

医療とAI・ビッグデータ応用

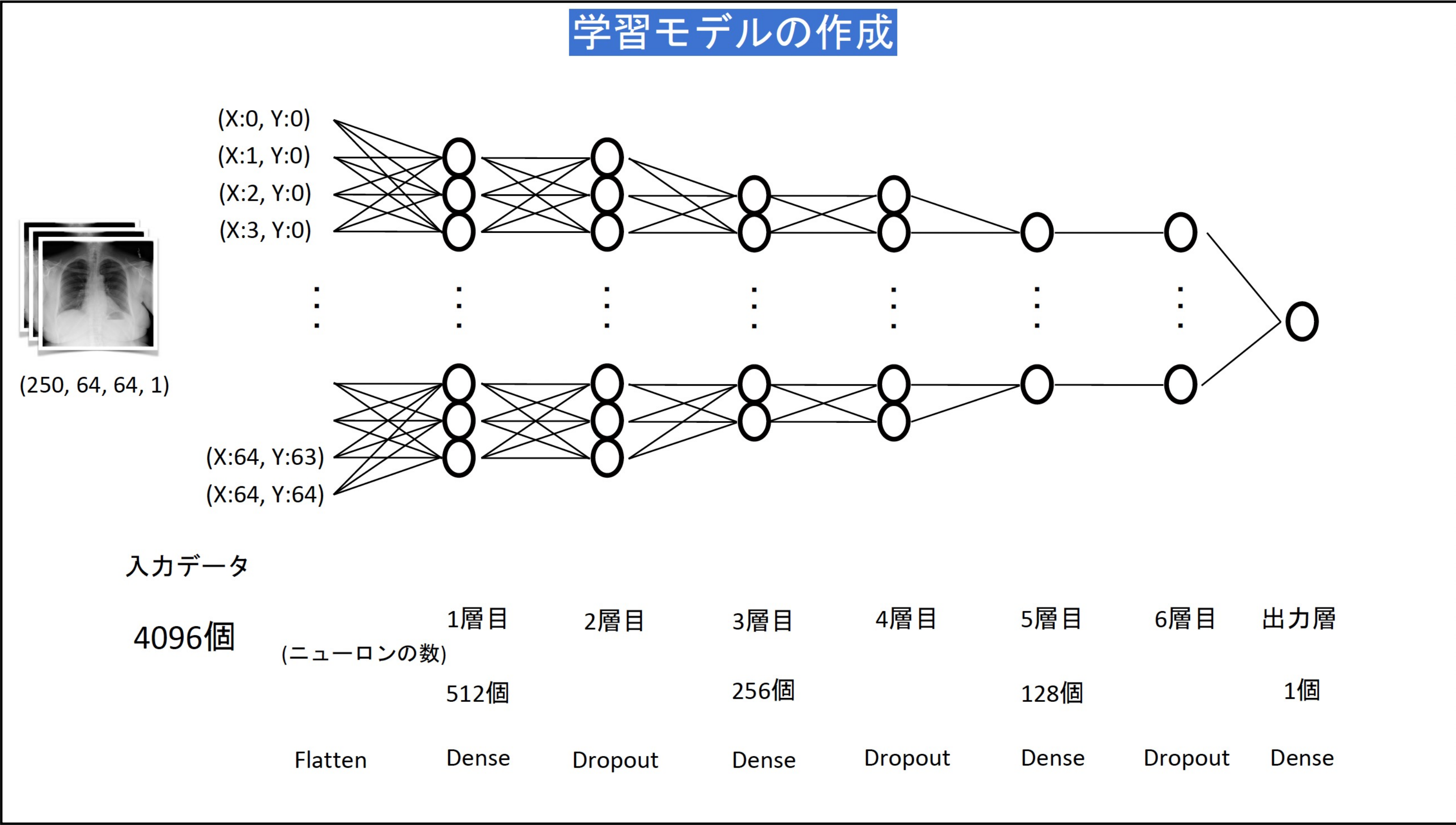
CNN

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

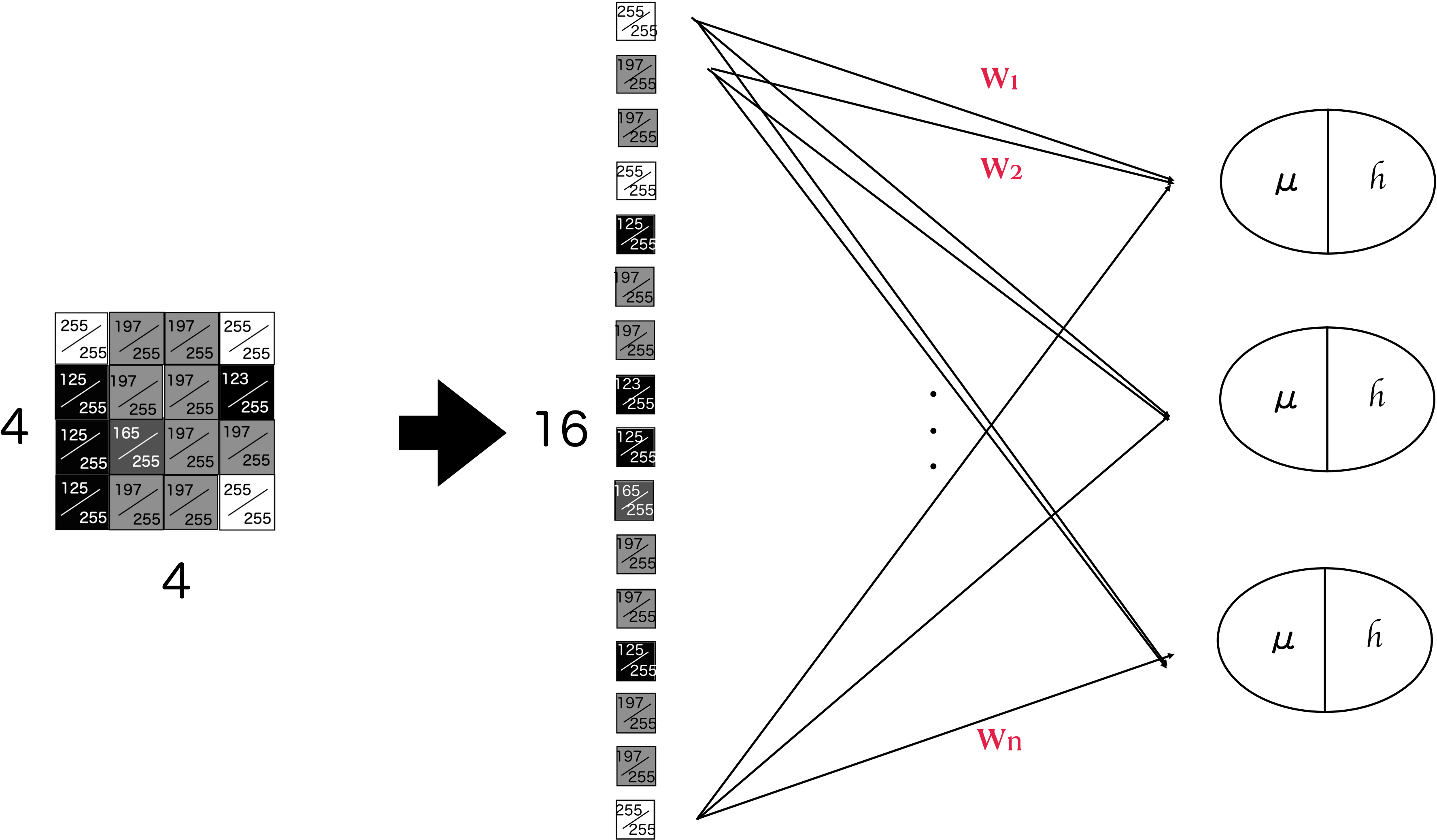
前回の深層学習はMLP



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

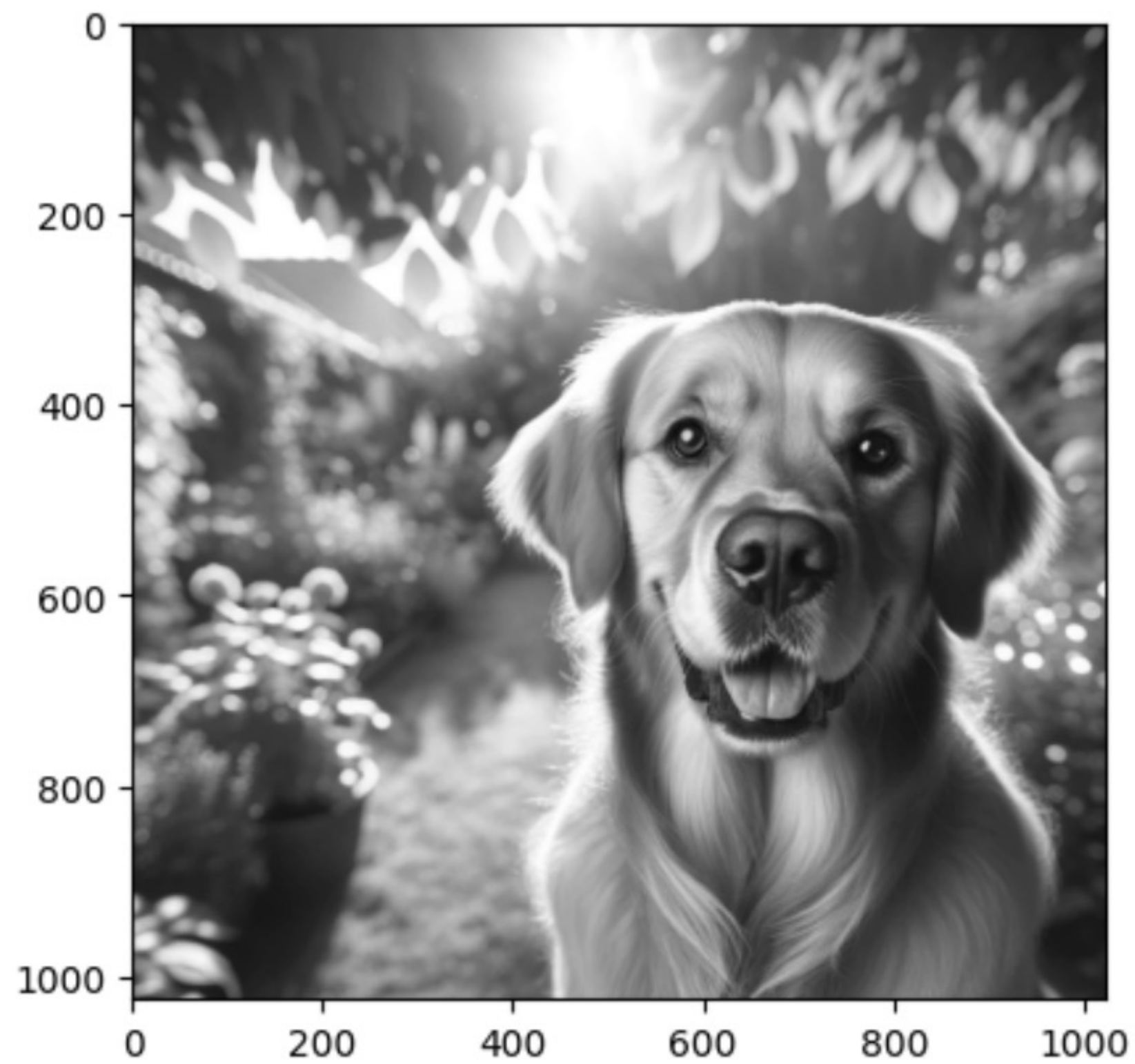
MLPでは画像サイズを1次元にして入力する→画像サイズ分の重みが存在



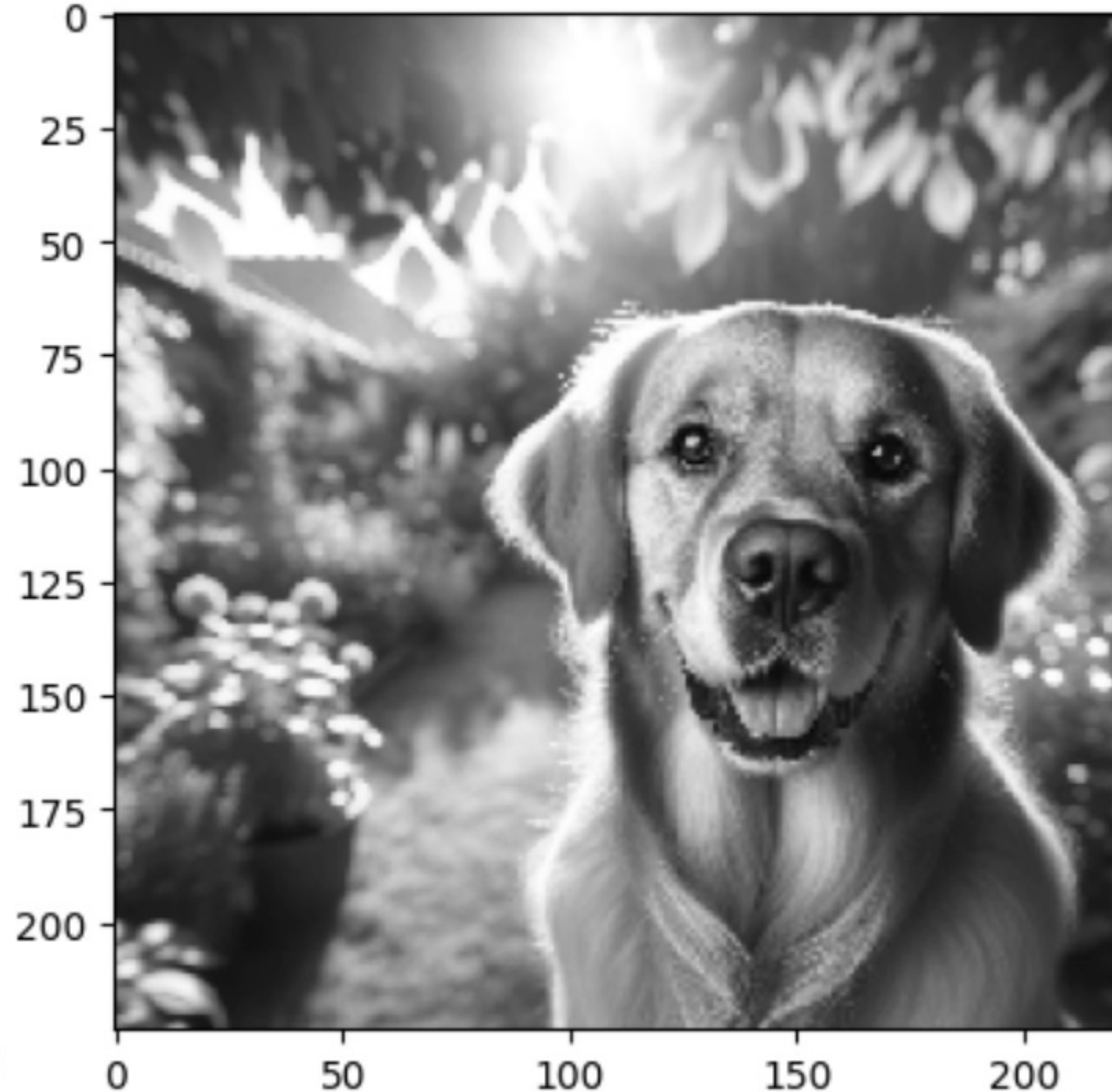
サイズが大きいほど調整する重みが増えてしまう

MLPの問題点1

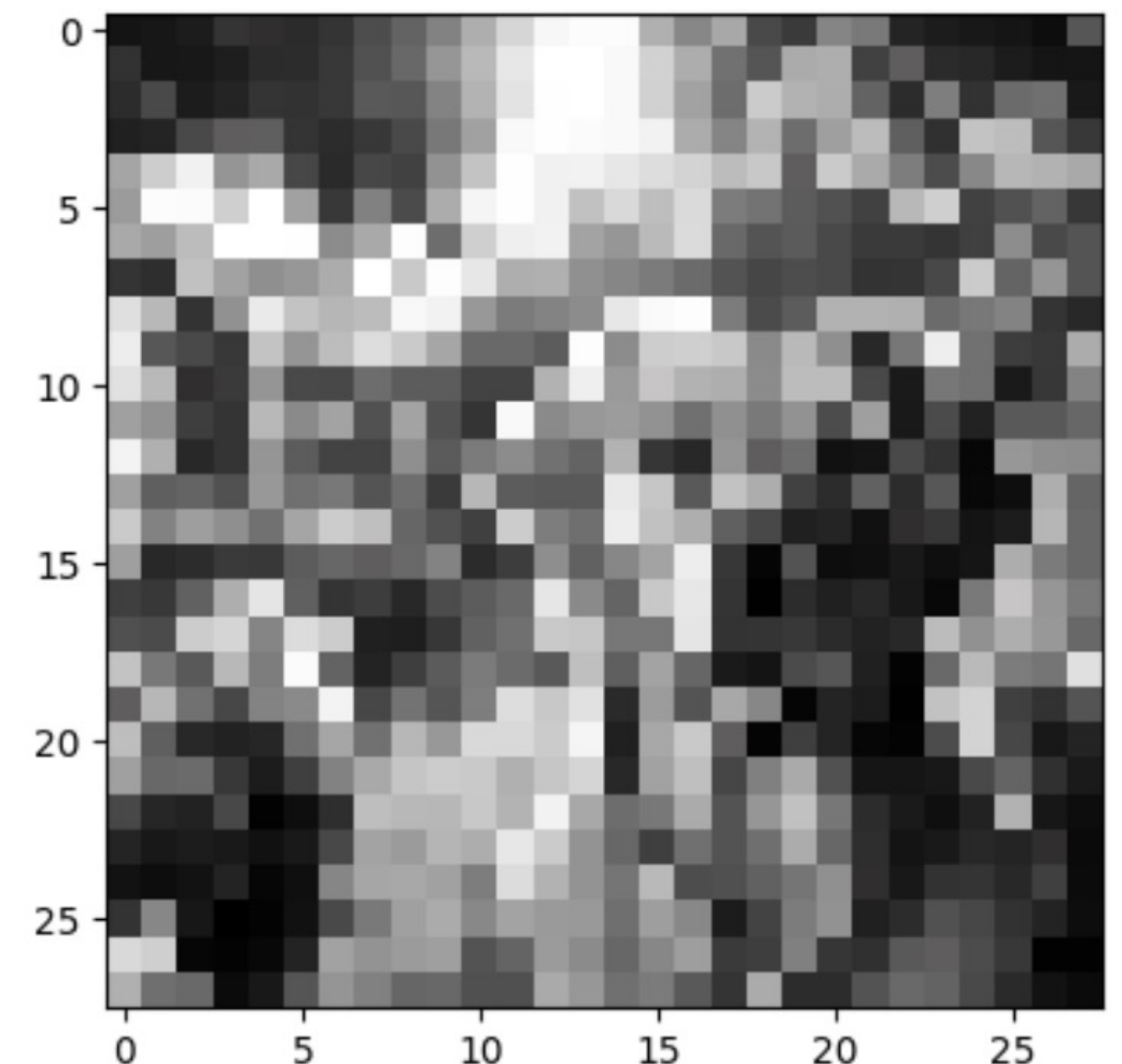
MLPでは画像サイズを1次元にして入力する→画像サイズ分の重みが存在



1024×1024



224×224

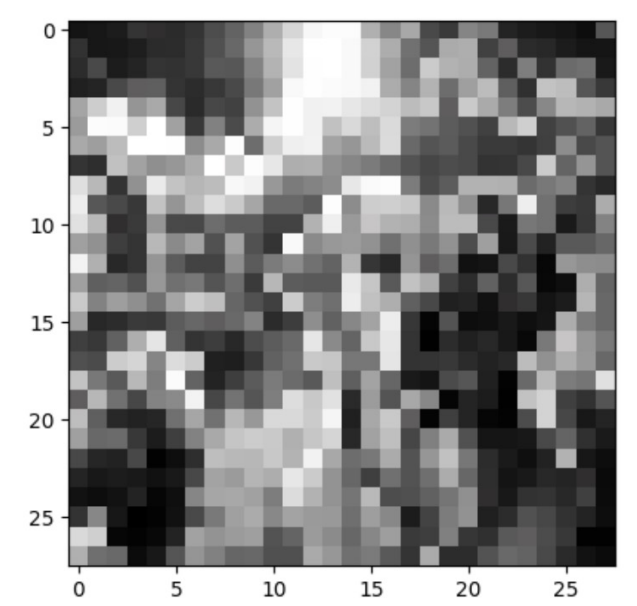


28×28

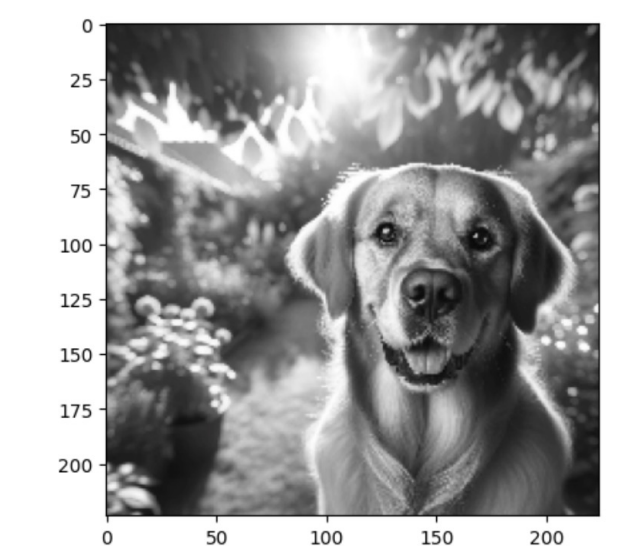
28×28では犬と分らない
→1024か224を入力したい

MLPの問題点1

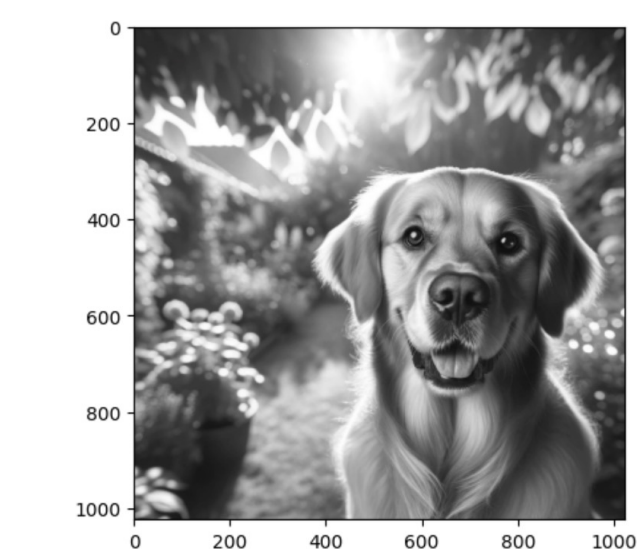
MLPでは画像サイズを1次元にして入力する→画像サイズ分の重みが存在



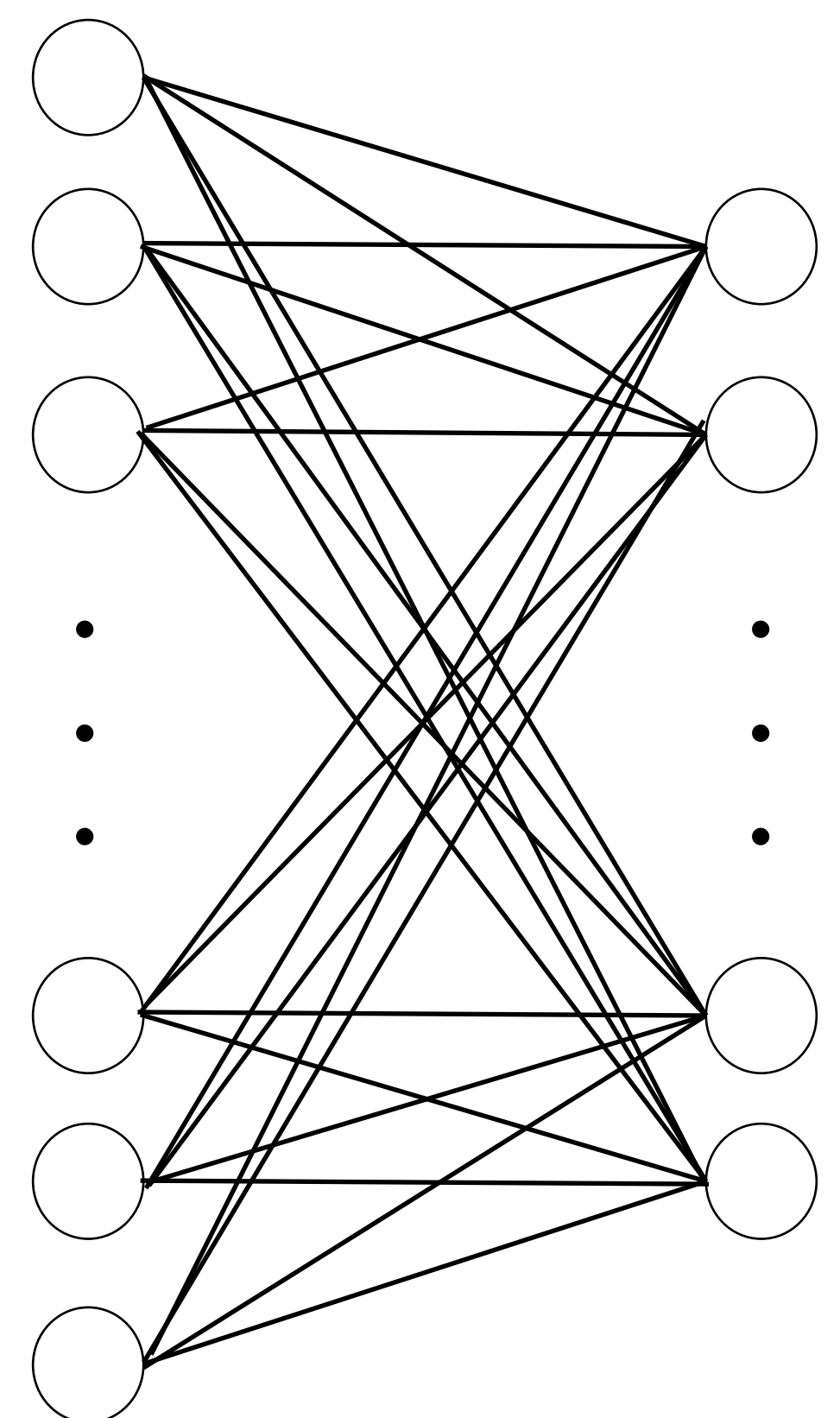
28×28



224×224



1024×1024

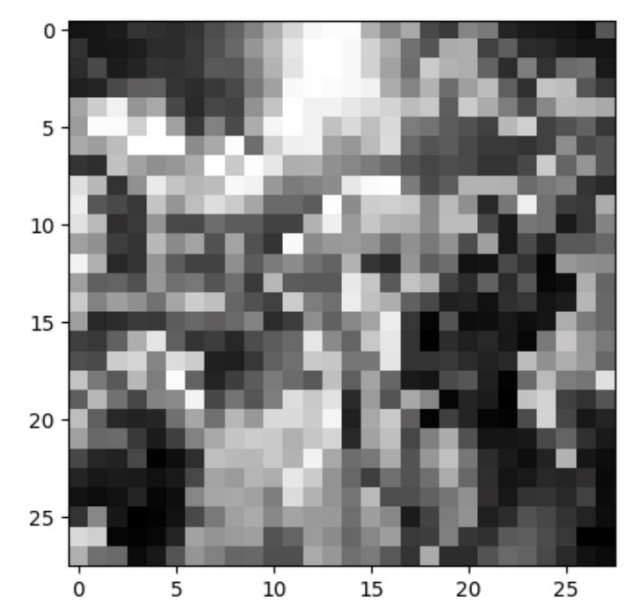


784 10

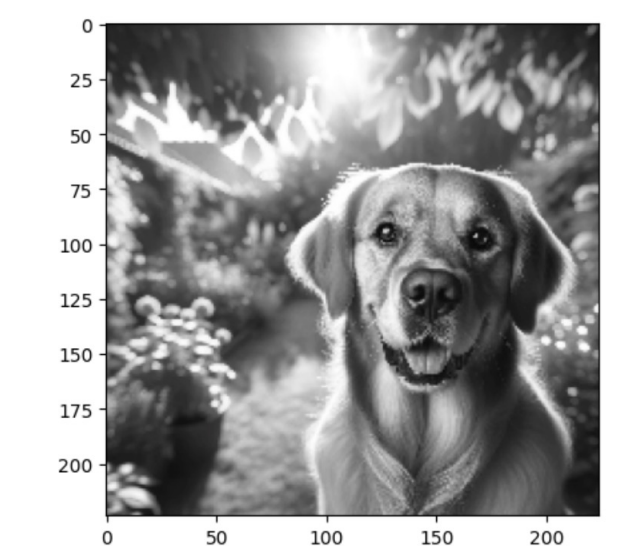
入力サイズが28×28
入力層と1つ目の中間層の間の重み
 $784 \times 10 = 7840$

MLPの問題点1

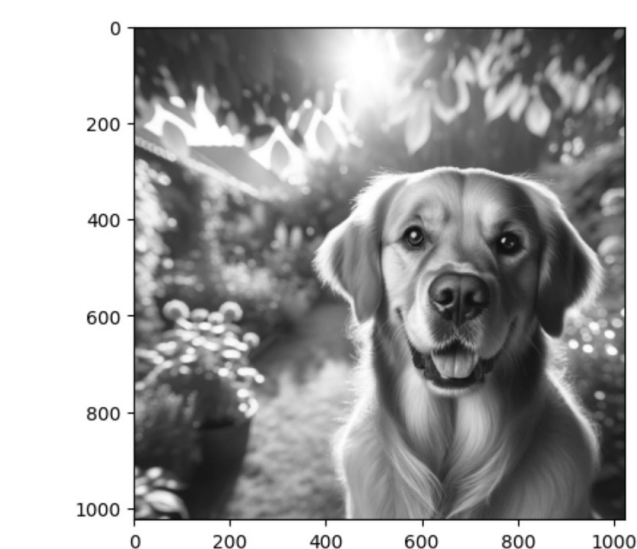
MLPでは画像サイズを1次元にして入力する→画像サイズ分の重みが存在



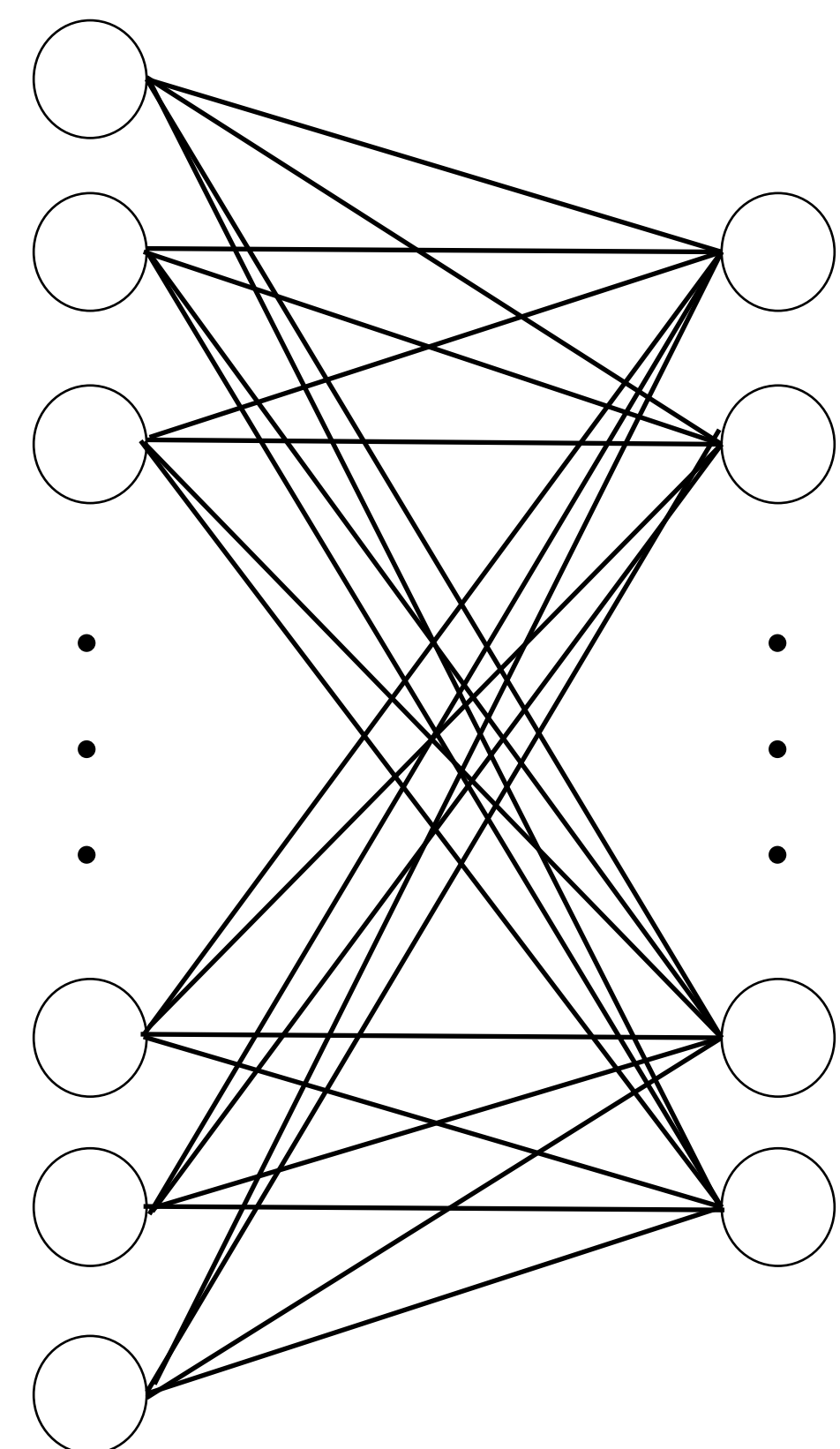
28×28



224×224



1024×1024



50176

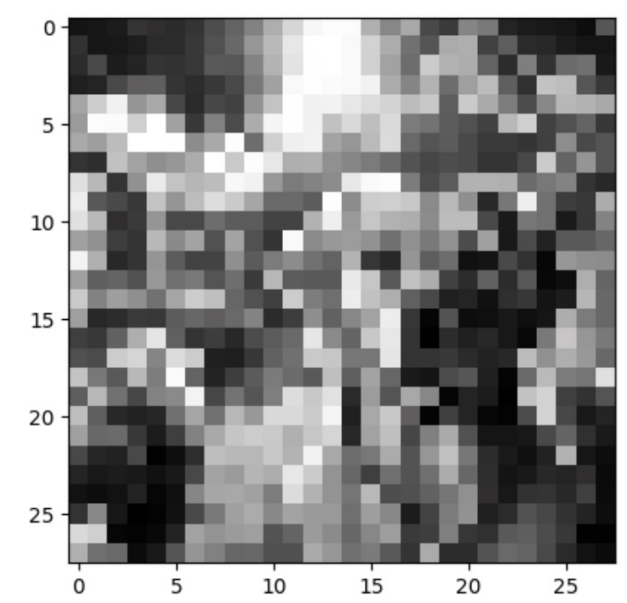
10

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

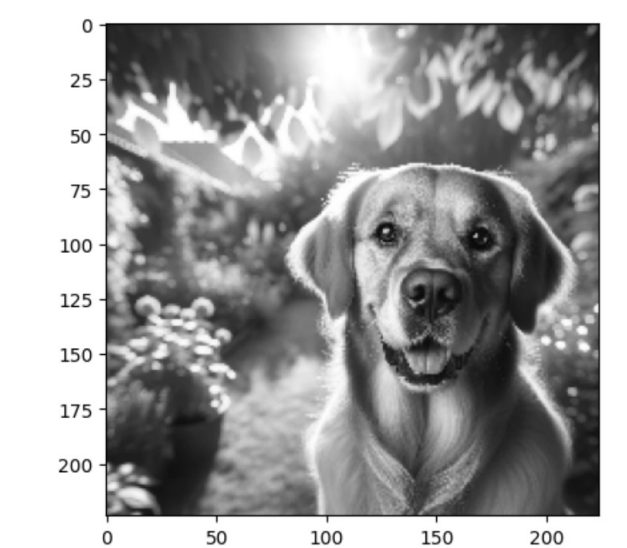
入力サイズが224×224
入力層と1つ目の中間層の間の重み
 $50176 \times 10 = 501760$

