VGGNet and ResNet Lab final-report

ByeongGeun Shin[1],
[1]Pusan National University.

1 Very Deep Convolutional Networks for Large-Scale Image Recognition Paper Summary

1.1 Paper Abstract

Very Deep Convolutional Networks for Large-Scale Image Recognition 은 ILSVRC 2014 대회에서 2등을 차지한 Karen Simonyan과 Andrew Zisserman이 발표한 VGGNet 모델에 대한 논문으로, VGG 등장 이전 AlexNet의 8층 layer구조를 넘어 더 깊은 19층 Layer 구조를 선보였다. 해당 논문에선 CNN 구조 설계에 있어 네트워크의 depth가 정확도에 미치는 영향에 대해 실험하고, depth를 늘리면서도 vanishing/exploding gradient 문제 등 훈련에 있어 다양한 문제를 어떻게 해결하였는지 서술한다. ILSVRC 대회에서는 GoogLeNet 보다 이미지 분류 성능은 낮았지만, 다른 연구에서는 좋은 성능을 보인다고 한다.

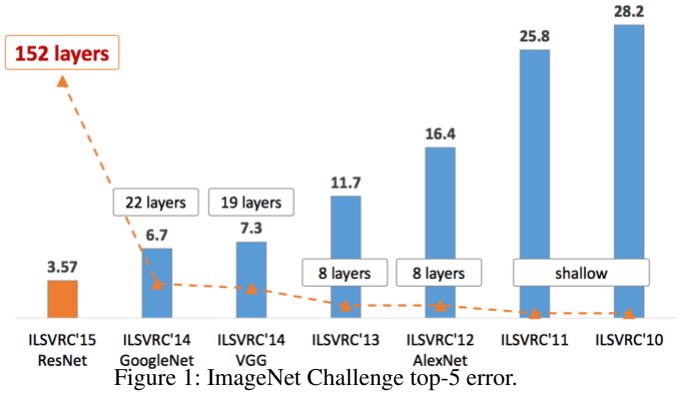

Figure 1: ImageNet Challenge top-5 error.

1.2 Paper Introduction

2015년 당시 고해상도의 이미지를 입력받아, 사물이나 물체를 인식하여 분류하는 대회인 ILSVRC에서 Convolution연산을 사용하는 AlexNet (2012) [1]가 영상 및 비디오 인식 결과에서 뛰어난 결과를 보인 영향으로, 이러한 딥러닝 모델을 사용하는 팀들이 우수한 성적들을 많이 보여주고 있었다.

VGGNet 연구팀은 대규모 이미지 인식에 있어 Conv 네트워크의 깊이가 정확도에 어떤 영향을 미치는 조사하였고, 3 x 3 Conv filter를 여러 개 쌓아 기존 CNN 모델의 layer 개수를 deep하게 늘렸고, 이것이 좋은 결과를 얻게 하였다고 한다.

1.3 Paper VGGNet Configurations

1.3.1 Architecture

VGGNet의 구조는 다음과 같다. Input Image는 224x224 RGB이고, 전처리는 RGB평균값을 빼는 것만 적용하였다. Conv layer는 이미지 요소의 left, right, up, down 등을 파악할 수 있는 최소한의 receptive field인 3x3을 적용했으며, stride는 1, padding은 1로 설정하였다. Pooling Layer는 Conv layer 다음에 적용되고, 총 2x2 크기, stride 2인 5개의 max pooling 층으로 구성된다. 그 다음 FC Layer는 처음 2개는 4,096 채널 마지막 층에선 1000 채널으로 구성했고 마지막으로 Softmax layer로 분류를 진행한다.

Conv Layer의 폭은 64부터 시작해서 max pooling layer를 통과할 때마다 2의 제곱만큼 커져, 최대 512까지 커진다. 실험에서 중심적으로 봐야할 점은, 더 큰 Conv layer를 사용한 AlexNet의 8-layer모델 보다 깊이가 2배 이상 늘어났음에도 파라미터 수는 오히려 줄었다.

