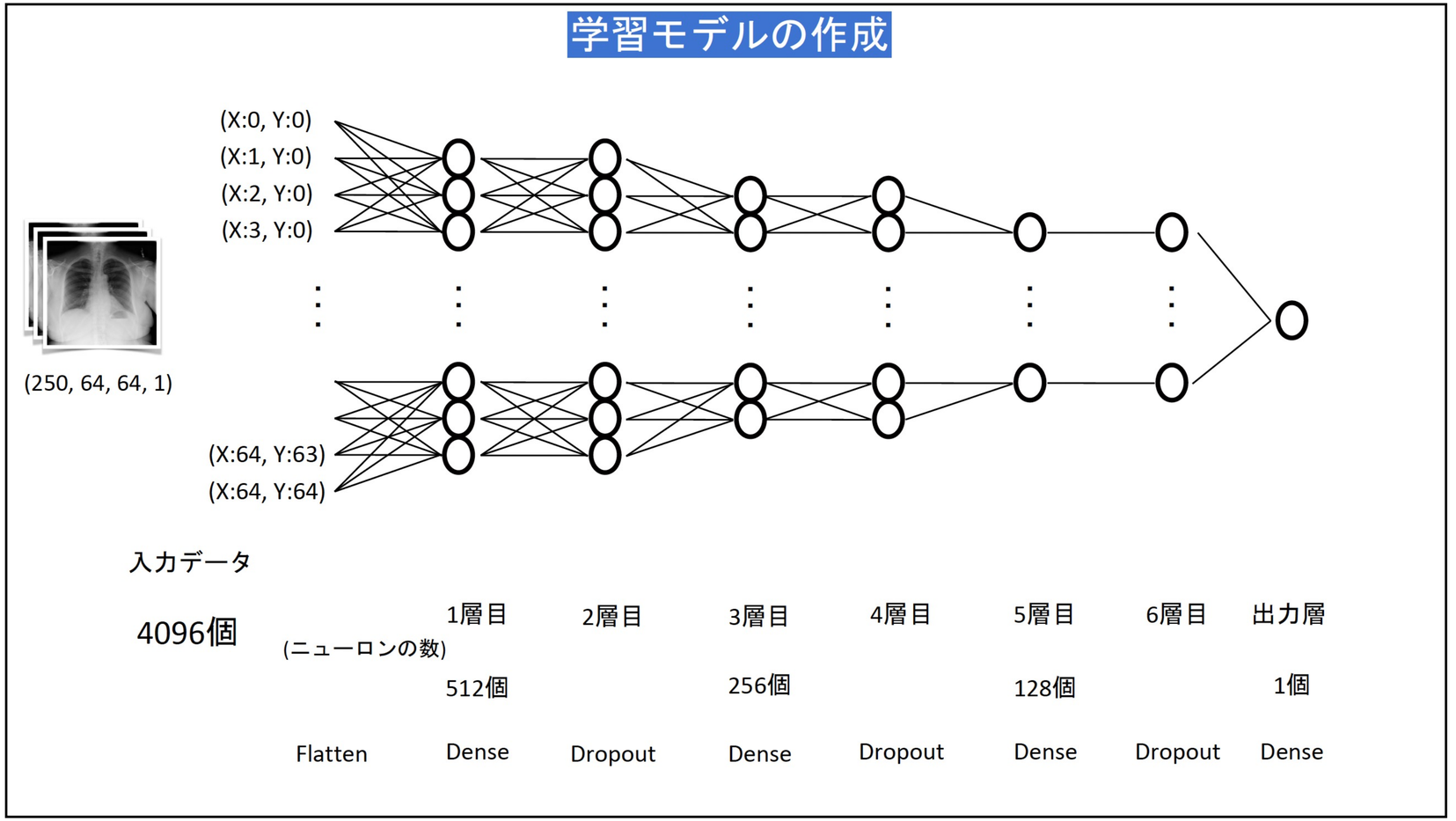
医療とAI・ビッグデータ応用
CNN

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。

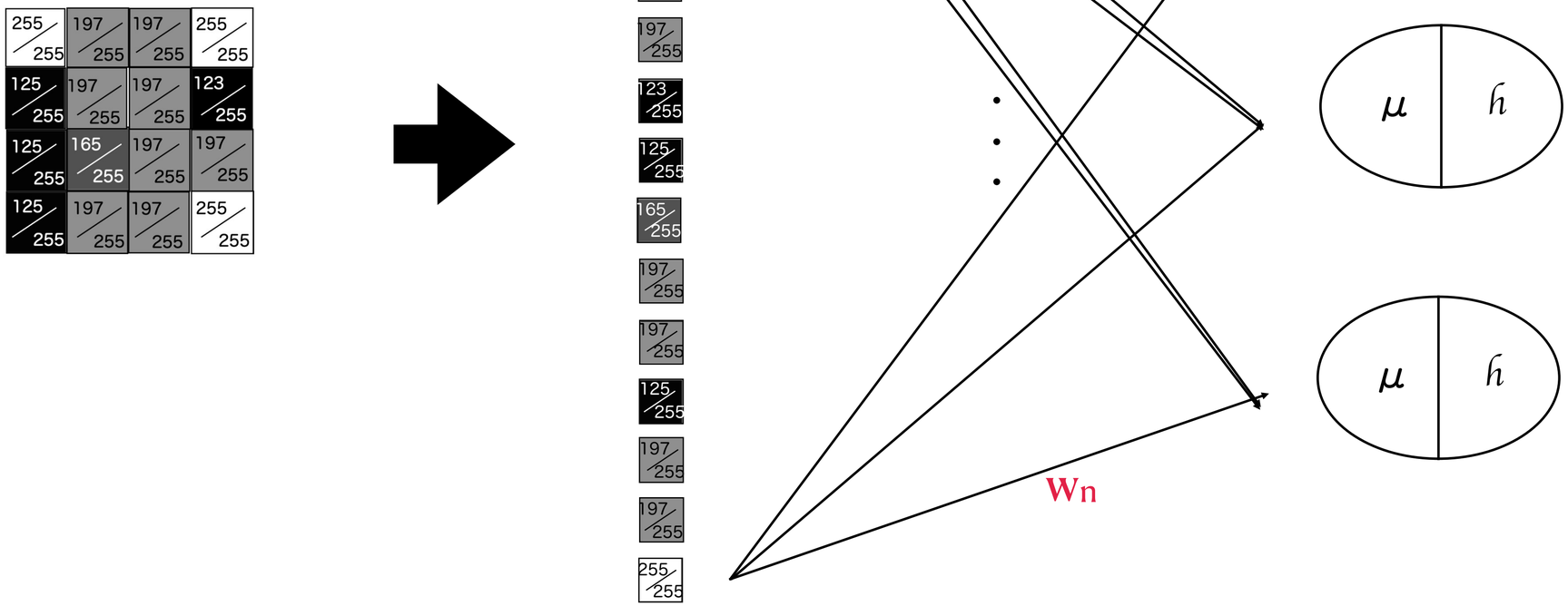
統合教育機構
須藤毅顕

前回までの深層学習は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。

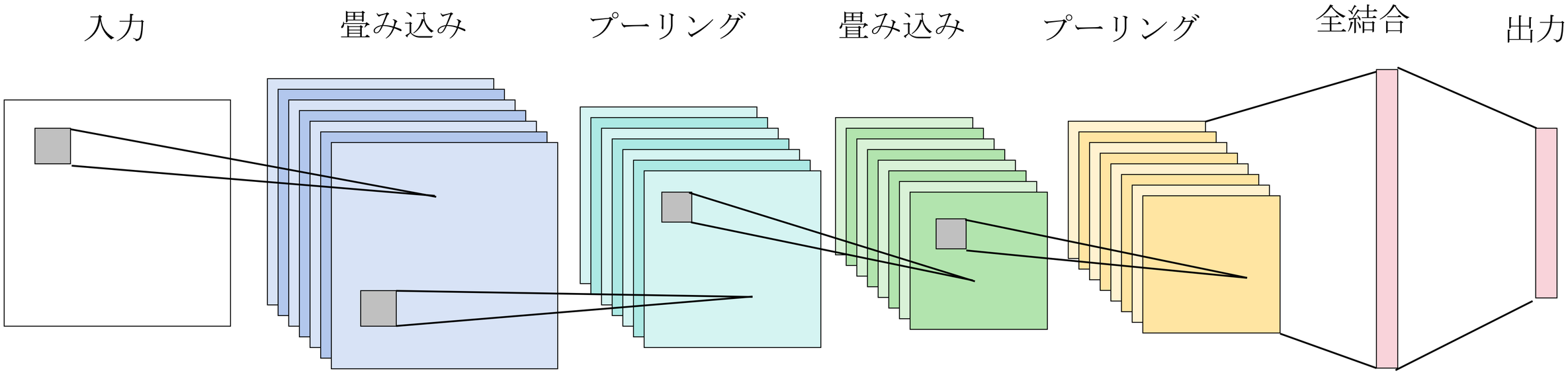
MLPでは画像サイズを１次元にして入力する→画像サイズ分の重みが存在



サイズが大きいほど調整する重みが増えてしまう

CNN(畳み込みニューラルネットワーク)

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



```
from tensorflow.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
(枚数、縦、横、色の数)
の4次元で入力する

```
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 tensorflow.keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# 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,
```

.shapeで確認してもいいですが、
spyderの変数エクスプローラーで
形状を確認出来ます

モデルの作成

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

model = Sequential()
model.add(Conv2D(filters=32,kernel_size=(3,3), strides = (1, 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()
```

