DeepLearning with MNIST

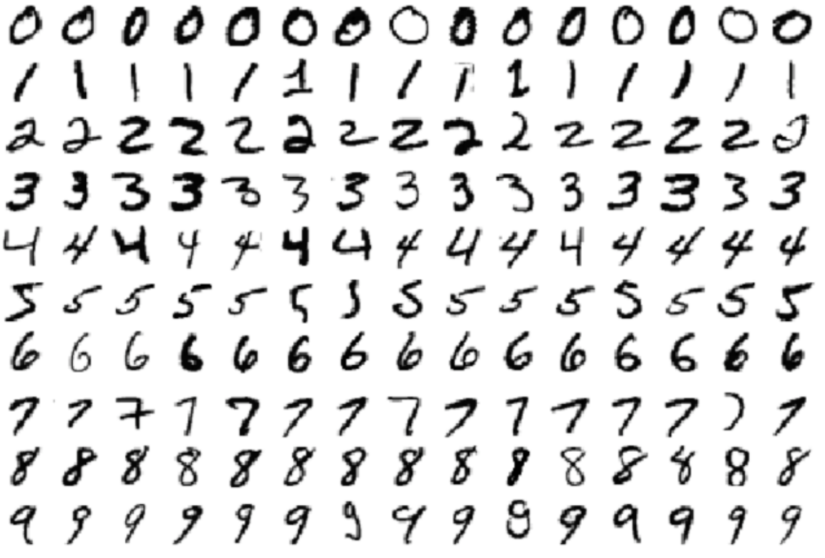
PinkLAB Edu

20 November, 2025

Table of Contents

1 MNIST 데이터

1.1 MNIST 데이터로 한 번 더 DL 과정을 복습해보자



- MNIST 데이터에 대해서는 kNN에서 설명을 했었다.

1.2 Tensorflow에서 MNSIT 읽기

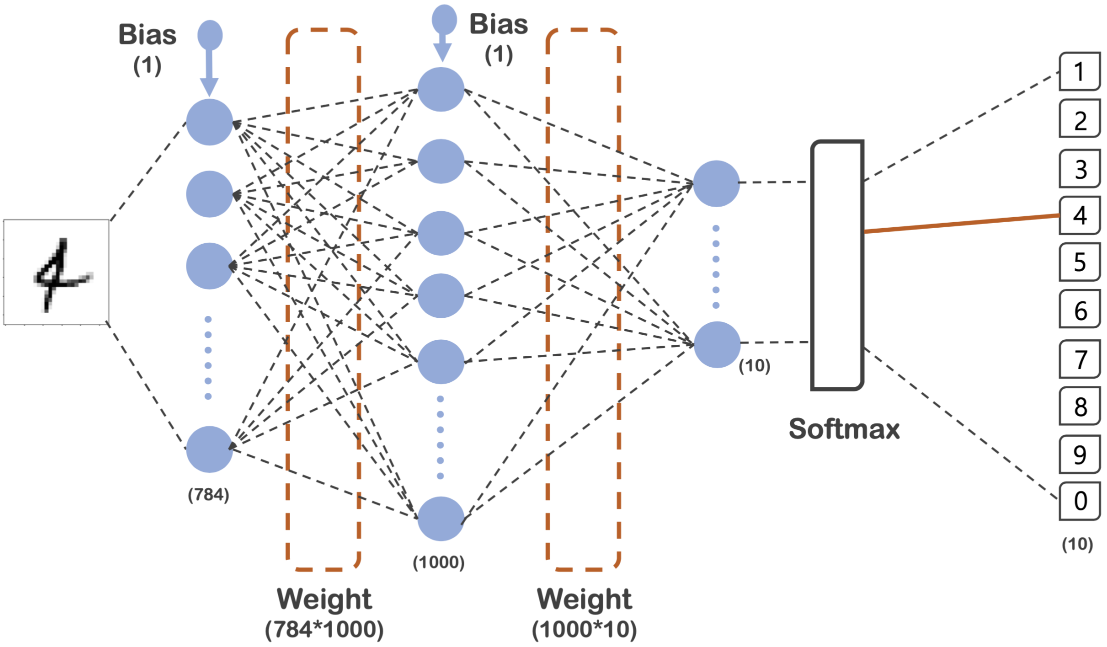
```python
import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

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

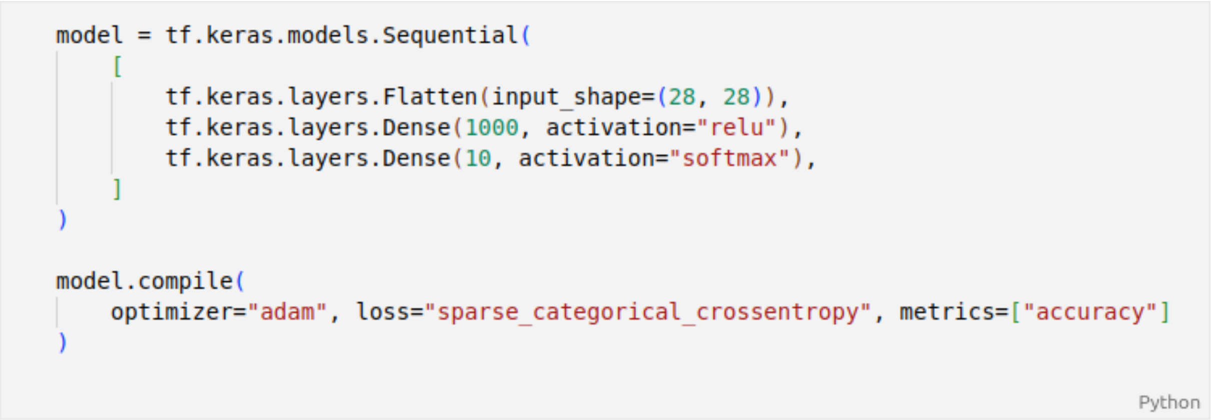
1.3 one-hot-encoding

- 이 타이밍에 one-hot-encoding을 해야 한다.
- 그런데 또 하나의 방법이 loss 함수를 `sparse_categorical_crossentropy` 로 설정하면 같은 효과이다.
- 그래서 pass~

1.4 이번 모델~

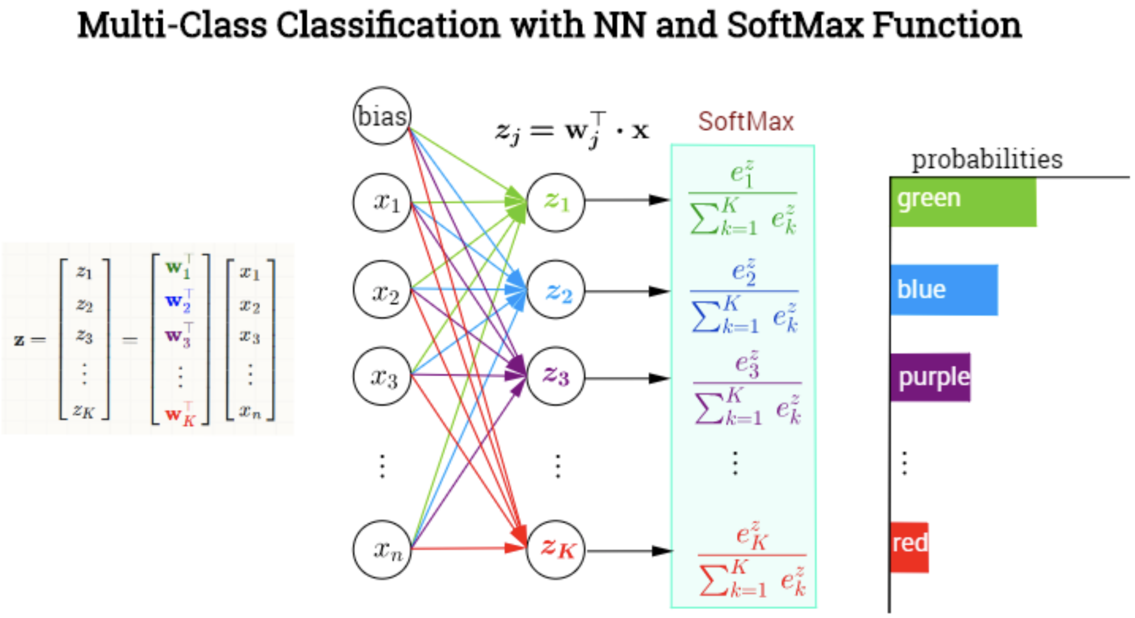


1.5 모델을 만들어보자

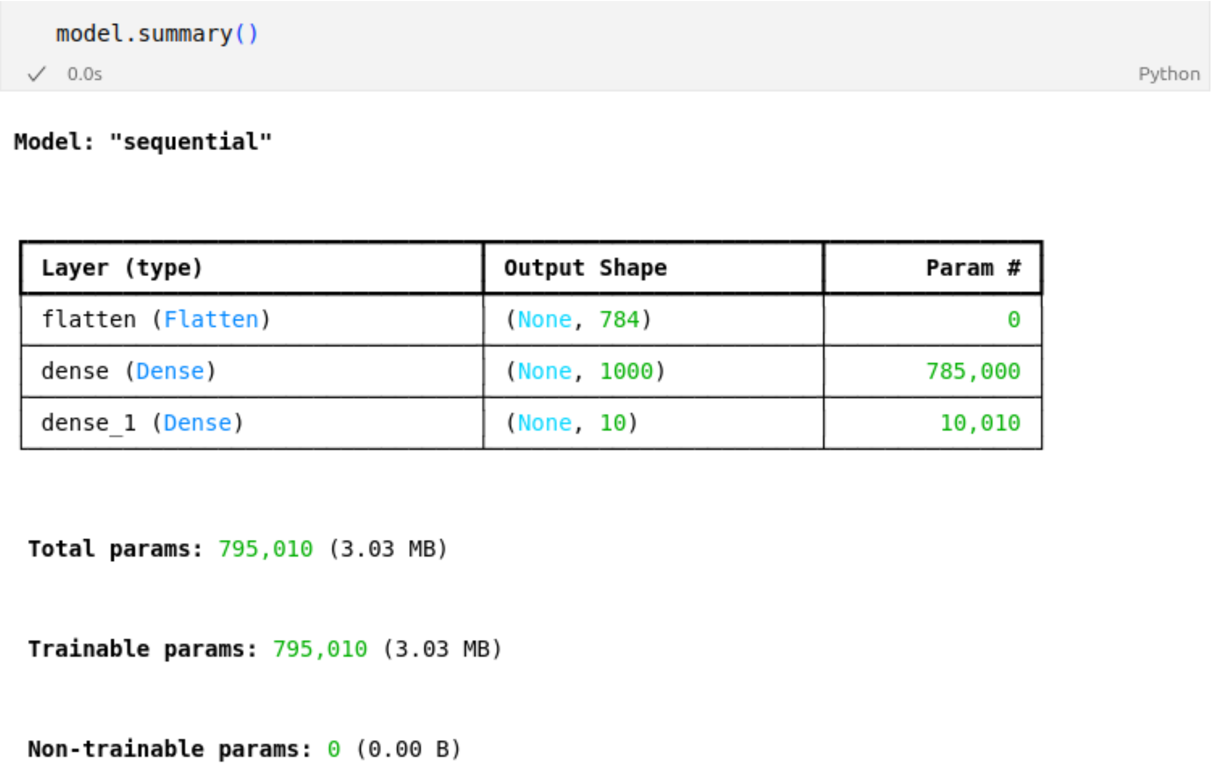


