**IVPG 과제1 보고서**

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### **전자정보통신공학과**

19010683

장성원

2025.03.24

## 

## **목차**

I. 서론

1. 문제 정의

a. Fashion-MNIST (1)

b. Fashion-MNIST (2)

c. CIFAR-10

II. 본론

1. 문제 풀이

a. Fashion-MNIST (1)

b. Fashion-MNIST (2)

c. CIFAR-10

III. 결론

1. 손실 그래프

2. 의문점

3. 전체 코드

a. Fashion-MNIST (1)

b. Fashion-MNIST (2)

c. CIFAR-10

**I. 서론**

1. 문제 정의

a. Fashion-MNIST (1)

1) Fashion MNIST dataset을 Trouser, Coat, Sneaker, Bag으로만 구성된 dataset으로 변경하세요.

이때, Trouser, Coat, Sneaker, Bag의 label을 각각 0, 1, 2, 3으로 설정하세요.

2) Flatten layer와 Dense layer 4개로 구성된 model을 만드세요.

이때, 각 layer의 neuron 개수는 적절하게 선택하세요.

3) 위의 모델을 학습하여 training data에서의 accuracy와 test data에서의 accuracy를 구하세요.

이때, hyperparameter들은 적절하게 선택하세요.

4) 4개 class에 대한 영상을 하나씩 선택하여 model.predict을 수행한 후, 네트워크의 출력을 확인하세요.

b. Fashion-MNIST (2)

1) Fashion MNIST dataset을 Trouser, Coat, Sneaker, Bag으로만 구성된 dataset으로 변경하세요.

이때, Trouser, Coat, Sneaker, Bag의 label을 각각 0, 1, 2, 3으로 설정하세요. (1번과 동일)

2) Lenet-5 model을 위의 4-class classification 문제를 풀도록 만드세요.

3) 위의 모델을 학습하여 training data에서의 accuracy와 test data에서의 accuracy를 구하세요.

이때, hyperparameter들은 적절하게 선택하세요.

4) 4개 class에 대한 영상을 하나씩 선택하여 model.predict을 수행한 후, 네트워크의 출력을 확인하세요.

c. CIFAR-10

1) CIFAR10 dataset을 airplane, automobile, ship으로만 구성된 dataset으로 변경하세요.

이때, airplane, automobile, ship의 label을 각각 0, 1, 2로 설정하세요.

2) VGG16을 불러와서 위의 3-class classification 문제를 풀도록 변경하세요.

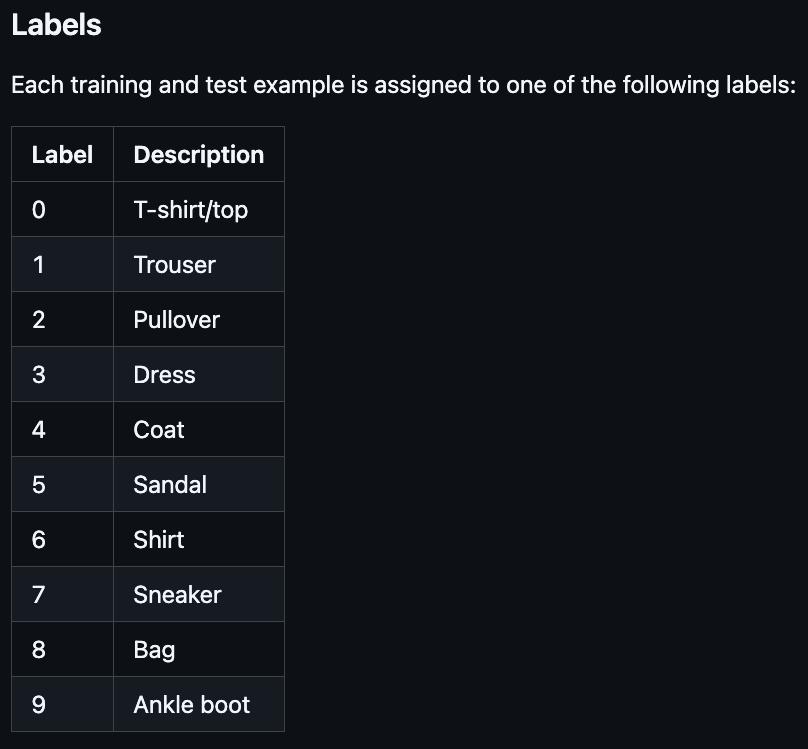
3) 위 모델에서 새로 추가된 layer들만을 학습하여 training data에서의 accuracy와 test data에서의 accuracy를 구하세요. 이때, hyperparameter들은 적절하게 선택하세요.

4) 위 모델의 모든 layer들을 학습하여 training data에서의 accuracy와 test data에서의 accuracy를 구하세요. 이때, hyperparameter들은 적절하게 선택하세요.

**II. 본론**

1. 문제풀이

a. Fashion-MNIST(1)

Fashion MNIST dataset의 경우 0번 라벨은 T-Shirt/top, 1번 라벨은 Trouser... 9번 라벨은 Ankle boot로 레이블이 정해져있는 데이터 셋이다. **이때 각각의 이미지는 28x28의 흑백 이미지이다.**

우선 목표로 하는 Trouser, Coat, Sneaker, Bag는 각각 1, 4, 7, 8번에 해당하므로 **해당 인덱스에 대한 데이터만 추출하여** 새로운 train\_subset\_dataset, test\_subest\_dataset을 구성한다. 이에 대한 코드는 다음과 같다.

| target\_labels = [1, 4, 7, 8]  train\_indices = [idx for idx, (img, label) in enumerate(train\_dataset) if label in target\_labels] test\_indices = [idx for idx, (img, label) in enumerate(test\_dataset) if label in target\_labels]  from torch.utils.data import Subset  train\_subset\_dataset = Subset(train\_dataset, train\_indices) test\_subset\_dataset = Subset(test\_dataset, test\_indices) |
| --- |

다음으로 Trouser, Coat, Sneaker, Bag의 label을 각각 0, 1, 2, 3으로 설정하기 위해 **데이터셋의 라벨을 바꾸어주는 함수를 작성**하고, train\_subset\_dataset과 test\_subset\_dataset에 대해 해당 함수를 적용한다. 이에 대한 코드는 다음과 같다.

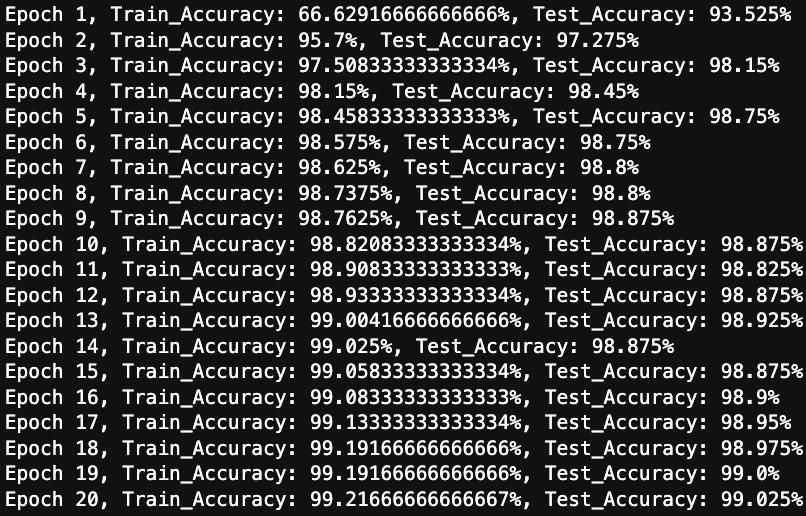
| class RemappedLabelDataset(Dataset):  def \_\_init\_\_(self, subset\_dataset, original\_label\_map, new\_label\_map):  self.subset\_dataset = subset\_dataset  self.original\_label\_map = original\_label\_map  self.new\_label\_map = new\_label\_map   def \_\_len\_\_(self):  return len(self.subset\_dataset)   def \_\_getitem\_\_(self, idx):  image, label = self.subset\_dataset[idx]  class\_name = self.original\_label\_map[label]  new\_label = self.new\_label\_map[class\_name]  return image, new\_label  original\_labels\_map = {0 : 'T-Shirt', 1 : 'Trouser', 2 : 'Pullover', 3 : 'Dress', 4 : 'Coat', 5 : 'Sandal', 6 : 'Shirt',  7 : 'Sneaker', 8 : 'Bag', 9 : 'Ankle Boot'}  my\_labels\_map = {'Trouser': 0, 'Coat': 1, 'Sneaker': 2, 'Bag': 3}  train\_subset\_dataset = RemappedLabelDataset(train\_subset\_dataset, original\_labels\_map, my\_labels\_map) test\_subset\_dataset = RemappedLabelDataset(test\_subset\_dataset, original\_labels\_map, my\_labels\_map) |
| --- |

