

# Assignment #3

2017706106 현민기

## 1. 개요

### Transfer Learning

-실제로 충분한 크기의 데이터셋을 갖추기는 상대적으로 드물기 때문에, (무작위 초기화를 통해) 바닥부터(from scratch) 전체 Convolutional Network를 학습 어려움

-매우 큰 데이터셋(예. 100가지 Category에 대해 120만개의 이미지가 포함된ImageNet)에서 Convolutional Network를 미리 학습(Pretrain)한 후, 이 Convolutional Network를 관심있는 작업을 위한 초기화, 고정 특징추출기로 사용

## 2. 구현 방법

```
class Mynet(nn.Module):
    def __init__(self, num_classes=2):
        super(Mynet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=5, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=5, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.fc = nn.Linear(32*56*56, num_classes)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        return out
```

```
device = 'cpu'
```

Mynet이라는  
Convolution  
network 구현

linear속의 숫자는 계  
산 후 58이라는 숫자  
가 나와서 (32\*56\*56)  
으로 나옴.

```

#Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

def train_model(model, criterion, optimizer, scheduler, num_epochs):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # 각 에폭(epoch)은 학습 단계와 검증 단계를 갖습니다.
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train() # 모델을 학습 모드로 설정
            else:
                model.eval() # 모델을 평가 모드로 설정

            running_loss = 0.0
            running_corrects = 0

            # 데이터를 반복
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # 매개변수 경사도를 0으로 설정
                optimizer.zero_grad()

                # 순전파
                # 학습 시에만 연산 기록을 추적
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                # 학습 단계인 경우 역전파 + 최적화
                if phase == 'train':
                    loss.backward()
                    optimizer.step()

            # 통계
            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds == labels.data)

        epoch_loss = running_loss / dataset_sizes[phase]
        epoch_acc = running_corrects.double() / dataset_sizes[phase]

        print('{} Loss: {:.4f} Acc: {:.4f}'.format(
            phase, epoch_loss, epoch_acc))

    # 모델을 깊은 복사(deep copy)함
    if phase == 'val' and epoch_acc > best_acc:
        best_acc = epoch_acc
        best_model_wts = copy.deepcopy(model.state_dict())

```

dataset에서 받은 개미와 벌을 학습과 검정하고, 정확도 계산하는 train\_model을 구현한다.

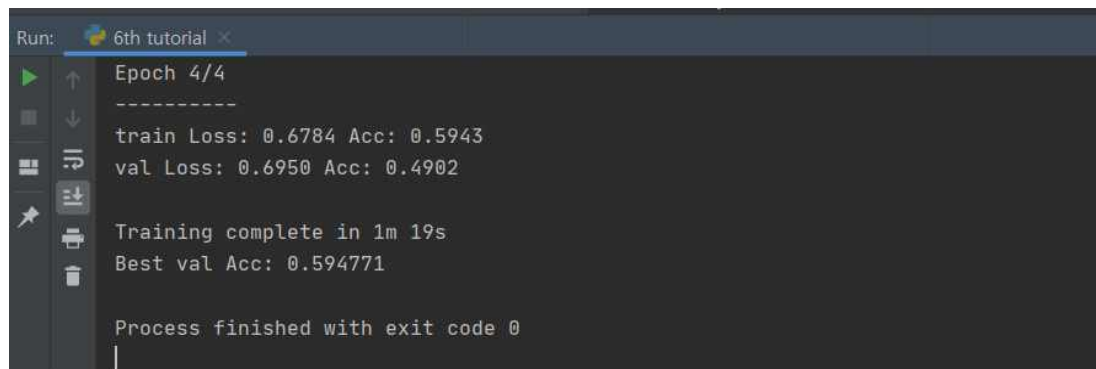
```
exp_lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)
```

```
train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs)
```

마지막 실행함.

결과

;



```
Run: 6th tutorial x
Epoch 4/4
-----
train Loss: 0.6784 Acc: 0.5943
val Loss: 0.6950 Acc: 0.4902

Training complete in 1m 19s
Best val Acc: 0.594771

Process finished with exit code 0
```