```
%tensorflow_version 1.x
```

```
TensorFlow 1.x selected.
```

```
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import optimizers
from tensorflow.keras.layers import Dense, Flatten

import time
```

데이터 준비

데이터 다운로드

```
(raw_train_x, raw_train_y), (raw_test_x, raw_test_y) = tf.keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

```
print(raw_train_x.shape)
print(raw_train_y.shape)
print(raw_test_x.shape)
print(raw_test_y.shape)
```

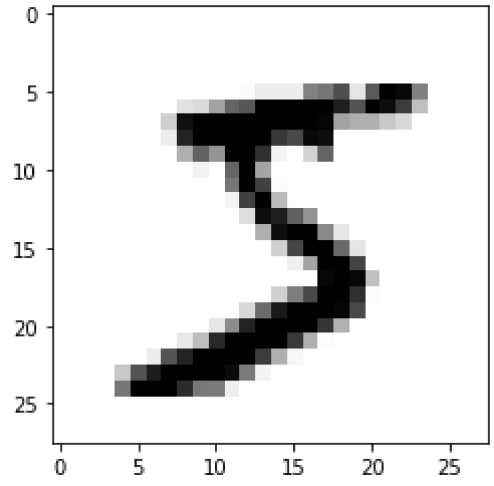
```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

```
print(raw_train_x[0])
print(raw_train_y[0])
```

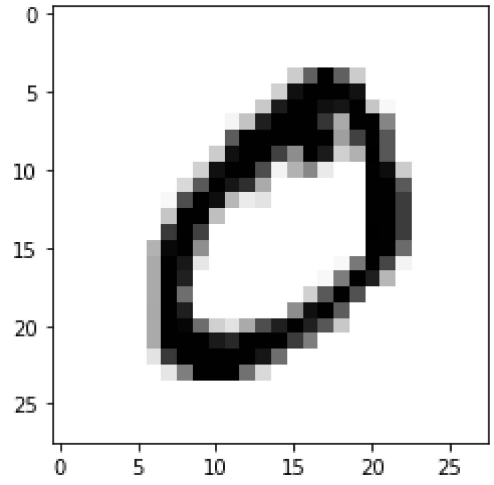
```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136
  175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
  225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251
   93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119
   25   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253
  150  27   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252
  253 187   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249
  253 249  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
  253 207   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253
  250 182   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201
   78   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0
```

```
for i in range(4):
  print(raw_train_y[i])
  plt.imshow(raw_train_x[i], cmap=plt.cm.binary)
  plt.show()
```

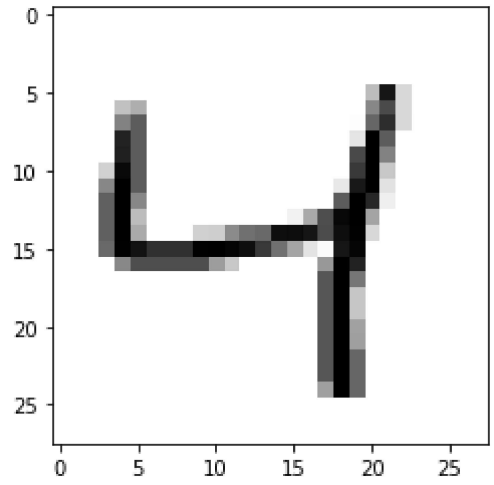
5



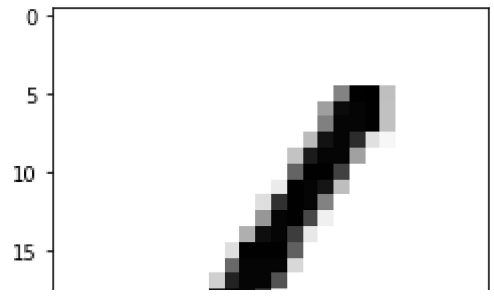
0



4



1



Normalization

```
print(np.max(raw_train_x[:,]))
print(np.max(raw_test_x[:,]))

train_x = raw_train_x/255
test_x = raw_test_x/255

train_y = raw_train_y
test_y = raw_test_y

print(np.max(train_x[:,]))
print(np.max(test_x[:,]))
```

```
255
255
1.0
1.0
```

```
data_count = train_x.shape[0]
data_size = train_x.shape[1]*train_x.shape[2]
train_x = train_x.reshape((data_count, data_size))

data_count = test_x.shape[0]
test_x = test_x.reshape((data_count, data_size))

print(train_x.shape)
print(test_x.shape)
```

```
(60000, 784)
(10000, 784)
```

