

Convolutional Neural Network (CNN)

<https://youtu.be/n6HhgrCizpc>

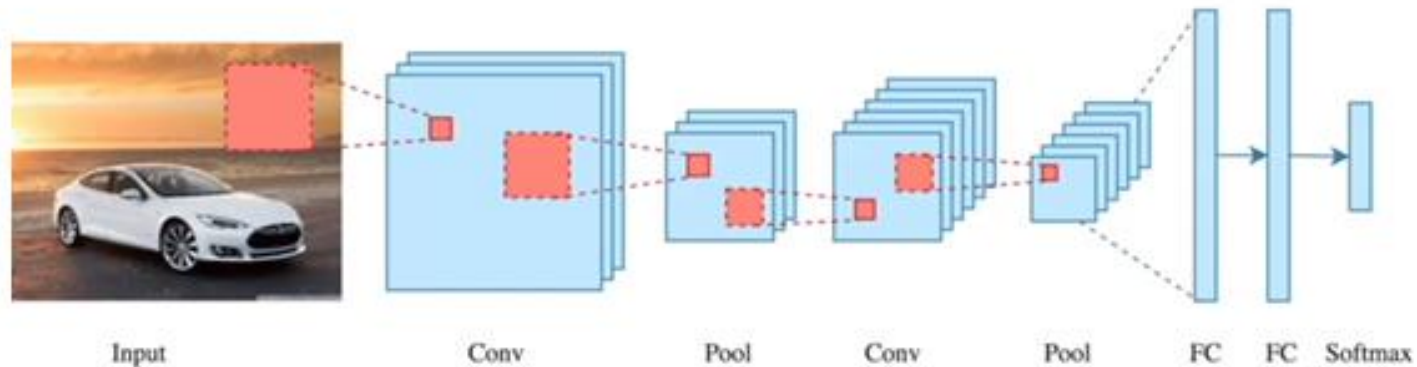
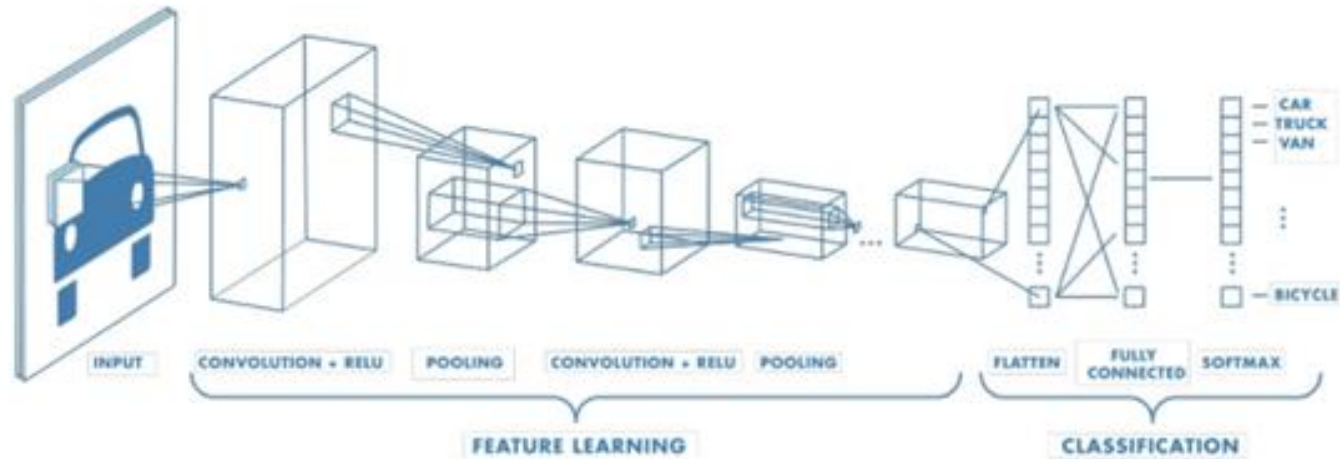
Convolutional Neural Network (CNN) 구조

