第5回
y=ax+bから始める
初心者向けML講座

これまでのあらすじ

DNNにどのように

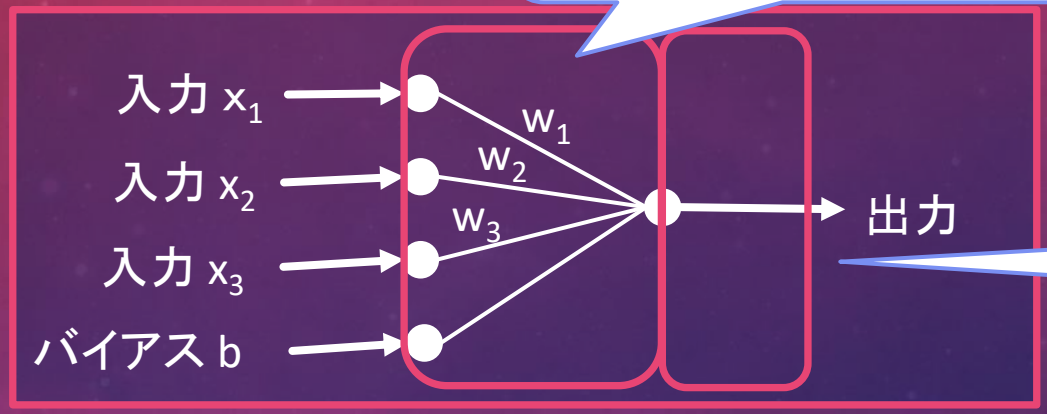
例：mnistを使った画像判定AIのモ

1層ごとに上下左右の関係を維持したまま
圧縮（畳み込み）するように計算していけば
上下左右の関係を維持したまま計算ができる！

画像で重要なのはデータの前後関係ではなく
あるドットを見たときに上下左右の矩形(四角形)
の関係性が重要

畳み込み処理の計算方法

パーセプトロン



畳み込み処理

入力X

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

× フィルターF

1	1	1
2	0	1
−1	−1	0

＋ バイアスb 3 ＝ 出力Y

7	11
23	27

入力値×フィルター
を全て足す。
※フィルターとbが学習するパラメータ

出力Yを活性化関数に通す

畳み込み処理の計算方法４

入力値×重み
を全て足す。
※重みwとbが学習するパラメータ

パーセプトロン

ね、かんたんでしょ？

つまりフィルターを使った
Y=ax+bが畳み込み処理！

活性化関数

処理

3	4
7	8
11	12
15	16

×

フィルターF

1	1	1
2	0	1
−1	−1	0

＋ バイアスb 3 ＝ 出力Y

7	11
23	27

入力値×フィルター
を全て足す。
※フィルターとbが学習するパラメータ

出力Yを活性化関数に通す

マックスプーリング

- マックスプーリングの場合もフィルター（枠のサイズだけ決める）を用意します。(今回は2×2のフィルターとします。)
- 枠のサイズで入力を分割していきます。
- 分割した中で最も大きいものが出力となるのがマックスプーリングです。

フィルター（枠だけ）

入力X

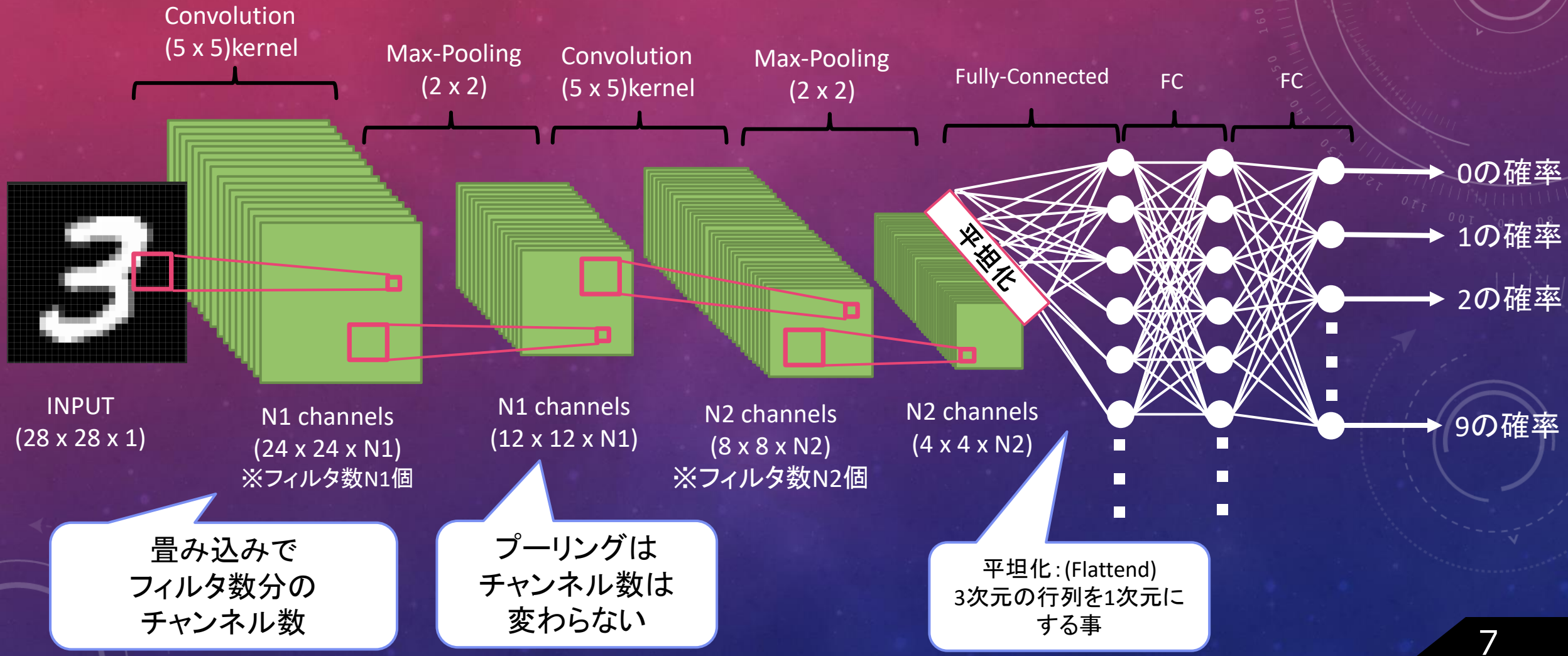
1	2	3	4
5	6	7	8
16	15	14	13
12	11	10	9

→ 出力Y

6	8
16	14

最終的なCNN(Convolutional Neural Network)

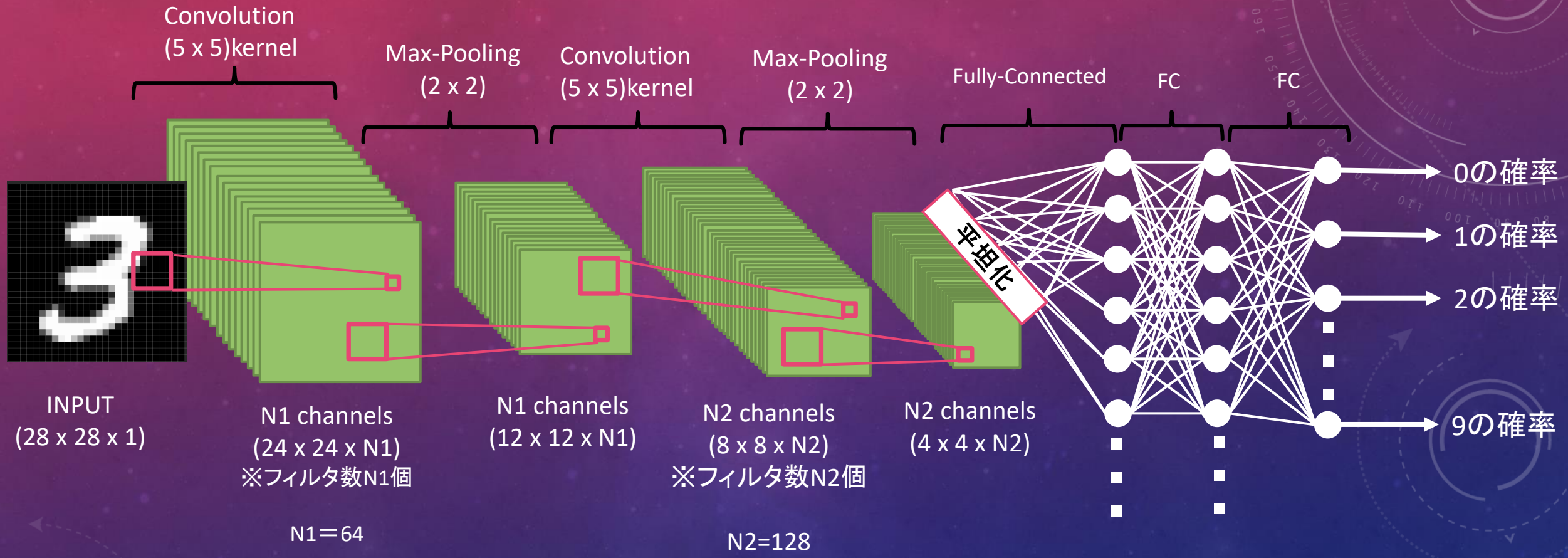
- 最終的に**畳み込み処理**と**プーリング処理**を交互に行い、DNN（全結合層）とつなげるとCNNとなります。