```
model = tf.keras.models.Sequential(
    [
        tf.keras.layers.Flatten(input_shape=(28, 28)),
        tf.keras.layers.Dense(1000, activation="relu"),
        tf.keras.layers.Dense(10, activation="softmax"),
    ]
)

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

1.6 다시 한 번 더 softmax란?


Multi-Class Classification with NN and SoftMax Function
$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$
bias
x_1
x_2
x_3
x_n
$z_j = \mathbf{w}_j^\top \cdot \mathbf{x}$
z_1
z_2
z_3
z_K
SoftMax
$\frac{e_1^z}{\sum_{k=1}^K e_k^z}$
$\frac{e_2^z}{\sum_{k=1}^K e_k^z}$
$\frac{e_3^z}{\sum_{k=1}^K e_k^z}$
$\frac{e_K^z}{\sum_{k=1}^K e_k^z}$
probabilities
green
blue
purple
red

1.7 model.summary

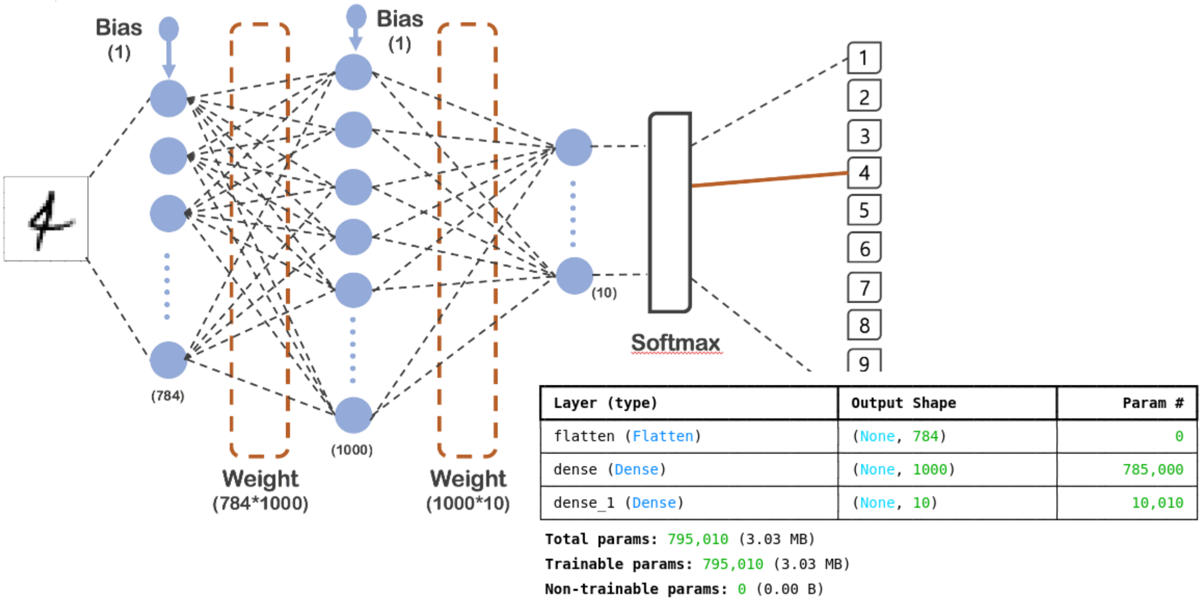


```python
model.summary()
```

✓ 0.0s Python

```
Model: "sequential"

┌─────────────────────────┬─────────────────────┬──────────┐
│ Layer (type)            │ Output Shape        │  Param # │
├─────────────────────────┼─────────────────────┼──────────┤
│ flatten (Flatten)       │ (None, 784)         │        0 │
├─────────────────────────┼─────────────────────┼──────────┤
│ dense (Dense)           │ (None, 1000)        │  785,000 │
├─────────────────────────┼─────────────────────┼──────────┤
│ dense_1 (Dense)         │ (None, 10)          │   10,010 │
└─────────────────────────┴─────────────────────┴──────────┘

 Total params: 795,010 (3.03 MB)

 Trainable params: 795,010 (3.03 MB)

 Non-trainable params: 0 (0.00 B)
```

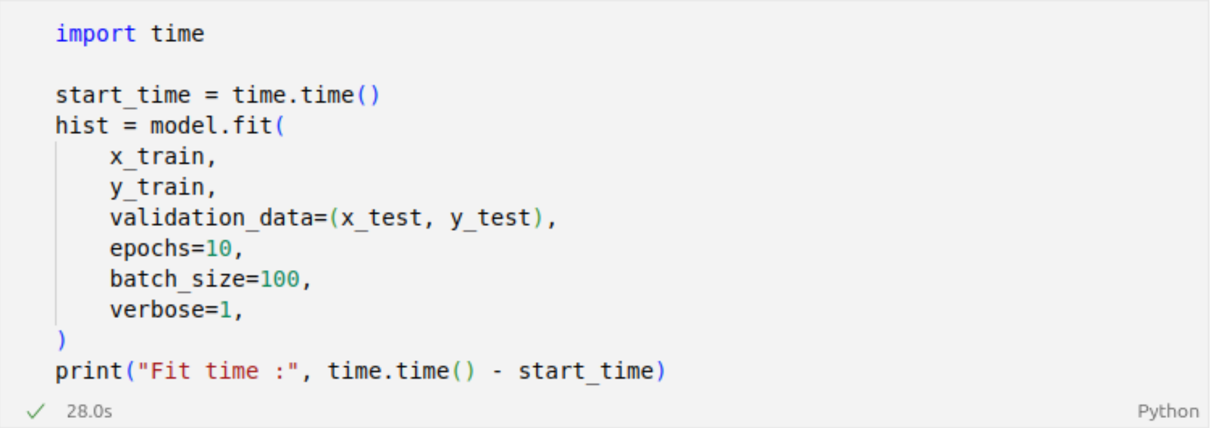
1.8 코드와 모델은 일치한다



Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 1000)	785,000
dense_1 (Dense)	(None, 10)	10,010

```
Total params: 795,010 (3.03 MB)
Trainable params: 795,010 (3.03 MB)
Non-trainable params: 0 (0.00 B)
```