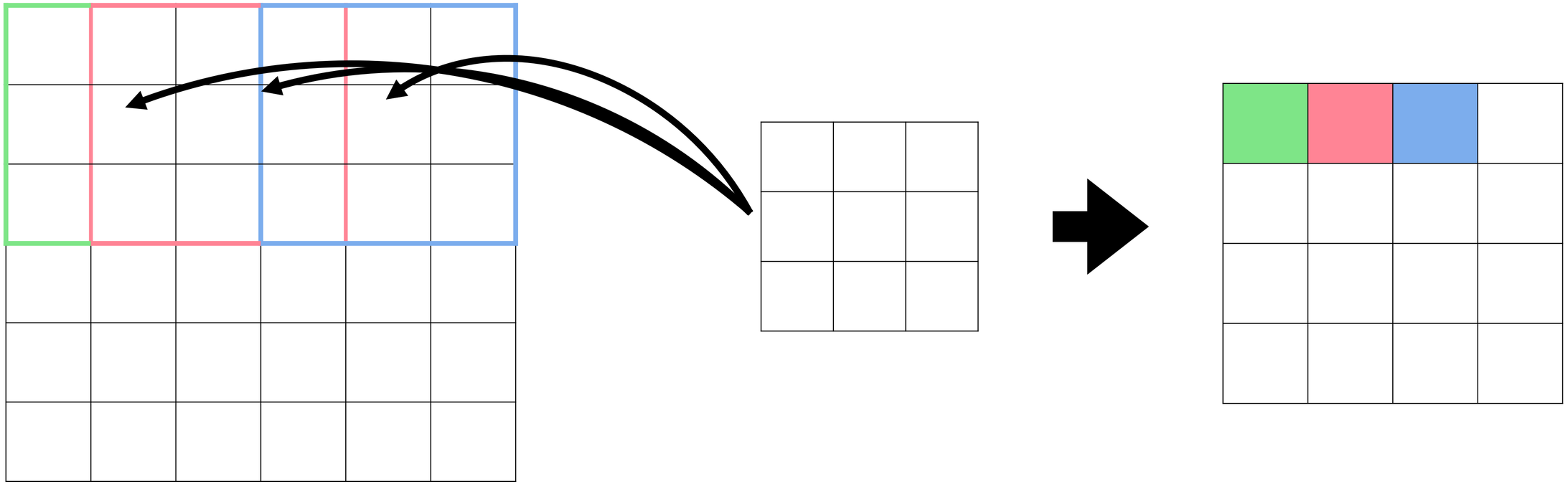
学習の実行

```
result = model.fit(x_train, y_train, epochs = 15, batch_size = 64, verbose = 1, validation_split=0.2, shuffle=True)
```

```
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

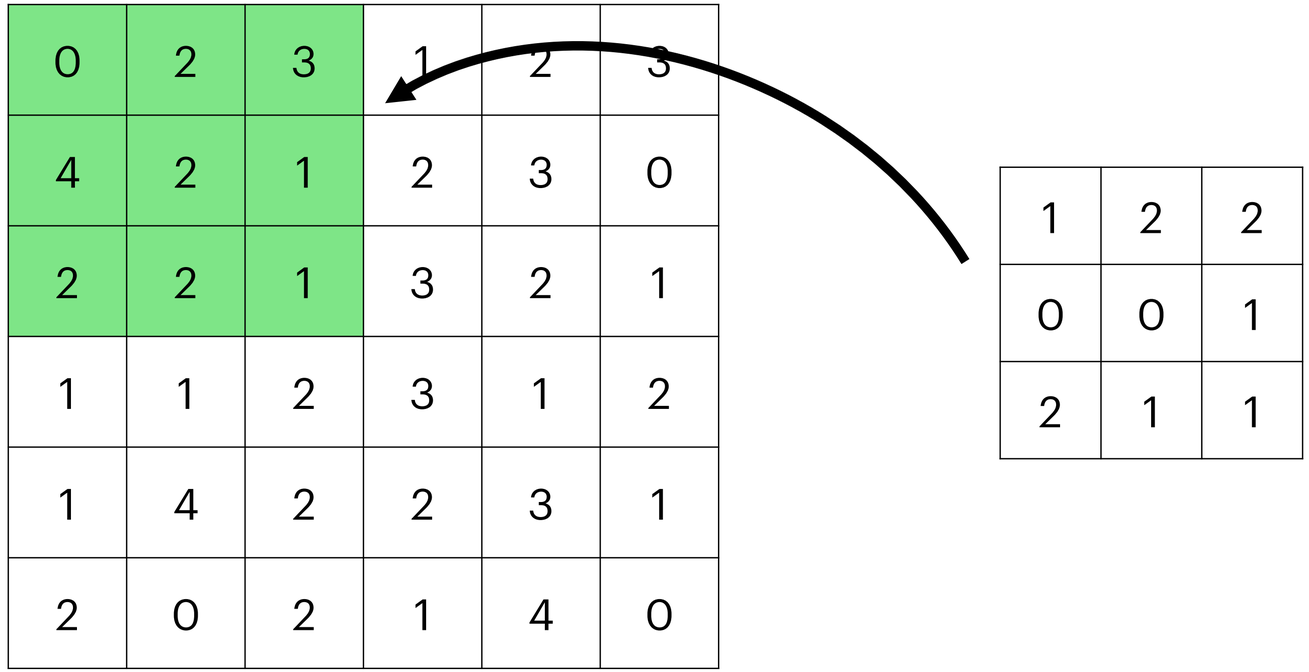
入力層

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

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

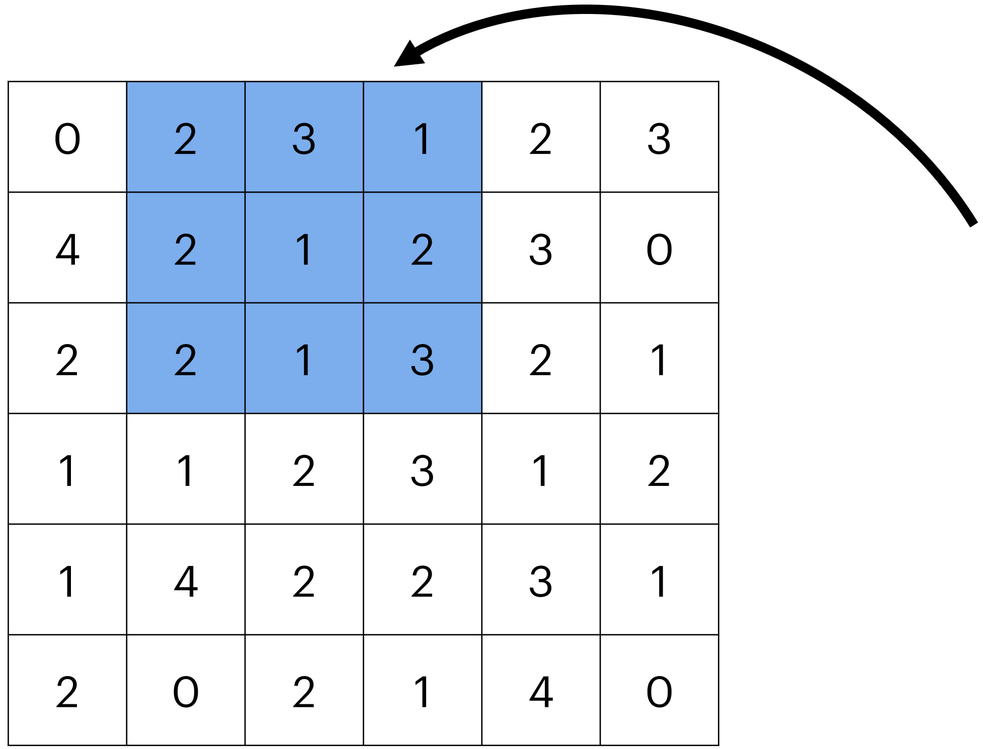
0×1	2×2	2×3
4×0	2×0	1×1
2×2	2×1	1×1

全ての積和を計算する

$$0\times1+2\times2+2\times3+4\times0+2\times0+1\times1+2\times2+2\times1+1\times1=18$$

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

2×1	3×2	1×2
2×0	1×0	2×1
2×2	1×1	3×1

全ての積和を計算する

$$2\times1+3\times2+1\times2+2\times0+1\times0+2\times1+2\times2+1\times1+3\times1=20$$

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

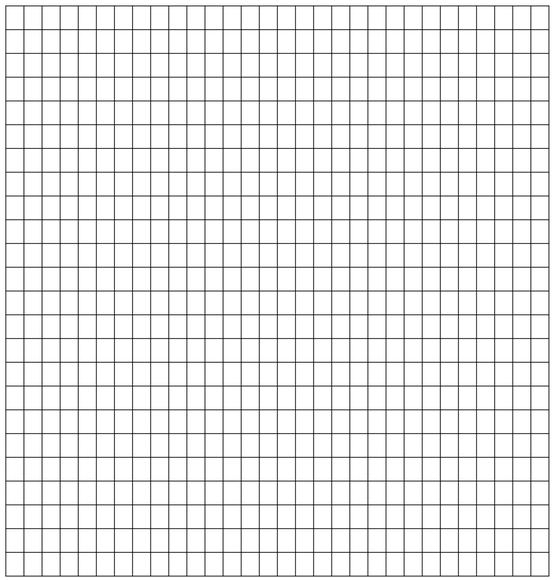
特徴マップ

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

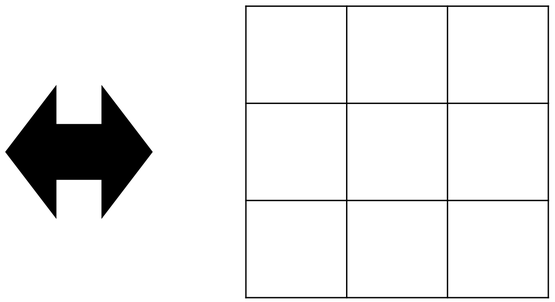
filters = 出力する特徴マップの数

input_shape = MLPと同様に入力層の形

kernel_size = カーネルの大きさ

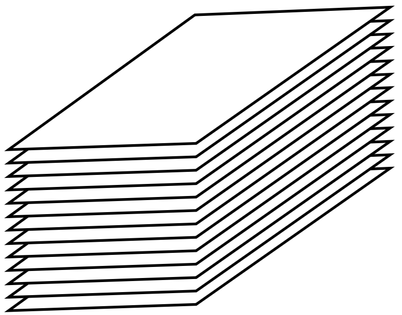


32 × 32



3 × 3

カーネルが32枚存在



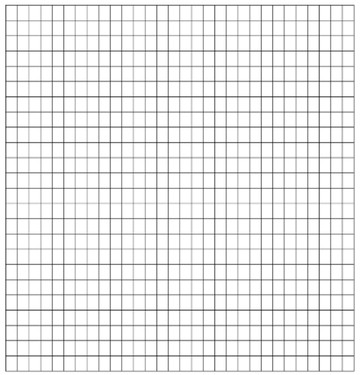
32

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

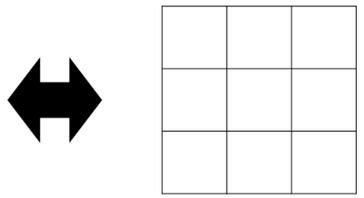
input_shape = (32, 32, 1)

白黒(1チャンネル)

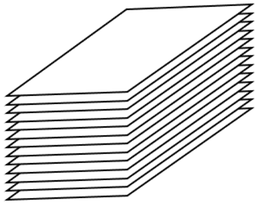
0~255で黒の濃さを表現



28 × 28



3 × 3



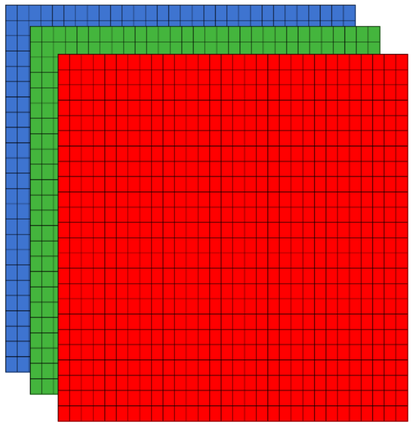
32

input_shape = (32, 32, 3)

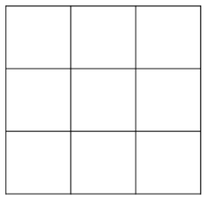
カラー(3チャンネル)

Ex)RGB

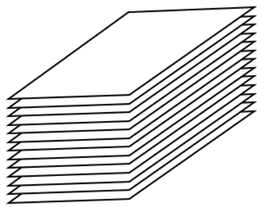
Red:0~255
Green:0~255
Blue:0~255
で色を表現



28 × 28



3 × 3



32

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

strides = カーネルをずらす幅

strides = 1
1つずつずれる

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
2つずつずれる

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

→

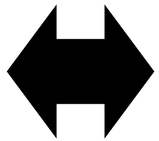
18	19

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

padding = データの端をどう扱うか

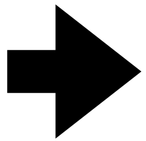
0	2	3	1	2	3
4	2	1	2	3	0
2	2	1	3	2	1
1	1	2	3	1	2
1	4	2	2	3	1
2	0	2	1	4	0

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)

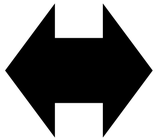
そのままだと特徴マップのサイズは入力層より小さくなる

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

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

特徴マップ(4×4)

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

モデルの作成

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

model = Sequential()
model.add(Conv2D(filters=32,kernel_size=(3,3), strides = (1, 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
_________________________________________________________________
```

Flatten()は3次元を
1次元に変換する(フラットにする)

