


```

1  import os
2  import torch
3  import torch.nn as nn
4  import torch.optim as optim
5  from torchvision import datasets, transforms, models
6  from torch.utils.data import DataLoader
7  from tqdm import tqdm
8  from torchvision.models import ResNet50_Weights, resnet50
9
10 # 경로 설정
11 train_dir = './train' # 학습 데이터 경로
12 test_dir = './test'   # 테스트 데이터 경로
13
14 # 데이터 전처리
15 transform_train = transforms.Compose([
16     transforms.Resize((224, 224)), # ResNet의 입력 크기와 일치
17     transforms.RandomHorizontalFlip(),
18     transforms.RandomRotation(20),
19     transforms.ToTensor(),
20     transforms.Normalize(mean=[0.485, 0.456, 0.406], # ResNet 사전 학습된 모델의 정규화 값
21                          std=[0.229, 0.224, 0.225])
22 ])
23
24 transform_test = transforms.Compose([
25     transforms.Resize((224, 224)), # ResNet의 입력 크기와 일치
26     transforms.ToTensor(),
27     transforms.Normalize(mean=[0.485, 0.456, 0.406], # ResNet 사전 학습된 모델의 정규화 값
28                          std=[0.229, 0.224, 0.225])
29 ])
30
31 # 데이터셋 및 데이터로더
32 train_dataset = datasets.ImageFolder(root=train_dir, transform=transform_train)
33 test_dataset = datasets.ImageFolder(root=test_dir, transform=transform_test)
34
35 train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, num_workers=4)
36 test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False, num_workers=4)
37
38 print(f"훈련셋 크기: {len(train_dataset)}")
39 print(f"테스트셋 크기: {len(test_dataset)}")

```

1. **데이터 전처리** :학습 데이터에 **랜덤 좌우 반전** 및 **랜덤 회전** 등의 증강 기법을 적용하여 모델의 일반화 성능을 높였습니다. 입력 이미지는 ResNet 모델의 요구사항에 맞게 **224x224** 크기로 조정되며, **정규화**를 수행하였습니다.
2. **데이터셋 로드**: PyTorch의 ImageFolder를 사용하여 폴더 구조를 기반으로 학습 및 테스트 데이터를 불러왔습니다.
3. **데이터로더**
 - DataLoader를 통해 데이터를 배치 단위로 나누어 모델 학습 및 평가에 효율적으로 사용될 수 있도록 준비하였습니다.



```

1  num_classes = len(train_dataset.classes)
2
3  # 사전 학습된 ResNet50 불러오기
4  model = models.resnet50(weights=ResNet50_Weights.IMAGENET1K_V1)
5
6  # 모든 파라미터 freeze (마지막 fc 제외)
7  for param in model.parameters():
8      param.requires_grad = False
9
10 # 최상위 레이어 변경 (랜덤 초기화)
11 model.fc = nn.Linear(model.fc.in_features, num_classes)
12
13 # GPU 설정
14 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
15 if torch.cuda.is_available():
16     print("CUDA is available. Using GPU for training.")
17 else:
18     print("CUDA is not available. Using CPU for training.")
19 model = model.to(device)
20