다음으로 Flatten layer과 Dense layer 4개로 구성된 모델을 구성했다. **28x28의 흑백 이미지이므로, 첫 레이어의 in\_features는 1x28x28=784로 설정한다**. **파이토치의 view함수를 사용하면 텐서의 크기를 바꾸어줄 수 있다.** 각 layer의 출력에는 비선형 함수인 ReLU 함수를 두었다. **4개 중 하나로 분류해야 하는 문제이기 때문에, 마지막 레이어의 경우 out\_features=4로 두었다**. 학습률은 10^-4을 사용했으며, 손실 함수로는 CrossEntropy, optimizer로는 Adam을 사용했다. 이에 대한 코드는 다음과 같다.

| class FashionDNN(nn.Module):  def \_\_init\_\_(self):  super(FashionDNN,self).\_\_init\_\_()  self.fc1 = nn.Linear(in\_features=784,out\_features=128)  self.fc2 = nn.Linear(in\_features=128,out\_features=64)  self.fc3 = nn.Linear(in\_features=64,out\_features=32)  self.fc4 = nn.Linear(in\_features=32,out\_features=4)   def forward(self,input\_data):  out = input\_data.view(-1, 784)  out = F.relu(self.fc1(out))  out = F.relu(self.fc2(out))≠  out = F.relu(self.fc3(out))  out = self.fc4(out)  return out  learning\_rate = 1e-4; model = FashionDNN(); model.to(device)  criterion = nn.CrossEntropyLoss(); optimizer = torch.optim.Adam(model.parameters(), lr=learning\_rate); |
| --- |

Batch size는 200, epoch는 20을 주어 학습을 진행했다. 최종적으로 Train dataset에 대한 Accuracy는 99.21%, Test dataset에 대한 Accuracy는 99.025%임을 확인할 수 있었다. 이에 대한 코드는 다음과 같다.

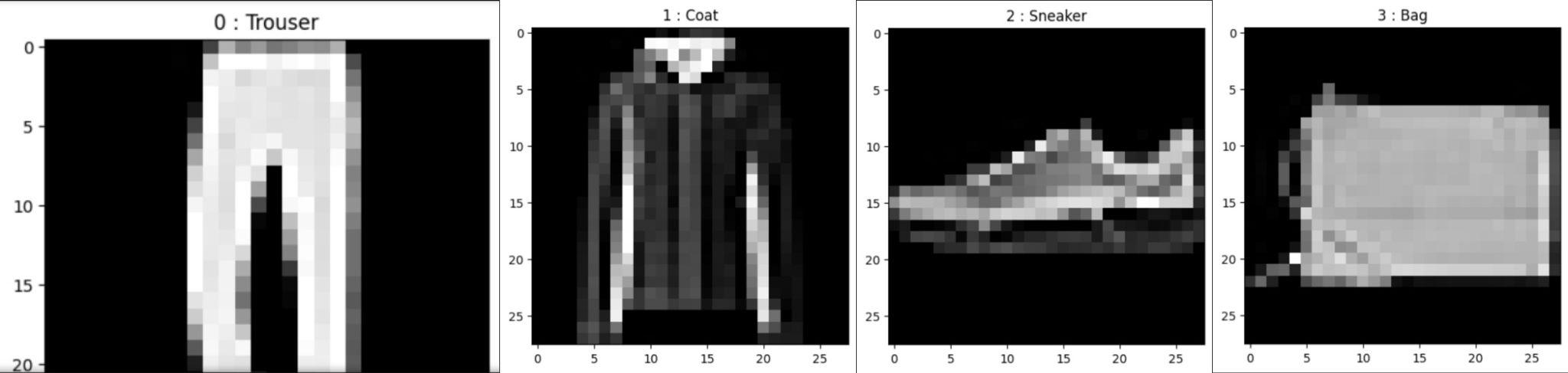
| num\_epochs = 20 count = 0 train\_loss\_list = [] test\_loss\_list = []  train\_accuracy\_list = [] test\_accuracy\_list = []  for epoch in range(num\_epochs):  train\_loss\_epoch = 0  test\_loss\_epoch = 0    # Training Phase  model.train() # 모델을 학습 모드로 설정  train\_correct = 0  train\_total = 0    for images, labels in train\_loader:  images, labels = images.to(device), labels.to(device)  train = images.view(-1, 1, 28, 28)    outputs = model(train)  loss = criterion(outputs, labels)   train\_predictions = torch.max(outputs, 1)[1]  train\_correct += (train\_predictions == labels).sum().item()  train\_total += labels.size(0)    optimizer.zero\_grad()  loss.backward()  optimizer.step()   train\_loss\_epoch += loss.item() # 미니배치 손실 합산   # Validation Phase  model.eval() # 모델을 평가 모드로 설정 (Dropout 등 비활성화)  test\_correct = 0  test\_total = 0    with torch.no\_grad(): # 기울기 업데이트 비활성화 (메모리 절약)  for images, labels in test\_loader:  images, labels = images.to(device), labels.to(device)  test = images.view(-1, 1, 28, 28)  outputs = model(test)    loss = criterion(outputs, labels)  test\_loss\_epoch += loss.item() # 미니배치 손실 합산    test\_predictions = torch.max(outputs, 1)[1]  test\_correct += (test\_predictions == labels).sum().item()  test\_total += labels.size(0)   train\_accuracy = train\_correct \* 100 / train\_total  test\_accuracy = test\_correct \* 100 / test\_total   train\_loss\_list.append(train\_loss\_epoch / len(train\_loader)) # 평균 손실 저장  test\_loss\_list.append(test\_loss\_epoch / len(test\_loader)) # 평균 손실 저장   train\_accuracy\_list.append(train\_accuracy)  test\_accuracy\_list.append(test\_accuracy)   print(f"Epoch {epoch+1}, Train\_Accuracy: {train\_accuracy}%, Test\_Accuracy: {test\_accuracy}%") |
| --- |



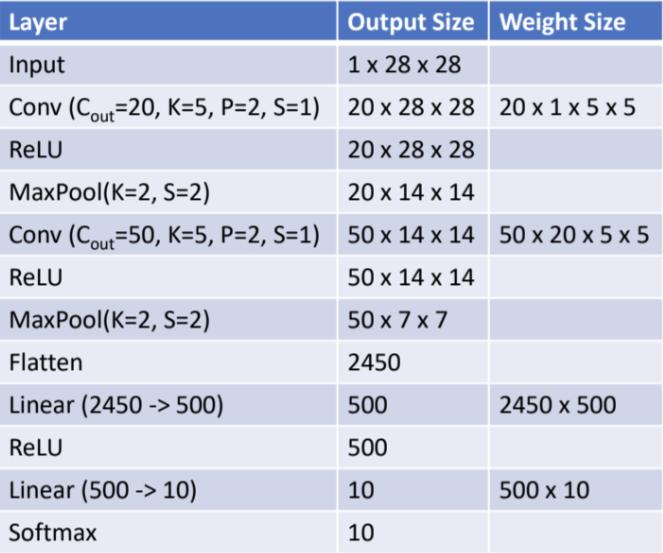
4개 클래스에 대한 영상을 하나씩 선택하여 결과를 출력했다. 이에 대한 코드는 다음과 같다.

| def test\_model(image) :  model.eval()  with torch.no\_grad() :  image = image.to(device)  test = image.view(-1, 1, 28, 28)  output = model(test)  predicted\_label = torch.argmax(output, 1).item()  predicted\_name = inv\_my\_labels\_map[predicted\_label]   image = image.to('cpu')  plt.imshow(image[0, :, :], cmap='grey')  plt.title(f'{predicted\_label} : {predicted\_name}')  img0, label0 = test\_subset\_dataset[0] test\_model(img0)   img1, label1 = test\_subset\_dataset[3] test\_model(img1)   img2, label2 = test\_subset\_dataset[4] test\_model(img2)   img3, label3 = test\_subset\_dataset[10] test\_model(img3) |
| --- |

아래는 테스트 데이터 셋의 각 영상에 대해 모델을 적용한 결과이다. 각 영상이 어떤 이미지인지 정확하게 예측하는 것을 확인할 수 있었다.



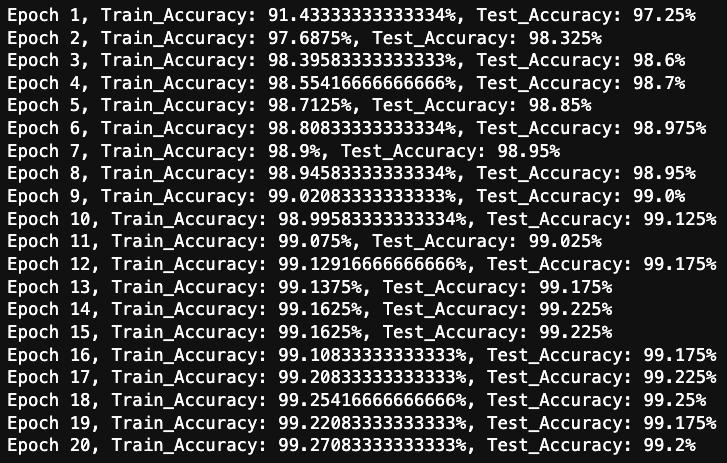
b. Fashion-MNIST(2)

앞선 Fashion-MNIST(1) 과제와 거의 유사하나, **Lenet-5 model을 직접 선언해주는 과정**이 필요했다. Lenet-5 model은 아래와 같은 구조로 이루어진 모델이다.