모델 준비

```
model = keras.Sequential()
# model.add(Dense(10, activation='relu', input_shape=(4,))) #IRIS 문제는 input이 4개
model.add(Dense(10, activation='relu', input_shape=(28*28,)))
model.add(Dense(10, activation='relu'))
# model.add(Dense(3, activation='softmax')) #Iris 노드 수 3개
model.add(Dense(10, activation='softmax'))

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 10)                7850
```

```
# start_time = time.time() # REMOVED
model.fit(train_x, train_y, epochs=5, verbose=1, batch_size=128)
# print("elapsed : {}".format(time.time() - start_time)) # REMOVED
```

```
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 2s 27us/sample - loss: 0.8322 - acc: 0.7408
Epoch 2/5
60000/60000 [==============================] - 1s 21us/sample - loss: 0.3822 - acc: 0.8909
Epoch 3/5
60000/60000 [==============================] - 1s 23us/sample - loss: 0.3169 - acc: 0.9093
Epoch 4/5
60000/60000 [==============================] - 1s 23us/sample - loss: 0.2844 - acc: 0.9179
Epoch 5/5
60000/60000 [==============================] - 1s 23us/sample - loss: 0.2658 - acc: 0.9234
<tensorflow.python.keras.callbacks.History at 0x7f292215ccc0>
```

```
loss, acc = model.evaluate(test_x, test_y)
print("loss=",loss)
print("acc=",acc)
```

```
10000/10000 [==============================] - 0s 31us/sample - loss: 0.2724 - acc: 0.9208
loss= 0.2723938460960984
acc= 0.9208
```

```
model.fit(train_x, train_y, epochs=50, verbose=1, batch_size=128) #많이 학습시켜보자.
```

```
Epoch 17/50
60000/60000 [==============================] - 1s 23us/sample - loss: 0.1803 - acc: 0.9468
Epoch 18/50
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1783 - acc: 0.9482
Epoch 19/50
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1766 - acc: 0.9485
Epoch 20/50
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1750 - acc: 0.9484
Epoch 21/50
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1741 - acc: 0.9491
Epoch 22/50
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1727 - acc: 0.9493
Epoch 23/50
60000/60000 [==============================] - 1s 21us/sample - loss: 0.1707 - acc: 0.9490
Epoch 24/50
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1693 - acc: 0.9504
Epoch 25/50
60000/60000 [==============================] - 1s 21us/sample - loss: 0.1685 - acc: 0.9497
Epoch 26/50
60000/60000 [==============================] - 1s 23us/sample - loss: 0.1676 - acc: 0.9505
Epoch 27/50
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1651 - acc: 0.9510
Epoch 28/50
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1650 - acc: 0.9511
Epoch 29/50
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1628 - acc: 0.9527
Epoch 30/50
60000/60000 [==============================] - 1s 23us/sample - loss: 0.1621 - acc: 0.9522
Epoch 31/50
60000/60000 [==============================] - 1s 23us/sample - loss: 0.1606 - acc: 0.9517
Epoch 32/50
60000/60000 [==============================] - 1s 23us/sample - loss: 0.1603 - acc: 0.9524
Epoch 33/50
60000/60000 [==============================] - 1s 23us/sample - loss: 0.1601 - acc: 0.9522
Epoch 34/50
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1585 - acc: 0.9532
Epoch 35/50
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1580 - acc: 0.9529
Epoch 36/50
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1569 - acc: 0.9532
Epoch 37/50
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1564 - acc: 0.9537
Epoch 38/50
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1554 - acc: 0.9544
Epoch 39/50
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1543 - acc: 0.9538
Epoch 40/50
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1537 - acc: 0.9546
Epoch 41/50
60000/60000 [==============================] - 1s 23us/sample - loss: 0.1537 - acc: 0.9550
Epoch 42/50
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1529 - acc: 0.9547
Epoch 43/50
60000/60000 [==============================] - 1s 21us/sample - loss: 0.1520 - acc: 0.9544
Epoch 44/50
60000/60000 [==============================] - 1s 23us/sample - loss: 0.1509 - acc: 0.9552
Epoch 45/50
60000/60000 [==============================] - 1s 23us/sample - loss: 0.1509 - acc: 0.9555
```

```
Epoch 46/50
60000/60000 [==============================] - 1s 23us/sample - loss: 0.1500 - acc: 0.9554
Epoch 47/50
60000/60000 [==============================] - 1s 23us/sample - loss: 0.1500 - acc: 0.9554
Epoch 48/50
60000/60000 [==============================] - 1s 23us/sample - loss: 0.1483 - acc: 0.9561
Epoch 49/50
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1486 - acc: 0.9555
Epoch 50/50
60000/60000 [==============================] - 1s 22us/sample - loss: 0.1475 - acc: 0.9569
<tensorflow.python.keras.callbacks.History at 0x7f292127c860>
```

