parrot

Data Science

MNIST Dataset

4기 1조

발표자: 장민우

조원: 김민강, 노지연, 신용진, 장민우

멘토: 3기 이동호

Contents

0. Preprocessing & Inspecting

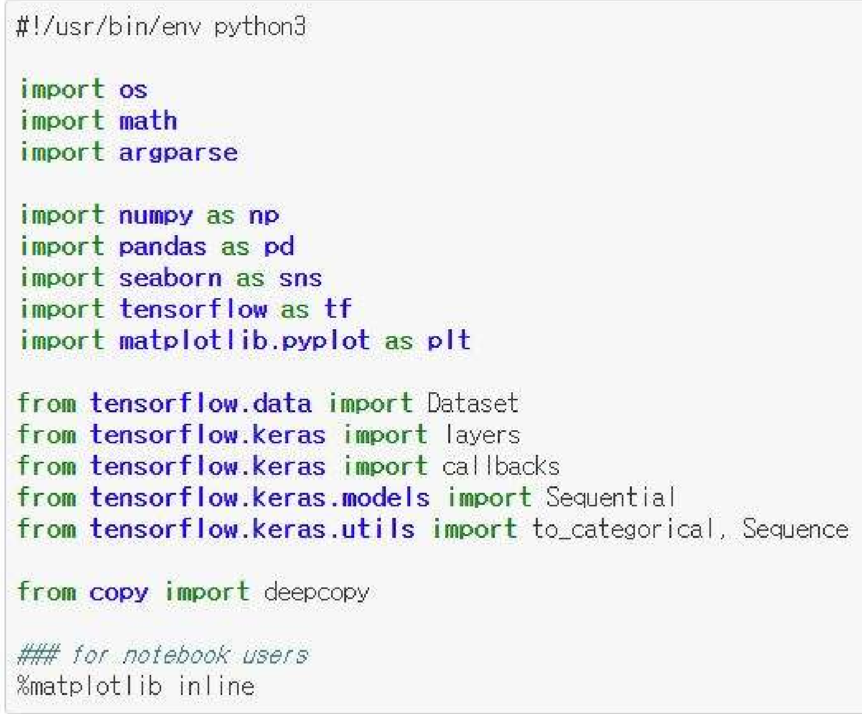
1. CNN Modeling / Experiment

2. Hyperparametrization

3. Evaluation & Test

0. Preprocessing & Inspecting

0. Preprocessing & Inspecting

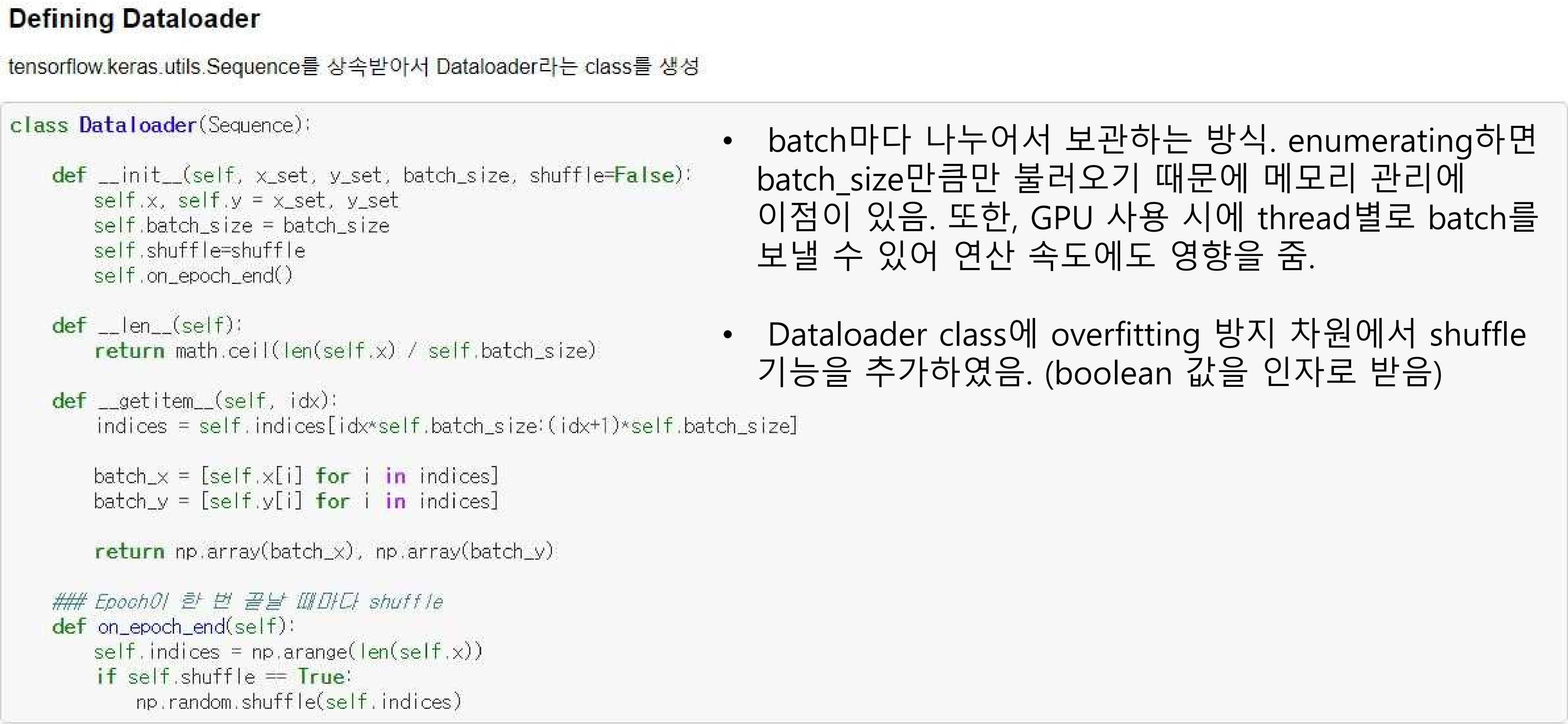


```
#!/usr/bin/env python3

import os
import math
import argparse

import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt

from tensorflow.data import Dataset
from tensorflow.keras import layers
from tensorflow.keras import callbacks
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical, Sequence

from copy import deepcopy

### for notebook users
%matplotlib inline
```

- argparse: argumentparser 관련 python 내장 package.
  hyperparametrization할 때에 사용됨.

- deepcopy: copy를 할 때에 원래의 객체와 다른 id를
  할당하는 function. 수정된 결과가 의도치
  않은 영향을 주는 것을 방지하는 역할.

0. Preprocessing & Inspecting



Defining Dataloader

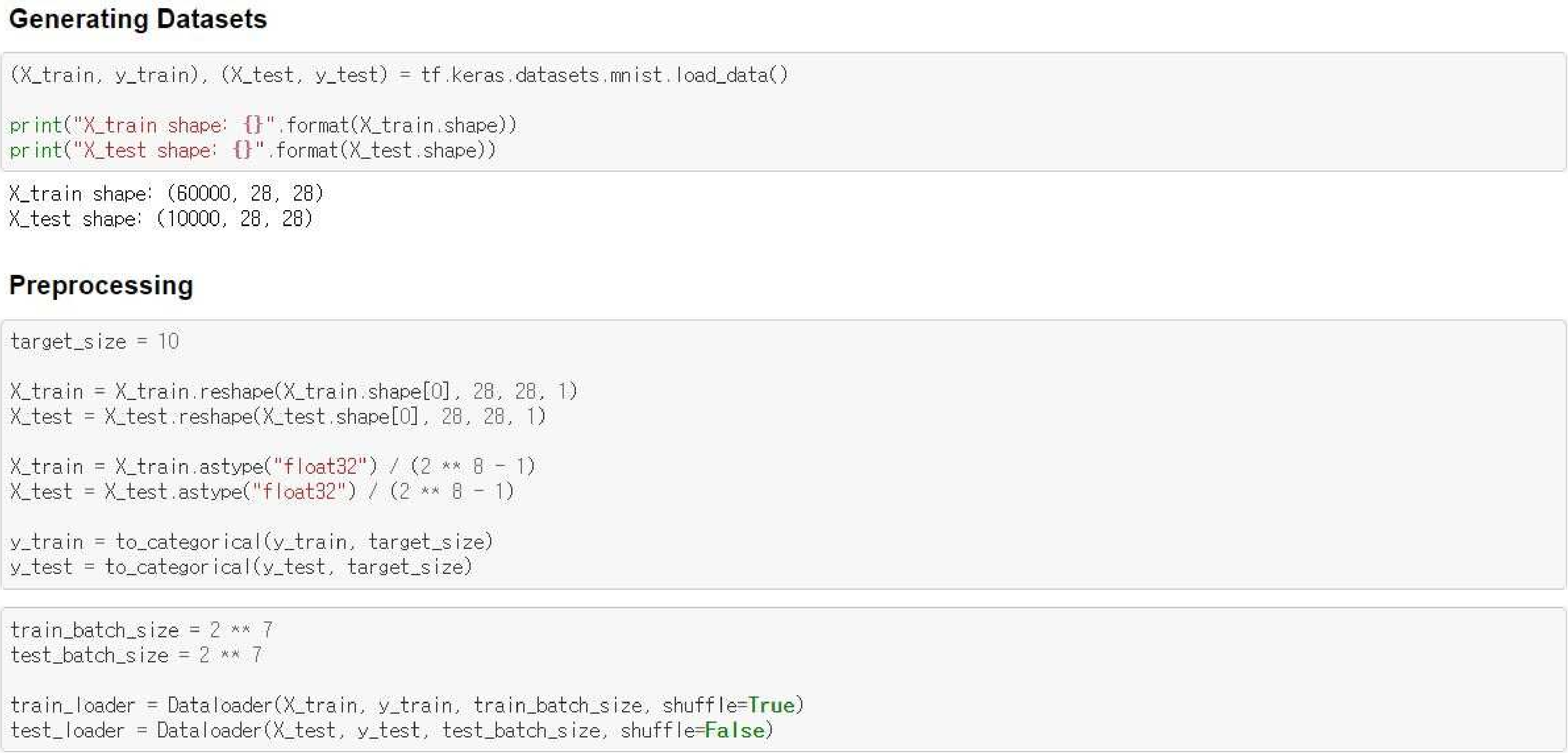
tensorflow.keras.utils.Sequence를 상속받아서 Dataloader라는 class를 생성

```
class Dataloader(Sequence):

    def __init__(self, x_set, y_set, batch_size, shuffle=False):
        self.x, self.y = x_set, y_set
        self.batch_size = batch_size
        self.shuffle=shuffle
        self.on_epoch_end()

    def __len__(self):
        return math.ceil(len(self.x) / self.batch_size)

    def __getitem__(self, idx):
        indices = self.indices[idx*self.batch_size:(idx+1)*self.batch_size]

        batch_x = [self.x[i] for i in indices]
        batch_y = [self.y[i] for i in indices]

        return np.array(batch_x), np.array(batch_y)

    ### Epoch이 한 번 끝날 때마다 shuffle
    def on_epoch_end(self):
        self.indices = np.arange(len(self.x))
        if self.shuffle == True:
            np.random.shuffle(self.indices)
```

- batch마다 나누어서 보관하는 방식. enumerating하면 batch_size만큼만 불러오기 때문에 메모리 관리에 이점이 있음. 또한, GPU 사용 시에 thread별로 batch를 보낼 수 있어 연산 속도에도 영향을 줌.