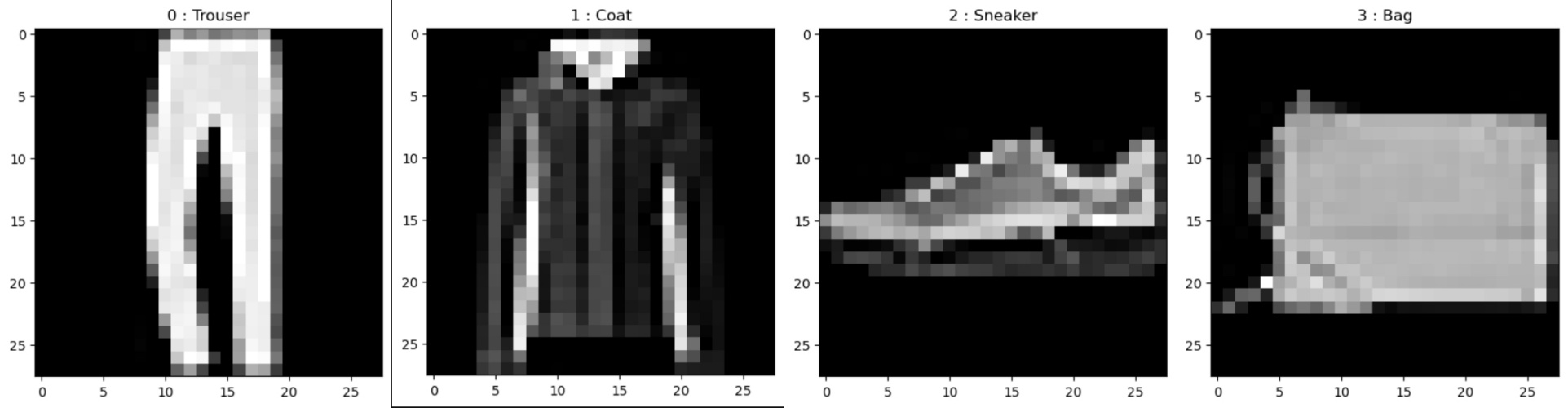
이에 대한 코드는 다음과 같다. 마찬가지로 **4개 중 하나로 분류해야 하는 문제이기 때문에, 마지막 fully connected layer에 대해 out\_features를 4로 설정했다**.

| class Lenet\_5(nn.Module):  def \_\_init\_\_(self):  super(Lenet\_5, self).\_\_init\_\_()  self.layer1 = nn.Sequential(  nn.Conv2d(in\_channels=1, out\_channels=20, kernel\_size=5, padding=2, stride=1),  nn.ReLU(),  nn.MaxPool2d(kernel\_size=2, stride=2)  )  self.layer2 = nn.Sequential(  nn.Conv2d(in\_channels=20, out\_channels=50, kernel\_size=5, padding=2, stride=1),  nn.ReLU(),  nn.MaxPool2d(kernel\_size=2, stride=2)  )  self.fc1 = nn.Linear(in\_features=2450, out\_features=500)  self.fc2 = nn.Linear(in\_features=500, out\_features=4)    def forward(self,input\_data):  out = self.layer1(input\_data)  out = self.layer2(out)  out = out.view(out.size(0), -1)  out = self.fc1(out)  out = F.relu(out)  out = self.fc2(out)  out = F.softmax(out, dim=1)  return out |
| --- |

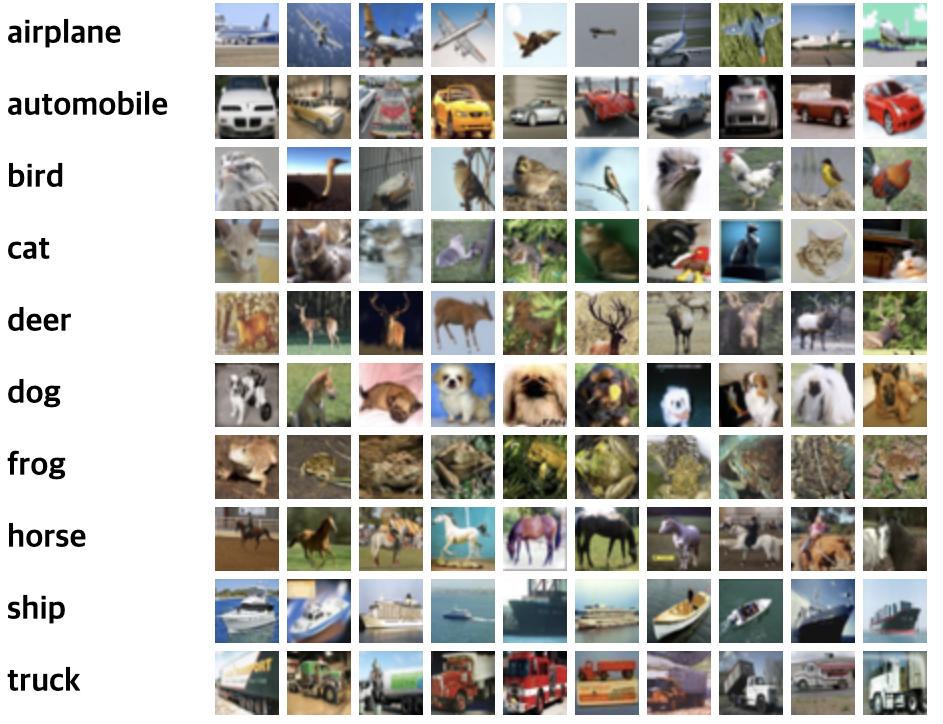
마찬가지로 Batch size는 200, epoch는 20을 주어 학습을 진행했다. 최종적으로 Train dataset에 대한 Accuracy는 99.27%, Test dataset에 대한 Accuracy는 99.2%임을 확인할 수 있었다. 코드는 위의 과정과 완전히 동일하다.



아래는 테스트 데이터 셋의 각 영상에 대해 모델을 적용한 결과이다. 각 영상이 어떤 이미지인지 정확하게 예측하는 것을 확인할 수 있었다.



c. CIFAR-10

CIFAR-10 dataset의 경우 0번 라벨은 airplane, 1번 라벨은 automobile… 9번 라벨은 truck로 레이블이 정해져 있는 데이터 셋이다. **이때 각각의 이미지는 32x32의 컬러 이미지이다.**

우선 목표로 하는 airplane, automobile, ship은 각각 0, 1, 8번에 해당하므로 **해당 인덱스에 대한 데이터만 뽑아내어** 새로운 train\_dataset, test\_dataset을 구성한다. 이에 대한 코드는 다음과 같다.

| target\_labels = [0, 1, 8] train\_indices = [idx for idx, (img, label) in enumerate(train\_dataset) if label in target\_labels] test\_indices = [idx for idx, (img, label) in enumerate(test\_dataset) if label in target\_labels]  from torch.utils.data import Subset train\_subset\_dataset = Subset(train\_dataset, train\_indices) test\_subset\_dataset = Subset(test\_dataset, test\_indices) |
| --- |

다음으로 airplane, automobile, ship의 label을 각각 0, 1, 2로 설정하기 위해 **데이터셋의 라벨을 바꾸어주는 함수를 작성**하고, train\_dataset과 test\_dataset에 대해 해당 함수를 적용한다. 이에 대한 코드는 다음과 같다.

| class RemappedLabelDataset(Dataset):  def \_\_init\_\_(self, subset\_dataset, original\_label\_map, new\_label\_map):  self.subset\_dataset = subset\_dataset  self.original\_label\_map = original\_label\_map  self.new\_label\_map = new\_label\_map   def \_\_len\_\_(self):  return len(self.subset\_dataset)   def \_\_getitem\_\_(self, idx):  image, label = self.subset\_dataset[idx]  class\_name = self.original\_label\_map[label]  new\_label = self.new\_label\_map[class\_name]  return image, new\_label  original\_labels\_map = {0 : 'airplane', 1 : 'automobile', 2 : 'bird', 3 : 'cat', 4 : 'deer', 5 : 'dog', 6 : 'frog',  7 : 'horse', 8 : 'ship', 9 : 'truck'}  my\_labels\_map = {'airplane': 0, 'automobile': 1, 'ship': 2}  train\_subset\_dataset = RemappedLabelDataset(train\_subset\_dataset, original\_labels\_map, my\_labels\_map) test\_subset\_dataset = RemappedLabelDataset(test\_subset\_dataset, original\_labels\_map, my\_labels\_map) |
| --- |

다음으로 VGG16 모델을 불러와 전이 학습을 활용하여 3개의 클래스 중 하나로 분류하는 문제를 풀어야 한다. **torchvision의 models 라이브러리를 활용하면 VGG16, ResNet과 같은 이미 잘 알려진 모델들을 불러올 수 있다**. 전이 학습을 위해 VGG16 모델의 특징 추출 layer만 불러오고, 이후 fully-connected layer은 직접 작성하는 방법을 사용했다.이때 입력 이미지가 3 x 32 x 32이며 **2x2 커널, stride = 2인 풀링 층을 5번 거쳤기 때문에 입력 이미지의 가로, 세로 길이가 2^5 만큼 작아지게 된다.** 따라서 영상의 가로, 세로의 길이는 32/32 = 1이 되기 때문에, fully-connected layer의 첫 nn.Linear을 nn.Linear(512 \* 1 \* 1, 4096)으로 구성하였다. 또한 **3개의 클래스에 대한 분류를 진행해야 하므로 마지막 레이어의 경우 nn.Linear(500, 3)으로 구성하였다.** 이에 대한 코드는 다음과 같다.