MLPの問題点1

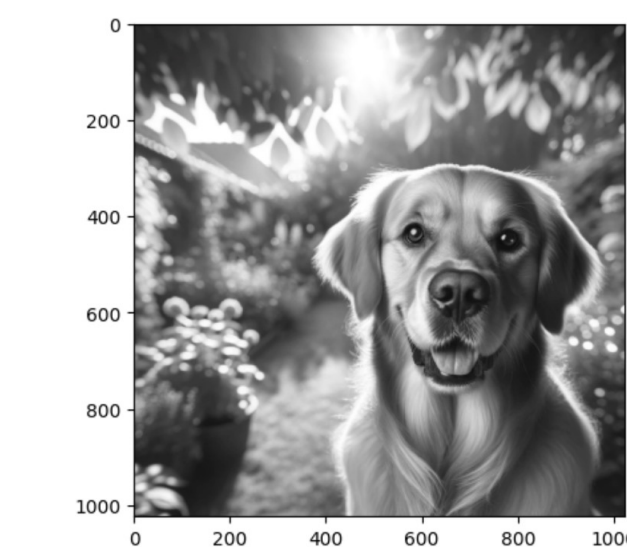
MLPでは画像サイズを1次元にして入力する→画像サイズ分の重みが存在



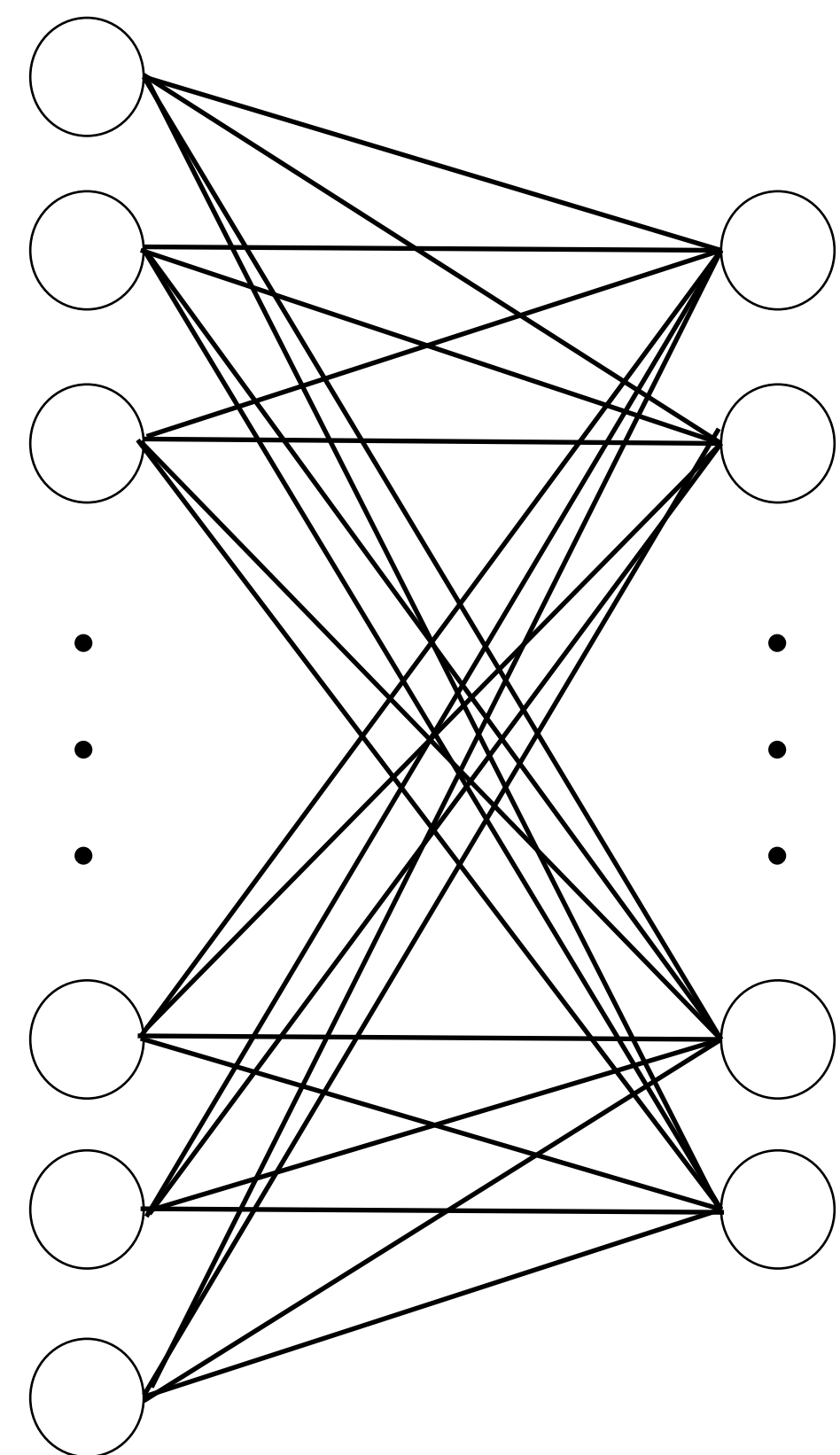
28×28



224×224



1024×1024



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
 $784 \times 10 = 7850$

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

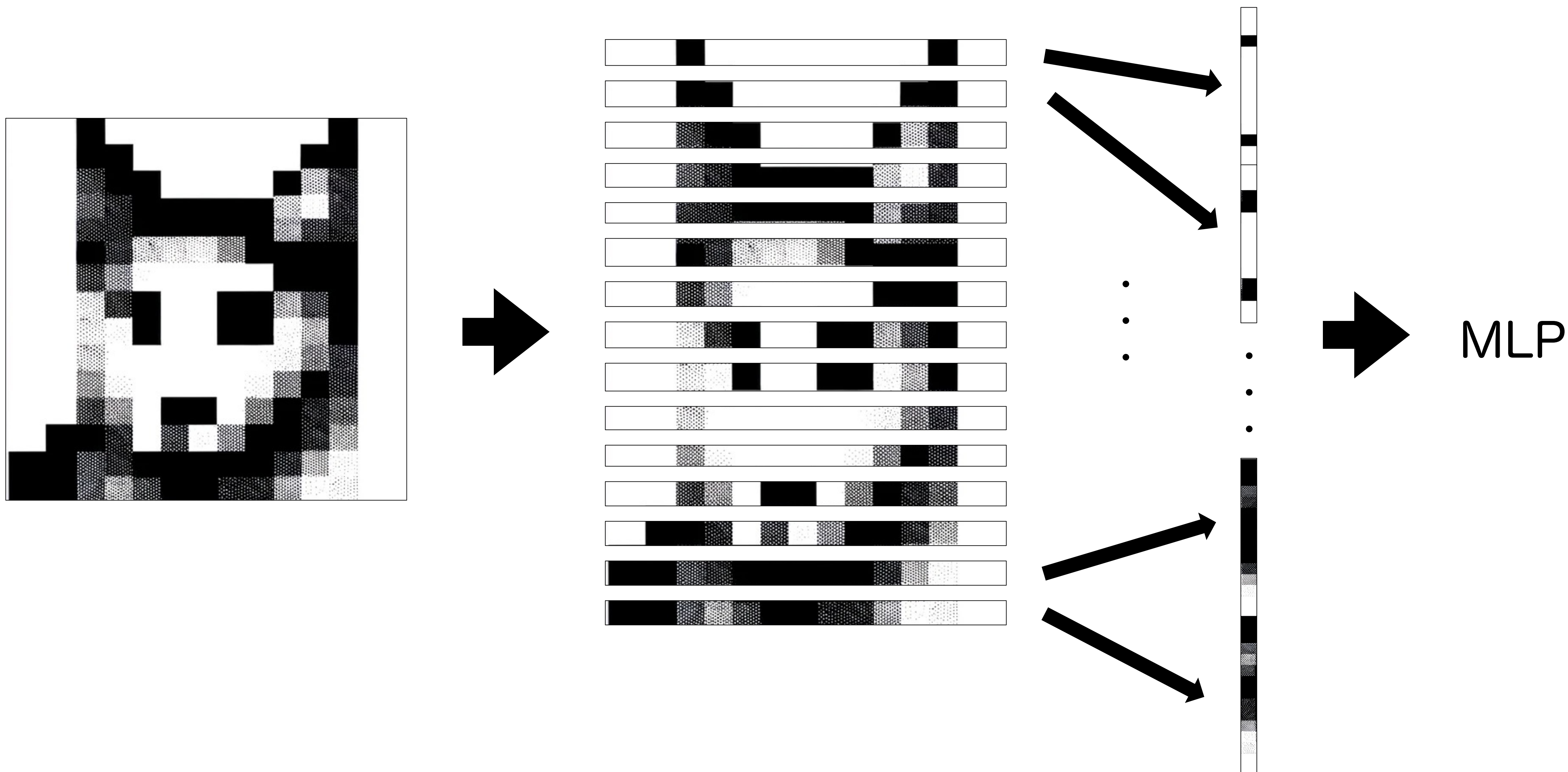
入力サイズが1024×1024
入力層と1つ目の中間層の間の重み
 $1048576 \times 10 = 10485760$

1048576

サイズが大きいかほど調整する重みが増えてしまう

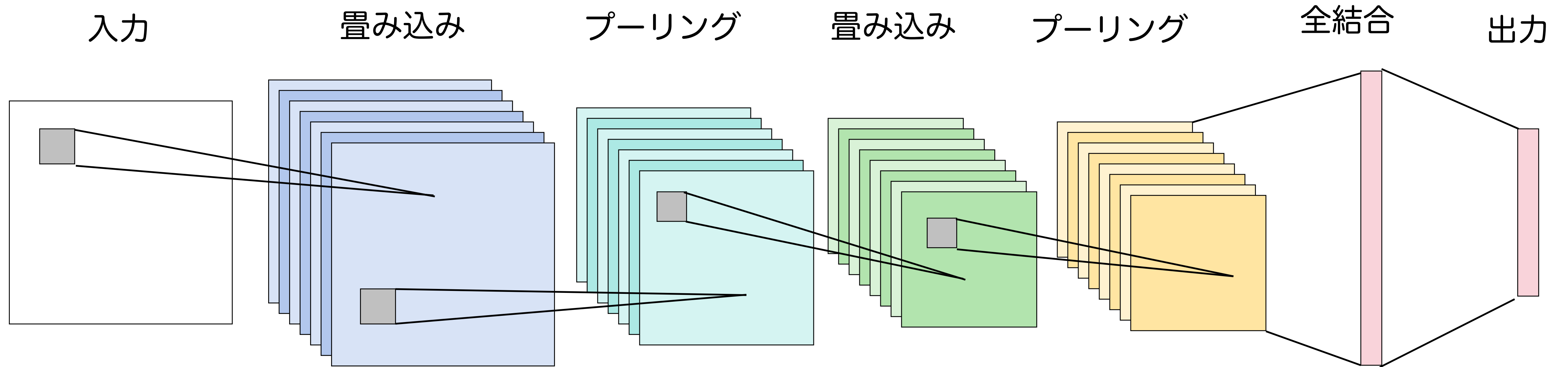
MLPの問題点2

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) (60000, 28, 28)
print(y_train.shape) (60000,)
print(x_test.shape) (10000, 28, 28)
print(y_test.shape) (10000,)
```

```
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) (60000, 28, 28, 1)
print(y_train.shape) (60000,)
print(x_test.shape) (10000, 28, 28, 1)
print(y_test.shape) (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

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		

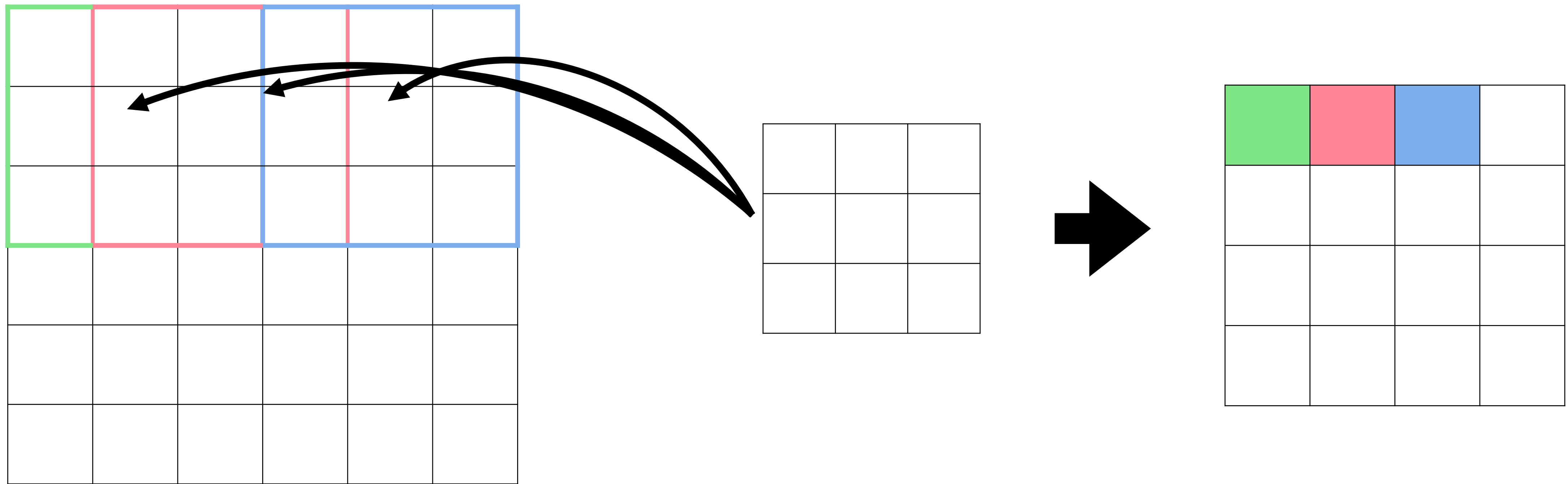
学習の実行

```
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

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

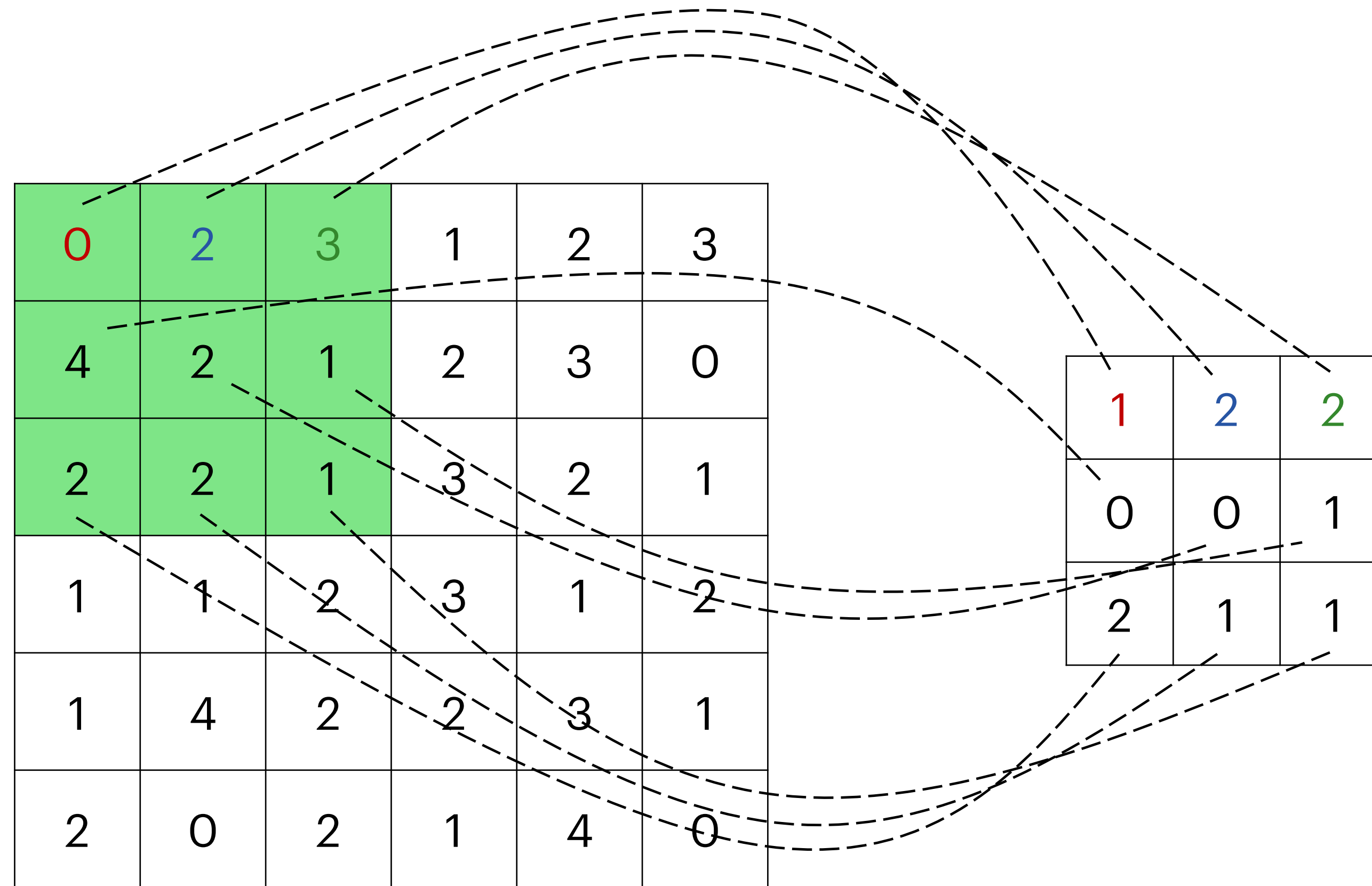
入力層

1	2	2
0	0	1
2	1	1

カーネル

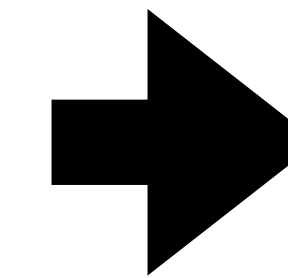
畳み込み層

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



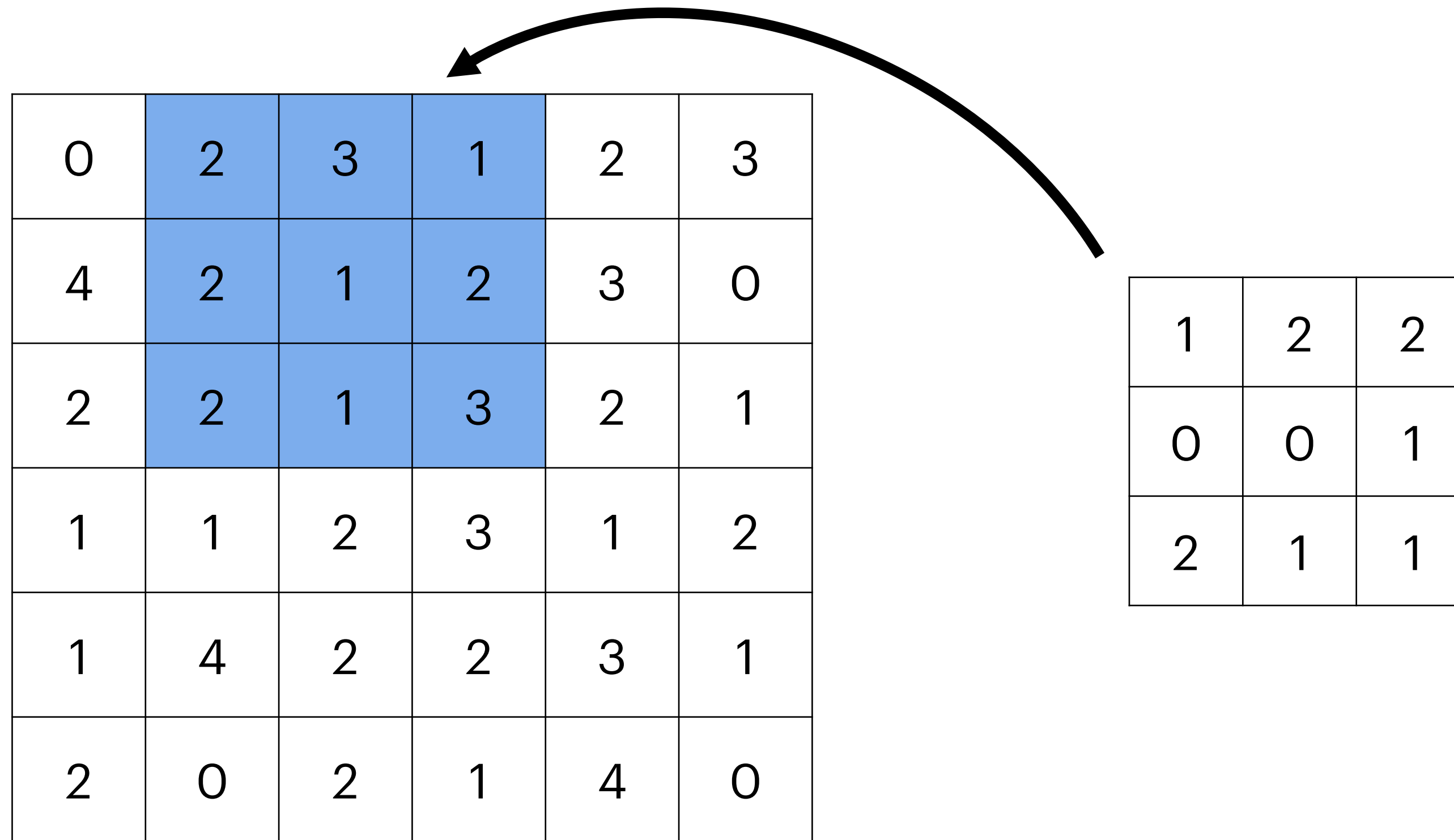
0×1	2×2	3×2
4×0	2×0	1×1
2×2	2×1	1×1

全ての積和を計算する

$$0 \times 1 + 2 \times 2 + 3 \times 2 + 4 \times 0 + 2 \times 0 + 1 \times 1 + 2 \times 2 + 2 \times 1 + 1 \times 1 = 18$$

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

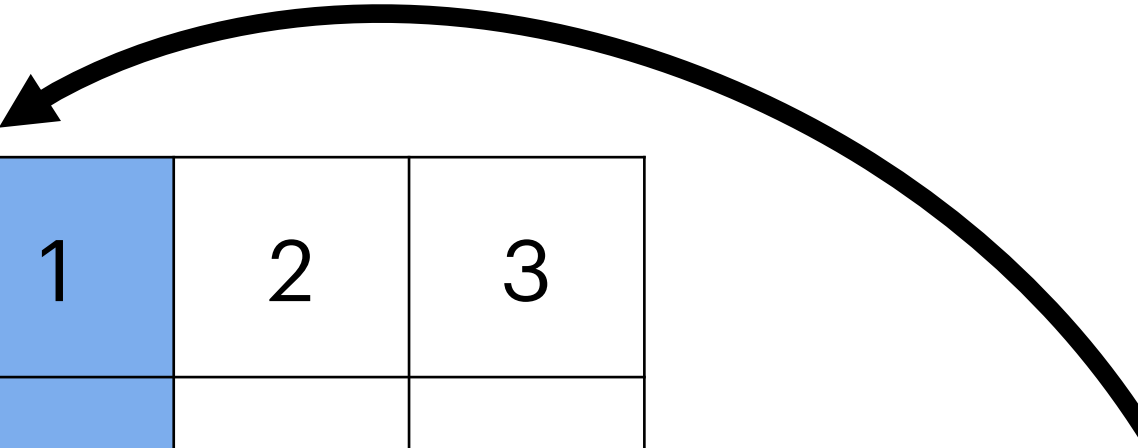


入力層

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

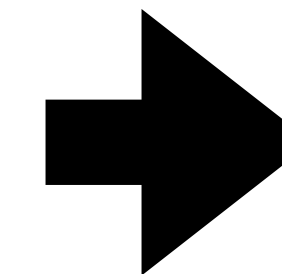


0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

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

$$\begin{aligned} &2 \times 1 + 3 \times 2 + 1 \times 2 + 2 \times 0 + 1 \times 0 \\ &+ 3 \times 1 + 2 \times 2 + 1 \times 1 + 3 \times 1 = 22 \end{aligned}$$

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

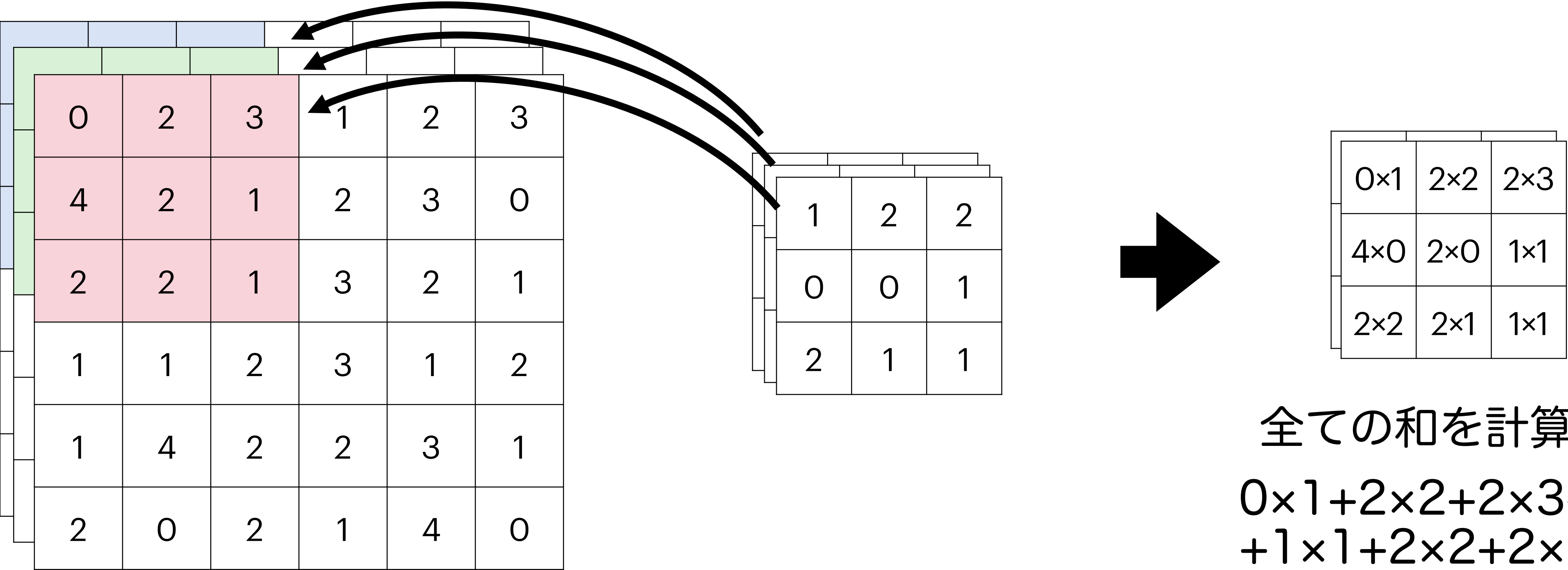
カーネル (3 × 3)

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

$$0 \times 1 + 2 \times 2 + 2 \times 3 + 4 \times 0 + 2 \times 0 + 1 \times 1 + 2 \times 2 + 2 \times 1 + 1 \times 1$$

+ (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

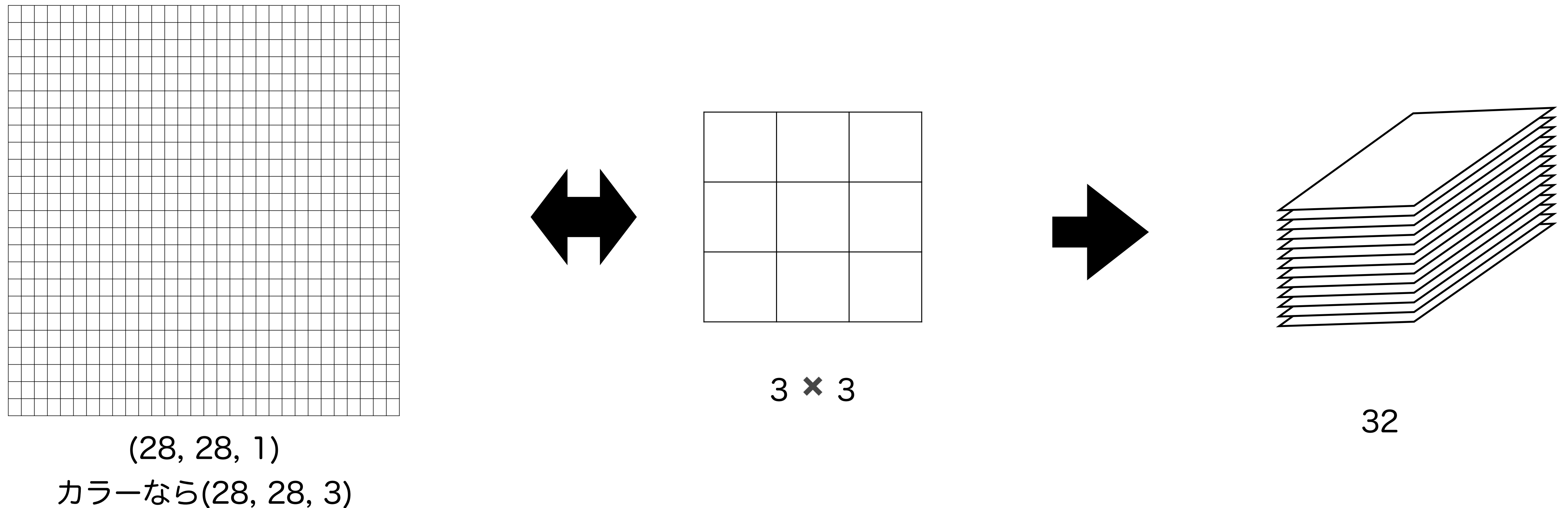
activation = 活性化関数