Convolutional Neural Network (CNN) 개념

Mnist 개념 및 데이터 구조

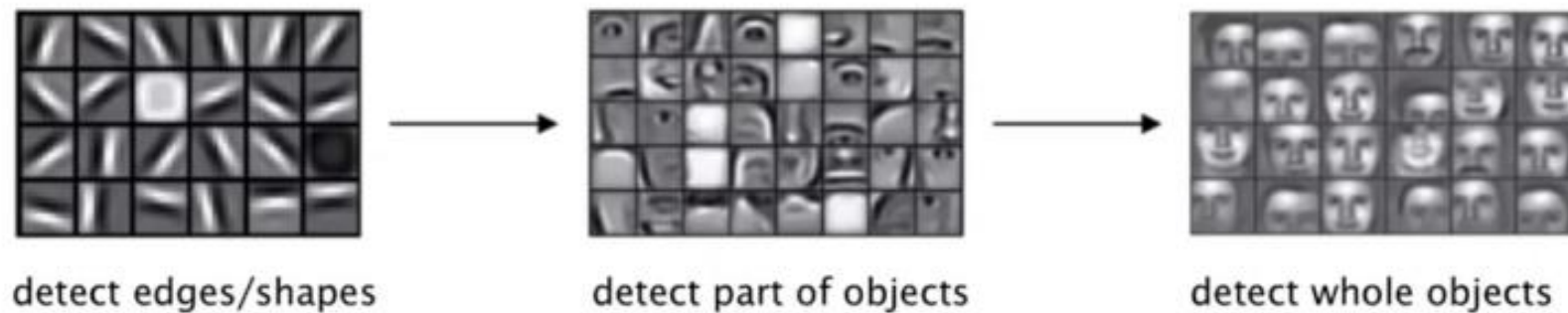
Mnist 실습

Convolutional Neural Network (CNN) 구조

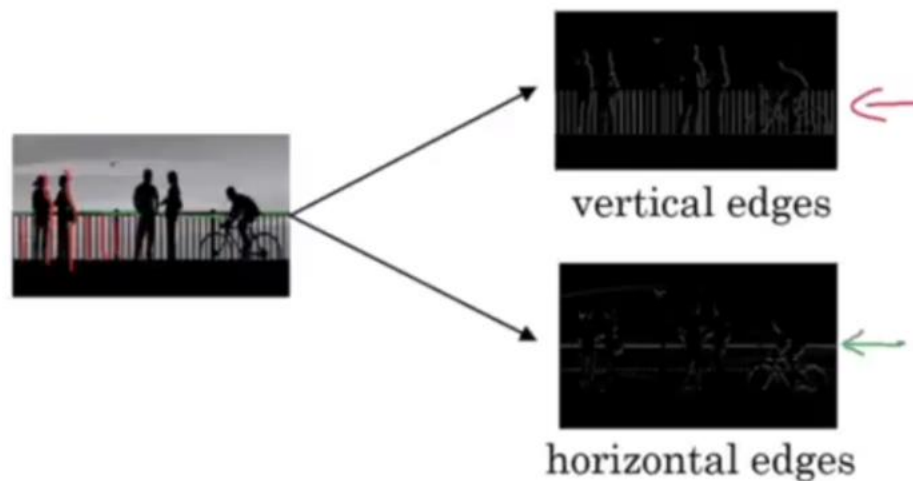


Convolutional Layer -> Pooling Layer -> (반복) -> Fully connected layers -> SoftMax Layer를 통과시켜 아웃풋을 얻어낸다.