CNNモデルの実装例

最終的なCNN(Convolutional Neural Network)

- 最終的に**畳み込み処理**と**プーリング処理**を交互に行い、DNN（全結合層）とつなげるとCNNとなります。



インポート文

```
1 from keras.engine.base_layer import TensorFlowOpLayer
2 import numpy as np
3 import sys
4 %matplotlib inline
5 import matplotlib.pyplot as plt
6 import keras
7 from keras.datasets import mnist
8 from keras.models import Sequential
9 from keras.layers import Dense, Activation, Flatten
10 from keras.layers import Conv2D, MaxPooling2D
11 from keras.optimizers import Adam
```

インポート文

Import文は適当に書いてるから
要らないものあるかも
ゆっくりインポートしていってね！

```
TensorFlowOpLayer
2 imp          y as np
3 i        sys
4    atplotlib inline
5 import matplotlib.pyplot as plt
6 import keras
7 from keras.datasets import mnist
8 from keras.models import Sequential
9 from keras.layers import Dense, Activation, Flatten
10 from keras.layers import Conv2D, MaxPooling2D
11 from keras.optimizers import Adam
```

データセット読み込み

学習用データ、学習用データの答え、
検証用の入力データと検証用
の答えの4つにデータを分ける。

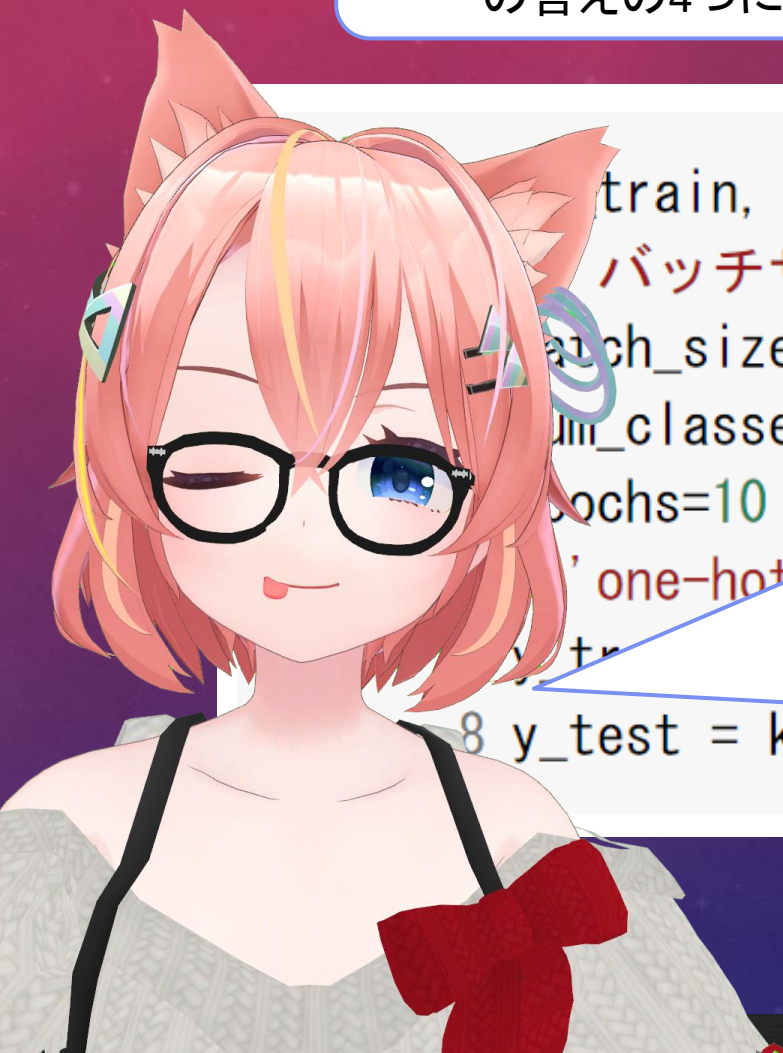
mnist読み込み

```
[2]  1 (x_train, y_train), (x_test, y_test) = mnist.load_data()
     2 '''バッチサイズ、クラス数、エポック数の設定'''
     3 batch_size=128
     4 num_classes=10
     5 epochs=10
     6 '''one-hotベクトル化'''
     7 y_train = keras.utils.to_categorical(y_train, num_classes)
     8 y_test = keras.utils.to_categorical(y_test, num_classes)
```

データセット読み込み

学習用データ、学習用データの答え、
検証用の入力データと検証用
の答えの4つにデータを分ける。

mnist読み込み



```
train, y_train), (x_test, y_test) = mnist.load_data()
バッチサイ
ch_size=
m_classes
ochs=10
' one-hot
8 y_test = ke
```

データは学習データとテストデータに分けて
学習に使わない「テストデータ」が無いと
未知のデータを想定して検証する事が出来なくなるよ！

気を付けてね！

データセット読み込み

1回の学習で
使うデータ件数
今回は128件

Mnistなので0～9の10分類

学習データを1回使い切ったら1エポック
今回は学習データは6万件なので
6万/128=469回学習したら1エポック

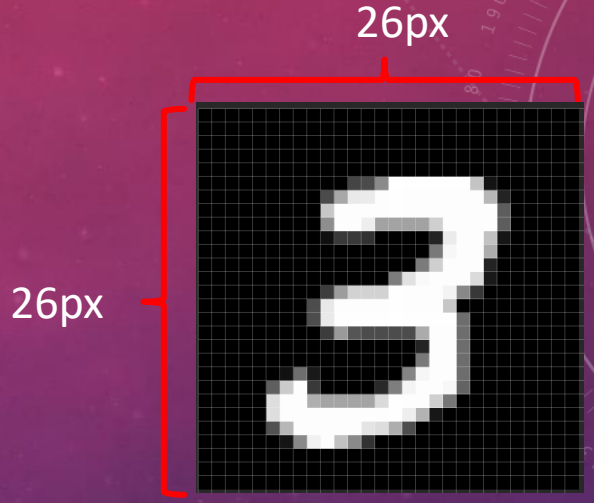
```
[2]  1 (x_train,  _train),  t,  y_test) = mnist.load_data()
     2 '''バッチサイズ、ク   ス数、エポック数
     3 batch_size=128
     4 num_classes=10
     5 epochs=10
     6 '''one-hotベクトル化'''
     7 y_train = keras.utils.to_categorical(y_train, num_classes)
     8 y_test = keras.utils.to_categorical(y_test, num_classes)
```

正解データはワンホットベクトルに
1が正解なら「0,1,0,0,0,0,0,0,0,0」になっている

データサイズ確認

```
1 print("x_train : ", x_train.shape)
2 print("y_train : ", y_train.shape)
3 print("x_test : ", x_test.shape)
4 print("y_test : ", y_test.shape)
```

```
x_train :  (60000, 28, 28)
y_train :  (60000, 10)
x_test :  (10000, 28, 28)
y_test :  (10000, 10)
```



上記のような画像データが6万件ある
28x28の行列データが6万個

答えは「0,1,0,0,0,0,0,0,0,0」
のような10個の行列データが同じ件数分

前処理　データ正規化

```
''' 色の強さは0～255の256段階なので最大が1になるように割る '''
x_train=x_train.astype('float32')
x_train/=255
x_test=x_test.astype('float32')
x_test/=255
```

前処理　データ正規化

データの範囲を0～1にしなくても学習は出来なくはないけど
（学習パラメータの数値が大きくなりすぎて）
学習が進みにくくなることはあるかも

```
の強さは0～255の256段階なので最大が1になるように割る　'''
n=x_train.astype('float32')
/=255
=x_test.astype('float32')
t/=255
```