```

1. 사전 학습된 ResNet-50 모델 로드 : torchvision에서 제공하는 ResNet-50 모델을 불러오고, ImageNet으로 사전 학습된 가중치를 사용했습니다.
2. 모델의 파라미터 동결 : Transfer Learning을 위해 기존의 레이어는 학습되지 않도록 freeze 처리하였으며, 최상위 fc 레이어만 학습합니다.
3. 최상위 레이어 변경 : 데이터셋 클래스 수에 맞게 새로운 Fully Connected Layer로 교체하였습니다.
4. GPU 설정 : 학습에 GPU를 활용하기 위해 모델을 CUDA 장치로 이동시켰습니다.

```

1 criterion = nn.CrossEntropyLoss()
2
3 def train_and_test(model, train_loader, test_loader, criterion, num_epochs=30, log_file='training_log.txt'):
4     best_test_acc = 0.0 # 최고 테스트 정확도 추적
5
6     # 로그 파일 초기화 및 헤더 작성
7     with open(log_file, mode='w') as file:
8         file.write("Epoch\tTrain Loss\tTrain Acc\tTest Loss\tTest Acc\n")
9
10    for epoch in range(num_epochs):
11        print(f"\nEpoch {epoch+1}/{num_epochs}")
12        print("-" * 30)
13
14        # **학습 단계**
15        model.train() # 학습 모드
16        running_loss = 0.0
17        running_corrects = 0
18
19        # 단계에 따라 optimizer와 학습 가능한 파라미터 설정
20        if epoch == 0:
21            # 첫 번째 epoch: 마지막 레이어만 학습
22            for param in model.parameters():
23                param.requires_grad = False
24            for param in model.fc.parameters():
25                param.requires_grad = True
26            optimizer = optim.Adam(model.fc.parameters(), lr=1e-4)
27            print("첫 번째 epoch: 마지막 출력층만 학습")
28        elif epoch == 1:
29            # 두 번째 epoch부터 전체 모델을 미세 조정
30            for param in model.parameters():
31                param.requires_grad = True
32            optimizer = optim.Adam(model.parameters(), lr=1e-5)
33            print("두 번째 epoch부터 전체 모델을 미세 조정")
34
35        # 학습 단계
36        for inputs, labels in tqdm(train_loader, desc="Training"):
37            # 데이터를 CUDA 장치로 이동
38            inputs = inputs.to(device)
39            labels = labels.to(device)
40
41            optimizer.zero_grad()
42            outputs = model(inputs)
43            loss = criterion(outputs, labels)
44            _, preds = torch.max(outputs, 1)
45
46            loss.backward()
47            optimizer.step()
48
49            running_loss += loss.item() * inputs.size(0)
50            running_corrects += torch.sum(preds == labels.data)
51
52        epoch_train_loss = running_loss / len(train_loader.dataset)
53        epoch_train_acc = running_corrects.double() / len(train_loader.dataset)
54        print(f"Train Loss: {epoch_train_loss:.4f} | Train Acc: {epoch_train_acc:.4f}")
55
56        # **평가 단계**
57        model.eval() # 평가 모드
58        test_loss = 0.0
59        test_corrects = 0
60
61        with torch.no_grad():
62            for inputs, labels in tqdm(test_loader, desc="Testing"):
63                inputs = inputs.to(device)
64                labels = labels.to(device)
65
66                outputs = model(inputs)
67                loss = criterion(outputs, labels)
68                _, preds = torch.max(outputs, 1)
69
70                test_loss += loss.item() * inputs.size(0)
71                test_corrects += torch.sum(preds == labels.data)
72
73        epoch_test_loss = test_loss / len(test_loader.dataset)
74        epoch_test_acc = test_corrects.double() / len(test_loader.dataset)
75        print(f"Test Loss: {epoch_test_loss:.4f} | Test Acc: {epoch_test_acc:.4f}")
76
77        # **로그 저장**
78        with open(log_file, mode='a') as file:
79            file.write(f"{epoch+1}\t{epoch_train_loss:.4f}\t{epoch_train_acc:.4f}\t{epoch_test_loss:.4f}\t{epoch_test_acc:.4f}\n")
80
81        # **최고 테스트 정확도 모델 저장**
82        if epoch_test_acc > best_test_acc:
83            best_test_acc = epoch_test_acc
84            torch.save(model.state_dict(), 'best_finetuned_resnet.pth')
85            print("Best model saved.")
86
87        print(f"\nTraining complete. Best Test Acc: {best_test_acc:.4f}")
88
89    return model
90
91 if __name__ == '__main__':
92     # 모델 학습 및 테스트 수행
93     model = train_and_test(model, train_loader, test_loader, criterion, num_epochs=30, log_file='resnet_training_log.txt')
94

```

1. **학습 및 평가 단계** : 첫 번째 Epoch에는 **마지막 레이어만 학습**, 두 번째 Epoch 이후에는 **전체 모델을 미세 조정**합니다.
2. **Train Step** : 각 배치마다 **Loss 계산 → Backpropagation → Optimizer Step**을 수행합니다.
3. **Test Step** : `model.eval()`로 평가 모드를 설정하고, Gradient 계산을 비활성화한 채로 손실과 정확도를 측정합니다.
4. **최고 성능 모델 저장** : 테스트 정확도가 가장 높을 때 모델의 가중치를 저장합니다.
5. **손실 함수 정의** : 다중 클래스 분류 문제를 해결하기 위해 `CrossEntropyLoss`를 사용했습니다.
6. **모델 학습 및 평가 실행** : `train_and_test` 함수를 호출하여 학습과 평가를 수행하며 로그를 저장하고 최고 성능 모델을 저장합니다.