- Dataloader class에 overfitting 방지 차원에서 shuffle 기능을 추가하였음. (boolean 값을 인자로 받음)

0. Preprocessing & Inspecting



Generating Datasets

```
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()

print("X_train shape: {}".format(X_train.shape))
print("X_test shape: {}".format(X_test.shape))
```

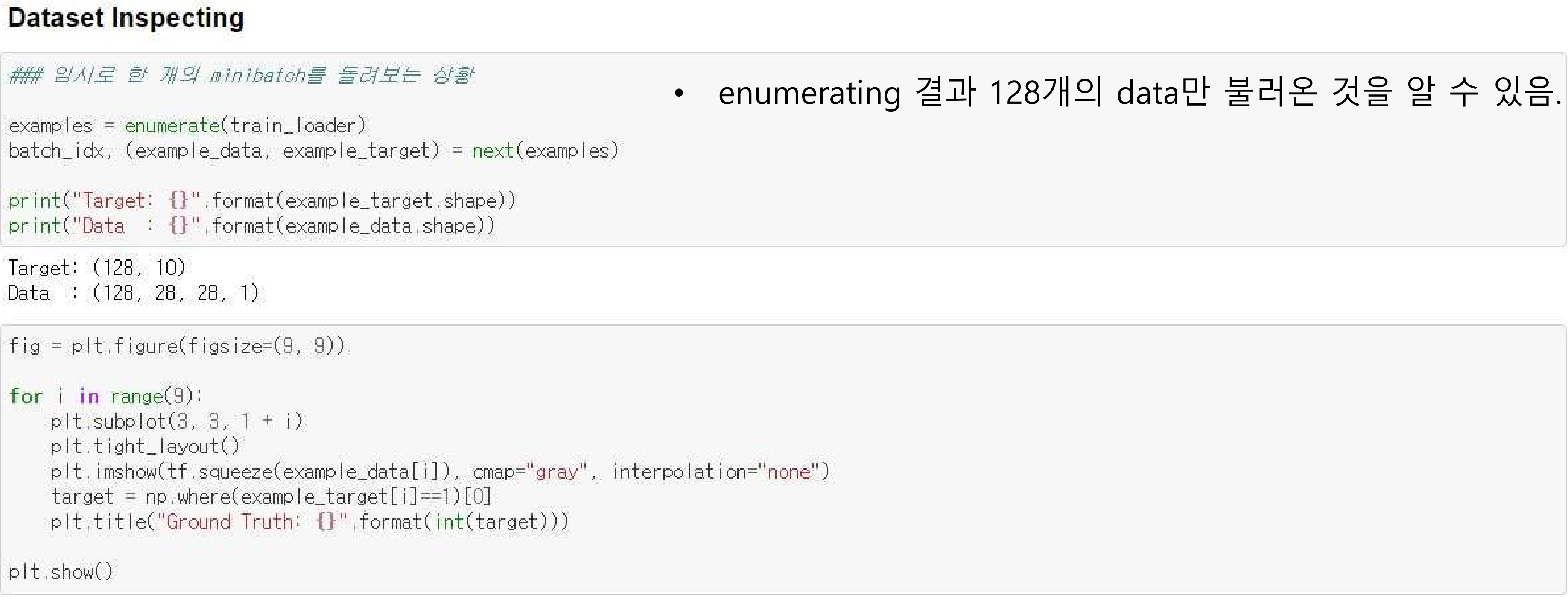
```
X_train shape: (60000, 28, 28)
X_test shape: (10000, 28, 28)
```

Preprocessing

```
target_size = 10

X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)

X_train = X_train.astype("float32") / (2 ** 8 - 1)
X_test = X_test.astype("float32") / (2 ** 8 - 1)

y_train = to_categorical(y_train, target_size)
y_test = to_categorical(y_test, target_size)
```

```
train_batch_size = 2 ** 7
test_batch_size = 2 ** 7

train_loader = Dataloader(X_train, y_train, train_batch_size, shuffle=True)
test_loader = Dataloader(X_test, y_test, test_batch_size, shuffle=False)
```

0. Preprocessing & Inspecting



Dataset Inspecting

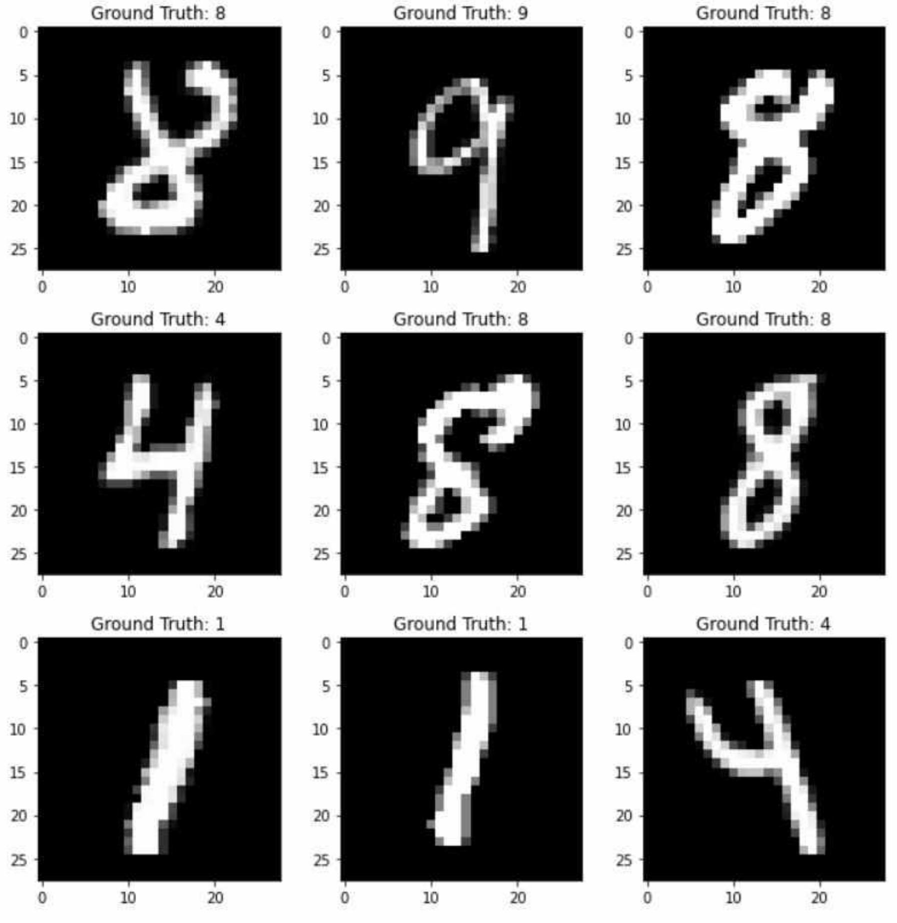
```
### 임시로 한 개의 minibatch를 돌려보는 상황

examples = enumerate(train_loader)
batch_idx, (example_data, example_target) = next(examples)

print("Target: {}".format(example_target.shape))
print("Data  : {}".format(example_data.shape))
```

```
Target: (128, 10)
Data  : (128, 28, 28, 1)
```

```
fig = plt.figure(figsize=(9, 9))

for i in range(9):
    plt.subplot(3, 3, 1 + i)
    plt.tight_layout()
    plt.imshow(tf.squeeze(example_data[i]), cmap="gray", interpolation="none")
    target = np.where(example_target[i]==1)[0]
    plt.title("Ground Truth: {}".format(int(target)))

plt.show()
```

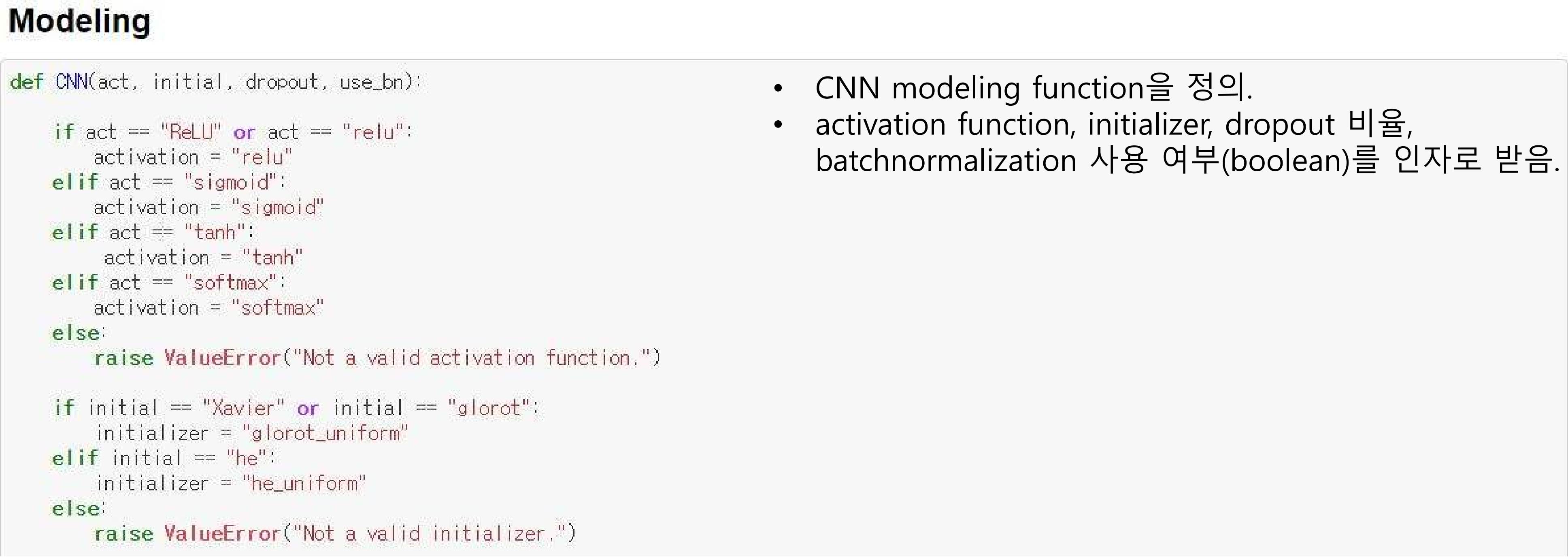
- enumerating 결과 128개의 data만 불러온 것을 알 수 있음.

