Transfer Learning, 모델 평가

Transfer Learning

- 내가 풀고자 하는 문제와 비슷한 문제로 사전학습 된 모델을 이용

- AlexNet, VGG, ResNet, DenseNet, GooLeNet, …

```
26
27  base = train_baseline(model_base, train_loader, val_loader, optimizer, EPOCH)
28
29  torch.save(base, 'baseline.pt')
```

```
train Loss :0.2185, Accuracy :93.12
Completed in 0m 33s
----------------------epoch 27 ----------------------
train Loss :0.0834, Accuracy :97.38
train Loss :0.2358, Accuracy :92.21
Completed in 0m 33s
----------------------epoch 28 ----------------------
train Loss :0.0785, Accuracy :98.03
train Loss :0.2262, Accuracy :92.75
Completed in 0m 33s
----------------------epoch 29 ----------------------
train Loss :0.0662, Accuracy :98.49
train Loss :0.2155, Accuracy :93.00
Completed in 0m 33s
----------------------epoch 30 ----------------------
train Loss :0.0588, Accuracy :98.62
train Loss :0.2041, Accuracy :93.52
Completed in 0m 33s
```

Baseline model?

- 모델 성능 검증의 기준이 되는 모델

```python
model_resnet50 = train_resnet(resnet, criterion, optimizer_ft, exp_lr_scheduler, num_epochs=EPOCH)

torch.save(model_resnet50, 'resnet50.pt')
```

```
valloss : 0.0299 Acc : 0.9895
Completed in 0m 53s
-------------------- epoch 28 ----------------
learning rate : [1.0000000000000006e-11]
trainloss : 0.0136 Acc : 0.9953
valloss : 0.0308 Acc : 0.9899
Completed in 0m 53s
-------------------- epoch 29 ----------------
learning rate : [1.0000000000000006e-11]
trainloss : 0.0150 Acc : 0.9947
valloss : 0.0335 Acc : 0.9894
Completed in 0m 54s
-------------------- epoch 30 ----------------
learning rate : [1.0000000000000006e-11]
trainloss : 0.0143 Acc : 0.9952
valloss : 0.0320 Acc : 0.9909
Completed in 0m 54s
best val Acc : 0.9908624358492927
```

Learning rate

- CNN베이스의 사전학습 모델을 사용할 때, 이전에 학습한 내용들을 모두 잊어버릴 위험이 있기 때문에 작은 learning rate를 사용

Baseline model 평가를 위한 전처리

```
transform_base = transforms.Compose([transforms.Resize([64, 64]), transforms.ToTensor()])
test_base = ImageFolder(root = 'C:/Users/Administrator/Desktop/AAI/AAI_data/splitted/test'
                        , transform = transform_base)
test_loader_base = torch.utils.data.DataLoader(test_base, batch_size = BATCH_SIZE
                                               , shuffle = True, num_workers = 4)

```

Shuffle : 정답의 순서를 기억하는 것을 방지하기 위해 데이터를 섞음

Transfer Learning 모델 평가를 위한 전처리

```
transform_resNet = transforms.Compose([
    transforms.Resize([64, 64]),
    transforms.RandomCrop(52),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])
test_resNet = ImageFolder(root = 'C:/Users/Administrator/Desktop/AAI/AAI_data/splitted/test'
                          , transform = transform_resNet)
test_loader_resNet = torch.utils.data.DataLoader(test_resNet, batch_size = BATCH_SIZE
                                                 , shuffle =True, num_workers = 4)
```

Baseline model 성능 평가

```
1  baseline = torch.load('baseline.pt')
2  baseline.eval()
3  test_loss, test_accuracy = evaluate(baseline, test_loader_base)
4
5  print('baseline test acc :', test_accuracy)
```

```
baseline test acc : 93.36587808236325
```

Transfer Learning 성능 평가

```
resnet50=torch.load('resnet50.pt')
resnet50.eval()
test_loss, test_accuracy = evaluate(resnet50, test_loader_resNet)

print('ResNet test acc:  ', test_accuracy)
```

```
ResNet test acc:   98.96107147327575
```