1.9 fit~

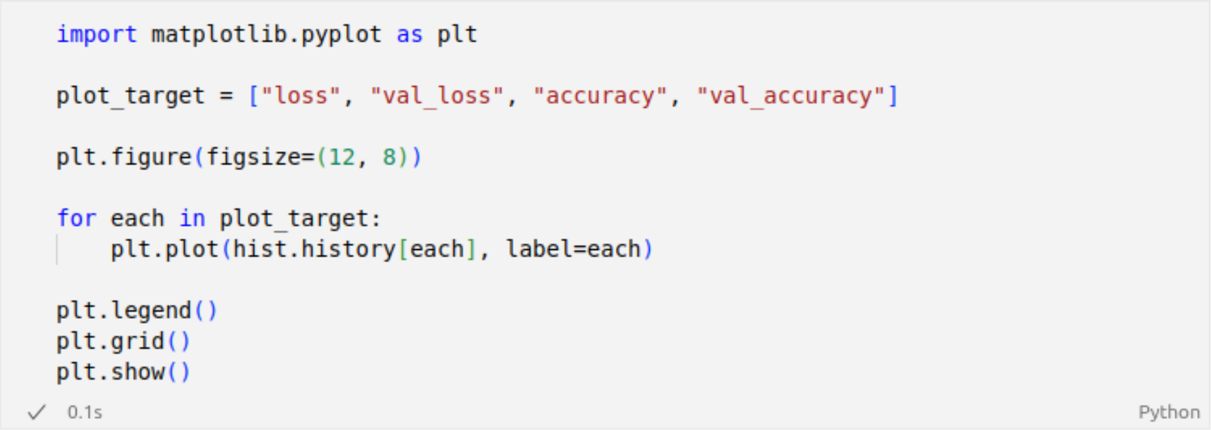


```python
import time

start_time = time.time()
hist = model.fit(
    x_train,
    y_train,
    validation_data=(x_test, y_test),
    epochs=10,
    batch_size=100,
    verbose=1,
)
print("Fit time :", time.time() - start_time)
```

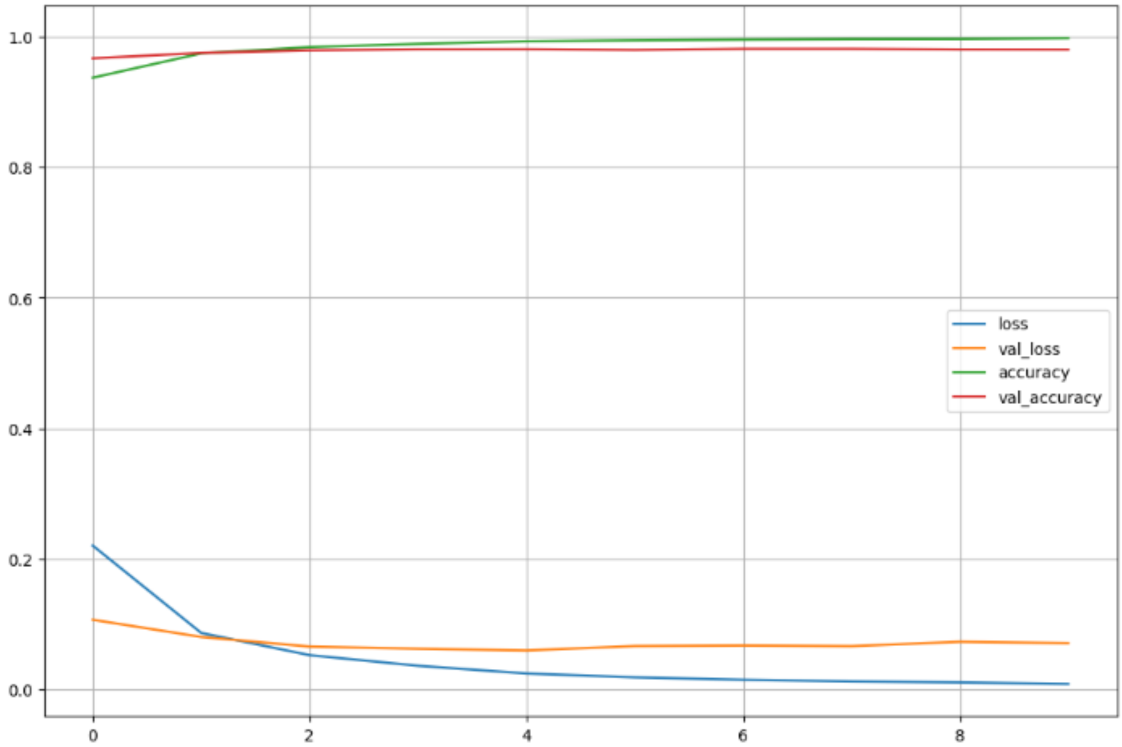
✓ 28.0s Python

```
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.9961 - loss: 0.0132 -
val_accuracy: 0.9812 - val_loss: 0.0672
Epoch 8/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.9968 - loss: 0.0108 -
val_accuracy: 0.9812 - val_loss: 0.0662
Epoch 9/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.9968 - loss: 0.0099 -
val_accuracy: 0.9803 - val_loss: 0.0729
Epoch 10/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.9976 - loss: 0.0076 -
val_accuracy: 0.9801 - val_loss: 0.0708
Fit time : 28.048307180404663
```

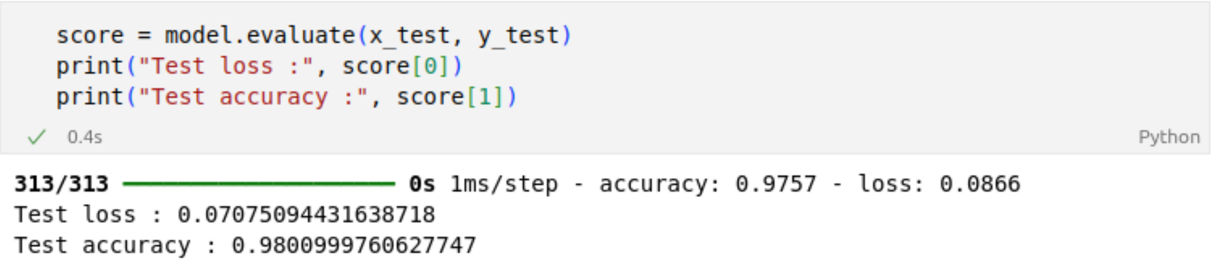
1.10 acc와 loss를 그려보자



```
import matplotlib.pyplot as plt

plot_target = ["loss", "val_loss", "accuracy", "val_accuracy"]

plt.figure(figsize=(12, 8))

for each in plot_target:
    plt.plot(hist.history[each], label=each)

plt.legend()
plt.grid()
plt.show()
```



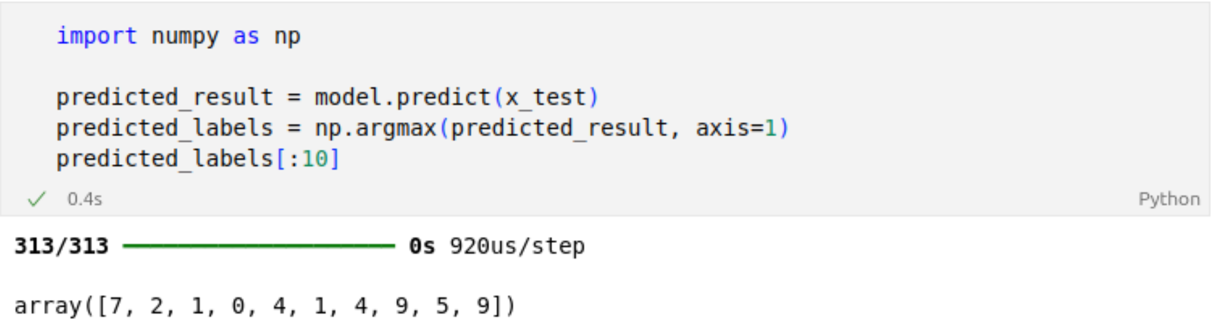
1.11 머신러닝에서 93%쯤 나왔던 결과대비 5%쯤 향상되었다



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

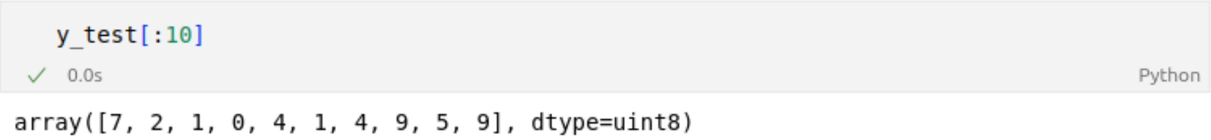
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9757 - loss: 0.0866
Test loss : 0.07075094431638718
Test accuracy : 0.9800999760627747
```

1.12 뭐가 틀렸나 확인해보자



