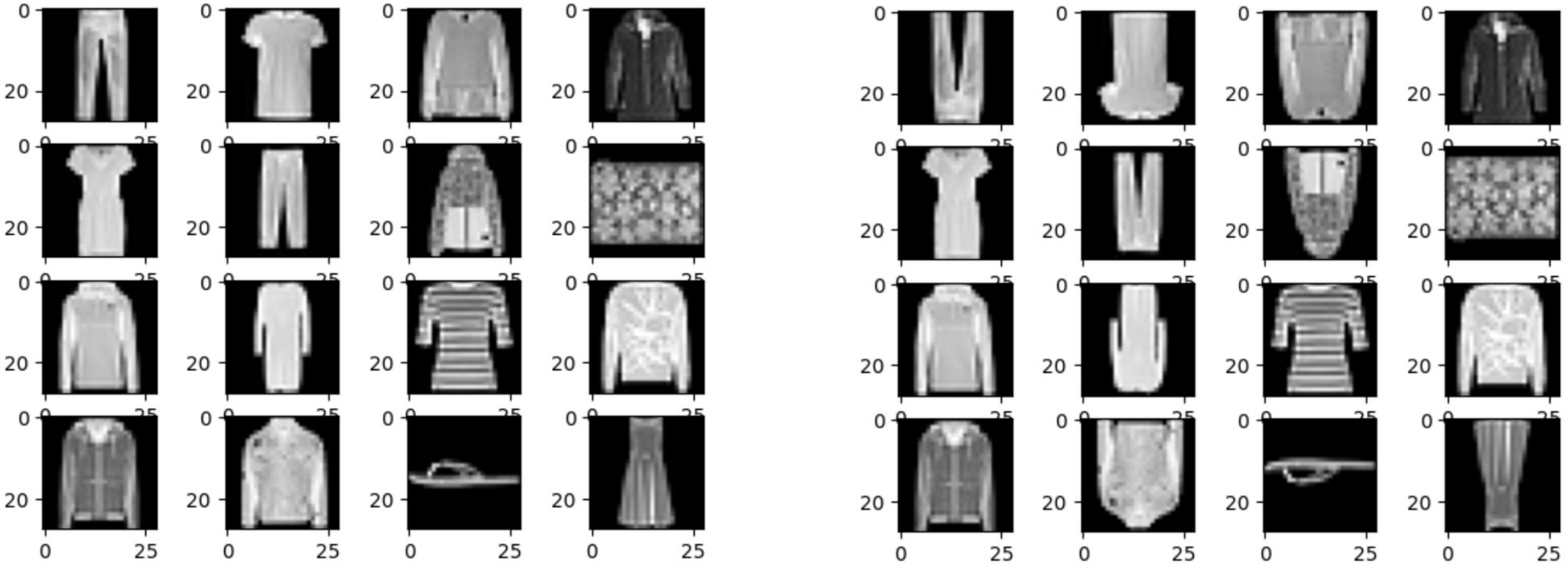
画像を16枚表示(次の16枚)

```
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

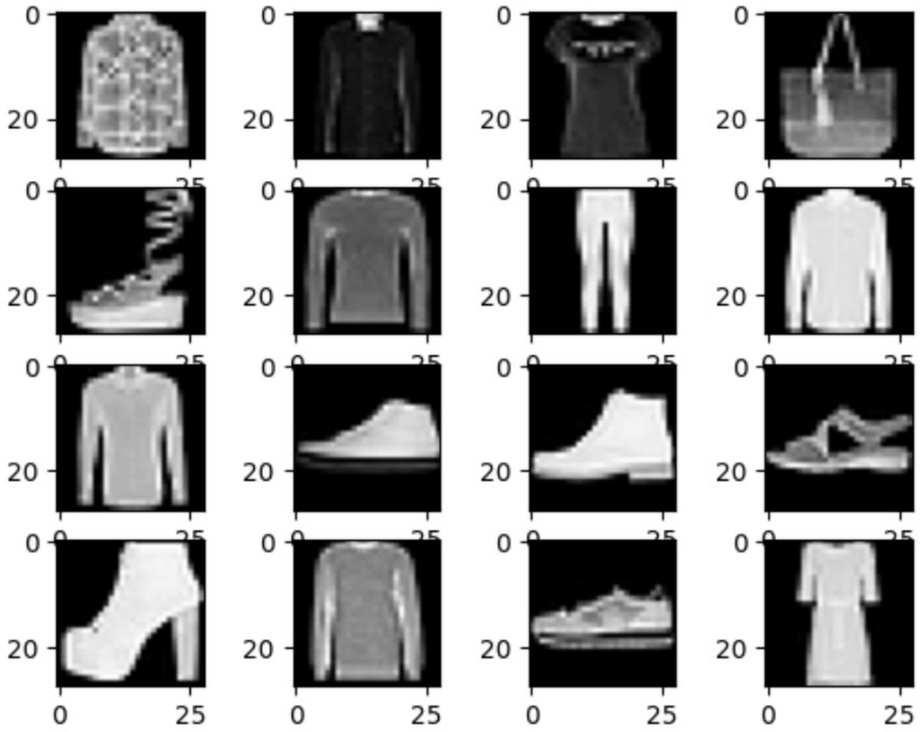
```
from keras.preprocessing.image import ImageDataGenerator

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

