심층신경망개론 Final 프로젝트

leaderboard name : zoomin_lee
2017310695 이주민

1. My Method

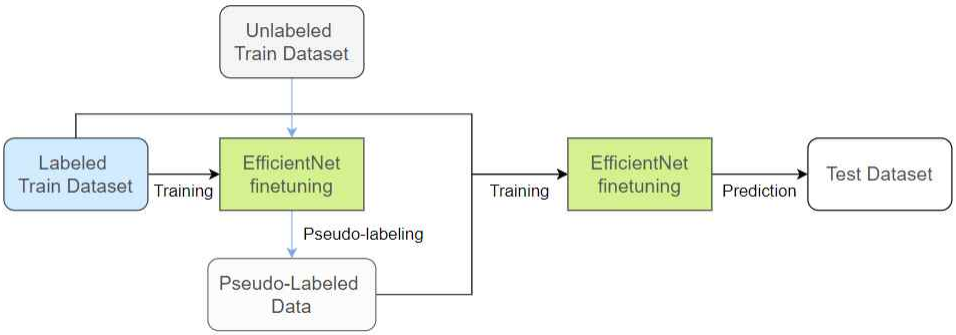

Fig 1. Architecture

(1) Model

model은 최근 ImageNet 대회에서 최고 성능을 낸 EfficientNet을 사용했다. AutoML and Compound Scaling을 사용하여 이전 모델들보다 경량화되어 속도가 빠르기 때문이다. 3-(2)의 결과로 최종적으로 EfficientNet b3을 사용하였다. 우리 프로젝트의 목표는 semi-supervised learning으로 labeled training dataset이 부족한 상황이다. 따라서, pretrained된 EfficientNet을 사용한다.

```
model = EfficientNet.from_pretrained('efficientnet-b3').to(device)
feature = model._fc.in_features
model._fc = nn.Linear(in_features=feature,out_features=10,bias=True).to(device)
```

(2) Architecture

전체적인 구조는 Fig 1과 같다. semi-supervised learning을 하기 위해서 먼저 data augmentation을 수행했다. resize, random horizontal flip, normalize를 이용하여 5000장으로 부족한 labeled dataset을 보충하였다. labeled train dataset 5000장을 training dataset 4500장, validation dataset 500장으로 나누어 학습을 진행했다. 이때는 우리가 가지고 있는 정확한 label이 달린 데이터셋에 적합한 weight를 학습시키기 위하여 finetuning을 하도록 하였다. 이 model을 이용하여 unlabeld train dataset의 10개 class에 대한 prediction값에 sharpening을 진행했다. 이렇게 예측한 sharpening predcition 값이 threshold 0.8을 넘는 데이터만 over_threshold_name_list.txt 파일로 저장하여 새로운 labeled dataset을 생성하였다.
이후, 이 데이터셋을 이용하여 다시 모델을 학습시켰다. 이때, labeled training dataset과 pseudo labeled training dataset을 랜덤하게 섞어서 0.25%는 validation set으로 나머지는 training set으로 사용하였다. 이와 같은 Pseudo labeling한 데이터셋 이용이 성능향상에 도움을 주는 이유는 model의 decision boundary를 결정할 때, 그 경계를 구분하는 지점의 데이터가 몰려있는 밀도가 낮아져 더 미세한 차이점도 구별할 수 있기 때문이다. 또한, shaperning prediction을 통해서 unlabeled data가 가지는 class별 확률에 대한 entropy를 최소화시킴으로써 다시 한번, 앞서 말한 low-density separation을 하는데 도움이 되기 때문이다. 이렇게 학습한 모델로 최종적으로 test dataset에 대해서 예측하였다.