Convolutional Neural Network (CNN) 개념



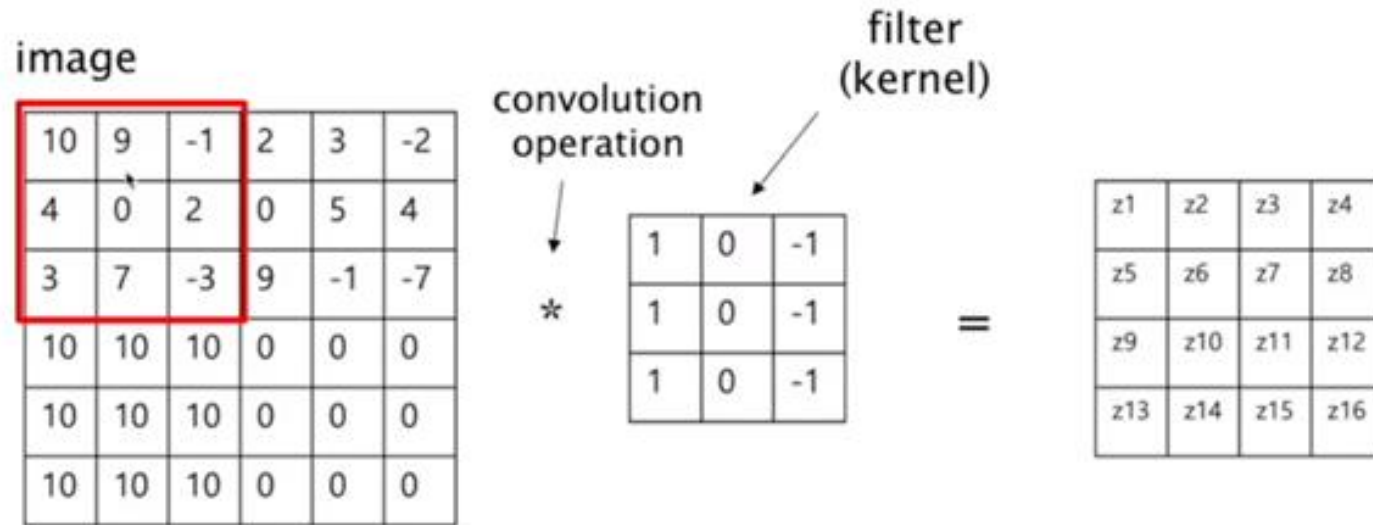
컴퓨터 비전에서 다음과 같이 선,모양 -> 부분 -> 전체 순으로 탐지한다.



이를 Convolution Operation을 이용하여 구조가 복잡하지 않은 선이나 모양(수직,수평)으로 나눌수있다.

Convolutional Neural Network (CNN) 개념

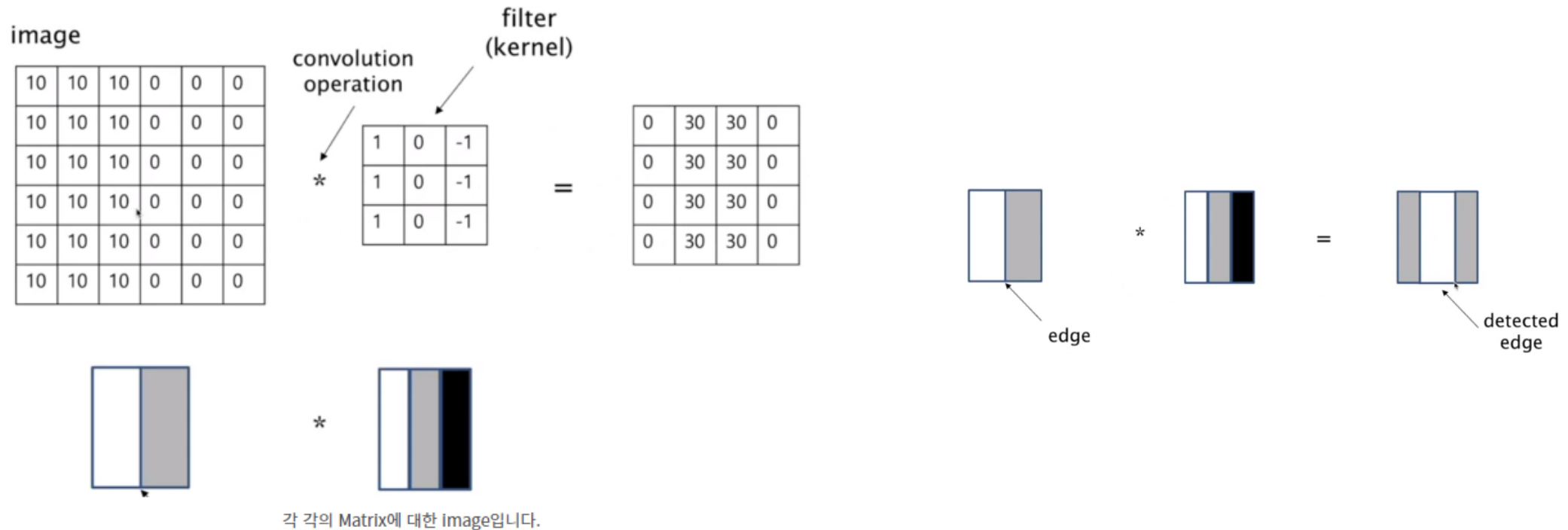
Convolution operation



이때 Matrix의 숫자는 색의 밝기를 표시합니다.

