SimCLR (Embedded Practice 1)

2017-19428

컴퓨터공학부 서준원

1. Training Loss Curve

노트북의 빈 칸을 완성했다. 첫째로 모델 구조를 완성한 후, Transform 구조를 instruction 대로 작성했다. Resnet 구조를 만든 후, 64차원으로 Encoding 된 feature를 pre-training에 사용하기 위한 head와 classifier 네트워크를 만들었다.

학습의 성능을 높이기 위해 torchvision.transforms 을 사용하였고 아래와 같이 Duplicated Transform을 만들었다.

```
    transforms.RandomResizedCrop(img_size),
    transforms.RandomHorizontalFlip(p=0.5),
    color_jitter,
    transforms.RandomGrayscale(p=0.2),
    GaussianBlur(img_size[0]//10),
```

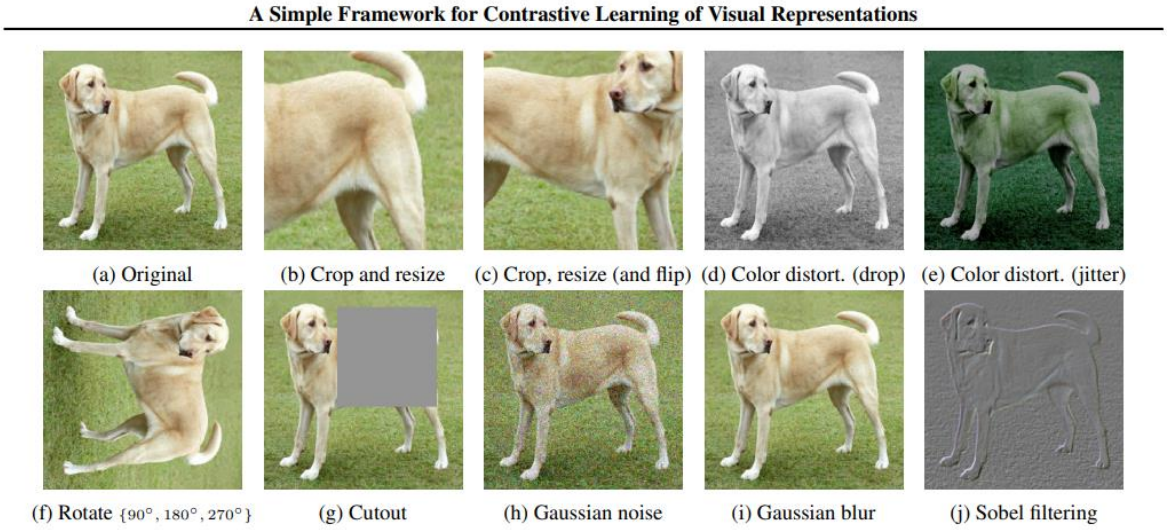
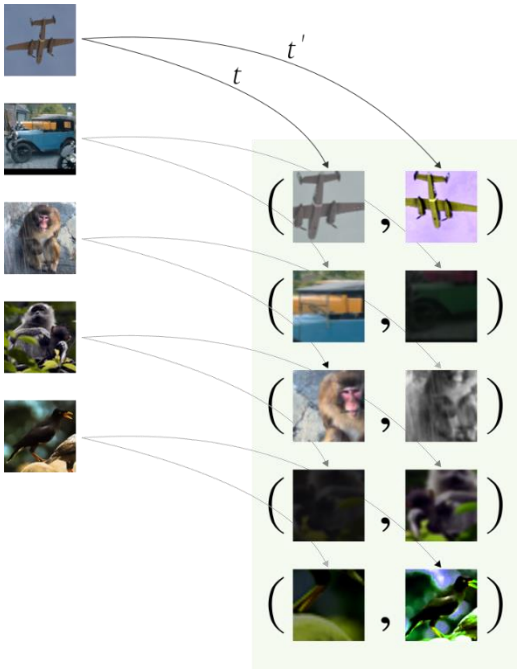


