DL 스터디&프로젝트

[Euron 중급 세션 17주차] 딥러닝 4단계 2. 케이스 스터디

by 공부하자_ 2024. 1. 1.

딥러닝 4단계: 합성곱 신경망 네트워크 (CNN)

2. 케이스 스터디

📌왜 케이스 스터디를 하나요?(C4W2L01)

핵심어: 합성곱 신경망

지난 강의에 이어서 몇 개의 효과적인 합성곱 신경망의 사례들을 살펴보고자 한다. 사례들을 살펴보는 이유는 수 년간의 컴퓨터 비전의 연구 대상은 합성곱 층과 풀링 층과 완전 연결 층 등 구성 요소들을 조합하여 효과적인 합성곱 신경망을 만드는 것인데, 이에 영감을 얻는 가장 좋은 방법이 예시를 보는 것이기 때문이다. 다른 사람의 코드를 읽으며 코드 작성법을 익혔듯, 합성곱 신경망을 구축하기 위해서는 효과적인 신경망을 살펴봐야 한다. 하나의 컴퓨터 비전 작업에서 잘 작동하는 신경망의 구조는 다른 작업에서도 잘 작동하는 경우가 많다. 만약 누군가가 발견해낸 하나의 신경망 구조가 고양이나 개나 사람을 인식하는 데 유용하다면 자율 주행 차를 개발하는 등의

분류 전체보기

DL 스터디&프로젝트

Data Science 프로젝트

Github 스터디

Data Science 개인 공부

Backend 프로젝트

기타 공부

공지사항

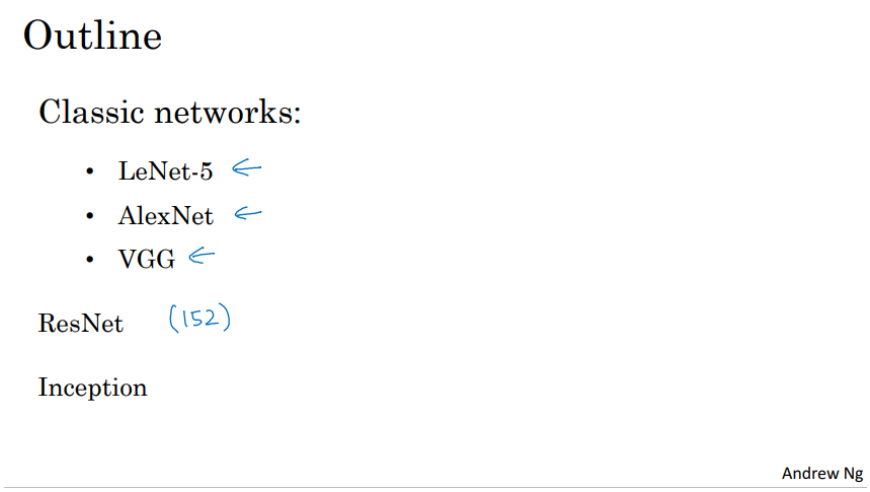
최근글 인기글

[Euron 중급 세션 17주차]…
2024.01.01

[Euron 중급 세션 16주차]…
2023.12.25

백엔드 프로젝트 12주차 - 스프링부트3 자바 …

다른 컴퓨터 비전 작업을 할 때도 다른 사람이 만든 신경망 구조를 적용할 수 있다.



이후에 다룰 내용의 개요로 몇 개의 대표적인 신경망(Classic networks)을 알아볼텐데, LeNet-5는 1980년대에 나온 신경망이고 AlexNet은 자주 이용되고, 또한 VGG망이라는 것도 있다. 이들은 꽤 효과적인 신경망이고 몇몇 개념들은 현대 컴퓨터 비전의 기틀을 마련했다.

그 다음으로 알아볼 것은 ResNet이다. 신경망의 깊이가 깊어질수록 ResNet 신경망은 152개의 층을 훈련시키고 굉장히 흥미로운 개념을 가지고 있다.

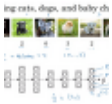
마지막으로는 Inception 신경망의 사례들을 살펴볼 것이며 이 과정을 따르면 어떻게 효과적인 합성곱 신경망을 구축할지 감이 올 것이다. 컴퓨터 비전의 분야를 다루지 않는다 할지라도 이 예시에서 나오는 수많은 개념들 ResNet이나 이전 신경망의 개념들은 다른 분야에서도 유용하게 쓰일 수 있다. 따라서 컴퓨터 비전을 구축하지 않게 되더라도 몇몇의 개념들은 유용하게 쓰일 것이다.

📌 고전적인 네트워크들(C4W2L02)

핵심어: LeNet-5, AlexNet, VGG

이번에는 몇 가지 고전적인 신경망의 구조를 살펴볼 것인데, LeNet-5와 AlexNet과 VGGNet이다.

2023.12.23

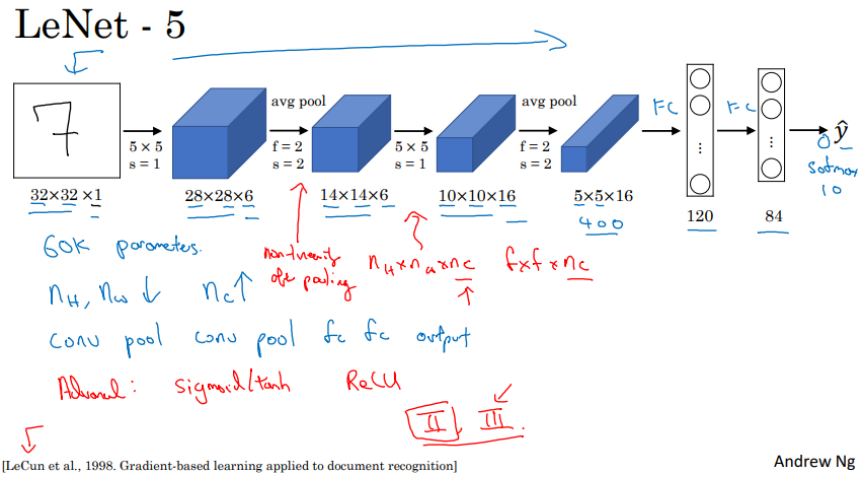
[Euron 중급 세션 15주차]…
2023.12.18

[Euron 중급 세션 12주차]…
2023.11.27


최근댓글

오늘 하루 고생 많으셨습니다.
좋은 글 잘 보고 가요! 감사…
좋은 글 잘 보고 가요! 감사…
잘보고 갑니다!