```python
import numpy as np

predicted_result = model.predict(x_test)
predicted_labels = np.argmax(predicted_result, axis=1)
predicted_labels[:10]
```

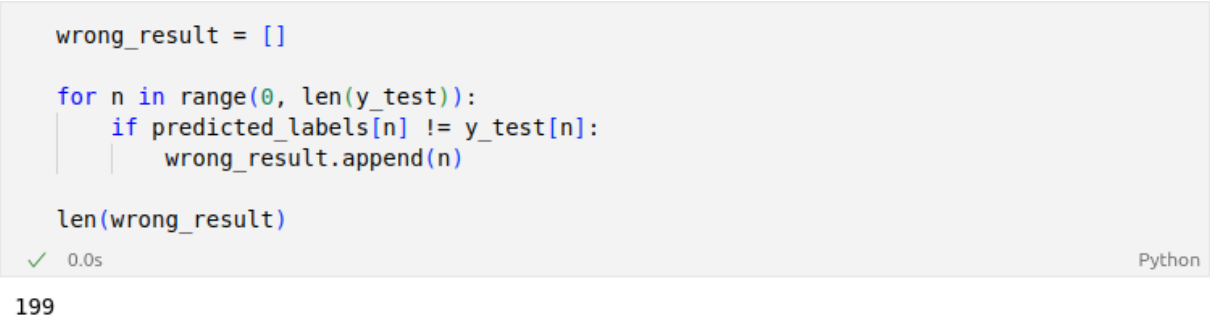
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 920us/step

array([7, 2, 1, 0, 4, 1, 4, 9, 5, 9])
```



```python
y_test[:10]
```

```
array([7, 2, 1, 0, 4, 1, 4, 9, 5, 9], dtype=uint8)
```

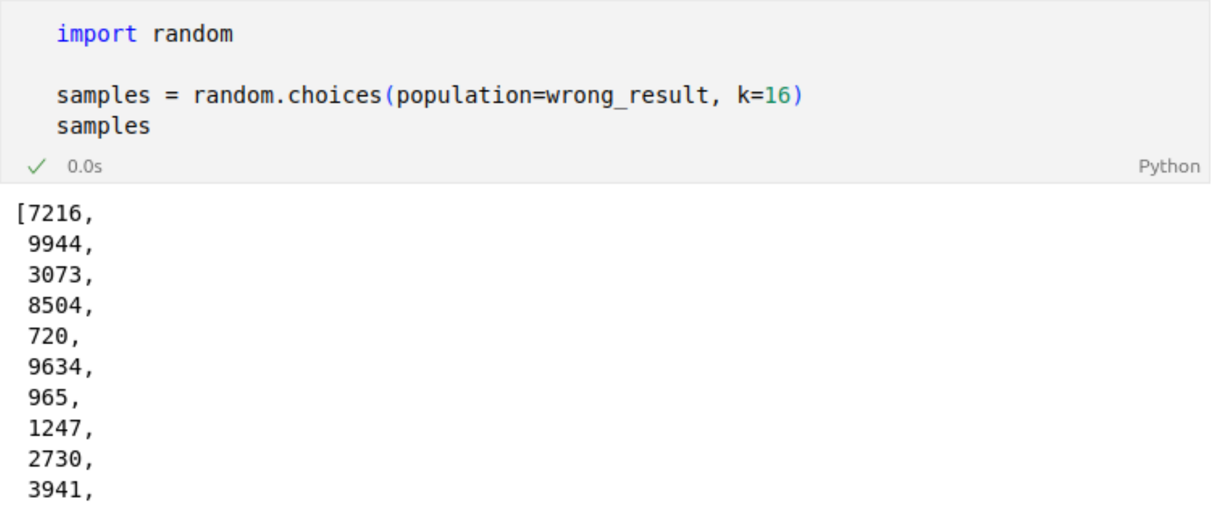
1.13 틀린 데이터의 인덱스만 모아서



```python
wrong_result = []

for n in range(0, len(y_test)):
    if predicted_labels[n] != y_test[n]:
        wrong_result.append(n)

len(wrong_result)
```

```
199
```

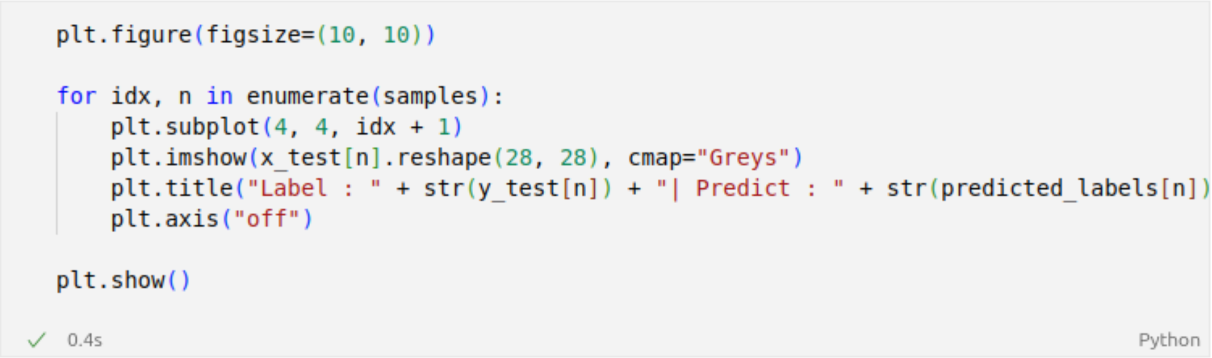
1.14 그중 16개만



```
import random

samples = random.choices(population=wrong_result, k=16)
samples
```

```
[7216,
 9944,
 3073,
 8504,
 720,
 9634,
 965,
 1247,
 2730,
 3941,
```

1.15 뭘 틀렸나?

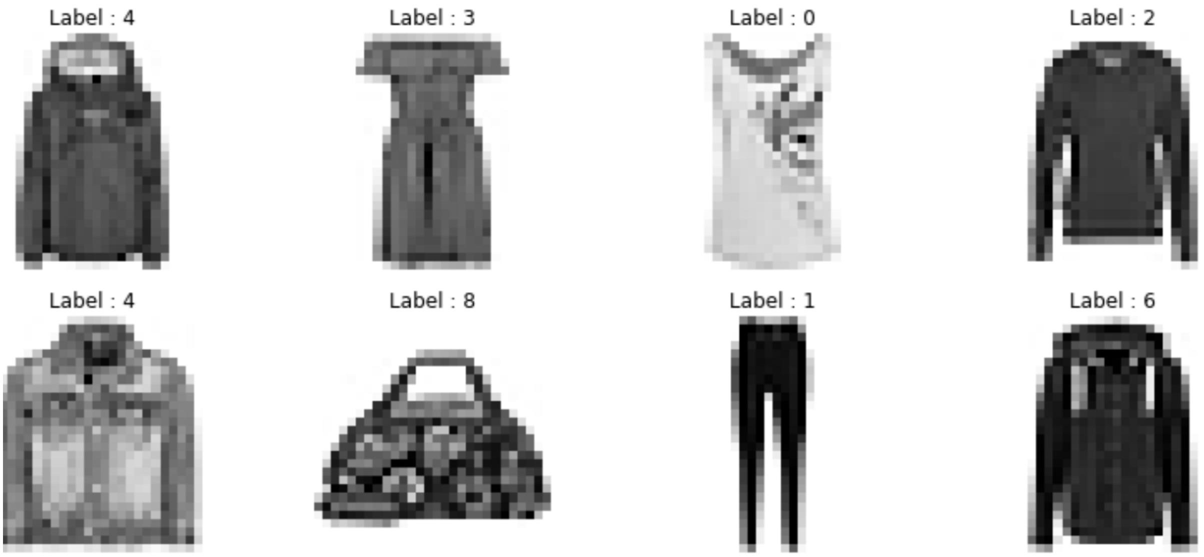


```
plt.figure(figsize=(10, 10))

for idx, n in enumerate(samples):
    plt.subplot(4, 4, idx + 1)
    plt.imshow(x_test[n].reshape(28, 28), cmap="Greys")
    plt.title("Label : " + str(y_test[n]) + "| Predict : " + str(predicted_labels[n]))
    plt.axis("off")

plt.show()
```