モデルの作成

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

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

学習の実行

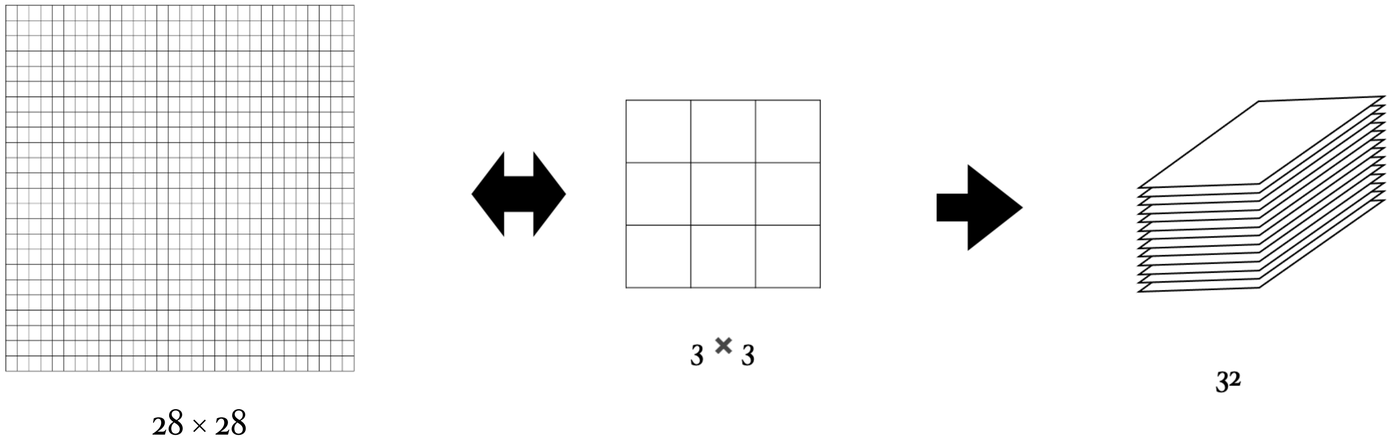
```
result = model.fit(x_train, y_train, epochs = 15, batch_size = 64, verbose = 1, validation_split=0.2, shuffle=True)
```

```
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
```

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

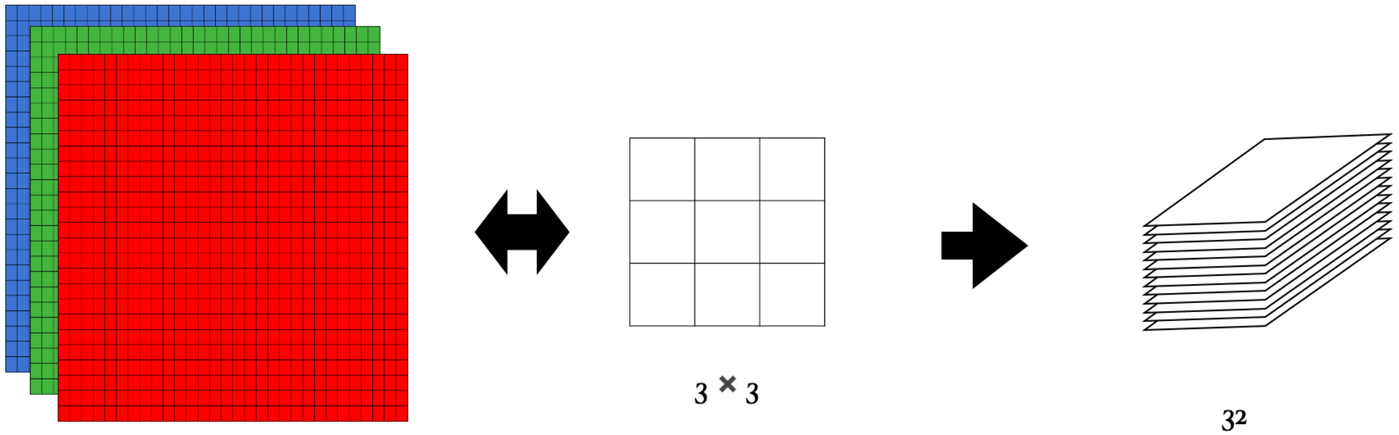
input_shape = (28, 28, 1)

白黒(1チャンネル)



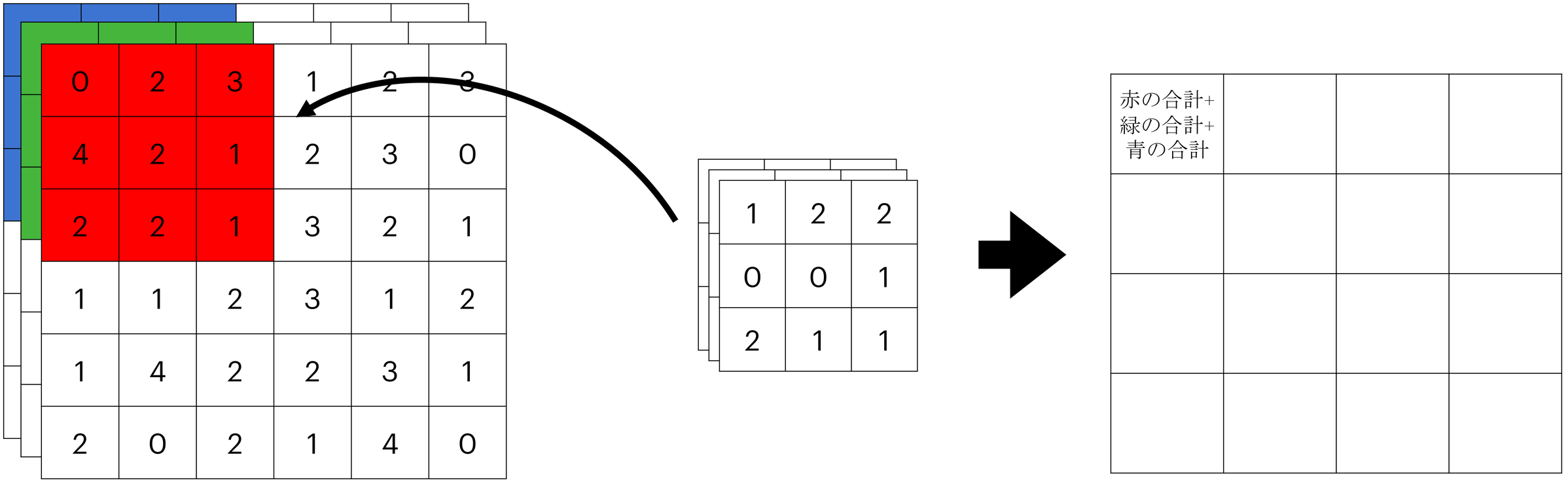
input_shape = (28, 28, 3)

カラー(3チャンネル)