Figure 4. Illustrations of the studied data augmentation operators. Each augmentation can transform data stochastically with some internal parameters (e.g. rotation degree, noise level). Note that we *only* test these operators in ablation, the *augmentation policy used to train our models* only includes *random crop (with flip and resize)*, *color distortion*, and *Gaussian blur*. (Original image cc-by: Von.grzanka)

그림 1 https://arxiv.org/pdf/2002.05709.pdf

그 후 학습과 테스트를 위해 CIFAR10 데이터셋을 로딩했다. Loss는 NTXentLoss 구현체를 사용했다. 논문에서는 InfoNCE를 사용했는데, 이는 메모리 소모가 너무 많다. (224,224)의 이미지를 batch size 256으로 학습하기 위해서 NTXentLoss를 사용했다. 2n-2개의 Negative Sample과 2개의 positive sample의 cosine similarity를 계산하는 방법이다. Logit을 나눠주는 계수인 temperature는 0.07로 세팅했다.



학습에는 SGD_with_lars optimizer와 WarmupScheduler를 사용했다. Warmup Scheduler는 learning rate를 처음 (여기선 20epoch / 200) 에 warm-up 한 후 그 후에는 원래의 scheduling을 한다. 여기서는 CosineAnnealingLR을 사용했다.

학습은 batch_size 256, Single GPU, epoch 200 환경에서 진행됐다. 한 에폭당 학습은 약 60-70초가 소모되었다. 200에폭을 도는 데에는 약 3-4시간이 소모되었으며, 코랩의 사용 제한을 막기 위해 브라우저 상에서 1분 마다 한번씩 특정 버튼을 클릭하게 했다. 그래도 GPU 사용이 너무 많아지면 약 12시간에 한번씩 꺼졌다. 이를 방지하기 위해 약 40에폭 당 한 번씩 모델을 저장했으며, 여러 개의 구글 아이디로 동시에 학습했다.

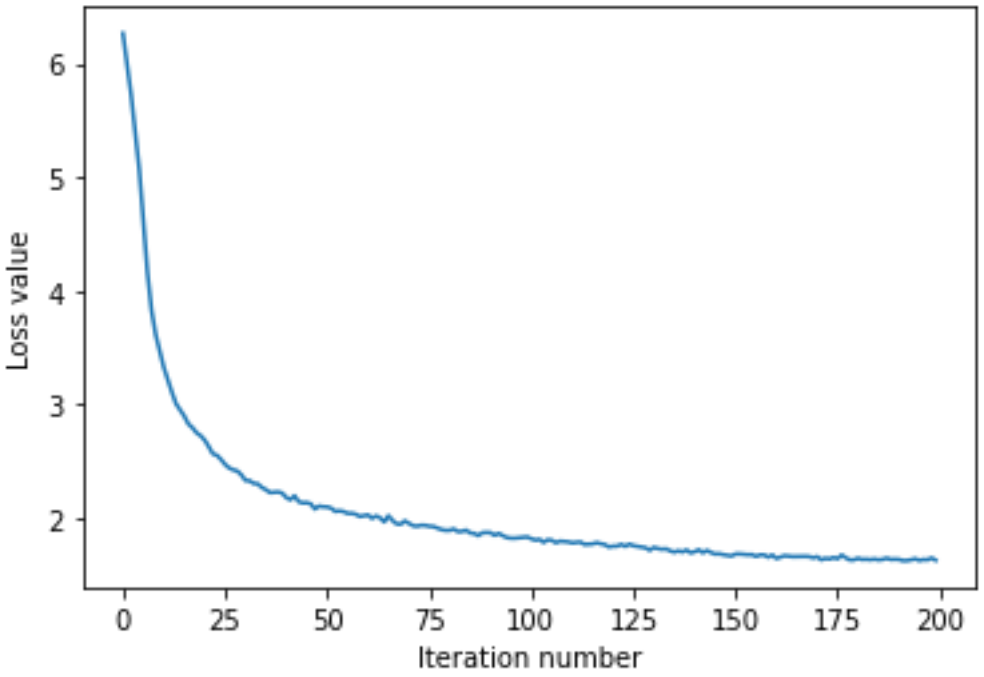
SimCLRNet의 구조는 아래와 같다.

```
SimCLRNet(
  (feat): ResNetCifar(
    (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (layer1): Sequential(
      (0): BasicBlock(
        (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
```

```
       (conv2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
     )
… 반복

  (head): SimCLRHead(
    (fc1): Linear(in_features=64, out_features=64, bias=True)
    (relu): ReLU(inplace=True)
    (fc2): Linear(in_features=64, out_features=32, bias=True)
    (norm): Normalize()
  )
  (classifier): Linear(in_features=64, out_features=10, bias=True)
  (norm): Normalize()
       )
```