CNNモデル実装部分

```
model = Sequential()
'''　畳み込み処理　フィルター数64個　カーネル5x5 入力データは28x28の1チャンネルの画像　'''
model.add(Conv2D(filters=64, kernel_size=(5,5), input_shape=(28,28,1), name='conv1'))
model.add(Activation('relu')) #活性化関数はReLU関数

'''　2x2のフィルターでマックスプーリング　'''
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool1'))

'''　畳み込み処理　フィルター数128　カーネル5x5 '''
model.add(Conv2D(filters=128, kernel_size=(5,5), name='conv2'))
model.add(Activation('relu')) #活性化関数はReLU関数

'''　2x2のフィルターでマックスプーリング　'''
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool2'))

model.add(Flatten(name='flatten')) #平坦化処理
model.add(Dense(units=2048, activation='relu', name='fc1'))#2048個のノードの全結合
model.add(Dense(units=1024, activation='relu', name='fc2'))#1024個のノードの全結合
''' 最終層は10個のノードで活性化関数はsoftmax '''
model.add(Dense(units=num_classes, activation='softmax', name='predictions'))
model.summary()
```

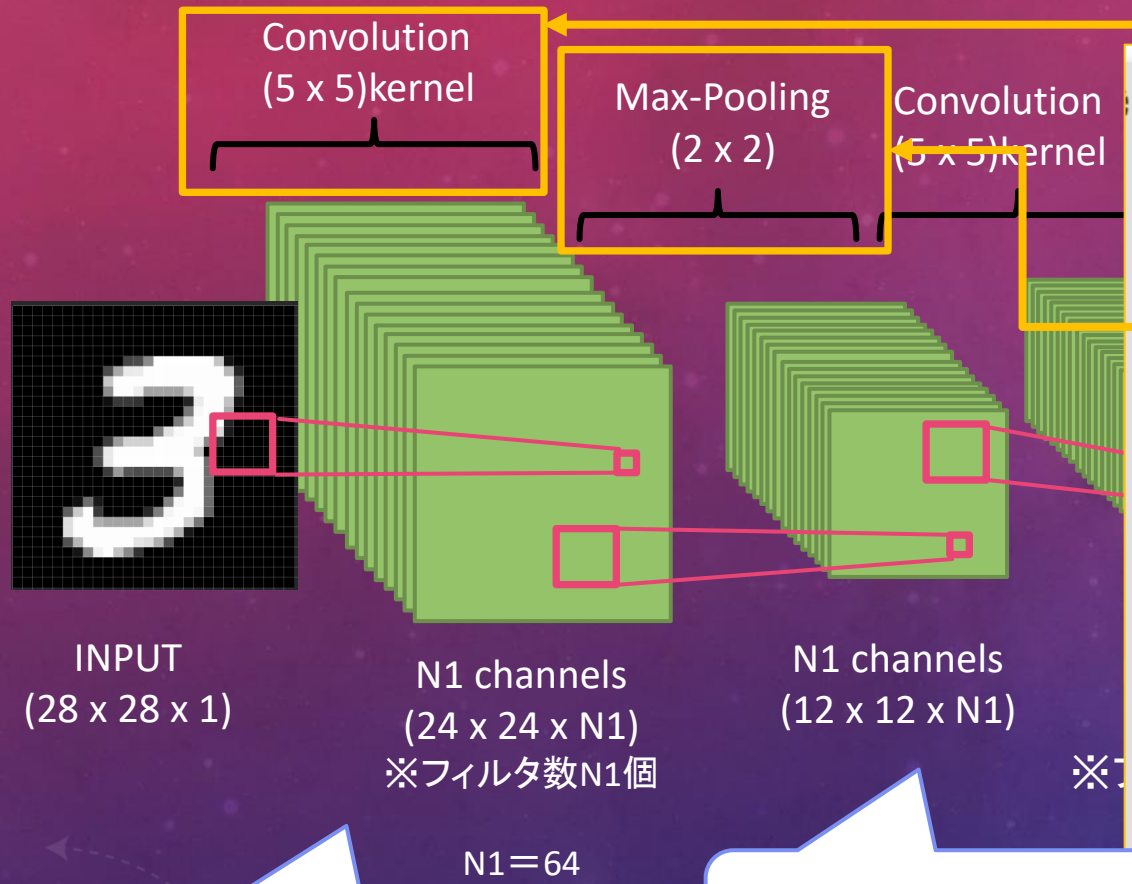
CNNモデル実装部分

Kerasなんかを使うとこれだけ簡単にモデルが書けるよ！
やったね！

```
model = Sequential()

入力データは28x28の1チャンネルの画像  '''
nput_shape=(28, 28, 1), name='conv1'))
数

    2x2のフィルターでマックスプーリング  '''
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool1'))

'''   畳み込み処理   フィルター数128   カーネル5x5 '''
filters=128, kernel_size=(5,5), name='conv2'))
ion('relu')) #活性化関数はReLU関数

ターでマックスプーリング  '''
ing2D(pool_size=(2,2), name='max_pool2'))

ten(name='flatten')) #平坦化処理
(units=2048, activation='relu', name='fc1'))#2048個のノードの全結合
(units=1024, activation='relu', name='fc2'))#1024個のノードの全結合
個のノードで活性化関数はsoftmax '''
(units=num_classes, activation='softmax', name='predictions'))
```

最終的なCNN(Convolutional Neural Network)

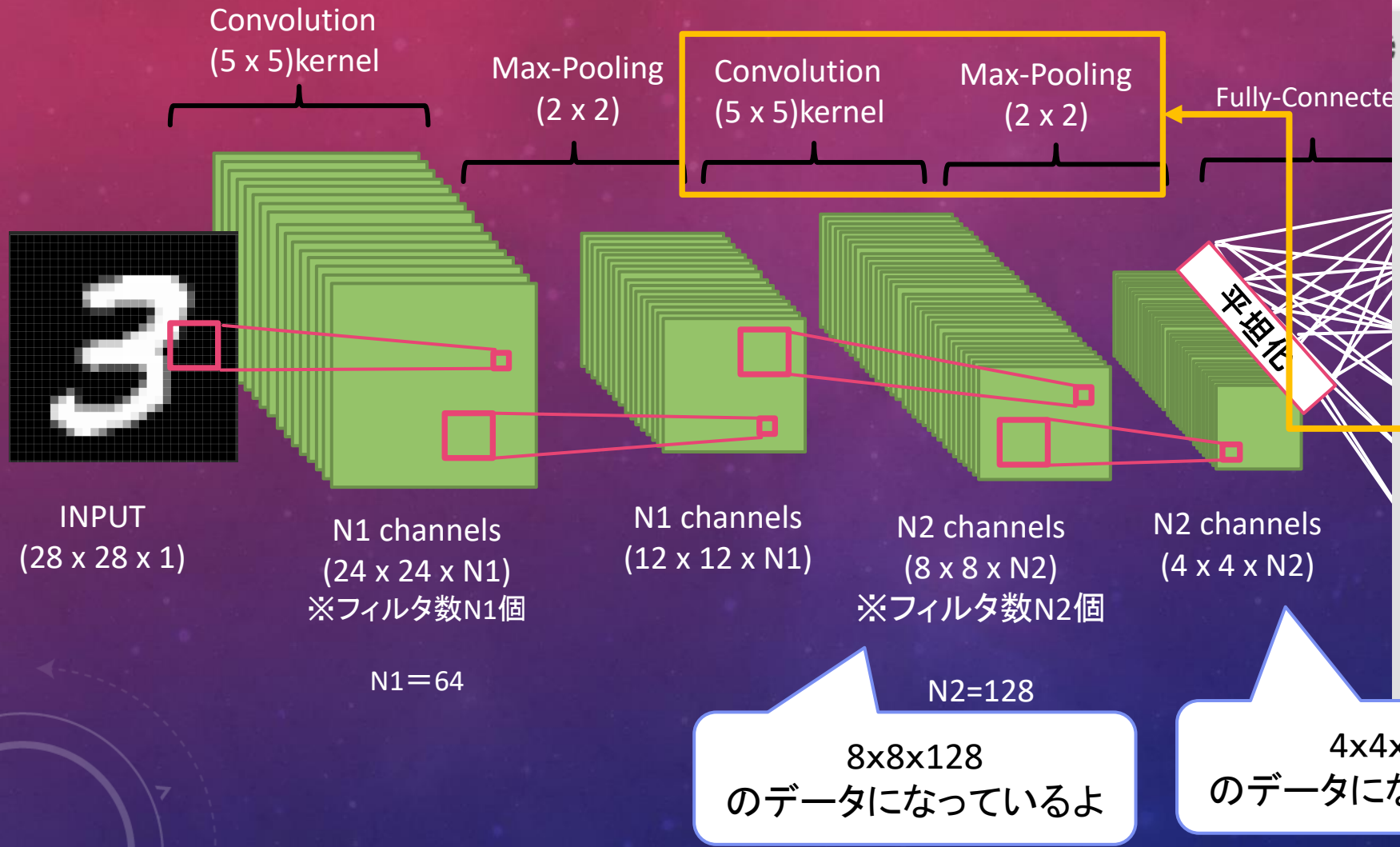
- 最終的に**畳み込み処理**と**プーリング処理**を交互に行い、DNN（全結合層）とつなげるとCNNとなります。




```
model = Sequential()
'''　畳み込み処理　フィルター数64個　カーネル5x5 入力データは28x28の1チャンネルの画像　'''
model.add(Conv2D(filters=64, kernel_size=(5,5), input_shape=(28,28,1), name='conv1'))
model.add(Activation('relu')) #活性化関数はReLU関数

'''　2x2のフィルターでマックスプーリング　'''
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool1'))

'''　畳み込み処理　フィルター数128　カーネル5x5 '''
model.add(Conv2D(filters=128, kernel_size=(5,5), name='conv2'))
model.add(Activation('relu')) #活性化関数はReLU関数

'''　2x2のフィルターでマックスプーリング　'''
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool2'))

model.add(Flatten(name='flatten')) #平坦化処理
model.add(Dense(units=2048, activation='relu', name='fc1'))#2048個のノードの全結合
model.add(Dense(units=1024, activation='relu', name='fc2'))#1024個のノードの全結合
''' 最終層は10個のノードで活性化関数はsoftmax '''
.add(Dense(units=num_classes, activation='softmax', name='predictions'))
.summary()
```