Convolution 연산자는 filter의 크기에 맞는 image의 크기만큼을 곱하여 구합니다.

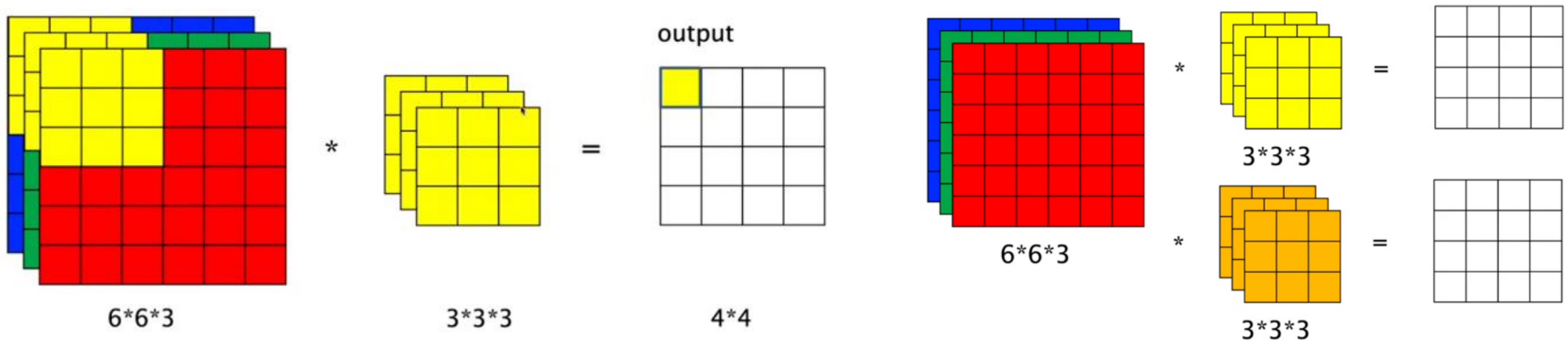
Convolutional Neural Network (CNN) 개념



*Convolutional layer*는 신경망에서 퓨터비전의 알고리즘 (filter를 사용하여 이미지에서 우리가 원하는 단위 edge(또는 shape)를 새로 생성하는 것)을 사용한 것 입니다.

