Semi-Supervised Learning

1. 모델 설명

Pi-Model은 레이블이 지정된(labeled) 및 지정되지 않은(unlabeled) 데이터에 대한 consistency을 활용하여 모델을 학습시키는 방법입니다. 이 모델은 크게 두 가지 단계로 나눌 수 있습니다.

- 레이블이 지정된 데이터 학습

레이블이 지정된 데이터에 대해 기본 모델을 사용하여 예측을 수행하고, Cross-Entropy Loss 을 계산합니다.

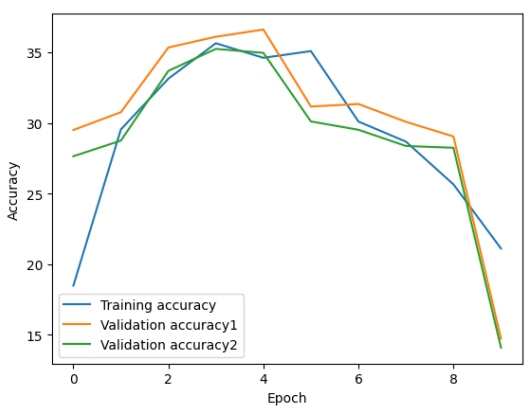
- 레이블이 지정되지 않은 데이터 학습

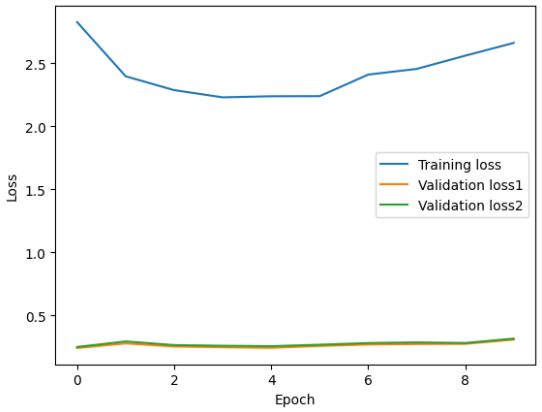
레이블이 지정되지 않은 데이터에 대해서는 예측 및 consistency 손실 계산이 이루어집니다.
예측된 가짜 레이블(pseudo-labels)을 생성하고, 이를 사용하여 일관성 손실을 계산합니다.
모델의 예측과 Teacher Model 의 예측 간의 일관성을 유지하기 위해 EMA(Exponential Moving Average)를 사용합니다. 주로 Mean Squared Error Loss(MSE Loss)를 사용하여 계산합니다.
이 일관성 손실은 모델이 레이블이 지정되지 않은 데이터에 대해서도 안정적으로 학습하도록 도와줍니다.

2. 결과

실험 결과, overfitting이 심각하게 발생하며 validation accuracy가 46-7%부근에서 잘 올라가지 않는 현상을 발견했다. dropout을 추가하고 early stopping을 하는 등 모델과 하이퍼 파라미터 등을 바꾸며 여러 가지로 시도해 보았지만 모든 경우에 대해 동일하게 accuracy가 상승하지 않는 모습을 보였다.

먼저 모델의 complex가 너무 높은 것이 의심되어 mobileNetV2로 실험을 진행하여 보니, 아래와 같은 결과가 나왔다.





따라서 기존의 resnet18을 그대로 사용하기로 하고, 테스트를 위해 pretrained model을 가져와서 test해보니, validation accuracy가 정상적으로 잘 증가함을 볼 수 있었다. 모델 자체의 문제라기 보다는 다른 문제가 있을 것이라 생각했다.

