컴퓨터비전 최종 프로젝트

한경빈

Computer Vision Final Project

Han Gyeongbin

요 약

CNN의 대표적인 세 모델 VggNet, ResNet, MobileNet-v2의 논문을 차례대로 분석하고 pytorch 구현 코드를 이해하며 직접 학습까지 시켜보며 Classification 모델의 발전 과정에서 어떻게 다양한 아이디어를 발전시키고 구현해 왔는지 학습한다.

Abstract

Analyze the papers of CNN's three representative models VggNet, ResNet, and MobileNet-v2 sequentially, understand pytorch implementation code, and learn how have been developed and implemented various ideads in the evolution of the Classification models.

Key words

Convolutional Neural Network, VggNet, ResNet, MobileNet-v2, pytorch

I. 서 론

ILSVRC 2014 대회에서 2위를 차지한 VggNet 모델을 베이스로 하여 수많은 CNN 모델들이 만들어지고 발전되어 왔다. 본 프로젝트에서는 VggNet과 이후에 등장한 ResNet과 MobileNet-v2에 대하여 이해한 후 pytorch를 통해 논문의 모델을 구현하고, 학습하는 과정을 경험하여 이후 컴퓨터비전 분야의 기초 배경지식을 공부하는 것을 목표로 한다.

실험 과정으로는 먼저 논문에서 소개하는 기술에 대해 이해하고, 네트워크 구성, 성능, 장단점에 대해서 학습 후 pytorch로 구현하는 과정을 익히고 직접 colab 환경에서 테스트해본 후 결과를 확인한다.

II. VggNet

VggNet은 ILSVRC 대회에서 2위를 차지한 모델로, 단순한 구조와 쉬운 설계로 인해 당시 1위를 차지한 GoogleNet보다도 더 주목받은 모델이다.

VggNet 논문에서 가장 중요한 쟁점으로 생각한 것은 네트워크의 깊이이며 깊이가 깊을수록 모델이 더 좋은 성능을 보인다는 것을 입증하였다. 필터의 경우는 아주 작은 3x3 conv layer만 사용하였는데 이러한 필터를 여러 층으로 쌓은 단순한 구조를 지니고 있다.

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					

그림 1. VggNet 논문의 구조도

논문의 구조를 보면 필터는 3x3 conv filter로 고정하고, 레이어의 깊이만 차이를 두었음을 확인할 수 있다.

필터의 크기를 3x3으로 고정한 이유는 pooling 없이 3x3 필터로 3번 convolution하는 것과 7x7 필터로 한번 convolution하는 것은 같은 성능을 보이기 때문이다. 또한 이 과정에서 3번의 convolution을 진행하기 때문에 비선형 함수인 ReLU를 3번 쓸 수 있고, 다음과 같이 학습에 필요한 파라미터의 수가 감소하는 효과를 얻을 수 있다.

$$(3 * (3^2C^2) < 7^2C^2$$

주로 VggNet을 활용할 때에는 그림1의 D나 E 모델을 사용하며, Vgg16과 Vgg19의 성능은 근소한 차이를 보이므로 연산량이 상대적으로 적은 Vgg16이 좀 더 선호되는 경향이 있다. 논문에서는 약 92%의 정확도로 소개되고 있다.

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

ConvNet config. (Table 1)	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**

그림 2. 논문에 소개된 VggNet의 성능

VggNet의 장점으로는 이해하기 쉬운 구조를 지니고 간단한 구조임에도 좋은 성능을 보이기 때문에 추후 등장한 많은 파생 CNN모델의 기본 틀 역할을 한다는 점이다. 단점은 fully connected layer를 사용하기 때문에 파라미터를 굉장히 많이 사용해 연산량이 크다는 점이다.

III. ResNet

ResNet은 VggNet을 기반으로 더 발전시킨 모델로 ILSVRC 2015에서 우승을 차지했다.