最終的なCNN(Convolutional Neural Network)

- 最終的に**畳み込み処理**と**プーリング処理**を交互に行い、DNN（全結合層）とつなげるとCNNとなります。



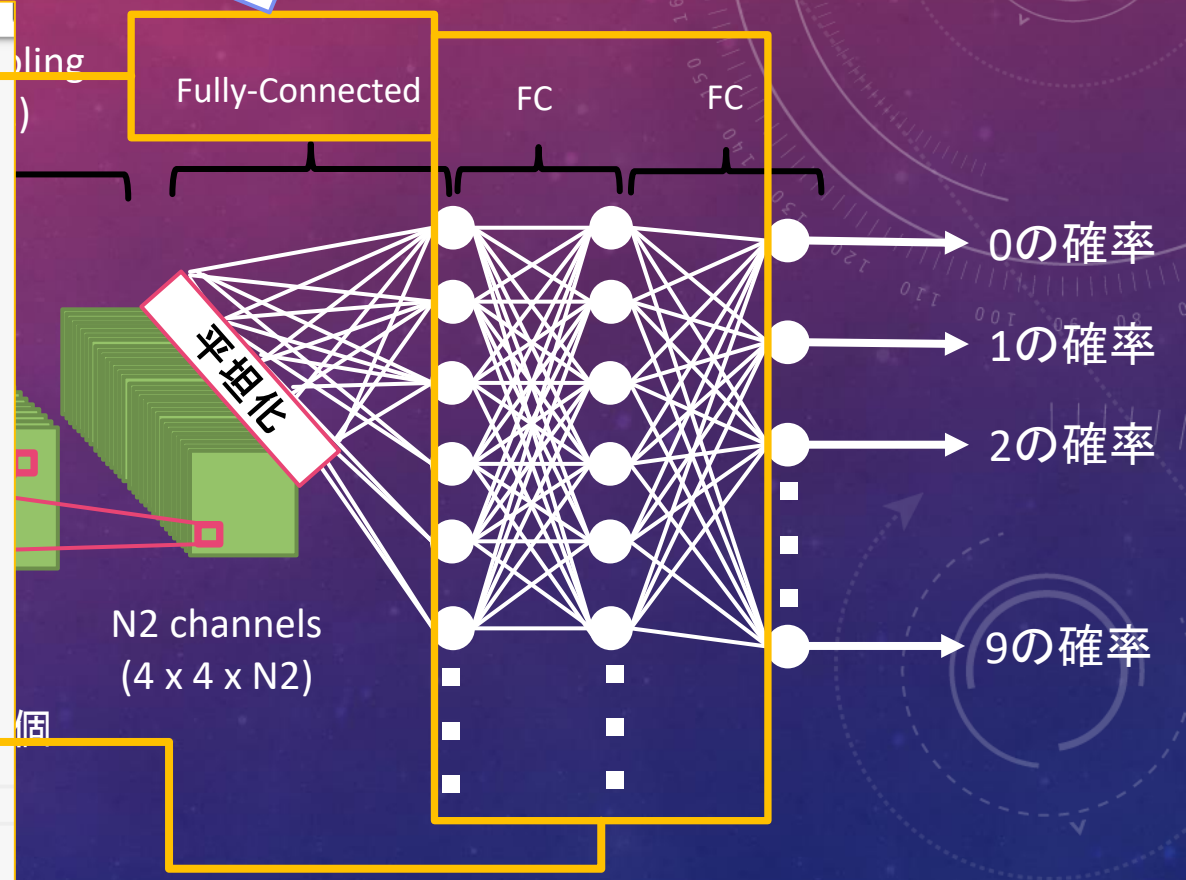

```
model = Sequential()
'''   畳み込み処理   フィルター数64個   カーネ
model.add(Conv2D(filters=64, kernel_size=(5
model.add(Activation('relu')) #活性化関数は

'''   2x2のフィルターでマックスプーリング   '
model.add(MaxPooling2D(pool_size=(2,2), nam

'''   畳み込み処理   フィルター数128   カーネル
model.add(Conv2D(filters=128, kernel_size=(
model.add(Activation('relu')) #活性化関数は

'''   2x2のフィルターでマックスプーリング   '
model.add(MaxPooling2D(pool_size=(2,2), nam

model.add(Flatten(name='flatten')) #平坦化処
model.add(Dense(units=2048, activation='rel
model.add(Dense(units=1024, activation='rel
'''最終層は10個のノードで活性化関数はsoftm
nse(units=num_classes, activati
y()
```

最終的なCNN(Convolutiona k)

- 最終的に**畳み込み処理**と**プーリング処理**を交互に行い、DNI ます。

Convolution

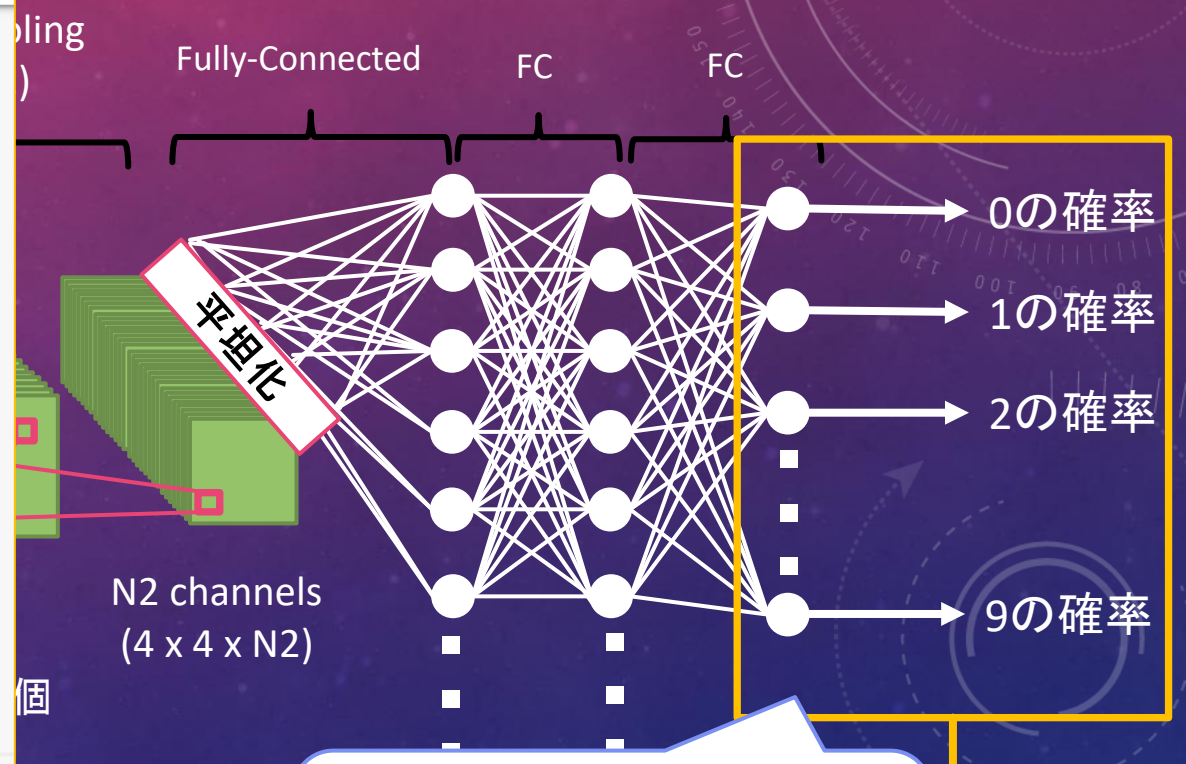
```
model = Sequential()
'''　畳み込み処理　フィルター数64個　カーネル5x5 入力データは28x28の1チャンネルの画像　'''
model.add(Conv2D(filters=64, kernel_size=(5,5), input_shape=(28,28,1), name='conv1'))
model.add(Activation('relu')) #活性化関数はReLU関数

'''　2x2のフィルターでマックスプーリング　'''
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool1'))

'''　畳み込み処理　フィルター数128　カーネル5x5 '''
model.add(Conv2D(filters=128, kernel_size=(5,5), name='conv2'))
model.add(Activation('relu')) #活性化関数はReLU関数

'''　2x2のフィルターでマックスプーリング　'''
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool2'))

model.add(Flatten(name='flatten')) #平坦化処理
model.add(Dense(units=2048, activation='relu', name='fc1'))#2048個のノードの全結合
model.add(Dense(units=1024, activation='relu', name='fc2'))#1024個のノードの全結合
''' 最終層は10個のノードで活性化関数はsoftmax '''
model.add(Dense(units=num_classes, activation='softmax', name='predictions'))
model.summary()
```