0. Preprocessing & Inspecting



1. CNN Modeling / Experiment

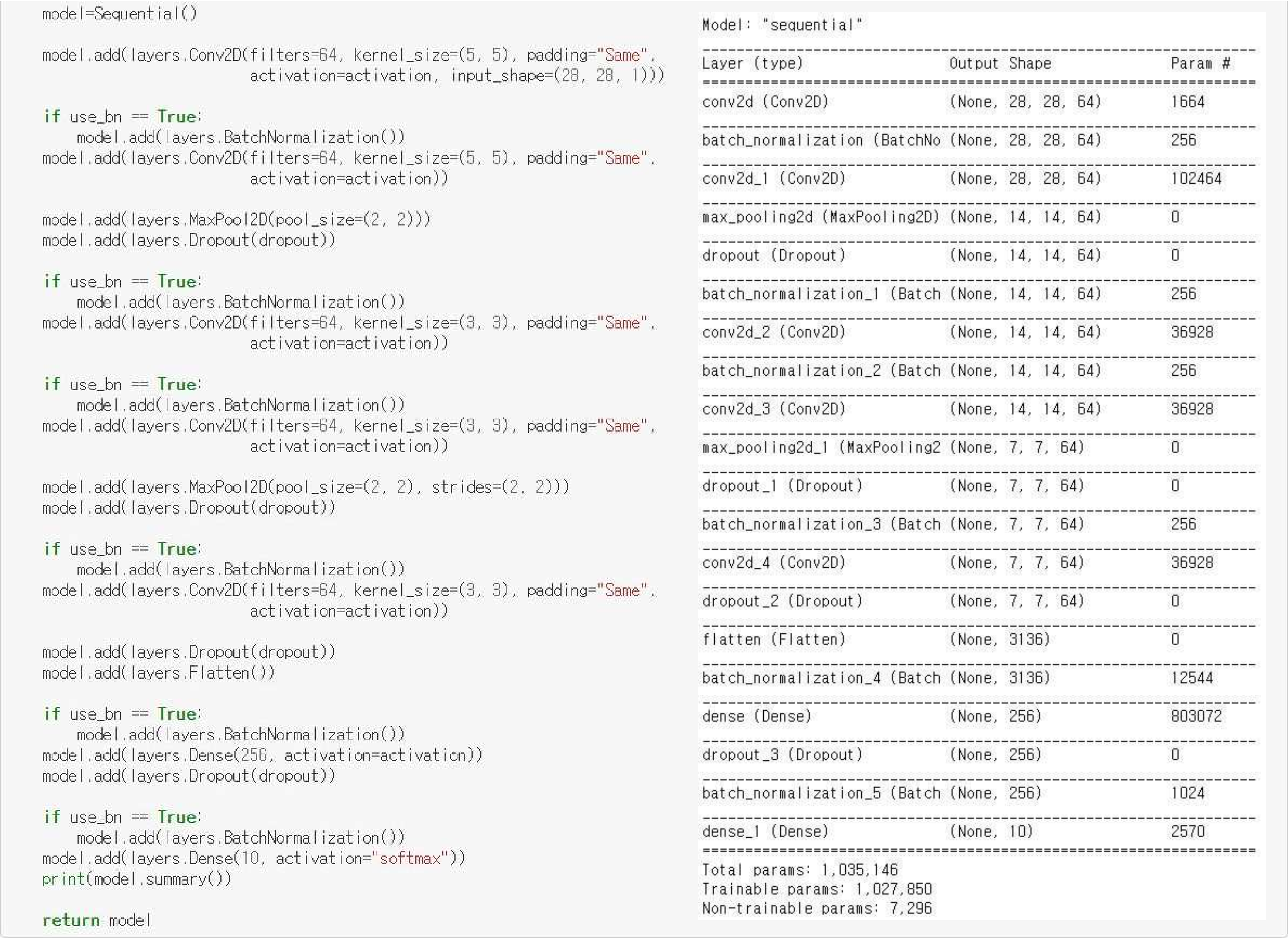
1. CNN Modeling / Experiment



Modeling

```
def CNN(act, initial, dropout, use_bn):

    if act == "ReLU" or act == "relu":
        activation = "relu"
    elif act == "sigmoid":
        activation = "sigmoid"
    elif act == "tanh":
         activation = "tanh"
    elif act == "softmax":
        activation = "softmax"
    else:
        raise ValueError("Not a valid activation function.")

    if initial == "Xavier" or initial == "glorot":
        initializer = "glorot_uniform"
    elif initial == "he":
        initializer = "he_uniform"
    else:
        raise ValueError("Not a valid initializer.")
```

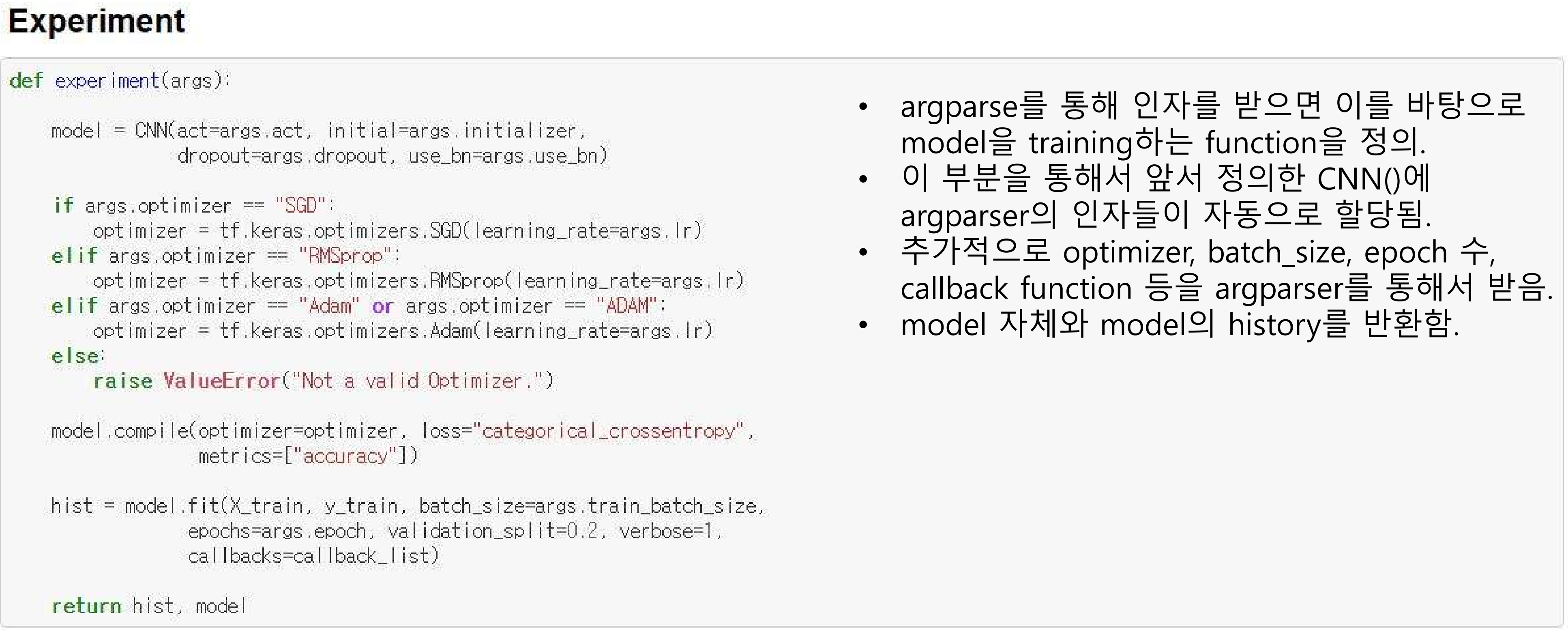
- CNN modeling function을 정의.
- activation function, initializer, dropout 비율, batchnormalization 사용 여부(boolean)를 인자로 받음.



```
model=Sequential()

model.add(layers.Conv2D(filters=64, kernel_size=(5, 5), padding="Same",
                        activation=activation, input_shape=(28, 28, 1)))

if use_bn == True:
    model.add(layers.BatchNormalization())
model.add(layers.Conv2D(filters=64, kernel_size=(5, 5), padding="Same",
                        activation=activation))

model.add(layers.MaxPool2D(pool_size=(2, 2)))
model.add(layers.Dropout(dropout))

if use_bn == True:
    model.add(layers.BatchNormalization())
model.add(layers.Conv2D(filters=64, kernel_size=(3, 3), padding="Same",
                        activation=activation))

if use_bn == True:
    model.add(layers.BatchNormalization())
model.add(layers.Conv2D(filters=64, kernel_size=(3, 3), padding="Same",
                        activation=activation))

model.add(layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2)))
model.add(layers.Dropout(dropout))

if use_bn == True:
    model.add(layers.BatchNormalization())
model.add(layers.Conv2D(filters=64, kernel_size=(3, 3), padding="Same",
                        activation=activation))

model.add(layers.Dropout(dropout))
model.add(layers.Flatten())

if use_bn == True:
    model.add(layers.BatchNormalization())
model.add(layers.Dense(256, activation=activation))
model.add(layers.Dropout(dropout))

if use_bn == True:
    model.add(layers.BatchNormalization())
model.add(layers.Dense(10, activation="softmax"))
print(model.summary())

return model
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 64)        1664
_________________________________________________________________
batch_normalization (BatchNo (None, 28, 28, 64)        256
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 64)        102464
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 14, 14, 64)        256
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 64)        36928
_________________________________________________________________
batch_normalization_2 (Batch (None, 14, 14, 64)        256
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
batch_normalization_3 (Batch (None, 7, 7, 64)          256
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 7, 7, 64)          36928
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
batch_normalization_4 (Batch (None, 3136)              12544
_________________________________________________________________
dense (Dense)                (None, 256)               803072
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
batch_normalization_5 (Batch (None, 256)               1024
_________________________________________________________________
dense_1 (Dense)              (None, 10)                2570
=================================================================
Total params: 1,035,146
Trainable params: 1,027,850
Non-trainable params: 7,296
```