연구팀은 3x3 Conv filter를 2개 사용하는 것은 5x5 Conv filter 1개를 사용하는 것과 동일한데 작은 사이즈로 여러번 나눠 적용하면 Relu와 같은 활성화 함수를 여러번 거쳐 비선형성이 증가하고 더욱 복잡한 패턴을 인식할 수 있다고 한다. 뿐만 아니라, 큰 Conv layer 1개를 사용하는 것보다 작은 Conv layer 여러번이 파라미터 수가 적어 Overfitting이 줄어들고 학습속도가 빨라진다.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Figure 2: VGGNet Architecture.

1.3.2 Paper Configurations

학습 시 최적화 전략으론 Optimizing multinomial logistic regression, mini-batch gradient descent, Momentum(0.9), Weight Decay(L2 Norm), Dropout(0.5), Learning rate 0.01로 초기화 후 서서히 줄이는 방식을 적용하였다. 이러한 방식으로 AlexNet보다 더 깊고 파라미터 수도 많지만 더 적은 Epoch를 달성하였고 모델의 깊이를 여러개로 나눠 실험할때, 처음 모델이 학습한 첫 4개 Conv layer와 마지막 3개 FC layer를 가져와 최적의 초기값을 설정하였다.

모델 학습 시 입력 이미지는 VGG 모델의 Input size에 맞게 바꿔주는데 원 사이즈의 비율을 지키며 사이즈를 Rescaling 하고 random하게 224x224 size로 crop한다. 스케일링 방식은 모든 입력 이미지를 동일한 크기로 변환하는 Single-scale training과 다양한 크기의 이미지로 변환시키는 Multi-scale training을 둘 다 지원하며, 이를 통해 한정적인 데이터 수를 늘리는 증강 효과와 더불어 하나의 오브젝트에 대해 다양한 측면을 반영시킴으로써 Overfitting을 방지하고 일반화 성능을 높였다.

1.4 Paper Testing

앞서 Training image를 전처리하는 것처럼 Test를 진행할 때도 rescale을 적용한다. 이때, 테스트 이미지의 Scale 파라미터(Q)가 훈련셋의 Scale 파라미터(S)와 같을 필요는 없는데 각각의 S값마다 다른 Q를 적용 시 VGG 모델의 성능이 좋아진다. Training 할 때의 구조와 Validation을 수행할 때의 구조가 다른데, Validation을 수행할 때에는 FC layer를 Conv layer로 바꿔준다. AlexNet에서는 10 augmented image를 사용하고, 10개 이미지에 대해 평균을 취하므로 속도가 매우 느려지지만, FC layer를 1 x 1 conv layer로 바꾸고, 약간 큰 사이즈의 이미지를 넣고 horizontal flipping만 적용했기에 속도가 빠르고 성능이 증가했다.

1.5 Paper Experiments

VGGNet의 실제 결과를 보면 3x3 convolution layer를 2개 겹쳐서 사용하는 경우와 5x5 convolution 1개를 사용하는 모델을 만들어 실험을 했는데,

결과는 3x3 2개를 사용하는 것이 5x5 1개보다 top-1 error에서 7% 정도 결과가 좋았다. 또한 Multi-Scale Evaluation에서 Single-Scale보다 Multi-scale이 좋다는 것을 확인할 수 있다.

VGGNet Evaluation

Table 3: **ConvNet performance at a single test scale.**

ConvNet config. (Table1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

Table 4: **ConvNet performance at multiple test scales.**

ConvNet config. (Table1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
B	256	224,256,288	28.2	9.6
C	256	224,256,288	27.7	9.2
	384	352,384,416	27.8	9.2
	[256; 512]	256,384,512	26.3	8.2
D	256	224,256,288	26.6	8.6
	384	352,384,416	26.5	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**
E	256	224,256,288	26.9	8.7
	384	352,384,416	26.7	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**

Figure 3: (a)Single-Scale (b)Multi-scale.