最終的なCNN(Convolutional Neural Network)

- 最終的に**畳み込み処理**と**プーリング処理**を交互に行い、DNN（全結合層）とつなげるとCNNとなります。

```python
model = Sequential()
'''　畳み込み処理　フィルター数64個　カーネル5x5 入力データは28x28の1チャンネルの画像　'''
model.add(Conv2D(filters=64, kernel_size=(5,5), input_shape=(28,28,1), name='conv1'))
model.add(Activation('relu')) #活性化関数はReLU関数

'''　2x2のフィルターでマックスプーリング　'''
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool1'))

'''　畳み込み処理　フィルター数128　カーネル5x5 '''
model.add(Conv2D(filters=128, kernel_size=(5,5), name='conv2'))
model.add(Activation('relu')) #活性化関数はReLU関数

'''　2x2のフィルターでマックスプーリング　'''
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool2'))

model.add(Flatten(name='flatten')) #平坦化処理
model.add(Dense(units=2048, activation='relu', name='fc1'))#2048個のノードの全結合
model.add(Dense(units=1024, activation='relu', name='fc2'))#1024個のノードの全結合
''' 最終層は10個のノードで活性化関数はsoftmax '''
model.add(Dense(units=num_classes, activation='softmax', name='predictions'))
model.summary()
```



10種類の分類問題なので
出力層は10個
活性化関数は
ソフトマックス関数

最終的なCNN(Convolutional Neu

- 最終的に**畳み込み処理**と**プーリング処理**を交互に行い、DNN

```
model = Sequential()
'''  畳み込み処理  フィルター数64個  カーネル5x5 入力データは28x28の1チャンネルの画像  '''
model.add(Conv2D(filters=64, kernel_size=(5,5), input_shape=(28,28,1), name='conv1'))
model.add(Activation('relu')) #活性化関数はReLU関数

'''  2x2のフィルターでマックスプーリング  '''
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool1'))

'''  畳み込み処理  フィルター数128  カーネル5x5 '''
model.add(Conv2D(filters=128, kernel_size=(5,5), name='conv2'))
model.add(Activation('relu')) #活性化関数はReLU関数

'''  2x2のフィルターでマックスプーリング  '''
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool2'))

model.add(Flatten(name='flatten')) #平坦化処理
model.add(Dense(units=2048, activation='relu', name='fc1'))#2048個のノードの全結合
model.add(Dense(units=1024, activation='relu', name='fc2'))#1024個のノードの全結合
''' 最終層は10個のノードで活性化関数はsoftmax '''
model.add(Dense(units=num_classes, activation='softmax', name='predictions'))
model.summary()
```

モデルのサマリーだよ！

全結合(Dense)層
が学習パラメータが
多いのが分かるよ！

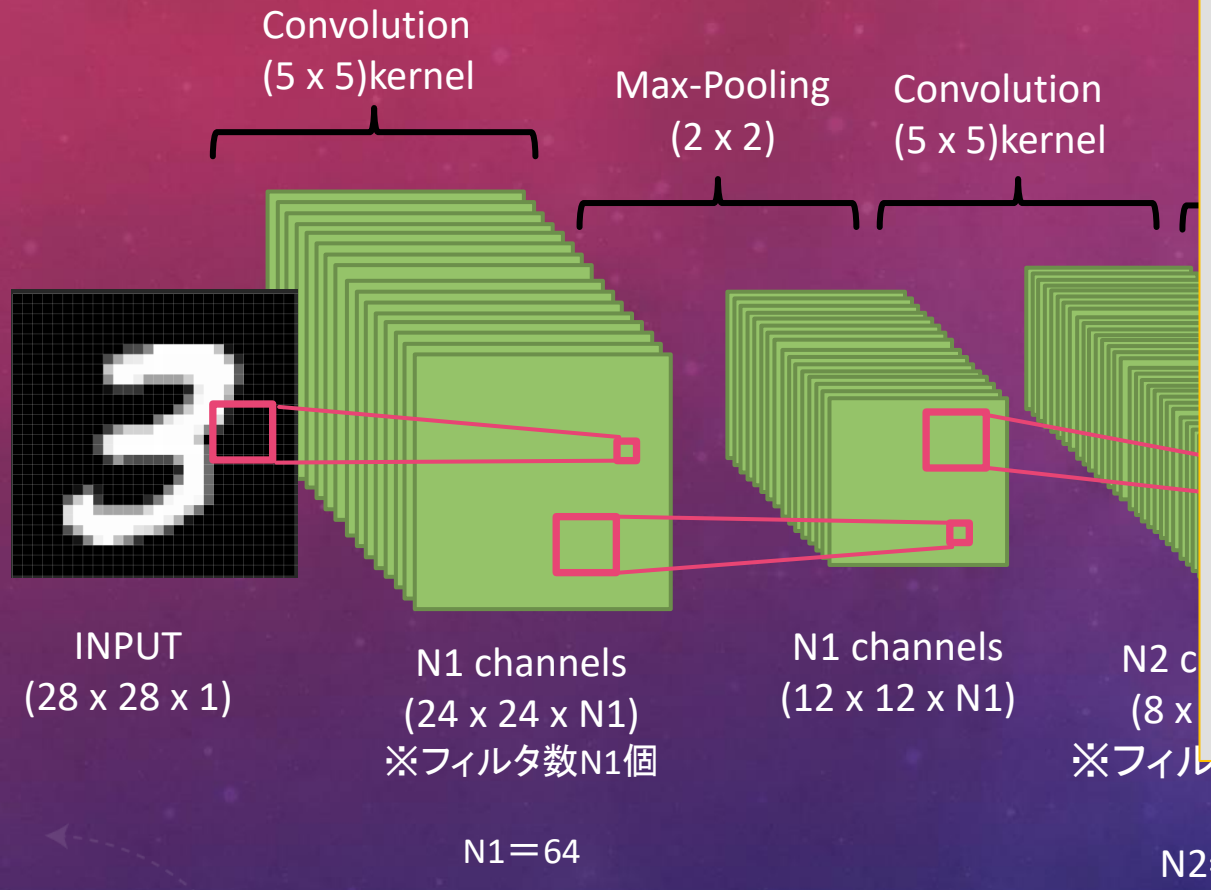
Model: "seque

Layer (type)	Output Shape	Param #
conv1 (Conv2D)	(None, 24, 24, 64)	1664
activation (Activation)	(None, 24, 24, 64)	0
max_pool1 (MaxPooling2D)	(None, 12, 12, 64)	0
conv2 (Conv2D)	(None, 12, 12, 128)	204928
activation_1 (Activation)	(None, 12, 12, 128)	0
max_pool2 (MaxPooling2D)	(None, 6, 6, 128)	0
flatten (Flatten)	(None, 4608)	0
fc1 (Dense)	(None, 2048)	9439232
fc2 (Dense)	(None, 1024)	2098176
predictions (Dense)	(None, 10)	10250