1. CNN Modeling / Experiment



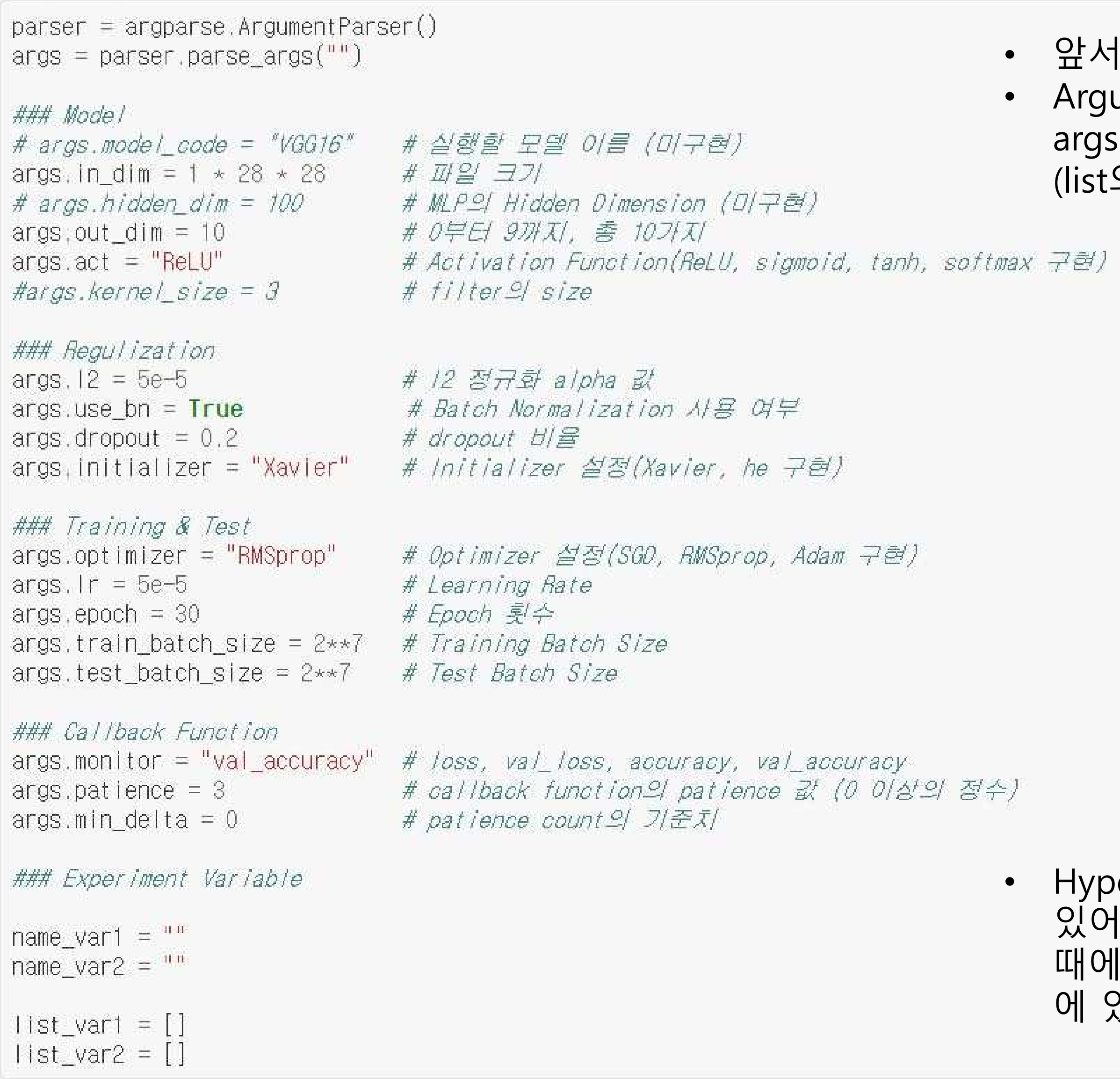
Experiment

```
def experiment(args):

    model = CNN(act=args.act, initial=args.initializer,
                dropout=args.dropout, use_bn=args.use_bn)

    if args.optimizer == "SGD":
        optimizer = tf.keras.optimizers.SGD(learning_rate=args.lr)
    elif args.optimizer == "RMSprop":
        optimizer = tf.keras.optimizers.RMSprop(learning_rate=args.lr)
    elif args.optimizer == "Adam" or args.optimizer == "ADAM":
        optimizer = tf.keras.optimizers.Adam(learning_rate=args.lr)
    else:
        raise ValueError("Not a valid Optimizer.")

    model.compile(optimizer=optimizer, loss="categorical_crossentropy",
                  metrics=["accuracy"])

    hist = model.fit(X_train, y_train, batch_size=args.train_batch_size,
                     epochs=args.epoch, validation_split=0.2, verbose=1,
                     callbacks=callback_list)

    return hist, model
```

- argparse를 통해 인자를 받으면 이를 바탕으로 model을 training하는 function을 정의.
- 이 부분을 통해서 앞서 정의한 CNN()에 argparser의 인자들이 자동으로 할당됨.
- 추가적으로 optimizer, batch_size, epoch 수, callback function 등을 argparser를 통해서 받음.
- model 자체와 model의 history를 반환함.

2. Hyperparametrization

Hyperparametrization

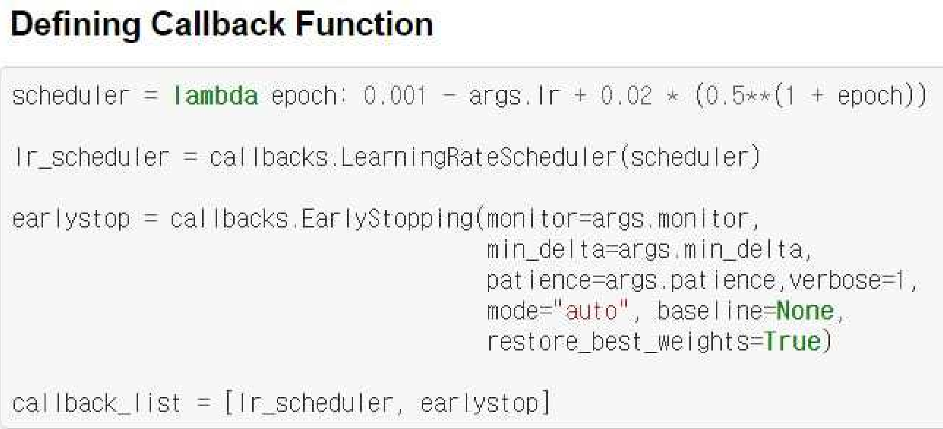


```
parser = argparse.ArgumentParser()
args = parser.parse_args("")

### Model
# args.model_code = "VGG16"   # 실행할 모델 이름 (미구현)
args.in_dim = 1 * 28 * 28     # 파일 크기
# args.hidden_dim = 100       # MLP의 Hidden Dimension (미구현)
args.out_dim = 10             # 0부터 9까지, 총 10가지
args.act = "ReLU"             # Activation Function(ReLU, sigmoid, tanh, softmax 구현)
#args.kernel_size = 3         # filter의 size

### Regulization
args.l2 = 5e-5                # l2 정규화 alpha 값
args.use_bn = True             # Batch Normalization 사용 여부
args.dropout = 0.2            # dropout 비율
args.initializer = "Xavier"   # Initializer 설정(Xavier, he 구현)

### Training & Test
args.optimizer = "RMSprop"    # Optimizer 설정(SGD, RMSprop, Adam 구현)
args.lr = 5e-5                # Learning Rate
args.epoch = 30               # Epoch 횟수
args.train_batch_size = 2**7  # Training Batch Size
args.test_batch_size = 2**7   # Test Batch Size

### Callback Function
args.monitor = "val_accuracy"  # loss, val_loss, accuracy, val_accuracy
args.patience = 3             # callback function의 patience 값 (0 이상의 정수)
args.min_delta = 0            # patience count의 기준치

### Experiment Variable

name_var1 = ""
name_var2 = ""

list_var1 = []
list_var2 = []
```

- 앞서 언급한 argparse가 여기에서 사용됨.
- ArgumentParser의 이름을 args라고 저장하면, args라는 namespace 안에 변수들이 담기는 형태. (list의 .append()와 유사하나, 그 깊이가 다름)

- Hyperparameter tuning은 결과를 도출하는 데에 있어 필수적인 과정이나, 최종적으로 reporting할 때에는 빠지게 되는 부분. 따라서 현대 제일 아래에 있는 Experiment Variable은 비워져 있는 상태.

