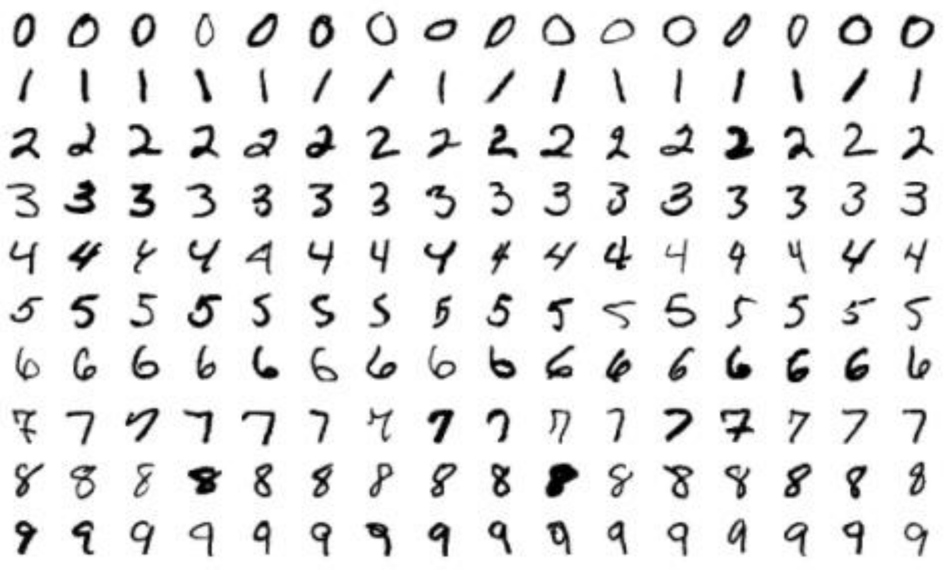
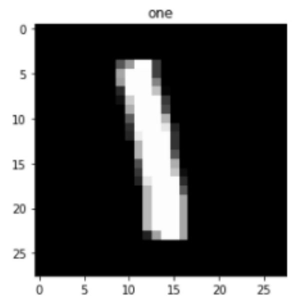
Exercise 3

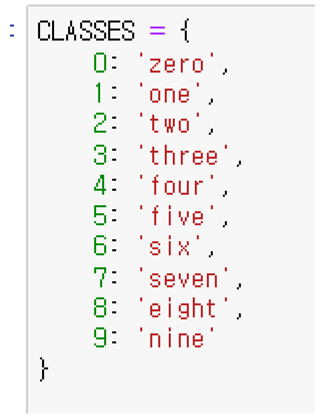
Dataset

http://yann.lecun.com/exdb/mnist/

- **Mnist (image , label)**



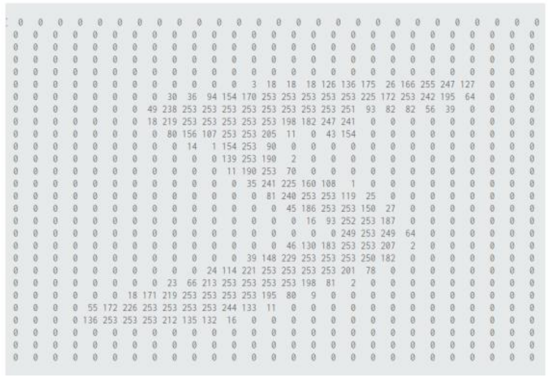




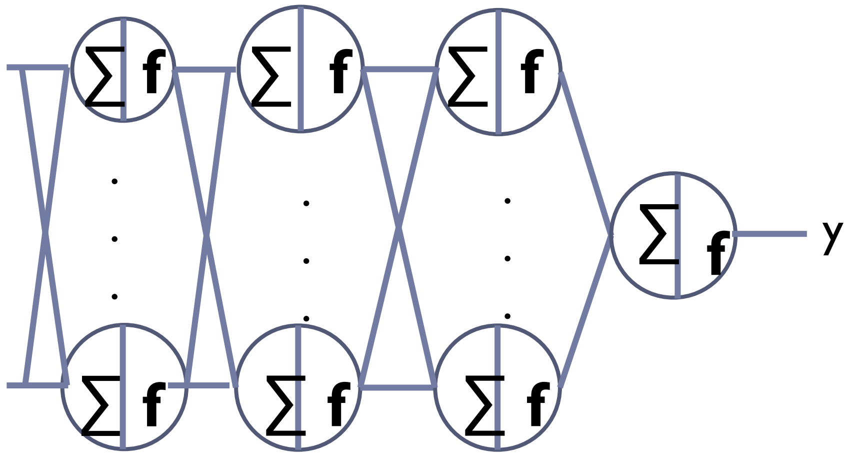
```
: CLASSES = {
      0: 'zero',
      1: 'one',
      2: 'two',
      3: 'three',
      4: 'four',
      5: 'five',
      6: 'six',
      7: 'seven',
      8: 'eight',
      9: 'nine'
  }
```

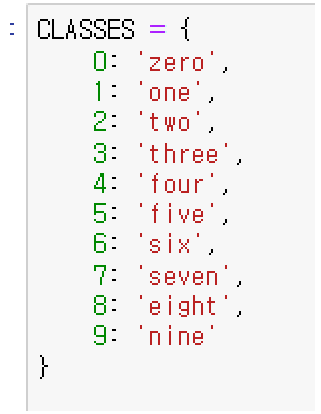
Coding 전에 생각해 볼 것

- 입력 데이터 : image(C , H , W)
- 출력 데이터 : class number (10개의 class)
- Optimizer: gradient descent method
- Loss function: cross entropy



```
:   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  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  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  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  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  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   0 139 253 190   2   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0  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  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  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  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  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   0 249 253 249  64   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  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  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  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  18 171 219 253 253 253 253 195  80   9   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0 136 253 253 253 212 135 132  16   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
```

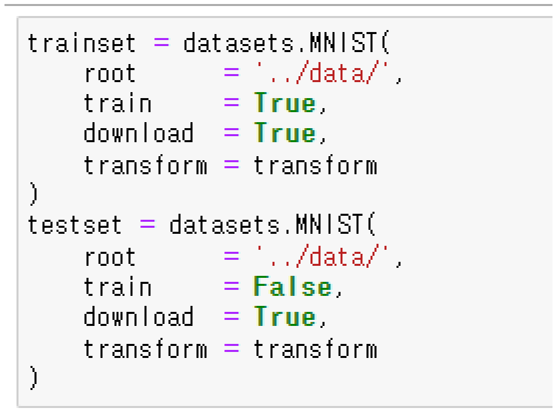




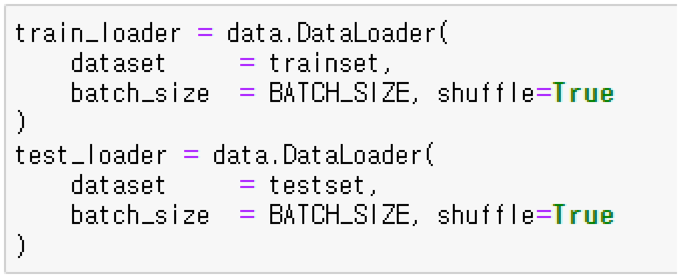
```
: CLASSES = {
      0: 'zero',
      1: 'one',
      2: 'two',
      3: 'three',
      4: 'four',
      5: 'five',
      6: 'six',
      7: 'seven',
      8: 'eight',
      9: 'nine'
  }
```