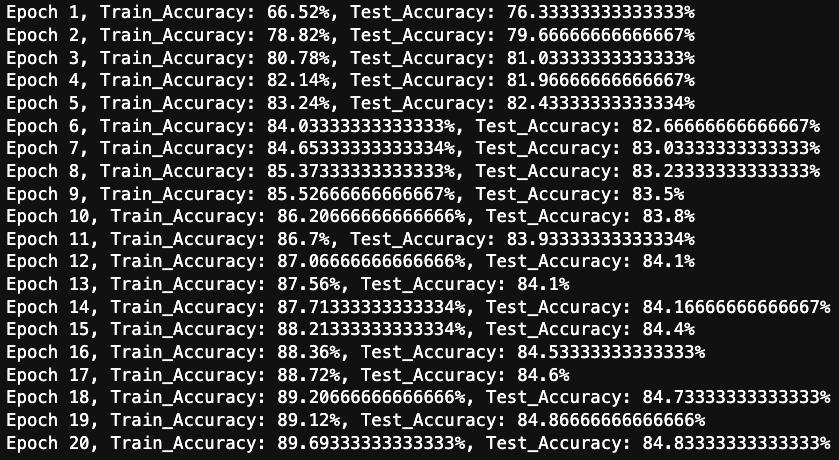
| from torchvision import models  vgg16 = models.vgg16(pretrained=True)  class MyVGG16Net(nn.Module):  def \_\_init\_\_(self):  super(MyVGG16Net, self).\_\_init\_\_()  # 기존의 VGG16 모델을 base model로써 사용  base\_model = models.vgg16(pretrained=True)  self.features = base\_model.features   # 새로운 classifier의 정의  self.fc = nn.Sequential(  nn.Linear(512 \* 1 \* 1, 4096), # MaxPooling을 5번 거쳤으므로 이미지의 가로, 세로 사이즈가 2^5만큼 작아지게 된다  nn.BatchNorm1d(4096),  nn.ReLU(),  nn.Linear(4096, 500),  nn.BatchNorm1d(500),  nn.ReLU(),  nn.Linear(500,3)  )   def forward(self, x):  x = self.features(x)  x = x.view(x.size(0), -1)  x = self.fc(x)  return x  my\_model = MyVGG16Net() my\_model.to(device) |
| --- |

우선 새로 추가된 layer에 대해서만 먼저 학습을 진행해보기 위해 **backbone network인 VGG-16넷의 파라미터를 얼리는 과정**이 필요하다. 이에 대한 코드는 다음과 같다.

| for param in my\_model.parameters():  param.requires\_grad = False # frozen  for param in my\_model.fc.parameters():  param.requires\_grad = True # 마지막 레이어는 autograd 활성화 |
| --- |

학습 진행 시 학습률은 4 \* 10^-4을 사용했으며, 손실 함수로는 CrossEntropy, optimizer로는 Adam을 사용했다. Batch size는 200, epoch는 20을 주어 학습을 진행했다. 최종적으로 Train dataset에 대한 Accuracy는 89.69%, Test dataset에 대한 Accuracy는 84.83%임을 확인할 수 있었다. 이에 대한 코드는 다음과 같다.

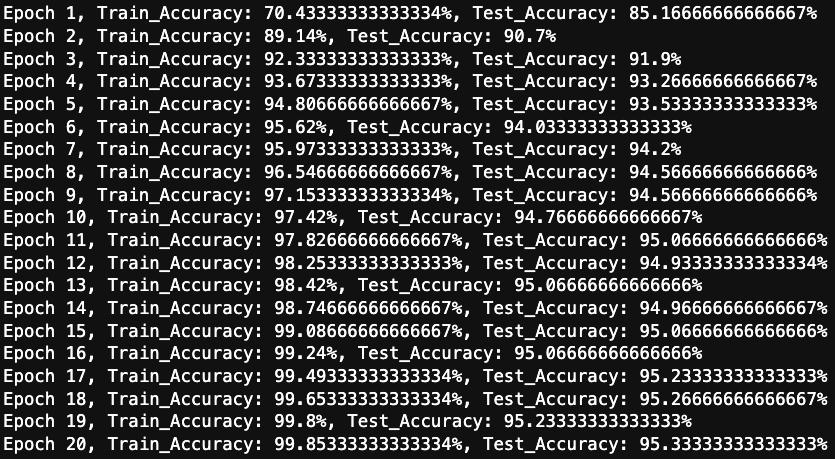
| learning\_rate = 4 \* 1e-06 criterion = nn.CrossEntropyLoss() optimizer = torch.optim.Adam(my\_model.parameters(), lr=learning\_rate)  num\_epochs = 20 count = 0 train\_loss\_list = [] test\_loss\_list = []  train\_accuracy\_list = [] test\_accuracy\_list = []  for epoch in range(num\_epochs):  train\_loss\_epoch = 0  test\_loss\_epoch = 0    # Training Phase  my\_model.train() # 모델을 학습 모드로 설정  train\_correct = 0  train\_total = 0    for images, labels in train\_loader:  images, labels = images.to(device), labels.to(device)  train = images.view(-1, 3, 32, 32)    outputs = my\_model(train)  loss = criterion(outputs, labels)   train\_predictions = torch.max(outputs, 1)[1]  train\_correct += (train\_predictions == labels).sum().item()  train\_total += labels.size(0)    optimizer.zero\_grad()  loss.backward()  optimizer.step()   train\_loss\_epoch += loss.item() # 미니배치 손실 합산   # Validation Phase  my\_model.eval() # 모델을 평가 모드로 설정 (Dropout 등 비활성화)  test\_correct = 0  test\_total = 0    with torch.no\_grad(): # 기울기 업데이트 비활성화 (메모리 절약)  for images, labels in test\_loader:  images, labels = images.to(device), labels.to(device)  test = images.view(-1, 3, 32, 32)  outputs = my\_model(test)    loss = criterion(outputs, labels)  test\_loss\_epoch += loss.item() # 미니배치 손실 합산    test\_predictions = torch.max(outputs, 1)[1]  test\_correct += (test\_predictions == labels).sum().item()  test\_total += labels.size(0)   train\_accuracy = train\_correct \* 100 / train\_total  test\_accuracy = test\_correct \* 100 / test\_total   train\_loss\_list.append(train\_loss\_epoch / len(train\_loader)) # 평균 손실 저장  test\_loss\_list.append(test\_loss\_epoch / len(test\_loader)) # 평균 손실 저장   train\_accuracy\_list.append(train\_accuracy)  test\_accuracy\_list.append(test\_accuracy)   print(f"Epoch {epoch+1}, Train\_Accuracy: {train\_accuracy}%, Test\_Accuracy: {test\_accuracy}%") |
| --- |



다음으로 모든 layer에 대해서 학습을 다시 진행해 보기 위해 **새로운 모델 my\_model\_2를 정의**하였다. 전체 레이어 모두에 대해 학습을 진행하기 위해 이번에는 VGG-16넷에 대해서도 param.requires\_grad = True로 설정하였다. 이에 대한 코드는 다음과 같다.

| from torchvision import models  vgg16 = models.vgg16(pretrained=True)  class MyVGG16Net\_2(nn.Module):  def \_\_init\_\_(self):  super(MyVGG16Net\_2, self).\_\_init\_\_()  # 기존의 VGG16 모델을 base model로써 사용  base\_model = models.vgg16(pretrained=True)  self.features = base\_model.features   # 새로운 classifier의 정의  self.fc = nn.Sequential(  nn.Linear(512 \* 1 \* 1, 4096), # MaxPooling을 5번 거쳤으므로 이미지의 가로, 세로 사이즈가 2^5만큼 작아지게 된다  nn.ReLU(),  nn.Linear(4096, 500),  nn.ReLU(),  nn.Linear(500, 3)  )   def forward(self, x):  x = self.features(x)  x = x.view(x.size(0), -1)  x = self.fc(x)  return x  my\_model\_2 = MyVGG16Net\_2() my\_model\_2.to(device)  for param in my\_model\_2.parameters():  param.requires\_grad = True    for param in my\_model\_2.fc.parameters():  param.requires\_grad = True |
| --- |