```
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

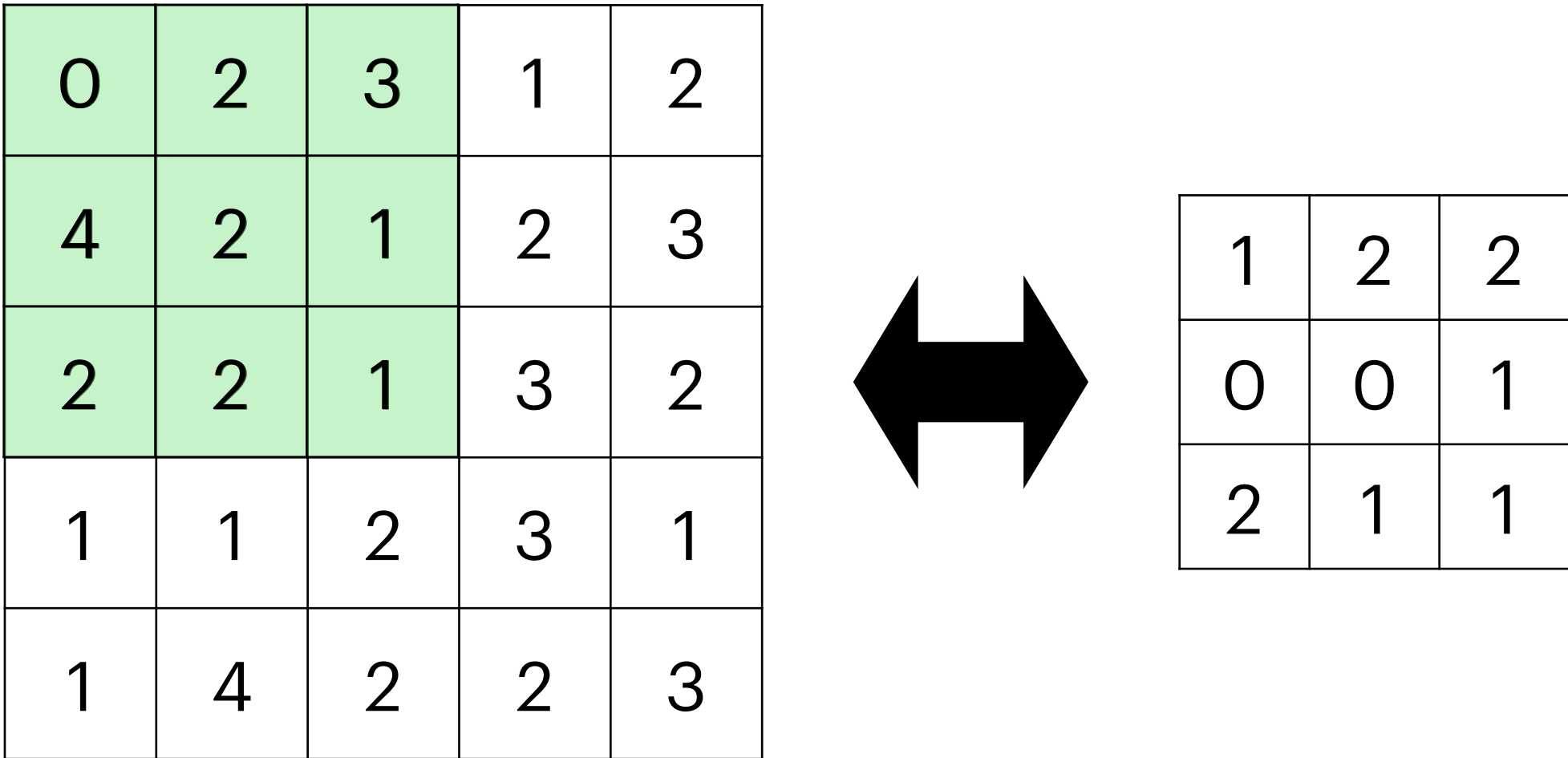
kernel_size = カーネルの大きさ



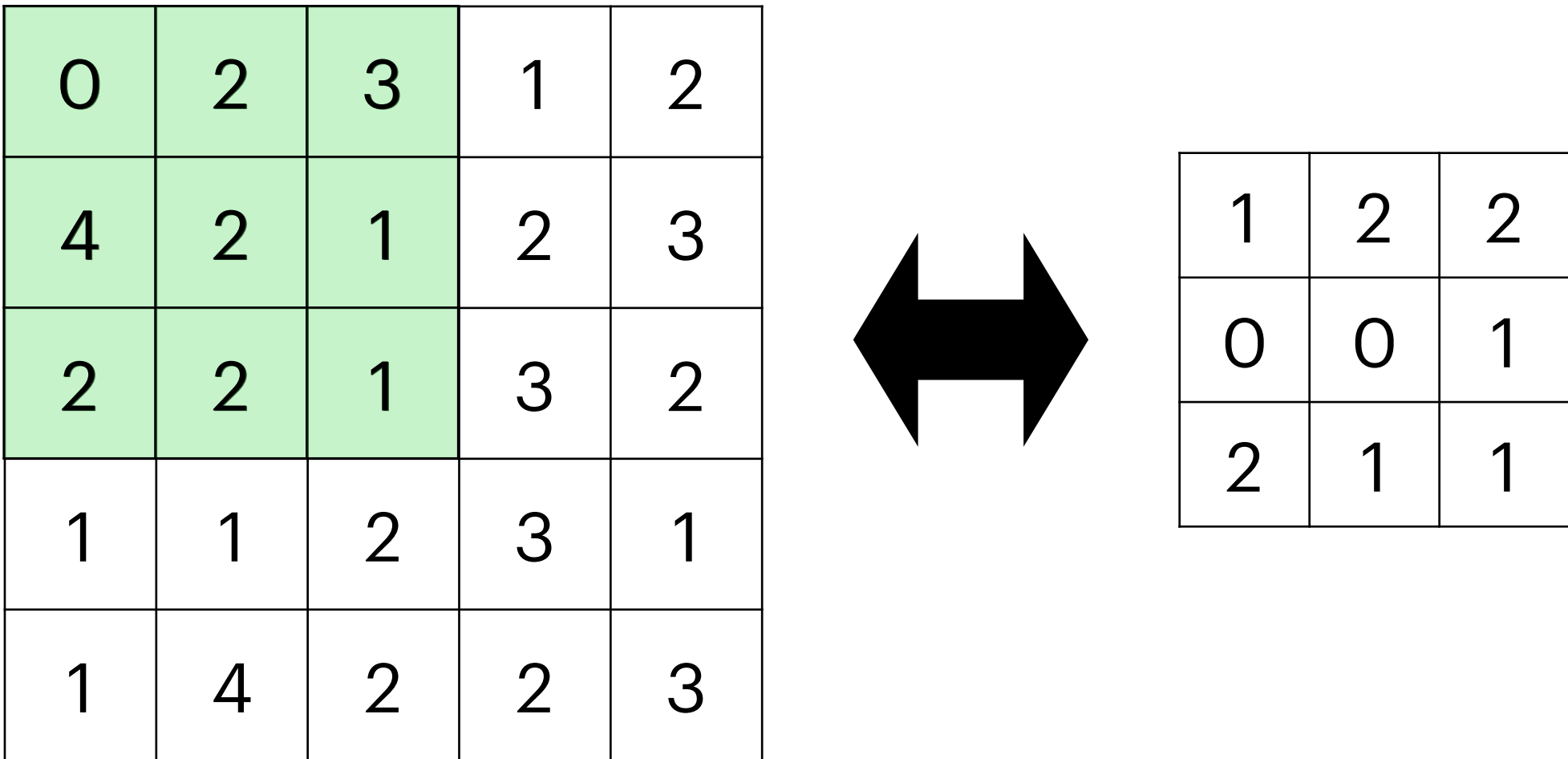
```
model.add(Conv2D(filters=32,kernel_size=3, strides=1,
padding='same',input_shape=(28,28,1),activation='relu'))
```

strides = カーネルをずらす幅

strides = 1



strides = 2

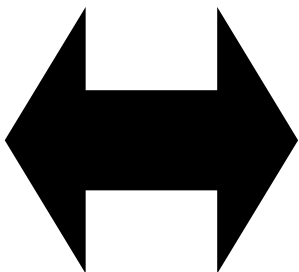


```
model.add(Conv2D(filters=32,kernel_size=3,strides=1,
padding='same',input_shape=(28,28,1),activation='relu'))
```

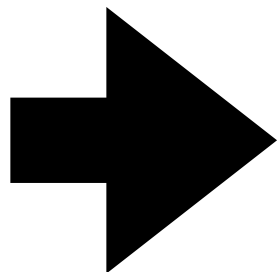
strides = カーネルをずらす幅

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



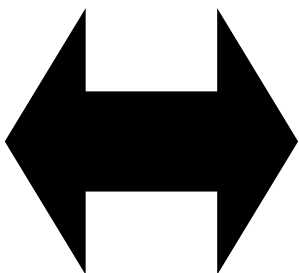
1	2	2
0	0	1
2	1	1



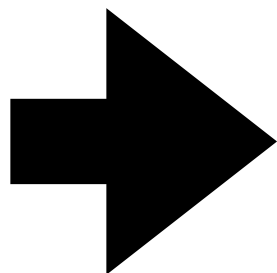
18	20	19

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



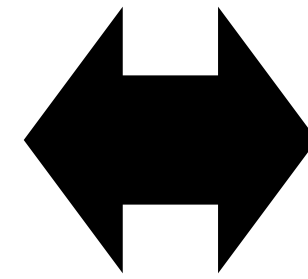
18	19

```
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

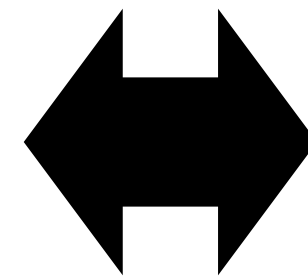
カーネル (3 × 3)


```
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

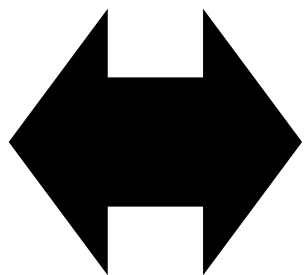
カーネル (3 × 3)

```
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

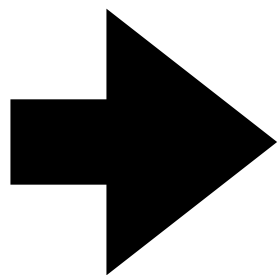
0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

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)

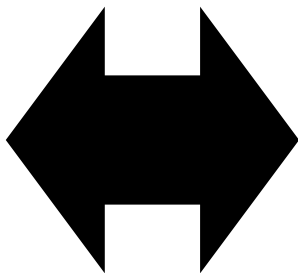
そのままだと特徴マップのサイズは入力層より小さくなる