태그

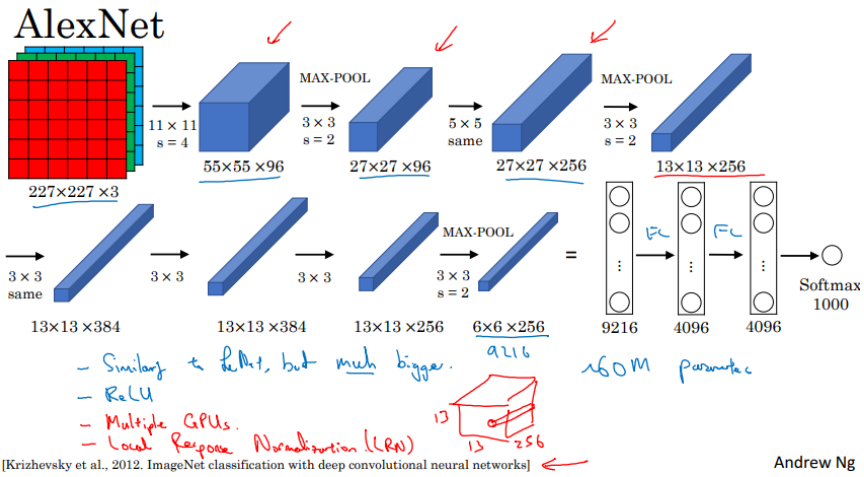
딥러닝스터디, 데이터사이언스, 딥러닝교과서, pandas, 판다스, Doit, bda, 데이터분석, 판다스입문, 이지스퍼블리싱



다음과 같이 LeNet-5 구조가 있다. 32*32*1의 이미지가 있을 때 LeNet-5의 목적은 손글씨의 숫자를 인식하는 것이며 흑백의 이미지에 훈련되어있다. 이 신경은 첫 번째로 6개의 5*5 필터와 1의 스트라이드를 사용하는데, 6개의 필터를 사용하기 때문에 28*28*6의 결과를 얻게 된다. 그리고 스트라이드는 1이고 패딩을 사용하지 않기 때문에 32*32가 28*28로 감소한다. 그리고 LeNet 신경망에서 풀링을 적용하는데, 이 당시에는 평균 풀링을 많이 사용했으며 필터 크기가 2이고 스트라이드가 2여서 높이와 너비의 크기를 절반으로 줄인 14*14*6이 된다. 그 다음에는 또 다른 합성곱 층을 적용하는데 이번에는 16개의 5*5 필터를 사용해 16개의 채널이 만든다. 이 논문이 쓰여질 당시인 1998년에는 패딩을 사용하지 않거나 유효 합성곱을 사용했으며 따라서 합성곱 층을 적용할 때마다 높이와 너비가 감소했다. 그래서 14*14이 10*10으로 줄어든 것이다. 또 하나의 풀링 층으로 높이와 너비가 절반인 5*5로 변하고 이 숫자들을 곱하면 5*5*16은 400이다. 25*16은 400이기 때문이다. 다음 층은 완전 연결 층인데 400개의 노드를 120개의 뉴런에 각각 연결하여 완전 연결해줘서 완전 연결 층을 만들어 준다. 또 다른 완전 연결 층이 있으며 마지막 과정은 이 84개의 속성을 가지고 하나의 최종 출력에 적용한다. 여기 노드 하나를 더 그려서 y의 예측값을 나타내고 y의 예측값은 10가지 가능성이 있는데 0부터 9까지의 숫자 인식에 해당한다. 요즘이라면 소프트맥스 층을 사용해서 10분류의 결과가 나오게 되지만, LeNet-5는 요즘 잘 사용하지 않는 분류기를 출력 층에서 사용한다. 이 신경망은 요즘 기준으로 적은 60000개의 변수를 가진다. 요즘에는 1000만 또는 1억개의 변수를 가지는 신경망도 있는데, 이것보다 1000배 정도 큰 신경망을 보는 것은 그리 어려운 일이 아니다. 하지만 요즘도 깊이가 깊어질수록 신경망의 왼쪽에서 오른쪽으로 이동할 때 높이와 너비가 감소한다. 32에서 28, 14, 10, 5로 감소하는 반면 채널의 수는 신경망의 층이 깊어질수록 1, 6, 16으로 증가한다.

이 신경망에서도 볼 수 있고 요즘에도 사용되는 또 다른 패턴은 몇 개의 합성곱 층 뒤에 풀링 층이 따라오고 또 합성곱 층에 풀링 층이 또 따라서 온 뒤 완전 연결 층이 있고 그 다음에 출력이 있는 것이다.

