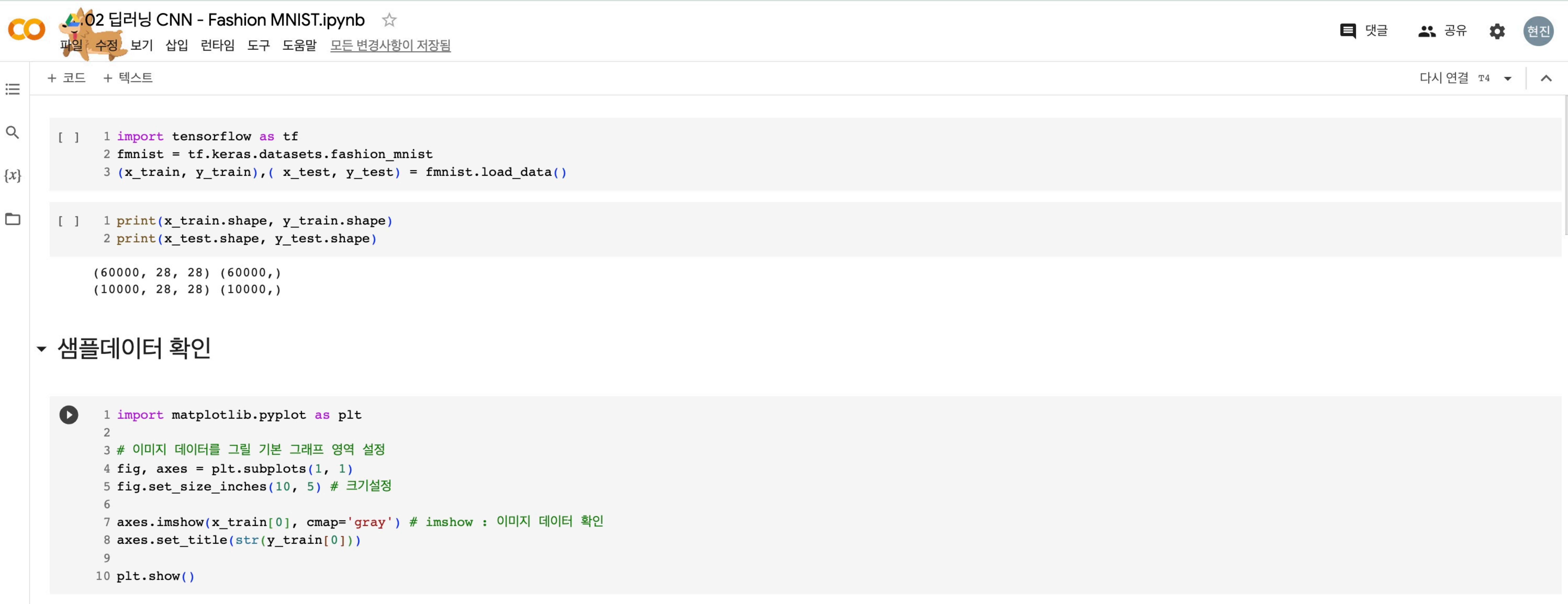


```
[ ]
import tensorflow as tf
fmnist = tf.keras.datasets.fashion_mnist
(x_train, y_train),( x_test, y_test) = fmnist.load_data()
```

```
[ ]
print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)
```