Convolutional Neural Network (CNN) 개념

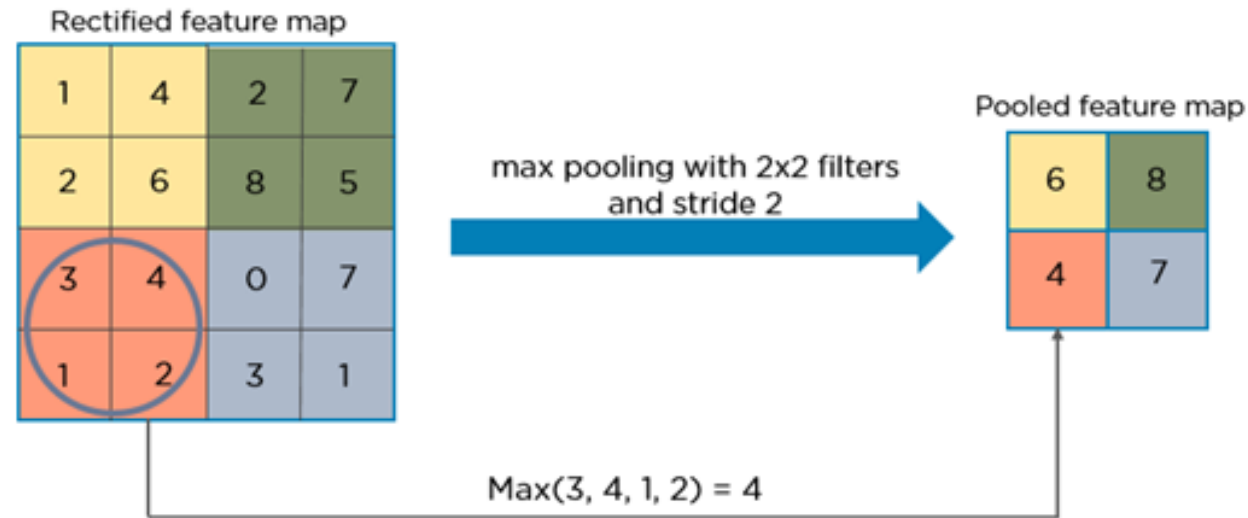
Convolutions on multi - channel images



여러개의 채널이 있다면, filter도 그 채널의 개수만큼 있으며
convolution 연산자도 채널의 개수만큼 작동한다.
다양한 패턴을 한 이미지에서 알기 위해 여러가지의 filter를 사용할 수 있다.
(보통 convolution layer에서도 많은 filter를 사용함.)

Convolutional Neural Network (CNN) 개념

Pooling Layer



풀링레이어는 쉽게 말하면 input 이미지의 차원(dimension)을 감소시켜 이미지를 추상화(단순화) 시키는 것입니다.
이를 통해 후속 차원에서 학습 parameter의 개수를 줄이기 위한 목적이 있습니다.

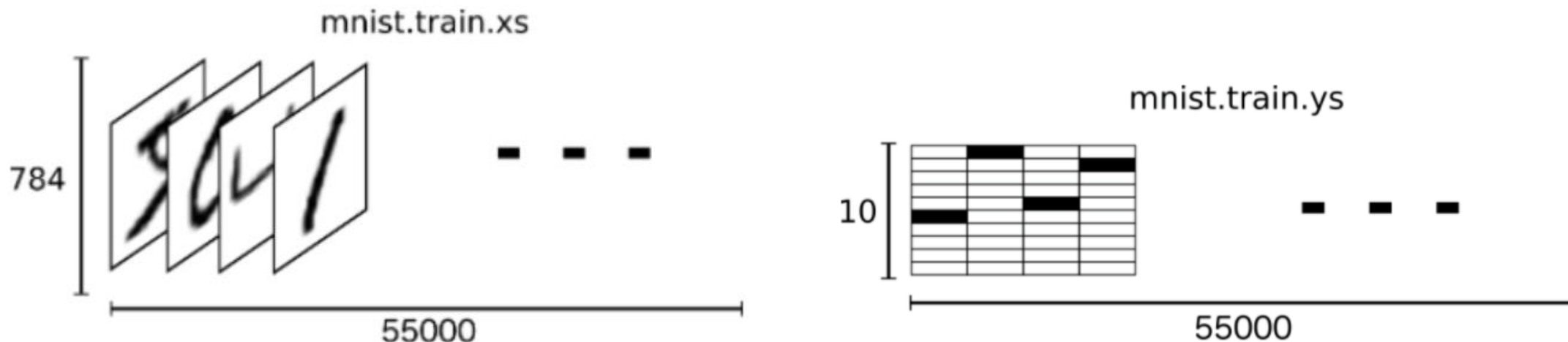
Mnist 기본 개념



데이터셋 명	행렬 차원	데이터 종류
mnist.train.images	55000 x 784	학습 이미지 데이터
mnist.train.labels	55000 x 10	학습 라벨 데이터
mnist.test.images	10000 x 784	테스트용 이미지 데이터
mnist.test.labels	10000 x 10	테스트용 라벨 데이터
mnist.validation.images	5000 x 784	확인용 이미지 데이터
mnist.validation.labels	5000 x 10	확인용 라벨 데이터

MNIST는 간단한 컴퓨터 비전 데이터 세트로, 위와 같이 손으로 쓰여진 이미지들로 구성되어 있다.
숫자는 0에서 1까지의 값을 갖는 고정 크기 이미지 (28x28 픽셀)로 크기 표준화되어 있다.