入力層がカラー(3チャンネル)の場合

カラーではそれぞれのチャンネルごとに畳み込みを行い、加算したものが1つのピクセルの値になる。

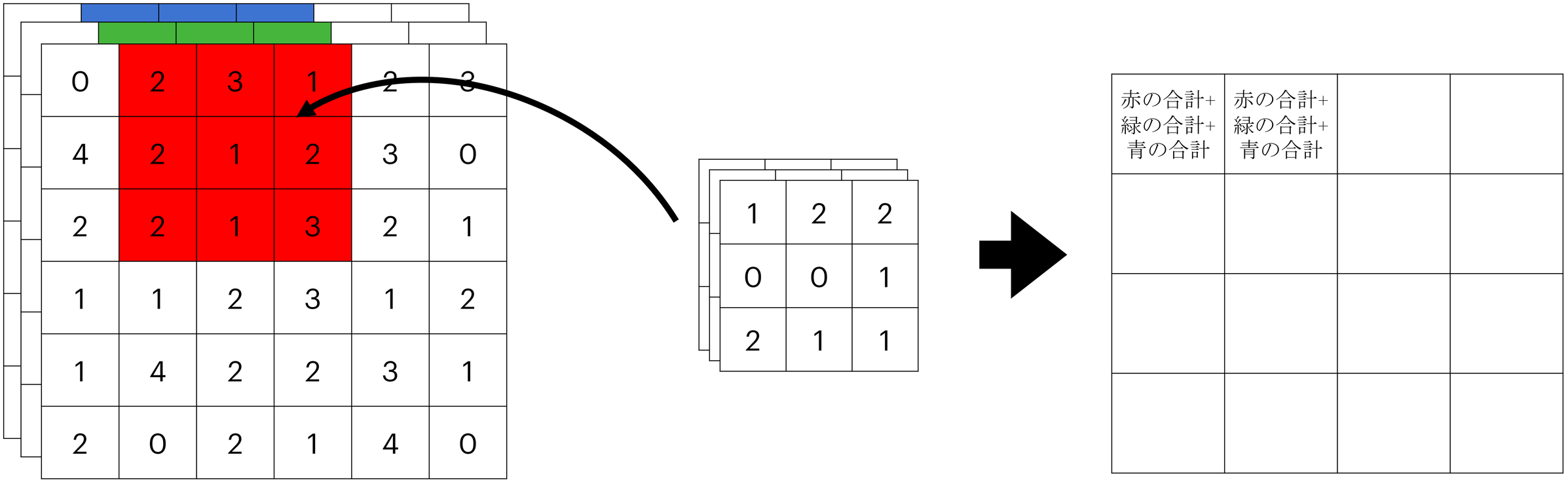


全ての和を計算する

入力層　　カーネル　　特徴マップ

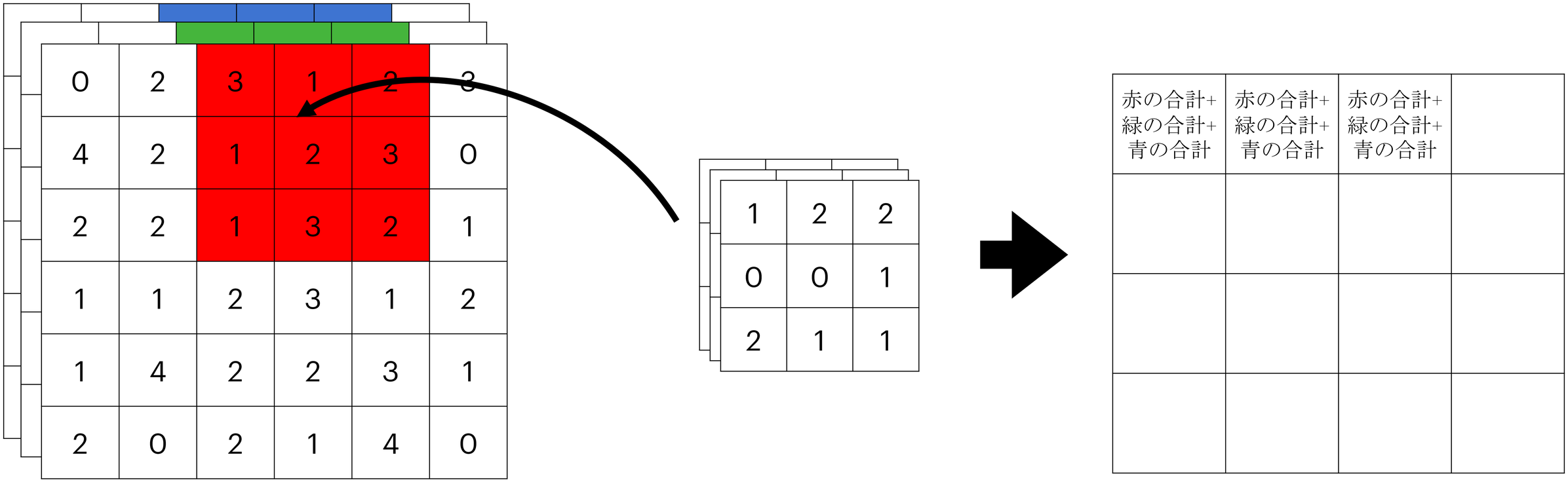
入力層がカラー(3チャンネル)の場合

カラーではそれぞれのチャンネルごとに畳み込みを行い、加算したものが1つのピクセルの値になる。



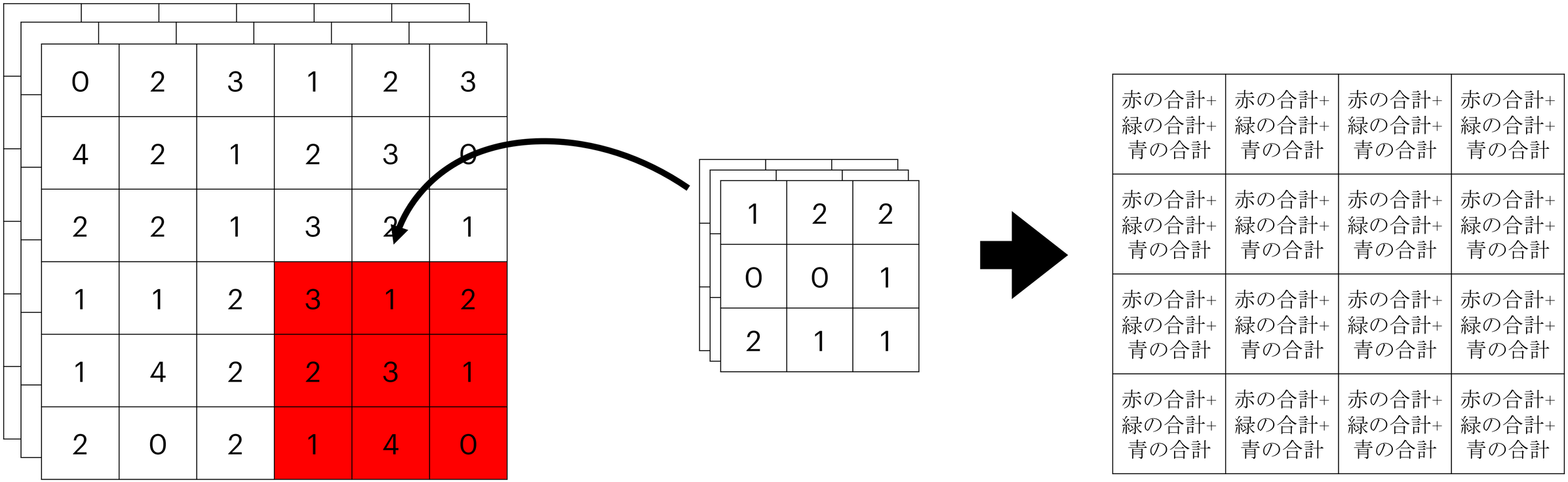
入力層がカラー(3チャンネル)の場合

カラーではそれぞれのチャンネルごとに畳み込みを行い、加算したものが1つのピクセルの値になる。



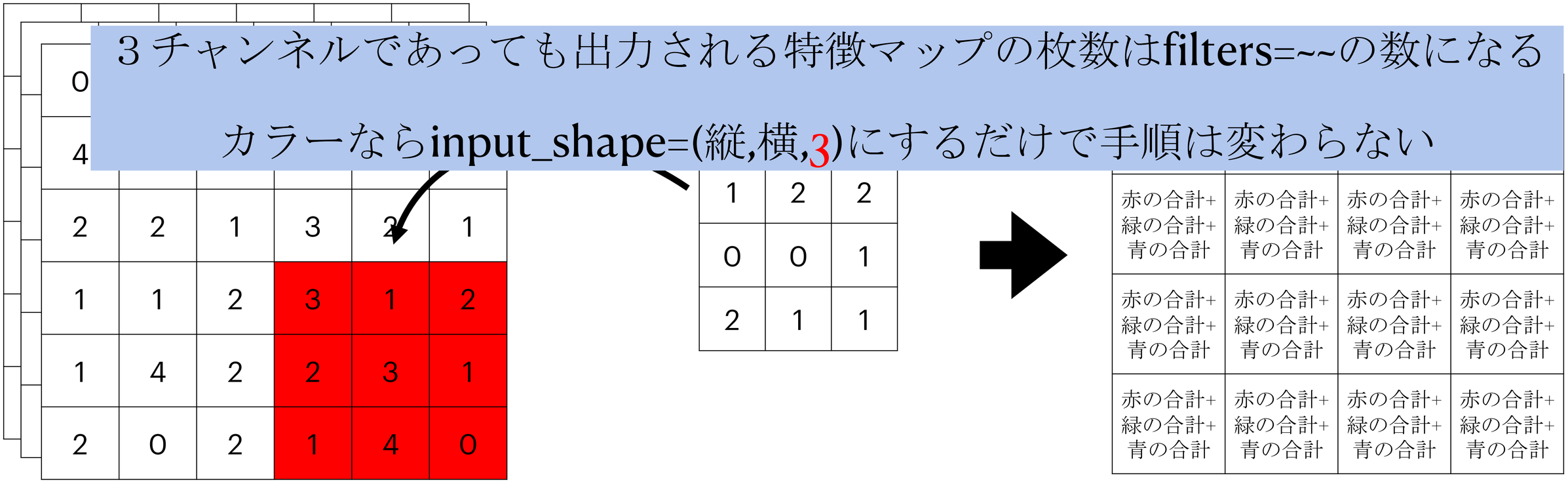
入力層がカラー(3チャンネル)の場合

カラーではそれぞれのチャンネルごとに畳み込みを行い、加算したものが1つのピクセルの値になる。

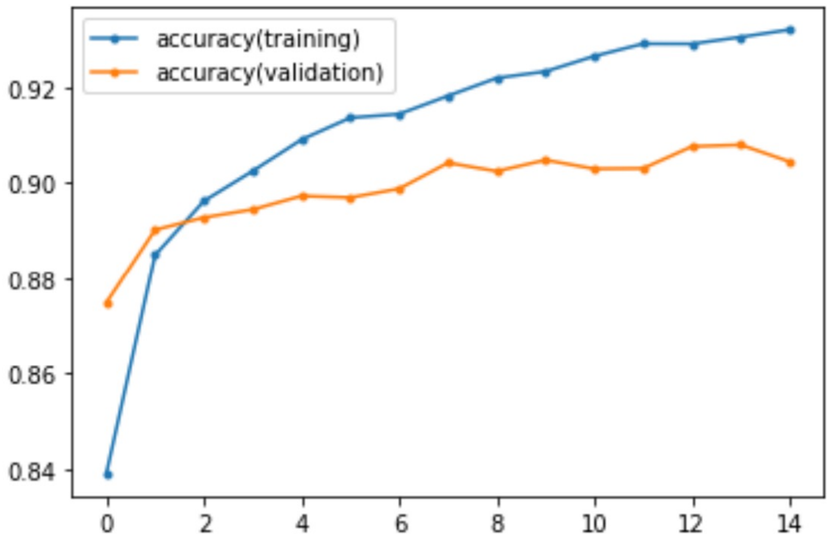


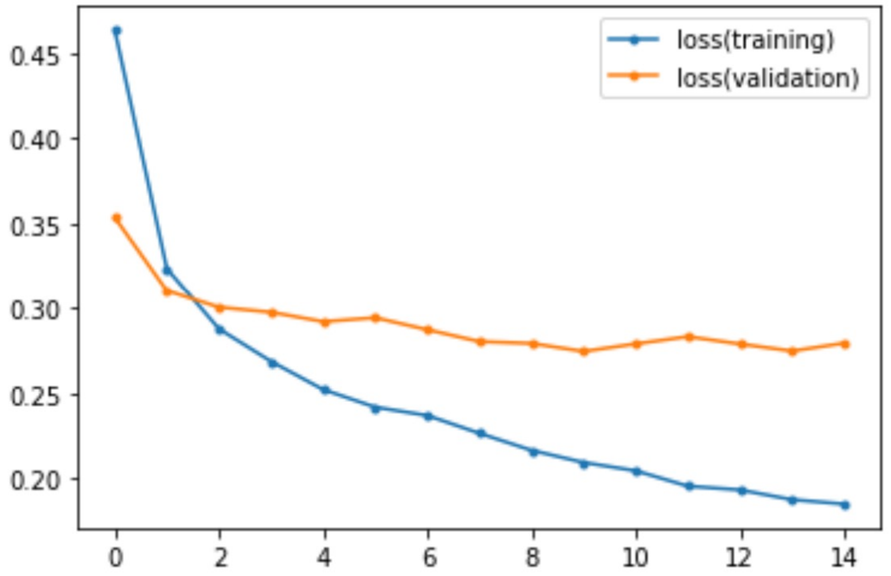
入力層がカラー(3チャンネル)の場合

カラーではそれぞれのチャンネルごとに畳み込みを行い、加算したものが1つのピクセルの値になる。



```
import matplotlib.pyplot as plt
plt.plot(result.history['accuracy'], marker='.', label='accuracy
plt.plot(result.history['val_accuracy'], marker='.', label='val_accuracy')
plt.legend()
plt.show()
plt.plot(result.history['loss'], marker='.', label='loss')
plt.plot(result.history['val_loss'], marker='.', label='val_loss')
plt.legend()
plt.show()
```





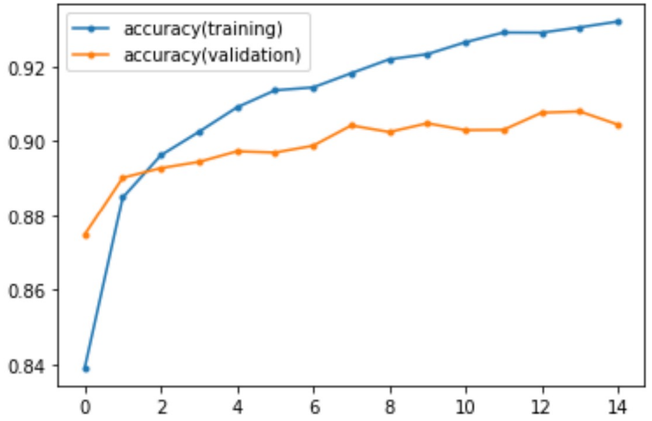
Epoch=15

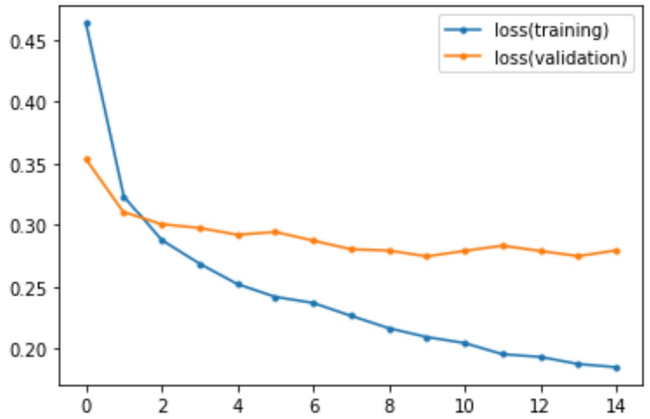
```
score = model.evaluate(x_test, y_test)
print('Test loss:',score[0])
print('Test accuracy:',score[1])
```

Epoch=15

Test loss: 0.2920217216014862
Test accuracy: 0.8999000191688538

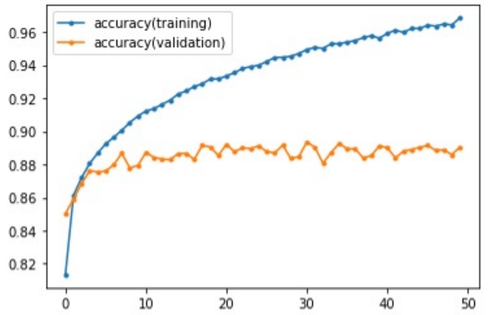
既にMLPよりも精度が良いことが分かる

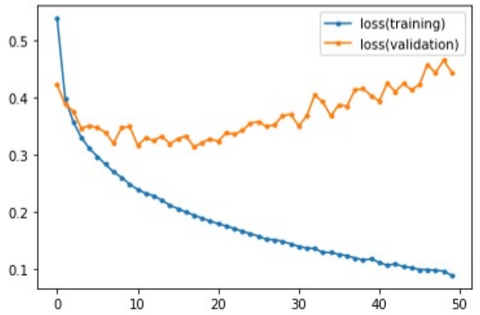




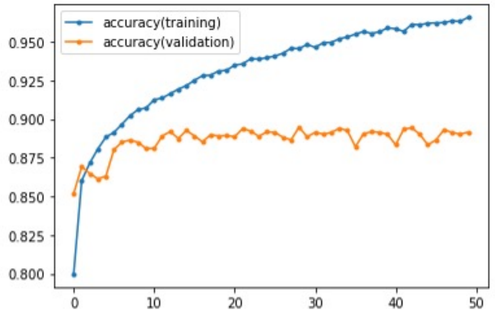
MLP（前回のスライド）

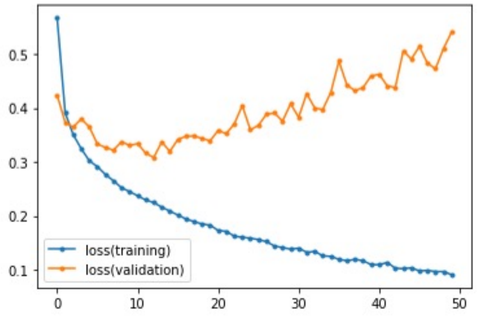
層を増やしても今回のデータでは精度あまり上がっていない





Test loss: 0.47562167048454285
Test accuracy: 0.8906000256538391





Test loss: 0.6009576916694641
Test accuracy: 0.8855999708175659

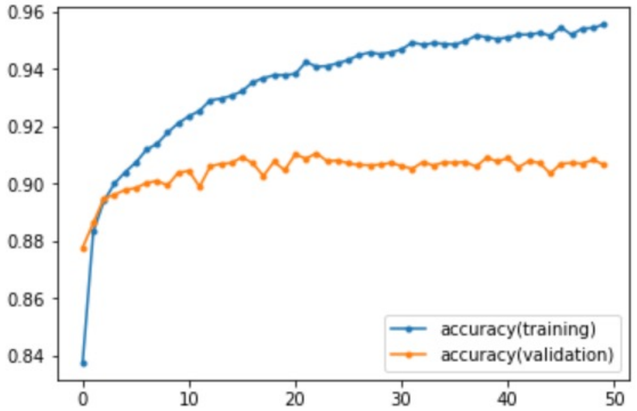
```
score = model.evaluate(x_test, y_test)
print('Test loss:',score[0])
print('Test accuracy:',score[1])
```