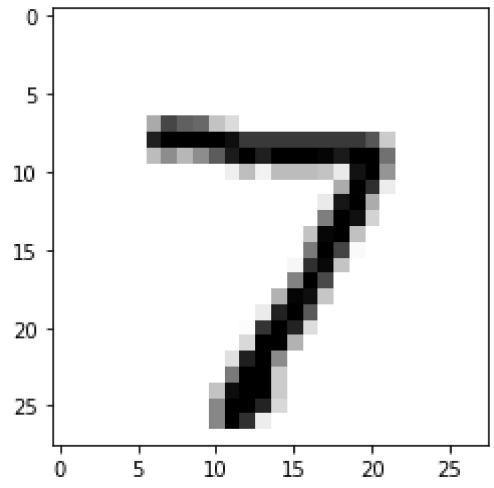
```
loss, acc = model.evaluate(test_x, test_y)
print("loss=",loss)
print("acc=",acc)
```

```
10000/10000 [==============================] - 0s 28us/sample - loss: 0.2107 - acc: 0.9430
loss= 0.2106697552215308
acc= 0.943
```

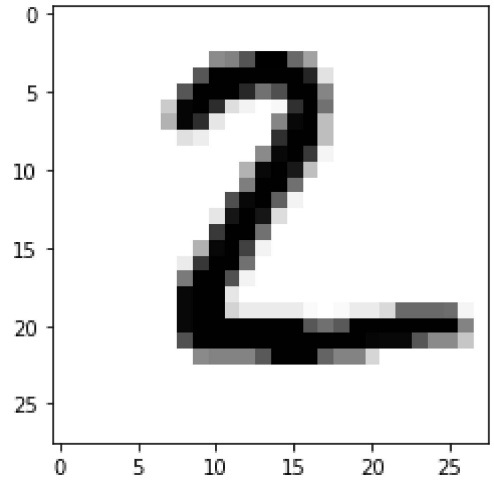
```
y_ = model.predict(test_x)
print(y_)
predicted = np.argmax(y_, axis=1)
print(predicted)

for i in range(4):
  print(raw_test_y[i])
  plt.imshow(raw_test_x[i], cmap=plt.cm.binary)
  plt.show()
```

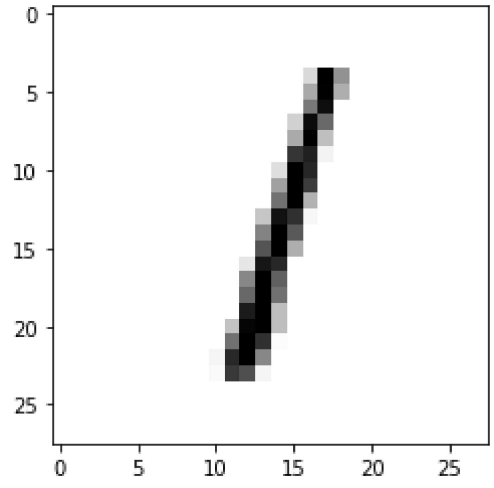
```
 [9.6932172e-06 9.9503458e-01 1.5572500e-03 ... 3.8112371e-04
  1.5355010e-03 1.5037402e-04]
 ...
 [3.2623584e-09 1.9872966e-11 4.3333424e-11 ... 2.8887121e-06
  5.1627366e-04 2.7427969e-03]
 [5.7646332e-10 4.4972080e-07 4.0589733e-11 ... 1.5566009e-07
  5.6155659e-03 2.6400019e-06]
 [1.9493720e-08 2.6645808e-13 1.8818086e-07 ... 4.3304992e-20
  2.0674543e-07 2.1663005e-10]]
[7 2 1 ... 4 5 6]
7
```



```
2
```



```
1
```



```
0
```

