CNN으로 패션 아이템 구분하기

Convolutional Neural Network (CNN) 을 이용하여 패션아이템 구분 성능을 높여보겠습니다.

In [28]:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torchvision import transforms, datasets
```

In [29]:

```
USE_CUDA = torch.cuda.is_available()
DEVICE = torch.device("cuda" if USE_CUDA else "cpu")
```

In [30]:

```
EPOCHS     = 40
BATCH_SIZE = 64
```

데이터셋 불러오기

In [31]:

```
train_loader = torch.utils.data.DataLoader(
    datasets.FashionMNIST('./.data',
                   train=True,
                   download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.2860,), (0.3205,))
                   ])),
    batch_size=BATCH_SIZE, shuffle=True)
test_loader = torch.utils.data.DataLoader(
    datasets.FashionMNIST('./.data',
                   train=False,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.2860,), (0.3205,))
                   ])),
    batch_size=BATCH_SIZE, shuffle=True)
```

뉴럴넷으로 Fashion MNIST 학습하기

In [32]:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return x
```

하이퍼파라미터

`to()` 함수는 모델의 파라미터들을 지정한 곳으로 보내는 역할을 합니다. 일반적으로 CPU 1개만 사용할 경우 필요는 없지만, GPU를 사용하고자 하는 경우 `to("cuda")` 로 지정하여 GPU로 보내야 합니다. 지정하지 않을 경우 계속 CPU에 남아 있게 되며 빠른 훈련의 이점을 누리실 수 없습니다.

최적화 알고리즘으로 파이토치에 내장되어 있는 `optim.SGD` 를 사용하겠습니다.

In [33]:

```
model     = Net().to(DEVICE)
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)
```

학습하기

In [34]:

```
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()))
```

테스트하기

In [35]:

```
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)

            # 배치 오차를 합산
            test_loss += F.cross_entropy(output, target,
                                         reduction='sum').item()

            # 가장 높은 값을 가진 인덱스가 바로 예측값
            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
```

코드 돌려보기

자, 이제 모든 준비가 끝났습니다. 코드를 돌려서 실제로 학습이 되는지 확인해봅시다!

In [*]:

```
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))
```

```
Train Epoch: 12 [25600/60000 (43%)]     Loss: 0.460840
Train Epoch: 12 [38400/60000 (64%)]     Loss: 0.331351
Train Epoch: 12 [51200/60000 (85%)]     Loss: 0.347815
[12] Test Loss: 0.3786, Accuracy: 85.90%
Train Epoch: 13 [0/60000 (0%)]  Loss: 0.537173
Train Epoch: 13 [12800/60000 (21%)]     Loss: 0.494969
Train Epoch: 13 [25600/60000 (43%)]     Loss: 0.555594
Train Epoch: 13 [38400/60000 (64%)]     Loss: 0.553815
Train Epoch: 13 [51200/60000 (85%)]     Loss: 0.493981
[13] Test Loss: 0.3724, Accuracy: 86.06%
Train Epoch: 14 [0/60000 (0%)]  Loss: 0.598501
Train Epoch: 14 [12800/60000 (21%)]     Loss: 0.388055
Train Epoch: 14 [25600/60000 (43%)]     Loss: 0.295060
Train Epoch: 14 [38400/60000 (64%)]     Loss: 0.407462
Train Epoch: 14 [51200/60000 (85%)]     Loss: 0.580346

[14] Test Loss: 0.3736, Accuracy: 85.67%
Train Epoch: 15 [0/60000 (0%)]  Loss: 0.409256
Train Epoch: 15 [12800/60000 (21%)]     Loss: 0.449005
Train Epoch: 15 [25600/60000 (43%)]     Loss: 0.530770
```

201700949 설재혁

In [ ]: