01. YAML

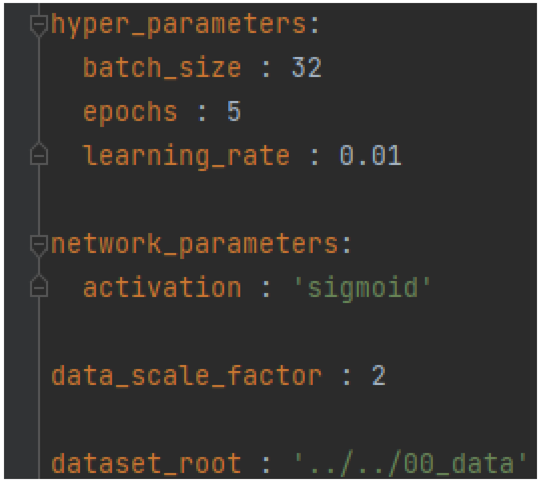
CheolSeong Park
tjd4987@naver.com

01. YAML

- JSON과 같이 데이터 직렬화에 사용되는 포맷
- 딥러닝 프로젝트에서 하이퍼파라미터 등의 변수를 YAML을 이용하여 관리할 수 있다.

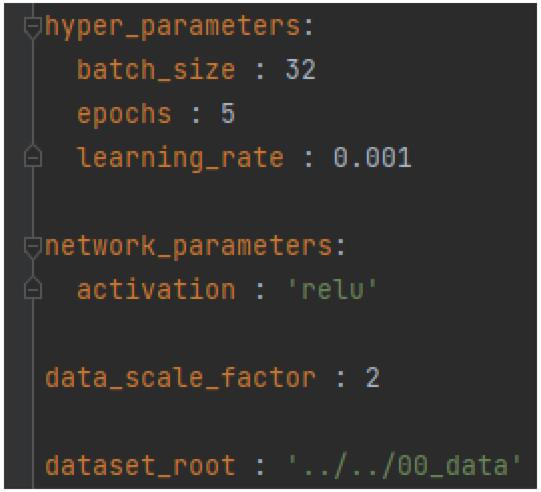
01. YAML

- YAML 예시



```yaml
hyper_parameters:
  batch_size : 32
  epochs : 5
  learning_rate : 0.01

network_parameters:
  activation : 'sigmoid'

data_scale_factor : 2

dataset_root : '../../00_data'
```

sigmoid.yaml



```yaml
hyper_parameters:
  batch_size : 32
  epochs : 5
  learning_rate : 0.001

network_parameters:
  activation : 'relu'

data_scale_factor : 2

dataset_root : '../../00_data'
```

relu.yaml

01. YAML

- Python에서는 yaml 라이브러리를 이용하여, *.yaml 파일을 읽을 수 있습니다.
  - 설치 예) pip install pyyaml
  - dict형태로 load 됨

```
import yaml

with open(‘sigmoid.yaml’) as f:
        config = yaml.safe_load(f)
```

01. YAML

- 무슨 장점이 있나요?
  - Argument로 코드 실행대비, cli 명령어가 간단해집니다
    - argument 사용

```
python main.py --batch_size 32 --epochs 5 --learning_rate
0.01 --activation ‘sigmoid’ --data_scale_factor 2
--dataset_root ../../00_data
```

    - yaml 사용

```
python main.py --config sigmoid.yaml
```

01. YAML

- 무슨 장점이 있나요?
  - 변수 추가시 코드 수정을 **덜** 할 수 있습니다.
    - argument 사용

```
…

parser.add_argument(‘--loss_weight’, type=float)

…

loss = loss * parser.loss_weight
```

- yaml 사용

```
loss = loss * config[‘loss_weight’]
```

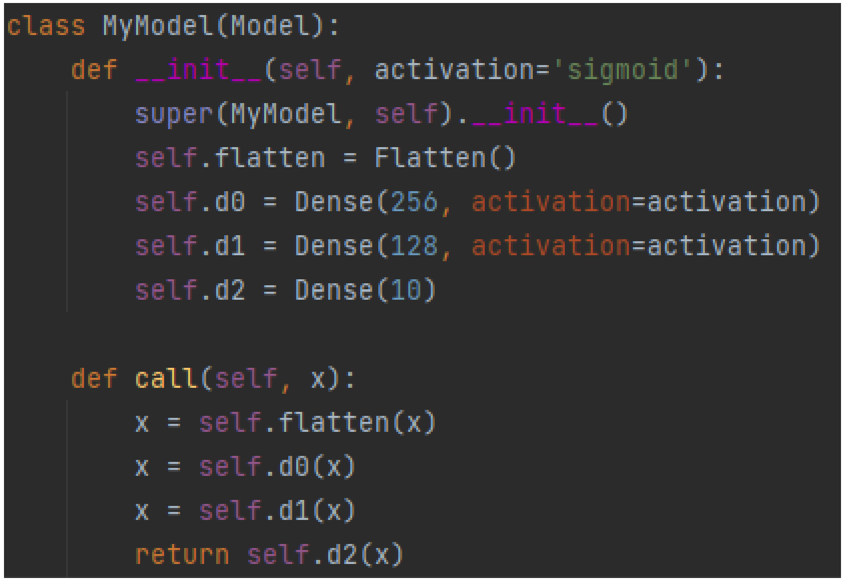
01. YAML

- 이제 MNIST 예제로 가볍게 사용해봅시다!

- sigmoid.yaml과 relu.yaml을 이용하여 activation이 sigmoid함수인 모델과 relu인 모델에 대하여 실험해 봅시다.

