#Assignment4

-2017204045 장지연

개요

목표 : 다양한 방법을 통해 학습시켜보고 비교해본다. Fine-tuning 실습을 진행해본다.

구현방법

앞서 사용 했었던 bee 와 ant 데이터를 가지고 진행

1. 나의 내트워크

```
class Net(torch.nn.Module):
    def __init__(self,num_classes=2):
        super(Net, 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(28* 28 * 64, num_classes)
```

3 개의 층으로 되어있고 crossentropyloss 와 SGD 등을 사용하였다. Epochs 를 10 으로 하여 나머지 방법들의 결과를 보며 정확도를 비교하고자 한다.

2. Strategy1 .- 전체 모델을 학습

Model 은 미리 만들어져 있는 resnet18 을 사용하여 데이터 셋에 맞게 전부 새로 학습 시킨다. bee 와 ant 를 분류하는 거니까 최종 분류 개수를 2 로 맞춰준다. (model_ft.fc) 내 네트워크에서 내가 만든 계층만 다르고 나머지는 같은 형식으로 진행된다.

```
model_ft = models.resnet18(pretrained=True)

num_ftrs = model_ft.fc.in_features
# bee & ants
model_ft.fc = nn.Linear(num_ftrs, 2)

model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()
```

3. Strategy3 – convolutional base 는 고정, classifier 만 새로 학습

convolutional base 는 고정, classifier 만 새로 학습시키기 위해서 conv 부분을 고정시켜야한다. 전체 모델을 학습시키는 것과 같은 방식으로 진행하되 코드 2 줄만 추가하면 된다. Conv 고정을 위해서 Requires_grad ==False 로 설정하여 매개변수를 고정하여 backward()중에 경사도가 계산되지 않도록 해야한다.

```
model_conv = torchvision.models.resnet18(pretrained=True)
for param in model_conv.parameters():
    param.requires_grad = False #freeze
```

결과 화면

Epoch	나의 네트워크		Strategy 1		Strategy 3	
	Val loss	acc	Val loss	acc	Val loss	acc
1	0.6851	0.6013	0.4101	0.8431	0.3243	0.8301
2	0.6774	0.6078	0.3540	0.8758	0.2203	0.9216
3	0.7057	0.4575	0.2741	0.9281	0.2082	0.9281

4	0.6810	0.5425	0.3370	0.9085	0.2152	0.9412
5	0.6925	0.5359	0.2753	0.8889	0.1653	0.9477
6	0.6850	0.5294	0.3461	0.8758	0.7236	0.7712
7	0.6683	0.5948	0.3799	0.8824	0.1668	0.9477
8	0.7013	0.5163	0.3100	0.9281	0.1703	0.9608
9	0.6718	0.5882	0.2644	0.9150	0.1831	0.9608
10	0.6766	0.6078	0.2660	0.9150	0.1705	0.9608

나의 네트워크

```
val Loss: 0.6683 Acc: 0.5948
train Loss: 0.6513 Acc: 0.5902
val Loss: 0.7013 Acc: 0.5163
train Loss: 0.6382 Acc: 0.6066
val Loss: 0.6718 Acc: 0.5882
train Loss: 0.6125 Acc: 0.6516
val Loss: 0.6766 Acc: 0.6078
```

Strategy1

```
Epoch 9/9
----------
train Loss: 0.3362 Acc: 0.8730
val Loss: 0.2660 Acc: 0.9150

Training complete in 18m 42s
Best val Acc: 0.928105
```

Strategy3

```
Epoch 9/9
----------
train Loss: 0.2959 Acc: 0.8730
val Loss: 0.1705 Acc: 0.9608

Training complete in 8m 58s
Best val Acc: 0.960784
```

결과를 보면 가장 정확한 것은 strategy3으로 classifier만 학습시킨 것이었다. 나의 네트워크 정확도는 60%로 다른 것들에 비해서는 낮다. Resnet 처럼 미리 만들어진 모델들을 사용하면 결과도 좋고 층을 만드는데 큰 힘을 쏟지 않아도 되서 정말 편했다. 정확도와 데이터 수에 비해서 96%로 엄청 좋고, 전체를 Strategy1에 비해서 시간도 적게 걸려서 앞으로는 층을 직접 만드는 것 보다는 resnet을 사용하는 게 더 나을 것 같다.