준비 단계 1

- 입력 데이터
- 출력 데이터

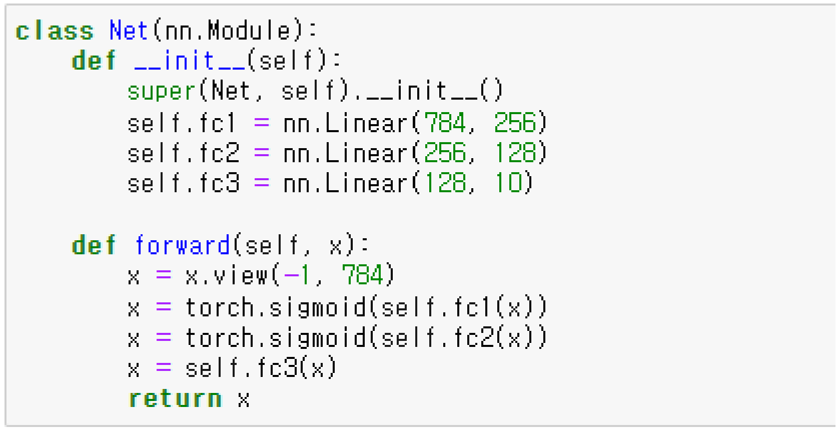


```
trainset = datasets.MNIST(
    root      = '../data/',
    train     = True,
    download  = True,
    transform = transform
)
testset = datasets.MNIST(
    root      = '../data/',
    train     = False,
    download  = True,
    transform = transform
)
```



```
train_loader = data.DataLoader(
    dataset     = trainset,
    batch_size  = BATCH_SIZE, shuffle=True
)
test_loader = data.DataLoader(
    dataset     = testset,
    batch_size  = BATCH_SIZE, shuffle=True
)
```

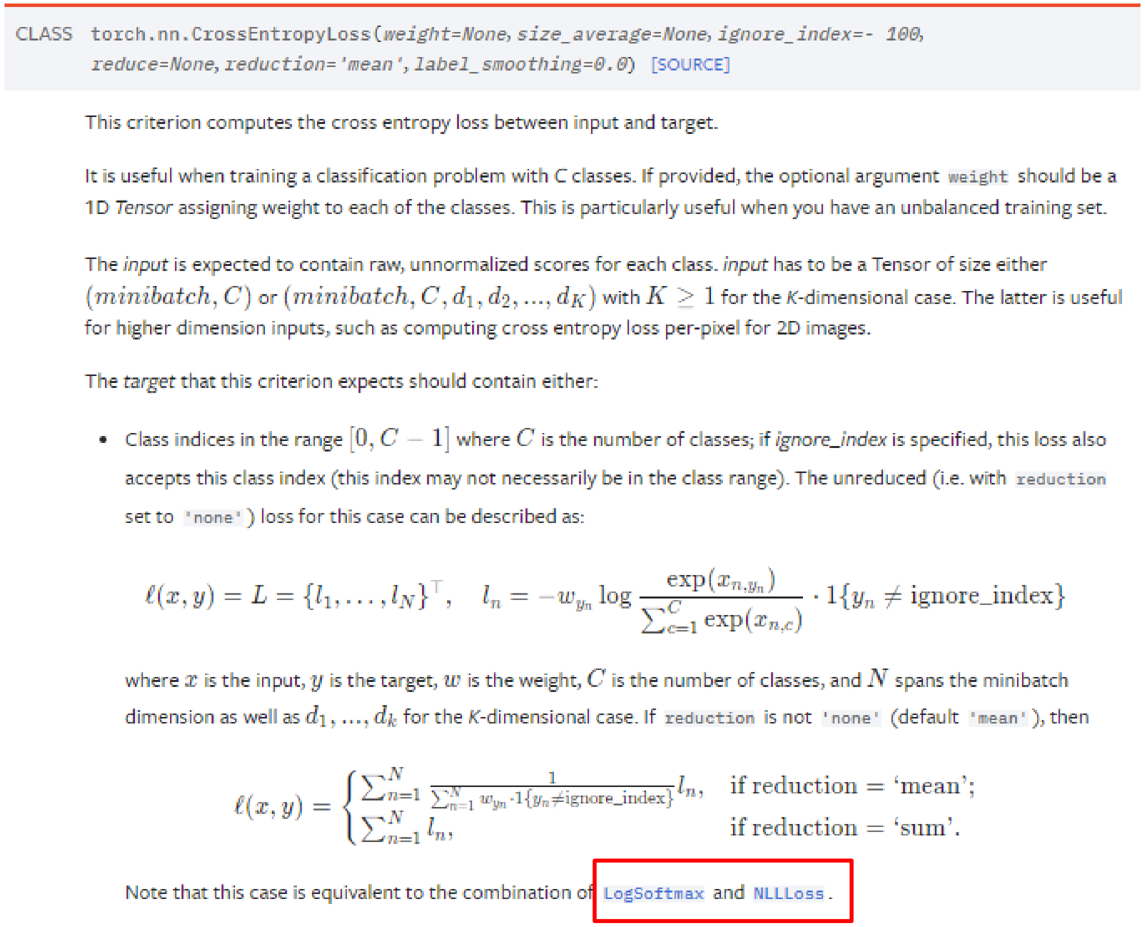
- Model



```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)

    def forward(self, x):
        x = x.view(-1, 784)
        x = torch.sigmoid(self.fc1(x))
        x = torch.sigmoid(self.fc2(x))
        x = self.fc3(x)
        return x
```