2. Hyperparametrization

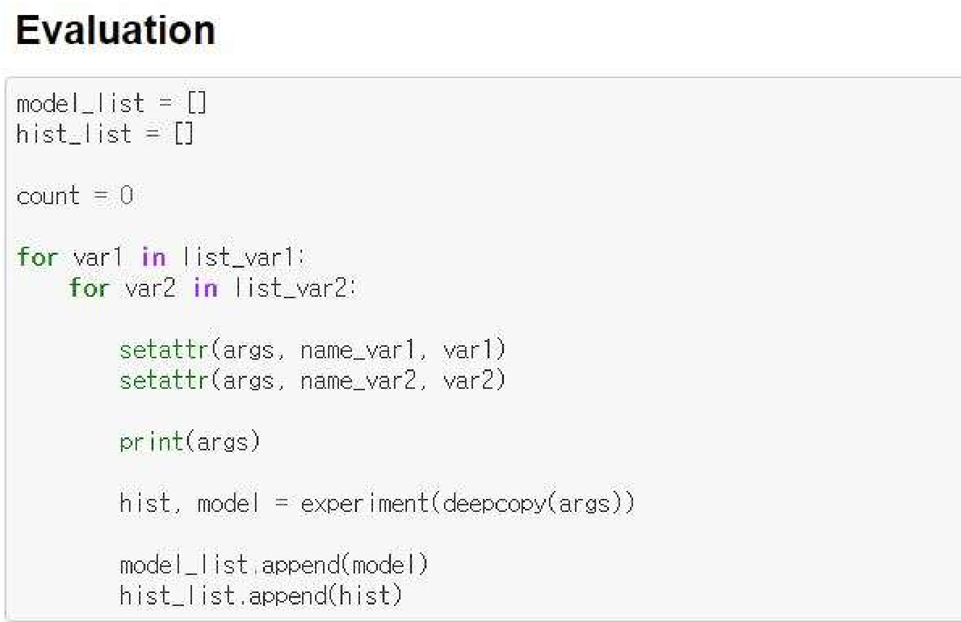


Defining Callback Function

```
scheduler = lambda epoch: 0.001 - args.lr + 0.02 * (0.5**(1 + epoch))

lr_scheduler = callbacks.LearningRateScheduler(scheduler)

earlystop = callbacks.EarlyStopping(monitor=args.monitor,
                                    min_delta=args.min_delta,
                                    patience=args.patience,verbose=1,
                                    mode="auto", baseline=None,
                                    restore_best_weights=True)

callback_list = [lr_scheduler, earlystop]
```

- callback function으로 LearningRateScheduler와 EarlyStopping을 사용하기로 결정, list로 묶어서 저장.
- monitor와 patience, 그리고 min_deltat를 변수화하여 args에 추가하였음.

2. Hyperparametrization



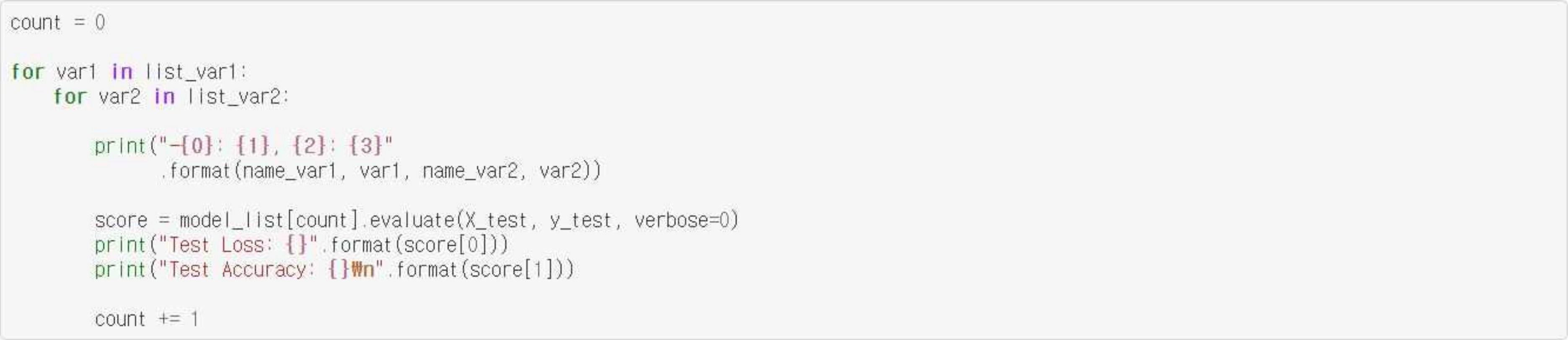
Evaluation

```
model_list = []
hist_list = []

count = 0

for var1 in list_var1:
    for var2 in list_var2:

        setattr(args, name_var1, var1)
        setattr(args, name_var2, var2)

        print(args)

        hist, model = experiment(deepcopy(args))

        model_list.append(model)
        hist_list.append(hist)
```

- 위에서 본 빈 list들에 관찰 대상을 입력하면, 한 번에 여러 번의 train을 한 후, model과 train 결과를 각각 다른 list에 저장.

2. Hyperparametrization

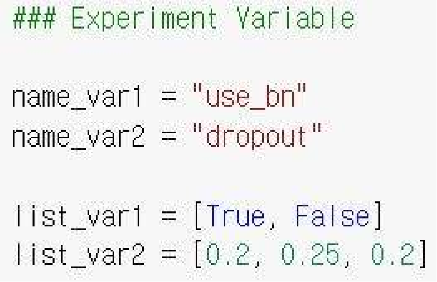
Test



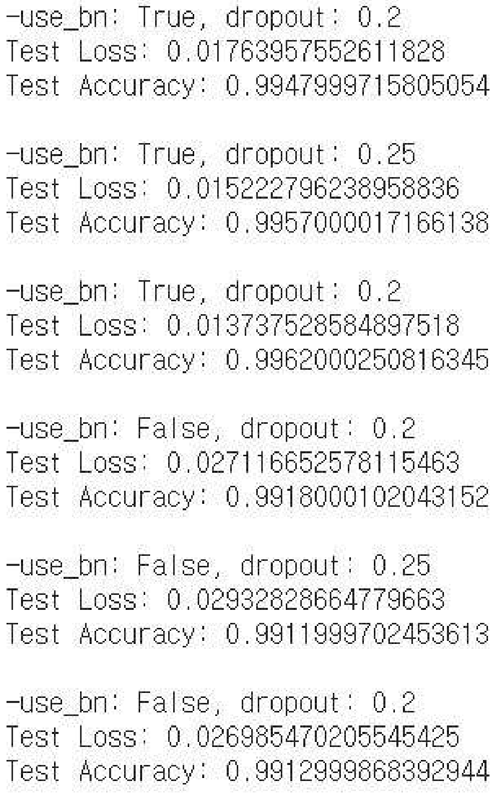
```
count = 0

for var1 in list_var1:
    for var2 in list_var2:

        print("-{0}: {1}, {2}: {3}"
              .format(name_var1, var1, name_var2, var2))

        score = model_list[count].evaluate(X_test, y_test, verbose=0)
        print("Test Loss: {}".format(score[0]))
        print("Test Accuracy: {}₩n".format(score[1]))

        count += 1
```

- 예시

2. Hyperparametrization



```
### Experiment Variable

name_var1 = "use_bn"
name_var2 = "dropout"

list_var1 = [True, False]
list_var2 = [0.2, 0.25, 0.2]
```

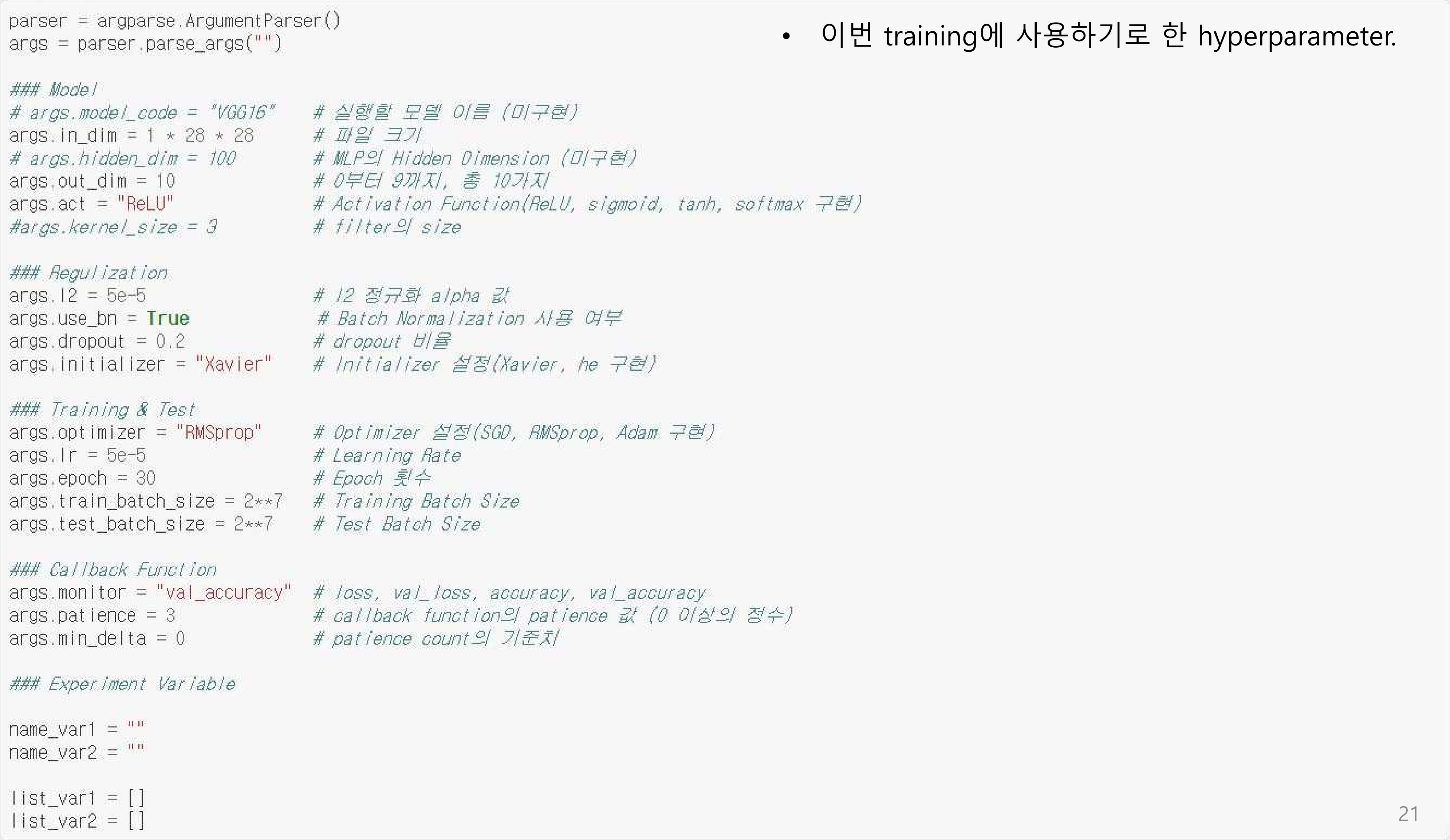


```
-use_bn: True, dropout: 0.2
Test Loss: 0.01763957552611828
Test Accuracy: 0.9947999715805054

-use_bn: True, dropout: 0.25
Test Loss: 0.015222796238958836
Test Accuracy: 0.9957000017166138

-use_bn: True, dropout: 0.2
Test Loss: 0.013737528584897518
Test Accuracy: 0.9962000250816345

-use_bn: False, dropout: 0.2
Test Loss: 0.027116652578115463
Test Accuracy: 0.9918000102043152

-use_bn: False, dropout: 0.25
Test Loss: 0.02932828664779663
Test Accuracy: 0.9911999702453613

-use_bn: False, dropout: 0.2
Test Loss: 0.026985470205545425
Test Accuracy: 0.9912999868392944
```

- list 안의 hyperparameter들마다 matching한 후, 각각의 결과를 반환(중복되어도 상관 없음).
- 가장 좋은 결과를 얻은 hyperparameter를 채택, 다른 hyperparameter들도 동일한 방식으로 여러 번 진행하면서 tuning을 마무리.