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

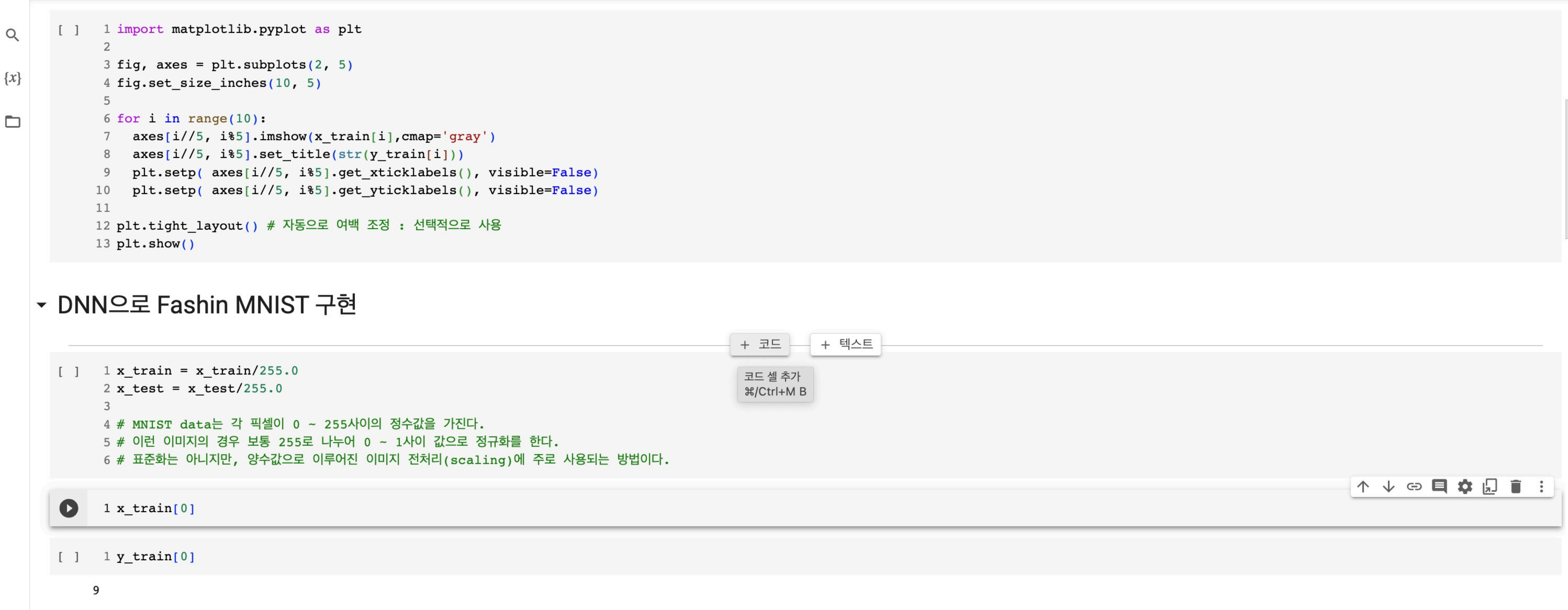
샘플데이터 확인

```
import matplotlib.pyplot as plt

# 이미지 데이터를 그릴 기본 그래프 영역 설정
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 5) # 크기설정

axes.imshow(x_train[0], cmap='gray') # imshow : 이미지 데이터 확인
axes.set_title(str(y_train[0]))

plt.show()
```



```python
import matplotlib.pyplot as plt

fig, axes = plt.subplots(2, 5)
fig.set_size_inches(10, 5)

for i in range(10):
  axes[i//5, i%5].imshow(x_train[i],cmap='gray')
  axes[i//5, i%5].set_title(str(y_train[i]))
  plt.setp( axes[i//5, i%5].get_xticklabels(), visible=False)
  plt.setp( axes[i//5, i%5].get_yticklabels(), visible=False)

plt.tight_layout() # 자동으로 여백 조정 : 선택적으로 사용
plt.show()
```

DNN으로 Fashin MNIST 구현

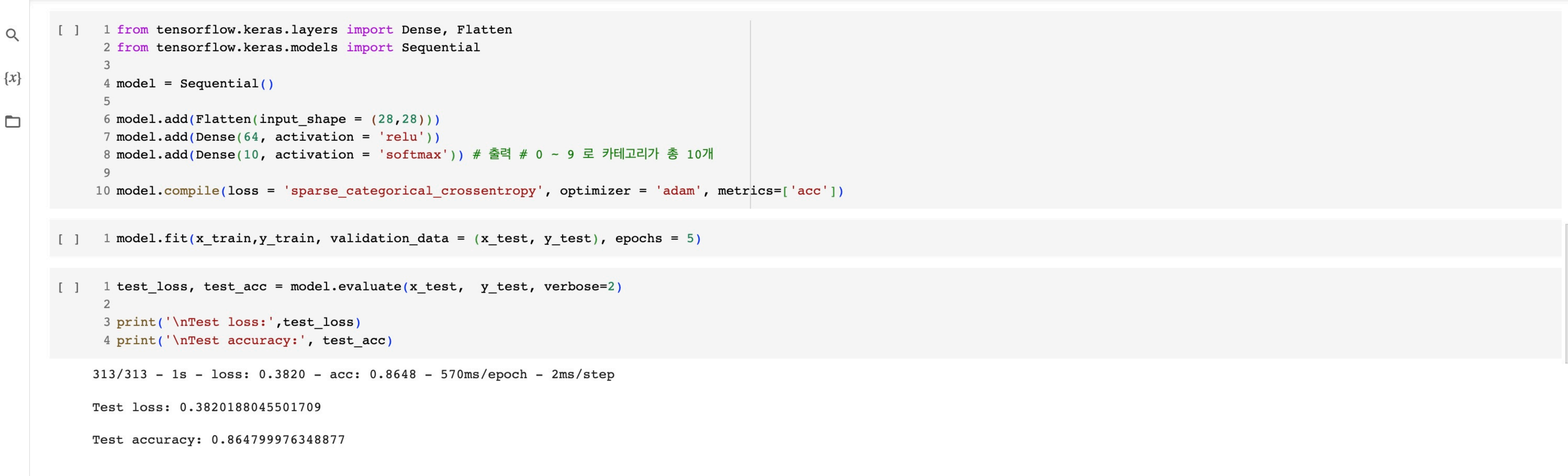
```python
x_train = x_train/255.0
x_test = x_test/255.0

# MNIST data는 각 픽셀이 0 ~ 255사이의 정수값을 가진다.
# 이런 이미지의 경우 보통 255로 나누어 0 ~ 1사이 값으로 정규화를 한다.
# 표준화는 아니지만, 양수값으로 이루어진 이미지 전처리(scaling)에 주로 사용되는 방법이다.
```

```python
x_train[0]
```

```python
y_train[0]
```

```
9
```



```python
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Sequential

model = Sequential()

model.add(Flatten(input_shape = (28,28)))
model.add(Dense(64, activation = 'relu'))
model.add(Dense(10, activation = 'softmax')) # 출력 # 0 ~ 9 로 카테고리가 총 10개

model.compile(loss = 'sparse_categorical_crossentropy', optimizer = 'adam', metrics=['acc'])
```

```python
model.fit(x_train,y_train, validation_data = (x_test, y_test), epochs = 5)
```