Exercise 3

Loss function



```
CLASS torch.nn.CrossEntropyLoss(weight=None, size_average=None, ignore_index=- 100,
      reduce=None, reduction='mean', label_smoothing=0.0) [SOURCE]
```

This criterion computes the cross entropy loss between input and target.

It is useful when training a classification problem with C classes. If provided, the optional argument `weight` should be a 1D *Tensor* assigning weight to each of the classes. This is particularly useful when you have an unbalanced training set.

The *input* is expected to contain raw, unnormalized scores for each class. *input* has to be a Tensor of size either $(minibatch, C)$ or $(minibatch, C, d_1, d_2, ..., d_K)$ with $K \geq 1$ for the *K*-dimensional case. The latter is useful for higher dimension inputs, such as computing cross entropy loss per-pixel for 2D images.

The *target* that this criterion expects should contain either:

- Class indices in the range $[0, C-1]$ where C is the number of classes; if *ignore_index* is specified, this loss also accepts this class index (this index may not necessarily be in the class range). The unreduced (i.e. with `reduction` set to `'none'`) loss for this case can be described as:

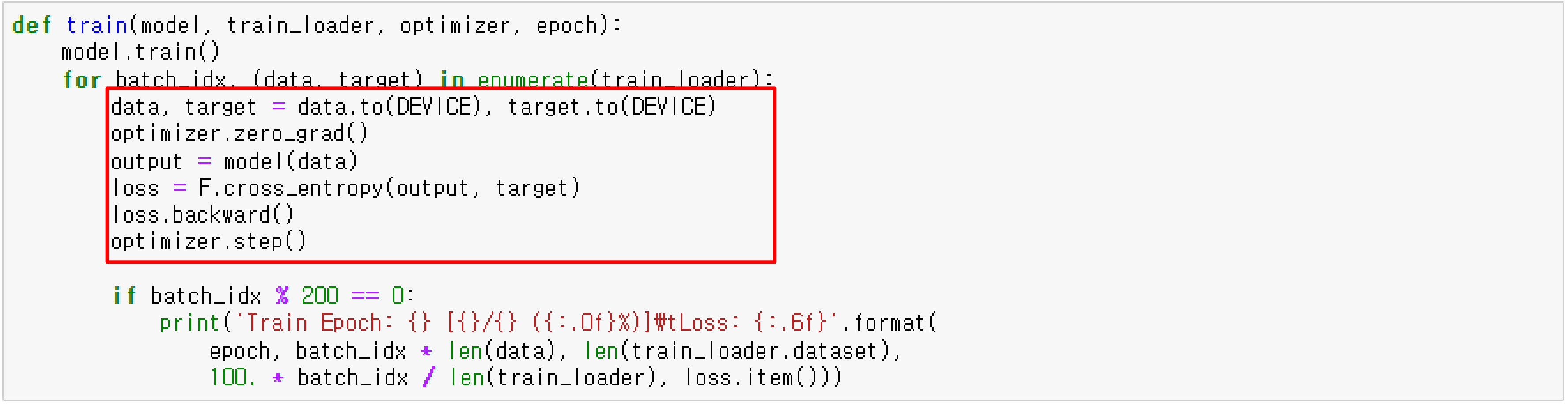
$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^{C} \exp(x_{n,c})} \cdot 1\{y_n \neq \text{ignore\_index}\}$$

where x is the input, y is the target, w is the weight, C is the number of classes, and N spans the minibatch dimension as well as $d_1, ..., d_k$ for the *K*-dimensional case. If `reduction` is not `'none'` (default `'mean'`), then