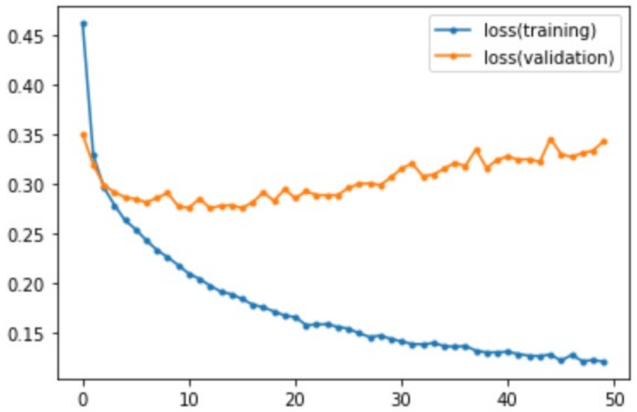
Epoch=50

Test loss: 0.344457149505615 23

Test accuracy: 0.9031999707221985

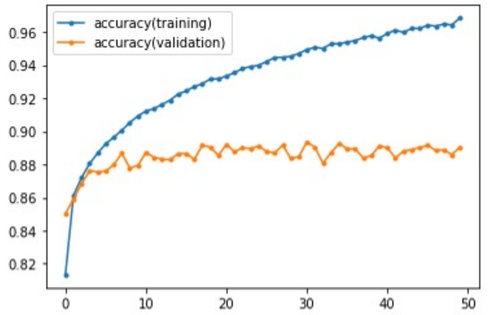
既にMLPよりも精度が良いことが分かる

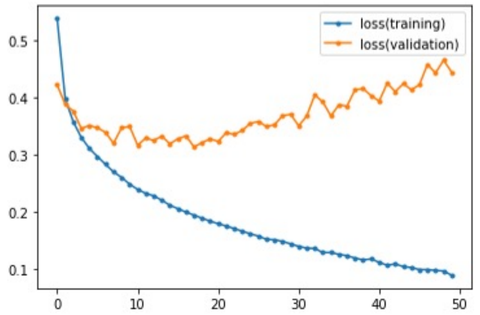




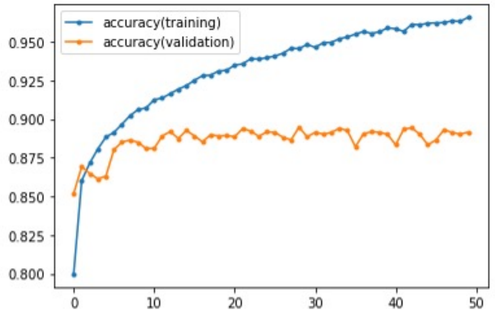
MLP（前回のスライド）

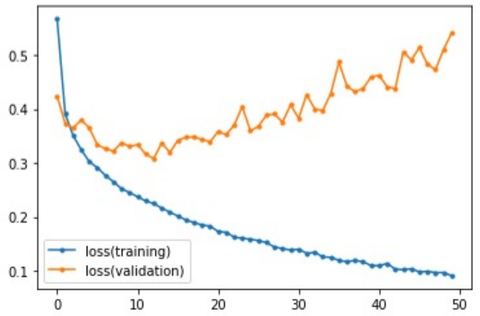
層を増やしても今回のデータでは精度あまり上がっていない





Test loss: 0.47562167048454285
Test accuracy: 0.8906000256538391





Test loss: 0.6009576916694641
Test accuracy: 0.8855999708175659

畳み込み層の追加

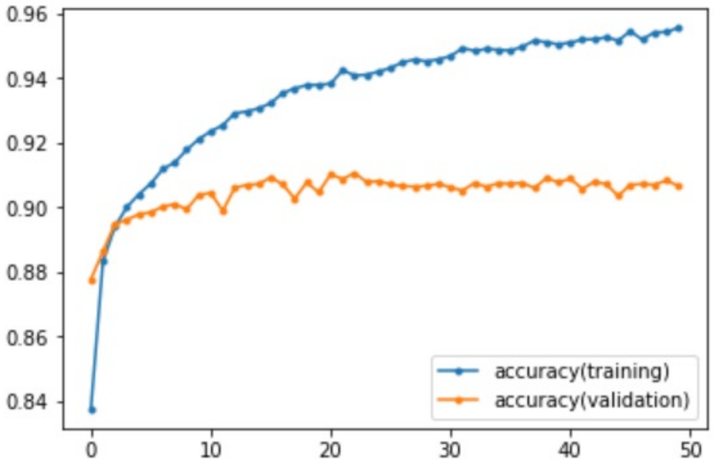
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv2D, Flatten, MaxPooling2D

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

model.add(Conv2D(filters=64,kernel_size=(3,3), strides = (1, 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()
```

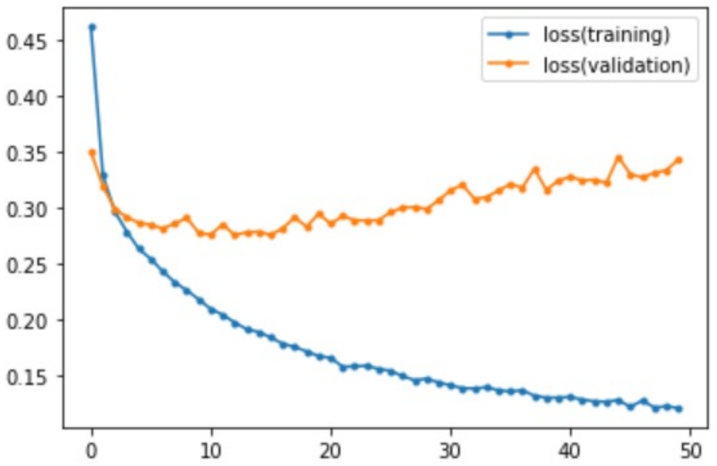
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
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: 0.344457149505615 23

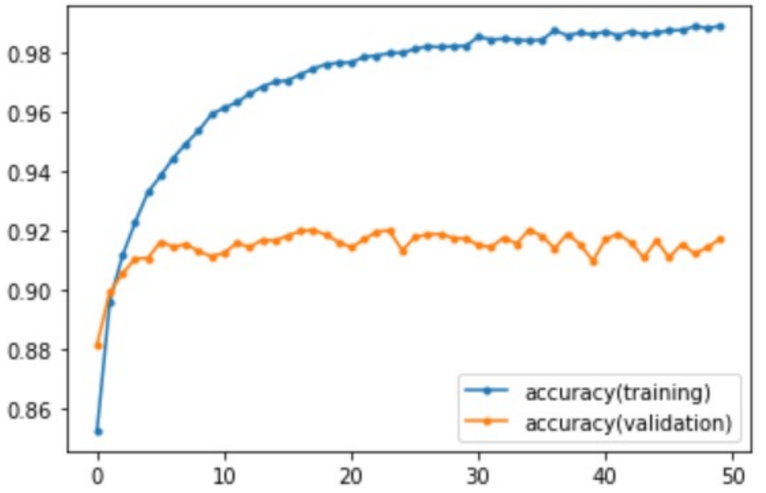
Test accuracy: 0.9031999707221985

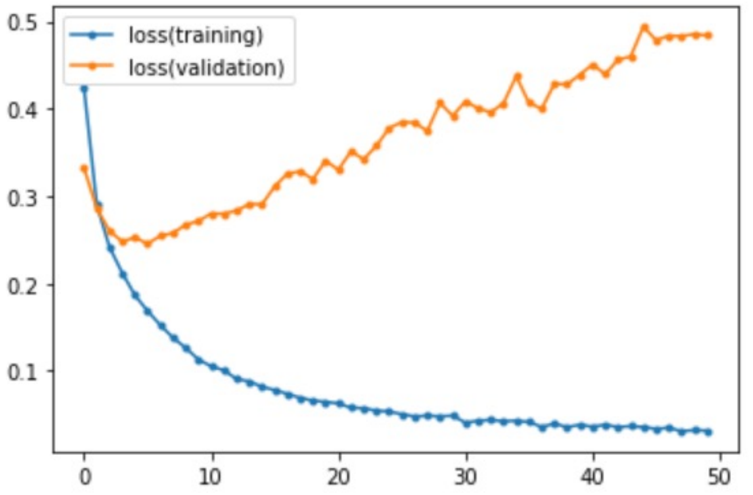




Test loss: 0.4930589497089386

Test accuracy: 0.9175000190734863





プーリング層の追加

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

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

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

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()
```

プーリング層の追加

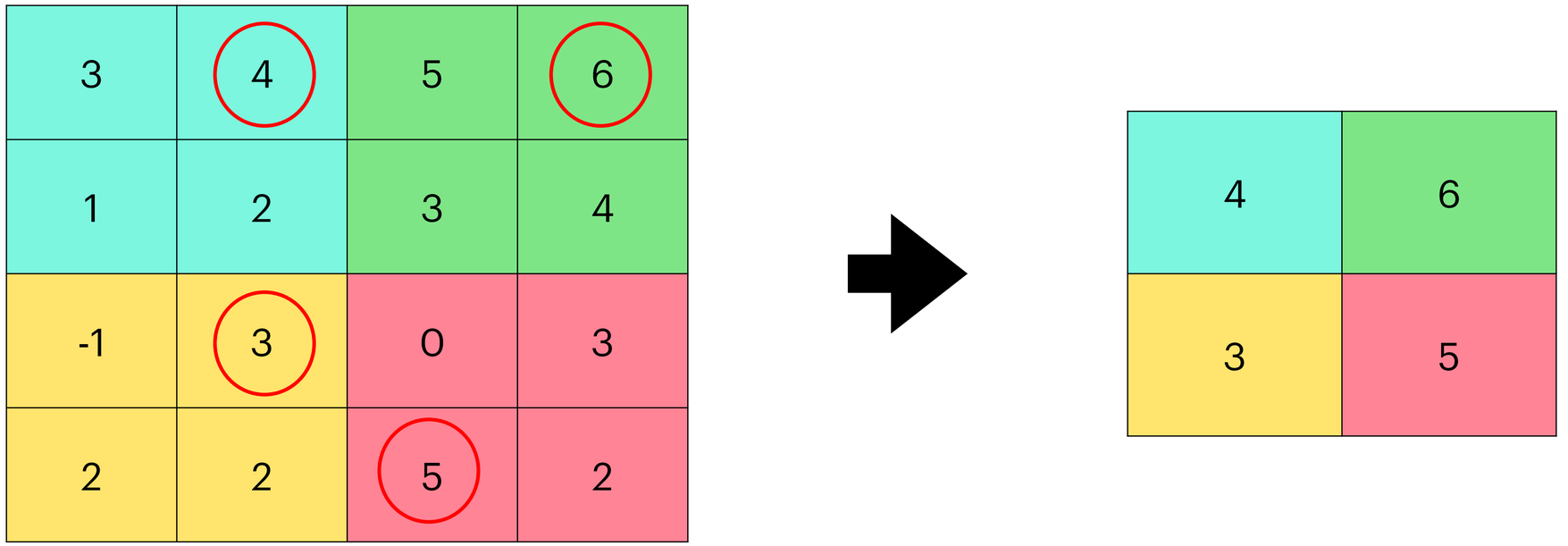
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv2D, Flatten, MaxPooling2D