1.6 Paper Conclusion

깊이를 11층부터 19층까지 하여 7개의 모델의 앙상블을 사용했을때 가장 좋은 성능을 보였으며 이 모델로 ILSVRC에서 2위를 달성할 수 있었다. 결과적으로 네트워크의 길이를 최대 19까지 설정한 이유는, 해당 데이터에서 분류 오차율이 19층에서 수렴하였기 때문에 멈춘 것이고 데이터 세트가 충분히 더 많다면 더 깊은 모델을 사용할 수도 있다고 한다. 이로써 VGGNet은 이미지 분류 성능에 있어 네트워크의 깊이를 늘릴수록 좋아지는 것을 증명하였다.

2 Lab 1-1. VGG16 Implementation

2.1 Lab Constraint

2.1.1 Data

계산 제약이 있어 CIFAR-10의 처음 세 범주에서 가져온 이미지를 불러온다. 세 가지 범주는 비행기, 자동차, 새로 이루어져있으며 총 훈련 이미지 수는 15,000 / 테스트 이미지 수는 3,000개이다. 이 이미지들은 훈련셋에서 뒤집기 및 무작위 자르기로 증강을 적용한다.

2.1.2 Architecture

논문에서 유형-D에 해당하는 VGG16을 구성한다. 활성화 함수로는 ReLU를 사용하고 단순화를 위해 Drop out은 적용하지 않는다. 학습에 적합하도록 논문에서 사용하지 않았던 모든 컨볼루션 층 이후에 배치 정규화 층을 적용한다.

2.1.3 Loss function and Optimizing

오차 함수로는 출력값과 실제 정답값의 교차 엔트로피 손실을 사용하고, 단순화를 위해 기본 가중치 초기화를 적용한 학습률 = 1e-2, momentum = 0.9, weight decay = 5e-4를 사용하는 SGD알고리즘을 Optimizer로 사용한다. 훈련은 20Epoch를 진행하고 학습 속도 스케줄링은 적용하지 않는다.

2.2 Code Implementation

데이터 준비와 전처리는 torch의 dataset 구조와 transform Compose를 이용해 Random Crop, RandomHorizontalFlip을 적용하고 이미지의 RGB 평균값 (0.4914, 0.4822, 0.4465)와 표준편차 (0.2023, 0.1994, 0.2010)를 이용해 정규화한다.

다음 Figure-4는 제약사항을 지켜 논문의 D구조인 VGG16을 구현한 코드이다. 먼저, cfg 리스트는 VGG16 네트워크의 각 계층 구성을 정의하고 있으며, 'MP'는 최대 풀링(Max Pooling) 계층을 나타낸다. 이 리스트는 합성곱 계층의 필터 수와 풀링 계층의 위치를 순차적으로 나타내고 있다.

VGG 클래스는 nn.Module을 상속받아 정의되었으며, 두 가지 주요 부분으로 나뉘어 있다: VGG16 합성곱 계층과 classifier 완전 연결 계층이다. 먼저, VGG16은 make layers 메서드를 통해 정의된 합성곱 및 풀링 계층의 연속적인 블록으로 구성된다. 이 메서드는 cfg 리스트에 따라 합성곱 계층과 최대 풀링 계층을 추가하는 방식으로 레이어를 구성한다.

합성곱 계층은 nn.Conv2d로 구현되며, 각 계층마다 Batch Normalization과 ReLU 활성화 함수가 적용된다. 최대 풀링은 nn.MaxPool2d로 처리되며, 커널 크기 2와 스트라이드 2로 설정되어 다운샘플링을 수행한다.

classifier는 완전 연결 계층으로, 입력된 512차원 벡터를 먼저 4096차원으로 확장한 후, 다시 4096차원으로 변환하고, 최종적으로 3개의 클래스를 출력한다. 이 과정에서 두 번의 ReLU 활성화 함수가 적용되며, 네트워크의 비선형성을 강화하고 있다.