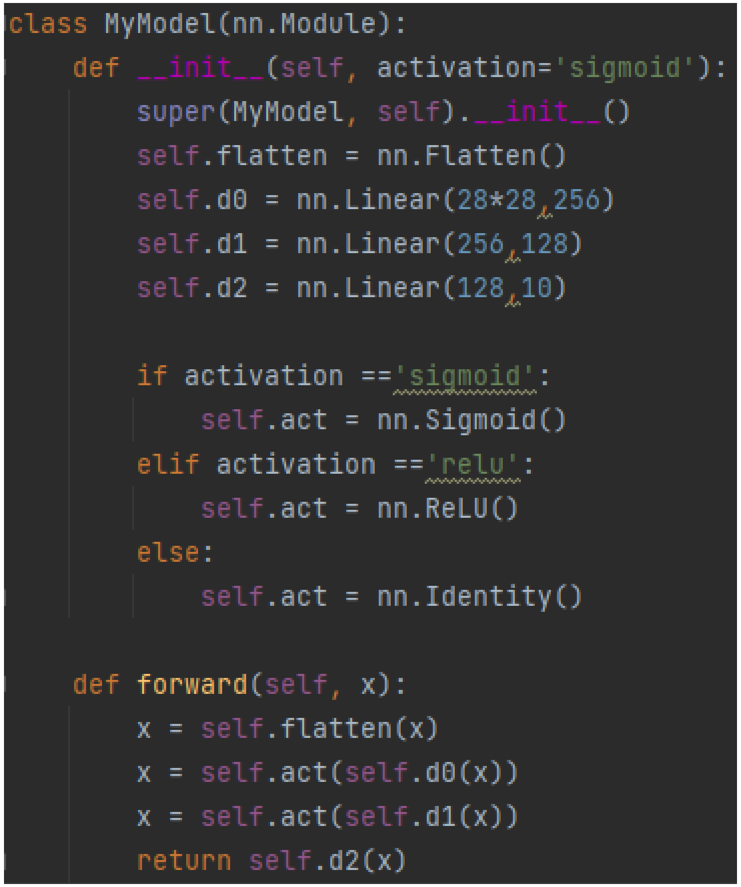
01. YAML

- 간단한 모델 분류기 정의



```python
class MyModel(Model):
    def __init__(self, activation='sigmoid'):
        super(MyModel, self).__init__()
        self.flatten = Flatten()
        self.d0 = Dense(256, activation=activation)
        self.d1 = Dense(128, activation=activation)
        self.d2 = Dense(10)

    def call(self, x):
        x = self.flatten(x)
        x = self.d0(x)
        x = self.d1(x)
        return self.d2(x)
```

tensorflow

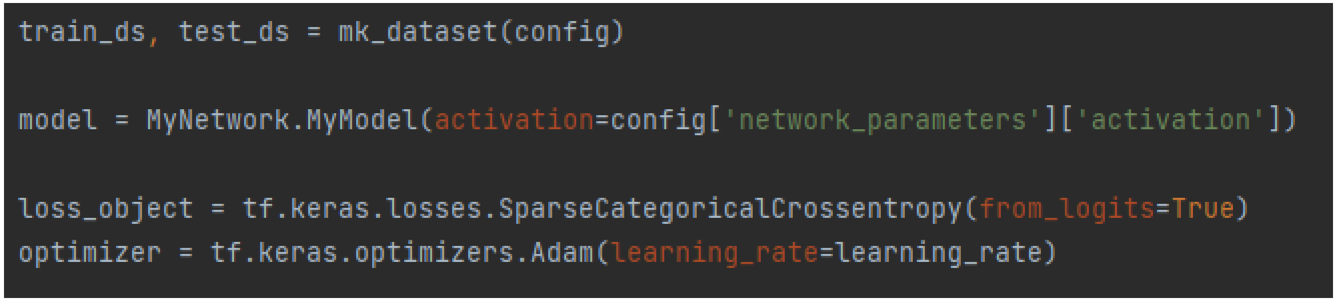


```python
class MyModel(nn.Module):
    def __init__(self, activation='sigmoid'):
        super(MyModel, self).__init__()
        self.flatten = nn.Flatten()
        self.d0 = nn.Linear(28*28,256)
        self.d1 = nn.Linear(256,128)
        self.d2 = nn.Linear(128,10)

        if activation =='sigmoid':
            self.act = nn.Sigmoid()
        elif activation =='relu':
            self.act = nn.ReLU()
        else:
            self.act = nn.Identity()

    def forward(self, x):
        x = self.flatten(x)
        x = self.act(self.d0(x))
        x = self.act(self.d1(x))
        return self.d2(x)
```

pytorch

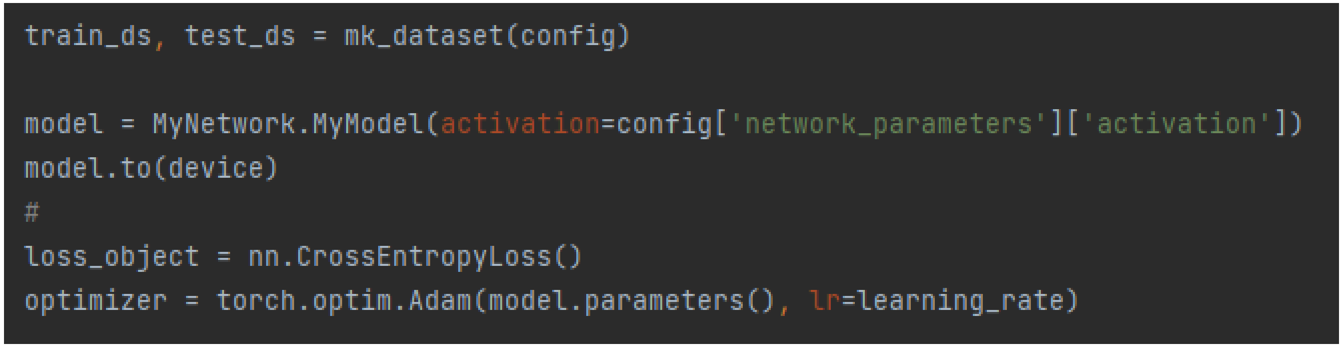
01. YAML

- dataset, model, loss, optimizer instance 정의
  - mk_dataset()은 github 혹은 appendix 참조



```
train_ds, test_ds = mk_dataset(config)

model = MyNetwork.MyModel(activation=config['network_parameters']['activation'])

loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
```

tensorflow

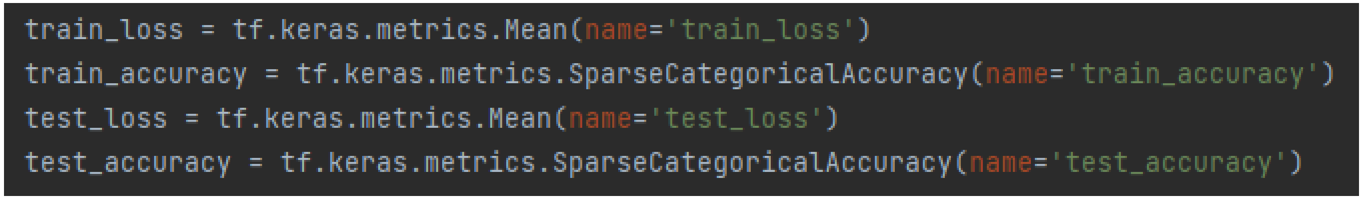


```
train_ds, test_ds = mk_dataset(config)

model = MyNetwork.MyModel(activation=config['network_parameters']['activation'])
model.to(device)
#
loss_object = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

pytorch

01. YAML

- metric helper instance 정의