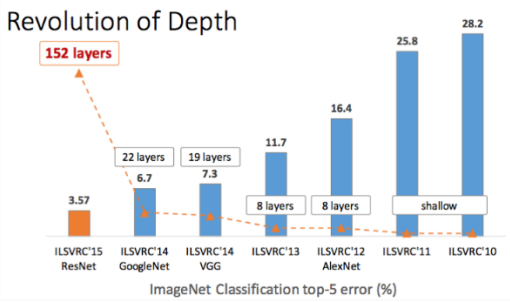

그림 3. CNN 모델 네트워크 깊이의 변화

ResNet의 가장 큰 특징은 망이 VggNet이나 GoogleNet에 비해서 훨씬 깊어짐에 있다. 또한 그러면서도 top-5 error를 기존 연구에 비해 절반 정도로 감소시킨 성능을 보인다.

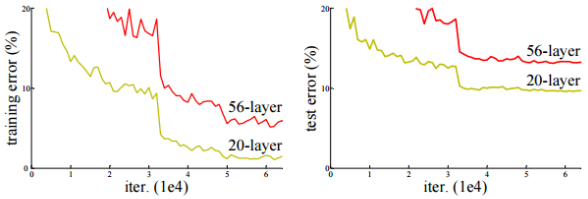

Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

그림 4. 네트워크 깊이에 따른 에러 비교

ResNet의 핵심 아이디어는 VggNet에서 출발하였는데, VggNet에서 설명한 더 깊은 네트워크일수록 더 좋은 성능을 보임을 더 나아가서 20 레이어 이상의 훨씬 깊은 레이어에서도 더 좋은 성능을 보이는지 확인해보았다.

그러나 그림 4에서 확인할 수 있듯 56개의 레이어를 쌓은 네트워크에서 오히려 오차가 커짐을 확인하였다. 즉, 기존의 방식 그대로 깊이만 더 깊게 한다고 해서 성능이 무한정 좋아지지 않음을 알게 되었다.

그 이유는 Gradient Vanishing 때문으로, 레이어가 깊어질수록 미분을 점점 많이 하기 때문에, Backpropagation을 수행해도 점점 미분값이 작아져 output에 영향을 별로 미치지 못하기 때문이다. 즉 이러한 Degradation으로 인해 학습이 잘 되지 않음을 의미한다.

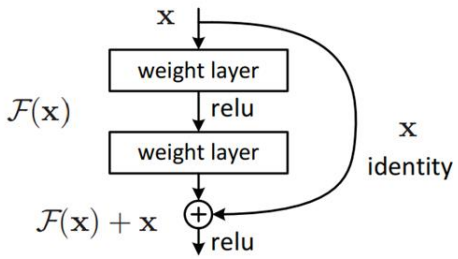

Figure 2. Residual learning: a building block.

그림 5. Skip/Shortcut connection

연구진들은 그러한 이유 때문에 그림 5와 같은 Residual block을 고안하였다. 기존 레이어와 차이는 입력값을 출력값에 Shortcut을 통해 더해주는 것뿐인데, 기존의 레이어는 입력값 x를 y로 매핑하기 위

한 함수 H(x)를 구하는 것이 목적이라면, Shortcut connection이 추가된 레이어는 함수를 통한 F(x)와 x를 더한 값 F(x)+x를 최소화하는 것을 목표로 한다. 여기서 x는 불변하므로 F(x)를 0에 가깝게 만드는 것과 같다. 이때 F(x)+x =H(x)라고 한다면 F(x) = H(x) – x 이므로 H(x) – x를 최소로 만드는 것과 동일한 의미를 지닌다. 이때 H(x) – x를 잔차(Residual)이라고 한다.