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

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

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
_________________________________________________________________
```

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。

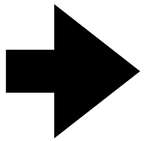


プーリング層

データを縮小する方法

平均プーリング：入力データを小さな領域に分割し、各領域の平均値をとってくることで、データを縮小する。

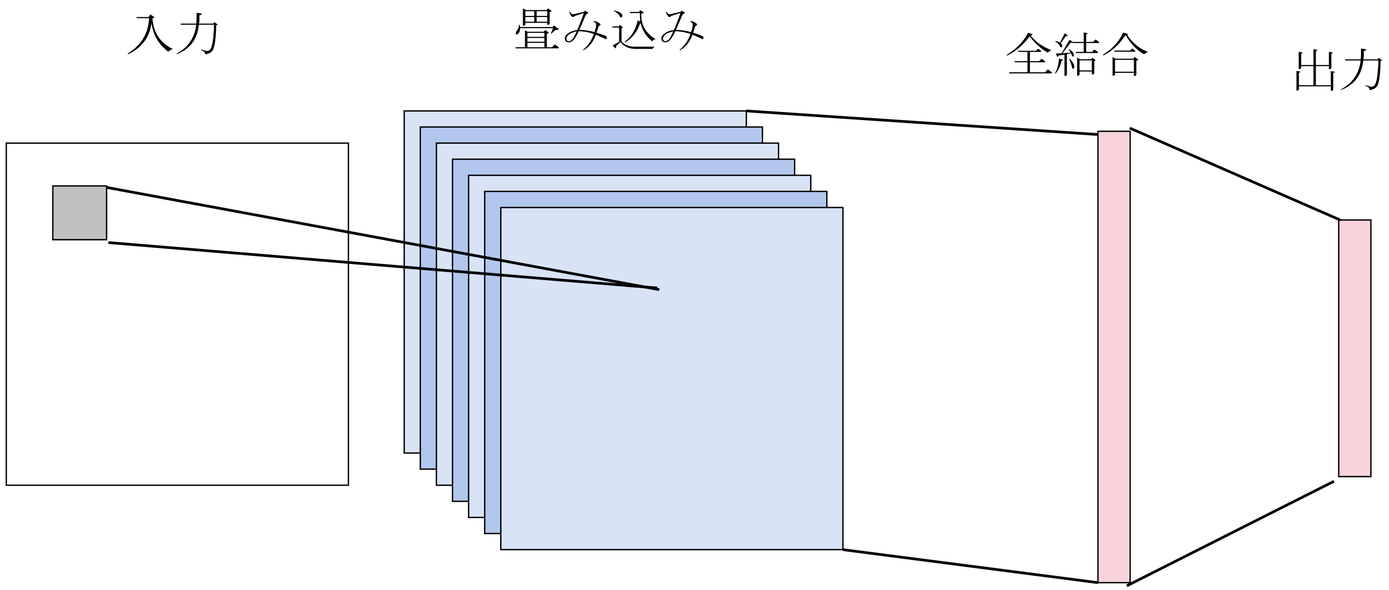
3	4	5	6
1	2	3	4
-1	3	0	3
2	2	5	2



(3 + 4 + 1 + 2) / 4 = 2.5	(5 + 6 + 3 + 4) / 4 = 4.5
((-1) + 3 + 2 + 2) / 4 = 1.5	(0 + 3 + 5 + 2) / 4 = 2.5

パラメータの数

畳み込み層のパラメータの数：
(カーネルのサイズ×前の層のチャンネル(or特徴マップ)数+1)×特徴マップ数

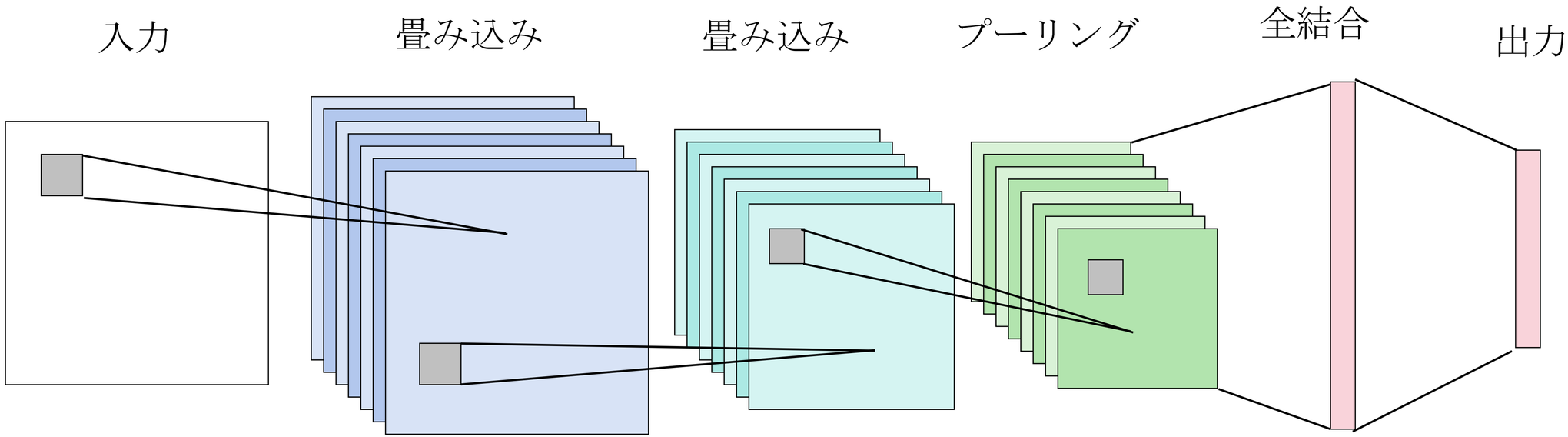



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
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
_________________________________________________________________
```

畳み込み層　((3×3)×1+1)×32=320
出力層　(25088+1)×10=250890

3×3のカーネルが入力層の数だけ
ある。それにバイアスを足したものが
32個の特徴マップだけパラメータがある

パラメータの数



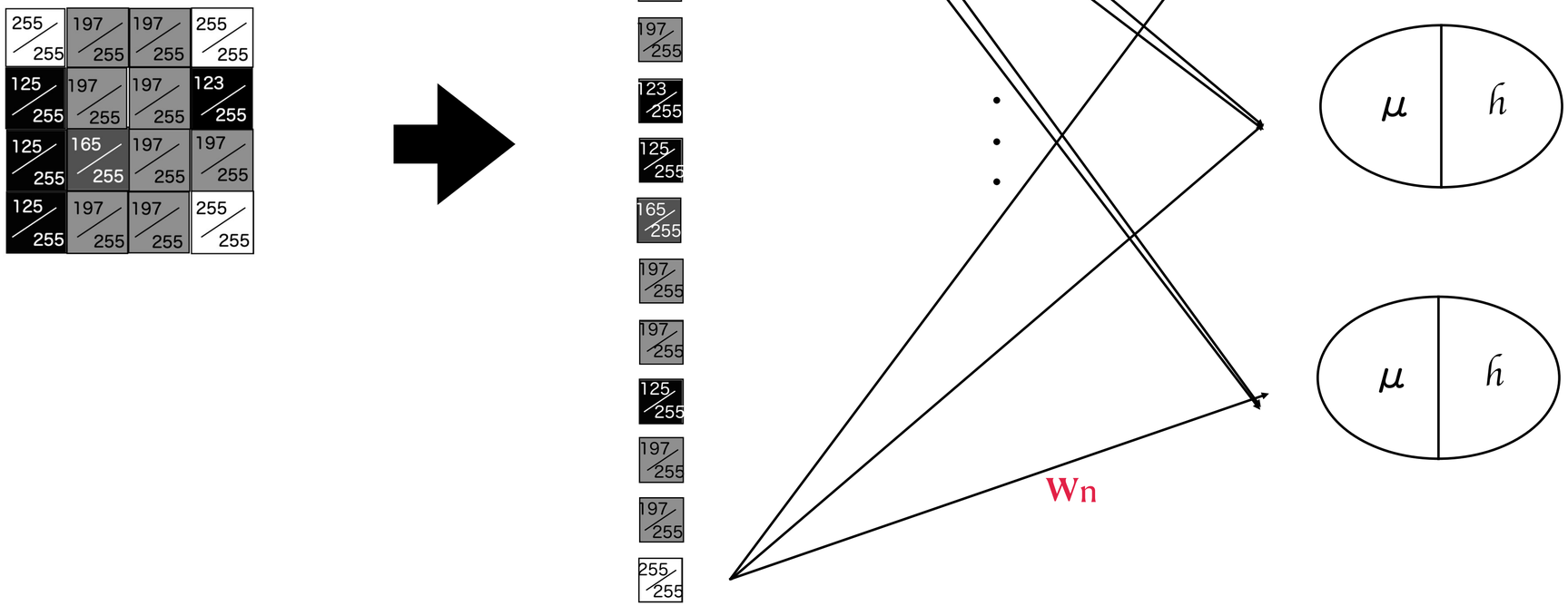
畳み込み層1　((3×3)×1+1)×32=320
畳み込み層2　((3×3)×32+1)×64=18496
出力層　(12544+1)×10=125450