문제가 datatransform이 너무 단순한 것에 있을것이라 추측하고, 기존의 아래였던 것을 바꾸어 보았다.

```
# Feel free to incorporate additional transforms, such as data
augmentation.
transform = transforms.Compose(
    [
        transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))]
    )
BATCH_SIZE = 8
```

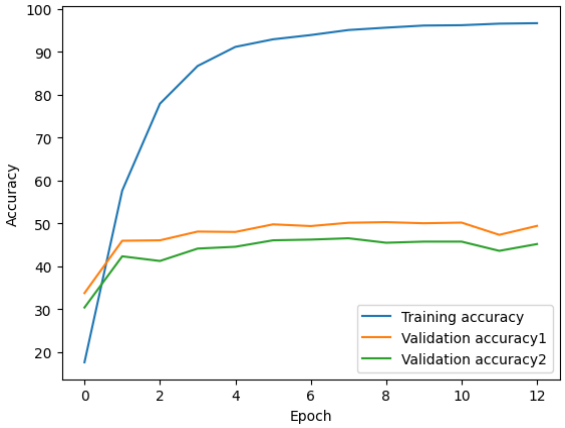
아래와 같이 randomResizeCrop, RandomHorizontalFlip, colorjitter을 추가하자 아래와 같이 성능이 더 많이 하락하는 것을 볼 수 있었다. 그 외에도 autoaugment cifar10을 이용해보았으나 마찬가지로 특별한 성능 향상은 관측되지 않았다.

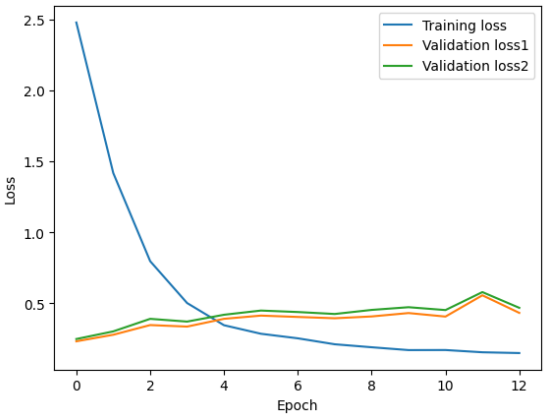
```
def get_color_distortion(s=0.5):  # 0.5 for CIFAR10 by default
    # s is the strength of color distortion
    color_jitter = transforms.ColorJitter(0.8*s, 0.8*s, 0.8*s, 0.2*s)
    rnd_color_jitter = transforms.RandomApply([color_jitter], p=0.8)
    rnd_gray = transforms.RandomGrayscale(p=0.2)
    color_distort = transforms.Compose([rnd_color_jitter, rnd_gray])
    return color_distort

transform = transforms.Compose([transforms.ToTensor(),
   0.5)),transforms.RandomResizedCrop(32),

transforms.RandomHorizontalFlip(p=0.5),get_color_distortion(s=0.5)] ,
transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, )
```

여러 실험 결과, 추가적인 random augmentation 없이 normalize 수치만 아래와 같이 바꾼 것이 가장 잘 작동하였다.

```
transform = transforms.Compose([transforms.ToTensor(),
   transforms.Normalize((0.4914, 0.4822, 0.4465), (0.247, 0.243,
0.261))
                                ])
```

train, validation loss를 뽑아보니, training loss는 하락하는 반면, validation loss는 증가하는 현상을 보이기는 한다.





validation loss가 증가하고는 있으나, accuracy가 증가하는 것에 무게중심을 두고 분석했다.