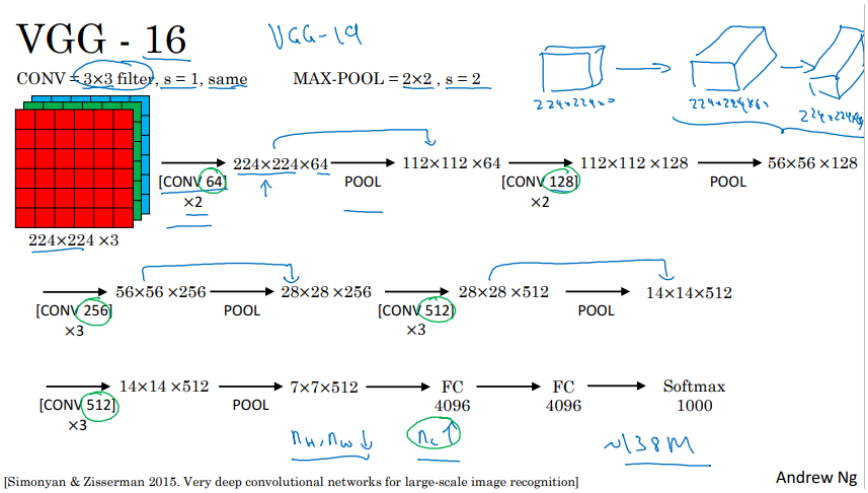
이러한 층의 배치는 요즘도 흔히 볼 수 있다. 추가적으로, 원래의 논문을 읽어보면 당시의 사람들은 시그모이드와 tanh 비선형성 함수를 많이 썼고 ReLU 비선형성 을 사용하지 않았다. 요즘 기준으로 이 신경망이 조금 우스운 점은 nH*nW*nC의 신경망에서 nC 개의 채널이 있을 때, f*f*nC 크기의 필터를 사용해 필터가 모든 채널을 살펴보게 하는데 반해 당시에는 컴퓨터가 훨씬 느려서 변수처럼 계산을 줄이기 위해 초기의 LeNet-5는 각각의 필터가 서로 다른 채널에 적용되었다. 하지만 요즘에는 이렇게 복잡한 방식으로 사용하지 않는다. 또 한 가지 당시에는 사용됐고 지금은 사용되지 않는 것이, 기존의 LeNet-5는 비선형성이 풀링 뒤에 있었다. 시그모이드 비선형성을 풀링 층 뒤에 적용한 것이다.



두 번째로 볼 신경망은 AlexNet이다. 입력은 227*227*3의 이미지로 시작하는데, 실제로노문에서는 224*224*3의 이미지를 사용하지만 숫자를 살펴보면 227*227일 때 좀 더 그럴듯한 숫자가 된다. 첫 번째 층에서는 96개의 11*11 필터를 사용하고 4의 스트라이드를 이용해서 크기가 55*55로 줄어든다. 스트라이드의 값 때문에 크기가 4분의 1로 줄어드는 것이다. 그리고 3*3의 크기와 2의 스트라이드인 최대 풀링을 적용하면 크기가 27*27*96으로 변하게 된다. 그리고 5*5의 동일 합성곱연산을 실행해주면 27*27*256로 크기가 변한다. 또 한 번 최대 풀링을 적용해주면 높이와 너비가 13으로 줄어들고 또 동일 합성곱을 적용해주면 13*13*384의 필터로 변하고 3*3의 동일 합성곱을계속 해주고 최대 풀링을 적용하면 6*6*256이 나오며 이를 계산하면 9216이 된다. 이를 전개해서 9216개의 노드로 만들면 완전 연결 층을 가지게 되고, 소프트맥스를 사용해서 가능한 1000개의 출력을 나타낸다.

이 신경망은 LeNet과 매우 유사하지만 훨씬 큰 크기를 가진다. LeNet이나 LeNet-5는 6만 개 정도의 매개 변수를 가졌지만, AlexNet의 경우는 6천만개 정도의 매개변수를 지닌다. 실제로 유사한 구성요소를 가질 수 있지만 더 많은 은닉 유닛과 더 많은 데이터를 통해 훈련하기 때문에 훨씬 더 뛰어난 성능을 보여줄 수 있는것이다. 이

구조가 LeNet과 구별되는 또 다른 하나의 특성은 ReLU 활성화 함수를 사용한다는 것이다. 논문을 읽었을 때 알 수 있는 것은 논문이 쓰여질 당시의 GPU는 여전히 느려서 두 개의 GPU를 훈련하는 복잡한 방법을 거친다. 기본 개념은 이러한 층들이 두 개의 구별된 GPU에 나눠져서 두 개의 GPU가 소통하는 방식이다. 논문에서는 AlexNet의 기본 구조에 지역 응답 정규화라는 층이 있다고 하는데, 자주 사용되는 층은 아니다. 지역 응답 정규화의 개념을 살펴보자면, 어떤 상자의 크기가 13*13*256이라고 하면 지역 반응 정규화인 LRN이 하는 일은 높이와 너비가 지정된 한 지점의 256개의 모든 채널을 보고 정규화 하는 것이다. 그 이유는 13*13이미지의 각 위치에서 높은 활성값을 가진뉴런을 너무 많지 원치 않기 때문이다. 하지만 이후의 연구에서 이는 그리 유용하지 않다는 것이 밝혀졌다. AlexNet은 상대적으로 복잡한 구조를 지니는데, 이를 만든 사람이 수많은 하이퍼 파라미터들을 사용했기 때문이다.



세 번째이자 마지막으로 알아볼 예시는 VGG-16 네트워크이다. 여기서 주목할 만한 점은 많은 하이퍼 파라미터들을 가지는 대신 합성곱에서 스트라이드가 1인 3*3 필터만을 사용해 동일합성곱을 하고, 최대 풀링층에서는 2의 스트라이드의 2*2를 사용한다. VGG-16은 아주 간결한 구조를 가진다. 그 구조를 한 번 살펴보고자 한다.

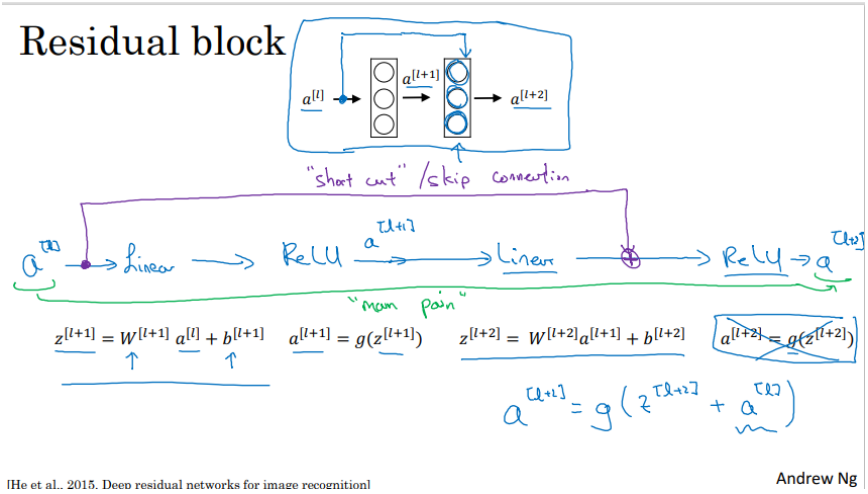
이미지에서 시작해 첫 두 층에서는 합성곱을 해준다. 따라서 첫 두 층에서는 64개의 필터를사용하며 동일합성곱이기 때문에 224*224의 크기이고 64개의 채널이 있다. 즉 224*224*3의 이미지가 합성곱을 통해 224*224*64가 되고 224*224*64의 또 다른 한 개의 층이 존재한다.따라서 이 [CONV 64]*2는 64개의 필터들을 가진 두 개의 합성곱층을 사용한다는 의미가 된다. 앞서 설명했듯 모든 필터는 3*3 크기를 가지면 1의 스트라이드와 동일 합성곱을 사용한다. 그 뒤로는 풀링 층을 사용하는데, 풀링층은 224*224의 크기를 112*112*64로 감소시킨다. 그리고 몇 개의 합성곱 층이 더 있는데, 128개의 필터를 가진 동일합성곱이면 크기가 112*112*128로 변한다. 이어서 풀링층이 있고 새로운 크기는 56*56*128이 된다.