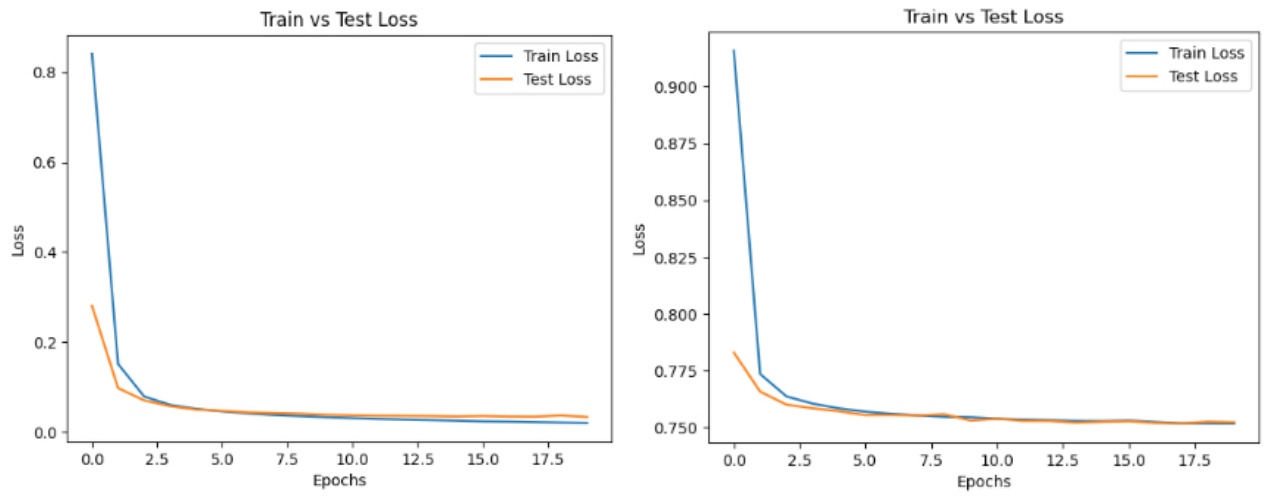
학습률, 손실함수, optimizer, 배치 크기는 위와 동일하게 하여 학습을 진행했다. 최종적으로 Train dataset에 대한 Accuracy는 99.85%, Test dataset에 대한 Accuracy는 95.33%임을 확인할 수 있었다. 코드는 위와 동일하다.



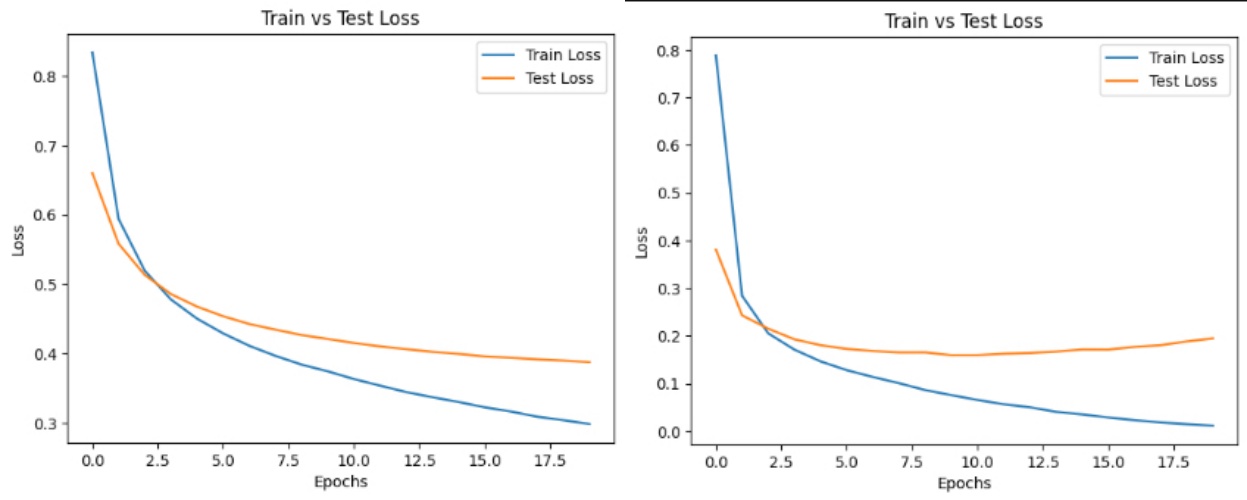
**III. 결론**

1. 손실 그래프

아래의 왼쪽 그래프는 Fashion MNIST 1번 문제, 오른쪽 그래프는 Fashion MNIST 2번 문제의 Epoch-Loss를 나타낸 그래프이다. Train Loss와 Test Loss 모두 Loss가 잘 떨어지므로 학습이 잘 이루어지고 있다고 말할 수 있다.

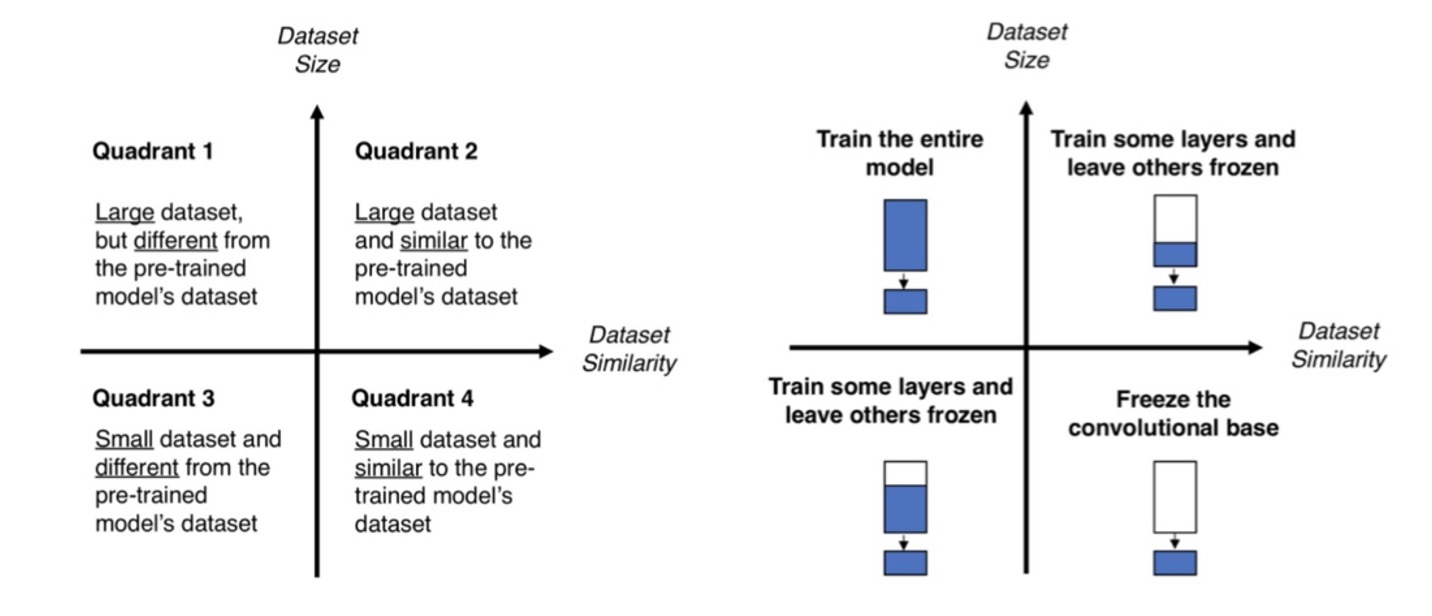


아래 그림은 CIFAR10-dataset에 대해 왼쪽 그래프는 새로 추가된 layer에 대해서만 학습한 경우, 오른쪽 그래프는 전체 layer에 대해서 학습된 경우를 나타낸 그래프이다. 새로 추가된 layer에 대해서만 학습한 경우 Train Loss와 Test Loss 모두 잘 떨어지므로 학습이 잘 이루어지고 있다고 말할 수 있으며, 전체 layer에 대해서 학습한 경우 Test Loss가 잘 떨어지다가 올라가는 것을 보아 과적합이 일어났다고 말할 수 있다.



2. 의문점

보통 전이 학습을 할 때 내가 가진 데이터 셋의 양이 적으면서 pre-trained된 모델의 데이터셋과 유사하다면 convolutional base를 얼리는 것이, 내가 가진 데이터 셋의 양이 많으면서 pre-trained된 모델의 데이터셋과 차이가 난다면 전체 모델에 대해 학습을 진행하는 것이 좋다고 한다. 이번 과제의 경우 **torchvision의 dataset은 ImageNet의 방대한 데이터 셋을 기반으로 train된 dataset**이기 때문에 상대적으로 내가 가진 데이터 셋의 양은 매우 적다고 할 수 있다. 그리고 **ImageNet Dataset과 CIFAR-10 데이터 셋은 상당히 유사하다고 생각**하여 convolutional network를 얼리는 것이 성능이 더 좋게 나올 것이라고 생각했으나, 결과가 정 반대를 보여주고 있어서 의문이였다.