한눈에 모아보면

```
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import optimizers
from tensorflow.keras.layers import Dense

import time


(raw_train_x, raw_train_y), (raw_test_x, raw_test_y) = tf.keras.datasets.mnist.load_data()

train_x = raw_train_x/255
test_x = raw_test_x/255

train_y = raw_train_y
test_y = raw_test_y

train_x = train_x.reshape((60000, 28*28))
test_x = test_x.reshape((10000, 28*28))



model = keras.Sequential()
model.add(Dense(10, activation='relu', input_shape=(28*28,)))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
model.summary()


model.fit(train_x, train_y, epochs=5, verbose=1, batch_size=128)


loss, acc = model.evaluate(test_x, test_y)
print("loss=",loss)
print("acc=",acc)

y_ = model.predict(test_x)
predicted = np.argmax(y_, axis=1)

print(predicted)
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 10)                7850
_________________________________________________________________
dense_7 (Dense)              (None, 10)                110
_________________________________________________________________
dense_8 (Dense)              (None, 10)                110
=================================================================
Total params: 8,070
Trainable params: 8,070
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 1s 24us/sample - loss: 1.0013 - acc: 0.6810
Epoch 2/5
60000/60000 [==============================] - 1s 23us/sample - loss: 0.4279 - acc: 0.8849
Epoch 3/5
60000/60000 [==============================] - 1s 24us/sample - loss: 0.3395 - acc: 0.9076
```

Flatten 레이어 사용

(None, 28, 28) shape의 train_x를 그대로 사용.

모델의 처음에 Flatten() 레이어를 두어, 입력 모양을 변경한다.

```
[7 2 1 ... 4 5 6]
```

```
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import optimizers
from tensorflow.keras.layers import Dense

import time


(raw_train_x, raw_train_y), (raw_test_x, raw_test_y) = tf.keras.datasets.mnist.load_data()

train_x = raw_train_x/255
test_x = raw_test_x/255

train_y = raw_train_y
test_y = raw_test_y

# data_count = train_x.shape[0] # COMMENT OUT
# data_size = train_x.shape[1]*train_x.shape[2] # COMMENT OUT
# train_x = train_x.reshape((data_count, data_size)) # COMMENT OUT

# data_count = test_x.shape[0] # COMMENT OUT
# test_x = test_x.reshape((data_count, data_size)) # COMMENT OUT
```

```
model = keras.Sequential()
model.add(Flatten(input_shape=(28, 28))) # ADD
# model.add(Dense(10, activation='relu', input_shape=(28*28,)))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
model.summary()


model.fit(train_x, train_y, epochs=5, verbose=1, batch_size=128)


loss, acc = model.evaluate(test_x, test_y)
print("loss=",loss)
print("acc=",acc)

y_ = model.predict(test_x)
predicted = np.argmax(y_, axis=1)

print(predicted)
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                7850
_________________________________________________________________
dense_10 (Dense)             (None, 10)                110
_________________________________________________________________
dense_11 (Dense)             (None, 10)                110
=================================================================
Total params: 8,070
Trainable params: 8,070
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 1s 24us/sample - loss: 0.9492 - acc: 0.7060
Epoch 2/5
60000/60000 [==============================] - 1s 24us/sample - loss: 0.4059 - acc: 0.8831
Epoch 3/5
60000/60000 [==============================] - 1s 24us/sample - loss: 0.3178 - acc: 0.9090
Epoch 4/5
60000/60000 [==============================] - 1s 24us/sample - loss: 0.2793 - acc: 0.9203
Epoch 5/5
60000/60000 [==============================] - 1s 24us/sample - loss: 0.2594 - acc: 0.9264
10000/10000 [==============================] - 0s 33us/sample - loss: 0.2546 - acc: 0.9286
loss= 0.2545512982606888
acc= 0.9286
[7 2 1 ... 4 5 6]
```