1.16 나도 틀릴만한게 있네

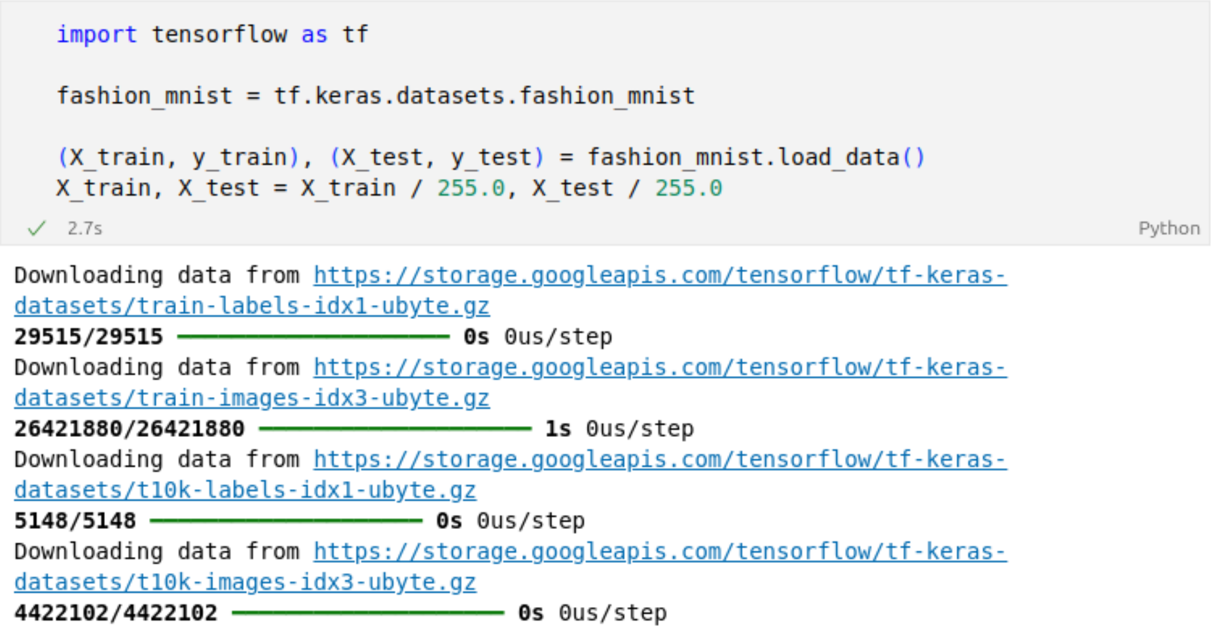
2 MNIST fashion

2.1 MNIST fashion data



- 숫자로 된 MNIST데이터처럼 28*28 크기의 패션과 관련된 10개 종류의 데이터

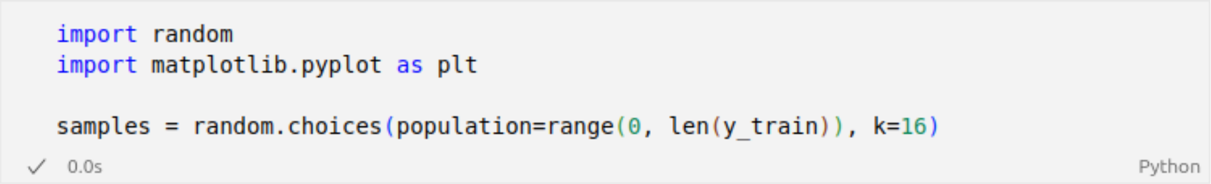
2.2 데이터 읽기



```python
import tensorflow as tf

fashion_mnist = tf.keras.datasets.fashion_mnist

(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0
```

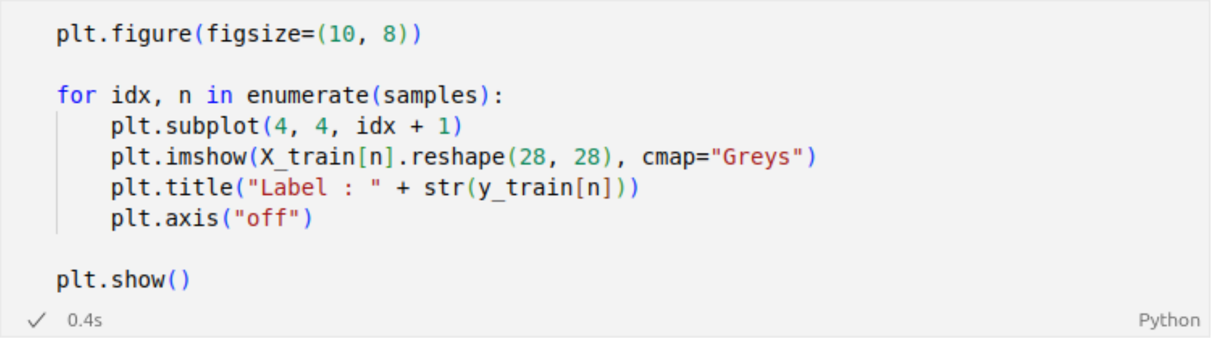
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-labels-idx1-ubyte.gz
29515/29515 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-images-idx3-ubyte.gz
26421880/26421880 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-labels-idx1-ubyte.gz
5148/5148 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

2.3 어떻게 생겼는지 확인해보자



```
import random
import matplotlib.pyplot as plt

samples = random.choices(population=range(0, len(y_train)), k=16)
```

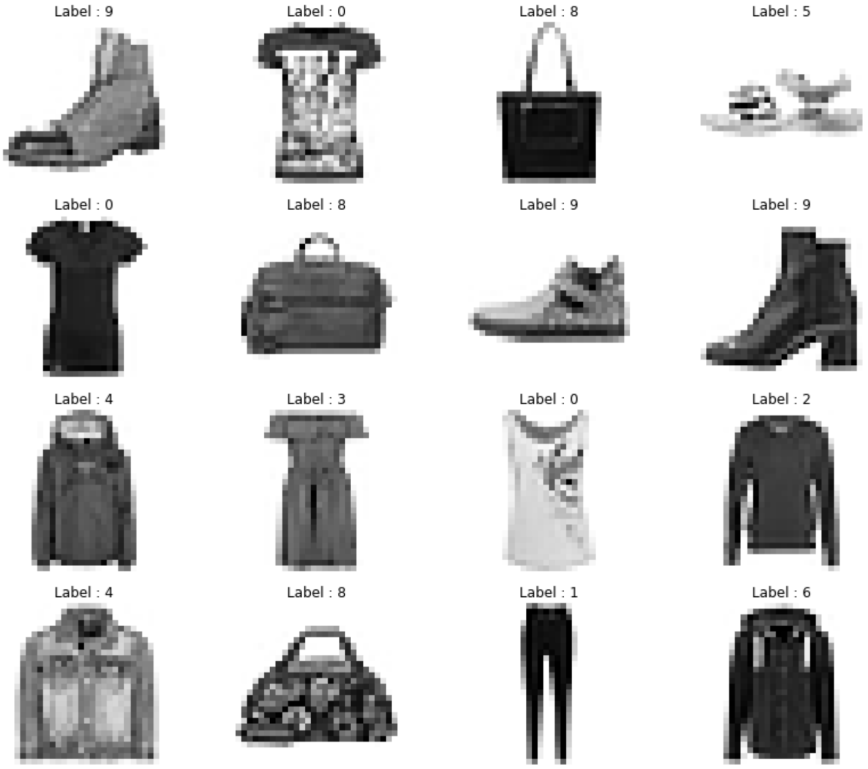
✓ 0.0s Python



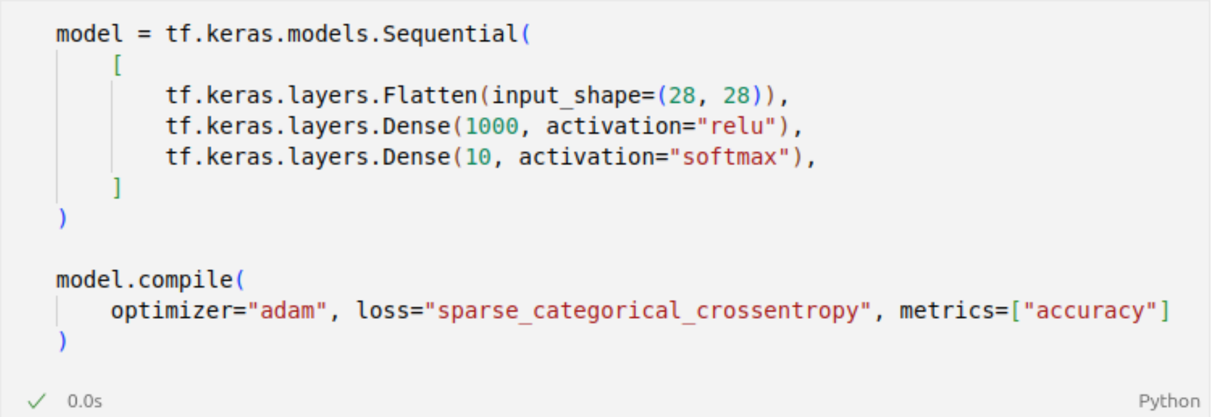
```
plt.figure(figsize=(10, 8))

for idx, n in enumerate(samples):
    plt.subplot(4, 4, idx + 1)
    plt.imshow(X_train[n].reshape(28, 28), cmap="Greys")
    plt.title("Label : " + str(y_train[n]))
    plt.axis("off")

plt.show()
```