그리고 256개의 필터를 합성곱층 3개를 사용하고 그 다음으로 풀링층을 사용, 그리고 몇 개의 합성곱층과 풀링층을 번갈아가며 사용하면 결국 7*7*512의 완전 연결층이 된다. 4096개의 유닛과 1000개의 소프트맥스 출력이 나오는 것이다. VGG-16의 16이라는 숫자는 16개의 가중치를 가진 층이 있다는 것을 의마한다. 1억3천8백만개 정도의 변수를 가진 상당히 큰 네트워크인데, 요즘 치고도 큰 편이다. 하지만 VGG-16의 구조적인 장점은 꽤나 균일하다는 것이며, 몇 개의 합성곱층 뒤에 풀링층이 높이와 너비를 줄여준다. 그리고 합성곱층의 필터의 개수를 한 번 살펴보면 64가 128, 256개, 512로 두 배씩 늘어난다. 아마 충분히 크기 때문에 더 늘리지 않는 것 같다. 합성곱층에서 매번 두 배씩 증가시키는 것은 이 네트워크를 이루는 간단한 규칙이 된다. 따라서 이 구조의 상대적인 획일성이 가지는 단점은 훈련시킬 변수의 개수가 많아 네트워크의 크기가 커진다는 것이다. 또한 VGG-19는 이것보다 더 큰 버전인데 이 또한 참고하면 좋을 듯하다. 다만 VGG-16은 VGG-19 만큼의 성능을 보여주기 때문에 더 많이 사용된다고 볼 수 있다. 깊이가 깊어질수록 보여지는 패턴이 풀링층에서는 높이와 너비가 반씩 줄어들고 합성곱 층에서는 채널 수가 매번 두 배 가량 늘어나는 규칙을 확인할 수 있다. 수치가 커지고 작아지는 것이 상당히 체계적으로 이루어지는 것이며, 그런 관점에서 매우 매력적인 논문이다.

이렇게 해서 3개의 고전적인 구조를 살펴보았다. 앞으로는 더 강력하고 고급의 신경망 구조를 살펴보고자 한다.

📌ResNets(C4W2L03)

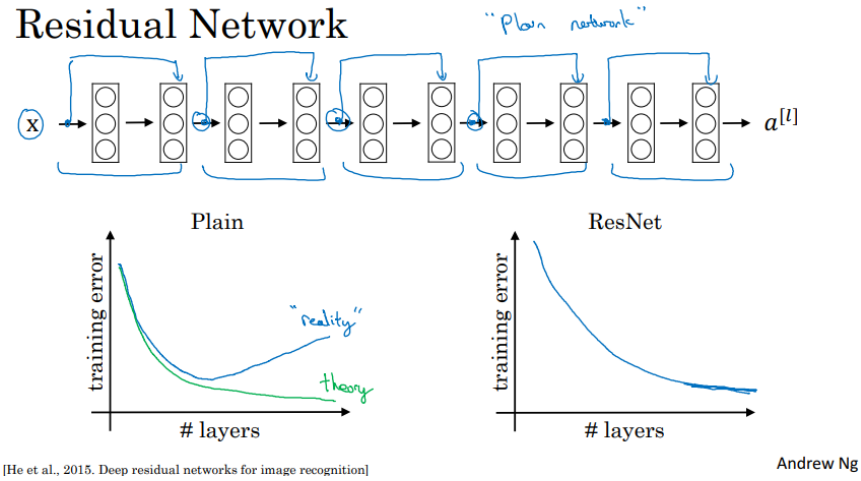
핵심어: Residual Networks, 스킵 연결(skip connection), 지름길(short cut), 잔여 블록(residual block), 평형망(plain network)



아주 깊은 신경망을 훈련시키기 어려운 이유는 경사가 소실되거나 폭발적으로 증가하는 문제 때문이다. 이번에는 스킵 연결에 대해 알아보며, 한 층의 활성값을 가지고 훨씬 깊은 층에 적용하는 방식이다. 이를 통해 ResNet을 학습할 수 있으며 100개가 넘는 깊이의 신경망을 학습할 수 있게 해준다.