forward 메서드는 입력 데이터를 먼저 VGG16 계층을 통해 합성곱 연산과 풀링을 진행한 후, 최종적으로 완전 연결 계층으로 전달하여 분류 결과를 출력한다.

```
print('STEP 3: CREATE MODEL CLASS (VGG16)')

cfg = [64, 64, 'MP', 128, 128, 'MP', 256, 256, 256, 'MP', 512, 512, 512, 'MP', 512, 512, 512, 'MP']

class VGG(nn.Module):
    def __init__(self):
        super(VGG, self).__init__()
        self.VGG16 = self._make_layers(cfg)
        self.classifier = nn.Sequential(
            nn.Linear(512, 4096),
            nn.ReLU(True),
            nn.Linear(4096, 4096),
            nn.ReLU(True),
            nn.Linear(4096, 3)
        )

    def forward(self, x):
        out = self.VGG16(x)
        out = out.view(out.size(0), -1)
        out = self.classifier(out)
        return out

    def _make_layers(self,cfg):
        layers = []
        in_channels = 3
        for x in cfg:
            if x == 'MP':
                layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
            else:
                layers += [nn.Conv2d(in_channels, x, kernel_size=3, padding=1),
                           nn.BatchNorm2d(x),
                           nn.ReLU(inplace=True)]
                in_channels = x

        return nn.Sequential(*layers)
```

Figure 4: VGG16 Code Implementation.

구현된 모델의 파라미터는 총 33,617,987개를 가지며 16층으로 이루어져 있다.

2.3 Lab Experiments

모델 학습을 위한 train model 함수를 정의한다. 각 에포크(epoch)마다 모델을 학습하고, 학습 및 테스트 데이터에 대한 손실과 정확도를 계산하여 기록한다. 학습 중에는 옵티마이저를 사용해 가중치를 업데이트하며, 테스트 데이터에서는 모델을 평가 모드로 전환하여 성능을 측정한다. 최종적으로 학습 시간과 결과를 출력하며, 훈련된 모델과 손실 및 정확도 기록을 반환한다. 해당 기록을 그래프로 확인한 결과는 다음과 같다.

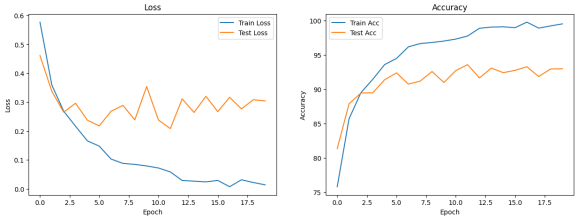

Figure 5: (a) VGG16 Loss (b) VGG16 Accuracy.

20Epoch 훈련 이후 Train Accuracy는 99.54% 최종 Test셋 3000개 이미지 기준 정확도는 93%를 기록하였고 훈련 시간은 111초가 소요되었다.

Figure 6: VGG16 VISUALIZE RESULT.

model을 eval모드로 전환하고 실제 그림과 예측결과를 같이 보면 다음과 같이 예측하는 것을 볼 수 있다.

2.4 Lab Conclusion

20에포크가 제약사항이었기 때문에 이후 학습은 적용하지 않았지만 이미 10에포크 쯤에서 Test셋 정확도와 오차는 수렴하는 것을 볼 수 있었고 이후엔 과적합으로 오히려 일반화 성능이 살짝 낮아지는 모습을 보였다. Lab에서는 단순화를 위해 드롭아웃을 적용하지 않았지만 FC Layer의 파라미터 수가 많기에 FC Layer에만 드롭아웃을 적용시켜 주더라도 성능향상을 기대할 수 있을 것 같다.

3 Deep Residual Learning for Image Recognition Paper Summary

3.1 Paper Abstract

Deep Residual Learning for Image Recognition에서 제시되는 ResNet은 위 VGG에서 증명한 네트워크의 깊이를 늘릴수록 좋아지는 것의 문제점을 해결하는 모델이다. 모델이 깊어질수록 학습을 하는 것이 어려워지는데, 이 논문에서는 Residual이란 요소를 도입함으로써 152개의 Layer를 쌓고도 VGG보다 복잡도가 낮았다. 이를 통해 ImageNet Dataset에 대해서 3.57 % Error를 달성하며 ILSVRC 2015 Classification Task에서 1등을 차지하였다.

