과목명	KW_VIP	-	-	담당교수	심동규 교수님
학과	전자통신공학과	학번	2017707066	이름	이성민
과제명: Assignment3					

1. 과제설명

Transfer Learning을 이용해서 hymenoptera의 데이터셋의 loss값과 정확도를 출력하는 모델을 구성합니다.

2. 주요소스코드 설명

```
    def __init__(self, num_classes=2):
        super(ConvNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer3 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.fc = nn.Linear(224*224, num_classes)
```

계층을 3개로 구성하였으며 데이터셋이 RGB형태이므로 Channel이 3이어서 3으로 설정을 하였고 MaxPooling 계층은 데이터 크기를 절반으로 나누는 역할을 하며, 이미지의 크기가 224로 설정되어 있어서 마지막 Classifier 부분에 224*224로 넣은 것을 확인할 수 있습니다.
num_classes는 개미인지 벌인지 구별하는 두 개의 범주로 나눠져 있기 때문에 2입니다.

```
    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = out.reshape(out.size(0),-1)
        out = self.fc(out)
        return out
```

forward()에 대한 메소드는 다음과 같이 설정을 했으며 layer계층 순서대로 진행을 하고 reshape로 차원을 바꿔 준 후 Fully Connected Layer인 Classifier을 지나 결과를 return하게 설정했습니다.

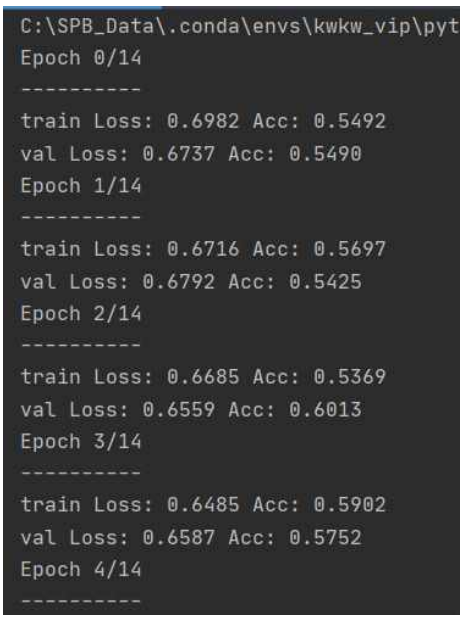
```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

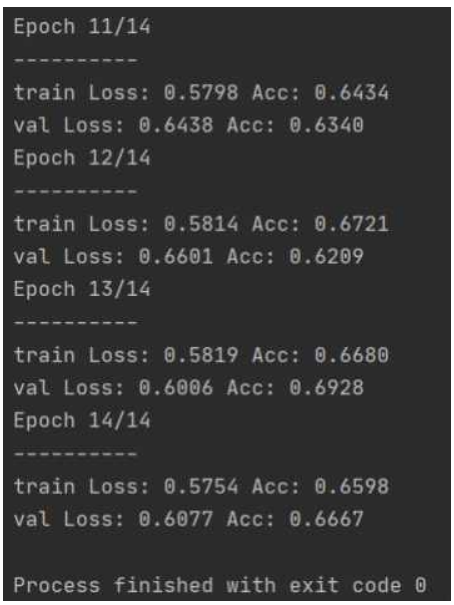
손실함수를 파악하는데 CrossEntropy 기법을 사용했으며, optimizer은 SGD로 설정했습니다.

```
for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
            else:
                model.eval()
```

train_model의 메소드 안의 코드이며, 학습과 검사를 하나의 코드로 합쳐놓아서 각 epoch마다 Training accuracy와 Test accuracy를 확인할 수 있습니다.
epoch을 크게해서 한 번 실행하고 나면, 적어도 그 epoch 범위 내에서는 최적의 epoch값을 찾을 수 있습니다.

3. 실행결과 및 설명





train의 결과에 대한 Loss값과 Acc값이 나오고 동시에 test값도 마찬가지로 나오는 것을 확인할 수 있습니다.
epoch은 15번을 돌려서 train을 시켰으며 처음보다 Loss가 약간 줄었고, accuracy가 높아진 것을 확인할 수 있습니다.

4. 전체 소스코드

```
import torch
import torch.nn as nn
import torchvision
from torchvision import datasets
import torchvision.transforms as transforms
import os


device = 'cpu'

num_epochs = 15
num_classes = 2
batch_size = 4
learning_rate = 0.001

class ConvNet(nn.Module):
    def __init__(self, num_classes=2):
        super(ConvNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer3 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.fc = nn.Linear(224*224, num_classes)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = out.reshape(out.size(0),-1)
        out = self.fc(out)
        return out

data_transforms ={
    'train' : transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val' : transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],[0.229, 0.224, 0.225])
    ])
}

data_dir = 'hymenoptera_data'
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                          data_transforms[x])
                  for x in ['train', 'val']}
dataloaders ={x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4,
                                             shuffle=True)
              for x in ['train', 'val']}

dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
class_names = image_datasets['train'].classes
```

```
model = ConvNet(num_classes).to(device)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

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

    for phase in ['train', 'val']:
        if phase == 'train':
            model.train()
        else:
            model.eval()

        running_loss = 0.0
        running_corrects = 0

        for images, labels in dataloaders[phase]:
            images = images.to(device)
            labels = labels.to(device)
            optimizer.zero_grad()

            with torch.set_grad_enabled(phase == 'train'):
                outputs = model(images)
                _, preds = torch.max(outputs, 1)
                loss = criterion(outputs, labels)

            if phase == 'train':
                loss.backward()
                optimizer.step()

            running_loss += loss.item() * images.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))
```