Total params: 11,754,250
Trainable params: 11,754,250
Non-trainable params: 0

最終的なCNN(Convolutional Neural Network)

- 最終的に**畳み込み処理**と**プーリング処理**を交互に行い




```python
model = Sequential()
'''　畳み込み処理　フィルター数64個　カーネル5x5 入力データは28x28の1チャンネルの画像　'''
model.add(Conv2D(filters=64, kernel_size=(5,5), input_shape=(28,28,1), name='conv1'))
model.add(Activation('relu')) #活性化関数はReLU関数

'''　2x2のフィルターでマックスプーリング　'''
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool1'))

'''　畳み込み処理　フィルター数128　カーネル5x5 '''
model.add(Conv2D(filters=128, kernel_size=(5,5), name='conv2'))
model.add(Activation('relu')) #活性化関数はReLU関数

'''　2x2のフィルターでマックスプーリング　'''
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool2'))

model.add(Flatten(name='flatten')) #平坦化処理
model.add(Dense(units=2048, activation='relu', name='fc1'))#2048個のノードの全結合
model.add(Dense(units=1024, activation='relu', name='fc2'))#1024個のノードの全結合
''' 最終層は10個のノードで活性化関数はsoftmax '''
model.add(Dense(units=num_classes, activation='softmax', name='predictions'))
model.summary()
```

出力層は10個
活性化関数は
ソフトマックス関数

学習

パラメータの定義はこちら

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
'''バッチサイズ、クラス数、エポック数の設定'''
batch_size=128
num_classes=10
epochs=10
'''one-hotベクトル化'''
y_train = keras.utils.to_categorical(y_train, num_classes)
st                                    m_classes)
```

誤差関数は分類用の
交差エントロピー誤差

学習手法 重みの更新手法
（本日は省略）

```
'''optimizer定義（学習アルゴリズム）'''
model.compile(loss='categorical_crossentropy',optimizer=Adam(learning_rate=0.001),metrics=['accuracy'])

'''学習'''
history=model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=(x_test, y_test))
```

128件の平均で学習

エポック数は10
6万件/128で469回を10周
4690回学習をするよ！

学習ログ

今回は1時間ほどで学習が完了したよ！

```
Epoch 1/10
469/469 [==============================] - 260s 552ms/step - loss: 0.1313 - accuracy: 0.9581 - val_loss: 0.0322 - val_accuracy: 0.9898
Epoch 2/10
469/469 [==============================] - 245s 523ms/step - loss: 0.0367 - accuracy: 0.9891 - val_loss: 0.0372 - val_accuracy: 0.9882
Epoch 3/10
469/469 [==============================] - 243s 519ms/step - loss: 0.0242 - accuracy: 0.9925 - val_loss: 0.0377 - val_accuracy: 0.9890
Epoch 4/10
469/469 [==============================] - 243s 519ms/step - loss: 0.0183 - accuracy: 0.9946 - val_loss: 0.0278 - val_accuracy: 0.9908
Epoch 5/10
469/469 [==============================] - 245s 522ms/step - loss: 0.0165 - accuracy: 0.9948 - val_loss: 0.0316 - val_accuracy: 0.9903
Epoch 6/10
469/469 [==============================] - 243s 518ms/step - loss: 0.0140 - accuracy: 0.9956 - val_loss: 0.0217 - val_accuracy: 0.9938
Epoch 7/10
469/469 [==============================] - 251s 535ms/step - loss: 0.0092 - accuracy: 0.9973 - val_loss: 0.0400 - val_accuracy: 0.9911
Epoch 8/10
469/469 [==============================] - 244s 521ms/step - loss: 0.0118 - accuracy: 0.9964 - val_loss: 0.0439 - val_accuracy: 0.9880
Epoch 9/10
469/469 [==============================] - 244s 521ms/step - loss: 0.0118 - accuracy: 0.9965 - val_loss: 0.0375 - val_accuracy: 0.9916
Epoch 10/10
469/469 [==============================] - 243s 519ms/step - loss: 0.0075 - accuracy: 0.9980 - val_loss: 0.0299 - val_accuracy: 0.9926
```

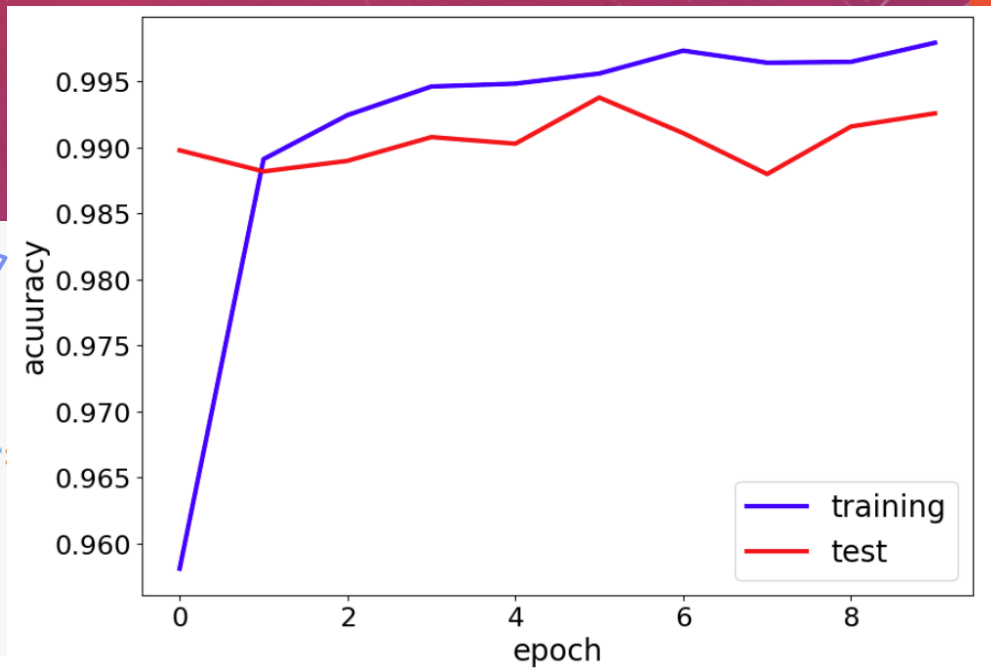
結果を可視化

```
'''結果の可視化'''
plt.figure(figsize=(10,7))
plt.plot(history.history['accuracy'], color='b', linewidth=3)
plt.plot(history.history['val_accuracy'], color='r', linewidth=3)
plt.tick_params(labelsize=18)
plt.ylabel('acuuracy', fontsize=20)
plt.xlabel('epoch', fontsize=20)
plt.legend(['training', 'test'], loc='best', fontsize=20)
plt.figure(figsize=(10,7))
plt.plot(history.history['loss'], color='b', linewidth=3)
plt.plot(history.history['val_loss'], color='r', linewidth=3)
plt.tick_params(labelsize=18)
plt.ylabel('loss', fontsize=20)
plt.xlabel('epoch', fontsize=20)
plt.legend(['training', 'test'], loc='best', fontsize=20)
plt.show()
```

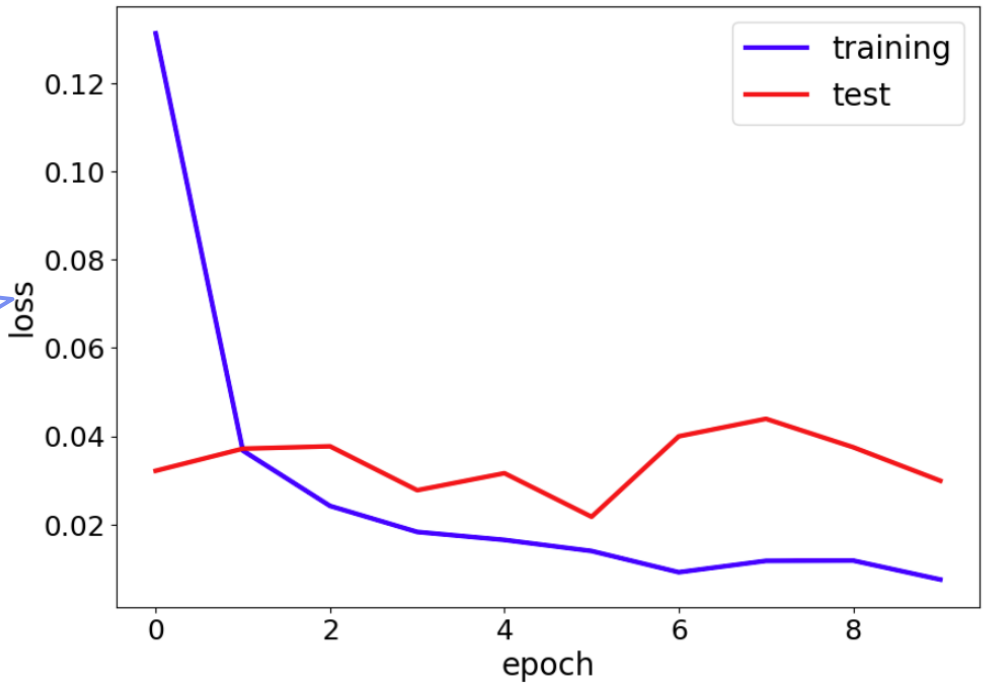
結果を可視化