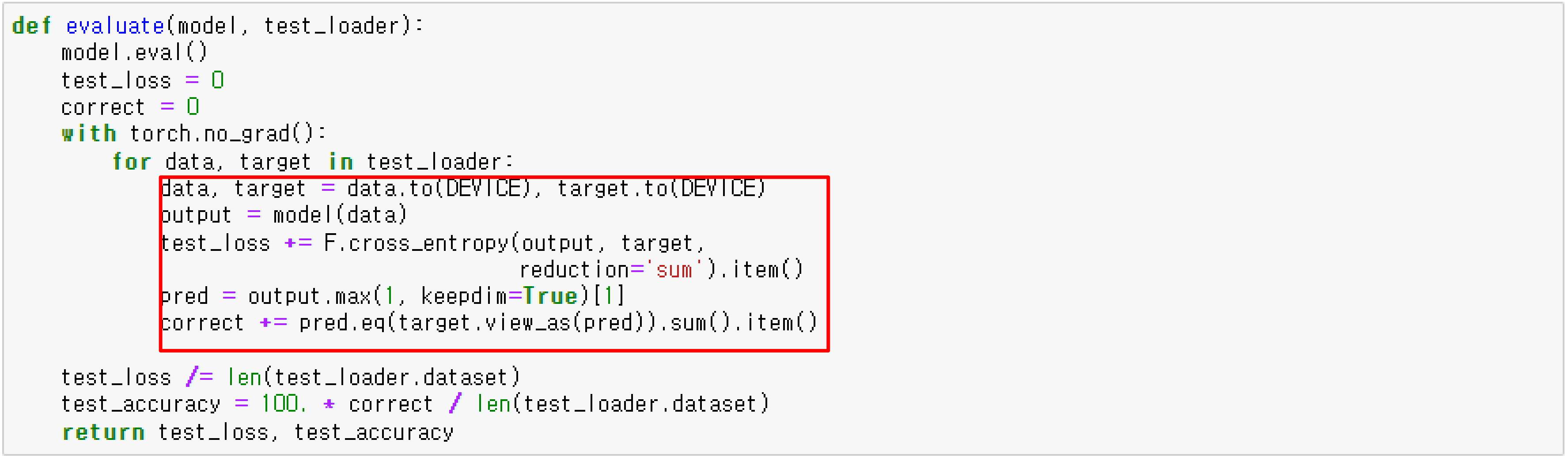
$$\ell(x, y) = \begin{cases} \sum_{n=1}^{N} \frac{1}{\sum_{n=1}^{N} w_{y_n} \cdot 1\{y_n \neq \text{ignore\_index}\}} l_n, & \text{if reduction} = \text{'mean'}; \\ \sum_{n=1}^{N} l_n, & \text{if reduction} = \text{'sum'}. \end{cases}$$

Note that this case is equivalent to the combination of `LogSoftmax` and `NLLLoss`.