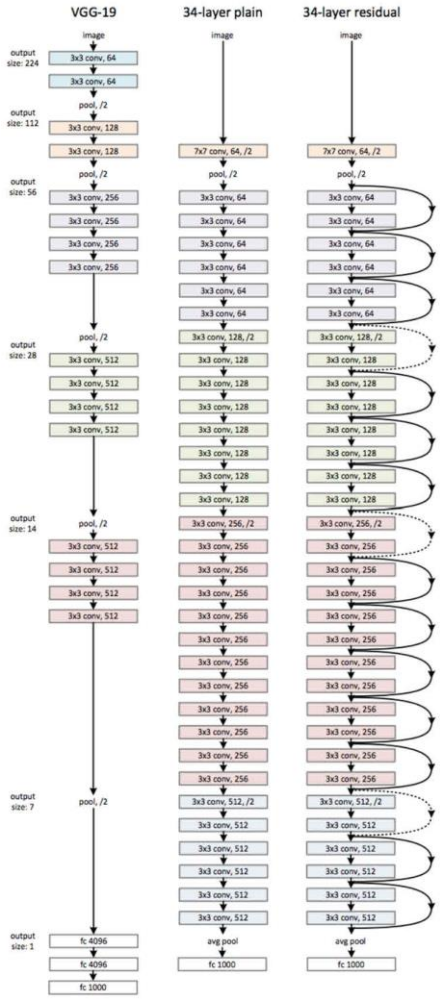

그림 6. VggNet과 ResNet의 비교

ResNet은 기본적으로 VggNet의 구조를 기초로 하여 3x3 conv 레이어로 이루어졌으며 더 깊은 망을 지닌 plain 모델을 구성한 후, Shortcut connection을 추가한 Residial 모델을 편성하였다.

또한 50 레이어가 넘어가는 더 깊은 네트워크의 경우 1x1 conv layer를 활용한 bottleneck 구조를 사용하여 연산시간이 감소하도록 하였다.

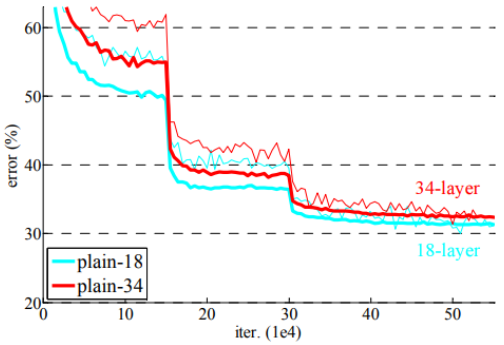

그림 7. Plain network

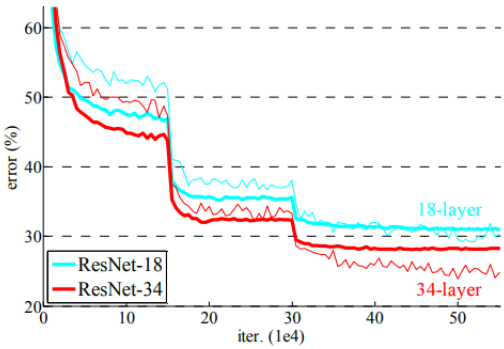

그림 8. Residual network

Plain block으로 이루어진 그림 7의 결과와 Residual block을 활용한 결과인 그림 8을 비교하면, Plain network의 경우 깊어질수록 에러가 커짐을 확인할 수 있으며, Residual network의 경우 더 깊은 모델에서 에러가 작아짐을 볼 수 있다.

즉 ResNet은 기존 VggNet보다 더 깊은 모델을 구성하면서도 Residual Connection을 사용해 복잡도와 성능을 개선하였으며, 역시 구현 또한 간단하고 논문에서 소개된 152층을 지닌 더 깊은 모델일수록 더 좋은 성능을 보임을 연구하였다.

하지만 깊이가 그보다 더 깊어져 1000레이어 이상이 될 경우 다시 Degradation이 발생해 오차가 커짐 역시 확인하였으며 이는 후속 연구 과제로 전달되었다.

IV. MobileNet-v2

위의 ResNet까지는 네트워크의 크기를 키워가며 최대한의 성능을 뽑아내는 것을 중점적으로 연구했다면 MobileNet의 경우는 Xception 논문에서 제안되었던 파라미터를 효율적으로 활용해 성능을 산출하는 것에 집중하였다. 특히 Depthwise Separable Convolution을 사용하여 경량화한 MobileNet-v1에 더하여 MobileNet-v2는 Inverted Residual 구조를 사용해 메모리 사용량을 더 최적화하였다.

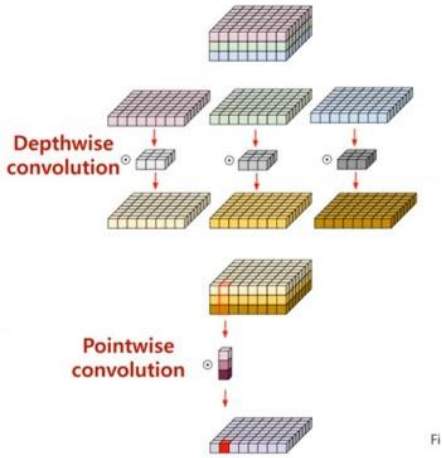

그림 10. Depthwise Separable Convolution

Depthwise Separable Convolution이란 기존의 convolution과 비교하여 인풋을 채널별로 나누어 각각을 3x3 필터를 사용해 convolution 연산을 한 다음 그 결과를 다시 합쳐 1x1 convolution을 해주는 것이다. 이렇게 같은 결과를 만들어내지만 연산량을 줄일 수 있다. 그 결과로 모바일 환경에서도 사용하기에 적절한 네트워크를 만들 수 있다는 장점이 있다.

