강병규
연세대 컴퓨터과학과 | 연세대 빅데이터 학회 YBIGTA
반갑습니당
home / archive / categories / tags

Home

Inception(GoogLeNet) 리뷰

17 JAN 2018 • 5 mins read

Inception(Going Deeper with Convolutions)

강병규

안녕하세요, 오늘은 Inception, GoogLeNet에 대한 논문을 읽고 간단하게 정리해보고자 합니다.

Going Deep with Convolutions(Szegedy et al.)은 Inception이라는 개념을 제안하는데요, 이를 바탕으로 구현한 모델인 GoogLeNet은 ImageNet Large-Scale Visual Recognition Challenge 2014(ILSVRC14)에서 좋은 성적을 거뒀습니다.

Inception 이전

일반적으로 Convolution Neural Network(이하 CNN)에서 네트워크가 깊어지면 깊어질수록, 즉 더 많은 레이어를 가질수록 성능이 좋아지는 것은 분명합니다. 그러나 이러한 방식의 문제점은 네트워크가 깊어질수록 학습해야하는 파라미터의 수가 늘어난다는 것에 있습니다. 특히 이러한 특징이 두드러지는 것은 깊은 곳에 위치한 Convolution연산입니다.

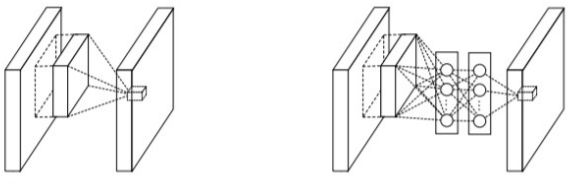
대부분의 CNN에서 이미지가 Convolution연산을 거치게 되면 channel은 커지고 height와 width는 줄어듭니다. 자 구체적인 예를 들어 자세히 알아봅시다. 192 x 28 x 28(C x H x W)의 크기를 가지는 데이터가 있다고 해볼까요? 필터의 크기는 5x5, 스트라이드는 1, 패딩은 2라고 합시다. 필터가 32개 있다고 하면 결과는 32 x 28 x 28이 되겠죠. 문제는 여기서 생겨납니다. 각 필터는 192 x 5 x 5의 크기를 갖고, 결과는 32x28x28이므로, 총 필요한 연산은 192x5x5x32x28x28이 됩니다. 이는 120만개에 이르는 엄청난 연산을 필요로 하죠. 이러한 연산들을 몇 층씩 쌓는다는 것은 너무 부담스러운 일입니다.

문제는 이것뿐만이 아닙니다. 저같은 사람이야 잘나온 논문의 모델을 사용하면 되지만, 만약 직접 모델을 설계해야하는 상황이라면? 필터의 크기와 같은 하이퍼파라미터를 설정하는 것에서부터 고민이 시작될 겁니다. 하지만 Inception이라는 것을 사용하면 이러한 문제를 어느정도 해결할 수 있습니다.

Network in Network

본격적으로 논문에 대해 설명하기 전에 잠깐 설명하고 넘어가고 싶은 것이 있습니다. Network in Network라는 논문입니다.

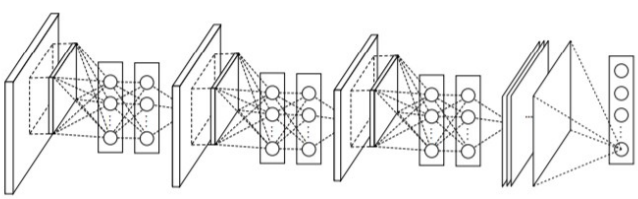
우리가 일반적으로 생각하는 Convolution layer는 필터가 움직이면서 해당하는 입력 부분과 곱/합 연산을 수행하고, 활성화함수를 거쳐 결과를 만들어낸다고 생각할 수 있습니다. 근데 만약 데이터의 분포가 저러한 선형 관계로 표현될 수 없는 비선형적인 관계라면? 그래서 이 논문에서는 비선형적 관계를 표현할 수 있도록, 단순한 곱/합 연산이 아니라 Multi Layer Perceptron, 즉 MLP를 중간에 넣어버립니다. 그래서 이 논문의 제목이 Network in Network(NIN)인 겁니다.



(a) Linear convolution layer (b) Mlpconv layer

> *왼쪽이 우리가 생각하는 일반적인 Conv 레이어, 오른쪽이 이 논문에서 제안하는 MLP Conv*

논문에 따르면, MLP 또한 Convolution의 필터처럼 역전파를 통해 학습되고, 그 자체가 깊은 구조를 가질 수 있으므로 MLP를 사용했다고 합니다.



