GoogLeNet 리뷰

Reference

- Going deeper with convolutions, Christian Szegedy et al.
- Network in Network
- https://ko.coursera.org/lecture/convolutional-neural-networks/networks-in-networks-and-1x1-convolutions-ZTb8x
- https://datascienceschool.net/view-notebook/8d34d65bcced42ef84996b5d56321ba9/
- https://medium.com/coinmonks/paper-review-of-googlenet-inception-v1-winner-of-ilsvlc-2014-image-classification-c2b3565a64e7
- https://kangbk0120.github.io/articles/2018-01/inception-googlenet-review

GoogLeNet을 리뷰하기 이전에 NIN(Network in Network)을 먼저 살펴보도록 한다. GoogLeNet은 NIN의 아이디어를 많이 이용했기 때문이다.

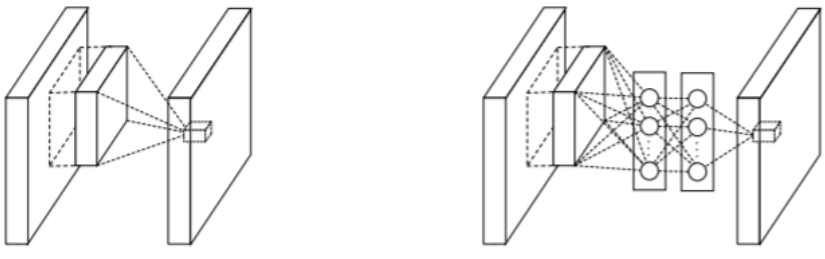
Overview of NIN

기존의 convolution layers는 linear convolutional filters을 통해 feature maps을 만들고 activation functions을 통해 nonlinearity을 적용한다. 이는 아래와 같이 계산된다.

$$f_{i,j,k} = max(w_k^T x_{i,j}, 0) \qquad (1)$$

여기서 (i, j)는 feature map의 pixel index, $x_{i,j}$는 위치 (i, j)에 center된 input patch, 그리고 k는 feature map의 channels index이다.
이러한 linear convolution은 latent concepts가 linearly separable하다면 효율적이다. 즉, 데이터의 분포가 linear하다면 괜찮은데, 사실 non-linear한 분포가 더 많을 것이다. 바로 이럴 때 linear convolution은 한계를 가지므로 뭔가 non-linearity을 추가해야할 필요성이 대두됐다.

NIN은 이러한 필요성에 의해서 나온 신경망 모델이다. 말 그대로 Network 안에 Network가 또 있는 것이다. 그러면 안에 있는 Network는 무엇일까? 딥러닝에서 흔히 non-linearity을 추가하기 위해서 사용하는 MLP(Multi Layer Perceptron)이 바로 그것이다.



(a) Linear convolution layer (b) Mlpconv layer

위 그림에서 (a)는 linear convolution layer, (b)는 논문에서 제안하는 MLP Layer이다. (b)을 보면 (a)와 같이 linear convolution layer을 수행한 후, 두 개의 layer을 가지는 MLP을 추가하였다.

그런데 NIN 논문에서 MLP convolution Layers을 아래와 같이 제시하였다.

$$f_{i,j,k_1}^1 = max(w_{k_1}^{1\,T} x_{i,j} + b_{k_1}, 0)$$

...

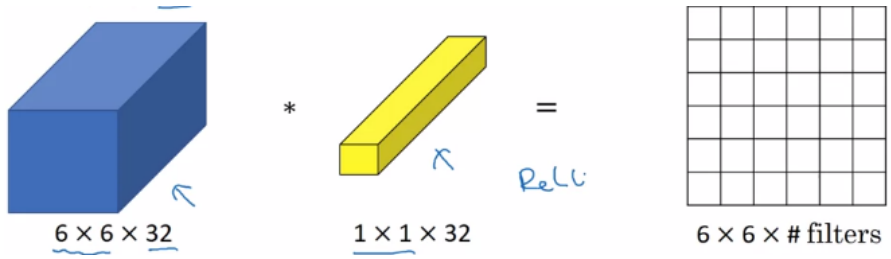
$$f_{i,j,k_n}^n = max(w_{k_n}^{n\ T} f_{i,j}^{n-1} + b_{k_n}, 0) \qquad (2)$$

즉, 하나의 MLP convolution Layers에 n개의 conv layer가 있는 듯한 느낌인데, 그 형태가 (1)과 거의 동일하다. 앞서 MLP convolution layer는 MLP를 이용한다고 했고, 위 그림 (b)에서도 FC를 하는 것으로 나와 있는데 왜 linear convolution layer을 여러번 하는 것으로 나왔을까? 이에 대해서 이해하기 위해 1×1 convolution filter에 대해서 살펴보자.

1×1 convolutional filters

1×1 conv filters를 2차원 tensor에 적용한다고 생각해보자. 이는 곧 아주 단순한 dot product이 될 것이고 아래와 같이 계산된다.

근데 만약 3차원 tensor에 적용한다고 하면 얘기가 달라진다.