2. Train and predict procedure

기존의 코드 training과 test 사이에 pseudo labeling을 하고 학습을 진행시키는 과정을 추가하였다.

pseudo labeling하는 함수

```
trainer._pseudo_label(train_unlabeled_dataset, train_batch)
pseudo_data = SSL_Dataset(root='../', transform=transform, mode='pseudo_train')
num_train = len(pseudo_data)
num_valid = int(num_train * 0.25)
train_pseudo_dataset, valid_pseudo_dataset = torch.utils.data.random_split(pseudo_data, [num_trai
n-num_valid, num_valid])

pseudo_trainloader = DataLoader(train_pseudo_dataset, batch_size=train_batch, shuffle=True)
pseudo_validloader = DataLoader(valid_pseudo_dataset, batch_size=test_batch, shuffle=False)
model.load_state_dict(torch.load(weight_path+model_name))
# for n, p in model.named_parameters():
#     if '_fc' not in n:
#         p.requires_grad = False
if test_only == False:
   print(f"# Pseudo Train data: {len(train_pseudo_dataset)}, #Pseudo Valid data: {len(valid_pseud
o_dataset)}")
   trainer._train(pseudo_trainloader, pseudo_validloader)
```

(1) Learning Rate 설정

: learning rate를 0.001, 0.003, 0.005, 0.007, 0.01로 설정한 후, 결과를 비교하였다. 그 결과 validation set에선 0.007로 설정한 결과 95.46%로 가장 좋게 나왔으며, test set에선 0.005로 설정한 것이 84.766%으로 가장 높게 나왔다. 따라서 learning rate은 0.005로 설정하였다.

```
| Train Epoch #11         Accuracy: 85.32%
| Valid Epoch #11         Accuracy: 88.71%
| Best Accuracy updated (88.69 => 88.71)
```

learning rate 0.001

```
| Train Epoch #12         Accuracy: 92.37%
| Valid Epoch #12         Accuracy: 94.95%
| Best Accuracy updated (94.91 => 94.95)
```

learning rate 0.003

```
| Train Epoch #7          Accuracy: 90.44%
| Valid Epoch #7          Accuracy: 95.27%
| Best Accuracy updated (95.04 => 95.27)
```

learning rate 0.005

```
| Train Epoch #10         Accuracy: 93.50%
| Valid Epoch #10         Accuracy: 95.46%
| Best Accuracy updated (95.18 => 95.46)
```

learning rate 0.007

```
| Train Epoch #4          Accuracy: 87.26%
| Valid Epoch #4          Accuracy: 93.80%
| Best Accuracy updated (92.84 => 93.80)
```

learning rate 0.01

3. Trial

(1) Data Augmentation : learning rate 0.005, efficientnet b0로 고정하고 진행하였다. 이때, RandomHorizontalFlip()과 Normalize를 추가로 실행하였을 때, 성능이 향상된 것을 볼 수 있다. 따라서, 다음의 Data Augmentation을 추가하였다.

```
| Train Epoch #32         Accuracy: 82.27%
| Valid Epoch #32         Accuracy: 80.20%
| Endure 10 out of 10
```

Data Augmentation
: Resize(size=(32, 32)),
ToTensor(),

```
| Train Epoch #38         Accuracy: 86.16%
| Valid Epoch #38         Accuracy: 81.60%
| Endure 10 out of 10
```

Data Augmentation
: Resize(size=(32, 32)),
ToTensor(),
RandomHorizontalFlip(),
Normalize([0.485, 0.456, 0.406],
[0.229, 0.224, 0.225])]

(2) EfficientNet Version

```
| Train Epoch #30         Accuracy: 82.49%
| Valid Epoch #30         Accuracy: 80.20%
| Endure 10 out of 10
| Train Epoch #20         Accuracy: 75.67%
| Valid Epoch #20         Accuracy: 83.20%
| Best Accuracy updated (83.00 => 83.20)
```

b1

```
| Train Epoch #56         Accuracy: 84.85%
| Valid Epoch #56         Accuracy: 80.20%
| Endure 15 out of 15
| Train Epoch #35         Accuracy: 76.50%
| Valid Epoch #35         Accuracy: 83.20%
| Best Accuracy updated (82.80 => 83.20)
```