```
train_loss = tf.keras.metrics.Mean(name='train_loss')
train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='train_accuracy')
test_loss = tf.keras.metrics.Mean(name='test_loss')
test_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='test_accuracy')
```

tensorflow

01. YAML

- train 및 test
  - Pseudo code

```
for epoch in EPOCHS
        init. metric helper

        train()
        test()

        print metric
```

  - 코드는 github 및 appendix 참조

01. YAML

- 실행
  - sigmoid model

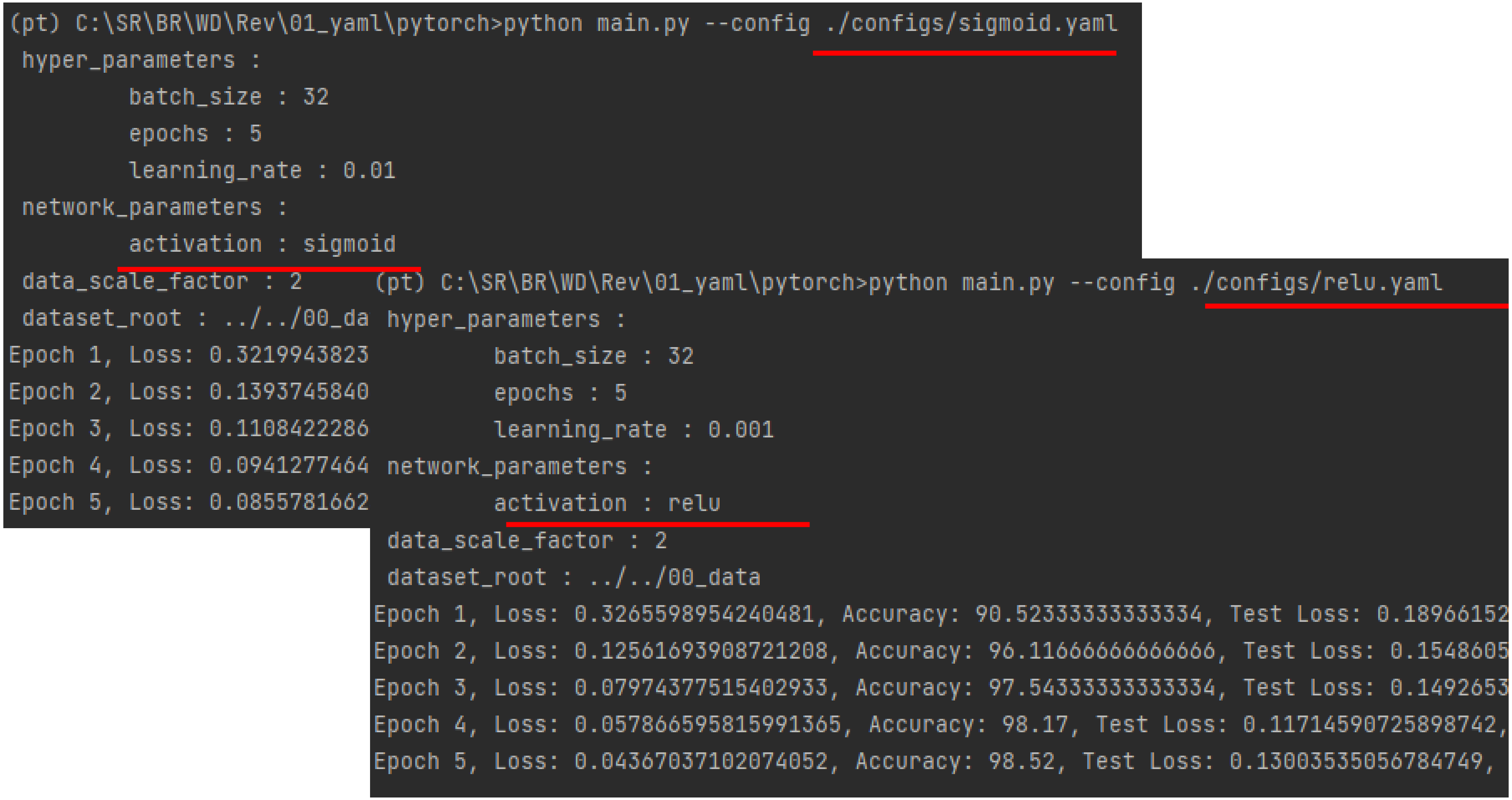
```
python main.py --config ./configs/sigmoid.yaml
```

  - relu model

```
python main.py --config ./configs/relu.yaml
```

01. YAML

- 실행 결과 예시

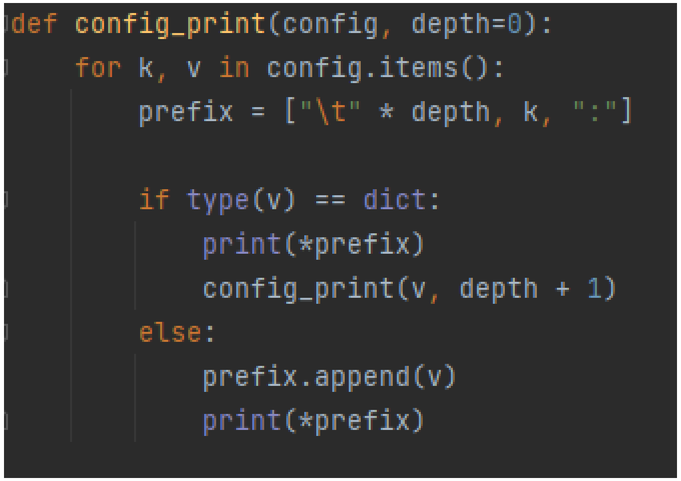


```
(pt) C:\SR\BR\WD\Rev\01_yaml\pytorch>python main.py --config ./configs/sigmoid.yaml
 hyper_parameters :
         batch_size : 32
         epochs : 5
         learning_rate : 0.01
 network_parameters :
         activation : sigmoid
 data_scale_factor : 2
 dataset_root : ../../00_da
Epoch 1, Loss: 0.3219943823
Epoch 2, Loss: 0.1393745840
Epoch 3, Loss: 0.1108422286
Epoch 4, Loss: 0.0941277464
Epoch 5, Loss: 0.0855781662
```