3.2 Paper Introduction

앞서 VGG에서도 서술하였듯이 분류 모델에 있어 모델의 깊이는 중요하다. 하지만 이를 늘린다고 항상 성능이 좋아지는 것은 아니다. 깊이가 깊어지면 Vanishing or Exploding Gradients가 일어나 훈련이 멈추거나, OverFitting이 발생하여 일반화 성능이 떨어질수도 있다. 해당 논문에서는 Residual Learning, Identity Mapping by Shortcuts, Residual Network로 문제를 해결한다.

3.3 Paper Deep Residual Learning

3.3.1 Residual Learning

Residual Learning이란 잔차 학습으로, 입력을 직접 출력에 전달하는 Shortcut Connection으로 더 깊은 네트워크에서 학습을 원활하게 하는데 x가 입력 정보 그대로 더해지므로, 신호가 원활하게 전달되어 기울기 소실 문제(vanishing gradient problem)를 완화할 수 있다.

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x} \tag{1}$$

3.3.2 Identity Mapping by Shortcuts

Identity Mapping by Shortcuts은 Residual 블록에서 중요한 것은 항등 매핑(identity mapping)이다. 잔차를 학습하는 동안, 입력 x가 그대로 다음 레이어로 전달되는데, 이는 단축 경로(shortcut connection)에 의한 항등 매핑으로 표현된다. 항등 매핑을 통해 네트워크의 깊이가 깊어져도 기울기 소실이 줄어들고 학습이 더 안정적으로 진행된다.

$$\mathbf{y} = \mathbf{x} + \mathcal{F}(\mathbf{x}) \tag{2}$$

3.3.3 Network Architectures

ResNet은 다양한 깊이의 네트워크로 실험되었고, 18, 34, 50, 101, 152 층의 네트워크가 소개되었다. 이때, 각 잔차 블록은 2개의 3x3 또는 3개의 3x3 필터로 구성된다. 더 깊은 네트워크는 Bottleneck 구조를 사용하며, 이는 다음과 같은 잔차 함수를 사용한다.

$$\mathcal{F}(\mathbf{x}) = W_3\sigma(W_2\sigma(W_1\mathbf{x})) \tag{3}$$

각 W는 1x1, 3x3, 1x1 필터를 의미하며 시그마는 활성화 함수를 나타낸다. 이 구조를 이용하여 파라미터 수를 줄이면서 성능을 유지한다.

3.4 Paper Implementation

가장 왼쪽이 VGG19, 중간이 Plain Network이며 오른쪽이 Residual Network이다. ImageNet에 사용할 때, Image의 크기를 256x480으로 Resize하고 Random Crop을 통해서 224x224의 Image를 생성하였다. Horizontal Flip을 이용하여 좌우 반전을 줘서 Data Augmentation을 진행하였다. Convolution과 Activation Function 사이에 Batch Normalization을 추가하였고, Weight Intialization을 직접 손으로 진행했다. SGD Optimizer를 사용하였고, 256의 mini-batch를 사용하였다. Learning Rate는 0.1부터 시작하여 Error가 수렴할 때마다 10으로 나눠주었다. Weight Decay는 0.0001을 사용하였고, Momentum은 0.9를 사용하였다.

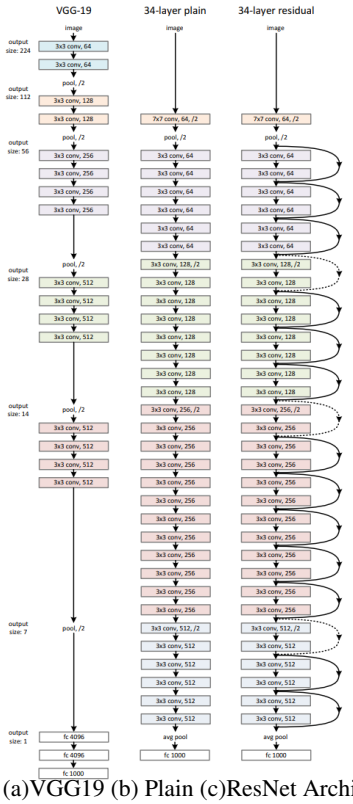

Figure 7: (a)VGG19 (b) Plain (c)ResNet Architecture.