```
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

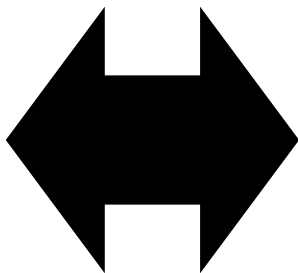
カーネル (3× 3)

```
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

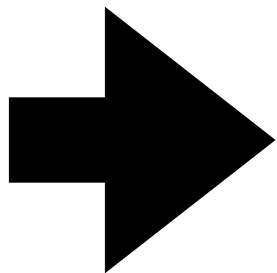
0	0	0	0	0	0	0	0
0	0	2	3	1	2	3	0
0	4	2	1	2	3	0	0
0	2	2	1	3	2	1	0
0	1	1	2	3	1	2	0
0	1	4	2	2	3	1	0
0	2	0	2	1	4	0	0
0	0	0	0	0	0	0	0

入力層 (6 × 6)



1	2	2
0	0	1
2	1	1

カーネル (3× 3)

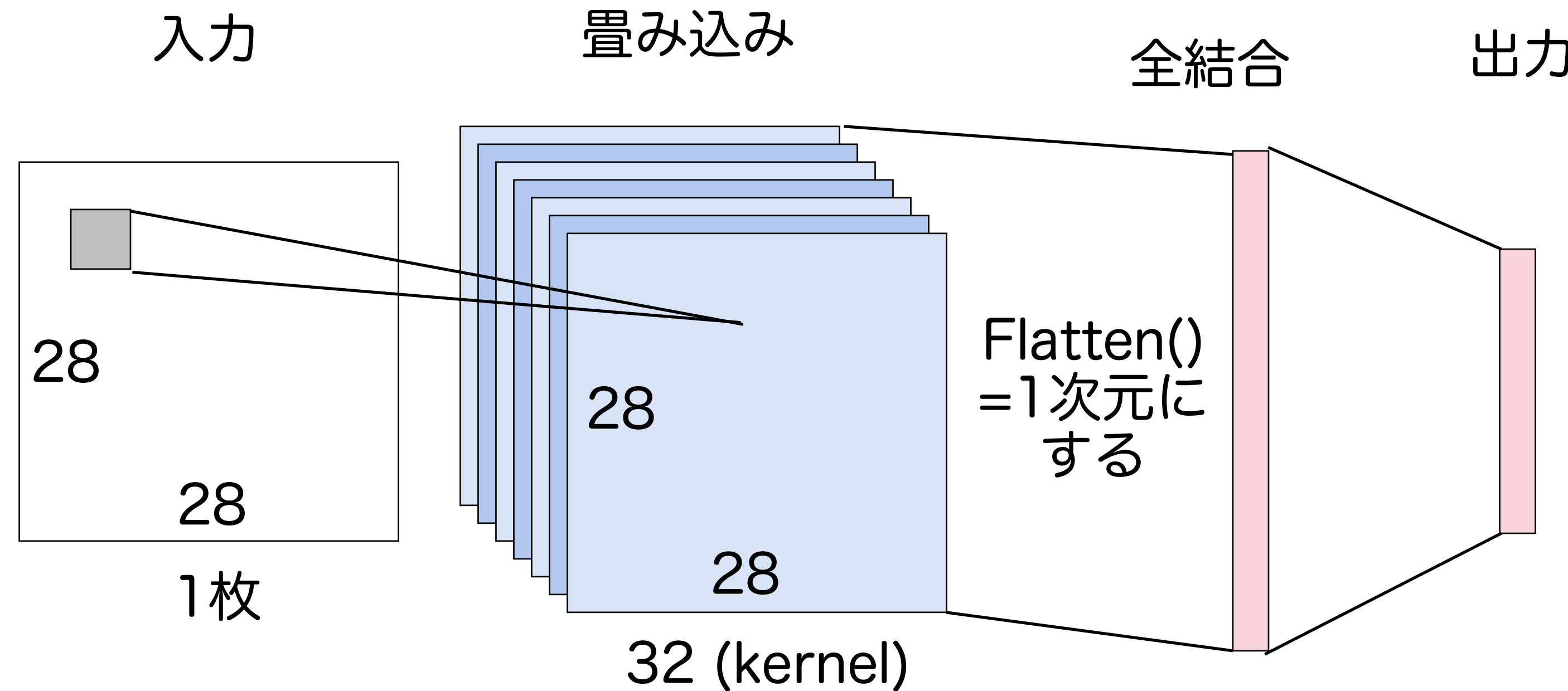


8	14	8	9	10	6
10	18	22	19	20	13
16	16	18	21	18	7
14	18	25	21	19	11
10	15	17	22	16	13
10	15	13	16	10	5

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



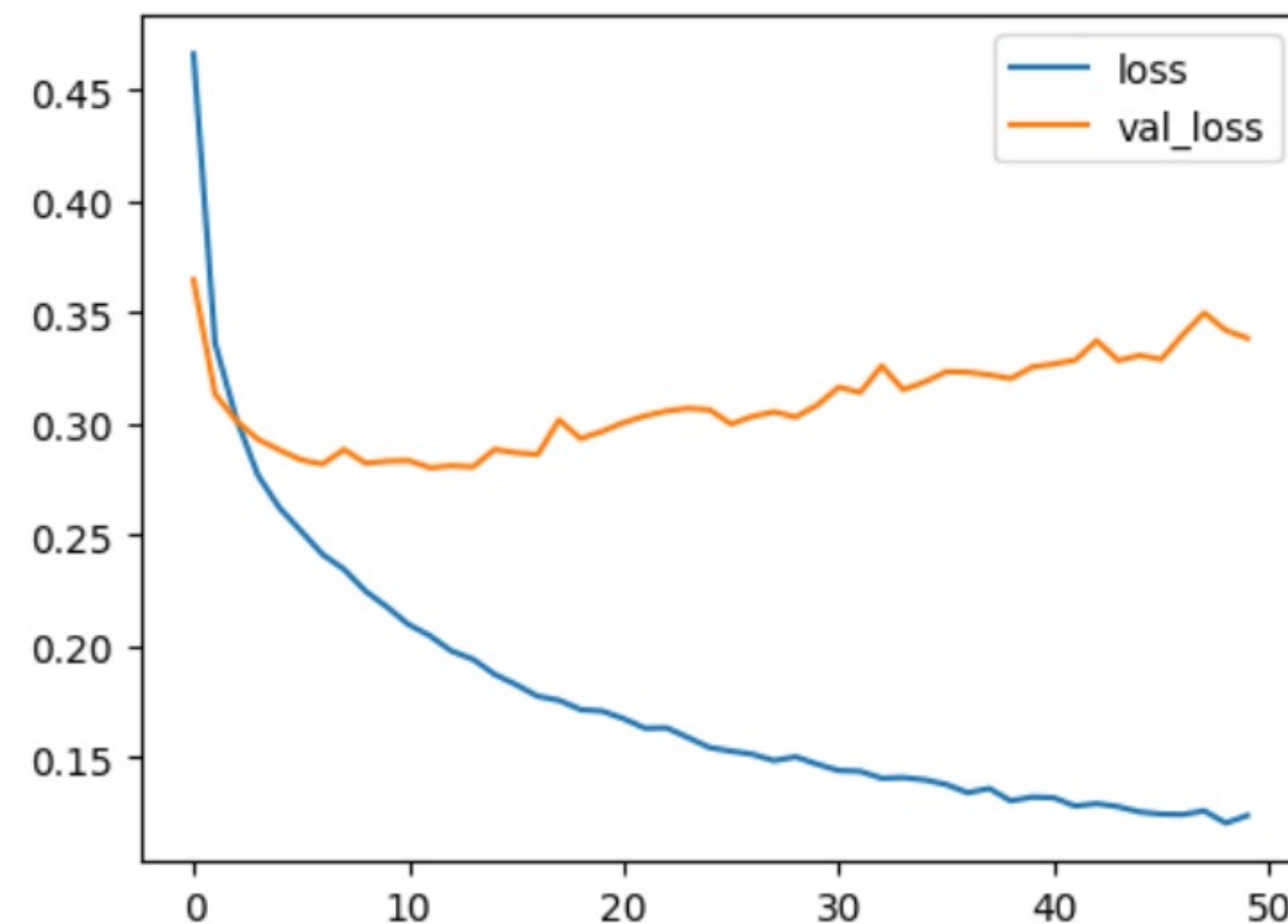
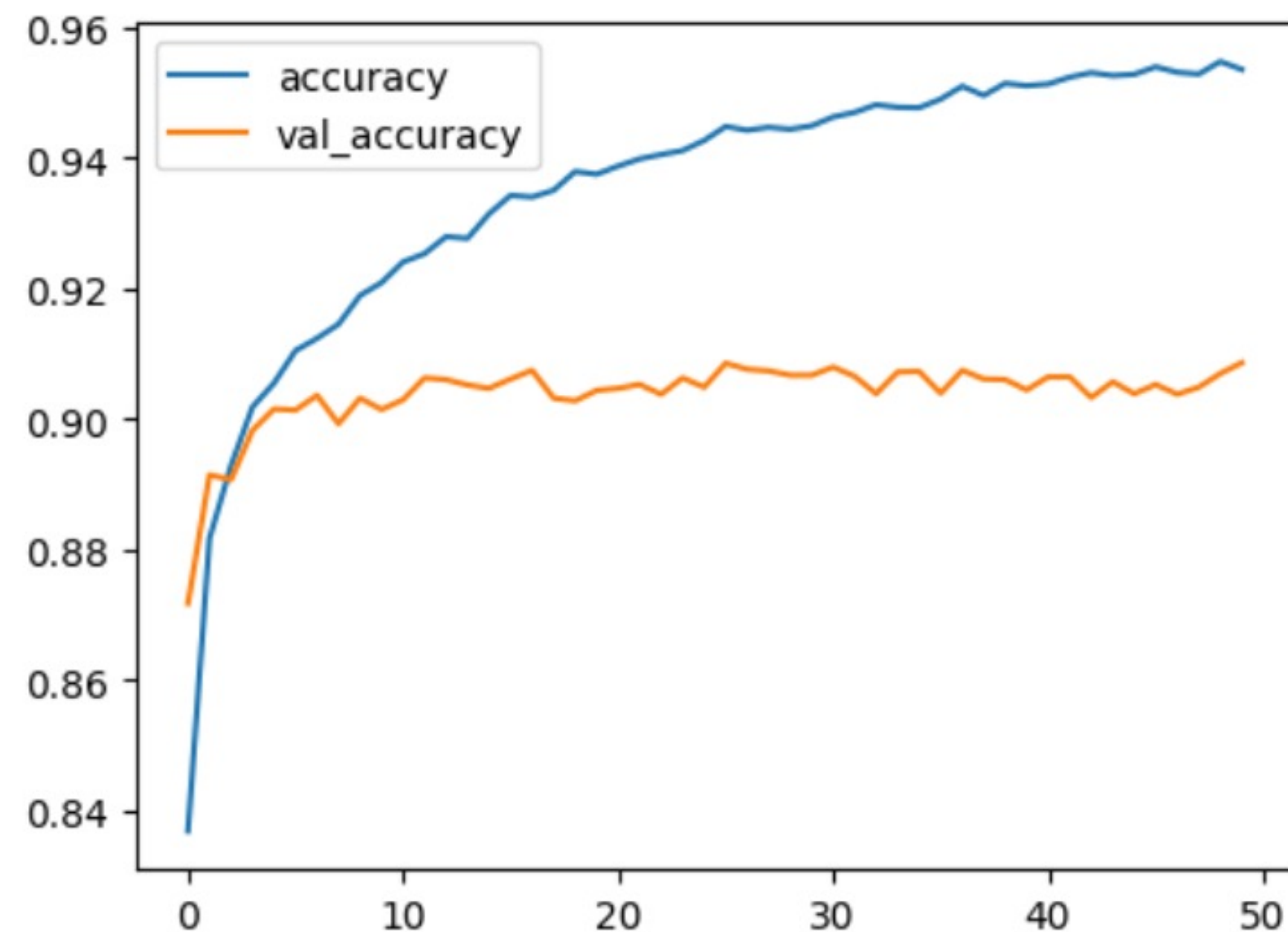
Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 28, 28, 32)	320
flatten_2 (Flatten)	(None, 25088)	0
dropout_2 (Dropout)	(None, 25088)	0
dense_8 (Dense)	(None, 10)	250890
Total params: 251,210		
Trainable params: 251,210		
Non-trainable params: 0		

$$(3 \times 3 + 1) \times 32 = 320$$

$$(28 \times 28 \times 32 + 1) \times 10 = 250890$$

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()
```

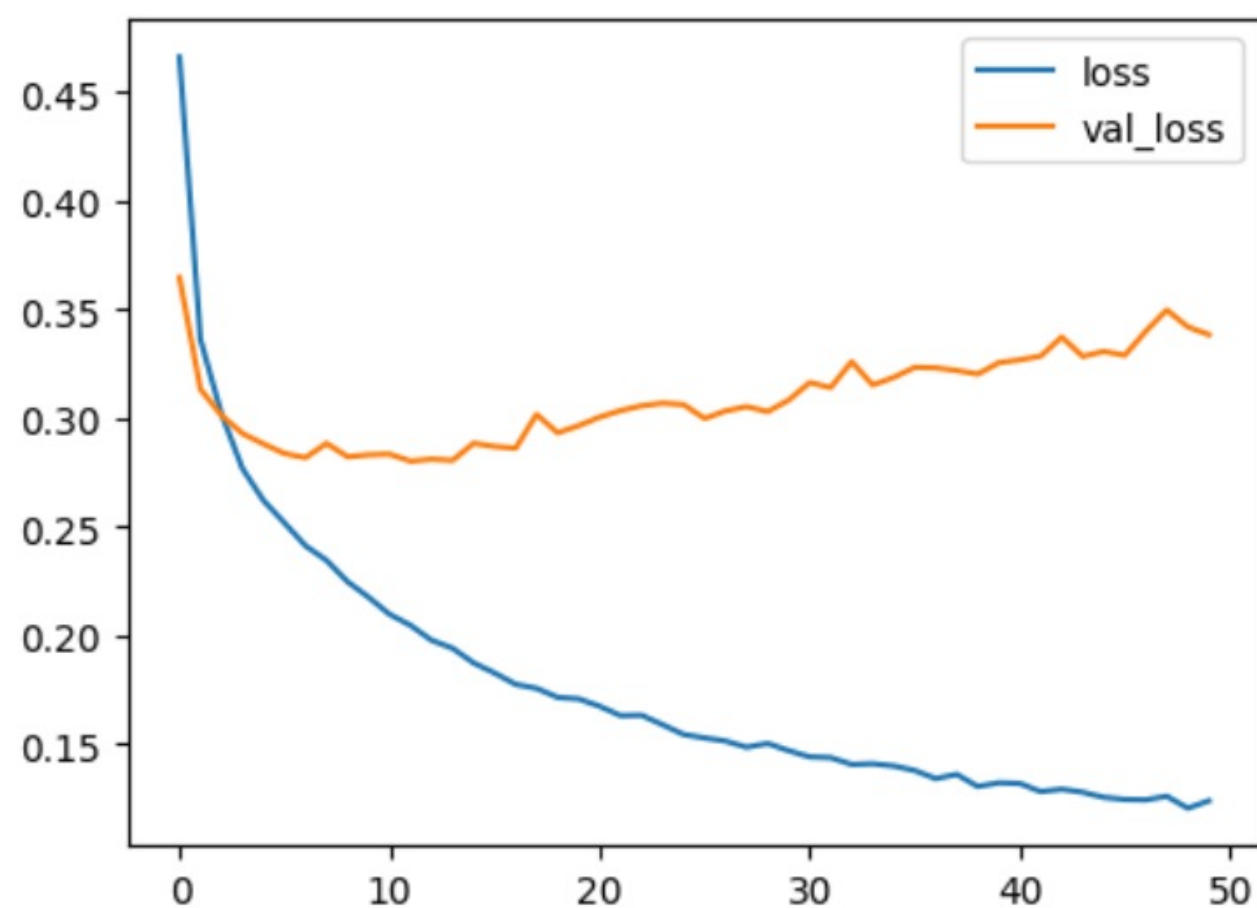
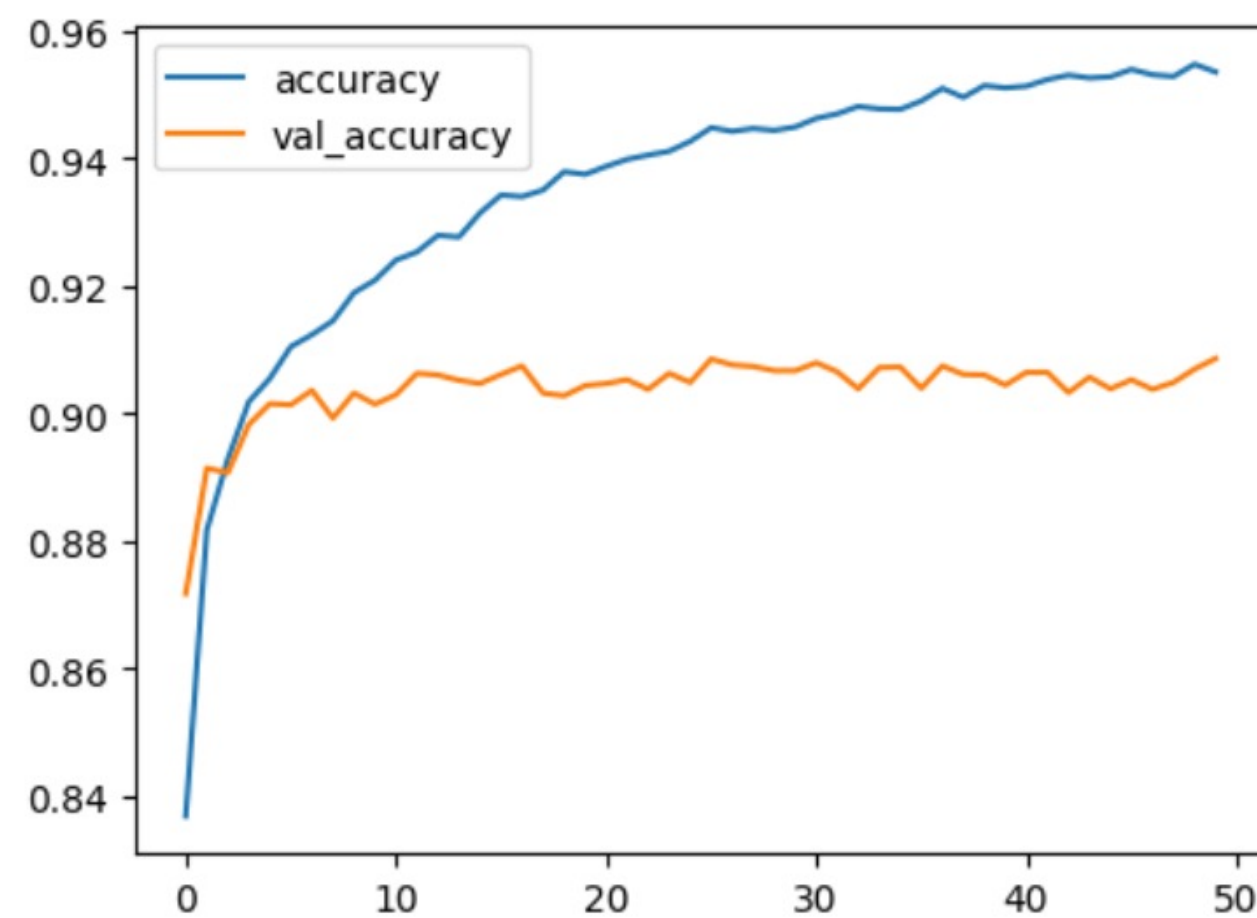



```
score = model.evaluate(x_test, y_test)
print('test loss:',score[0])
print('test accuracy:',score[1])
```