DNN classification Template

```
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import optimizers
from tensorflow.keras.layers import Dense

import time


(train_x, train_y), (test_x, test_y) = tf.keras.datasets.mnist.load_data()

train_x = train_x/255
test_x = test_x/255


model = keras.Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
model.summary()


model.fit(train_x, train_y, epochs=5, verbose=1, batch_size=128)


loss, acc = model.evaluate(test_x, test_y)
print("loss=",loss)
print("acc=",acc)

y_ = model.predict(test_x)
predicted = np.argmax(y_, axis=1)

print(predicted)
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_12 (Dense)             (None, 10)                7850
_________________________________________________________________
dense_13 (Dense)             (None, 10)                110
_________________________________________________________________
dense_14 (Dense)             (None, 10)                110
=================================================================
Total params: 8,070
Trainable params: 8,070
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 2s 25us/sample - loss: 0.7757 - acc: 0.7709
Epoch 2/5
60000/60000 [==============================] - 1s 24us/sample - loss: 0.3600 - acc: 0.8960
```

Normalization

```
60000/60000 [==============================] - 1s 25us/sample - loss: 0.2928 - acc: 0.9164
```

Normalization을 하지 않으면

위에서는 0 ~ 255의 값을 0 ~ 1로 normalization하여 학습했다.

이 과정을 생략하고 그대로 실행한다.

```
(train_x, train_y), (test_x, test_y) = tf.keras.datasets.mnist.load_data()

# train_x = train_x/255
# test_x = test_x/255
```

```
model = keras.Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
model.summary()
```

```
model.fit(train_x, train_y, epochs=5, verbose=1, batch_size=128)
```

```
loss, acc = model.evaluate(test_x, test_y)
print("loss=",loss)
print("acc=",acc)
```

```
print( acc= ,acc)

y_ = model.predict(test_x)
predicted = np.argmax(y_, axis=1)

print(predicted)
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_2 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_15 (Dense)             (None, 10)                7850
_________________________________________________________________
dense_16 (Dense)             (None, 10)                110
_________________________________________________________________
dense_17 (Dense)             (None, 10)                110
=================================================================
Total params: 8,070
Trainable params: 8,070
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 1s 22us/sample - loss: 2.6273 - acc: 0.3071
Epoch 2/5
60000/60000 [==============================] - 1s 21us/sample - loss: 1.5557 - acc: 0.4613
Epoch 3/5
60000/60000 [==============================] - 1s 21us/sample - loss: 1.2142 - acc: 0.6106
Epoch 4/5
60000/60000 [==============================] - 1s 21us/sample - loss: 0.9483 - acc: 0.7116
Epoch 5/5
60000/60000 [==============================] - 1s 21us/sample - loss: 0.8204 - acc: 0.7587
10000/10000 [==============================] - 0s 35us/sample - loss: 0.7457 - acc: 0.7956
loss= 0.745745251083374
acc= 0.7956
[7 2 1 ... 4 5 6]
```

학습이 진행되지만 normalization했을 때와 비교하면 더디게 진행된다.

아래는 0~1로 normalization을 했을 때

```
Epoch 1/5
60000/60000 [==============================] - 1s 20us/sample - loss: 0.8862 - acc: 0.7278
Epoch 2/5
60000/60000 [==============================] - 1s 20us/sample - loss: 0.3659 - acc: 0.8950
Epoch 3/5
60000/60000 [==============================] - 1s 20us/sample - loss: 0.3164 - acc: 0.9100
Epoch 4/5
60000/60000 [==============================] - 1s 20us/sample - loss: 0.2904 - acc: 0.9171
Epoch 5/5
60000/60000 [==============================] - 1s 20us/sample - loss: 0.2724 - acc: 0.9225
10000/10000 [==============================] - 0s 27us/sample - loss: 0.2613 - acc: 0.9265
```

```
loss= 0.26132496373653413
acc= 0.9265
```