```
(pt) C:\SR\BR\WD\Rev\01_yaml\pytorch>python main.py --config ./configs/relu.yaml
 hyper_parameters :
         batch_size : 32
         epochs : 5
         learning_rate : 0.001
 network_parameters :
         activation : relu
 data_scale_factor : 2
 dataset_root : ../../00_data
Epoch 1, Loss: 0.3265598954240481, Accuracy: 90.52333333333334, Test Loss: 0.18966152
Epoch 2, Loss: 0.12561693908721208, Accuracy: 96.11666666666666, Test Loss: 0.1548605
Epoch 3, Loss: 0.07974377515402933, Accuracy: 97.54333333333334, Test Loss: 0.1492653
Epoch 4, Loss: 0.057866595815991365, Accuracy: 98.17, Test Loss: 0.11714590725898742,
Epoch 5, Loss: 0.04367037102074052, Accuracy: 98.52, Test Loss: 0.13003535056784749,
```

Appendix

- config_print()



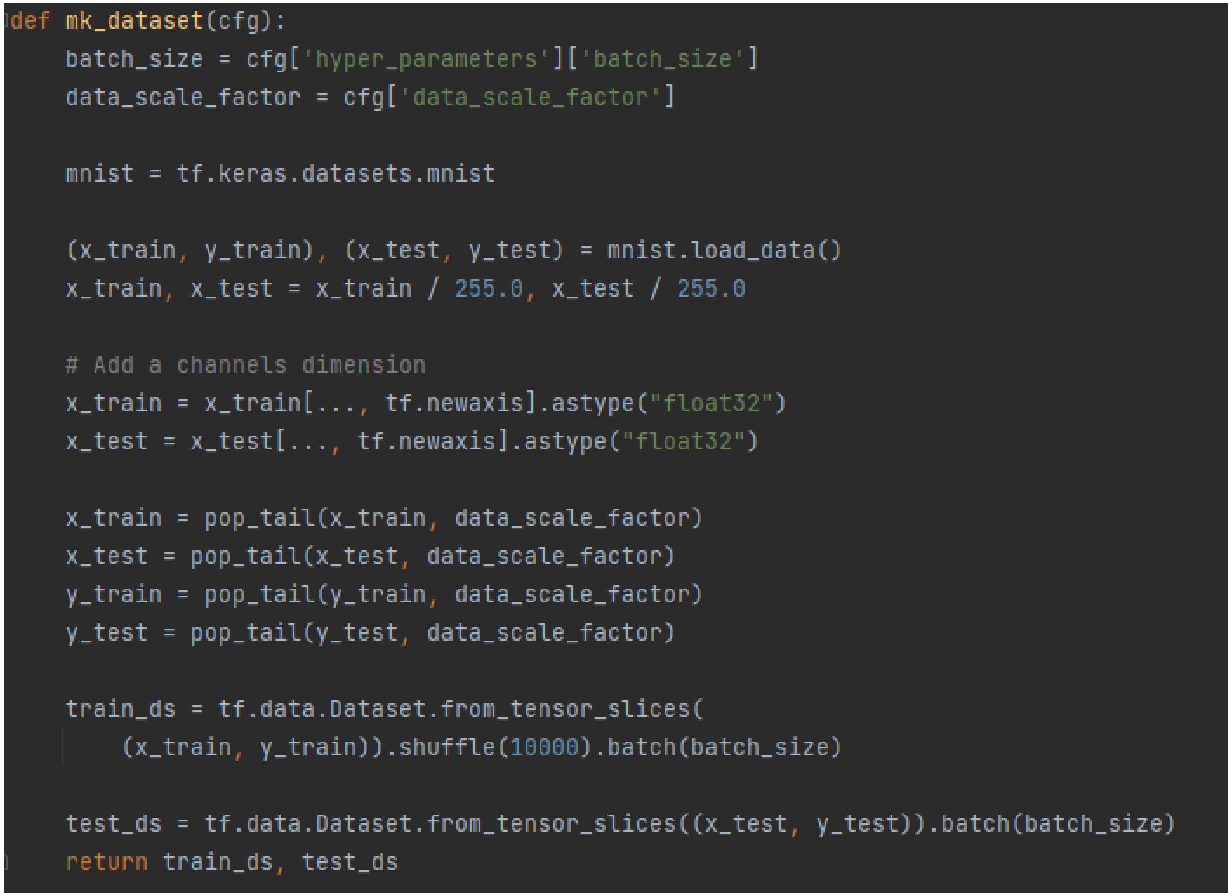
```
def config_print(config, depth=0):
    for k, v in config.items():
        prefix = ["\t" * depth, k, ":"]

        if type(v) == dict:
            print(*prefix)
            config_print(v, depth + 1)
        else:
            prefix.append(v)
            print(*prefix)
```

  - usage

```
import yaml

with open(‘sigmoid.yaml’) as f:
       config = yaml.safe_load(f)
       config_print(config)
```

Appendix_tensorflow

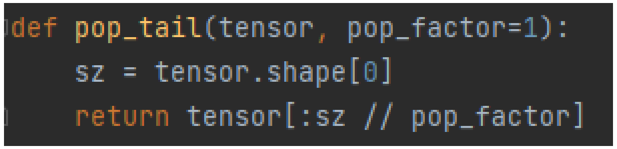
- mk_dataset()



```
def mk_dataset(cfg):
    batch_size = cfg['hyper_parameters']['batch_size']
    data_scale_factor = cfg['data_scale_factor']

    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

    # Add a channels dimension
    x_train = x_train[..., tf.newaxis].astype("float32")
    x_test = x_test[..., tf.newaxis].astype("float32")

    x_train = pop_tail(x_train, data_scale_factor)
    x_test = pop_tail(x_test, data_scale_factor)
    y_train = pop_tail(y_train, data_scale_factor)
    y_test = pop_tail(y_test, data_scale_factor)

    train_ds = tf.data.Dataset.from_tensor_slices(
        (x_train, y_train)).shuffle(10000).batch(batch_size)