3.5 Paper Experiments

3.5.1 ImageNet Classification

실험은 1000개의 Class로 이루어진 2012 ImageNet Classification Dataset으로 진행하였다. 34개의 Layer를 가진 Plain 모델은 18 Layer를 가진 Plain Network 보다 높은 Validation Error를 보였는데, 이를 통해 깊어질수록 최적화가 어렵다는 것을 알 수 있다. Residual Network 또한 18개, 34개 Layer를 비교하였는데, 34개의 Layer를 가진 모델이 18개의 Layer를 가진 모델보다 Error가 작았다. 여기서 모델을 더 깊게 만들려고 하면, 학습 시간이 오래 걸린다. 이를 위해 Bottleneck 구조를 제안하였는데, 각 Residual Function마다 2개의 Layer 대신 3개의 Layer를 사용하였고, 이는 각각 1x1, 3x3, 1x1 Convolution으로 이루어져 있다. 1x1 Layer는 Dimension을 줄였다가 키우는 역할을 한다. 해당 구조는 Layer가 1개 더 많음에도 불구하고 Time Complexity는 둘이 비슷하게 측정된다. 여기서 Identity Shortcut이 중요한 역할을 한다.

이 구조를 도입하여 더 깊은 101, 152개 Layer를 가진 모델을 만들었고 Depth가 크게 증가했지만 VGG16/19보다 더 적은 복잡도를 가지고 34Layer모델보다 높은 정확도를 보였다.

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

Figure 8: Comparisons with State-of-the-art Methods.

3.5.2 CIFAR-10 and Analysis

ImageNet 뿐만 아니라 CIFAR-10 Dataset에서도 ResNet이 깊어질수록 좋은 성능을 보이는것이 확인되었으며 학습률 조정으로 더욱 높은 성능을 보였다.

3.6 Paper Conclusion

연구자들은 모델의 Layer를 계속해서 늘리면 더 좋은 성능은 얻는 것을 확인하고자 1202개의 Layer를 가진 모델을 실험하였는데, 최적화 과정에 문제는 없었지만 110 Layer 모델보다 성능이 좋지 않았다. 이는 데이터셋보다 모델이 너무 복잡하여 Overfitting 된 것으로 ResNet을 사용해도 깊이를 무한정 늘린다고 성능이 올라가진 않았다. 이 모델은 분류 문제뿐 아니라 다른 Recognition Task에도 유효함을 보였는데, 좀 더 복잡한 문제인 Object Detection에서도 좋은 성능을 보였다.

training data	07+12	07++12
test data	VOC 07 test	VOC 12 test
VGG-16	73.2	70.4
ResNet-101	**76.4**	**73.8**

Table 7. Object detection mAP (%) on the PASCAL VOC 2007/2012 test sets using **baseline** Faster R-CNN. See also Table 10 and 11 for better results.

metric	mAP@.5	mAP@[.5, .95]
VGG-16	41.5	21.2
ResNet-101	**48.4**	**27.2**

Table 8. Object detection mAP (%) on the COCO validation set using **baseline** Faster R-CNN. See also Table 9 for better results.

Figure 9: Object Detection on PASCAL and MS COCO.

4 Lab 1-2. Resnet50 Implementation

4.1 Lab Constraint

4.1.1 Data

사용하는 데이터는 이전 VGG16 Lab의 구현시에 사용했던 데이터셋과 동일한 CIFAR-10의 3클래스 축소본을 사용한다.

4.1.2 Architecture

보틀넥 블록이 포함된 Resnet50을 구성한다. CIFAR-10 데이터셋의 이미지 크기가 너무 작기에 논문의 첫번째 conv1레이어와 구조를 다르게 설정하고 최대 풀링 레이어를 제거한다. 또한 최대 풀링 레이어 대신 다운샘플링을 위한 스트라이드 컨볼루션을 구현한다. 활성화 함수로는 ReLU를 사용하고 단순화를 위해 Drop out은 적용하지 않는다. 학습에 적합하도록 논문에서 사용하지 않았던 모든 컨볼루션 층 이후에 배치 정규화 층을 적용한다.