Loss 커브는 아래 그림과 같이 얻어졌다. 최종 에폭에서 로스는 약 1.62였다.




```
Epoch    200    Loss    1.6273717764096383    Time    64.58815407752991
Finished training. Train time was: 13051.887996673584
```

2. Implement linear evaluation protocol

A. Training Setting

Pre Train 된 CNN 을 Freeze 하고 Linear Classifier 를 학습했다. 학습에는 Transform 을 적용하지 않고 Linear Evaluation 을 적용했다. Logit 을 loss_fn = torch.nn.CrossEntropyLoss()를 사용해 Loss 를 적용했다. 아래와 같이 requires_grad 설정으로 Freezing 을 했다.

```
    loss_fn = torch.nn.CrossEntropyLoss()

    # Freezing
```

```
    net.feat.requires_grad = False
    net.head.requires_grad = False
    net.norm.requires_grad = False

    optimizer = torch.optim.Adam(net.classifier.parameters(), lr=1e-3)
```

또한 Optimizer는 Adam을 사용하였으며 Learning rate는 1e-3으로 고정했다. 학습은 30에폭만 실행했다. Instruction대로 세팅을 했는데 60% 이상의 정확도가 나와 다른 세팅에 대해서 실험해보지는 않았다.

B. Accuracy

```
Epoch  1     Loss    2.03468404121888   Time    17.137782096862793
Epoch  2     Loss    1.6301073435025337       Time    17.24441909790039
Epoch  3     Loss    1.4283286290291028       Time    17.066750288009644
Epoch  4     Loss    1.3150144711518899       Time    17.026099681854248
Epoch  5     Loss    1.2440750293242626       Time    17.067901849746704

Epoch  30    Loss    0.9929774443308512       Time    17.321990966796875
```

```
test_dataset = datasets.CIFAR10(root='.',
                                train=False,
                                download=True,
                                transform=transforms.ToTensor()
                               )

test_loader = DataLoader(test_dataset,
                         batch_size=256,
                         num_workers=4,
                         shuffle=True,
                         drop_last=True
                        )
```

```
Accuracy : 64 %
```

Batch_size = 256, temperature =0.07, epoch=200, linear_epoch = 30 세팅에 대해서 64%의 accuracy를 기록했다. 테스트에는 CIFAR10의 test 데이터셋을 사용했다.

3. Ablation Studies on pre-training step

위의 과정을 다양한 Pre-training parameter 세팅에 대해서 반복해보고 Accuracy를 비교했다. 마찬가지로 학습에 오랜 시간이 걸리는데 자꾸 코랩이 꺼져서 고생을 했다. 브라우저에서 계속 버튼을 클릭하게끔 매크로를 설정하고 여러 개의 구글 계정으로 나누어 학습을 했다.

Batch는 [64, 128, 256, 512, 1024], Temperature는 [0.01, 0.05, 0.1, 0.5, 1.0], Epoch은 [100, 200,

300, 400]에 대해서 실험했다. 각각의 조건에 대해 Pretrain 한 후 같은 횟수만큼 Linear Classifier를 학습시킨 후 Accuracy를 측정했다.

논문에서는 Batch 사이즈가 클수록, Pre training Epoch이 많을수록 Accuracy가 증가한다. 아래는 논문의 표이다.