    test_ds = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(batch_size)
    return train_ds, test_ds
```

Appendix_tensorflow

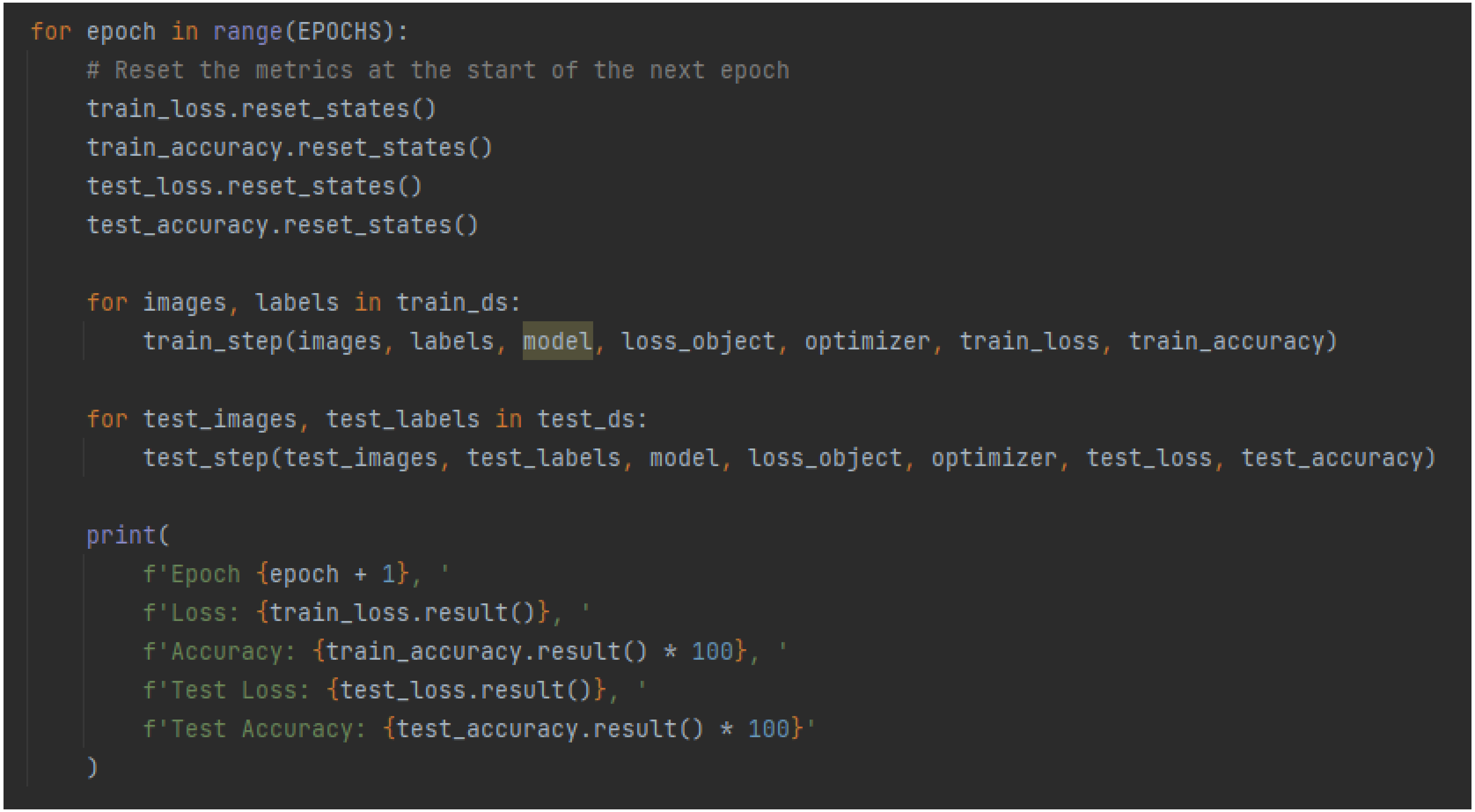
- pop_tail()
  - gpu를 사용하지 않는 경우, runtime 시간이 길어지기 때문에, 데이터셋 크기를 줄여주는 함수
  - *.yaml에서 $\frac{1}{dataset_scale_factor}$만큼 데이터셋 크기를 줄임



```
def pop_tail(tensor, pop_factor=1):
    sz = tensor.shape[0]
    return tensor[:sz // pop_factor]
```

Appendix_tensorflow

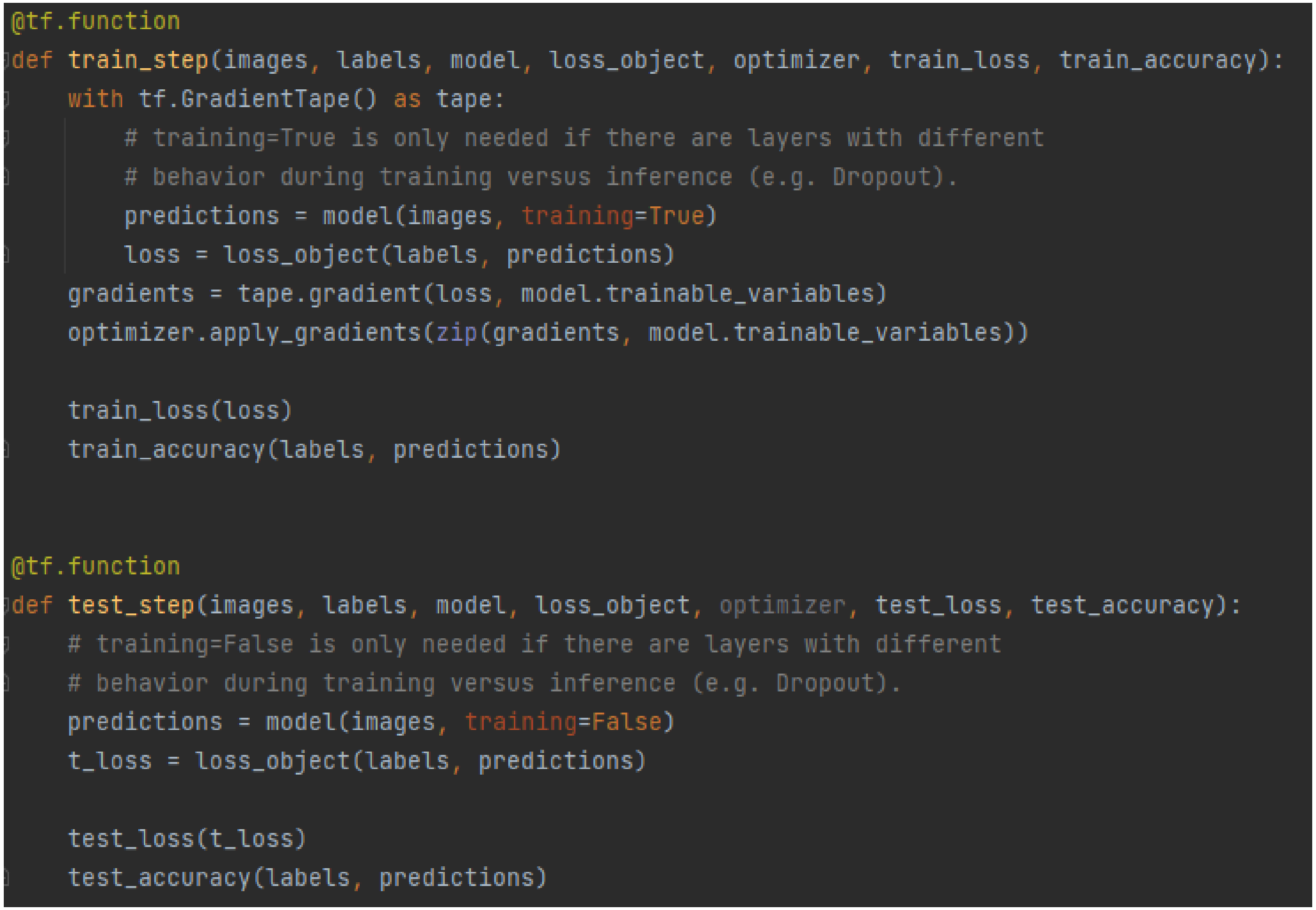
- train_test_pipeline