-1 ~ 1로 Normalization 하면

```
(train_x, train_y), (test_x, test_y) = tf.keras.datasets.mnist.load_data()

# train_x = train_x/255
# test_x = test_x/255
train_x = train_x/127.5 - 1
test_x = test_x/127.5 - 1



model = keras.Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
model.summary()


model.fit(train_x, train_y, epochs=5, verbose=1, batch_size=128)


loss, acc = model.evaluate(test_x, test_y)
print("loss=",loss)
print("acc=",acc)

y_ = model.predict(test_x)
predicted = np.argmax(y_, axis=1)

print(predicted)
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_3 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_18 (Dense)             (None, 10)                7850
_________________________________________________________________
dense_19 (Dense)             (None, 10)                110
_________________________________________________________________
dense_20 (Dense)             (None, 10)                110
=================================================================
Total params: 8,070
Trainable params: 8,070
```

학습이 진행되지만 normalization했을 때와 비교하면 더디게 진행된다.

아래는 0~1로 normalization을 했을 때

```
Epoch 1/5
60000/60000 [==============================] - 1s 20us/sample - loss: 0.8862 - acc: 0.7278
Epoch 2/5
60000/60000 [==============================] - 1s 20us/sample - loss: 0.3659 - acc: 0.8950
Epoch 3/5
60000/60000 [==============================] - 1s 20us/sample - loss: 0.3164 - acc: 0.9100
Epoch 4/5
60000/60000 [==============================] - 1s 20us/sample - loss: 0.2904 - acc: 0.9171
Epoch 5/5
60000/60000 [==============================] - 1s 20us/sample - loss: 0.2724 - acc: 0.9225
10000/10000 [==============================] - 0s 27us/sample - loss: 0.2613 - acc: 0.9265
loss= 0.26132496373653413
acc= 0.9265
```

-255 ~ 255로 하면

```
(train_x, train_y), (test_x, test_y) = tf.keras.datasets.mnist.load_data()

# train_x = train_x/255
# test_x = test_x/255
train_x = train_x*2 - 255
test_x = test_x*2 - 255


model = keras.Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
model.summary()
```

```
model.summary()


model.fit(train_x, train_y, epochs=5, verbose=1, batch_size=128)


loss, acc = model.evaluate(test_x, test_y)
print("loss=",loss)
print("acc=",acc)

y_ = model.predict(test_x)
predicted = np.argmax(y_, axis=1)

print(predicted)
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_3 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_15 (Dense)             (None, 10)                7850
_________________________________________________________________
dense_16 (Dense)             (None, 10)                110
_________________________________________________________________
dense_17 (Dense)             (None, 10)                110
=================================================================
Total params: 8,070
Trainable params: 8,070
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
60000/60000 [==============================] - 1s 18us/sample - loss: 4.1636 - acc: 0.1091
Epoch 2/5
60000/60000 [==============================] - 1s 17us/sample - loss: 2.3027 - acc: 0.1123
Epoch 3/5
60000/60000 [==============================] - 1s 16us/sample - loss: 2.3011 - acc: 0.1126
Epoch 4/5
60000/60000 [==============================] - 1s 16us/sample - loss: 2.2933 - acc: 0.1176
Epoch 5/5
60000/60000 [==============================] - 1s 17us/sample - loss: 2.1994 - acc: 0.1655
10000/10000 [==============================] - 0s 27us/sample - loss: 2.1274 - acc: 0.1931
loss= 2.127422957611084
acc= 0.1931
[1 2 1 ... 1 1 1]
```

학습이 진행되지만 normalization 안했을 때 보다 더디게 진행된다.