画像のサイズに影響しない

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

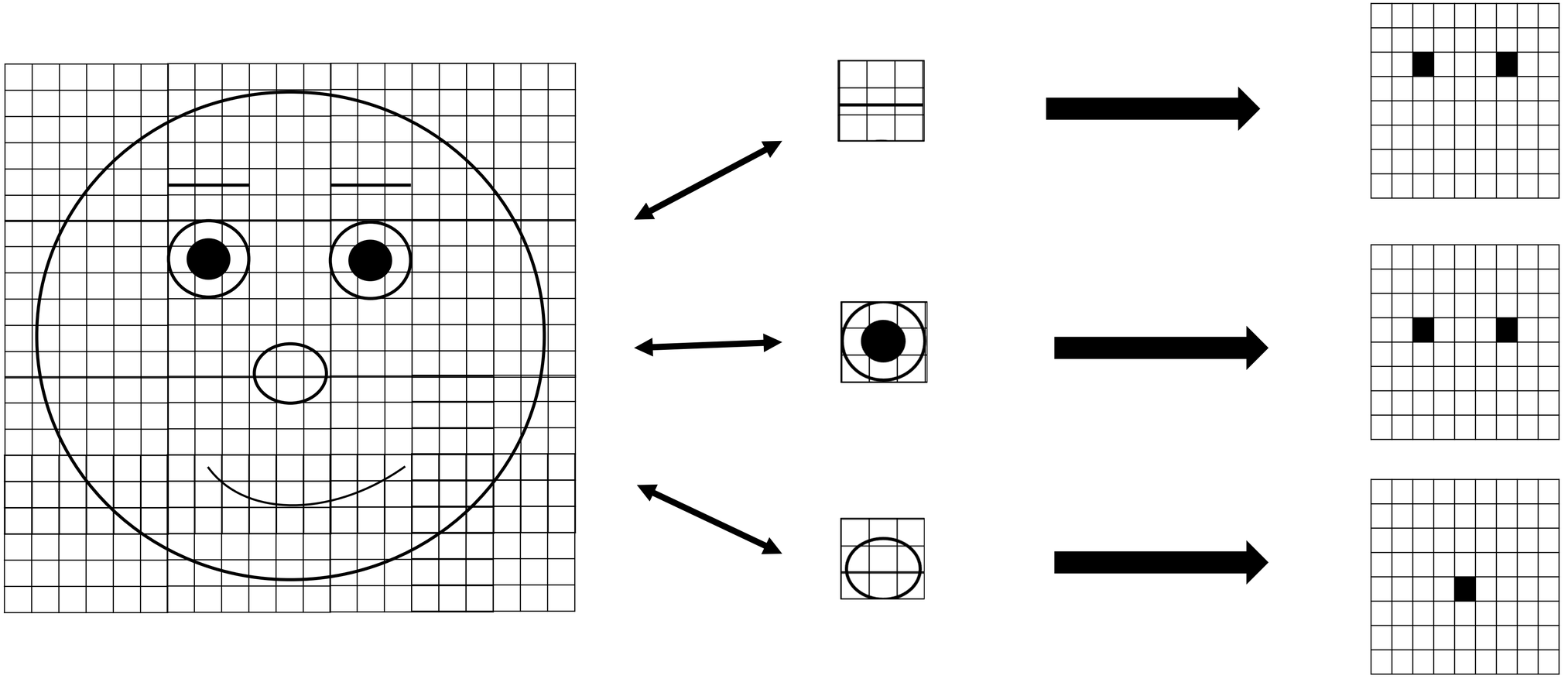
MLPでは画像サイズを１次元にして入力する→画像サイズ分の重みが存在



サイズが大きいほど調整する重みが増えてしまう

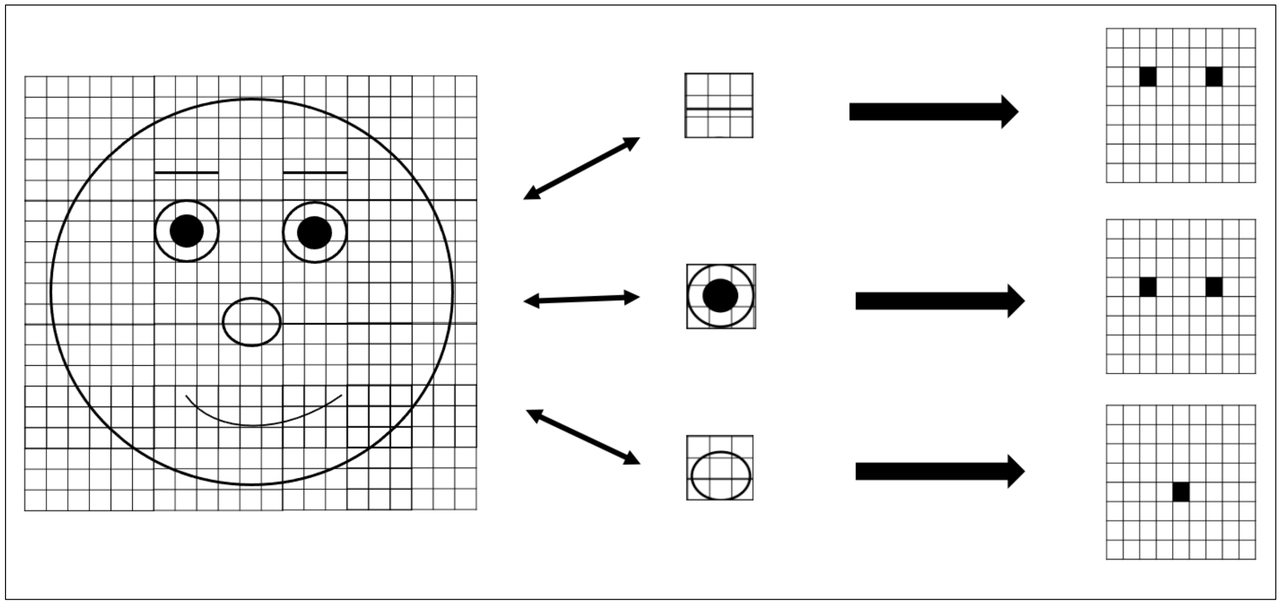
畳み込み層は何をしているのか

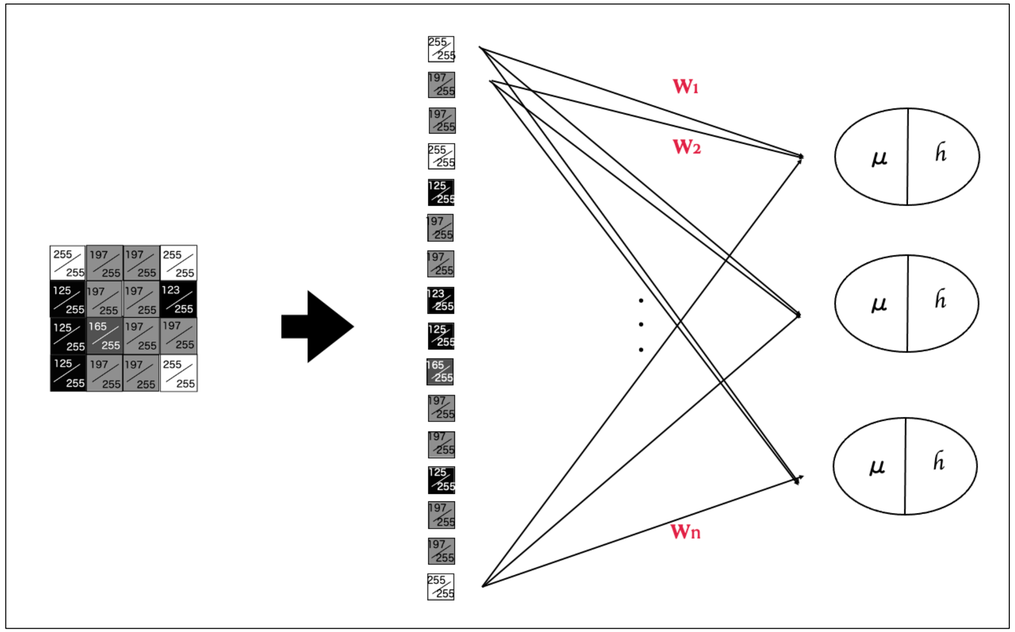
カーネル≒認識パターン



複数のパターンを使って入力画像の特徴を抽出していく作業

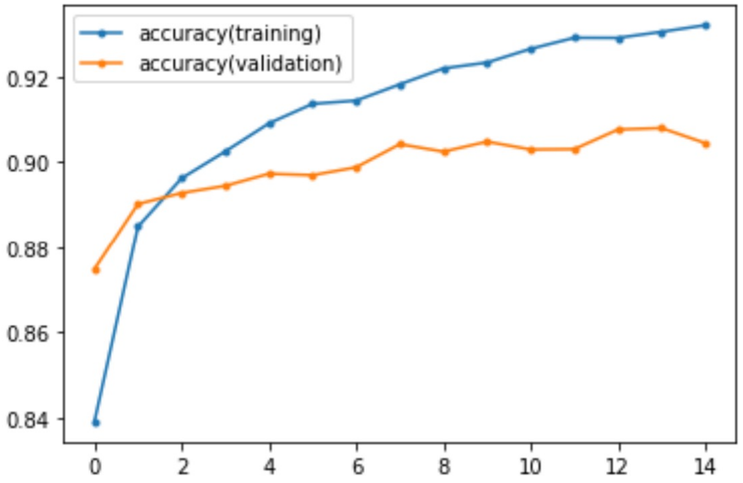
MLPは画像を1次元にしてしまうので、画像の特徴を失ってしまう
CNNは画像の特徴を保存したまま、特徴を抽出するように学習する

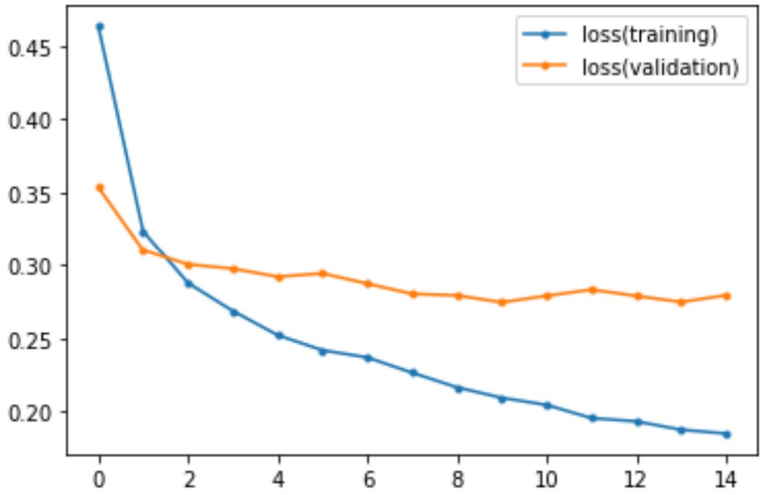




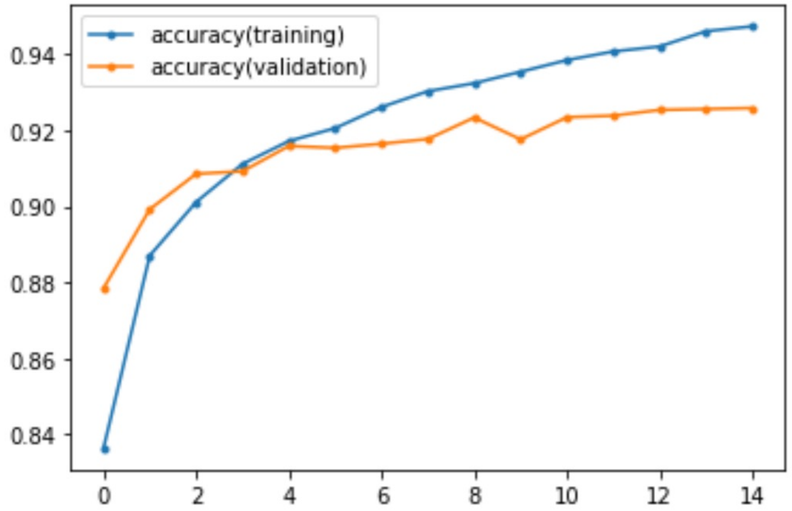
Epochs=15

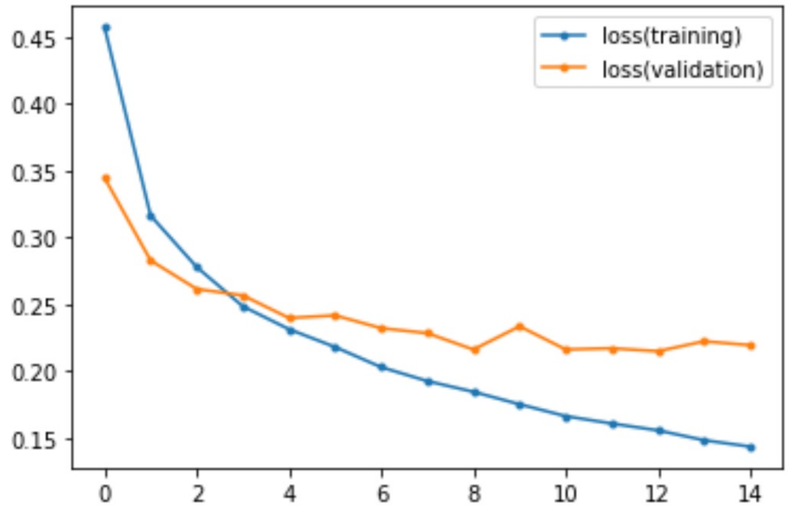
Test loss: 0.2920217216014862
Test accuracy: 0.8999000191688538