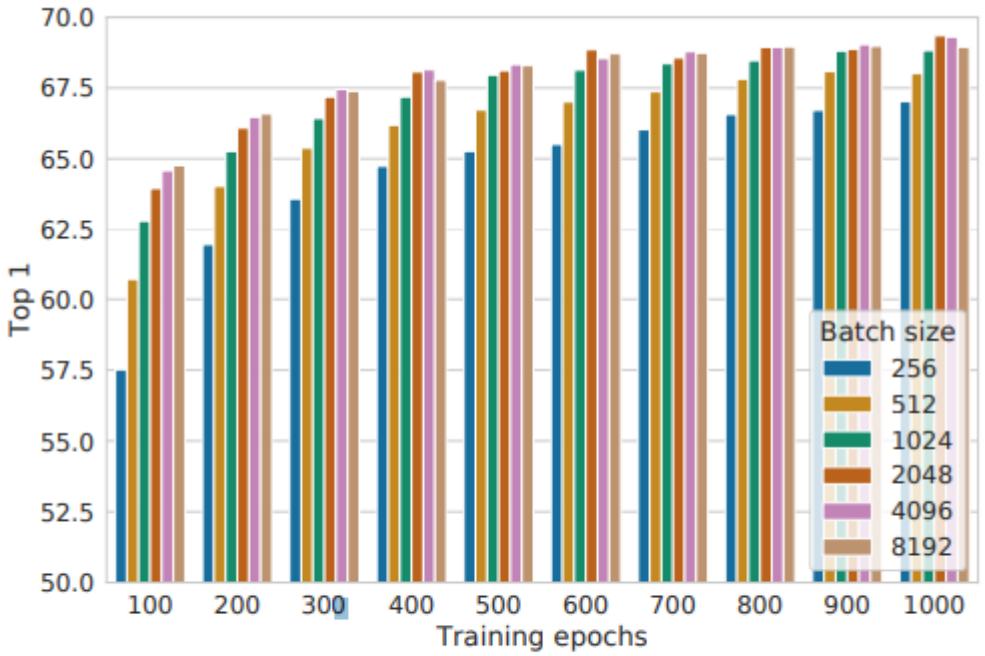
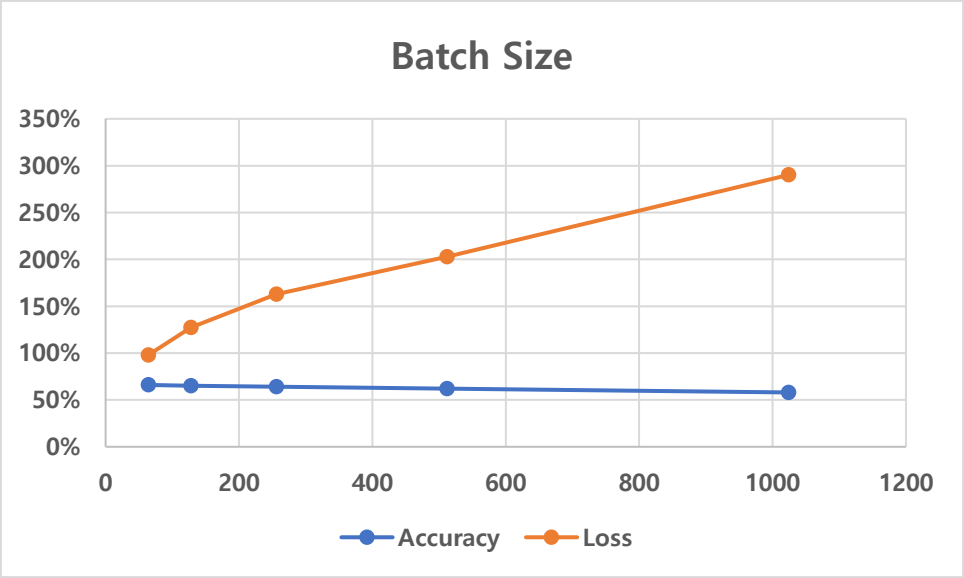