> *실제 네트워크의 구조*

이런 식으로 MLP Conv를 세 개 쌓고, 마지막에 Fully-Connected Layer를 넣는 대신 Global Average Pooling을 넣었습니다. 이는 오버피팅을 방지하는 효과가 있다고 합니다. 그런데 여기서 한가지 중요한 점이 생겨납니다.

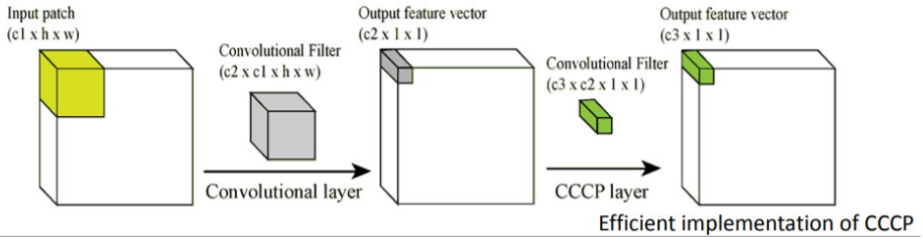
자 여기서 MLP Conv의 연산을 식으로 나타내면(i, j는 feature map에서 픽셀의 위치, k는 feature map의 k번째 채널, n은 MLP Conv의 n번째 레이어)

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

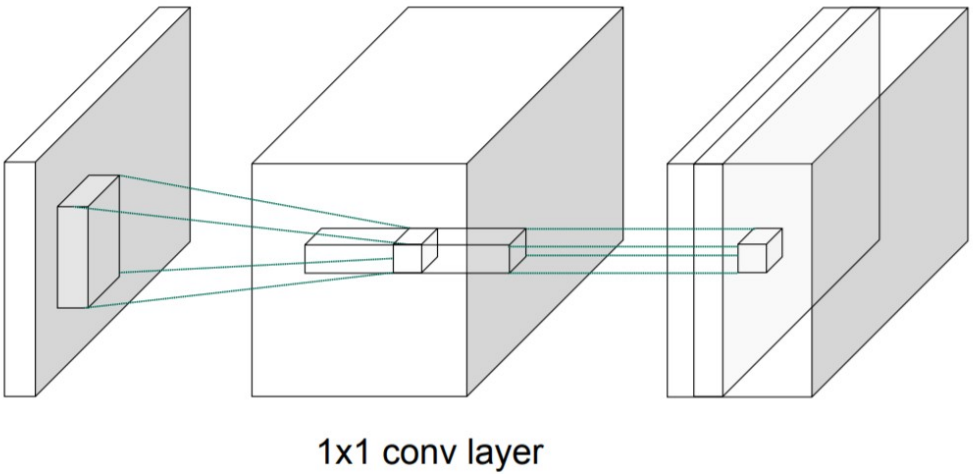
...

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

이 됩니다. 첫번째식은 일반적인 CNN 연산($f_{i,j,k} = max(w_k^\top x_{i,j}, 0)$, i, j는 픽셀의 index, $x_{i,j}$는 그 점을 중심으로 하는 input patch이며 k는 채널의 index입니다)과 동일하다고 볼 수 있습니다. 물론 bias가 더해져 있긴 하지만 CNN에서도 bias를 추가해줄 수 있으니까요. 중요한 것은 마지막 식입니다. 마지막 식은 cross channel parametri(c pooling(CCCP)을 일반적인 Conv 레이어에 적용한 것과 같습니다. 이는 첫번째 식이 일반적인 CNN과 동일하기 때문인데요, 무슨 소린지 모르겠죠?



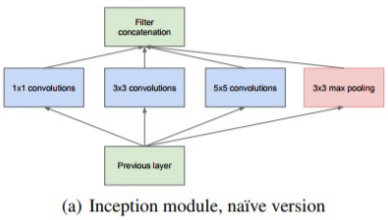
이렇게 일반적인 Convolution 연산을 거치고, 마지막에 하나의 값으로 매핑하는 과정을 CCCP라고 볼 수 있습니다. 즉