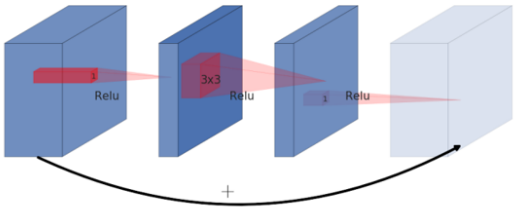

그림 11. Residual Block

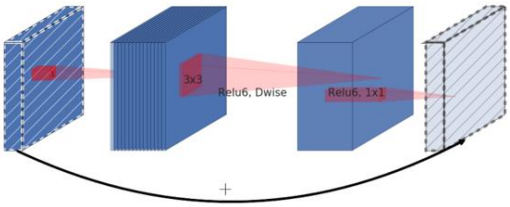

그림 12. Inverted Residual Block

또한 v2에 들어서 기존의 Residual Block과 다른 형태인 Inverted Residual Block을 사용하였다. 그림 11의 기존 wide->narrow->wide 형태인 Residual block은 중간의 좁은 형태가 bottleneck 구조를 만들어 준다. 즉 처음에 wide한 채널이 들어오면 1x1 convolution을 활용해 채널을 줄여 3x3 convolution으로 연산 후 다시 skip connection과 합치기 위해 원래의 사이즈로 복구한다.

그러나 MobileNet-v2에서 제안된 Inverted Residual Block은 정반대로, ReLU를 거치지 않은 linear bottleneck에서, 다시 wide한 채널로 바꾼 후 ReLU 연산을 한 결과를 다시 차원 축소하는 형식으로 수행된다. 이렇게 구성하더라도 이미 narrow한 채널에 필요한 정보는 저장되어 있다고 가정하였기 때문에, 메모리 사용량을 줄이면서도 필요한 만큼의 학습량을 확보할 수 있다.

Size	MobileNetV1	MobileNetV2	ShuffleNet (2x,g=3)
112x112	64/1600	16/400	32/800
56x56	128/800	32/200	48/300
28x28	256/400	64/100	400/600K
14x14	512/200	160/62	800/310
7x7	1024/199	320/32	1600/156
1x1	1024/2	1280/2	1600/3
max	1600K	**400K**	600K

그림 13. 연산량 비교

Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet (1.5)	71.5	**3.4M**	292M	-
ShuffleNet (x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MobileNetV2	**72.0**	**3.4M**	**300M**	**75ms**
MobileNetV2 (1.4)	**74.7**	6.9M	585M	**143ms**

그림 14. 파라미터 수와 시간 비교

이러한 연구의 결과로 MobileNet-v2는 적은 연산량으로 인해 학습이 빠르면서도, 충분한 정확도를 확보한 경량화된 모델을 만들어낼 수 있었다.

이렇게 메모리 효율의 측면에서 진일보한 MobileNet-v2는 모바일 환경을 타겟으로 유용하다는 장점이 있어 이후에도 경량 모델이 필요한 상황에서 자주 사용되고 있다. 단점으로는 정확도만 보자면 타 연구들에 비해서는 크게 향상시킨 점은 없다고 할 수 없다는 점을 들 수 있다.

V. pytorch

위의 세 논문에서 소개된 모델을 직접 테스트해보기 위하여 여러 레퍼런스를 참고하여 pytorch 코드를 직접 실행시켜보았다.

pytorch에 대해 이해가 부족하였기 때문에 최대한 많은 코드를 살펴보며 논문의 구조가 실제 코드로 어떻게 옮겨지는 지에 대해 학습하는 것에 집중하였다.

직접 학습한 결과는 논문에 소개된 정확도보다 일부 높은 성능을 보였는데, 이는 데이터 전처리 등 추가적인 customization이 가미되었기 때문으로 사료된다.