Test loss: 0.23001757264137268
Test accuracy: 0.9223999977111816

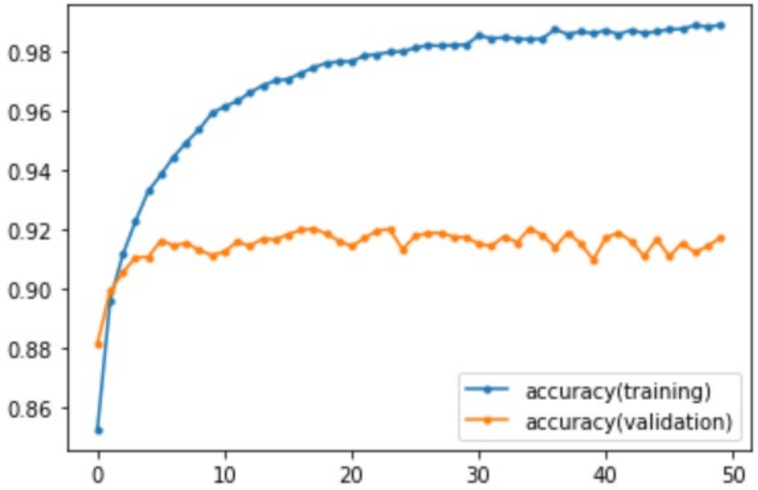


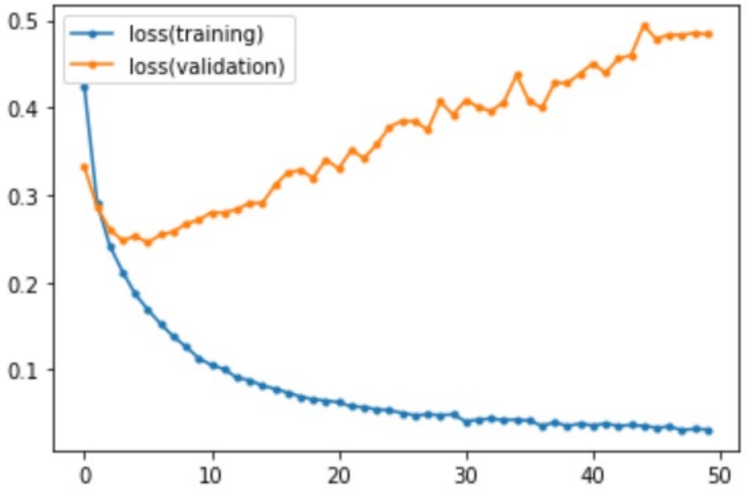


Epochs=50

Test loss: 0.4930589497089386

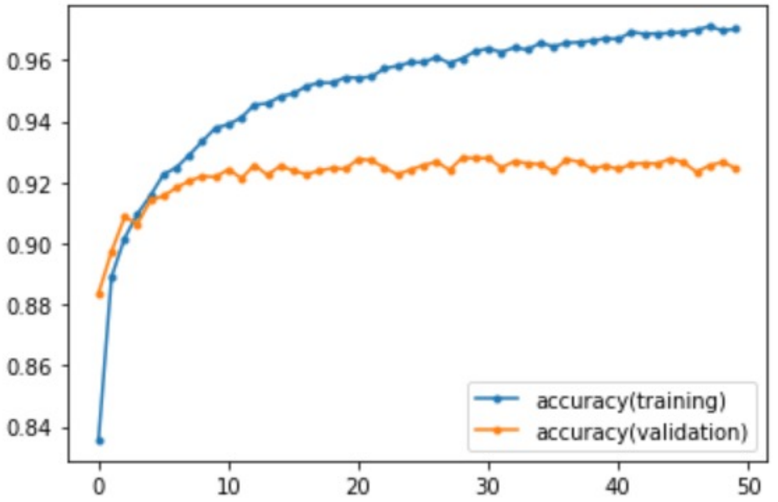
Test accuracy: 0.9175000190734863

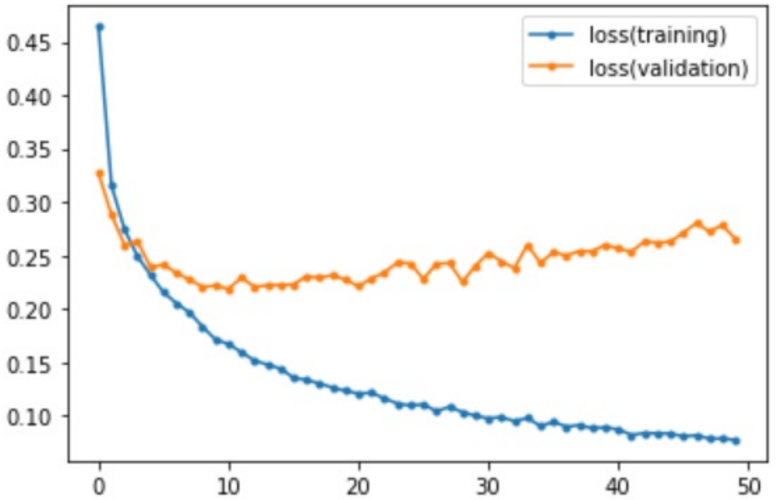




Test loss: 0.2698012888431549

Test accuracy: 0.9247000217437744



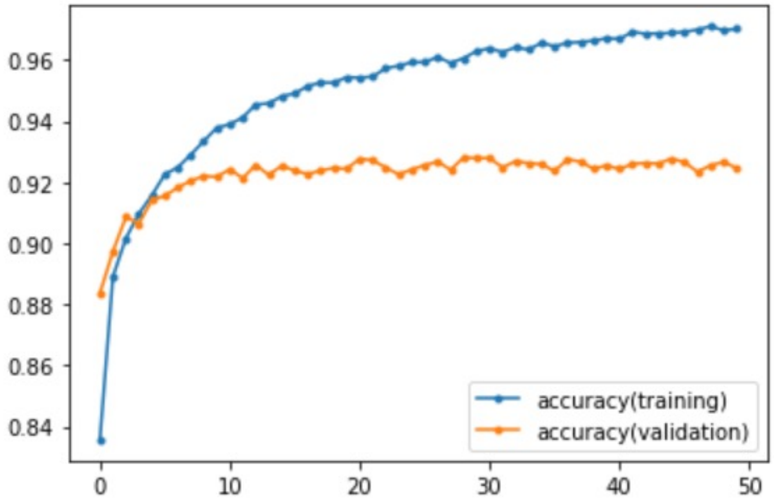


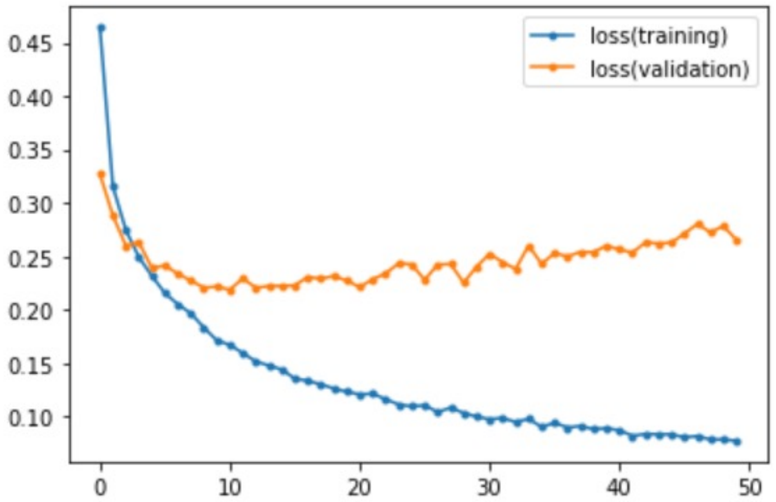
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv2D, Flatten, MaxPooling2D
model = Sequential()
model.add(Conv2D(filters=32,kernel_size=(3,3),padding='same',input_shape=(28,28,1),activation='relu'))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.5))
model.add(Conv2D(filters=64,kernel_size=(3,3), strides = (1, 1), padding='same', activation='relu'))
model.add(Conv2D(filters=128,kernel_size=(3,3), strides = (1, 1), padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
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 = 256, verbose = 1, validation_split=0.2, shuffle=True)
```

Test loss: 0.2698012888431549

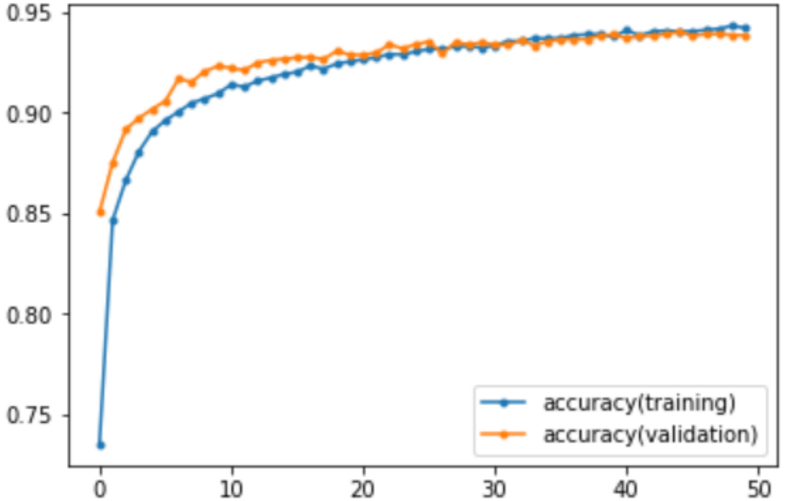
Test accuracy: 0.92470002174377444

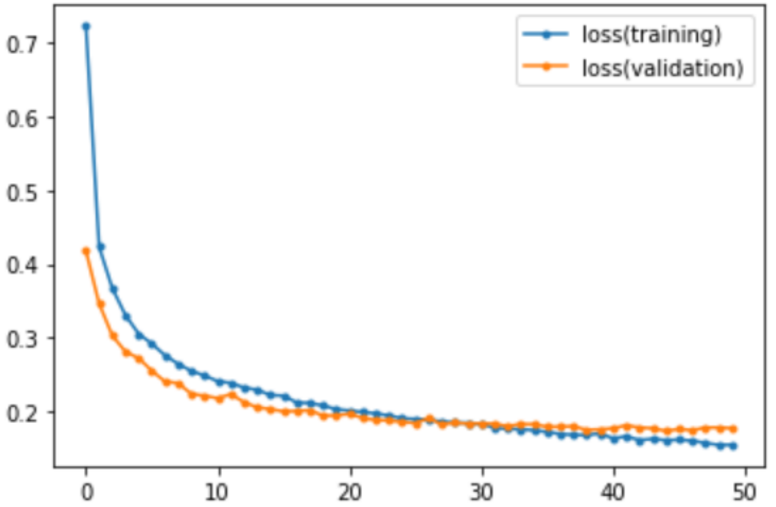




Test loss: 0.19490012526512146

Test accuracy: 0.93260002136230474

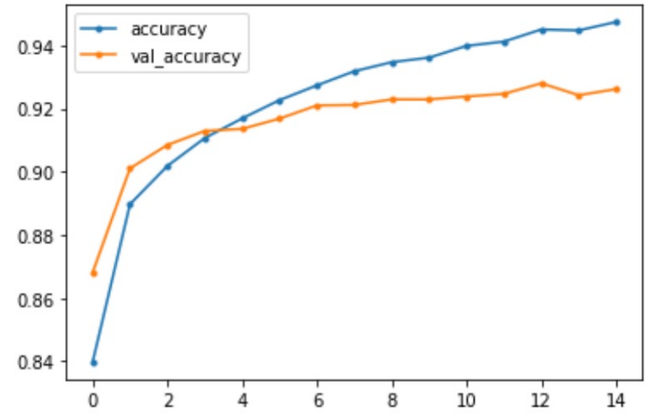




課題

FASHION-MNISTではなく、MNISTでCNNを実践して下さい
学習結果の正解率の図を添付してください
(spyderの保存アイコンから図を保存すること)

・畳み込み層とプーリング層を2つ以上いれること
・Dropout()をいれること



test用データのうち、自分の学籍番号下4桁+10番目の画像の
予測結果(数字とその確率)を示しなさい
Ex) ○○○○2015　→　2025番目