4.1.3 Loss function and Optimizing

오차 함수로는 출력값과 실제 정답값의 교차 엔트로피 손실을 사용하고, 단순화를 위해 기본 가중치 초기화를 적용한 학습률 = 1e-2, momentum = 0.9, weight decay = 5e-4를 사용하는 SGD알고리즘을 Optimizer로 사용한다. 훈련은 15Epoch를 진행하고 학습 속도 스케줄링은 적용하지 않는다.

4.2 Code Implementation

데이터 준비와 전처리 또한 이전 VGG16 Lab 구현시에 활용했던 torch의 dataset 구조와 transform Compose를 이용해 Random Crop, RandomHorizontalFlip을 적용하고 이미지의 RGB 평균값 (0.4914, 0.4822, 0.4465)와 표준편차 (0.2023, 0.1994, 0.2010)를 이용해 정규화한다.

다음 Figure-10는 제약사항에 명시된 Bottle neck Block을 구현한 코드이다. 이 코드는 ResNet block 클래스를 정의한 것으로, ResNet에서 사용되는 기본 블록을 구현하고 있다. 블록은 1x1, 3x3, 1x1 합성곱 계층과 배치 정규화, ReLU 활성화 함수로 구성된다. 입력과 출력 차원이 다를 경우, identity 연결을 위해 1x1 합성곱 계층을 통해 차원을 맞춘다. 마지막으로, skip connection을 통해 입력값과 출력값을 더하고 ReLU 활성화를 적용한다. 이를 통해 잔차 학습을 구현한다.

```python
class ResNet_block(nn.Module):
    def __init__(self, in_c, intra_c, out_c, down_sample=False, stride=1):
        super(ResNet_block, self).__init__()
        self.expand = in_c != out_c
        self.down_sample = down_sample

        # 1x1 conv
        self.conv1 = nn.Conv2d(in_c, intra_c, kernel_size=1, stride=stride, padding=0, bias=False)
        self.bn1 = nn.BatchNorm2d(intra_c)

        # 3x3 conv
        self.conv2 = nn.Conv2d(intra_c, intra_c, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(intra_c)

        # 1x1 conv
        self.conv3 = nn.Conv2d(intra_c, out_c, kernel_size=1, stride=1, padding=0, bias=False)
        self.bn3 = nn.BatchNorm2d(out_c)

        # Dimension converter for identity (skip connection)
        if self.expand:
            self.convert_id = nn.Sequential(
                nn.Conv2d(in_c, out_c, kernel_size=1, stride=stride, padding=0),
                nn.BatchNorm2d(out_c)
            )

    def forward(self, x):
        identity = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = F.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)
        out = F.relu(out)

        out = self.conv3(out)
        out = self.bn3(out)

        if self.expand:
            identity = self.convert_id(x)

        # Skip connection
        out += identity
        out = F.relu(out)
```

Figure 10: Bottleneck Block Code Implementation.

Figure-11 코드는 CIFAR-10 데이터셋에 맞게 수정된 ResNet 모델을 구현한 것이다. init block은 3x3 합성곱 계층과 배치 정규화, ReLU 활성화를 포함하며, CIFAR-10의 작은 이미지 크기에 맞게 최대 풀링 계층을 제거했다. 네트워크는 4개의 ResNet 블록으로 구성되어 있으며, 각 블록은 다운샘플링과 잔차 연결을 처리한다. 마지막으로, Adaptive Average Pooling을 적용하여 고정된 출력 크기로 변환한 후, 완전 연결 계층에서 3개의 클래스로 분류한다. make layers 함수는 주어진 구성에 따라 ResNet 블록을 생성한다.

```python
class ResNet(nn.Module):
    def __init__(self):
        super(ResNet, self).__init__()
        # Initial layer for CIFAR-10 (3x3 conv, no max pooling)
        self.init_block = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True)
        )

        # Bottleneck layers
        self.ResBlock1 = self._make_layers(64, 64, 256, cfg[0], down_sample=False)
        self.ResBlock2 = self._make_layers(256, 128, 512, cfg[1], down_sample=True)
        self.ResBlock3 = self._make_layers(512, 256, 1024, cfg[2], down_sample=True)
        self.ResBlock4 = self._make_layers(1024, 512, 2048, cfg[3], down_sample=True)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.classifier = nn.Linear(2048, 3) # 3 classes for CIFAR-10

    def forward(self, x):
        out = self.init_block(x)
        out = self.ResBlock1(out)
        out = self.ResBlock2(out)
        out = self.ResBlock3(out)
        out = self.ResBlock4(out)
        out = self.avgpool(out)
        out = out.view(out.size(0), -1)
        out = self.classifier(out)

        return out

    def _make_layers(self, in_c, intra_c, out_c, num_block, down_sample):
        layers = []
        # First block with possible down-sampling
        layers.append(ResNet_block(in_c, intra_c, out_c, down_sample=down_sample, stride=2 if down_sample else 1))
        # Rest blocks without down-sampling
        for _ in range(1, num_block):
            layers.append(ResNet_block(out_c, intra_c, out_c, down_sample=False))
        return nn.Sequential(*layers)
```