ResNet은 잔여블록이라는 것으로 구성되어있다. 위 슬라이드에 신경망의 두 층이 있는데, 활성값 a[l]에서 시작해서 a[l+1]으로 가고 두 층이 지난 뒤 활성값은 a[l+2]이다. 이 계산 과정을 살펴보면 a[l]에 선형 연산을 적용해주고, a[l]에서 z[l+1]로 가기 위해 가중치 행렬을 곱해주고 편향 벡터를 더해준다. 그리고 ReLU 비선형성을 적용해 a[l+1]을 계산한다. a[l+1]은 g(z-[l+1])이고 하다. 그 다음 층에서는 또 선형 연산을 적용해주면 비슷한 식이 나타난다. 마지막으로 또 한 번 ReLU 연산을 적용해주며 여기서 g가 ReLU 비선형성이다. 그리고 여기서 a[l+2]를 얻는다. 다시 말해 a[l]의 정보가 a[l+2]로 흐르기 위해서는 이 모든 과정을 거쳐야 한다. 이것을 여러 층의 main path라 하고, ResNet에서는 이것을 조금 바꿔서 a[l]을 복제하여 신경망의 더 먼 곳 까지 단번에 가게 만든 뒤 ReLU 비선형성을 적용하기 전에 a[l]을 더해주고 이것을 short cut이라고 부를것이다. 그러면 main path을 따르는 대신 a[l]의 정보는 short cut을 따라서 신경망의 더 깊은 곳으로 갈 수 있게된다. 이것으로 마지막 식이 필요가 없어지고, 대신 a[l+2]=g(z[l+2]+a[l])이 더해진다. 여기서 이 a[l]은 잔여 블록이 된다. 위 그림에서도 2번째 층으로 가는 지름길을 그릴 수 있는데, 두 번째 층으로 가는 이유는 ReLU 전에 더해지기 때문이다. 여기 각 노드(두 번째 단계의 노드들)들은 선형 연산과 ReLU를 적용하는데, a[l]은 선형 연산 뒤에 삽입되고 ReLU 연산 전에 들어간다. 그리고 shrot cut 대신 스킵 연결이라는 표현도 쓰는데, 이는 a[l]이 정보를 전달하기 위해 층을 뛰어넘는 것을 의미한다. 신경망의 더 깊은 곳으로 말이다.

ResNet의 개발자가 발견한 것은 잔여 블록을 사용하면 훨씬 깊은 신경망을 훈련시킬 수 있다는 것이다. ResNet을 구축하는 방법은 이러한 잔여 블록들을 쌓아서 깊은 신경망을 만드는 것이다.

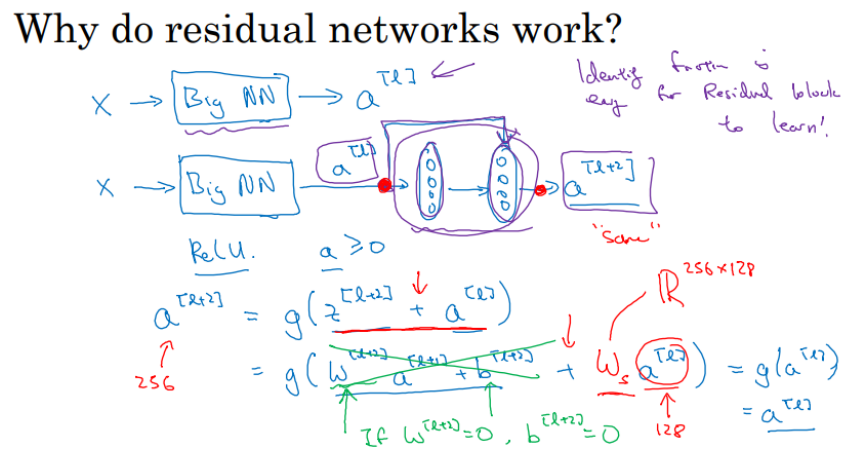


위 네트워크는 ResNet 네트워크가 아닌 plain 네트워크(평형망)인데, 이것을 ResNet으로 바꾸려면 스킵 연결을 더해주면 된다. 따라서 두 층 마다 이전 슬라이드에서 본 잔여 블록으로 만들어주기 위한 변화가 생긴다. 이 그림은 다섯 개의 잔여블록이 합쳐진 것이고 이것이 ResNet이다. 만약 표준 최적화 알고리즘을 사용한다면 경사 하강법이나 다른 최적화 알고리즘을 사용해 평형망을 훈련시키고(방금 그린 스킵 연결을 사용하지 않고) , 경험적으로 층의 개수를 늘릴수록 훈련 오류는 감소하다 다시 증가하는데, 이론 상으로는 신경망이 깊어질수록 훈련 세트에서 점점 더 나아져야 한다. 이론 상으로는 깊은 신경망이 도움 되지만 실제로는 평형망의 깊이가 매우 깊다면 최적화 알고리즘으로 훈련을 하는 것이 더 어려워질 것이고 너무 깊은 신겨망을 선택하면 훈련 오류는 더 많아진다.

하지만 ResNet에서는 층이 깊어져도 훈련 오류가 계속 감소하는 성능을 가질 수 있다. 100개 이상의 층을 훈련시킨다고 해도 말이다. 그리고 몇몇의 사람들은 수 천개의 층을 가지고 신경망을 실험 중이다. 활성값 x 또는 중간 활성값을 취하는 것으로 훨씬 더 깊은 신경망을 사용할 수 있게 해준다. 이것이 경사 소실 문제에 많은 도움이 되고, 성능에 큰 저하 없이 더 깊은 신경망을 훈련하게 해준다. 언젠가는 이것들을 펼쳐져서 더 이상 깊어지지 못할 수도 있지만, ResNet이 깊은 신경망 훈련에 효과적인 것은 분명하다.

📌왜 ResNets이 잘 작동할까요?(C4W2L04)

핵심어: Residual Networks, 스킵 연결(skip connection)/지름길(short cut)



신경망을 학습시키는 것은 어렵다. 특히 네트워크가 깊어질수록 더 어려워지는데, 왜냐하면 각 층이 이전 층의 출력을 입력으로 받아 처리하고 이는 연쇄적으로 이루어지기 때문이다. 이 때문에 네트워크가 깊어질수록 각 층이 올바른 출력을 만들어내는 것이 점점 더 어려워지며 이것을 '깊은 네트워크의 문제'라고 할 수 있다. ResNet은 이 문제를 '스킵 연결' 또는 '잔차 블록'을 통해 해결하려고 한다. 스킵 연결은 입력 데이터를 네트워크의 더 깊은 층으로 바로 전달하는 방법이다. 이렇게 하면, 깊은 층에서도 원본 입력 데이터를 직접 참조할 수 있기 때문에 네트워크가 깊어져도 학습이 잘 이루어질 수 있다. 예를 들어 네트워크가 '고양이' 이미지를 인식하려고 하는데, 깊은 층에서는 '고양이의 귀'를 인식하는 것이 어렵다고 생각해보자. 하지만 스킵 연결이 있다면, '고양이의 귀' 정보를 직접적으로 깊은 층으로 전달할 수 있다. 따라서 깊은 층에서도 '고양이의 귀'를 잘 인식할 수 있게 된다. ResNet은 깊은 네트워크에서도 훈련 세트를 잘 다루는 데 효과적이다. 또한 이는 테스트 세트에서도 좋은 결과를 가져다 주는데, 훈련이 잘 되면 테스트에서도 잘 작동하기 때문이다. 이러한 방법으로 ResNet은 깊은 네트워크의 문제를 해결하고, 더 깊은 네트워크를 사용하여 더 복잡한 문제를 해결할 수 있게 한다.