✓ 0.4s Python

2.4 패션~



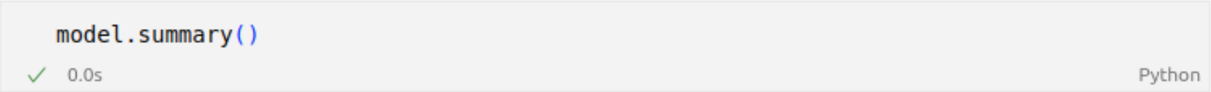
2.5 모델은 숫자때와 동일한 구조로 두자



```python
model = tf.keras.models.Sequential(
    [
        tf.keras.layers.Flatten(input_shape=(28, 28)),
        tf.keras.layers.Dense(1000, activation="relu"),
        tf.keras.layers.Dense(10, activation="softmax"),
    ]
)

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

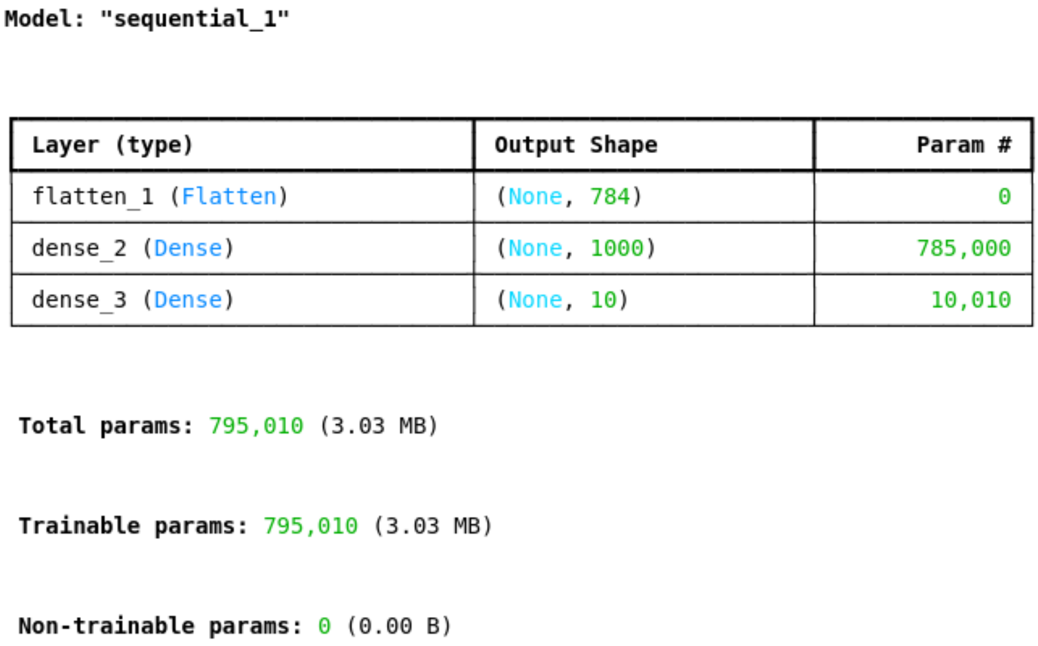
✓ 0.0s Python

2.6 summary



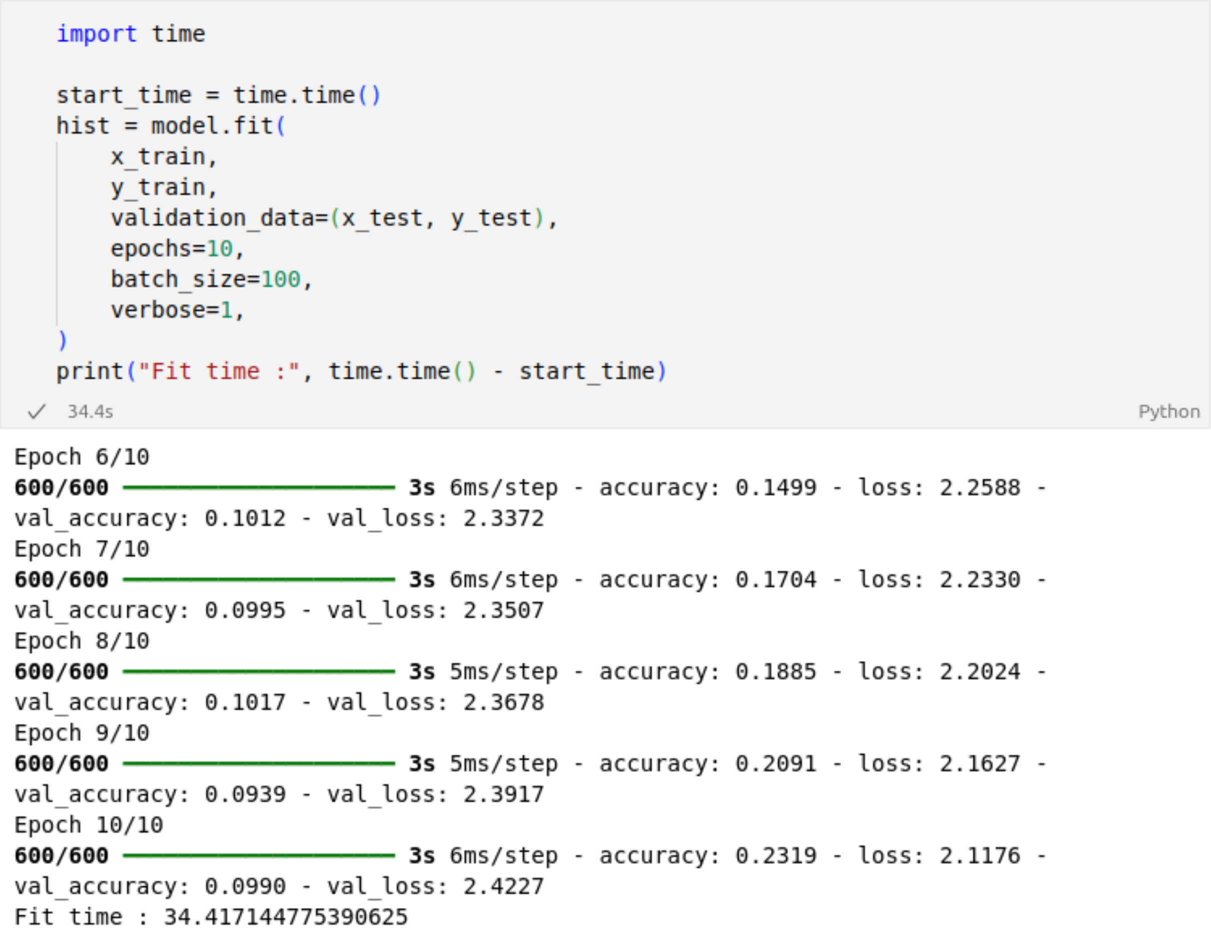
```python
model.summary()
```

✓ 0.0s Python



```
Model: "sequential_1"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ flatten_1 (Flatten)             │ (None, 784)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 1000)           │       785,000 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_3 (Dense)                 │ (None, 10)             │        10,010 │
└─────────────────────────────────┴────────────────────────┴───────────────┘

 Total params: 795,010 (3.03 MB)

 Trainable params: 795,010 (3.03 MB)

 Non-trainable params: 0 (0.00 B)
```

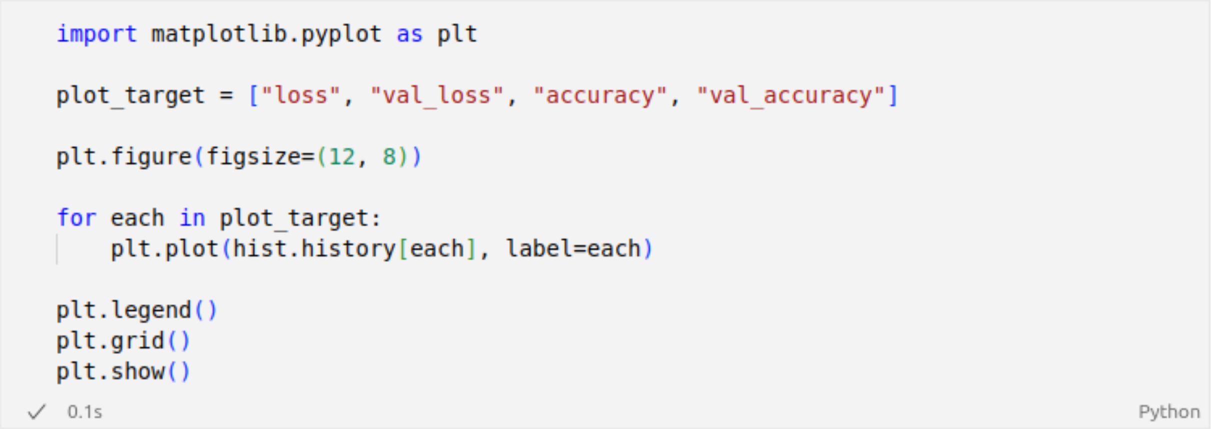
2.7 fit~~