Figure 11: Resnet50 Code Implementation.

구현된 모델의 파라미터는 총 23,510,339개를 가지며 50층의 깊은 네트워크임에도 VGG16의 파라미터 수보다 적다.

4.3 Lab Experiments

VGG16 Lab에서 작성한 모델 학습을 위한 train model 함수를 그대로 사용할 수 있다. 각 에포크(epoch)마다 모델을 학습하고, 학습 및 테스트 데이터에 대한 손실과 정확도를 계산하여 기록한다. 학습 중에는 옵티마이저를 사용해 가중치를 업데이트하며, 테스트 데이터에서는 모델을 평가 모드로 전환하여 성능을 측정한다. 최종적으로 학습 시간과 결과를 출력하며, 훈련된 모델과 손실 및 정확도 기록을 반환한다. 해당 기록을 그래프로 확인한 결과는 다음과 같다.

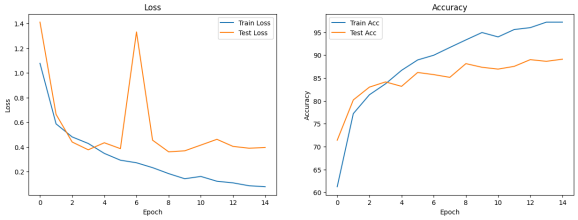

Figure 12: (a) resnet50 Loss (b) resnet50 Accuracy.

15Epoch 훈련 이후 Train Accuracy는 97.20% 최종 Test셋 3000개 이미지 기준 정확도는 89.1%를 기록하였고 훈련 시간은 2분 46초가 소요되었다.

Figure 13: resnet50 VISUALIZE RESULT.

model을 eval모드로 전환하고 실제 그림과 예측결과를 같이 보면 다음과 같이 예측하는 것을 볼 수 있다.

4.4 Lab Conclusion

15에포크가 제약사항이었기 때문에 이후 학습은 적용하지 않았지만 정확도와 오차 그래프의 수렴 진행 형태를 보아 좀 더 많은 Epoch를 이용해 학습시켜면 좋은 결과가 나올 것 같다. 15epoch기준으로는 VGG16보다 정확도가 낮지만 더 깊은 신경망인만큼 여러 Epoch를 학습시키는 것이 중요할 것 같고 논문의 접근 방식과 아이디어를 따라 50층임에도 불구하고 16층의 VGG보다 파라미터 수가 약 1천만개 적다는 것이 매우 놀라웠다. 또한 이 모델은 깊은 모델이기에 좀 더 복잡한 Task를 진행하는 것이 결과치가 더 좋게 나올 것 같다. 오히려 3클래스로 축소한 데이터셋이기에 데이터에 비해 신경망이 복잡하여 Underfitting이 살짝 일어나는 것 같다.

[1] Ilya Sutskever Geoffrey E. Hinton Krizhevsky, Alex. Imagenet classification with deep convolutional neural networks. In *Communications of the ACM*, volume 60.6, pages 84–90, 2012.