잔차 블록은 '스킵 연결'을 통해 입력 데이터를 몇 층을 건너뛰고 바로 다음 층으로 전달하는 방식이다. 이렇게 하면 신경망의 깊이가 깊어져도 입력 데이터가 직접 전달되므로 학습이 잘 이루어질 수 있다. ReLU 함수는 0이하의 값을 0으로 만들고, 0이상의 값을 그대로 출력하는 함수이며 이 함수를 활성화 함수로 사용하면, 신경망의 출력 값이 0 또는 양수가 된다. 잔차 블록의 작동 방식을 수식으로 표현하면 a[l+2] = ReLU(z[l+2}+a[l])처럼 나타낼 수 있다. 여기서 z[l+2]는 l+2번째 층의 입력 값에 가중치를 곱하고 편향을 더한 값이고, a[l]은 l번째 층의 출력 값이다. 이 두 값을 더한 후 ReLU 함수를 적용하면 a[l+2] 값을 얻을 수 있다. 이 과정에서 +a[l] 부분이 바로 스킵 연결을 나타낸다. 이 스킵 연결은 l번째 층의 출력 값을 l+2번째 층의 출력 값에 바로 더하는 것을 말합니다. 이렇게 하면 l번째 층의 출력 값이 l+2번째 층에 직접적으로 영향을 미치게 된다. 이런 방식으로 스킵 연결

은 신경망의 깊이가 깊어져도 학습이 잘 이루어질 수 있게 하는 ResNet의 핵심 아이디어입니다.

신경망에서 가중치가 줄어드는 현상, 즉 '가중치 붕괴'는 L2 규제와 같은 방법을 사용하면 발생할 수 있다. L2 규제는 신경망의 가중치 값을 작게 유지하는 역할을 한다. 이는 모델의 복잡도를 줄이고 오버피팅을 방지하는 효과를 가지는데, 이 과정에서 가중치의 값이 너무 작아지는 현상, 즉 '가중치 붕괴'가 발생할 수 있는 것이다. L2 규제는 손실 함수에 가중치의 제곱합을 추가한다. 이는 모델이 학습하는 과정에서 가중치 값을 가능한 한 작게 만들려는 효과를 낸다. 이 때 가중치가 너무 작아져서 0에 가까워지면, 해당 가중치는 실질적으로 모델의 예측에 거의 기여하지 않게 된다. 이런 현상을 '가중치 붕괴'라고 한다. 따라서 L2 규제를 사용할 때는 적절한 규제 계수를 설정하는 것이 중요하다. 규제 계수가 너무 크면 가중치 붕괴가 발생할 수 있고, 이는 모델의 성능을 저하시킬 수 있다. 반대로 규제 계수가 너무 작으면 규제의 효과가 미미할 수 있다. 이렇게 규제 계수를 적절히 설정하여 모델의 복잡도와 성능 사이의 균형을 맞추는 것이 중요하게 된다.

또한 이러한 경우 w[l+2]같은 가중치가 0에 가까워지게 되고, 이는 신경망의 출력에 큰 영향을 미친다. w[l+2]와 b[l+2]가 모두 0이라면 식 w[l+2]a[l] + b[l+2]는 0이 되고, 이는 a[l+2] = ReLU(0+a[l])을 의미하게 된다. 즉, a[l+2] = a[l]이 되는 것이다. 이는 잔차 블록의 특징으로, 이 블록을 통과한 출력값이 그대로 다음 층으로 전달된다. 이러한 잔차 블록의 특성은 신경망이 깊어져도 학습이 잘 이루어질 수 있도록 하는데, 왜냐하면 잔차 블록을 통해 추가된 층들이 실제로 아무런 기능을 하지 않아도, 즉 항등 함수를 학습하더라도, 신경망의 성능은 최소한 개선되지 않은 채 유지될 수 있기 때문이다. 이는 신경망이 깊어질수록 성능이 저하되는 문제를 완화한다. 하지만 우리의 목표는 단지 성능을 유지하는 것이 아니라, 성능을 향상시키는 것이다. 잔차 블록의 존재로 인해 추가된 층들이 항등 함수 이상의 역할을 하게 되면, 신경망의 성능은 더욱 향상될 것이다. 정리하자면, 잔차 블록은 신경망이 깊어져도 성능 저하를 방지하고, 성능 향상을 위한 여지를 제공하는 중요한 역할을 한다.

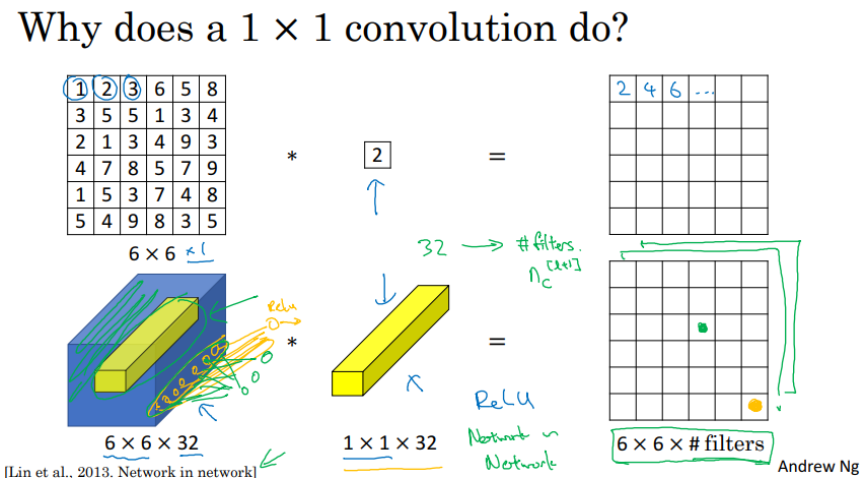
잔차 블록이 추가되면 추가된 층들은 항등 함수를 학습하는 것이 용이해진다. 이는 신경망의 성능 저하 없이 더 깊은 신경망을 구성할 수 있게 만들어주며, 따라서 성능 향상의 가능성을 열어준다. 잔차 블록의 동작 원리를 이해하려면, 신경망의 '합성곱' 연산에 대한 이해가 필요하다. 합성곱 연산은 입력 데이터와 필터(가중치)를 곱한 후 더하는 연산으로, 이미지 처리에서 주로 사용된다. 이때 잔차 블록에서는 '동일 합성곱'이라는 기법을 사용하여 입력과 출력의 차원을 일치시킨다. 동일 합성곱이란 입력과 출력의 차원이 동일하게 유지되도록 하는 합성곱 방식인데, 이로 인해 잔차 블록의 입력 a[l]과 출력