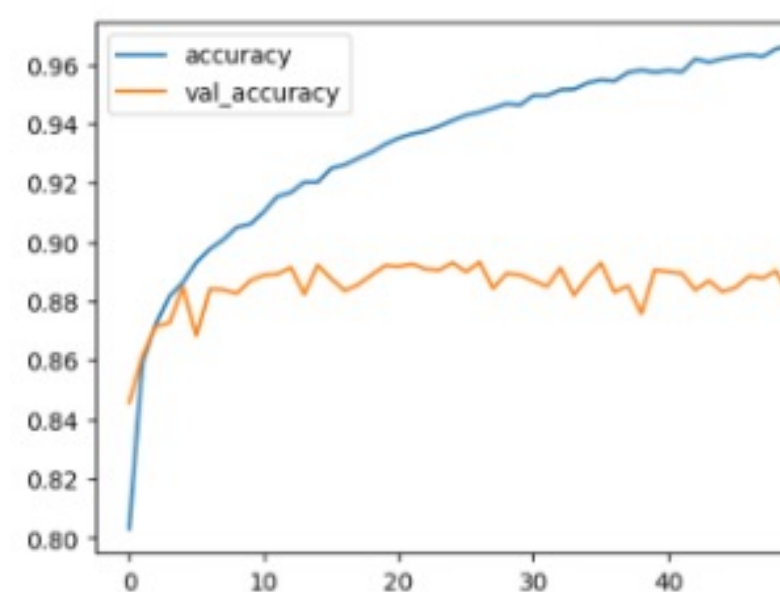
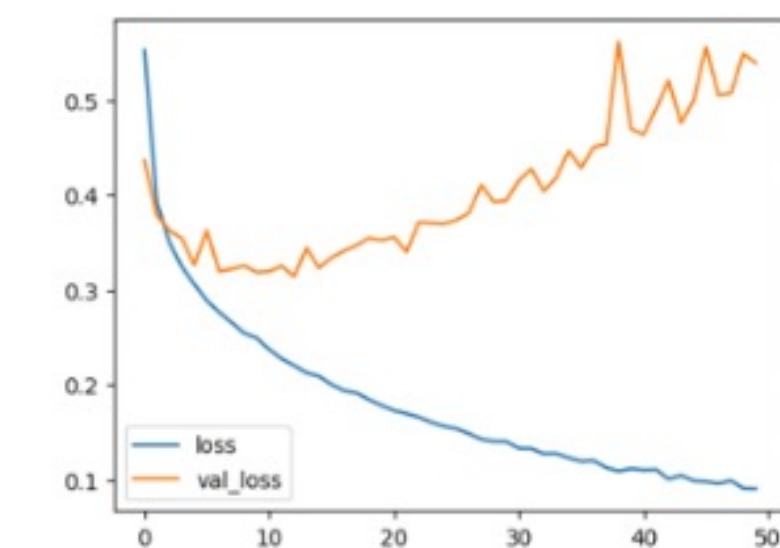
test loss: 0.34445714950561523

test accuracy: 0.9031999707221985

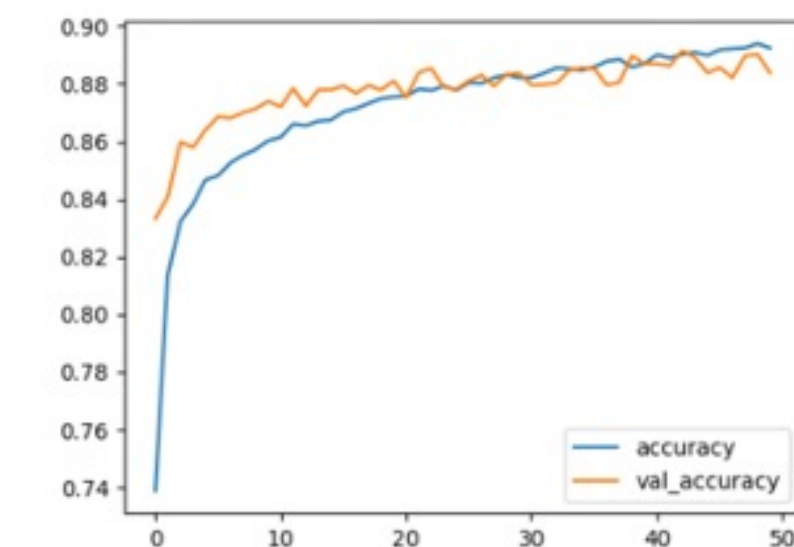
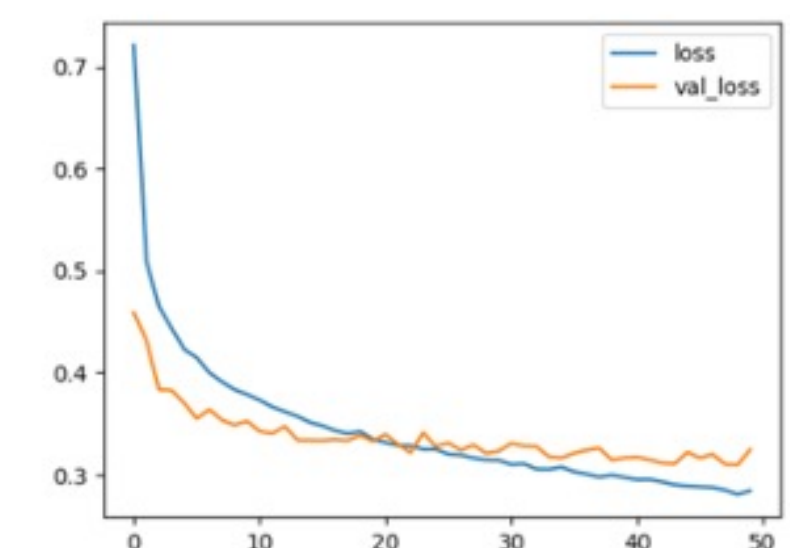
既にMLPよりも精度が良いことが分かる
MLP（前回のスライド）