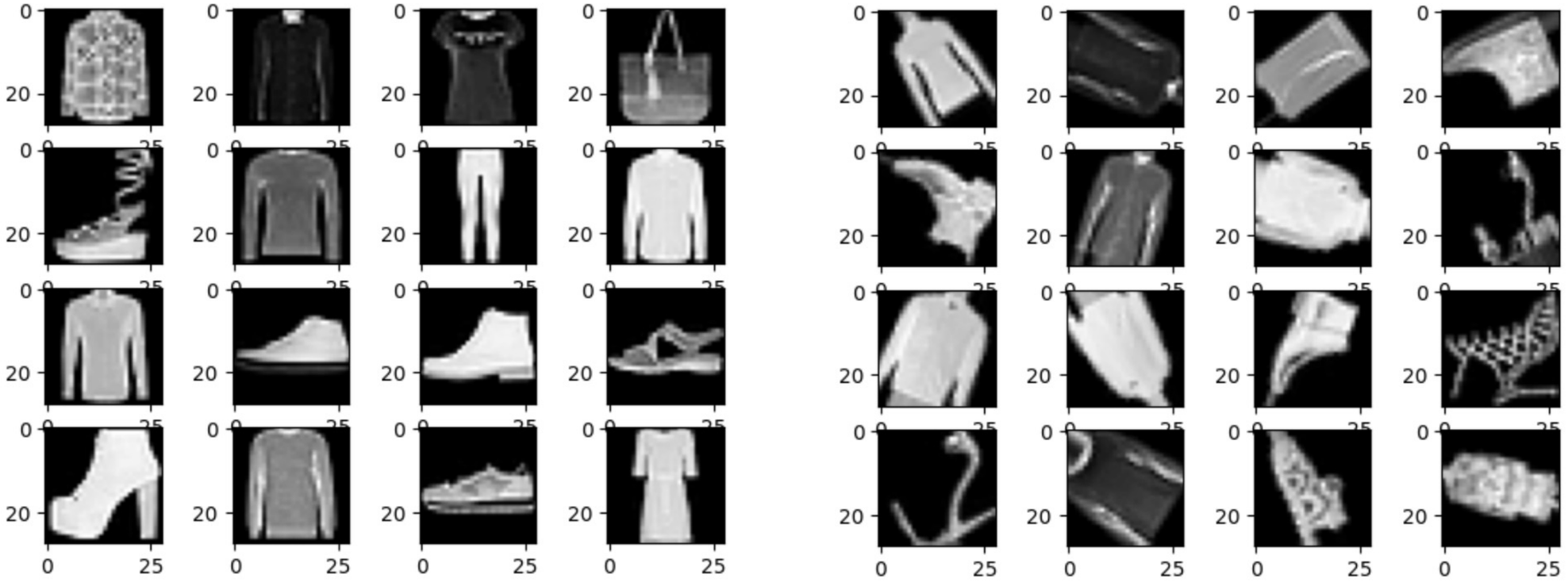
画像を16枚表示(次の16枚)

```
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[1][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)

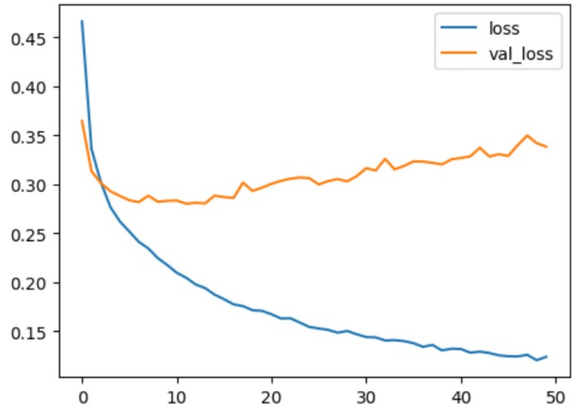
(x_train.shape[0]は48000、”//”は割り算の整数)