```python
import time

start_time = time.time()
hist = model.fit(
    x_train,
    y_train,
    validation_data=(x_test, y_test),
    epochs=10,
    batch_size=100,
    verbose=1,
)
print("Fit time :", time.time() - start_time)
```

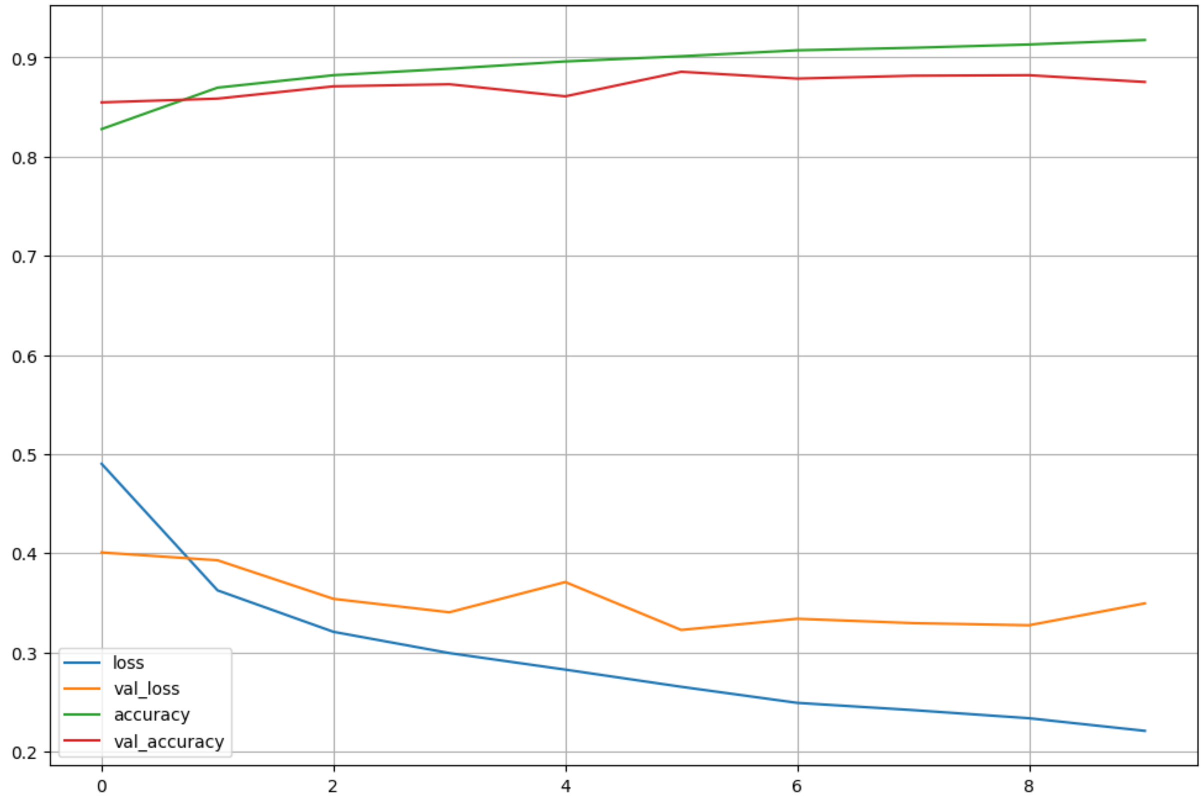
```
Epoch 6/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.1499 - loss: 2.2588 -
val_accuracy: 0.1012 - val_loss: 2.3372
Epoch 7/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.1704 - loss: 2.2330 -
val_accuracy: 0.0995 - val_loss: 2.3507
Epoch 8/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.1885 - loss: 2.2024 -
val_accuracy: 0.1017 - val_loss: 2.3678
Epoch 9/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.2091 - loss: 2.1627 -
val_accuracy: 0.0939 - val_loss: 2.3917
Epoch 10/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.2319 - loss: 2.1176 -
val_accuracy: 0.0990 - val_loss: 2.4227
Fit time : 34.417144775390625
```

2.8 학습 상황을 관찰해 보자



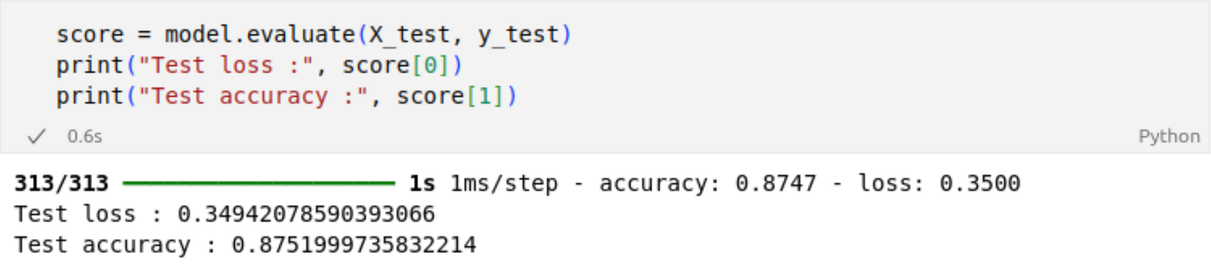
```python
import matplotlib.pyplot as plt

plot_target = ["loss", "val_loss", "accuracy", "val_accuracy"]

plt.figure(figsize=(12, 8))

for each in plot_target:
    plt.plot(hist.history[each], label=each)

plt.legend()
plt.grid()
plt.show()
```

2.9 잘 학습이 되는듯 하지만,



- val_loss와 train loss사이에 간격이 발생한다.

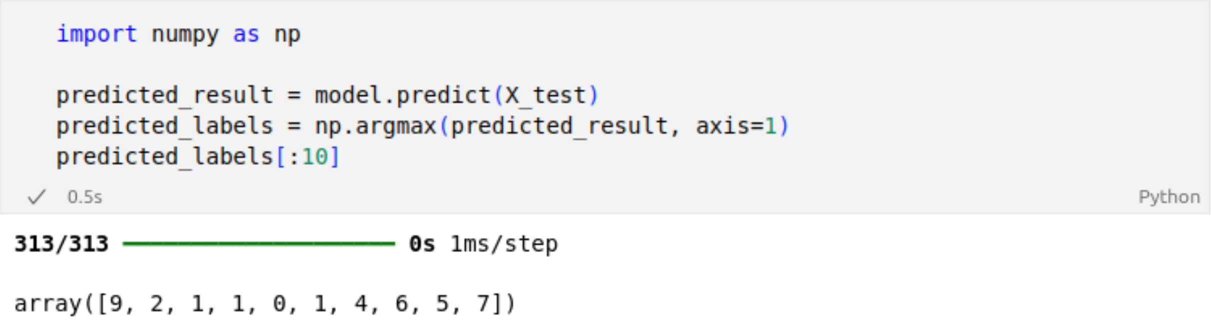
2.10 테스트데이터 accuracy



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

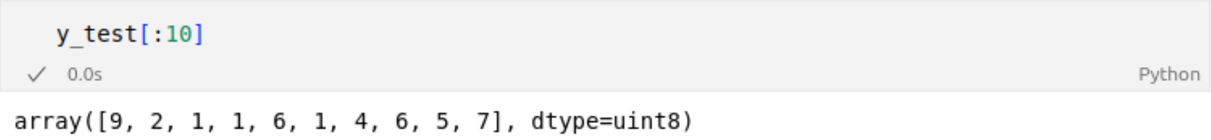
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8747 - loss: 0.3500
Test loss : 0.34942078590393066
Test accuracy : 0.8751999735832214
```

2.11 어떤 데이터가 틀렸는지 추출하고



```python
import numpy as np

predicted_result = model.predict(X_test)
predicted_labels = np.argmax(predicted_result, axis=1)
predicted_labels[:10]
```