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'])
```

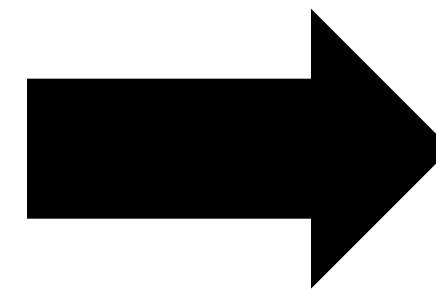
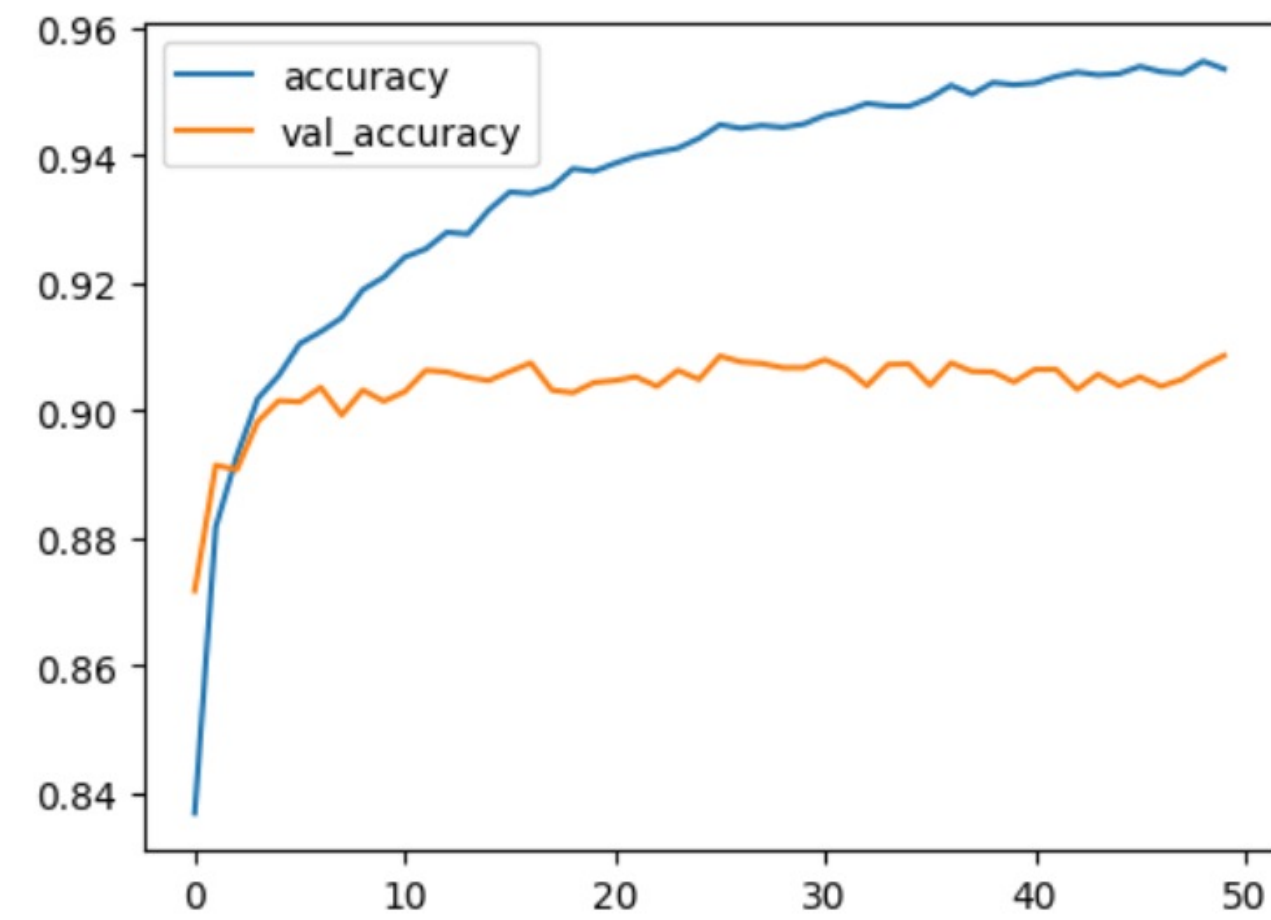
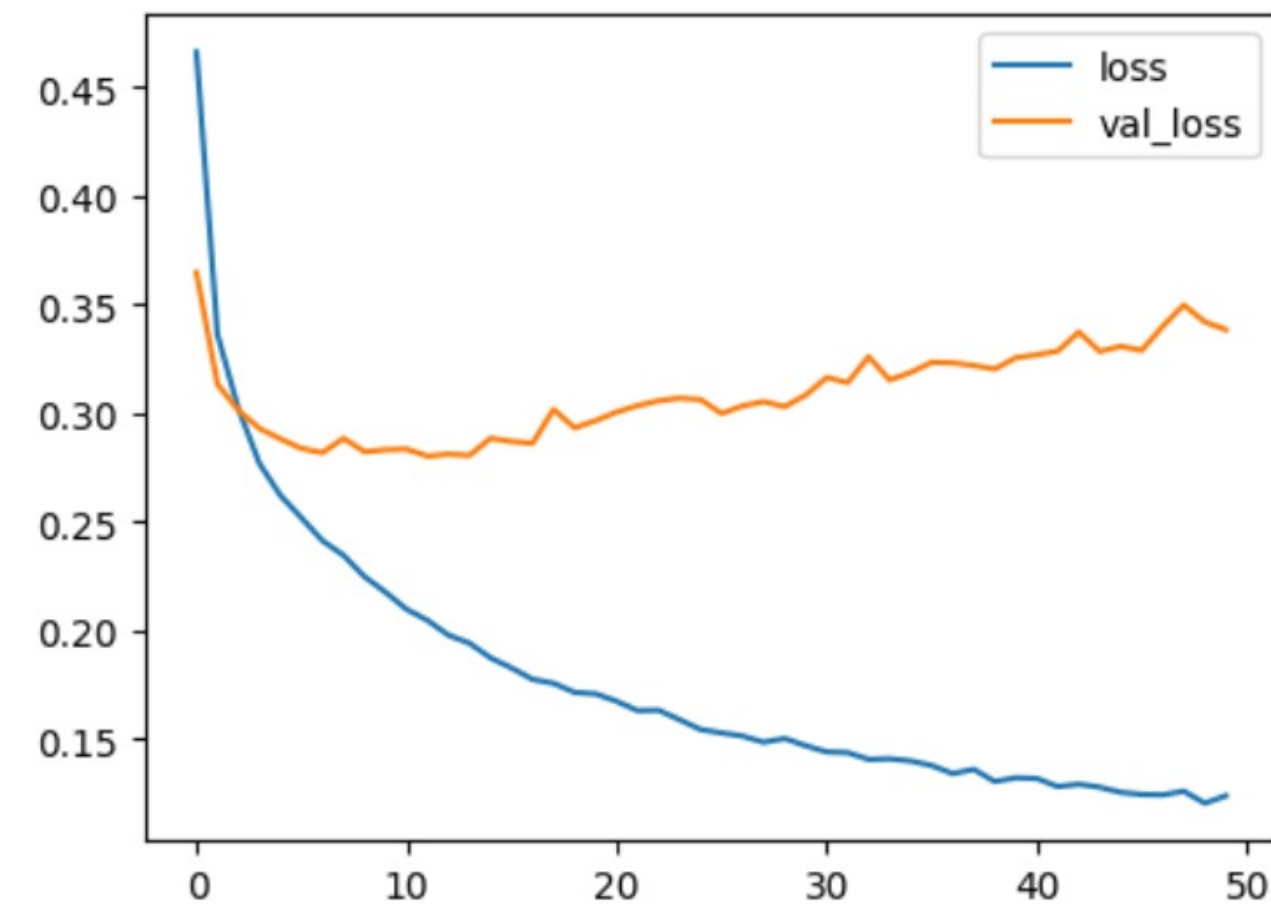
```
model.summary()
result = model.fit(x_train, y_train, epochs=50, batch_size=64, validation_split=0.2)
```

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

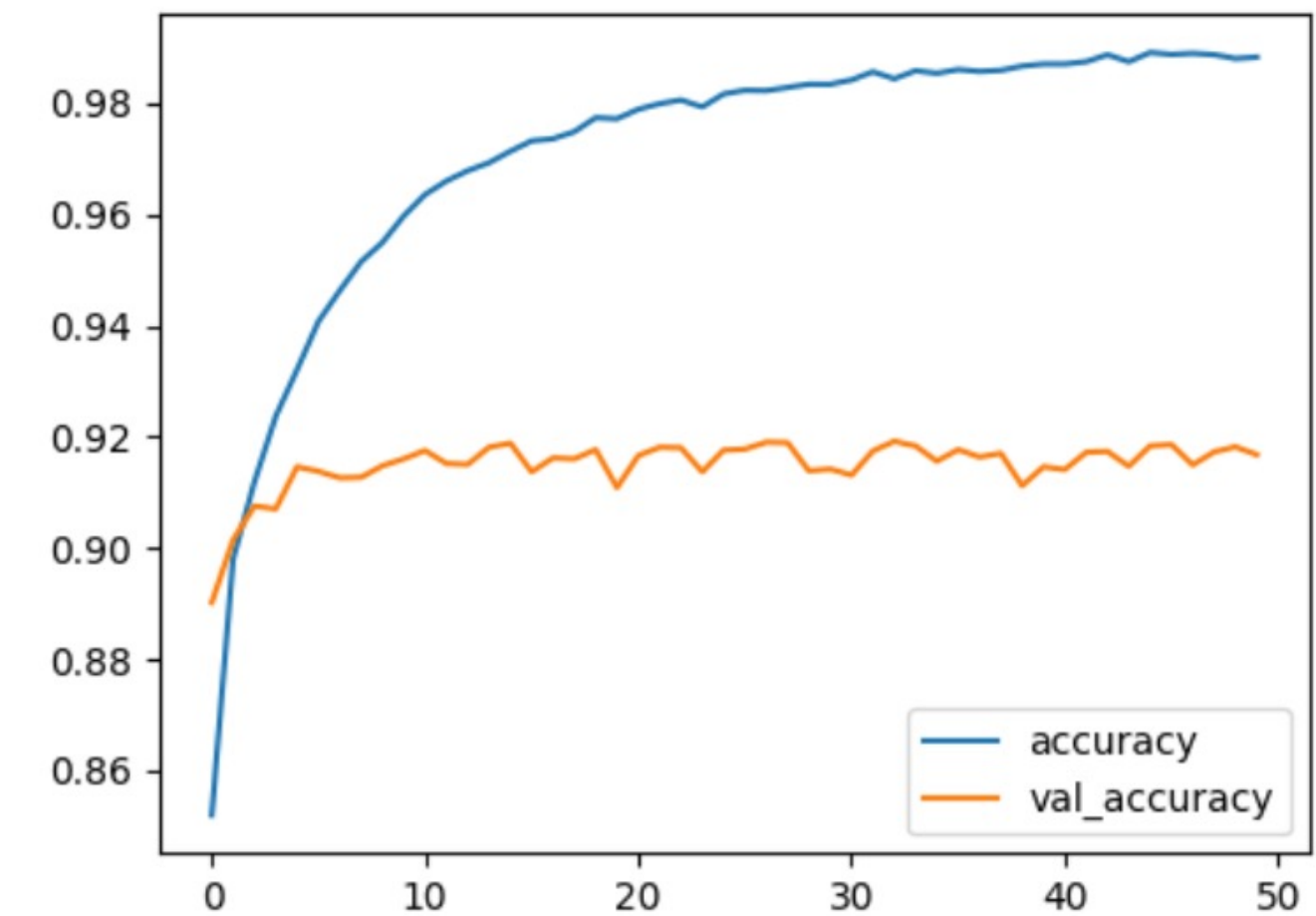
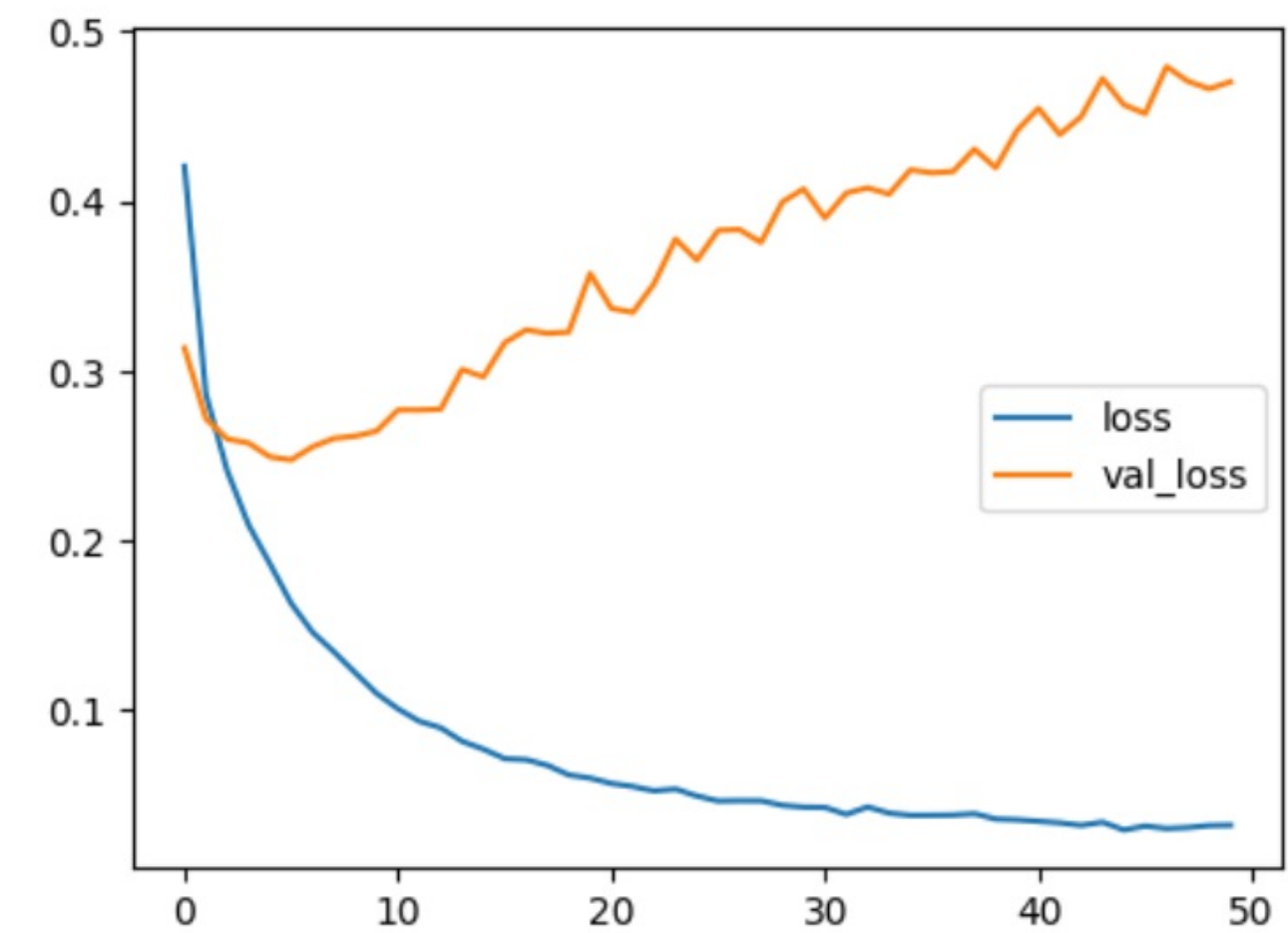
test loss: 0.34445714950561523

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

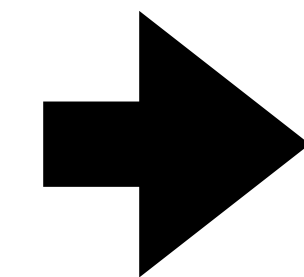
データを縮小する方法

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することで、データを縮小する。

3	<u>4</u>	5	<u>6</u>
1	2	3	4
-1	<u>3</u>	0	3
2	2	<u>5</u>	2

プーリング前の特徴マップ

MaxPooling



(2,2)

4	6
3	5

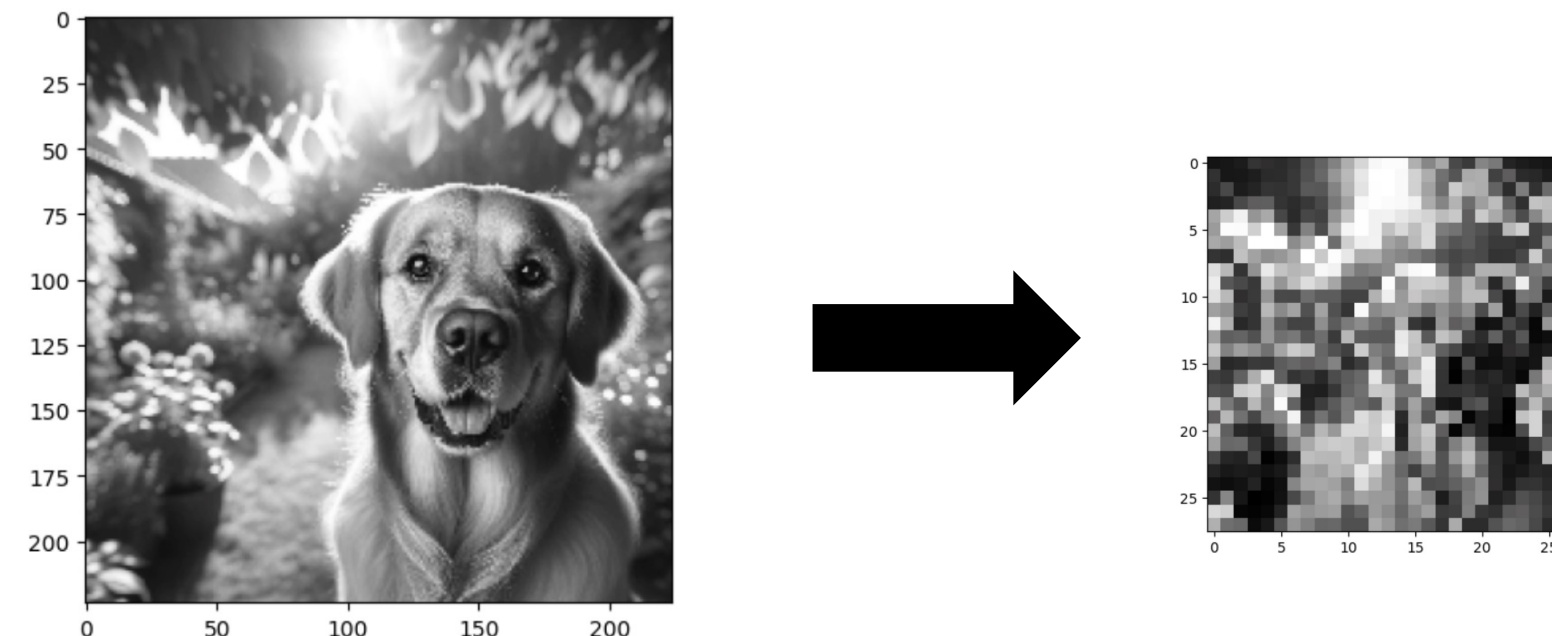
プーリング後の特徴マップ

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, <u>14</u> , <u>14</u> , 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

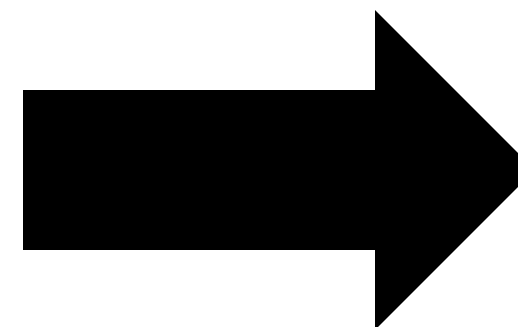
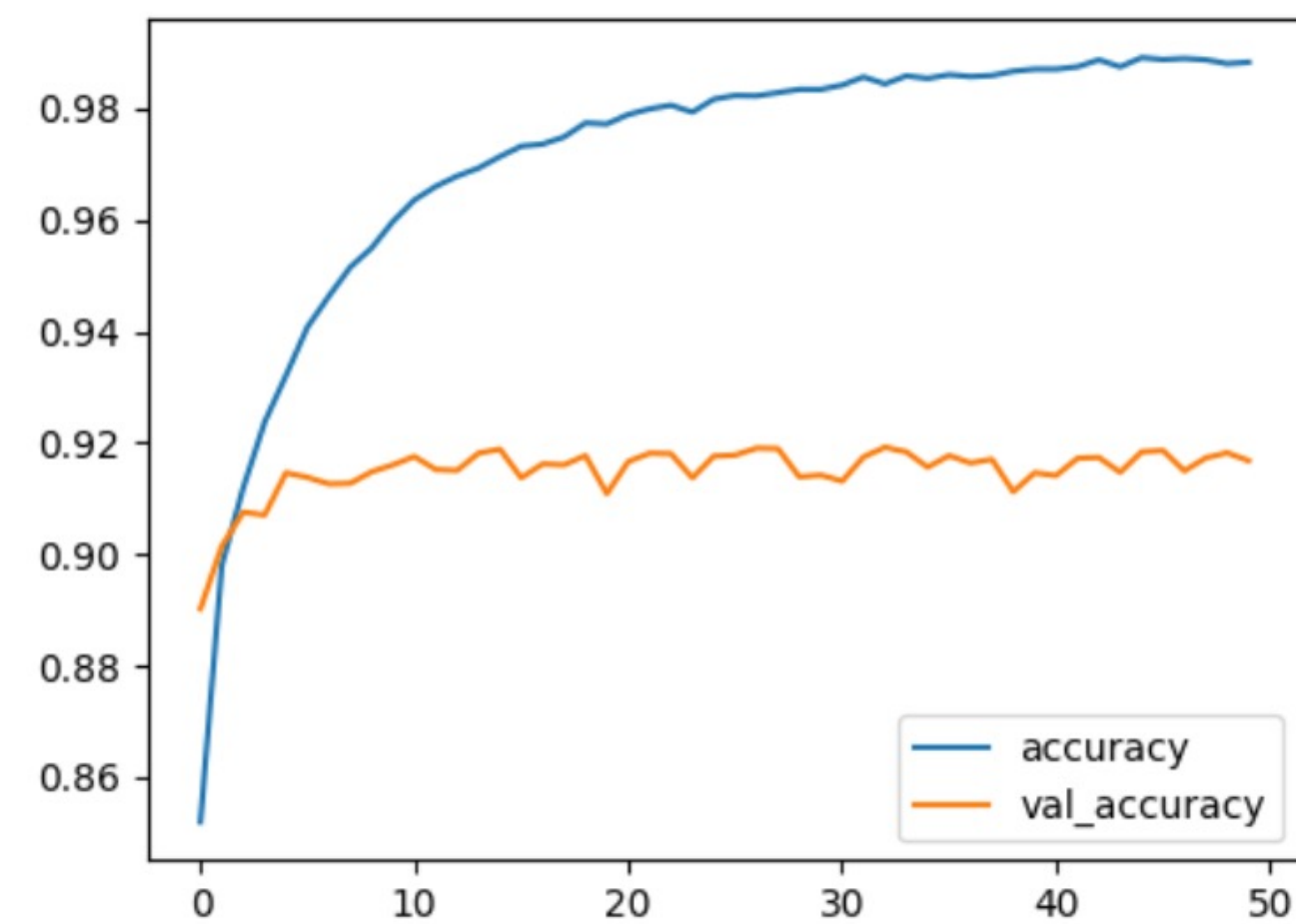
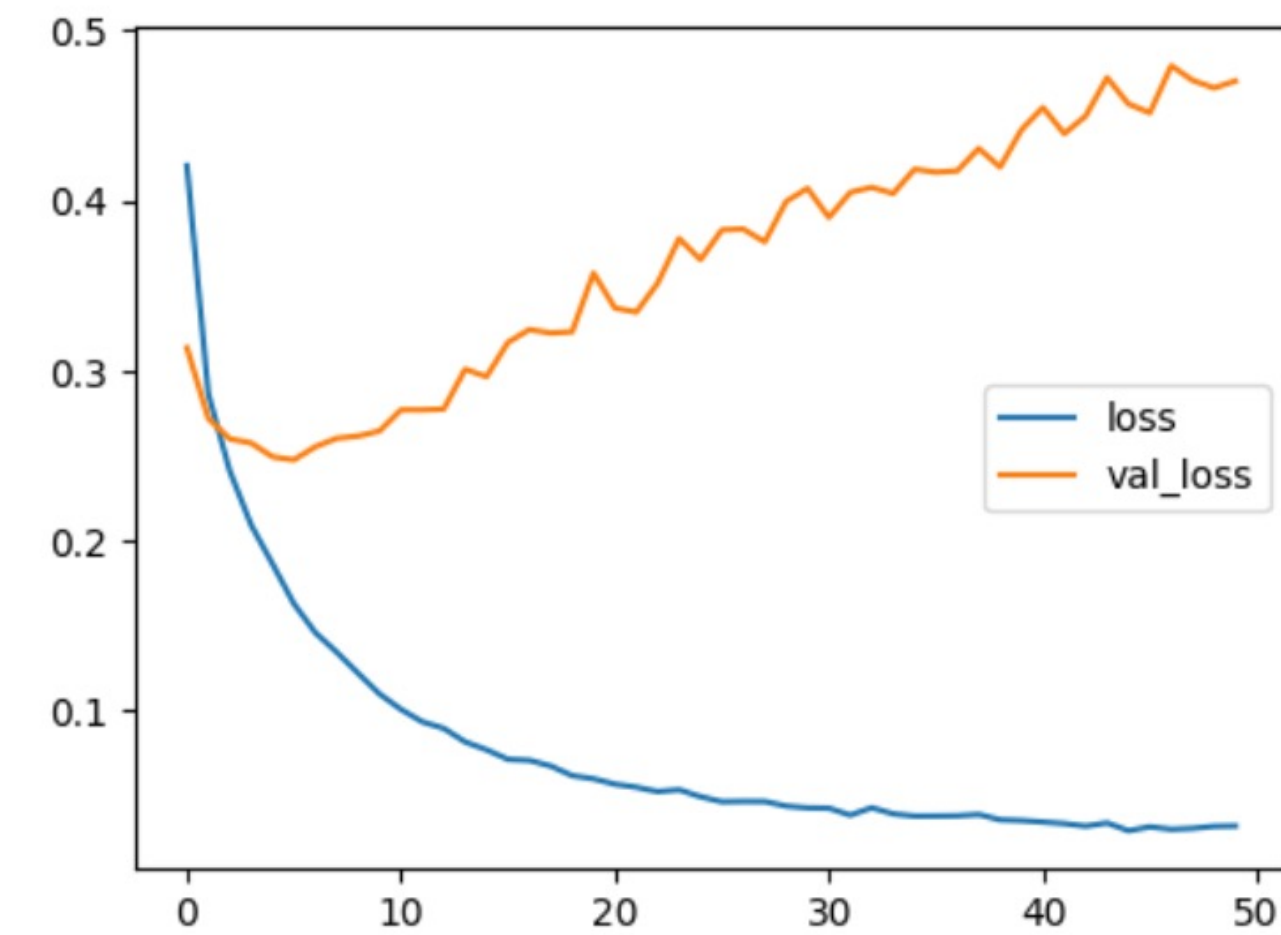
Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 28, 28, 32)	320
conv2d_4 (Conv2D)	(None, 28, 28, 64)	18496
flatten_3 (Flatten)	(None, 50176)	0
dropout_3 (Dropout)	(None, 50176)	0