```
for epoch in range(EPOCHS):
    # Reset the metrics at the start of the next epoch
    train_loss.reset_states()
    train_accuracy.reset_states()
    test_loss.reset_states()
    test_accuracy.reset_states()

    for images, labels in train_ds:
        train_step(images, labels, model, loss_object, optimizer, train_loss, train_accuracy)

    for test_images, test_labels in test_ds:
        test_step(test_images, test_labels, model, loss_object, optimizer, test_loss, test_accuracy)

    print(
        f'Epoch {epoch + 1}, '
        f'Loss: {train_loss.result()}, '
        f'Accuracy: {train_accuracy.result() * 100}, '
        f'Test Loss: {test_loss.result()}, '
        f'Test Accuracy: {test_accuracy.result() * 100}'
    )
```

Appendix_tensorflow

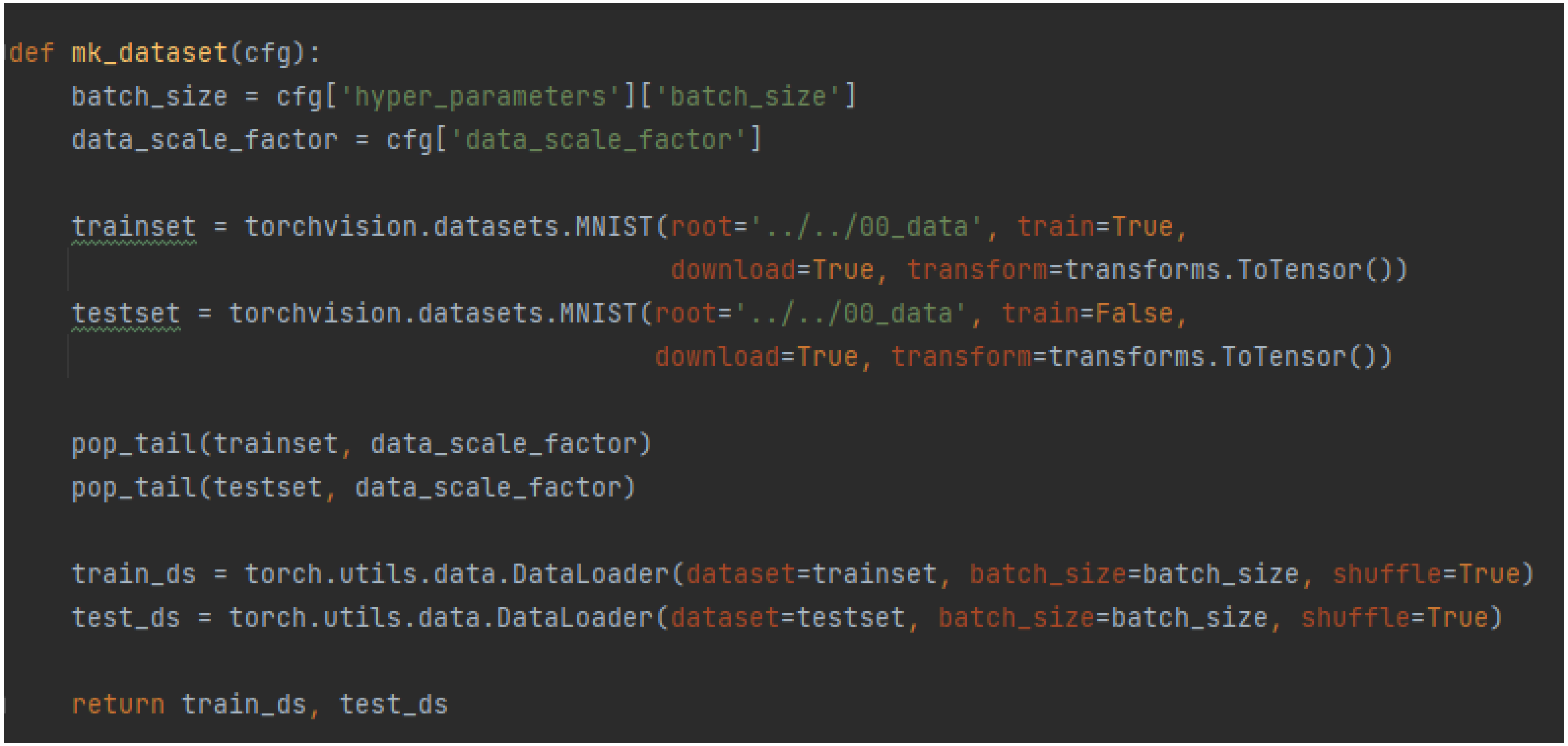
- train_step() and test_step()



```
@tf.function
def train_step(images, labels, model, loss_object, optimizer, train_loss, train_accuracy):
    with tf.GradientTape() as tape:
        # training=True is only needed if there are layers with different
        # behavior during training versus inference (e.g. Dropout).
        predictions = model(images, training=True)
        loss = loss_object(labels, predictions)
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))

    train_loss(loss)
    train_accuracy(labels, predictions)


@tf.function
def test_step(images, labels, model, loss_object, optimizer, test_loss, test_accuracy):
    # training=False is only needed if there are layers with different
    # behavior during training versus inference (e.g. Dropout).
    predictions = model(images, training=False)
    t_loss = loss_object(labels, predictions)

    test_loss(t_loss)
    test_accuracy(labels, predictions)
```

Appendix_pytorch

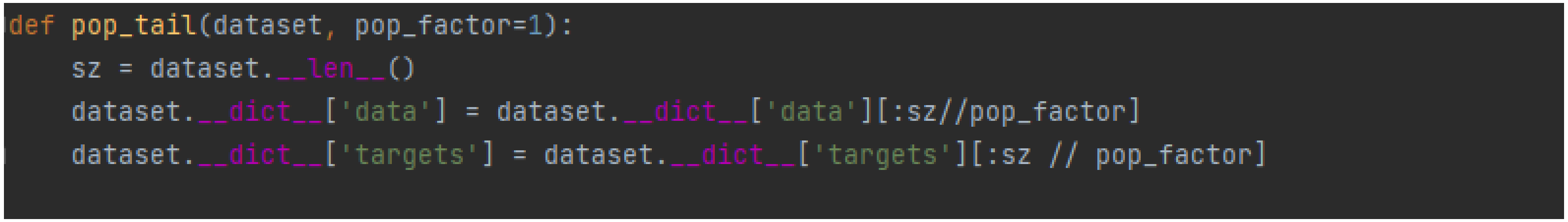
- mk_dataset()