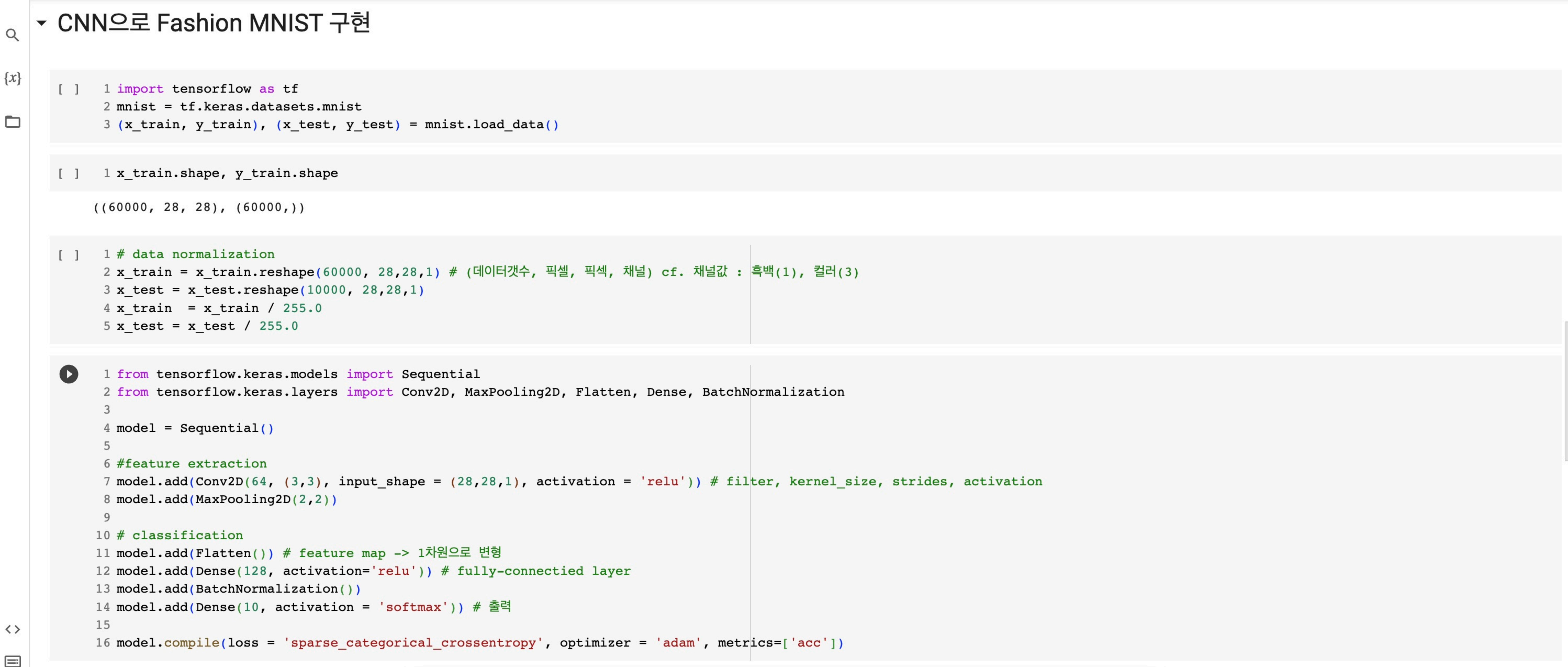
```python
test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)

print('\nTest loss:',test_loss)
print('\nTest accuracy:', test_acc)
```

```
313/313 - 1s - loss: 0.3820 - acc: 0.8648 - 570ms/epoch - 2ms/step

Test loss: 0.3820188045501709

Test accuracy: 0.864799976348877
```