3. Evaluation & Test

Hyperparametrization

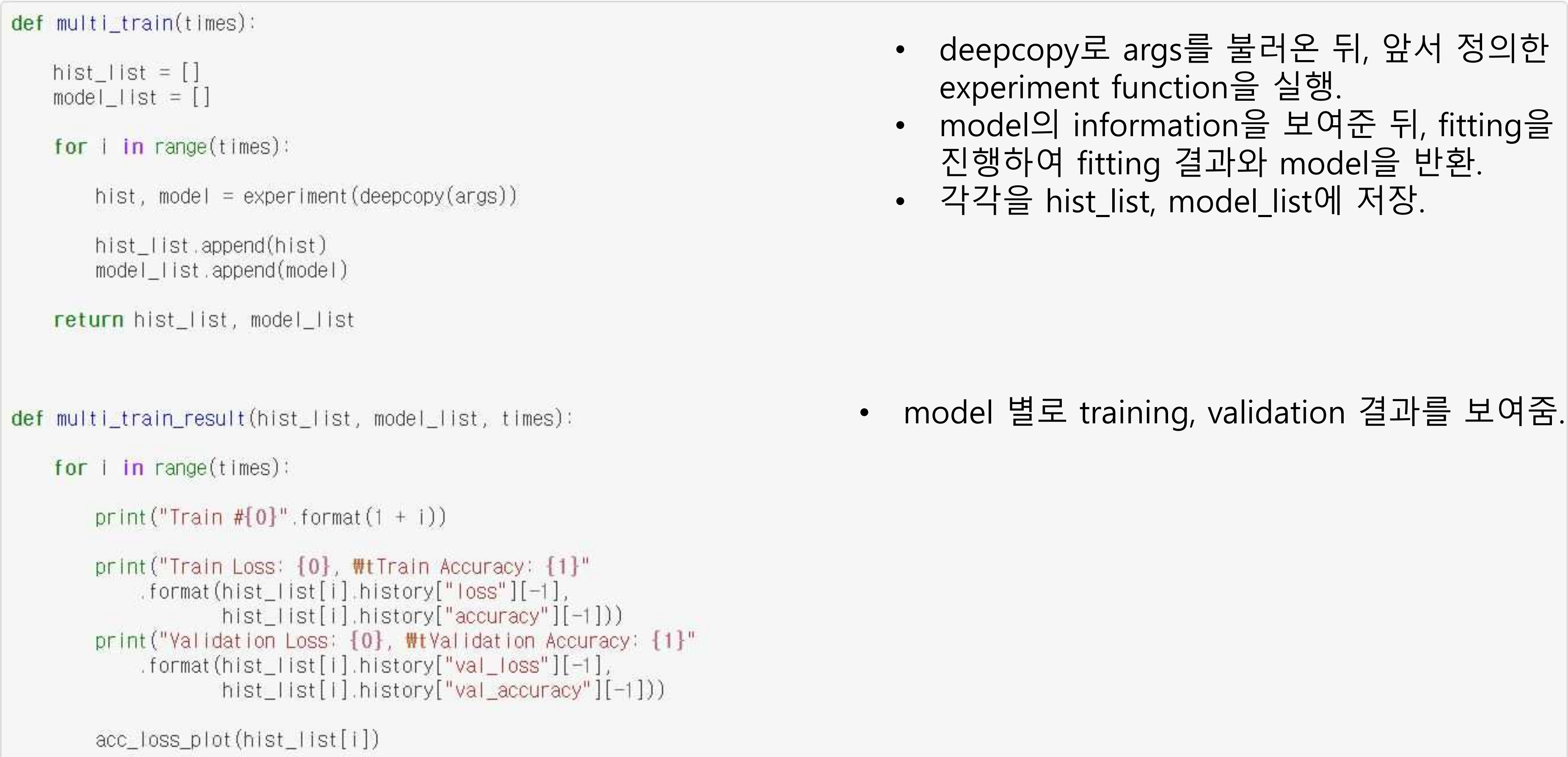


```
parser = argparse.ArgumentParser()
args = parser.parse_args("")

### Model
# args.model_code = "VGG16"    # 실행할 모델 이름 (미구현)
args.in_dim = 1 * 28 * 28      # 파일 크기
# args.hidden_dim = 100        # MLP의 Hidden Dimension (미구현)
args.out_dim = 10              # 0부터 9까지, 총 10가지
args.act = "ReLU"              # Activation Function(ReLU, sigmoid, tanh, softmax 구현)
#args.kernel_size = 3          # filter의 size

### Regulization
args.l2 = 5e-5                 # l2 정규화 alpha 값
args.use_bn = True              # Batch Normalization 사용 여부
args.dropout = 0.2             # dropout 비율
args.initializer = "Xavier"    # Initializer 설정(Xavier, he 구현)

### Training & Test
args.optimizer = "RMSprop"     # Optimizer 설정(SGD, RMSprop, Adam 구현)
args.lr = 5e-5                 # Learning Rate
args.epoch = 30                # Epoch 횟수
args.train_batch_size = 2**7   # Training Batch Size
args.test_batch_size = 2**7    # Test Batch Size

### Callback Function
args.monitor = "val_accuracy"  # loss, val_loss, accuracy, val_accuracy
args.patience = 3              # callback function의 patience 값 (0 이상의 정수)
args.min_delta = 0             # patience count의 기준치

### Experiment Variable

name_var1 = ""
name_var2 = ""

list_var1 = []
list_var2 = []
```

- 이번 training에 사용하기로 한 hyperparameter.

3. Evaluation & Test

FINAL RESULT

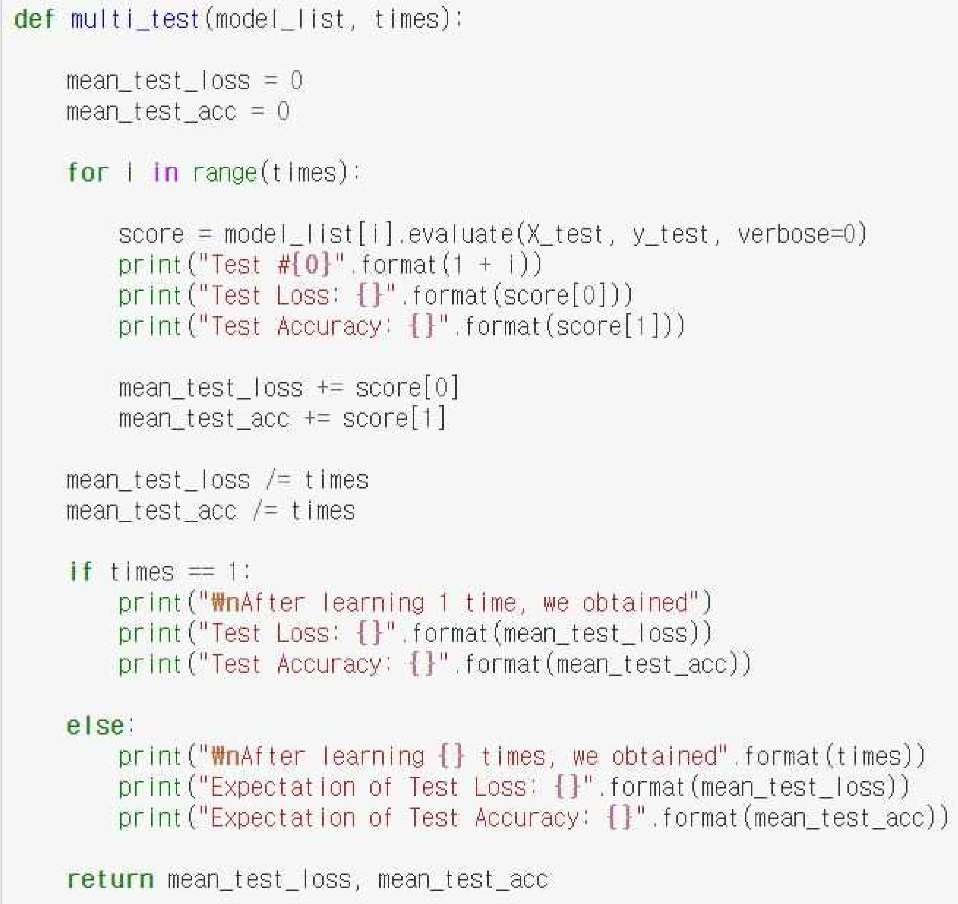


```
def multi_train(times):

    hist_list = []
    model_list = []

    for i in range(times):

        hist, model = experiment(deepcopy(args))

        hist_list.append(hist)
        model_list.append(model)

    return hist_list, model_list


def multi_train_result(hist_list, model_list, times):

    for i in range(times):

        print("Train #{0}".format(1 + i))

        print("Train Loss: {0}, ₩tTrain Accuracy: {1}"
            .format(hist_list[i].history["loss"][-1],
                    hist_list[i].history["accuracy"][-1]))
        print("Validation Loss: {0}, ₩tValidation Accuracy: {1}"
            .format(hist_list[i].history["val_loss"][-1],
                    hist_list[i].history["val_accuracy"][-1]))

        acc_loss_plot(hist_list[i])
```

- deepcopy로 args를 불러온 뒤, 앞서 정의한 experiment function을 실행.
- model의 information을 보여준 뒤, fitting을 진행하여 fitting 결과와 model을 반환.
- 각각을 hist_list, model_list에 저장.

- model 별로 training, validation 결과를 보여줌.