요 사진의 연산을 수행한다고 볼 수 있습니다. 위의 두 사진에서 볼 수 있듯, 이 과정(n개의 레이어를 사용하는 MLP Conv)은 일반적인 Convolution 연산을 적용한 다음필터 크기가 1x1인 Convolution을 적용한 것과 동일합니다. 앞에서 말한 NIN의 목적이 무엇이었는지 기억하시나요? MLP의 도입을 통해 비선형적인 관계를 더 잘 표현하는 것이었습니다. 바로 위에서 보였듯, 결국 MLP를 통해 구하는 관계는 일반적인 CNN과 1x1 Conv의 결합으로도 표현할 수 있습니다. 즉 NIN의 의의는 이러한 1x1 Conv의 도입이라고 할 수 있죠. 결과적으로 1x1 Conv를 적절하게 사용하면 비선형적 함수를 더 잘 만들어낼 수 있게 되는 것입니다. 또한 1x1 Conv의 장점은 이것만이 아닙니다. 1x1 Conv는 채널 단위에서 Pooling을 해줍니다. 즉 1x1 Conv의 수를 입력의 채널보다 작게 하면 dimension reduction, 차원 축소가 가능한 것이죠.

Deeper and Deeper

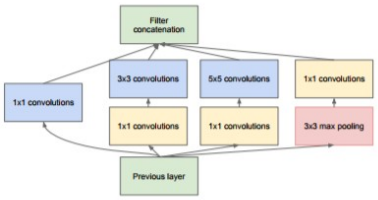
그런데 CNN이 깊어진다고 해서 무조건 성능이 좋아질까요? 우리가 가진 데이터셋은 한정적입니다. 지나치게 깊은 네트워크는 오버피팅의 위험, Vanishing gradient의 위험을 가집니다. 이 뿐만이 아니라, 깊어질수록 필요한 연산의 양 또한 따라서 증가합니다. 이러한 문제를 해결하는 것은 Dropout을 사용하는 네트워크처럼 sparse하게 연결되는 구조를 만드는 것입니다. 하지만 컴퓨터의 연산은 Dense할수록 빠르다는 특징이 있습니다. 이 둘 사이의 타협을 위해 Inception 모듈이 등장하게 됩니다.



(a) Inception module, naïve version

Inception 모듈에서는 feature를 효율적으로 추출하기 위해 1x1, 3x3, 5x5의 Convolution 연산을 각각 수행합니다. 3x3의 Max pooling 또한 수행하는데, 입력과 출력의 H, W가 같아야하므로 Pooling 연산에서는 드물게 Padding을 추가해줍니다. 결과적으로 feature 추출 등의 과정은 최대한 sparse함을 유지하고자 했고, 행렬 연산 자체는 이들을 합쳐 최대한 dense하게 만들고자 했습니다. 그러나 위에서 언급했듯 이렇게 되면 연산량이 너무 많아지게 됩니다.

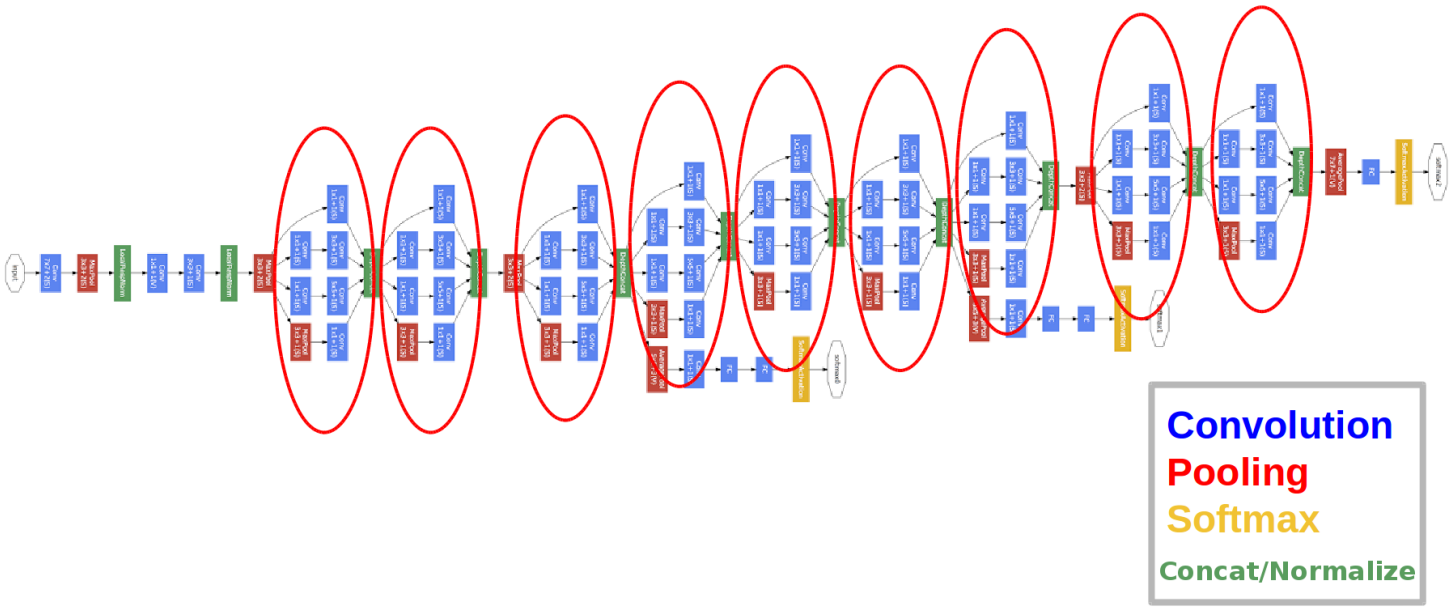
이를 해결하기 위해서 논문은 1x1 Conv를 사용해 dimension reduction을 수행합니다.