```python
def mk_dataset(cfg):
    batch_size = cfg['hyper_parameters']['batch_size']
    data_scale_factor = cfg['data_scale_factor']

    trainset = torchvision.datasets.MNIST(root='../../00_data', train=True,
                                          download=True, transform=transforms.ToTensor())
    testset = torchvision.datasets.MNIST(root='../../00_data', train=False,
                                         download=True, transform=transforms.ToTensor())

    pop_tail(trainset, data_scale_factor)
    pop_tail(testset, data_scale_factor)

    train_ds = torch.utils.data.DataLoader(dataset=trainset, batch_size=batch_size, shuffle=True)
    test_ds = torch.utils.data.DataLoader(dataset=testset, batch_size=batch_size, shuffle=True)

    return train_ds, test_ds
```

Appendix_pytorch

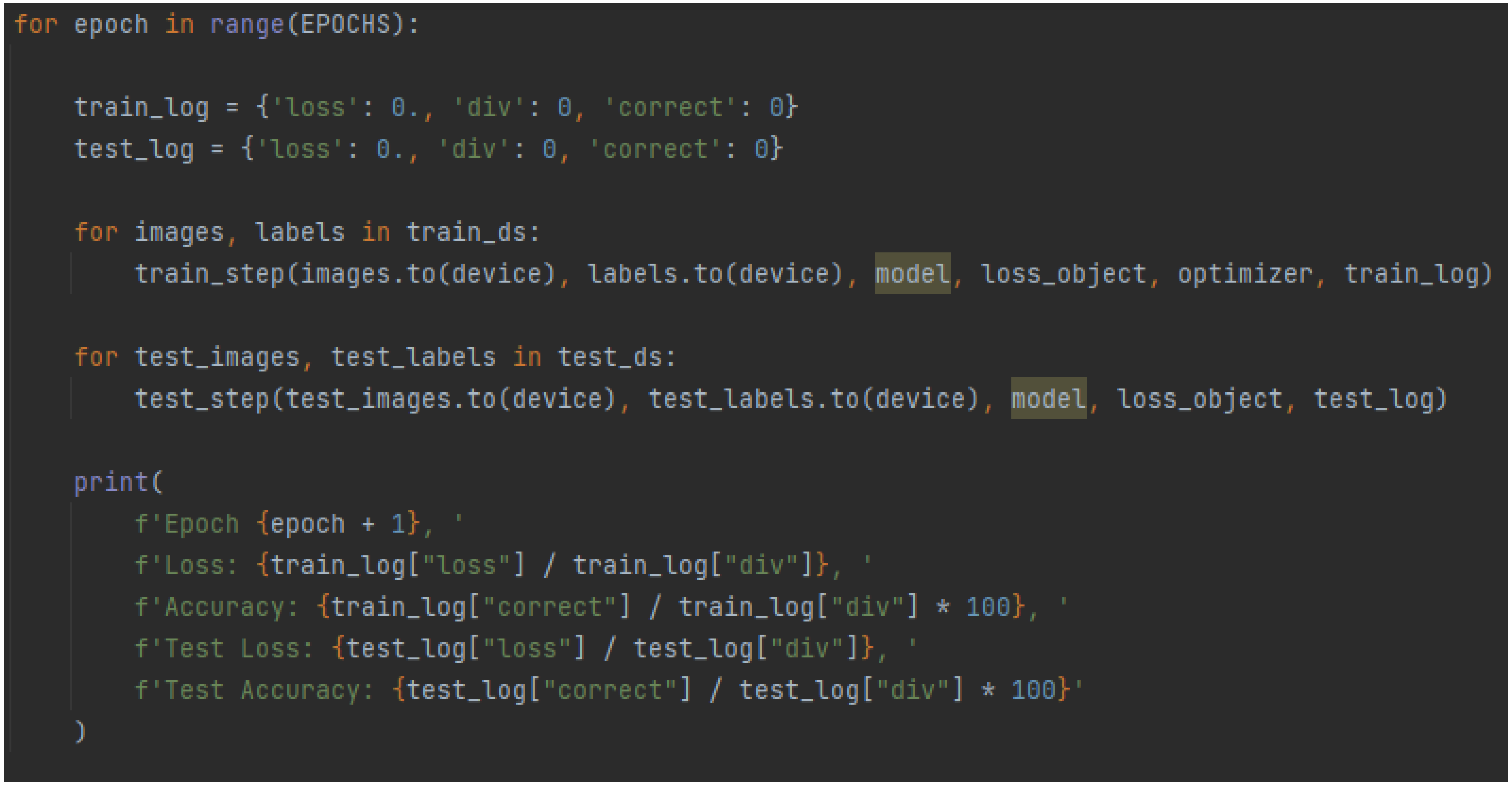
- pop_tail()
  - gpu를 사용하지 않는 경우, runtime 시간이 길어지기 때문에, 데이터셋 크기를 줄여주는 함수
  - *.yaml에서 $\frac{1}{dataset_scale_factor}$만큼 데이터셋 크기를 줄임