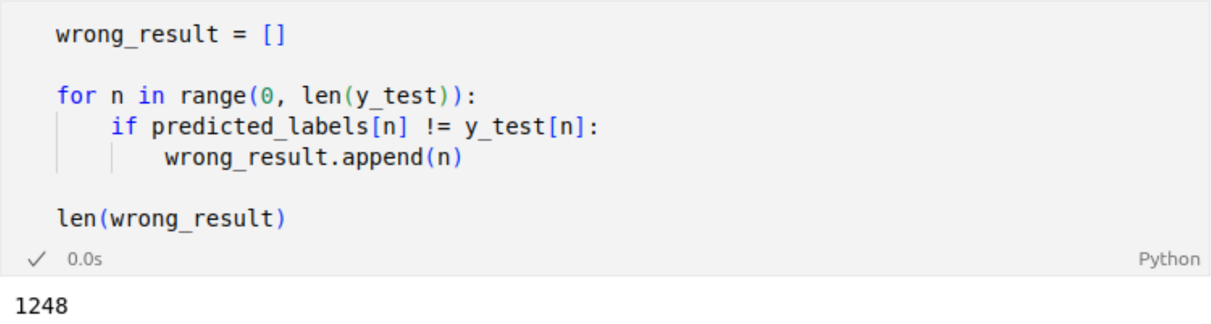
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step

array([9, 2, 1, 1, 0, 1, 4, 6, 5, 7])
```



```python
y_test[:10]
```

```
array([9, 2, 1, 1, 6, 1, 4, 6, 5, 7], dtype=uint8)
```

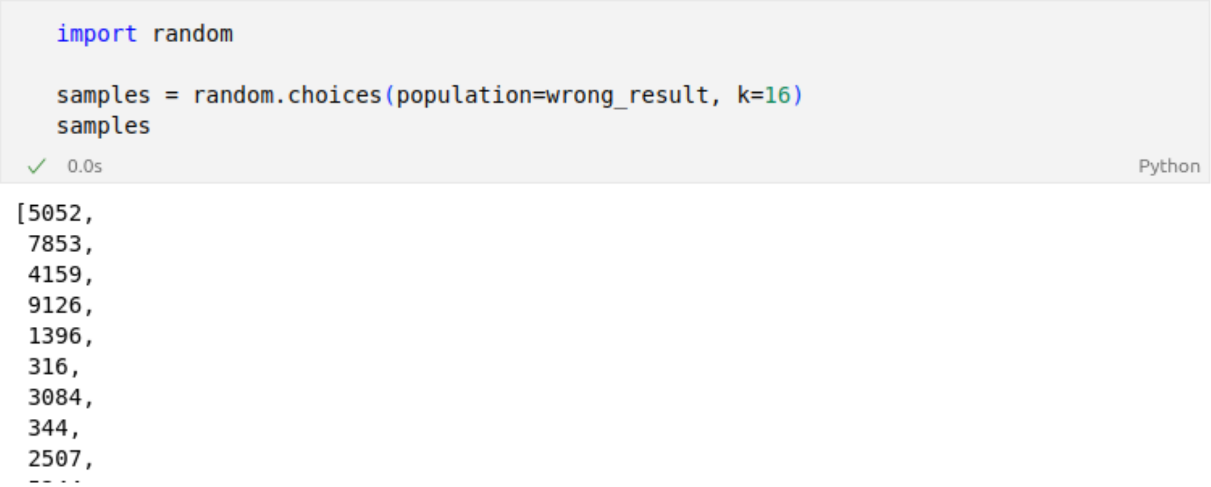
2.12 틀린 데이터를 모아서~



```python
wrong_result = []

for n in range(0, len(y_test)):
    if predicted_labels[n] != y_test[n]:
        wrong_result.append(n)

len(wrong_result)
```

```
1248
```

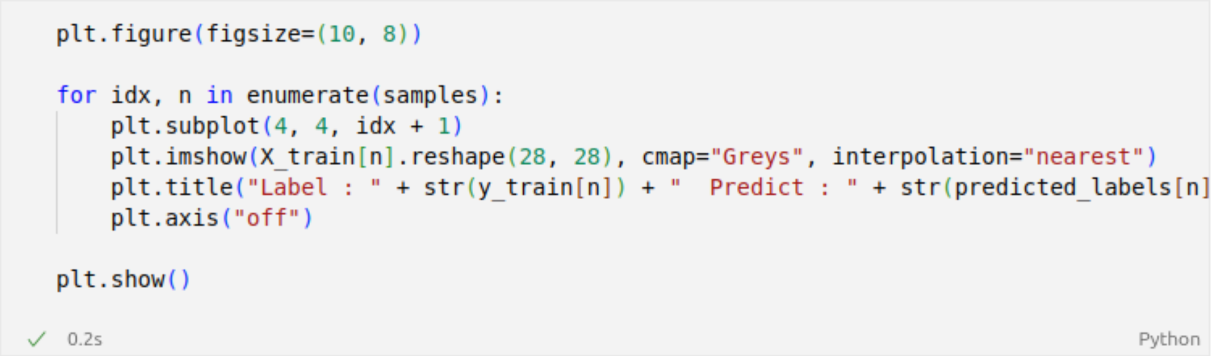
2.13 16개만 선택해서



```python
import random

samples = random.choices(population=wrong_result, k=16)
samples
```

```
[5052,
 7853,
 4159,
 9126,
 1396,
 316,
 3084,
 344,
 2507,
```

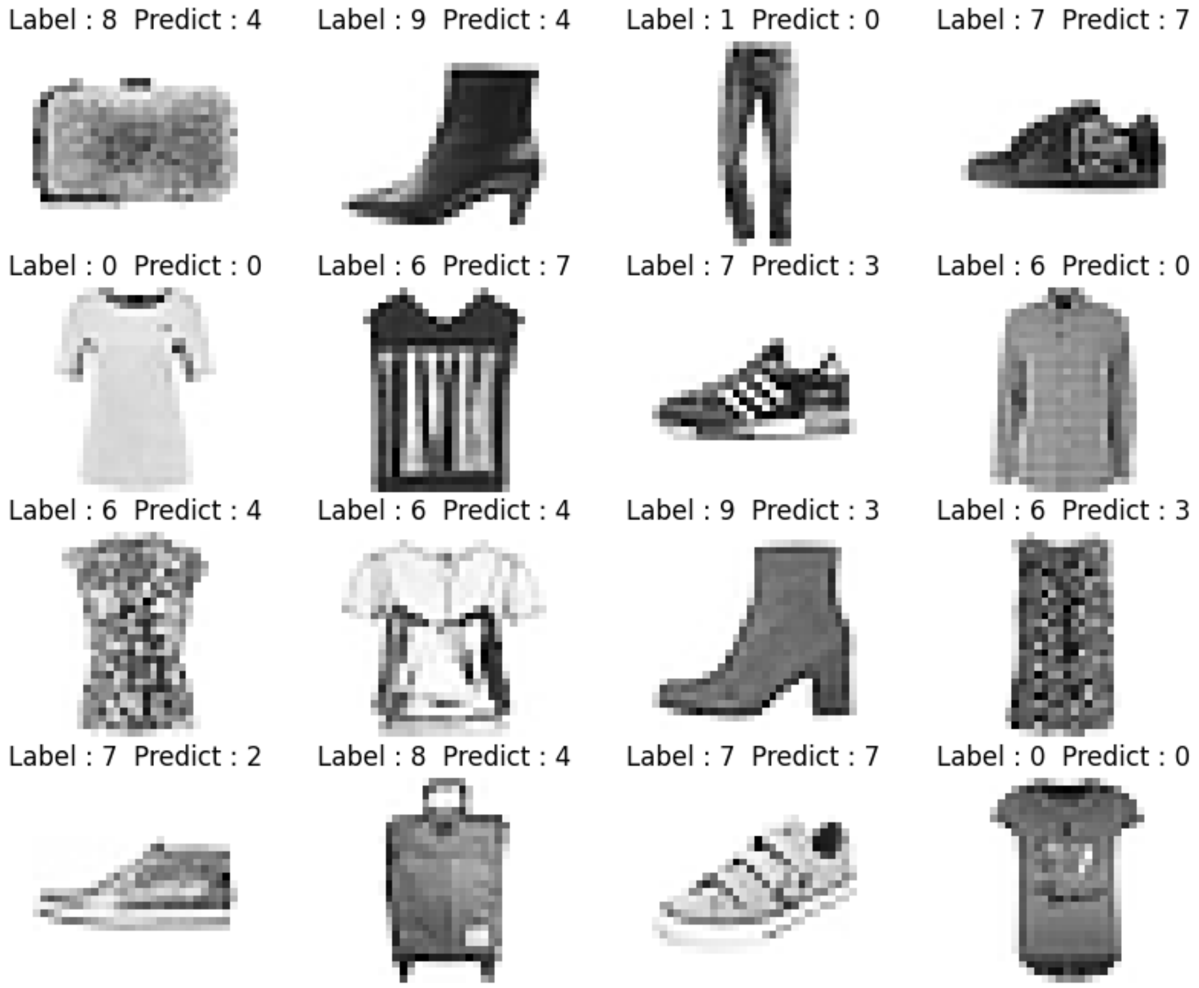
2.14 그려보자



```
plt.figure(figsize=(10, 8))

for idx, n in enumerate(samples):
    plt.subplot(4, 4, idx + 1)
    plt.imshow(X_train[n].reshape(28, 28), cmap="Greys", interpolation="nearest")
    plt.title("Label : " + str(y_train[n]) + "  Predict : " + str(predicted_labels[n]
    plt.axis("off")

plt.show()
```

✓ 0.2s Python

2.15 결과



- 0 : 티셔츠
- 1 : 바지
- 2 : 스웨터

- 3 : 드레스
- 4 : 코트
- 5 : 샌들
- 6 : 셔츠
- 7 : 운동화
- 8 : 가방
- 9 : 부츠