이는 곧, $6 \times 6 \times 32$ tensor에서 36번($= 6 \times 6$)에 걸쳐서 1×1 conv filter을 적용하는 것이다. 즉, $6 \times 6 \times 32$ tensor는 36개의 $1 \times 1 \times 32$ tensor로 구성되어있고 이 tensor와 1×1 conv filter끼리 dot product을 하고 ReLU을 적용하는 것이다.

만약 $1 \times 1 \times 32$ conv filter의 갯수가 1개라면 위 그림의 오른쪽과 같이 6×6 tensor output이 나올 것이다. 그런데 만약 이 filter의 갯수가 m개 라면 6×6 tensor output이 m개가 나오는 것이다. 즉, m개의 $1 \times 1 \times 32$ conv filter을 적용하면 $6 \times 6 \times m$ tensor output을 얻는다. 여기서 1×1 conv filter의 특징 하나를 도출할 수 있다. **a) 바로 dimension reduction(augmentation)을 수행할 수 있다는 것이다.** 만약 m이 input tensor의 channel보다 작다면, dimension reduction을 하는 것이고 크다면 dimension augmentation을 하는 것이다.

b) 이를 이미지의 channel 입장에서 cross channel pooling 이라고 한다. 말 그대로, pooling을 채널을 넘어서 하겠다는 뜻이다.

위 그림을 보면 세 개의 channel을 가지는 tensor에서 같은 위치의 3개의 값들 중 가장 큰 값을 output tensor로 반환한다.

c) 또한 1×1 conv filter을 적용하는 것은 fully connected layer와 동일하다. 이에 대한 자세한 내용은 여기를 참고하자.

따라서 MLP를 쌓아 올린 MLP convolution layer에 대한 수식적 표현은 **c)**에 의해서 마치 그냥 일반 convoltion layer을 쌓은 수식 (2)로 도출되는 것이다.

여기서 NIN은 왜 MLP를 쌓은 것인지 이유에 대해서 다시 생각해보자. 결국 non-linearity 관계를 표현하기 위함인데, MLP를 쌓은 것이 1×1 conv filter을 적용하는 것과 동일하고 결국 cross channel pooling을 한 것과 동일하기 때문에 non-linearity을 잘 나타낼 수 있다고 한다.

Conclusion of NIN

NIN은 그 자체로는 많이 쓰이지 않지만, 1×1 conv filter의 개념을 처음으로 소개했다는 의의를 가진다. 1×1 conv filter는 **a), b)** 의 장점을 가지고 있고 또 파라미터의 갯수도 줄여주기 때문이다. 다음으로 소개할 GoogLeNet은 이러한 장점을 이용하기 위해서 1×1 conv filter을 적극 활용한다.

Motivation of Inception Module

깊은 신경망의 성능을 높이는 가장 확실한 방법은 size를 늘리는 것이다. 여기서 size란 신경망의 depth, 즉 layers의 수와 width, 즉 각 layer에서 units의 수를 늘리는 것을 뜻한다. 만약 양질의 훈련 데이터가 있다면 이 방법은 좋은 모델을 위한 가장 쉽고 안전한 방법이다.

하지만 이와 같은 방법에는 두 가지 단점이 있다. 깊은 신경망과 넓은 alyer는 더 많은 파라미터를 요구하고 이는 과적합, gradient vanishing 으로 이어질 수 있으며 특히 훈련 데이터 세트가 제한적인 경우 그렇다. 실제 상황에서, 양질의 훈련 데이터를 확보하기 어렵기 때문에 이와 같이 무작정 사이즈를 늘리는 것이 능사가 아니다.

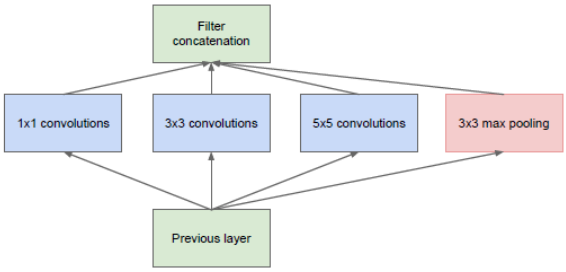
이를 해결하기 위해 sparse connectivity가 제시되었다. 지금까지 배운 convolution 연산은 Densely 연결 되어 있다. 이를 높은 관련성(correlation)을 가진 노드들 끼리만 연결하도록, 다시 말해 노드들 간의 연결을 sparse 하도록 바꾼다면, 연산량과 파리미터 수가 줄고 따라서 overfitting 또한 개선 될 것이라고 생각했다. Fully connected network에서 사용하는 Dropout과 비슷한 기능을 할 것이라고 본 것이다.
하지만, 실제로는 Dense matrix연산 보다 Sparse Matrix연산이 더 큰 Computational resource를 사용한다. LeNet 시절의 CNN만 하더라도 Sparse 한 CNN 연산을 사용했다. 이 후, 연산을 병렬처리 하기위해서 Dense Connection을 사용했고, Dense Matrix의 연산 기술이 발전했다. 반면, Sparse Matrix연산은 그 만큼 발전하지 못했고, Dense Matrix 연산 보다 비효율적이다. 따라서, 위의 목적을 달성하기 위해서 Sparse connectivity를 사용하는 것은 해결방안이 될 수 없었다.