```python
def pop_tail(dataset, pop_factor=1):
    sz = dataset.__len__()
    dataset.__dict__['data'] = dataset.__dict__['data'][:sz//pop_factor]
    dataset.__dict__['targets'] = dataset.__dict__['targets'][:sz // pop_factor]
```

Appendix_pytorch

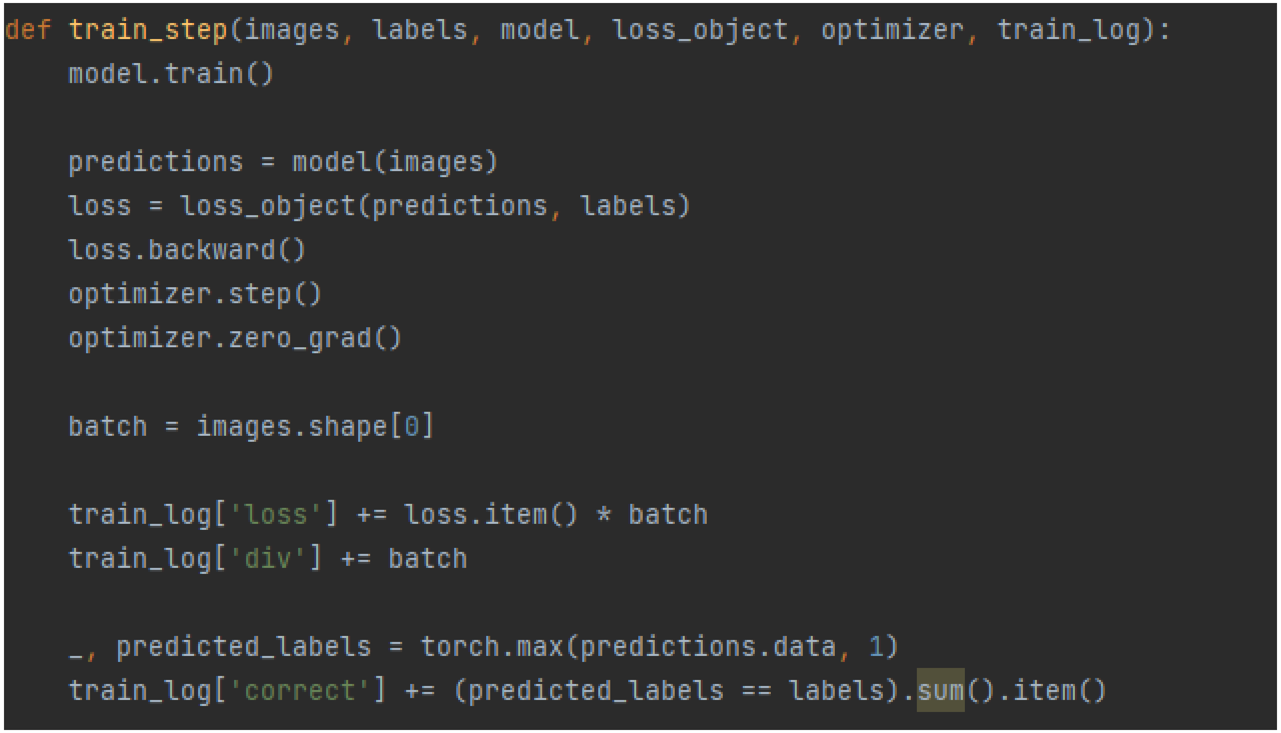
- train_test_pipeline



```
for epoch in range(EPOCHS):

    train_log = {'loss': 0., 'div': 0, 'correct': 0}
    test_log = {'loss': 0., 'div': 0, 'correct': 0}

    for images, labels in train_ds:
        train_step(images.to(device), labels.to(device), model, loss_object, optimizer, train_log)

    for test_images, test_labels in test_ds:
        test_step(test_images.to(device), test_labels.to(device), model, loss_object, test_log)

    print(
        f'Epoch {epoch + 1}, '
        f'Loss: {train_log["loss"] / train_log["div"]}, '
        f'Accuracy: {train_log["correct"] / train_log["div"] * 100}, '
        f'Test Loss: {test_log["loss"] / test_log["div"]}, '
        f'Test Accuracy: {test_log["correct"] / test_log["div"] * 100}'
    )
```

Appendix_pytorch

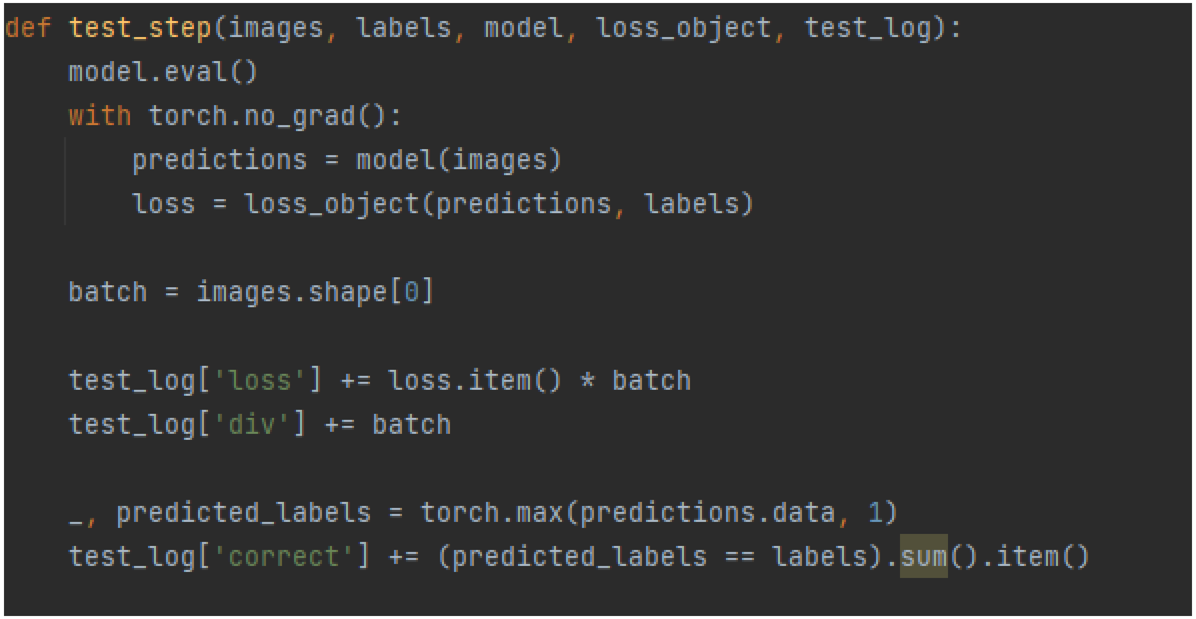
- train_step()



```
def train_step(images, labels, model, loss_object, optimizer, train_log):
    model.train()

    predictions = model(images)
    loss = loss_object(predictions, labels)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()

    batch = images.shape[0]

    train_log['loss'] += loss.item() * batch
    train_log['div'] += batch

    _, predicted_labels = torch.max(predictions.data, 1)
    train_log['correct'] += (predicted_labels == labels).sum().item()
```

Appendix_pytorch

- test_step()



```python
def test_step(images, labels, model, loss_object, test_log):
    model.eval()
    with torch.no_grad():
        predictions = model(images)
        loss = loss_object(predictions, labels)

    batch = images.shape[0]

    test_log['loss'] += loss.item() * batch
    test_log['div'] += batch

    _, predicted_labels = torch.max(predictions.data, 1)
    test_log['correct'] += (predicted_labels == labels).sum().item()
```