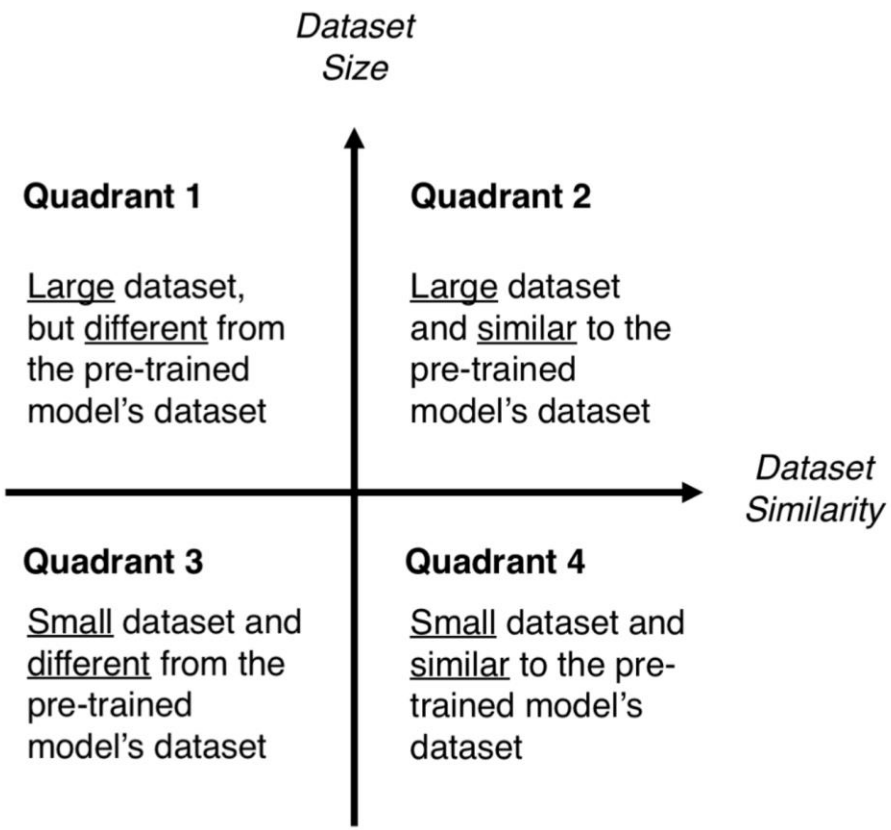
성능 평가

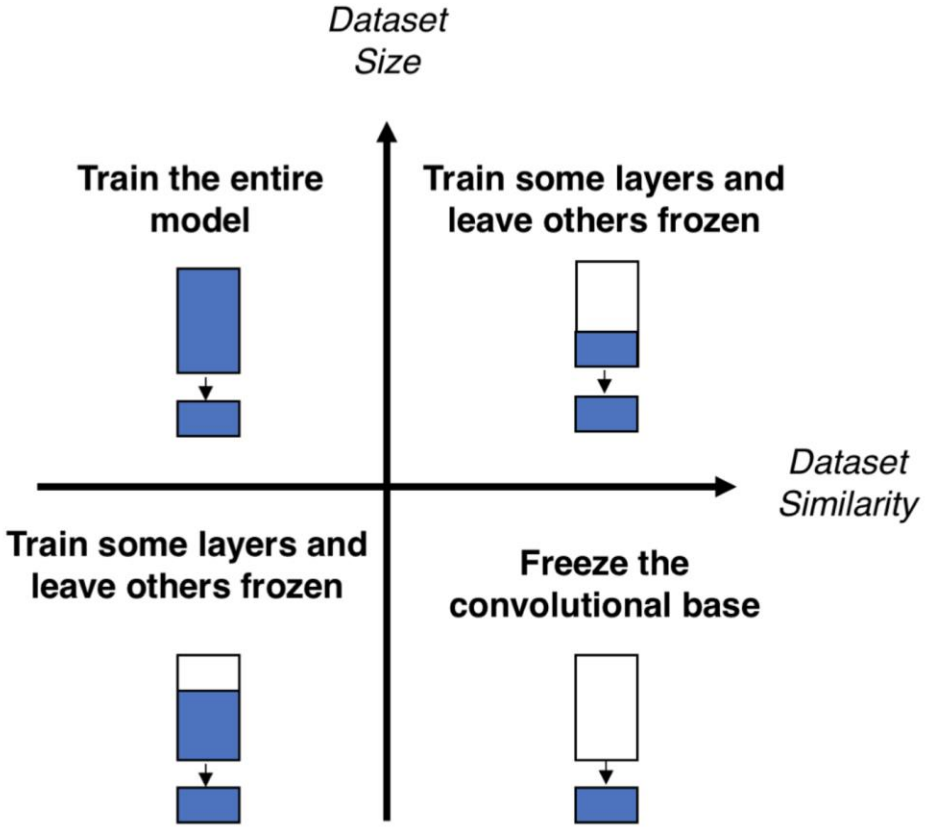
- 베이스라인 모델 : 93.3% 정확도
- Transfer Learning 모델 : 98.9% 정확도