Mnist 데이터 구조



각 데이터는 위와 같이 학습용 데이터 55000개가 shape의 형태로 변환되어 shape=[55000,784]가 되며 라벨은 이미지가 나타내는 것이 어떤 숫자인지를 나타내는 데이터로 10개의 숫자로 이루어진 1행 행렬이므로 그 숫자이면 1 아니면 0으로 표현된다. (예를 들어 1인 경우에는 [0,1,0,0,0,0,0,0,0,0,0]로 표현)

Mnist 실습

```
[2] import numpy as np
import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Dense, Flatten
from keras.utils import to_categorical
from keras.datasets import mnist
```

✓
0초



```
[(X_train, y_train)], (X_test, y_test) = mnist.load_data()
print(X_train, '\n', '--'*2, '\n', y_train[2])
```

✓
0초

```
[4] print('X_train shape = ', X_train.shape)
print('y_train shape = ', y_train.shape)
```

```
X_train shape = (60000, 28, 28)
y_train shape = (60000,)
```

✓
0초

```
[5] print('X_test shape = ', X_test.shape)
print('y_test shape = ', y_test.shape)
```

```
X_test shape = (10000, 28, 28)
y_test shape = (10000,)
```

Mnist 실습

```
0초 ▶ plt.imshow(X_train[0], cmap='gray')  
print('이 이미지의 값은', y_train[0], '입니다.')
```

☞ 이 이미지의 값은 5 입니다.



```
0초 [7] def Check_MNIST_data():  
      (X_train, y_train), (X_test, y_test) = mnist.load_data()  
  
      print(X_train, '\n', '--*2, '\n', y_train)  
      print(X_train[0])  
      print(y_train[0])  
      print('X_train shape = ', X_train.shape)  
      print('y_train shape = ', y_train.shape)  
      print('X_test shape = ', X_test.shape)  
      print('y_test shape = ', y_test.shape)  
      plt.imshow(X_train[0], cmap='gray')  
      print('이 이미지의 값은', y_train[0], '입니다.')
```

Mnist 실습

```
✓  
0초 [8] X_train = X_train.reshape(60000,28,28,1)  
      X_test = X_test.reshape(10000,28,28,1)
```

```
✓  
0초 ▶ y_train = to_categorical(y_train)  
      y_test = to_categorical(y_test)  
      print(y_train[0])  
  
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

Mnist 실습

```
model = Sequential([
    Input(shape = (28,28,1), name = 'input_layer'),
    Conv2D(32, kernel_size=3, activation= 'relu', name = 'cov_layer1'),
    MaxPooling2D(pool_size= 2),
    Flatten(),
    Dense(10, activation = 'softmax', name = '')
])
```

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
cov_layer1 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
flatten (Flatten)	(None, 5408)	0
(Dense)	(None, 10)	54090

```
=====
Total params: 54,410
Trainable params: 54,410
Non-trainable params: 0
=====
```

Mnist 실습


```
✓ 3분 [11] model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])  
      history = model.fit(X_train, y_train, validation_data=(X_test, y_test), batch_size=10, epochs=10)
```

```
Epoch 1/10  
6000/6000 [=====] - 25s 3ms/step - loss: 0.4735 - accuracy: 0.9435 - val_loss: 0.1343 - val_accuracy: 0.9623  
Epoch 2/10  
6000/6000 [=====] - 16s 3ms/step - loss: 0.1043 - accuracy: 0.9697 - val_loss: 0.1014 - val_accuracy: 0.9704  
Epoch 3/10  
6000/6000 [=====] - 17s 3ms/step - loss: 0.0878 - accuracy: 0.9742 - val_loss: 0.1178 - val_accuracy: 0.9708  
Epoch 4/10  
6000/6000 [=====] - 19s 3ms/step - loss: 0.0754 - accuracy: 0.9779 - val_loss: 0.1413 - val_accuracy: 0.9688  
Epoch 5/10  
6000/6000 [=====] - 16s 3ms/step - loss: 0.0691 - accuracy: 0.9802 - val_loss: 0.1833 - val_accuracy: 0.9682  
Epoch 6/10  
6000/6000 [=====] - 17s 3ms/step - loss: 0.0626 - accuracy: 0.9827 - val_loss: 0.1500 - val_accuracy: 0.9720  
Epoch 7/10  
6000/6000 [=====] - 17s 3ms/step - loss: 0.0601 - accuracy: 0.9840 - val_loss: 0.1642 - val_accuracy: 0.9730  
Epoch 8/10  
6000/6000 [=====] - 16s 3ms/step - loss: 0.0624 - accuracy: 0.9841 - val_loss: 0.1879 - val_accuracy: 0.9697  
Epoch 9/10  
6000/6000 [=====] - 17s 3ms/step - loss: 0.0595 - accuracy: 0.9851 - val_loss: 0.2404 - val_accuracy: 0.9702  
Epoch 10/10  
6000/6000 [=====] - 16s 3ms/step - loss: 0.0579 - accuracy: 0.9862 - val_loss: 0.2632 - val_accuracy: 0.9691
```

Mnist 실습