a[l+2]의 차원이 같아지므로 둘을 더하는 연산이 가능해지는 것이다. 그러나 모든 경우에 입력과 출력의 차원이 같지는 않다. 예를 들어, a[l]이 128차원이고 a[l+2]가 256차원인 경우가 있을 수 있다. 이런 경우에는 행렬 Ws를 사용하여 a[l]의 차원을 늘려주는 방식을 사용한다. Ws는 256x128 크기의 행렬로, 이 행렬과 a[l]을 곱하면 결과는 256차원이 된다. 이렇게 차원을 늘려주면 a[l]과 a[l+2]의 차원이 일치하게 되므로, 둘을 더하는 연산이 가능해진다. 이때 Ws는 학습 가능한 가중치를 가질 수도 있고, 고정된 값을 가질 수도 있다. 고정된 값을 가지는 경우에는 a[l]의 원소들에 0을 채워넣는 방식(제로 패딩)을 사용하여 차원을 늘려주는 방법이 주로 사용되며, 이 두 가지 방식 중 어느 것을 사용할지는 문제에 따라 달라진다.

Ws가 고정값을 가질 때는 대체로 제로 패딩 방식을 사용하여 행렬의 나머지 부분을 0으로 채워준다. 반면에 Ws를 학습 가능한 가중치로 설정하면 신경망이 학습하는 과정에서 최적의 Ws 값을 찾아가게 된다. 이는 일종의 차원 변경 학습으로 볼 수 있다. 이 두 가지 방식은 모두 사용 상황과 목적에 따라 선택된다.

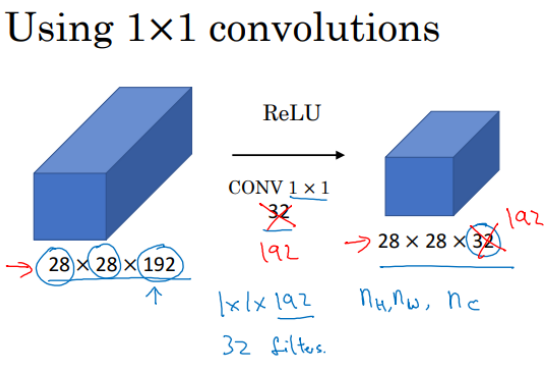
📌Network 속의 Network(C4W2L05)

핵심어: 합성곱 신경망(Convolutional Neural Network), 필터(filter)



합성곱 신경망을 구축할 때 1*1 합성곱은 매우 유용하다. 1*1 합성곱이 무슨 일을 할지에 대해 질문을 가질 수 있는데, 그냥 숫자 하나를 곱하는 것이라 의미없어 보이지만 실제로는 그렇지 않다. 예를 들어 숫자 2의 1*1 필터가 있다고 하자. 여기 6*6 이미지를 1*1 필터와 합성곱 연산을 하면, 이미지에 2만큼 곱해주는 셈이 된다. 이렇듯 1*1 필터와의 합성곱은 숫자 하나를 곱하는 것이기 때문에 그렇게 유용해 보이지 않는다. 그러나 이것은 6*6의 크기에 1개의 채널만 있

을 때이고, 6*6*32라고 한다면 1*1 필터와의 합성곱을 하는 것은 훨씬 의미 있게 된다. 1*1 합성곱이 하는 일은 36개의 위치를 각각 살펴보고, 그 안의 32개의 숫자를 필터의 32개의 숫자와 곱해준다. 그리고 ReLU 비선형성을 적용해준다. 그럼 36개 중 하나의 위치에 한 조각이 생기고, 여기 32개의 숫자를 이 한 조각의 숫자와 곱해주면 한 지점에 하나의 숫자만 남게 된다. 이 1*1*32 필터 안에 있는 32개의 숫자에 대해서는 마치 하나의 뉴런이 32개의 숫자를 입력받고 32개의 숫자를 각각 같은 높이와 너비에 해당하는 채널 각각에 곱해준 후 ReLU 비선형성을 적용해주면 해당하는 값이 각 위치에 출력되는 것이다. 그리고 만약 하나가 아닌 다수의 필터가 있다면 여러 개의 유닛을 입력으로 받아서 한 조각으로 묶는 셈이 되고 출력은 6*6가 필터의 수만큼 있게 된다. 그래서 1*1 필터에 대해 이해하는 한 방법은 완전 연결 신경망을 36개의 위치에 각각 적용해서 32개의 숫자를 입력값으로 받고 필터의 수 만큼 출력하는 것이다. 그래서 이 부분은 nC[l+1]이다. 이것을 36개의 위치에 각각 실행하게 되면 6*6*(필터의 수)를 결과로 얻고 입력값에 대한 자명하지 않은 계산을 해야한다. 종종 1*1 합성곱이라고 불리곤 하는데, 네트워크 안의 네트워크라고도 한다. 1*1 합성곱 또는 네트워크 안의 네트워크라는 개념은 다른 신경망 구조에 많은 영감을 주었다. 예를 들면 인셉션 네트워크 등이 있다.