3. 전체 코드

1. Fashion-MNIST (1)

| import numpy as np import matplotlib.pyplot as plt  import torch import torch.nn as nn from torch.autograd import Variable import torch.nn.functional as F  import torchvision import torchvision.transforms as transforms from torch.utils.data import Dataset, DataLoader device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu") print(device) # GPU를 사용할 수 있는지 확인  from torchvision import datasets, transforms  transform = transforms.Compose([  transforms.ToTensor(),  transforms.Normalize(mean=[0.5,], std=[0.5,]) ])  train\_dataset = torchvision.datasets.FashionMNIST("FashionMNIST/", download=True, train=True, transform=transform) test\_dataset = torchvision.datasets.FashionMNIST("FashionMNIST/", download=True, train=False, transform=transform)  target\_labels = [1, 4, 7, 8]  train\_indices = [idx for idx, (img, label) in enumerate(train\_dataset) if label in target\_labels] test\_indices = [idx for idx, (img, label) in enumerate(test\_dataset) if label in target\_labels]  from torch.utils.data import Subset  train\_subset\_dataset = Subset(train\_dataset, train\_indices) test\_subset\_dataset = Subset(test\_dataset, test\_indices)  class RemappedLabelDataset(Dataset):  def \_\_init\_\_(self, subset\_dataset, original\_label\_map, new\_label\_map):  self.subset\_dataset = subset\_dataset  self.original\_label\_map = original\_label\_map  self.new\_label\_map = new\_label\_map   def \_\_len\_\_(self):  return len(self.subset\_dataset)   def \_\_getitem\_\_(self, idx):  image, label = self.subset\_dataset[idx]  class\_name = self.original\_label\_map[label]  new\_label = self.new\_label\_map[class\_name]  return image, new\_label  original\_labels\_map = {0 : 'T-Shirt', 1 : 'Trouser', 2 : 'Pullover', 3 : 'Dress', 4 : 'Coat', 5 : 'Sandal', 6 : 'Shirt',  7 : 'Sneaker', 8 : 'Bag', 9 : 'Ankle Boot'}  my\_labels\_map = {'Trouser': 0, 'Coat': 1, 'Sneaker': 2, 'Bag': 3}  train\_subset\_dataset = RemappedLabelDataset(train\_subset\_dataset, original\_labels\_map, my\_labels\_map) test\_subset\_dataset = RemappedLabelDataset(test\_subset\_dataset, original\_labels\_map, my\_labels\_map)  train\_loader = torch.utils.data.DataLoader(train\_subset\_dataset,   batch\_size=200, shuffle = True) test\_loader = torch.utils.data.DataLoader(test\_subset\_dataset,  batch\_size=200)  fig = plt.figure(figsize=(8,8)); columns = 4; rows = 5;  inv\_my\_labels\_map = {v: k for k, v in my\_labels\_map.items()}  # 0 : Trouser, 1 : Coat, 2 : Sneaker, 3 : Bag로 잘 구성된 것을 확인할 수 있다 for i in range(1, columns\*rows +1):  img\_xy = np.random.randint(len(train\_subset\_dataset));  img, label = train\_subset\_dataset[img\_xy]  class\_name = inv\_my\_labels\_map[label]   fig.add\_subplot(rows, columns, i)  plt.axis('off')  plt.imshow(img[0, :, :], cmap='grey')  plt.title(f'{label} : {class\_name}') plt.show()  class FashionDNN(nn.Module):  def \_\_init\_\_(self):  super(FashionDNN,self).\_\_init\_\_()  self.fc1 = nn.Linear(in\_features=784,out\_features=128)  self.fc2 = nn.Linear(in\_features=128,out\_features=64)  self.fc3 = nn.Linear(in\_features=64,out\_features=32)  self.fc4 = nn.Linear(in\_features=32,out\_features=4)   def forward(self,input\_data):  out = input\_data.view(-1, 784)  out = F.relu(self.fc1(out))  out = F.relu(self.fc2(out))  out = F.relu(self.fc3(out))  out = self.fc4(out)  return out  learning\_rate = 1e-4; model = FashionDNN(); model.to(device)  criterion = nn.CrossEntropyLoss(); optimizer = torch.optim.Adam(model.parameters(), lr=learning\_rate);  print(model) print(optimizer)  num\_epochs = 20 count = 0 train\_loss\_list = [] test\_loss\_list = []  train\_accuracy\_list = [] test\_accuracy\_list = []  for epoch in range(num\_epochs):  train\_loss\_epoch = 0  test\_loss\_epoch = 0    # Training Phase  model.train() # 모델을 학습 모드로 설정  train\_correct = 0  train\_total = 0    for images, labels in train\_loader:  images, labels = images.to(device), labels.to(device)  train = images.view(-1, 1, 28, 28)    outputs = model(train)  loss = criterion(outputs, labels)   train\_predictions = torch.max(outputs, 1)[1]  train\_correct += (train\_predictions == labels).sum().item()  train\_total += labels.size(0)    optimizer.zero\_grad()  loss.backward()  optimizer.step()   train\_loss\_epoch += loss.item() # 미니배치 손실 합산   # Validation Phase  model.eval() # 모델을 평가 모드로 설정 (Dropout 등 비활성화)  test\_correct = 0  test\_total = 0    with torch.no\_grad(): # 기울기 업데이트 비활성화 (메모리 절약)  for images, labels in test\_loader:  images, labels = images.to(device), labels.to(device)  test = images.view(-1, 1, 28, 28)  outputs = model(test)    loss = criterion(outputs, labels)  test\_loss\_epoch += loss.item() # 미니배치 손실 합산    test\_predictions = torch.max(outputs, 1)[1]  test\_correct += (test\_predictions == labels).sum().item()  test\_total += labels.size(0)   train\_accuracy = train\_correct \* 100 / train\_total  test\_accuracy = test\_correct \* 100 / test\_total   train\_loss\_list.append(train\_loss\_epoch / len(train\_loader)) # 평균 손실 저장  test\_loss\_list.append(test\_loss\_epoch / len(test\_loader)) # 평균 손실 저장   train\_accuracy\_list.append(train\_accuracy)  test\_accuracy\_list.append(test\_accuracy)   print(f"Epoch {epoch+1}, Train\_Accuracy: {train\_accuracy}%, Test\_Accuracy: {test\_accuracy}%")  # plot loss graph  plt.plot(range(num\_epochs), train\_loss\_list, label="Train Loss")  plt.plot(range(num\_epochs), test\_loss\_list, label="Test Loss") plt.xlabel("Epochs") plt.ylabel("Loss") plt.title("Train vs Test Loss") plt.legend() plt.show()  # 훈련된 모델의 테스트를 위한 함수의 작성  def test\_model(image) :  model.eval()  with torch.no\_grad() :  image = image.to(device)  test = image.view(-1, 1, 28, 28)  output = model(test)  predicted\_label = torch.argmax(output, 1).item()  predicted\_name = inv\_my\_labels\_map[predicted\_label]   image = image.to('cpu')  plt.imshow(image[0, :, :], cmap='grey')  plt.title(f'{predicted\_label} : {predicted\_name}')  img0, label0 = test\_subset\_dataset[0] test\_model(img0)   img1, label1 = test\_subset\_dataset[3] test\_model(img1)   img2, label2 = test\_subset\_dataset[4] test\_model(img2)   img3, label3 = test\_subset\_dataset[10] test\_model(img3) |
| --- |