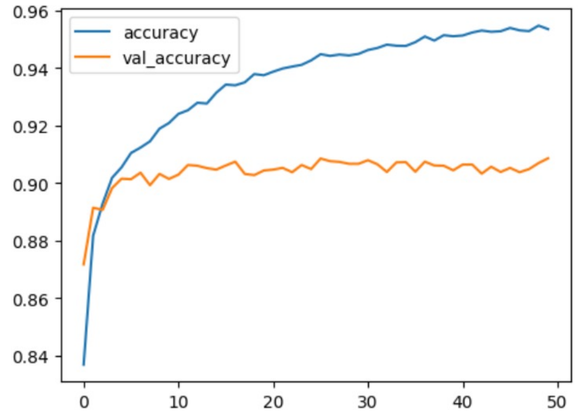
結果の比較

畳み込み層1つ

test loss: 0.34445714950561523

test accuracy: 0.9031999707221

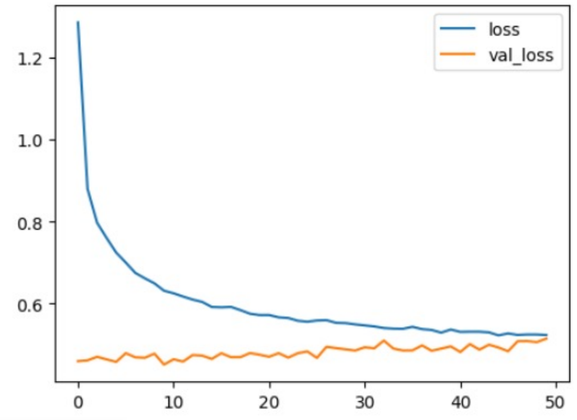


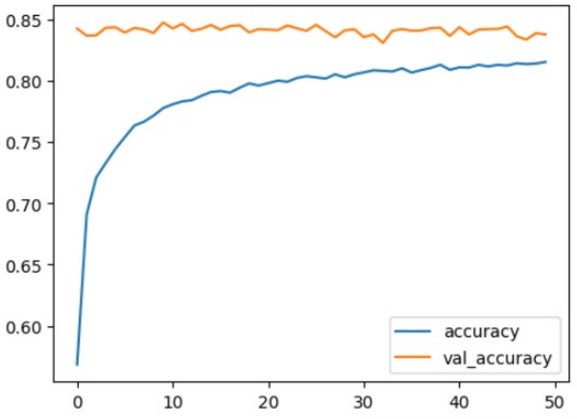


畳み込み層1つ(データ拡張)

test loss: 0.5629728436470032

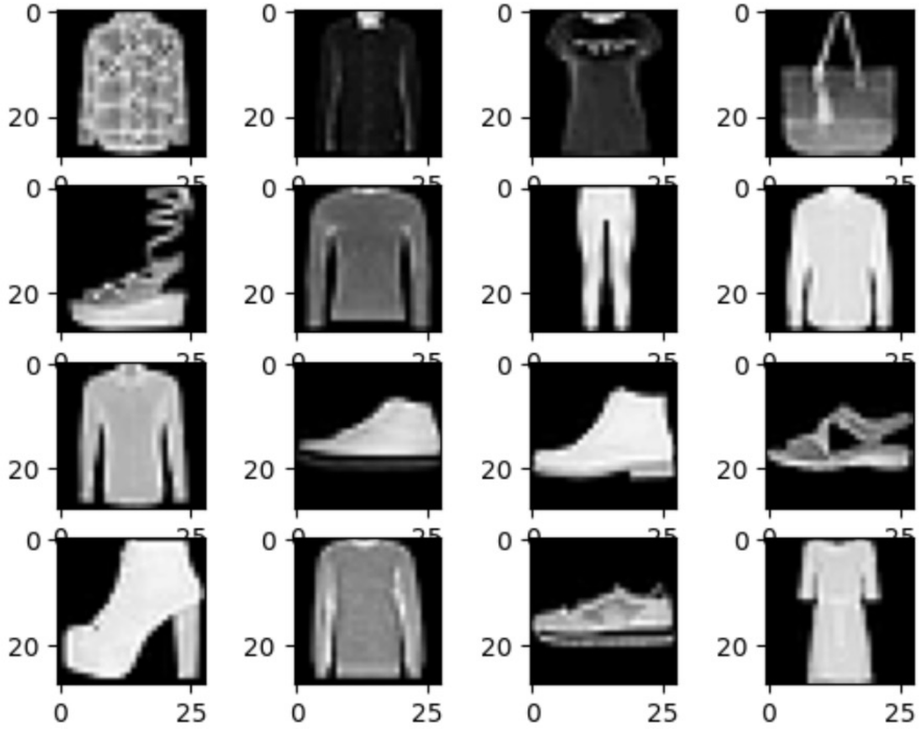