Self-Supervised Learning

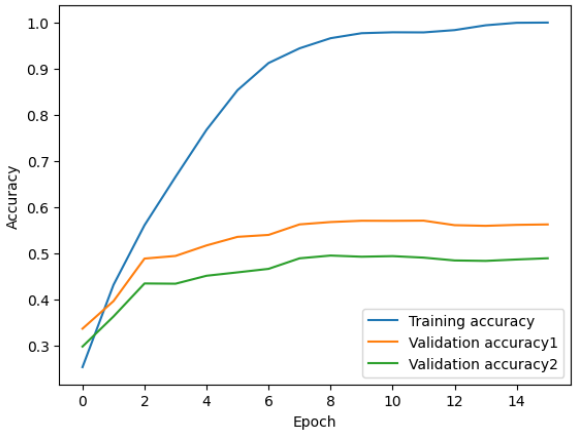
1. 모델 설명

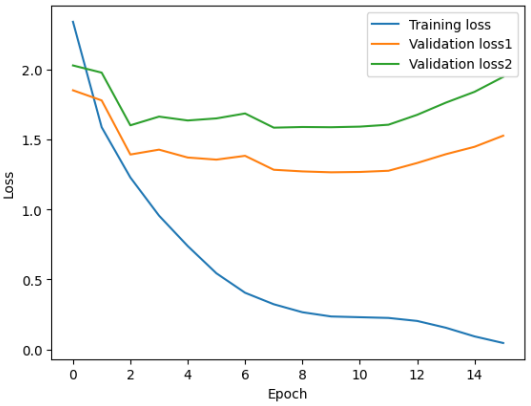
simCLR을 활용했다.
https://github.com/p3i0t/SimCLR-CIFAR10/tree/master 여기 있는 implementation을 많이 참고하였다. backbone은 semi-supervised와 마찬가지로 resnet18을 활용했다.
입력 데이터에 대한 두 가지 다른 변환(Transform)을 정의합니다. 각각 transform1과 transform2로 명명되었으며, 각각에는 이미지 크기 조정, 무작위 가로 뒤집기, 색상 왜곡, 정규화 등이 포함됩니다. 이 두 변환은 DoubleTransform 클래스를 통해 구현되었습니다. 이 클래스는 주어진 unlabeled_train_dataset에 적용되어, 이를 묶어서 새로운 텐서로 반환하였다.
SimCLR 클래스는 SimCLR의 핵심 모델을 정의합니다. 이 모델은 기본 인코더 모델을 사용하여 특성을 추출하고, 이를 MLP 기반의 프로젝터를 통해 저차원으로 매핑합니다.
SimCLR은 Noise-Contrastive Loss를 사용하여 모델을 학습합니다. 이 함수는 두 이미지 샘플 간의 유사성을 측정하는 데에 사용됩니다.
주어진 self-supervised 학습 방법으로 학습된 SimCLR 모델의 인코더를 사용하여 선형 분류기를 정의한 LinModel 클래스를 구현합니다. 이 클래스는 입력 데이터에 대한 특성을 추출하고, 선형 레이어를 통해 최종 클래스 예측을 수행하는 finetuning을 실행합니다.
Cosine Annealing 기반의 학습률 스케줄러(LambdaLR)를 사용하여 학습률을 동적으로 조절합니다. 학습률은 학습 과정 동안 코사인 함수를 따라 감소하도록 설계되어 있습니다. 이를 활용한 이유는 대부분의 self-supervised learning, contrastive learning 계열이 이를 활용하는 것처럼 보였기 때문이다.

2 결과

앞선 semi-supervised에 비해 좀 더 성능이 개선되었음을 확인할 수 있다.
아래는 finetuning 을 마친 모델의 성능이다. epoch 12 정도를 넘어가는 과정에서 overfitting이 심하게 일어남을 관찰할 수 있다. 이에 best model은 validation loss가 최저가 되는 시점에서 잡았다.
앞선 semi-supervised와는 달리, get_color_distortion에서 color jittering을 이용한 transform을 추가했다. self-supervised pretraining 과정에서는 좀 더 데이터 그 자체에 대한 심화된 학습이 필요하다고 생각했기 때문에 더 많은 Transform을 넣어줬다.
resnet34 등의 좀 더 고도화된 모델을 base encoder로 사용해 보아도 크게 성능이 향상되지는 않았다.

```
def get_color_distortion(s=0.5):  # 0.5 for CIFAR10 by default
    # s is the strength of color distortion
    color_jitter = transforms.ColorJitter(0.8*s, 0.8*s, 0.8*s, 0.2*s)
    rnd_color_jitter = transforms.RandomApply([color_jitter], p=0.8)
    rnd_gray = transforms.RandomGrayscale(p=0.2)
    color_distort = transforms.Compose([rnd_color_jitter, rnd_gray])
    return color_distort
```


Accuracy
Epoch
Training accuracy
Validation accuracy1
Validation accuracy2


Loss
Epoch
Training loss
Validation loss1
Validation loss2