(b) Inception module with dimension reductions

이렇게 말이죠. 이전의 입력을 1x1 Conv에 넣어 channel을 줄였다가, 3x3나 5x5 Conv를 거치게해 다시 확장하는 느낌입니다. 이렇게 되면 필요한 연산의 양이 확 줄어들게 됩니다. 또 Pooling의 경우 1x1 Conv를 뒤에 붙였는데요, 이는 Pooling연산의 결과 채널의 수가 이전의 입력과 동일하므로 이를 줄여주기 위함입니다. 이렇게 sparse하게 각 연산을 거친 다음, dense한 output을 만들어내는데요, H와 W는 모두 동일하다는 것에 주의해야합니다. 즉 Concat연산을 channel에 적용한다고 보시면 될 것 같습니다.

이렇게 Inception 모듈을 활용한 최종 모델의 구조는 아래와 같습니다.



잘 보이시나요? 엄청나게 깊은 모델이죠? 빨간색 동그라미가 쳐져있는 부분은 Inception 모듈을 사용한 곳입니다. 이외에도 중간중간 Pooling layer를 추가로 삽입해 크기가 줄어들 수 있게 했습니다. 이때 주목해야할 부분이 2가지가 있습니다.

첫번째로 네트워크의 얕은 부분, 입력과 가까운 부분에는 Inception 모듈을 사용하지 않았다는 것입니다. 논문에 따르면 이 부분에는 Inception의 효과가 없었다고 합니다. 따라서 우리가 일반적으로 CNN하면 떠올리는, Conv와 Pooling 연산을 수행합니다. 두 번째는 softmax를 통해 결과를 뽑아내는 부분이 맨 끝에만 있는 것이 아니라, 중간 중간에 있다는 점입니다. 이를 논문에서는 auxiliary classifier라고 부릅니다. 엄청나게 깊은 네트워크에서 Vanishing Gradient 문제를 걱정하지 않을 수 없죠. 그래서 auxiliary classifier를 덧붙인 겁니다. Loss를 맨 끝뿐만 아니라 중간 중간에서 구하기 때문에 gradient가 적절하게 역전파된다고 합니다. 대신 지나치게 영향을 주는 것을 막기 위해 auxiliary classifier의 loss는 0.3을 곱했습니다. 물론 실제로 테스트하는 과정에서는 auxiliary classifier를 제거하고 맨 끝, 제일 마지막의 softmax만을 사용하구요.

Conclusion

이렇게 해서 Inception과 이를 활용한 GoogLeNet에 대해 간략하게 알아보았습니다. 각 필터의 결과를 합쳐서(Concat) 표현하는 것이 Inception 모듈이라 할 수 있겠습니다. GoogLeNet의 경우에는 Inception 모듈을 사용해 엄청나게 깊은 층을 가지는데요, 연산량을 줄이기 위해 1x1 Conv를 적극적으로 사용한 점이 특징이라고 할 수 있겠습니다. 이러한 아이디어는 Network in Network에서 비롯되었다고 할 수 있을 것 같습니다.

Reference

Network in Network1: Convolutional Neural Network Architectures: from LeNet to ResNet

Network in Network2: Case study of Convolutional Neural Network

Inception1: Google Inception Model

Inception2: Best CNN Architecture - GoogLeNet