test accuracy: 0.8313000202178955



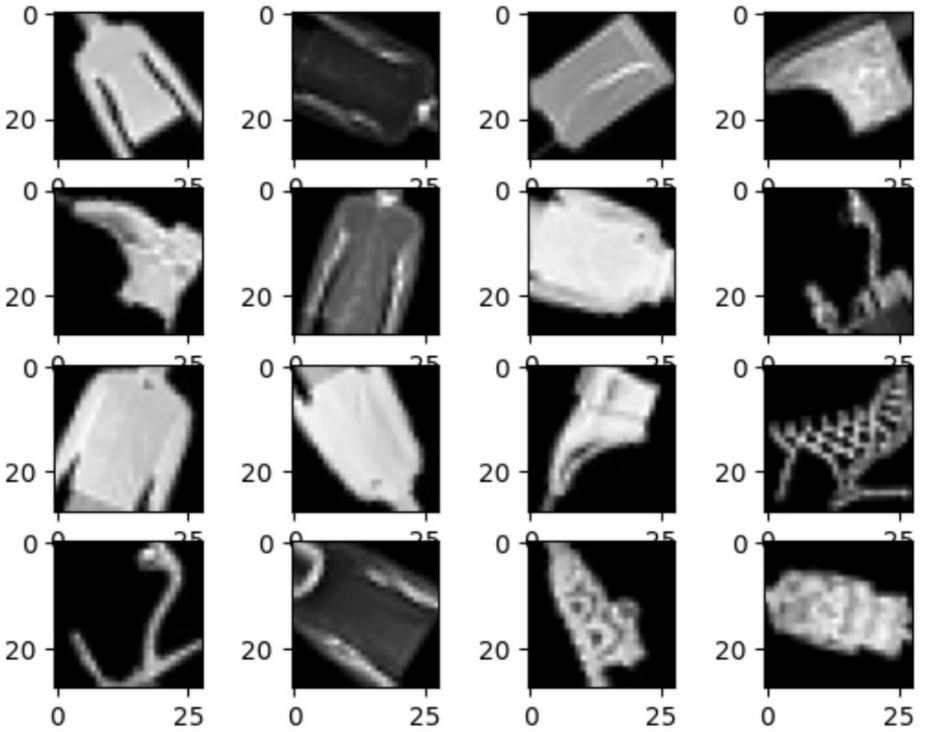


あれ？

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”kadai7.ipynb”をやってみましょう
・実行したら”学籍番号_名前_7.ipynb”という名前で保存して提出して下さい。

締め切りは2週間後の12/07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