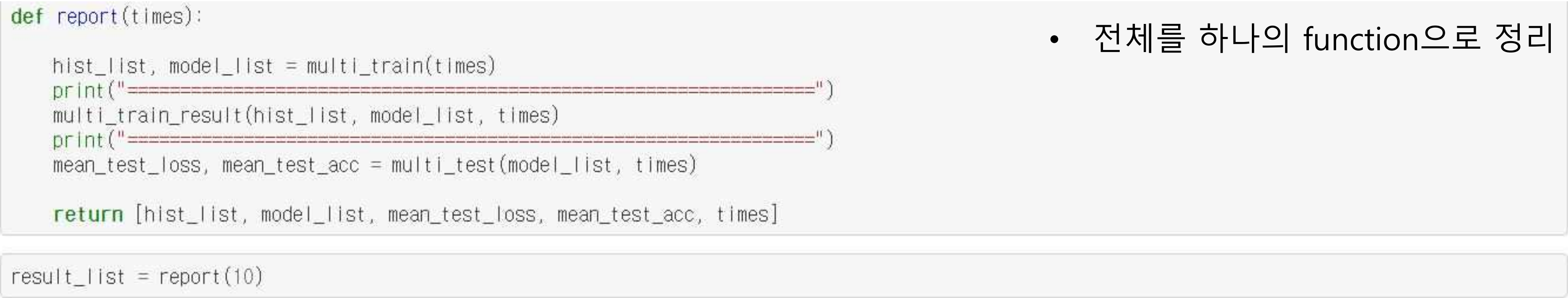
3. Evaluation & Test



```
def multi_test(model_list, times):

    mean_test_loss = 0
    mean_test_acc = 0

    for i in range(times):

        score = model_list[i].evaluate(X_test, y_test, verbose=0)
        print("Test #{0}".format(1 + i))
        print("Test Loss: {}".format(score[0]))
        print("Test Accuracy: {}".format(score[1]))

        mean_test_loss += score[0]
        mean_test_acc += score[1]

    mean_test_loss /= times
    mean_test_acc /= times

    if times == 1:
        print("₩nAfter learning 1 time, we obtained")
        print("Test Loss: {}".format(mean_test_loss))
        print("Test Accuracy: {}".format(mean_test_acc))

    else:
        print("₩nAfter learning {} times, we obtained".format(times))
        print("Expectation of Test Loss: {}".format(mean_test_loss))
        print("Expectation of Test Accuracy: {}".format(mean_test_acc))

    return mean_test_loss, mean_test_acc
```

- “times”만큼 실행 후, 각 모델 별 test_loss, test_acc를 구하여, 평균 test_loss, 평균 test_acc를 도출

3. Evaluation & Test

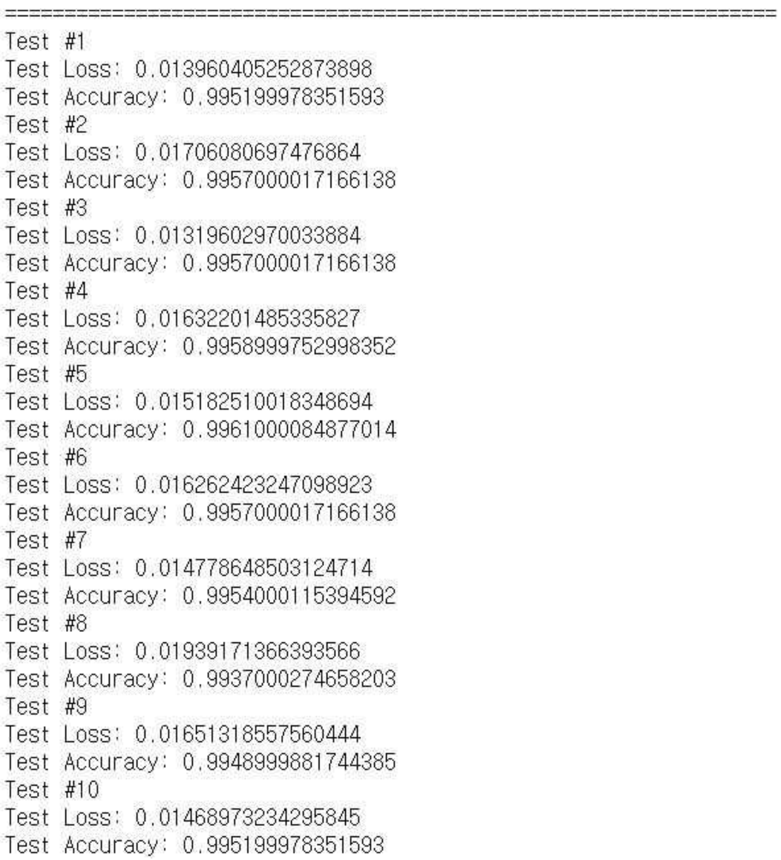


```
def report(times):

    hist_list, model_list = multi_train(times)
    print("=================================================================")
    multi_train_result(hist_list, model_list, times)
    print("=================================================================")
    mean_test_loss, mean_test_acc = multi_test(model_list, times)

    return [hist_list, model_list, mean_test_loss, mean_test_acc, times]
```

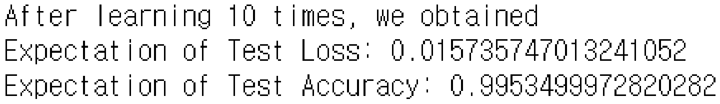
```
result_list = report(10)
```

- 전체를 하나의 function으로 정리

3. Evaluation & Test



```
===================================================================
Test #1
Test Loss: 0.013960405252873898
Test Accuracy: 0.9951999978351593
Test #2
Test Loss: 0.01706080697476864
Test Accuracy: 0.9957000017166138
Test #3
Test Loss: 0.01319602970033884
Test Accuracy: 0.9957000017166138
Test #4
Test Loss: 0.01632201485335827
Test Accuracy: 0.9958999752998352
Test #5
Test Loss: 0.015182510018348694
Test Accuracy: 0.9961000084877014
Test #6
Test Loss: 0.016262423247098923
Test Accuracy: 0.9957000017166138
Test #7
Test Loss: 0.014778648503124714
Test Accuracy: 0.9954000115394592
Test #8
Test Loss: 0.01939171366393566
Test Accuracy: 0.9937000274658203
Test #9
Test Loss: 0.01651318557560444
Test Accuracy: 0.9948999881744385
Test #10
Test Loss: 0.01468973234295845
Test Accuracy: 0.9951999978351593
```

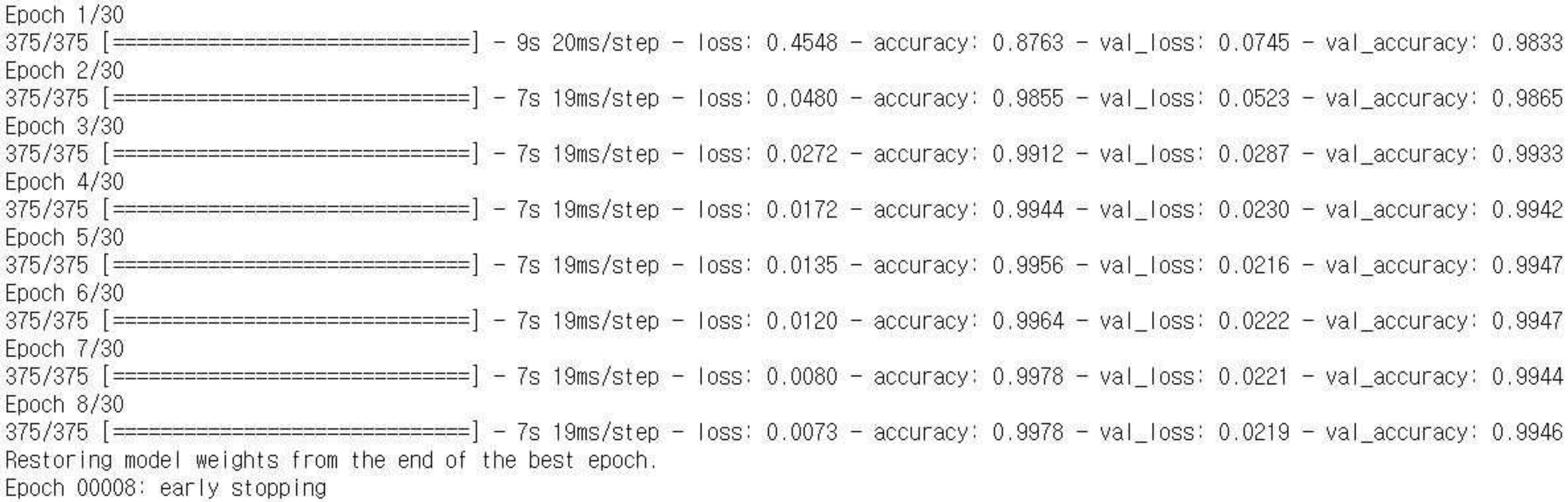


```
After learning 10 times, we obtained
Expectation of Test Loss: 0.015735747013241052
Expectation of Test Accuracy: 0.9953499972820282
```

- 평균적으로 99.535%의 정확도를 보임.
- 자세한 코드는 colab / github에서…

- 예시 (Model 5)