CNN으로 Fashion MNIST 구현

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
x_train.shape, y_train.shape
```

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

```
# data normalization
x_train = x_train.reshape(60000, 28,28,1) # (데이터갯수, 픽셀, 픽섹, 채널) cf. 채널값 : 흑백(1), 컬러(3)
x_test = x_test.reshape(10000, 28,28,1)
x_train  = x_train / 255.0
x_test = x_test / 255.0
```

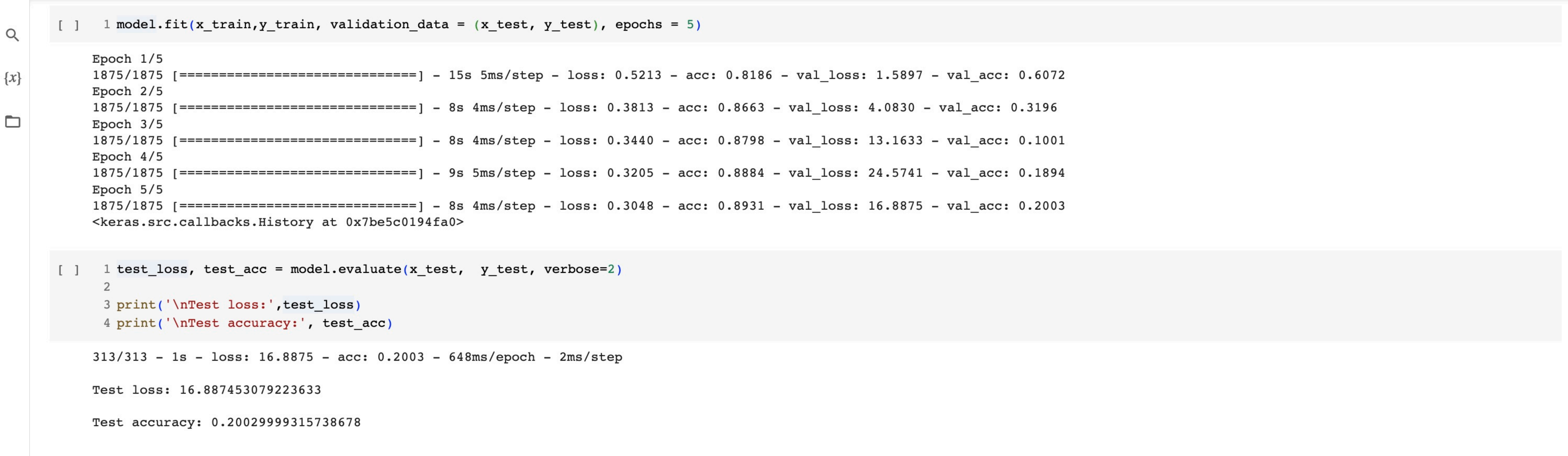
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, BatchNormalization

model = Sequential()

#feature extraction
model.add(Conv2D(64, (3,3), input_shape = (28,28,1), activation = 'relu')) # filter, kernel_size, strides, activation
model.add(MaxPooling2D(2,2))

# classification
model.add(Flatten()) # feature map -> 1차원으로 변형
model.add(Dense(128, activation='relu')) # fully-connected layer
model.add(BatchNormalization())
model.add(Dense(10, activation = 'softmax')) # 출력

model.compile(loss = 'sparse_categorical_crossentropy', optimizer = 'adam', metrics=['acc'])
```



```
model.fit(x_train,y_train, validation_data = (x_test, y_test), epochs = 5)
```

```
Epoch 1/5
1875/1875 [==============================] - 15s 5ms/step - loss: 0.5213 - acc: 0.8186 - val_loss: 1.5897 - val_acc: 0.6072
Epoch 2/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.3813 - acc: 0.8663 - val_loss: 4.0830 - val_acc: 0.3196
Epoch 3/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.3440 - acc: 0.8798 - val_loss: 13.1633 - val_acc: 0.1001
Epoch 4/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.3205 - acc: 0.8884 - val_loss: 24.5741 - val_acc: 0.1894
Epoch 5/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.3048 - acc: 0.8931 - val_loss: 16.8875 - val_acc: 0.2003
<keras.src.callbacks.History at 0x7be5c0194fa0>
```

```
test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)

print('\nTest loss:',test_loss)
print('\nTest accuracy:', test_acc)
```

```
313/313 - 1s - loss: 16.8875 - acc: 0.2003 - 648ms/epoch - 2ms/step

Test loss: 16.887453079223633

Test accuracy: 0.20029999315738678
```