아래는 normalization을 안했을 때

```
Epoch 1/5
60000/60000 [==============================] - 1s 18us/sample - loss: 2.4654 - acc: 0.1538
Epoch 2/5
60000/60000 [==============================] - 1s 19us/sample - loss: 1.9504 - acc: 0.2509
Epoch 3/5
```

```
60000/60000 [==============================] - 1s 17us/sample - loss: 1.7553 - acc: 0.3037
Epoch 4/5
60000/60000 [==============================] - 1s 18us/sample - loss: 1.6404 - acc: 0.3354
Epoch 5/5
60000/60000 [==============================] - 1s 18us/sample - loss: 1.5597 - acc: 0.3471
10000/10000 [==============================] - 0s 26us/sample - loss: 1.5352 - acc: 0.3417
loss= 1.5351685056686402
acc= 0.3417
```

0 ~ 0.5로 하면

```
(train_x, train_y), (test_x, test_y) = tf.keras.datasets.mnist.load_data()

# train_x = train_x/255
# test_x = test_x/255
train_x = train_x/255/2
test_x = test_x/255/2


model = keras.Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
model.summary()


model.fit(train_x, train_y, epochs=5, verbose=1, batch_size=128)


loss, acc = model.evaluate(test_x, test_y)
print("loss=",loss)
print("acc=",acc)

y_ = model.predict(test_x)
predicted = np.argmax(y_, axis=1)

print(predicted)
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_6 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_24 (Dense)             (None, 10)                7850
_________________________________________________________________
dense_25 (Dense)             (None, 10)                110
_________________________________________________________________
dense_26 (Dense)             (None, 10)                110
=================================================================
Total params: 8,070
Trainable params: 8,070
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
60000/60000 [==============================] - 1s 20us/sample - loss: 1.0266 - acc: 0.6823
Epoch 2/5
```

학습이 되며 0~1로 normalization했을 때 보다 살짝 더디다

아래는 0~1로 normalization을 했을 때

```
Epoch 1/5
60000/60000 [==============================] - 1s 20us/sample - loss: 0.8862 - acc: 0.7278
Epoch 2/5
60000/60000 [==============================] - 1s 20us/sample - loss: 0.3659 - acc: 0.8950
Epoch 3/5
60000/60000 [==============================] - 1s 20us/sample - loss: 0.3164 - acc: 0.9100
Epoch 4/5
60000/60000 [==============================] - 1s 20us/sample - loss: 0.2904 - acc: 0.9171
Epoch 5/5
60000/60000 [==============================] - 1s 20us/sample - loss: 0.2724 - acc: 0.9225
10000/10000 [==============================] - 0s 27us/sample - loss: 0.2613 - acc: 0.9265
loss= 0.26132496373653413
acc= 0.9265
```

0 ~ 2로 하면

```
(train_x, train_y), (test_x, test_y) = tf.keras.datasets.mnist.load_data()

# train_x = train_x/255
# test_x = test_x/255
train_x = train_x/255*2
test_x = test_x/255*2


model = keras.Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(10, activation='relu'))
```

```
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
model.summary()


model.fit(train_x, train_y, epochs=5, verbose=1, batch_size=128)


loss, acc = model.evaluate(test_x, test_y)
print("loss=",loss)
print("acc=",acc)

y_ = model.predict(test_x)
predicted = np.argmax(y_, axis=1)

print(predicted)
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_7 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_27 (Dense)             (None, 10)                7850
_________________________________________________________________
dense_28 (Dense)             (None, 10)                110
_________________________________________________________________
dense_29 (Dense)             (None, 10)                110
=================================================================
Total params: 8,070
Trainable params: 8,070
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
60000/60000 [==============================] - 1s 21us/sample - loss: 0.7630 - acc: 0.7654
Epoch 2/5
60000/60000 [==============================] - 1s 19us/sample - loss: 0.3363 - acc: 0.9045
Epoch 3/5
60000/60000 [==============================] - 1s 19us/sample - loss: 0.2965 - acc: 0.9152
Epoch 4/5
60000/60000 [==============================] - 1s 19us/sample - loss: 0.2724 - acc: 0.9212
Epoch 5/5
60000/60000 [==============================] - 1s 19us/sample - loss: 0.2548 - acc: 0.9267
10000/10000 [==============================] - 0s 31us/sample - loss: 0.2541 - acc: 0.9268
loss= 0.25408969558775424
acc= 0.9268
[7 2 1 ... 4 5 6]
```

학습이 되며 0~1로 normalization했을 때 보다 살짝 빠르다

아래는 0~1로 normalization을 했을 때