```
✓ [12] def plot_loss_curve(history):  
0초      plt.figure(figsize = (5,3))  
  
      plt.plot(history['loss'])  
      plt.plot(history['val_loss'])  
  
      plt.title('model loss')  
      plt.xlabel('epoch')  
      plt.ylabel('loss')  
      plt.legend(['train','test'], loc = 'upper right')  
      plt.show()  
  
def plot_accuracy_curve(history):  
      plt.figure(figsize = (5,3))  
  
      plt.plot(history['accuracy'])  
      plt.plot(history['val_accuracy'])  
  
      plt.title('model Accuracy')  
      plt.xlabel('epoch')  
      plt.ylabel('Acc')  
      plt.legend(['train','test'], loc = 'upper right')  
      plt.show()
```


Mnist 실습

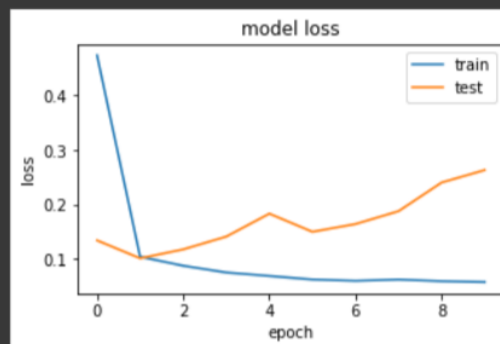
```
0초  plot_loss_curve(history.history)

print('train loss =', history.history['loss'][-1])
print('validation loss =', history.history['val_loss'][-1])

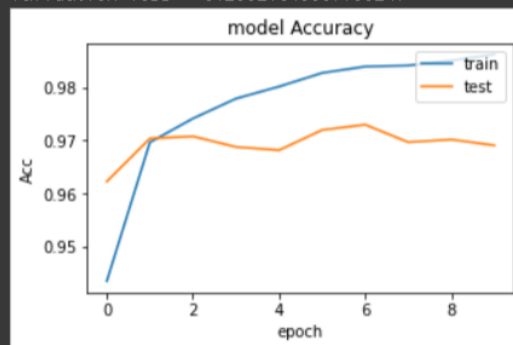
plot_accuracy_curve(history.history)

print('train acc =', history.history['accuracy'][-1])
print('validation acc =', history.history['val_accuracy'][-1])

print(history.history)
```



```
train loss = 0.05793749913573265
validation loss = 0.26321345567703247
```



```
train acc = 0.9862333536148071
validation acc = 0.9690999984741211
```

Mnist 실습

```
[17] def train_mnist_model():
    (X_train, y_train), (X_test, y_test) = mnist.load_data()
    X_train = X_train.reshape(60000, 28, 28, 1)
    X_test = X_test.reshape(10000, 28, 28, 1)
    y_train = to_categorical(y_train)
    y_test = to_categorical(y_test)

    model = Sequential([
        Input(shape = (28, 28, 1), name = 'input_layer_1'),

        Conv2D(32, kernel_size=3, activation = 'relu', name = 'conv_layer_2'),
        MaxPooling2D(pool_size=2, name = 'Pooling_layer_3'),

        Conv2D(64, kernel_size=3, activation='relu', name = 'conv_layer_4'),
        MaxPooling2D(pool_size=2, name = 'Pooling_layer5'),