dense_9 (Dense)	(None, 10)	<u>501770</u>
Total params: 520,586		
Trainable params: 520,586		
Non-trainable params: 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, <u>14, 14, 64</u>)	0
flatten_4 (Flatten)	(None, 12544)	0
dropout_4 (Dropout)	(None, 12544)	0
dense_10 (Dense)	(None, 10)	<u>125450</u>
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

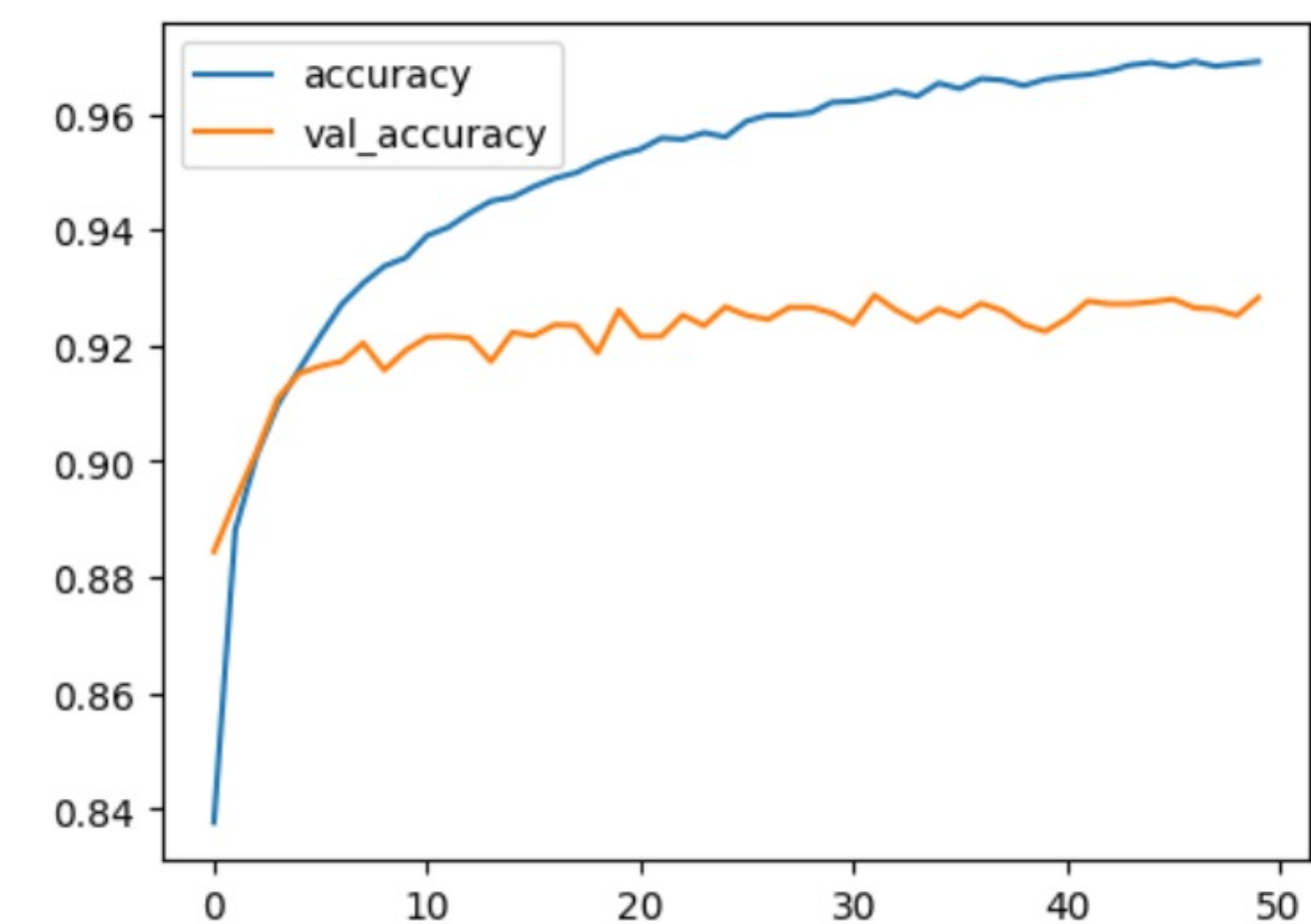
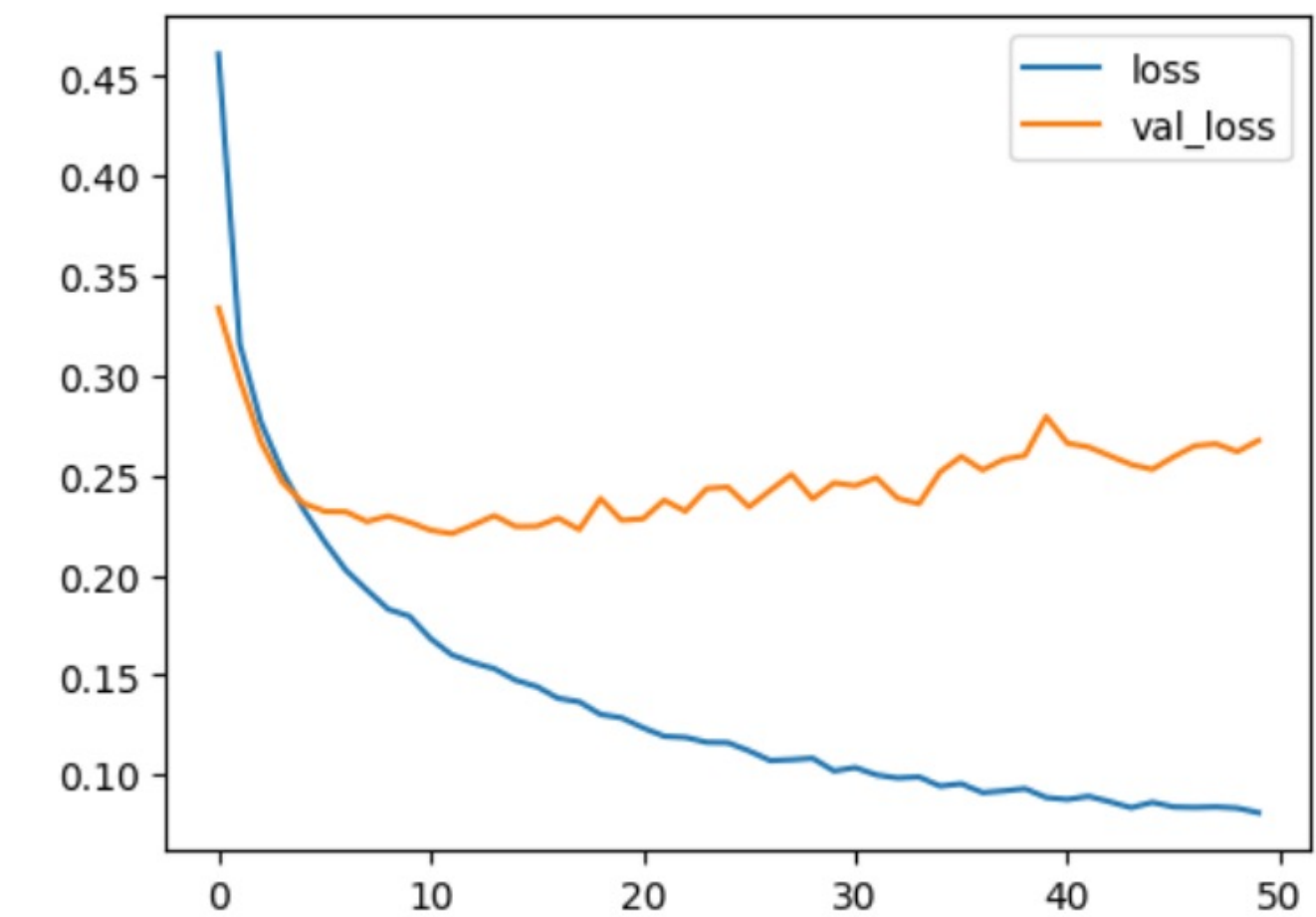
test loss: 0.4966644048690796

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()

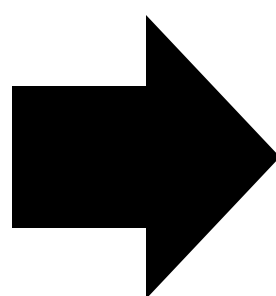
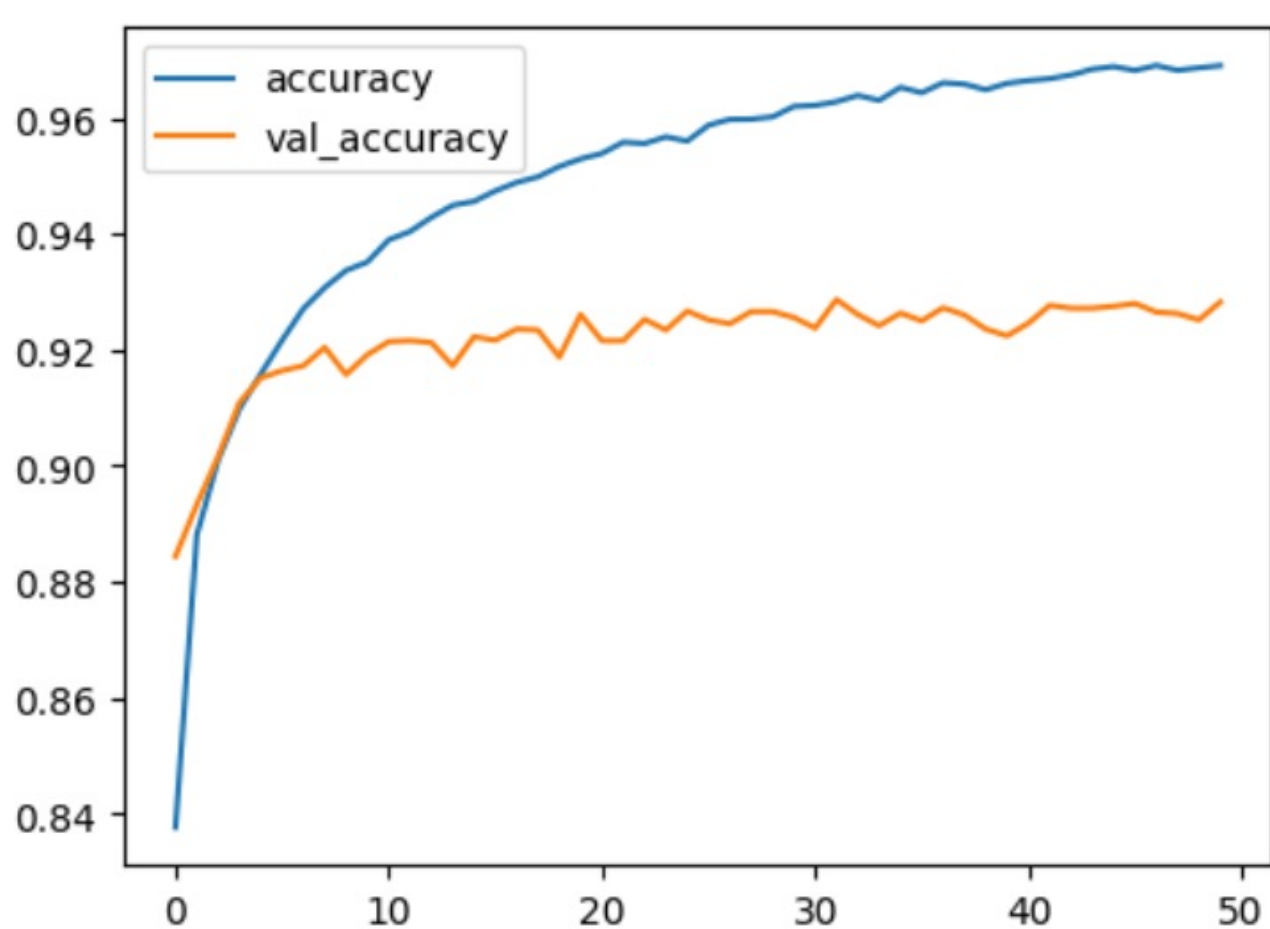
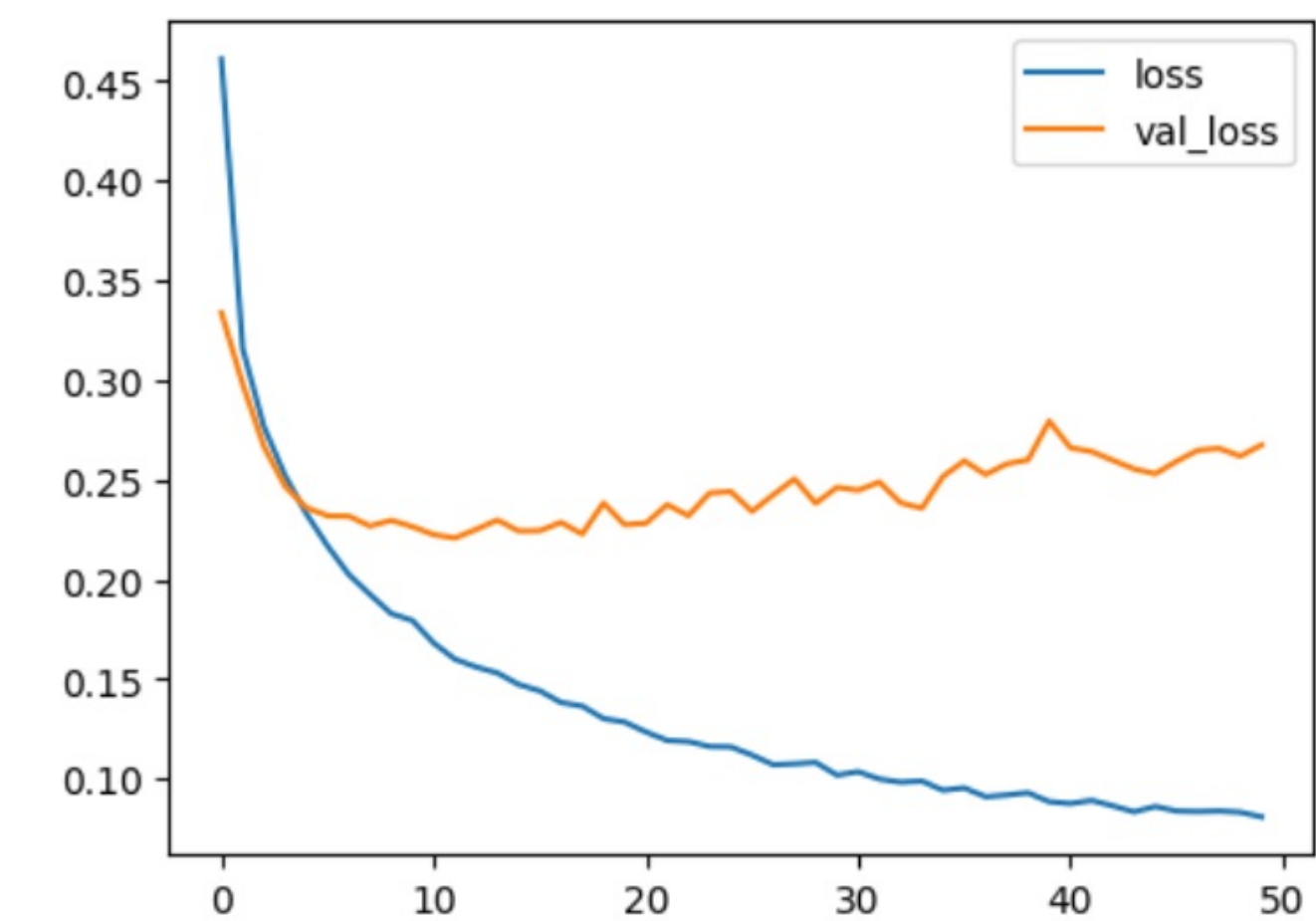
model.add(Conv2D(filters=32, kernel_size=3, strides=1,
                  padding='same', input_shape=(28, 28, 1), activation='relu'))
model.add(Conv2D(filters=64, kernel_size=3, strides=1, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.5))

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))

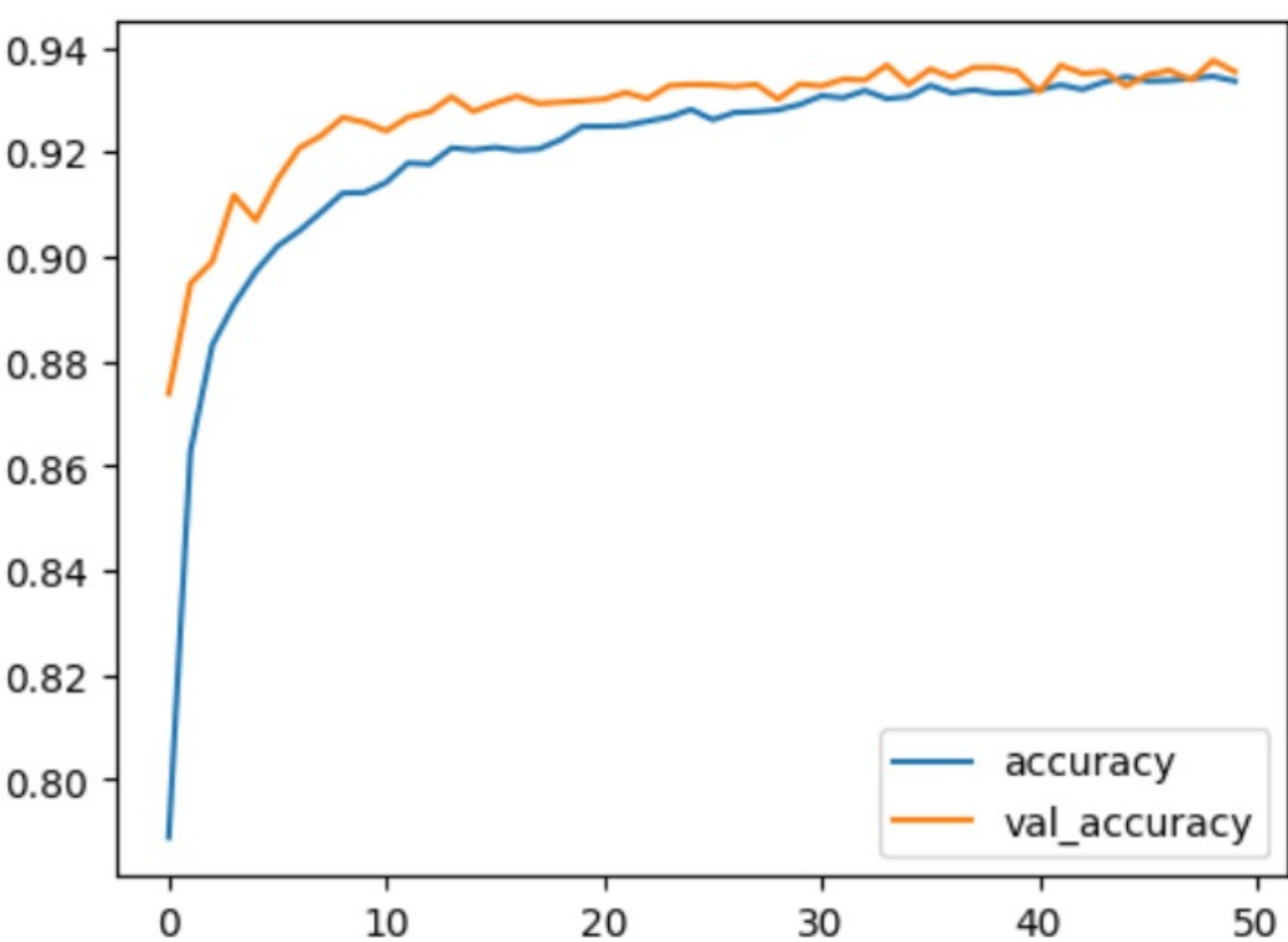
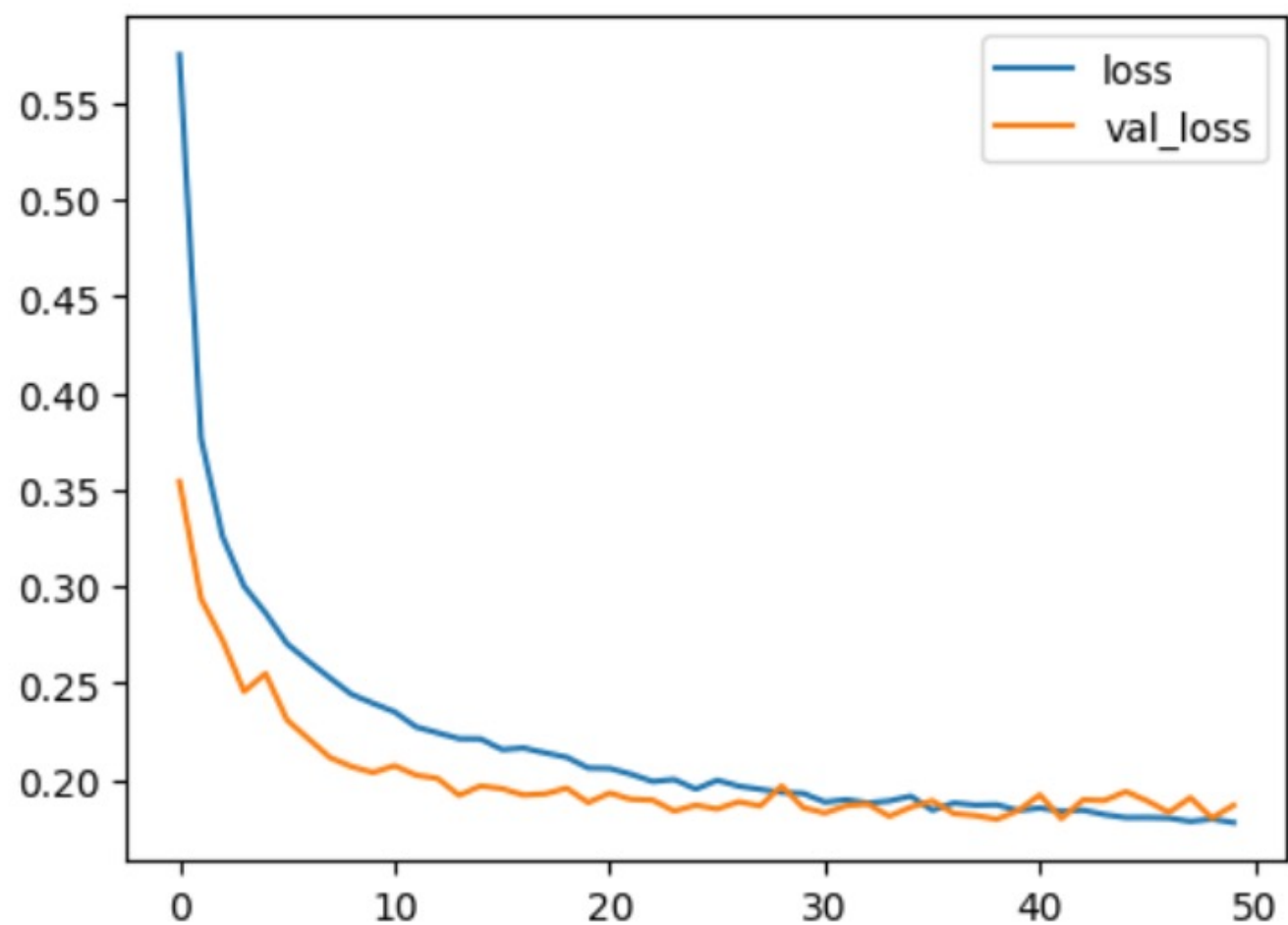
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['accuracy'])
model.summary()
```

```
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: 0.9290000200271606



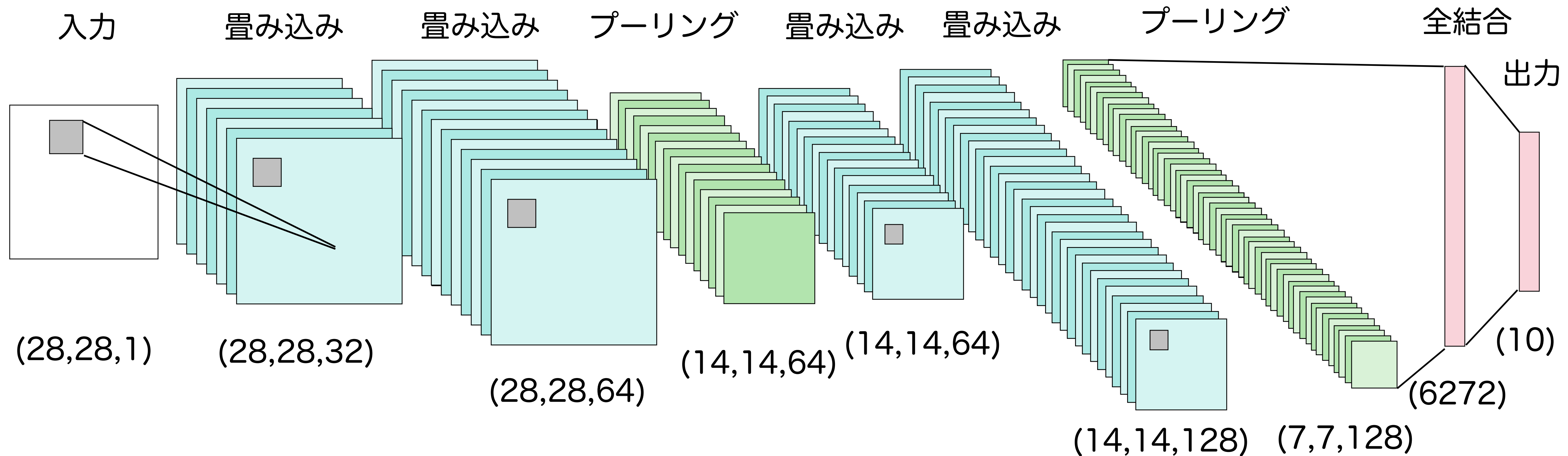