Exercise 3

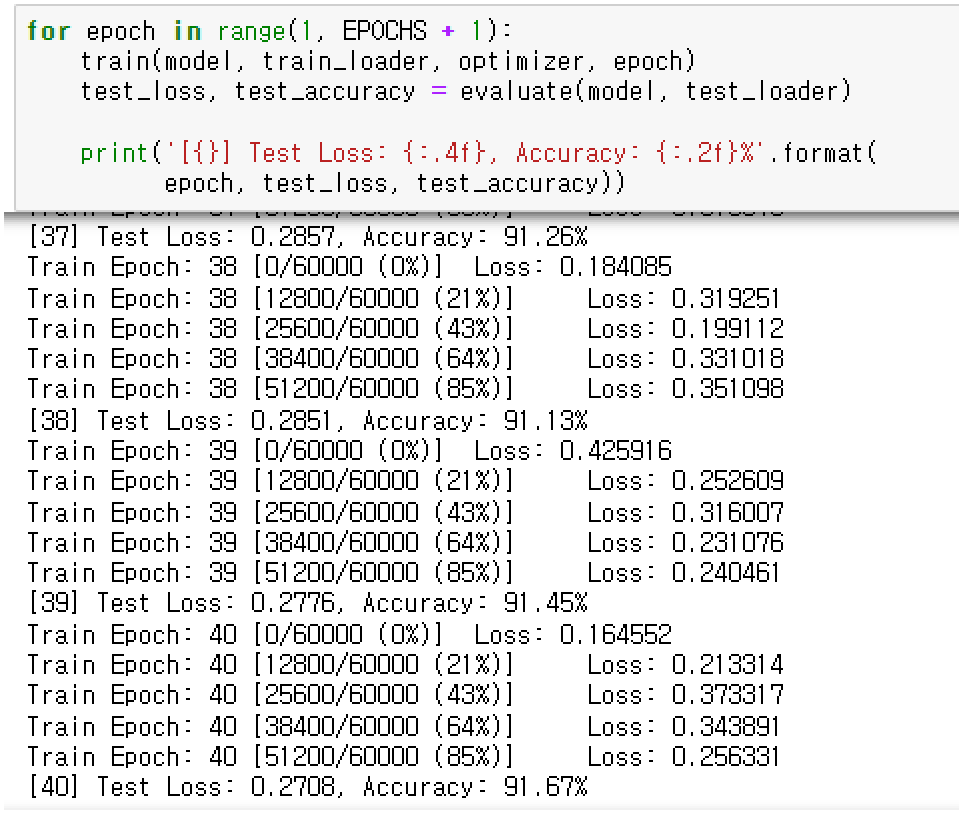


```
def train(model, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(DEVICE), target.to(DEVICE)
        optimizer.zero_grad()
        output = model(data)
        loss = F.cross_entropy(output, target)
        loss.backward()
        optimizer.step()

        if batch_idx % 200 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]₩tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
```



```
def evaluate(model, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(DEVICE), target.to(DEVICE)
            output = model(data)
            test_loss += F.cross_entropy(output, target,
                                         reduction='sum').item()
            pred = output.max(1, keepdim=True)[1]
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)
    test_accuracy = 100. * correct / len(test_loader.dataset)
    return test_loss, test_accuracy
```

Exercise 3



```python
for epoch in range(1, EPOCHS + 1):
    train(model, train_loader, optimizer, epoch)
    test_loss, test_accuracy = evaluate(model, test_loader)

    print('[{}] Test Loss: {:.4f}, Accuracy: {:.2f}%'.format(
          epoch, test_loss, test_accuracy))
```

```
[37] Test Loss: 0.2857, Accuracy: 91.26%
Train Epoch: 38 [0/60000 (0%)]	Loss: 0.184085
Train Epoch: 38 [12800/60000 (21%)]	Loss: 0.319251
Train Epoch: 38 [25600/60000 (43%)]	Loss: 0.199112
Train Epoch: 38 [38400/60000 (64%)]	Loss: 0.331018
Train Epoch: 38 [51200/60000 (85%)]	Loss: 0.351098
[38] Test Loss: 0.2851, Accuracy: 91.13%
Train Epoch: 39 [0/60000 (0%)]	Loss: 0.425916
Train Epoch: 39 [12800/60000 (21%)]	Loss: 0.252609
Train Epoch: 39 [25600/60000 (43%)]	Loss: 0.316007
Train Epoch: 39 [38400/60000 (64%)]	Loss: 0.231076
Train Epoch: 39 [51200/60000 (85%)]	Loss: 0.240461
[39] Test Loss: 0.2776, Accuracy: 91.45%
Train Epoch: 40 [0/60000 (0%)]	Loss: 0.164552
Train Epoch: 40 [12800/60000 (21%)]	Loss: 0.213314
Train Epoch: 40 [25600/60000 (43%)]	Loss: 0.373317
Train Epoch: 40 [38400/60000 (64%)]	Loss: 0.343891
Train Epoch: 40 [51200/60000 (85%)]	Loss: 0.256331
[40] Test Loss: 0.2708, Accuracy: 91.67%
```

Question and Answer