Figure 9. Linear evaluation models (ResNet-50) trained with different batch size and epochs. Each bar is a single run from scratch.[10]

그림 2 : https://arxiv.org/pdf/2002.05709.pdf

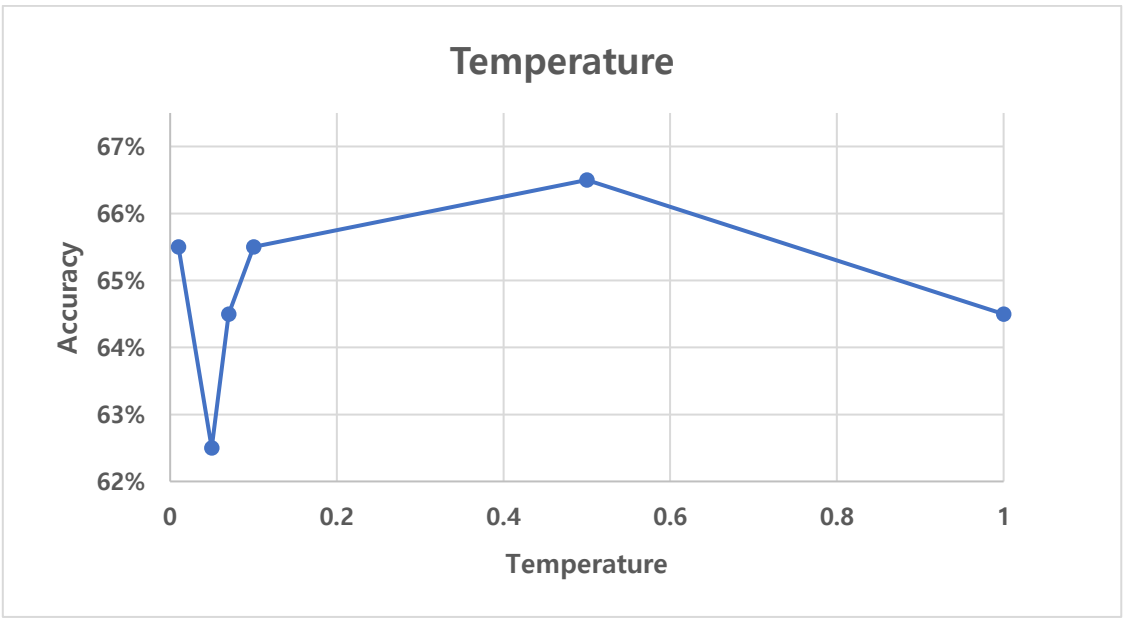
결과는 다음과 같다. Batch를 키웠을 때는 Loss가 오히려 증가하였다. 이유는 잘 모르겠다. Negative Sample의 수가 증가해서 Loss가 커져서 학습이 잘 되지 않은 것 같은데 이유는 확실히 모르겠다. Loss가 증가함에 따라 Accuracy도 떨어지는 양상을 보였다.

Batch	Acc	Loss
64	66%	0.97925
128	65%	1.27144
256	64%	1.62737
512	62%	2.02743
1024	58%	2.90124



여러 Temperature에 대해서 테스트를 해봤다. 0.07, 0.1, 0.5 일 때 가장 높았다. 이유는 Parameter 여서 제대로 해석할 수 없었다. 잘 모르겠다.

Temperature	Acc
0.01	65%
0.05	62%
0.07	64%
0.1	65%
0.5	66%
1	64%



마지막으로 다양한 Epoch에 대해서 테스트 해봤다. 이것이 유일하게 논문의 결과와 일치한다 Epoch이 증가할수록 accuracy가 증가하고 Loss가 감소했다. 또한 Epoch에 따라 Time도 Linear하게 증가함을 당연하지만 확인해봤다.

Epoch	Acc	Time	Loss
100	62%	6517.78	1.84345
200	64%	13051.9	1.62737
300	64%	19923.2	1.5205
400	65%	25988.3	1.5562