여기에서, 구글이 고민한 것은 어떻게 노드 간의 연결을 줄이면서(Sparse connectivity), 행렬 연산은 Dense 연산을 하도록 처리하는가 였다. 그리고, 이 고민의 결과가 Inception module 이다.

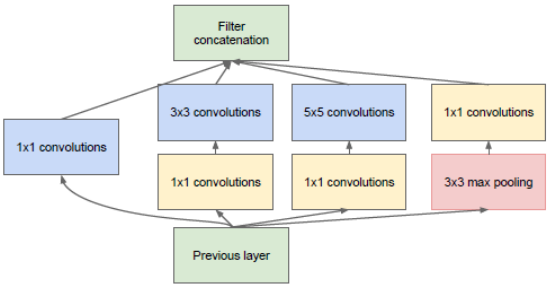
논문에서 Inception 아키텍쳐의 main idea를 아래와 같이 얘기하는 부분에서 이를 엿볼 수 있다.

*The main idea of the Inception architecture is based on finding out how an optimal local **sparse structure** in a convolutional vision network can be approximated and covered by readily available **dense components.***

Architecture of Inception Module



(a) Inception module, naïve version



(b) Inception module with dimension reductions

위 그림은 Inception Module을 표현한 것이다. 차례대로 특징을 사려보자.

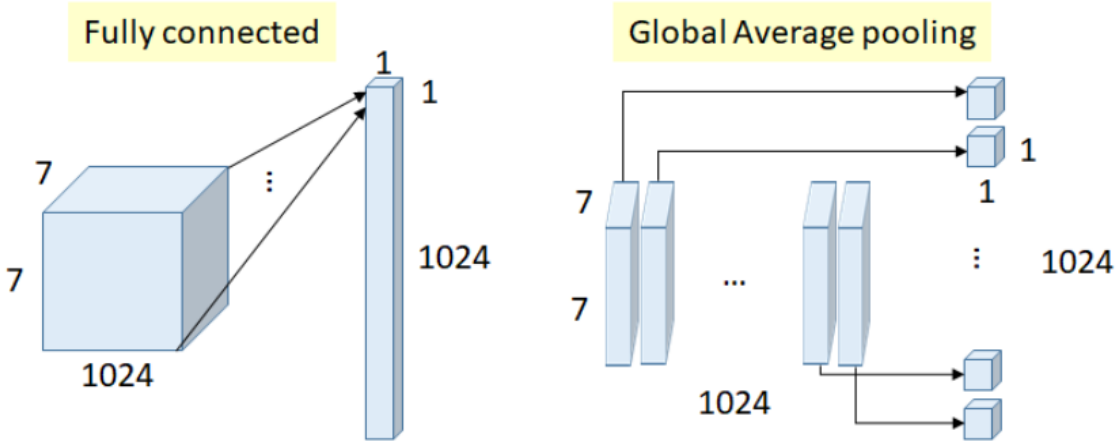
- size가 다양한 conv filter을 사용한다. 이는 다른 종류의 features을 뽑아내기 위함이라고 볼 수 있다. 즉 논문에서 아래 구절과 통하는 부분이다.
  All we need is to find the optimal local construction and to repeat it spatially.
  optimal local construction을 찾기 위해 다양한 size의 conv filter을 사용하고, 이를 spatially 반복한다는 뜻은 Inception module을 쌓아 올린다는 뜻이다.
- alternative parallel pooling path을 추가함으로써 추가적으로 얻을 수 있는 이익을 기대한다.
- (a)는 계산량이 매우 크다. 따라서 1×1 conv filter을 추가하여 계산량을 줄이고 dimension reduction의 효과를 얻는다.

GooLeNet

Inception Module을 쌓아 올려서 만든 것이 GoogLeNet이다.

특징은 다음과 같다.

- Global Average Pooling
  아래 그림의 FC layer의 weights 갯수는 $7 \times 7 \times 1024 \times 1024 = 51.3\text{M}$ 이고 global average pooling의 weigts은 0개이다. 이를 사용해서 계산도 훨씬 빨라지고 정확도도 0.6% 올랐다고 한다.



Fully Connected Layer VS Global Average Pooling

- Auxiliary Classifiers for training
  중간 중간에 softmax branches가 있다. 모델이 매우 깊기 때문에 vanishing gradient 문제를 걱정하지 않을 수 없었고 이를 중간에 있는 softmax branches로 해결하려고 했다. 여기서 구한 loss에 0.3을 곱해서 최종 loss을 구했고 이는 training 과정에만 한다. 즉, test에서는 이 branch를 생략한다.
- 입력과 가까운 부분
  이 부분에는 Inception 모듈을 사용하지 않고 일반 CNN을 수행한다. 논문에 따르면 여기서는 Inception의 효과가 없었다고 한다.