1. Fashion-MNIST(2)

| import numpy as np import matplotlib.pyplot as plt  import torch import torch.nn as nn from torch.autograd import Variable import torch.nn.functional as F  import torchvision import torchvision.transforms as transforms from torch.utils.data import Dataset, DataLoader  device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu") print(device) # GPU를 사용할 수 있는지 확인  from torchvision import datasets, transforms  transform = transforms.Compose([  transforms.ToTensor(),  transforms.Normalize(mean=[0.5,], std=[0.5,]) ])  train\_dataset = torchvision.datasets.FashionMNIST("FashionMNIST/", download=True, train=True, transform=transform) test\_dataset = torchvision.datasets.FashionMNIST("FashionMNIST/", download=True, train=False, transform=transform)  target\_labels = [1, 4, 7, 8]  train\_indices = [idx for idx, (img, label) in enumerate(train\_dataset) if label in target\_labels] test\_indices = [idx for idx, (img, label) in enumerate(test\_dataset) if label in target\_labels]  from torch.utils.data import Subset  train\_subset\_dataset = Subset(train\_dataset, train\_indices) test\_subset\_dataset = Subset(test\_dataset, test\_indices)  class RemappedLabelDataset(Dataset):  def \_\_init\_\_(self, subset\_dataset, original\_label\_map, new\_label\_map):  self.subset\_dataset = subset\_dataset  self.original\_label\_map = original\_label\_map  self.new\_label\_map = new\_label\_map   def \_\_len\_\_(self):  return len(self.subset\_dataset)   def \_\_getitem\_\_(self, idx):  image, label = self.subset\_dataset[idx]  class\_name = self.original\_label\_map[label]  new\_label = self.new\_label\_map[class\_name]  return image, new\_label  original\_labels\_map = {0 : 'T-Shirt', 1 : 'Trouser', 2 : 'Pullover', 3 : 'Dress', 4 : 'Coat', 5 : 'Sandal', 6 : 'Shirt',  7 : 'Sneaker', 8 : 'Bag', 9 : 'Ankle Boot'}  my\_labels\_map = {'Trouser': 0, 'Coat': 1, 'Sneaker': 2, 'Bag': 3}  train\_subset\_dataset = RemappedLabelDataset(train\_subset\_dataset, original\_labels\_map, my\_labels\_map) test\_subset\_dataset = RemappedLabelDataset(test\_subset\_dataset, original\_labels\_map, my\_labels\_map)  train\_loader = torch.utils.data.DataLoader(train\_subset\_dataset,   batch\_size=200, shuffle = True) test\_loader = torch.utils.data.DataLoader(test\_subset\_dataset,  batch\_size=200)  fig = plt.figure(figsize=(8,8)); columns = 4; rows = 5;  inv\_my\_labels\_map = {v: k for k, v in my\_labels\_map.items()}  # 0 : Trouser, 1 : Coat, 2 : Sneaker, 3 : Bag로 잘 구성된 것을 확인할 수 있다 for i in range(1, columns\*rows +1):  img\_xy = np.random.randint(len(train\_subset\_dataset));  img, label = train\_subset\_dataset[img\_xy]  class\_name = inv\_my\_labels\_map[label]   fig.add\_subplot(rows, columns, i)  plt.axis('off')  plt.imshow(img[0, :, :], cmap='grey')  plt.title(f'{label} : {class\_name}') plt.show()  class Lenet\_5(nn.Module):  def \_\_init\_\_(self):  super(Lenet\_5, self).\_\_init\_\_()  self.layer1 = nn.Sequential(  nn.Conv2d(in\_channels=1, out\_channels=20, kernel\_size=5, padding=2, stride=1),  nn.ReLU(),  nn.MaxPool2d(kernel\_size=2, stride=2)  )  self.layer2 = nn.Sequential(  nn.Conv2d(in\_channels=20, out\_channels=50, kernel\_size=5, padding=2, stride=1),  nn.ReLU(),  nn.MaxPool2d(kernel\_size=2, stride=2)  )  self.fc1 = nn.Linear(in\_features=2450, out\_features=500)  self.fc2 = nn.Linear(in\_features=500, out\_features=4)    def forward(self,input\_data):  out = self.layer1(input\_data)  out = self.layer2(out)  out = out.view(out.size(0), -1)  out = self.fc1(out)  out = F.relu(out)  out = self.fc2(out)  out = F.softmax(out, dim=1)  return out  learning\_rate = 1e-4; model = Lenet\_5(); model.to(device)  criterion = nn.CrossEntropyLoss() optimizer = torch.optim.Adam(model.parameters(), lr=learning\_rate);  print(model) print(optimizer)  num\_epochs = 20 count = 0 train\_loss\_list = [] test\_loss\_list = []  train\_accuracy\_list = [] test\_accuracy\_list = []  for epoch in range(num\_epochs):  train\_loss\_epoch = 0  test\_loss\_epoch = 0    # Training Phase  model.train() # 모델을 학습 모드로 설정  train\_correct = 0  train\_total = 0    for images, labels in train\_loader:  images, labels = images.to(device), labels.to(device)  train = images.view(-1, 1, 28, 28)    outputs = model(train)  loss = criterion(outputs, labels)   train\_predictions = torch.max(outputs, 1)[1]  train\_correct += (train\_predictions == labels).sum().item()  train\_total += labels.size(0)    optimizer.zero\_grad()  loss.backward()  optimizer.step()   train\_loss\_epoch += loss.item() # 미니배치 손실 합산   # Validation Phase  model.eval() # 모델을 평가 모드로 설정 (Dropout 등 비활성화)  test\_correct = 0  test\_total = 0    with torch.no\_grad(): # 기울기 업데이트 비활성화 (메모리 절약)  for images, labels in test\_loader:  images, labels = images.to(device), labels.to(device)  test = images.view(-1, 1, 28, 28)  outputs = model(test)    loss = criterion(outputs, labels)  test\_loss\_epoch += loss.item() # 미니배치 손실 합산    test\_predictions = torch.max(outputs, 1)[1]  test\_correct += (test\_predictions == labels).sum().item()  test\_total += labels.size(0)   train\_accuracy = train\_correct \* 100 / train\_total  test\_accuracy = test\_correct \* 100 / test\_total   train\_loss\_list.append(train\_loss\_epoch / len(train\_loader)) # 평균 손실 저장  test\_loss\_list.append(test\_loss\_epoch / len(test\_loader)) # 평균 손실 저장   train\_accuracy\_list.append(train\_accuracy)  test\_accuracy\_list.append(test\_accuracy)   print(f"Epoch {epoch+1}, Train\_Accuracy: {train\_accuracy}%, Test\_Accuracy: {test\_accuracy}%")  # plot loss graph  plt.plot(range(num\_epochs), train\_loss\_list, label="Train Loss") plt.plot(range(num\_epochs), test\_loss\_list, label="Test Loss") plt.xlabel("Epochs") plt.ylabel("Loss") plt.title("Train vs Test Loss") plt.legend() plt.show()  # 훈련된 모델의 테스트를 위한 함수의 작성  def test\_model(image) :  model.eval()  with torch.no\_grad() :  image = image.to(device)  test = image.view(-1, 1, 28, 28)  output = model(test)  predicted\_label = torch.argmax(output, 1).item()  predicted\_name = inv\_my\_labels\_map[predicted\_label]   image = image.to('cpu')  plt.imshow(image[0, :, :], cmap='grey')  plt.title(f'{predicted\_label} : {predicted\_name}')  img0, label0 = test\_subset\_dataset[0] test\_model(img0)  img1, label1 = test\_subset\_dataset[3] test\_model(img1)   img2, label2 = test\_subset\_dataset[4] test\_model(img2)   img3, label3 = test\_subset\_dataset[10] test\_model(img3) |
| --- |