Baseline 데이터의 수 : 약 40000개
ResNet50 데이터 수 : 약 14000000개

Baseline 초기 parameter : random
ResNet50 초기 parameter : 미리 학습해놓은 모델의 parameter

Fine-tuning 전략





Fine-tuning 전략

- # 전략 1 : 전체 모델을 새로 학습시키기

- # 전략 2 : Convolutional base의 일부분은 고정시킨 상태로, 나머지 계층과 classifier를 새로 학습시키기

- # 전략 3 : Convloutional base는 고정시키고, classifier만 새로 학습시키기

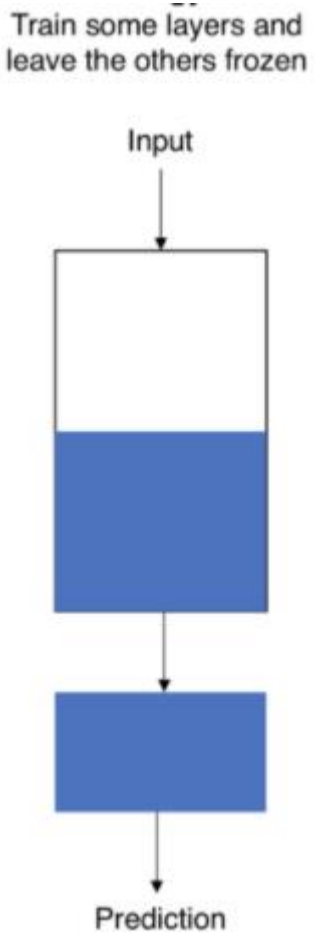
1. 크기가 크고 유사성이 작은 데이터셋일 때

- 데이터셋의 크기가 크므로 모델을 다시 처음부터 완전히 다시 학습.
- 사전 학습 모델의 구조와 매개변수를 가지고 시작하는 것은 random한 매개변수를 가지고 시작하는 것보다 유리함.

- # 전략 1

2. 크기가 크고 유사성이 높은 데이터셋일 때

- 가장 좋은 상황.
- 데이터셋이 유사하다는 이점을 이용하여 모델이 이전에 학습한 지식을 활용한다.
- #전략 1, 2, 3 다 가능하지만 효율을 위해 #전략2 사용



3. 크기가 작고 유사성도 작은 데이터셋일 때

- 가장 좋지 않은 상황.

- 너무 많은 계층을 새로 학습시키면 작은 데이터셋에 overfitting 위험성
- 너무 적은 계층을 새로 학습시키면 학습이 잘 되지 않을 것.

→ 작은 데이터셋을 보완하기 위해 data augmentation등의 기술 사용

4. 크기가 작지만 유사성은 높은 데이터셋일 때

- Classifier만 학습시킨다.

- #전략 3