세 모델의 학습 대상은 cifar-10으로 하였고, 전처리, 옵티마이저, 스케줄러 등은 모두 똑같이 설정하였으며

epoch수 역시 200회로 통일하여 결과를 내보았다. 직접 train해본 세 모델의 성능은 다음과 같다.

```
Epoch: 199
train
Loss: 0.001 | Acc: 99.984% (49992/50000)
test
Loss: 0.302 | Acc: 93.800% (9380/10000)
```

그림 15. VggNet-19 (Batch Normalization)

먼저 VggNet-19의 경우 약 93.8%의 좋은 성능을 보였는데, 이는 논문에는 없는 Batch Normalization 레이어를 추가하여 네트워크를 구성한 것과 Random Crop, Flip 등의 데이터 전처리를 통해 더 풍성한 train set을 만든 효과로 생각된다.

```
Epoch: 199
train
Loss: 0.002 | Acc: 99.996% (49998/50000)
test
Loss: 0.173 | Acc: 95.470% (9547/10000)
```

그림 16. ResNet18

ResNet18의 경우 약 95%의 정확도로 매우 좋은 성능을 보였지만 실제 학습 속도는 더 느려 ResNet18보다 더 큰 ResNet50 등의 모델은 colab 환경에서 학습에 어려움이 있었다.

```
Epoch: 199
train
Loss: 0.056 | Acc: 98.226% (49113/50000)
test
Loss: 0.253 | Acc: 92.430% (9243/10000)
```

그림 17. MobileNet-v2

MobileNet-v2의 경우 다른 모델들에 비해 더 낮은 성능을 보였다. 대신 실제 epoch을 매번 확인한 결과 위 모델들에 비해 더 빠르게 학습이 됨을 확인할 수 있었다.

VI. 결론

각 논문을 읽으며 컴퓨터비전의 기초 Classification 모델에 대해 학습하고 직접 train해보는 과정을 통해 새로운 네트워크를 꾸준히 발전시켜온 아이디어를 얻는 과정과 실제 구현 과정을 학습할 수 있는 기회였으며, 또한 pytorch를 처음 다뤄보면서 수많은 레퍼런스를 참고하고 어떤 방식으로 수식적으로만 다뤄진 논문의 내용을 실제로 테스트하는지 등을 공부할 수 있었다.

특히 VggNet이라는 단순한 구조를 지닌 네트워크부터 ResNet, MobileNet-v2를 차례대로 학습하며 점점 진화하는 모델들의 순서를 하나하나 살펴볼 수 있었고 이후 Object Detection, Segmentation 등의 분야에서 범용적인 Backbone으로 쓰이는 주요 Classification에 대한 기반지식을 학습하였다.

아쉬운 점으로는 논문 이해와 학습에는 논문을 리딩하고 여러 레퍼런스를 읽어가며 천천히 각 모델들을 알아갈 수 있었으나, 실제 pytorch 구현에 대해서는 대부분 이미 만들어진 코드를 그대로 참고하였음에 그쳤다는 점이다. 추후 pytorch에 대한 추가적인 학습을 통해 다루는 법을 익혀 다른 모델의 경우에는 직접 구현해보면서 이해하고 싶다.

VII. 부록

깃허브 주소

https://github.com/hanabzu/Computer-Vision-Project

VIII. 참고문헌

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. Deep Residual Learning for Image Recognition. arXiv preprent arXiv:1512.03385, 2015.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. arXiv preprint arXiv:1801.04381. 2019.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shift. arXiv: 1502.03167v3, 2015.

https://www.researchgate.net/figure/Ball-chart-reporting-the-Top-1-and-Top-5-accuracy-vs-computational-complexity-Top-1-and_fig1_328509150

http://cs231n.stanford.edu/

https://github.com/pytorch/vision/blob/6db1569c89094cf23f3bc41f79275c45e9fcb3f3/torchvision/models/vgg.py#L24

https://github.com/kuangliu/pytorch-cifar

https://junklee.tistory.com/111

https://bskyvision.com/644

https://aistudy9314.tistory.com/25

https://dev-jm.tistory.com/7

https://gaussian37.github.io/dl-concept-mobilenet_v2/

https://velog.io/@woojinn8/LightWeight-Deep-Learning-7.-MobileNet-v2