        Flatten(),
        Dense(16, activation = 'relu', name = 'FC_layer_6'),
        Dense(10, activation = 'softmax', name = 'Output_layer_7')
    ])

    model.summary()

    model.compile(optimizer = 'adam', loss='categorical_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, validation_data=(X_test, y_test), batch_size=20, epochs = 10)

    plot_loss_curve(history.history)
    print('train loss={}, validation loss = {}'.format(history.history['loss'][-1], history.history['val_loss'][-1]))

    plot_accuracy_curve(history.history)
    print('train Acc = {}, validation Acc = {}'.format(history.history['accuracy'][-1], history.history['val_accuracy'][-1]))

    model.save('mnist_model')
    return model

if __name__ == '__main__':
    train_mnist_model()
```

Mnist 실습

Model: "sequential_2"

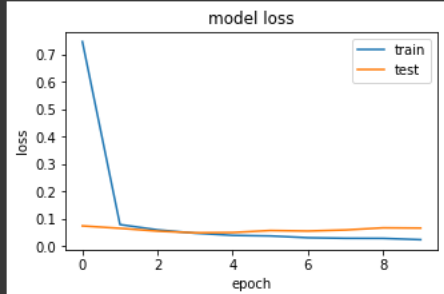
Layer (type)	Output Shape	Param #
conv_layer_2 (Conv2D)	(None, 26, 26, 32)	320
Pooling_layer_3 (MaxPooling2D)	(None, 13, 13, 32)	0
conv_layer_4 (Conv2D)	(None, 11, 11, 64)	18496
Pooling_layer5 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten_2 (Flatten)	(None, 1600)	0
FC_layer_6 (Dense)	(None, 16)	25616
Output_layer_7 (Dense)	(None, 10)	170

=====
Total params: 44,602
Trainable params: 44,602
Non-trainable params: 0

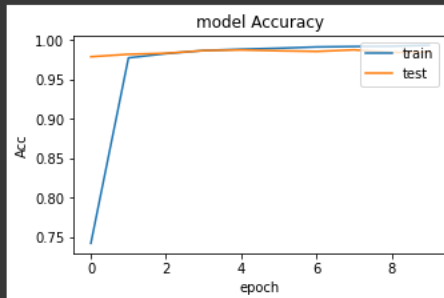
=====
Epoch 1/10
3000/3000 [=====] - 11s 3ms/step - loss: 0.7459 - accuracy: 0.7419 - val_loss: 0.0735 - val_accuracy: 0.9785
Epoch 2/10
3000/3000 [=====] - 10s 3ms/step - loss: 0.0783 - accuracy: 0.9771 - val_loss: 0.0647 - val_accuracy: 0.9816
Epoch 3/10
3000/3000 [=====] - 10s 3ms/step - loss: 0.0591 - accuracy: 0.9825 - val_loss: 0.0544 - val_accuracy: 0.9829
Epoch 4/10
3000/3000 [=====] - 11s 4ms/step - loss: 0.0470 - accuracy: 0.9864 - val_loss: 0.0484 - val_accuracy: 0.9863
Epoch 5/10
3000/3000 [=====] - 11s 4ms/step - loss: 0.0390 - accuracy: 0.9880 - val_loss: 0.0494 - val_accuracy: 0.9871
Epoch 6/10
3000/3000 [=====] - 10s 3ms/step - loss: 0.0368 - accuracy: 0.9893 - val_loss: 0.0566 - val_accuracy: 0.9860
Epoch 7/10
3000/3000 [=====] - 10s 3ms/step - loss: 0.0303 - accuracy: 0.9909 - val_loss: 0.0547 - val_accuracy: 0.9852
Epoch 8/10
3000/3000 [=====] - 10s 3ms/step - loss: 0.0283 - accuracy: 0.9915 - val_loss: 0.0586 - val_accuracy: 0.9872
Epoch 9/10
3000/3000 [=====] - 10s 3ms/step - loss: 0.0280 - accuracy: 0.9918 - val_loss: 0.0664 - val_accuracy: 0.9839
Epoch 10/10
3000/3000 [=====] - 11s 4ms/step - loss: 0.0233 - accuracy: 0.9931 - val_loss: 0.0655 - val_accuracy: 0.9844

Mnist 실습

0.000/0.000 t: 0.000 s / step: 1.000 s loss: 0.70260 accuracy: 0.75501 var_loss: 0.00000 var_accuracy: 0.00000



train loss=0.023307979106903076 , validation loss = 0.06554820388555527



train Acc = 0.9930833578109741, validation Acc = 0.9843999743461609

WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 2 of 2). These functions will not be directly callable after loading.

Q & A