1. CIFAR-10

| import numpy as np import matplotlib.pyplot as plt  import torch import torch.nn as nn from torch.autograd import Variable import torch.nn.functional as F  import torchvision import torchvision.transforms as transforms from torch.utils.data import Dataset, DataLoader  from torchvision import datasets, transforms  device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")  print(device) # GPU를 사용할 수 있는지 확인  transform = transforms.Compose([  transforms.ToTensor(),  transforms.Normalize(mean=[0.5,], std=[0.5,]) ])  train\_dataset = torchvision.datasets.CIFAR10("CIFAR10/", download=True, train=True, transform=transform) test\_dataset = torchvision.datasets.CIFAR10("CIFAR10/", download=True, train=False, transform=transform)  target\_labels = [0, 1, 8]  train\_indices = [idx for idx, (img, label) in enumerate(train\_dataset) if label in target\_labels] test\_indices = [idx for idx, (img, label) in enumerate(test\_dataset) if label in target\_labels]  from torch.utils.data import Subset  train\_subset\_dataset = Subset(train\_dataset, train\_indices) test\_subset\_dataset = Subset(test\_dataset, test\_indices)  class RemappedLabelDataset(Dataset):  def \_\_init\_\_(self, subset\_dataset, original\_label\_map, new\_label\_map):  self.subset\_dataset = subset\_dataset  self.original\_label\_map = original\_label\_map  self.new\_label\_map = new\_label\_map   def \_\_len\_\_(self):  return len(self.subset\_dataset)   def \_\_getitem\_\_(self, idx):  image, label = self.subset\_dataset[idx]  class\_name = self.original\_label\_map[label]  new\_label = self.new\_label\_map[class\_name]  return image, new\_label  original\_labels\_map = {0 : 'airplane', 1 : 'automobile', 2 : 'bird', 3 : 'cat', 4 : 'deer', 5 : 'dog', 6 : 'frog',  7 : 'horse', 8 : 'ship', 9 : 'truck'}  my\_labels\_map = {'airplane': 0, 'automobile': 1, 'ship': 2}  train\_subset\_dataset = RemappedLabelDataset(train\_subset\_dataset, original\_labels\_map, my\_labels\_map) test\_subset\_dataset = RemappedLabelDataset(test\_subset\_dataset, original\_labels\_map, my\_labels\_map)  train\_loader = torch.utils.data.DataLoader(train\_subset\_dataset,   batch\_size=200, shuffle = True) test\_loader = torch.utils.data.DataLoader(test\_subset\_dataset,  batch\_size=200)  fig = plt.figure(figsize=(8,8)); columns = 4; rows = 5;  inv\_my\_labels\_map = {v: k for k, v in my\_labels\_map.items()}  # 0 : Trouser, 1 : Coat, 2 : Sneaker, 3 : Bag로 잘 구성된 것을 확인할 수 있다 for i in range(1, columns\*rows +1):  img\_xy = np.random.randint(len(train\_subset\_dataset));  img, label = train\_subset\_dataset[img\_xy]  class\_name = inv\_my\_labels\_map[label]   fig.add\_subplot(rows, columns, i)  plt.axis('off')  plt.imshow(img[0, :, :], cmap='gray')  plt.title(f'{label} : {class\_name}') plt.show()  # 1. 새로 추가된 fc1 레이어에 대해서만 추가 학습을 진행  from torchvision import models  vgg16 = models.vgg16(pretrained=True)  class MyVGG16Net(nn.Module):  def \_\_init\_\_(self):  super(MyVGG16Net, self).\_\_init\_\_()  # 기존의 VGG16 모델을 base model로써 사용  base\_model = models.vgg16(pretrained=True)  self.features = base\_model.features   # 새로운 classifier의 정의  self.fc = nn.Sequential(  nn.Linear(512 \* 1 \* 1, 4096), # MaxPooling을 5번 거쳤으므로 이미지의 가로, 세로 사이즈가 2^5만큼 작아지게 된다  nn.BatchNorm1d(4096),  nn.ReLU(),  nn.Linear(4096, 500),  nn.BatchNorm1d(500),  nn.ReLU(),  nn.Linear(500,3)  )   def forward(self, x):  x = self.features(x)  x = x.view(x.size(0), -1)  x = self.fc(x)  return x  my\_model = MyVGG16Net() my\_model.to(device)  for param in my\_model.parameters():  param.requires\_grad = False # frozen   for param in my\_model.fc.parameters():  param.requires\_grad = True # 마지막 레이어는 autograd 활성화  print(my\_model)  learning\_rate = 4 \* 1e-06  criterion = nn.CrossEntropyLoss() optimizer = torch.optim.Adam(my\_model.parameters(), lr=learning\_rate)  num\_epochs = 20 count = 0 train\_loss\_list = [] test\_loss\_list = []  train\_accuracy\_list = [] test\_accuracy\_list = []  for epoch in range(num\_epochs):  train\_loss\_epoch = 0  test\_loss\_epoch = 0    # Training Phase  my\_model.train() # 모델을 학습 모드로 설정  train\_correct = 0  train\_total = 0    for images, labels in train\_loader:  images, labels = images.to(device), labels.to(device)  train = images.view(-1, 3, 32, 32)    outputs = my\_model(train)  loss = criterion(outputs, labels)   train\_predictions = torch.max(outputs, 1)[1]  train\_correct += (train\_predictions == labels).sum().item()  train\_total += labels.size(0)    optimizer.zero\_grad()  loss.backward()  optimizer.step()   train\_loss\_epoch += loss.item() # 미니배치 손실 합산   # Validation Phase  my\_model.eval() # 모델을 평가 모드로 설정 (Dropout 등 비활성화)  test\_correct = 0  test\_total = 0    with torch.no\_grad(): # 기울기 업데이트 비활성화 (메모리 절약)  for images, labels in test\_loader:  images, labels = images.to(device), labels.to(device)  test = images.view(-1, 3, 32, 32)  outputs = my\_model(test)    loss = criterion(outputs, labels)  test\_loss\_epoch += loss.item() # 미니배치 손실 합산    test\_predictions = torch.max(outputs, 1)[1]  test\_correct += (test\_predictions == labels).sum().item()  test\_total += labels.size(0)   train\_accuracy = train\_correct \* 100 / train\_total  test\_accuracy = test\_correct \* 100 / test\_total   train\_loss\_list.append(train\_loss\_epoch / len(train\_loader)) # 평균 손실 저장  test\_loss\_list.append(test\_loss\_epoch / len(test\_loader)) # 평균 손실 저장   train\_accuracy\_list.append(train\_accuracy)  test\_accuracy\_list.append(test\_accuracy)   print(f"Epoch {epoch+1}, Train\_Accuracy: {train\_accuracy}%, Test\_Accuracy: {test\_accuracy}%")  # plot loss graph  plt.plot(range(num\_epochs), train\_loss\_list, label="Train Loss") plt.plot(range(num\_epochs), test\_loss\_list, label="Test Loss") plt.xlabel("Epochs") plt.ylabel("Loss") plt.title("Train vs Test Loss") plt.legend() plt.show()  # plot accuracy graph  plt.plot(range(num\_epochs), train\_accuracy\_list, label="Train Accuracy") plt.plot(range(num\_epochs), test\_accuracy\_list, label="Test Accuracy") plt.xlabel("Epochs") plt.ylabel("Loss") plt.title("Train vs Test Accuracy") plt.legend() plt.show()  # 2. 전체 Layer에 대해서 학습을 진행  from torchvision import models  vgg16 = models.vgg16(pretrained=True)  class MyVGG16Net\_2(nn.Module):  def \_\_init\_\_(self):  super(MyVGG16Net\_2, self).\_\_init\_\_()  # 기존의 VGG16 모델을 base model로써 사용  base\_model = models.vgg16(pretrained=True)  self.features = base\_model.features   # 새로운 classifier의 정의  self.fc = nn.Sequential(  nn.Linear(512 \* 1 \* 1, 4096), # MaxPooling을 5번 거쳤으므로 이미지의 가로, 세로 사이즈가 2^5만큼 작아지게 된다  nn.ReLU(),  nn.Linear(4096, 500),  nn.ReLU(),  nn.Linear(500, 3)  )   def forward(self, x):  x = self.features(x)  x = x.view(x.size(0), -1)  x = self.fc(x)  return x  my\_model\_2 = MyVGG16Net\_2() my\_model\_2.to(device)  for param in my\_model\_2.parameters():  param.requires\_grad = True    for param in my\_model\_2.fc.parameters():  param.requires\_grad = True   print(my\_model\_2)  learning\_rate = 4 \* 1e-06  criterion = nn.CrossEntropyLoss() optimizer = torch.optim.Adam(my\_model\_2.parameters(), lr=learning\_rate)  num\_epochs = 20 count = 0 train\_loss\_list = [] test\_loss\_list = []  train\_accuracy\_list = [] test\_accuracy\_list = []  for epoch in range(num\_epochs):  train\_loss\_epoch = 0  test\_loss\_epoch = 0    # Training Phase  my\_model\_2.train() # 모델을 학습 모드로 설정  train\_correct = 0  train\_total = 0    for images, labels in train\_loader:  images, labels = images.to(device), labels.to(device)  train = images.view(-1, 3, 32, 32)    outputs = my\_model\_2(train)  loss = criterion(outputs, labels)   train\_predictions = torch.max(outputs, 1)[1]  train\_correct += (train\_predictions == labels).sum().item()  train\_total += labels.size(0)    optimizer.zero\_grad()  loss.backward()  optimizer.step()   train\_loss\_epoch += loss.item() # 미니배치 손실 합산   # Validation Phase  my\_model\_2.eval() # 모델을 평가 모드로 설정 (Dropout 등 비활성화)  test\_correct = 0  test\_total = 0    with torch.no\_grad(): # 기울기 업데이트 비활성화 (메모리 절약)  for images, labels in test\_loader:  images, labels = images.to(device), labels.to(device)  test = images.view(-1, 3, 32, 32)  outputs = my\_model\_2(test)    loss = criterion(outputs, labels)  test\_loss\_epoch += loss.item() # 미니배치 손실 합산    test\_predictions = torch.max(outputs, 1)[1]  test\_correct += (test\_predictions == labels).sum().item()  test\_total += labels.size(0)   train\_accuracy = train\_correct \* 100 / train\_total  test\_accuracy = test\_correct \* 100 / test\_total   train\_loss\_list.append(train\_loss\_epoch / len(train\_loader)) # 평균 손실 저장  test\_loss\_list.append(test\_loss\_epoch / len(test\_loader)) # 평균 손실 저장   train\_accuracy\_list.append(train\_accuracy)  test\_accuracy\_list.append(test\_accuracy)   print(f"Epoch {epoch+1}, Train\_Accuracy: {train\_accuracy}%, Test\_Accuracy: {test\_accuracy}%")  # plot loss graph  plt.plot(range(num\_epochs), train\_loss\_list, label="Train Loss") plt.plot(range(num\_epochs), test\_loss\_list, label="Test Loss") plt.xlabel("Epochs") plt.ylabel("Loss") plt.title("Train vs Test Loss") plt.legend() plt.show()  # plot accuracy graph  plt.plot(range(num\_epochs), train\_accuracy\_list, label="Train Accuracy") plt.plot(range(num\_epochs), test\_accuracy\_list, label="Test Accuracy") plt.xlabel("Epochs") plt.ylabel("Loss") plt.title("Train vs Test Accuracy") plt.legend() plt.show() |
| --- |