b2

```
| Train Epoch #40         Accuracy: 87.24%
| Valid Epoch #40         Accuracy: 82.60%
| Endure 15 out of 15
| Train Epoch #25         Accuracy: 80.36%
| Valid Epoch #25         Accuracy: 84.00%
| Best Accuracy updated (83.40 => 84.00)
```

b3

```
| Train Epoch #24         Accuracy: 77.22%
| Valid Epoch #24         Accuracy: 82.40%
| Endure 10 out of 10
| Train Epoch #14         Accuracy: 66.92%
| Valid Epoch #14         Accuracy: 84.20%
| Best Accuracy updated (82.60 => 84.20)
```

b4

Model	Top-1 Acc.	Top-5 Acc.	#Params
EfficientNet-B0	**77.3%**	**93.5%**	**5.3M**
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M
EfficientNet-B1	**79.2%**	**94.5%**	**7.8M**
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M
Xception (Chollet, 2017)	79.0%	94.5%	23M
EfficientNet-B2	**80.3%**	**95.0%**	**9.2M**
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M
EfficientNet-B3	**81.7%**	**95.6%**	**12M**
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M
EfficientNet-B4	**83.0%**	**96.3%**	**19M**
SENet (Hu et al., 2018)	82.7%	96.2%	146M
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M
EfficientNet-B5	**83.7%**	**96.7%**	**30M**
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M
EfficientNet-B6	**84.2%**	**96.8%**	**43M**
EfficientNet-B7	**84.4%**	**97.1%**	**66M**
GPipe (Huang et al., 2018)	84.3%	97.0%	557M

EfficientNet B0~B7 #Parameters

efficientnet의 version은 총 b0~b7까지 있다. 이때 b4부터 parameter의 수 대비 성능향상이 미미하므로, b0~b4 모델의 성능을 비교하였다. b0의 실험 결과는 (1)의 두 번째 실험과 동일하므로 b1~b4까지의 실험 결과는 다음과 같다. 이때 b3과 b4의 결과가 84%를 넘는 accuracy를 보이며, 둘의 0.2%차이인데 b3일 때 training accuracy가 b4일때보다 높으므로 더 안정된 학습을 하고 있다고 판단하여, b3을 사용하였다.

(3) FineTuning vs Feature Extraction

```
| Train Epoch #7          Accuracy: 90.44%
| Valid Epoch #7          Accuracy: 95.27%
| Best Accuracy updated (95.04 => 95.27)
| Train Epoch #16         Accuracy: 94.89%
| Valid Epoch #16         Accuracy: 95.18%
| Endure 9 out of 9
```

FineTuning

```
| Train Epoch #11         Accuracy: 85.32%
| Valid Epoch #11         Accuracy: 88.71%
| Best Accuracy updated (88.69 => 88.71)
| Train Epoch #20         Accuracy: 86.93%
| Valid Epoch #20         Accuracy: 87.26%
| Endure 9 out of 9
```

Feature Extraction

Pseudo Labeling을 한 데이터셋을 학습할 때, fc layer의 weight만 학습시키는 feature extraction과 앞단의 weight까지 학습시키는 finetunig 기법 중에서 어떤 기법을 사용해야 더 성능이 올라가는지 확인하기 위해서 실험을 진행했다. 그 결과, training accuracy는 약 5%, validation accuracy는 약 7%, test accuracy는 약 3% 가량 마지막단만 학습시키는 것보다 앞단부터 학습을 시키는 것이 성능이 좋음을 확인하였다.

(4) Pseudo Labeling Threshold

sharpening한 prediction 값이 0.9이상 데이터셋을 사용한 결과 테스트셋에서 0.84100의 accuracy를 보였으며, 0.8이상 데이터셋을 사용한 결과 테스트 셋에서 0.84766 accuracy를 보였다. 따라서, threshold는 0.8로 지정하였다.

4. Final results

22	zoomin_lee		0.84766