```
'''結果の可視化'
plt.figure(figsi
plt.plot(history.
plt.plot(history.
plt.tick_params(labelsize=18)
plt.ylabel('acuuracy', fontsize=20)
plt.xlabel('epoch', fontsize=20)
plt.legend(['training', 'test'], loc='best', fo
plt.figure(figsize=(10,7))
plt.plot(history.history['loss'], color='b', l
plt.plot(history.history['val_loss'], color='r'
plt.tick_pa
plt.ylabel
plt.xlabel( epoch , fontsize=20)
plt.legend(['training', 'test'], loc='best', fo
plt.show()
```

正解率だよ！
青が学習データ
赤がテストデータ



損失関数の数値の遷移だよ！



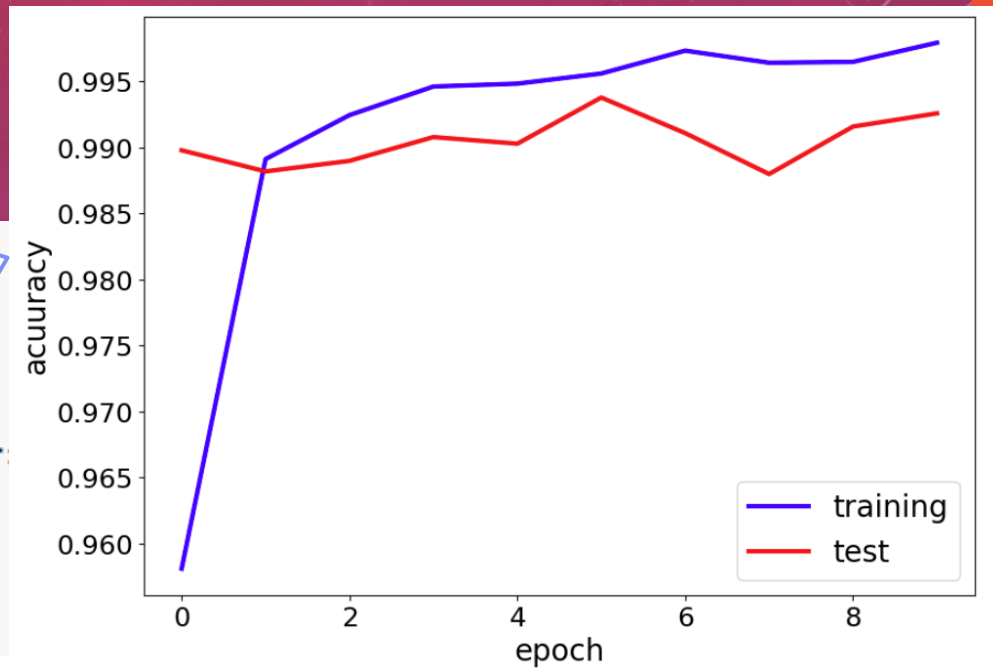
結果を可視化

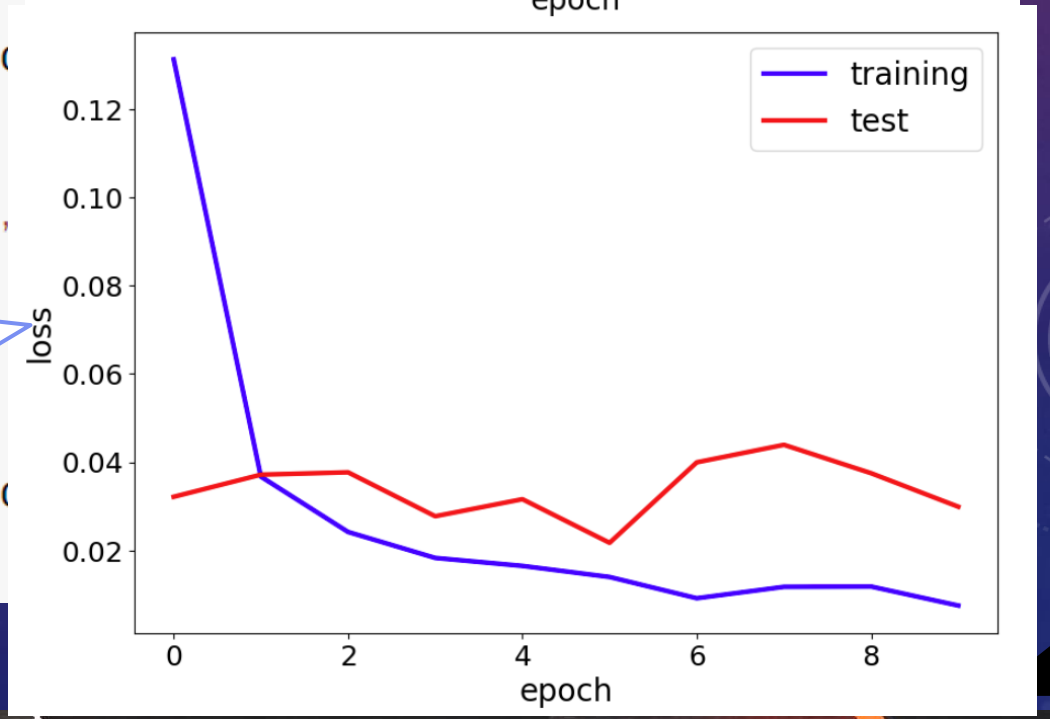
ね？かんたんでしょ？

よ！
データ
赤がテストデータ