```

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))

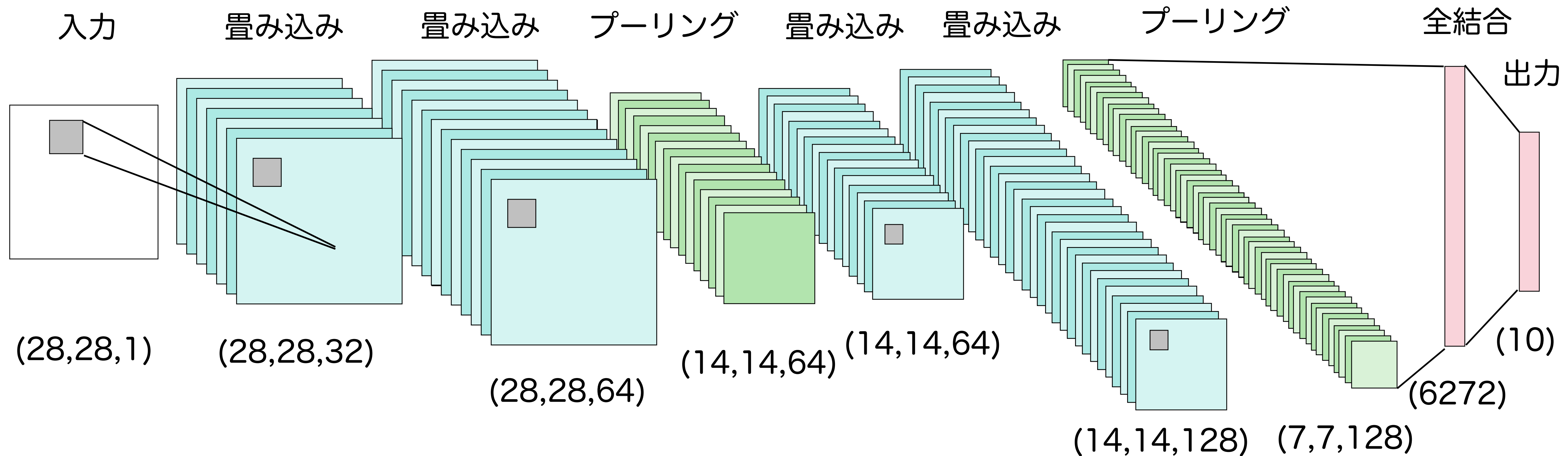
model.add(Conv2D(filters=64,kernel_size=3,strides=1, padding='same',activation='relu'))
model.add(Conv2D(filters=128,kernel_size=3,strides=1,padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.5))

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”kada6.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