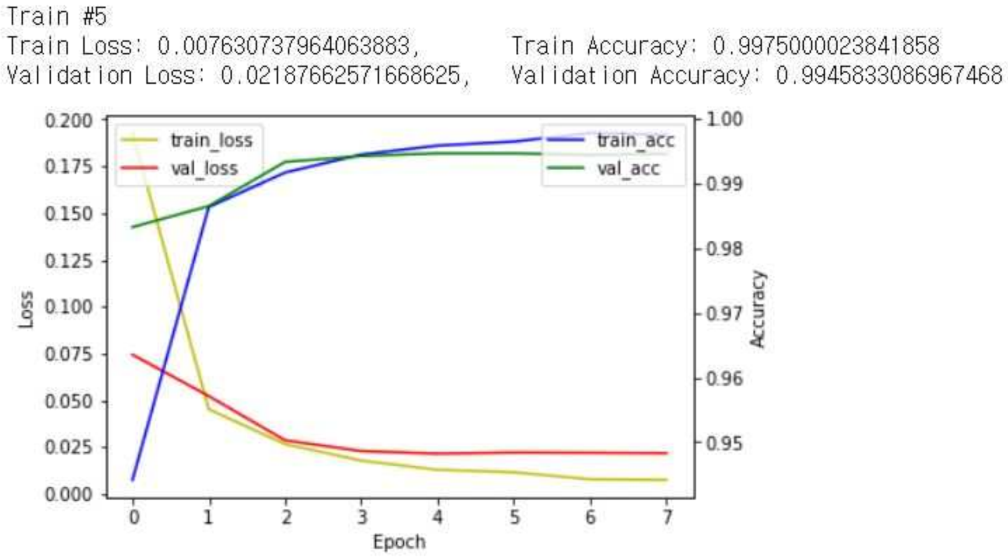
3. Evaluation & Test



```
Epoch 1/30
375/375 [==============================] - 9s 20ms/step - loss: 0.4548 - accuracy: 0.8763 - val_loss: 0.0745 - val_accuracy: 0.9833
Epoch 2/30
375/375 [==============================] - 7s 19ms/step - loss: 0.0480 - accuracy: 0.9855 - val_loss: 0.0523 - val_accuracy: 0.9865
Epoch 3/30
375/375 [==============================] - 7s 19ms/step - loss: 0.0272 - accuracy: 0.9912 - val_loss: 0.0287 - val_accuracy: 0.9933
Epoch 4/30
375/375 [==============================] - 7s 19ms/step - loss: 0.0172 - accuracy: 0.9944 - val_loss: 0.0230 - val_accuracy: 0.9942
Epoch 5/30
375/375 [==============================] - 7s 19ms/step - loss: 0.0135 - accuracy: 0.9956 - val_loss: 0.0216 - val_accuracy: 0.9947
Epoch 6/30
375/375 [==============================] - 7s 19ms/step - loss: 0.0120 - accuracy: 0.9964 - val_loss: 0.0222 - val_accuracy: 0.9947
Epoch 7/30
375/375 [==============================] - 7s 19ms/step - loss: 0.0080 - accuracy: 0.9978 - val_loss: 0.0221 - val_accuracy: 0.9944
Epoch 8/30
375/375 [==============================] - 7s 19ms/step - loss: 0.0073 - accuracy: 0.9978 - val_loss: 0.0219 - val_accuracy: 0.9946
Restoring model weights from the end of the best epoch.
Epoch 00008: early stopping
```

3. Evaluation & Test

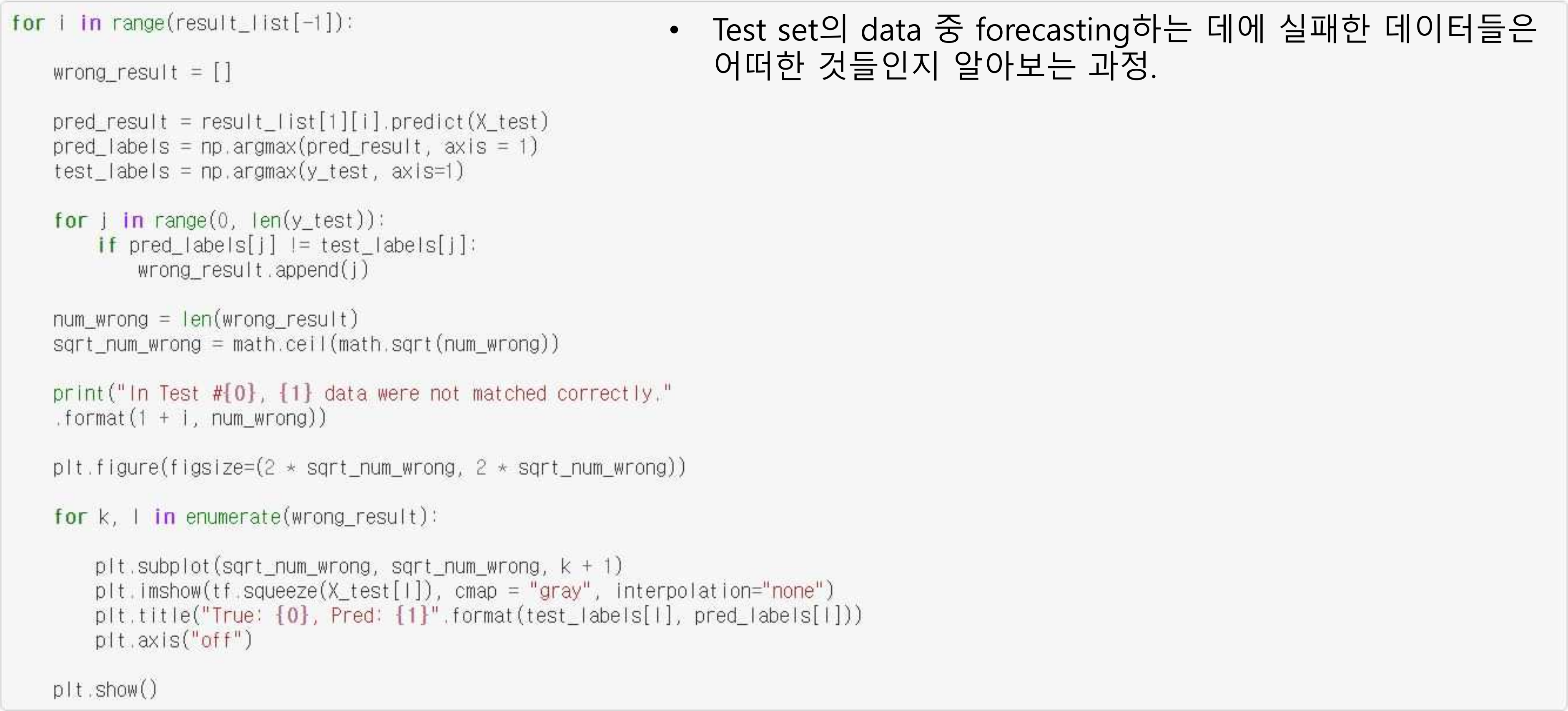
- 예시 (Model 5)



```
Train #5
Train Loss: 0.007630737964063883,      Train Accuracy: 0.9975000023841858
Validation Loss: 0.02187662571668625,   Validation Accuracy: 0.9945833086967468
```

3. Evaluation & Test

Review

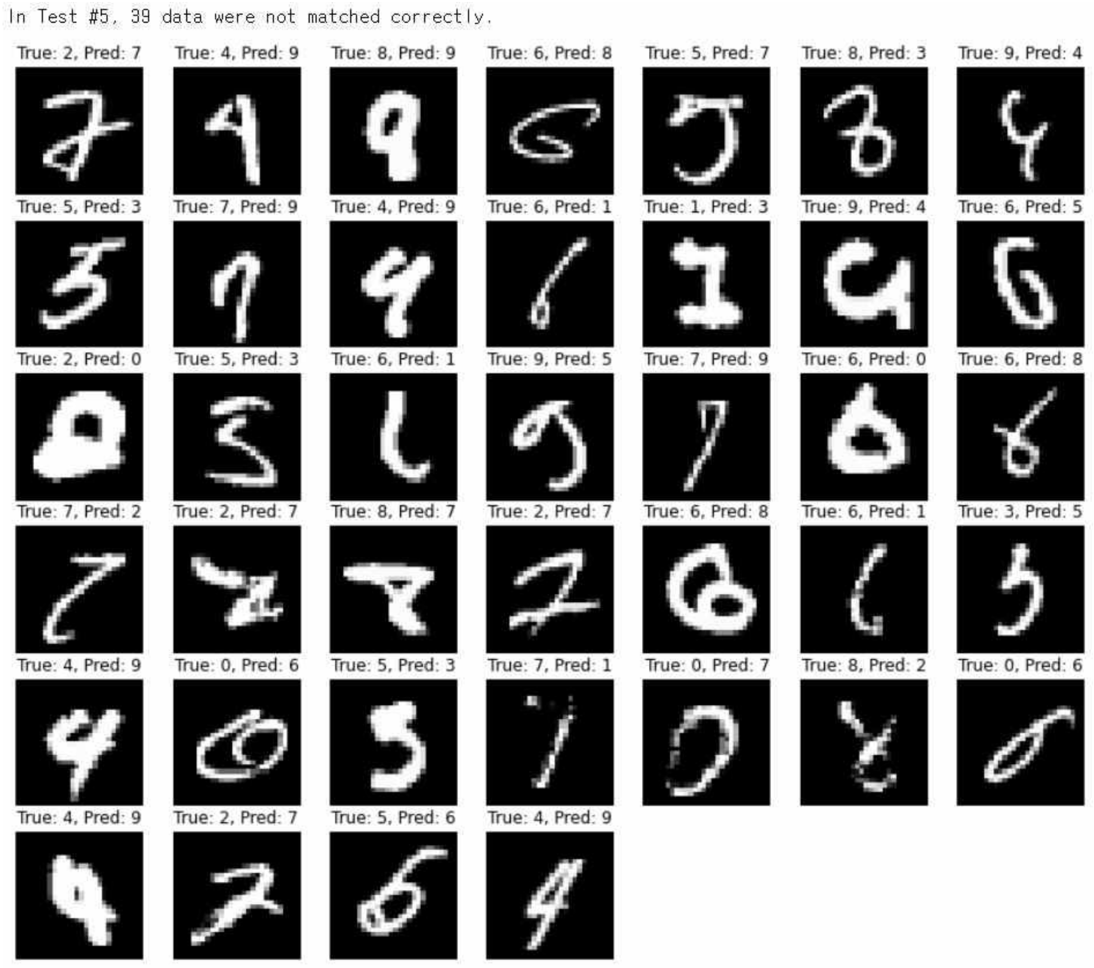


```
for i in range(result_list[-1]):

    wrong_result = []

    pred_result = result_list[1][i].predict(X_test)
    pred_labels = np.argmax(pred_result, axis = 1)
    test_labels = np.argmax(y_test, axis=1)

    for j in range(0, len(y_test)):
        if pred_labels[j] != test_labels[j]:
            wrong_result.append(j)

    num_wrong = len(wrong_result)
    sqrt_num_wrong = math.ceil(math.sqrt(num_wrong))

    print("In Test #{0}, {1} data were not matched correctly."
    .format(1 + i, num_wrong))

    plt.figure(figsize=(2 * sqrt_num_wrong, 2 * sqrt_num_wrong))

    for k, l in enumerate(wrong_result):

        plt.subplot(sqrt_num_wrong, sqrt_num_wrong, k + 1)
        plt.imshow(tf.squeeze(X_test[l]), cmap = "gray", interpolation="none")
        plt.title("True: {0}, Pred: {1}".format(test_labels[l], pred_labels[l]))
        plt.axis("off")

    plt.show()
```

- Test set의 data 중 forecasting하는 데에 실패한 데이터들은 어떠한 것들인지 알아보는 과정.

- 예시 (Model 5)

3. Evaluation & Test



```
In Test #5, 39 data were not matched correctly.
```

질의응답

감사합니다.