```
3 plt.p      story.
  plt.p    (history.
  lt.tick_params(labelsize=18)
      uracy', fontsize=20)
      och', fontsize=20)
      raining', 'test'], loc='best', fo
     gsize=(10,7))
     tory.history['loss'], color='b', l
     ory.history['val_loss'], color='r'
14   xlabel('epoch', fontsize=20)
     d(['training', 'test'], loc='best', fo
```

損失関数の数値の遷移だよ！





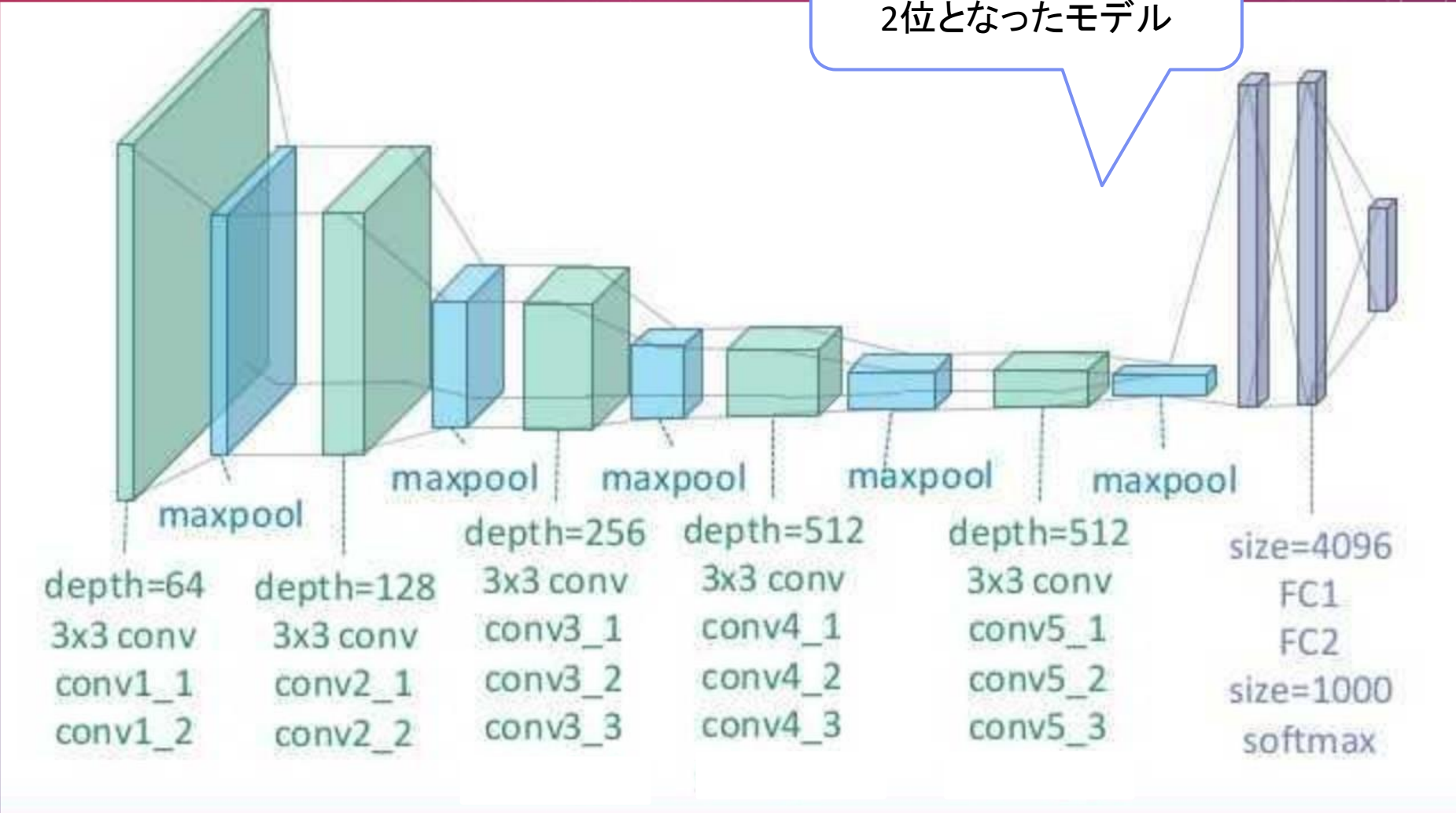
サンプルコードのURL

- KerasでCNN実装例

  https://colab.research.google.com/drive/1UsmWgIjXeK-BEndFDIfXlxGxWjh_IKPx?usp=sharing

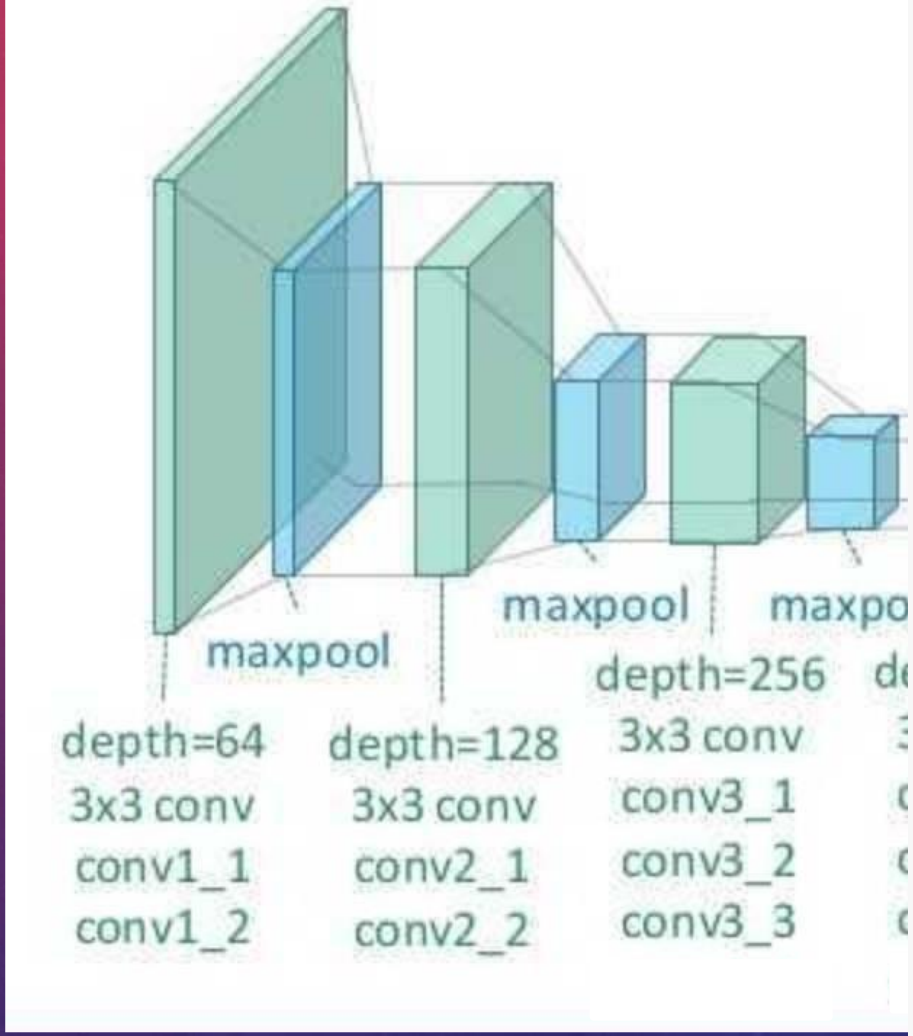
一応、サンプルコードも
公開しておくね！

VGG16



※引用元：https://github.com/scofield7419/basic_NNs_in_frameworks/blob/master/assets/4.png

VGG16




```python
'''VGG16'''
model = Sequential()
model.add(Conv2D(filters=64, kernel_size=(3,3),padding='same', input_shape=input_shape, name='conv1_1'))
model.add(Activation('relu'))
model.add(Conv2D(filters=64, kernel_size=(3,3),padding='same', name='conv1_2'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2), padding='same', name='pool'))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding='same', name='conv2_1'))
model.add(Activation('relu'))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding='same', name='conv2_2'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2), padding='same', name='pool2'))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding='same', name='conv3_1'))
model.add(Activation('relu'))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding='same', name='conv3_2'))
model.add(Activation('relu'))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding='same', name='conv3_3'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2), padding='same', name='pool3'))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding='same', name='conv4_1'))
model.add(Activation('relu'))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding='same', name='conv4_2'))
model.add(Activation('relu'))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding='same', name='conv4_3'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2), padding='same', name='pool4'))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding='same', name='conv5_1'))
model.add(Activation('relu'))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding='same', name='conv5_2'))
model.add(Activation('relu'))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding='same', name='conv5_3'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2), padding='same', name='pool5'))
model.add(Flatten(name='flatten'))
model.add(Dense(units=4096, activation='relu', name='fc1'))
model.add(Dense(units=4096, activation='relu', name='fc2'))
model.add(Dense(units=num_classes, activation='softmax', name='predictions'))
model.summary()
```

※引用元：https://github.com/scofield7419/basic_NNs_in_frameworks/blob/master/assets/4.png

VGG16

簡単に書けたけど、
地味に疲れたよ！


```python
'''VGG16'''
model = Sequential()
model.add(Conv2D(filters=64, kernel_size=(3,3),padding='same', input_shape=input_shape, name='conv1_1'))
model.add(Activation('relu'))
model.add(Conv2D(filters=64, kernel_size=(3,3),padding='same', name='conv1_2'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2), padding='same', name='pool'))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding='same', name='conv2_1'))
model.add(Activation('relu'))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding='same', name='conv2_2'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2), padding='same', name='pool2'))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding='same', name='conv3_1'))
model.add(Activation('relu'))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding='same', name='conv3_2'))
model.add(Activation('relu'))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding='same', name='conv3_3'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2), padding='same', name='pool3'))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding='same', name='conv4_1'))
model.add(Activation('relu'))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding='same', name='conv4_2'))
model.add(Activation('relu'))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding='same', name='conv4_3'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2), padding='same', name='pool4'))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding='same', name='conv5_1'))
model.add(Activation('relu'))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding='same', name='conv5_2'))
model.add(Activation('relu'))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding='same', name='conv5_3'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2), padding='same', name='pool5'))
model.add(Flatten(name='flatten'))
model.add(Dense(units=4096, activation='relu', name='fc1'))
model.add(Dense(units=4096, activation='relu', name='fc2'))
model.add(Dense(units=num_classes, activation='softmax', name='predictions'))
model.summary()
```

om/scofield7419/basic_NNs_in_frameworks/blob/master/assets/4.png

VGG16　サンプルコードURL

- KerasでVGG16実装例

  https://colab.research.google.com/drive/1kE95Vdlz56t-z2MwJYIZqXezIWwFdbgC?usp=sharing

おわり

ありがとうございました