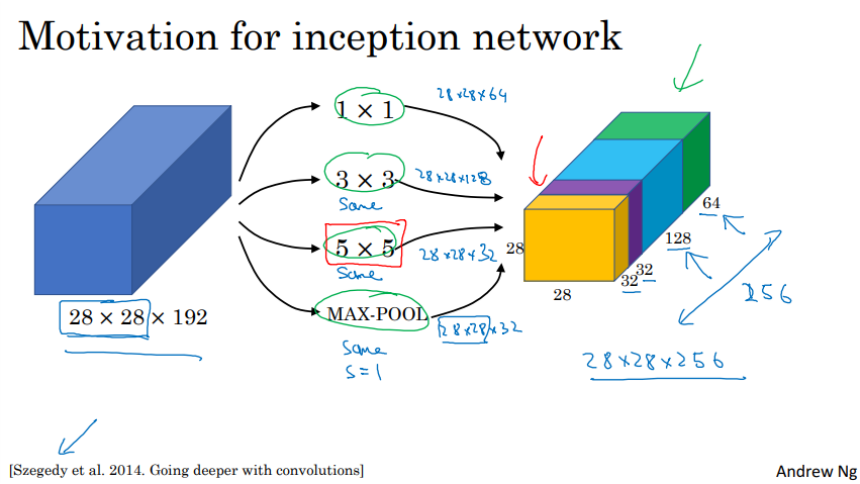
[Lin et al., 2013. Network in network] Andrew Ng

1*1 합성곱이 유용한 실제 경우를 살펴보면, 28*28*192의 입력이 있다고 하자. 만약 높이와 너비를 줄이고 싶다면 풀링 층을 사용하면 된다. 그렇다면 만약 채널의 수가 너무 많아서 줄이려면 어떻게 해야 할까? 어떻게 28*28*32로 줄일 수 있을까. 바로 32개의 1*1필터를 사용하면 된다. 실제로는 각 필터가 1*1*192의 크기를 가질 것이다. 필터와 입력의 채널 수가 일치해야 하기 때문이다. 만약 32개의 필터를 사용하면 출력은 28*28*32이 크기를 가질 것이다. 또한 이것은 nC를 줄이는 방법이기도 하다. 풀링 층은 높이와 너비인 nH와 nW만 줄일 수 있는 반면에 말이다. 이후 어떻게 1*1 합성곱이 채널 수를 줄여서 네트워크 계산을 용이하게 하는지 살펴볼 것이다. 물론 192개

의 채널 수를 유지해도 괜찮다. 1*1 합성곱의 효과는 비선형을 더해주고 하나의 층을 더해줌으로써 더 복잡한 함수를 학습할 수 있다. 28*28*192를 입력 받아 28*28*192를 출력하는 층 말이다. 이것이 1*1 합성곱 층이 하게되는 중요한 역할이다. 네트워크에 비선형성을 더해주고 채널의 수를 조절할 수 있게 된다. 이것이 인셉션 신경망 구축에 매우 유용하게 사용되는데, 다음 섹션에서 살펴볼 것이다. 즉 1*1 합성곱 신경망은 채널의 수를 원하는대로 줄이거나 늘릴 수 있게 만들어주는 매우 중요한 역할을 한다.

📌Inception 네트워크의 아이디어(C4W2L06)

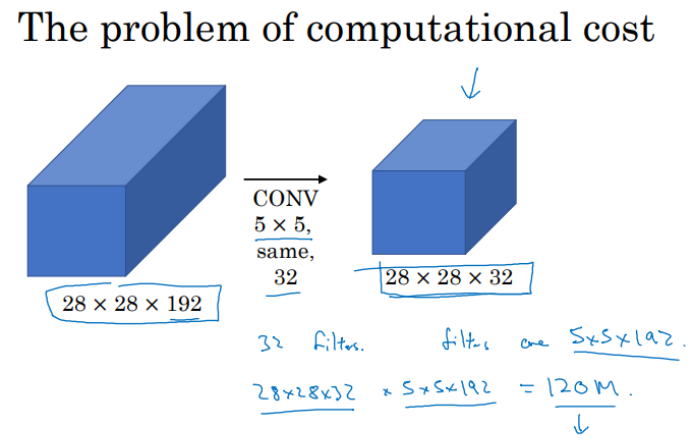
핵심어: 인셉션 네트워크(Inception Network), 병목 층(botleneck layer)



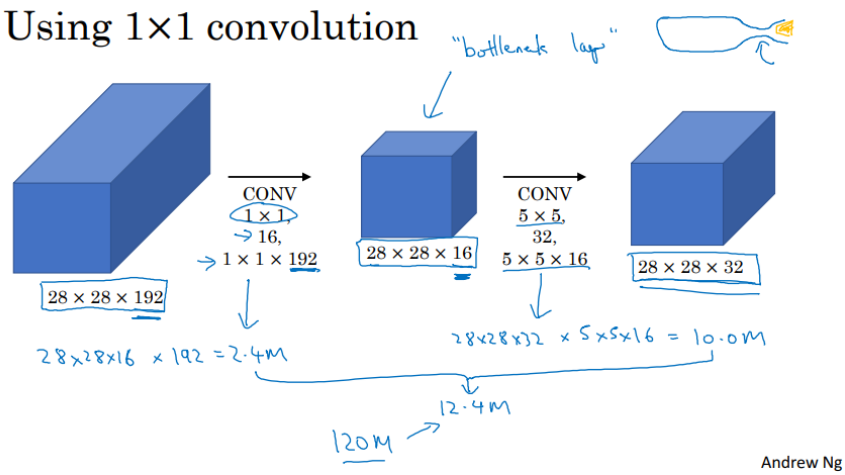
합성곱 신경망의 층을 디자인할 때 1*3 필터를 사용할지 아니면 3*3, 5*5 혹은 풀링층을 원하는지 결정해야 한다. 인셉션 네트워크가 하는 것은 이것들을 모두 합친 것이다. 네트워크가 복잡해지기는 하지만 성능은 뛰어나다. 예시로 28*28*192 크기의 입력값이 있다고 하자. 인셉션 네트워크는 필터의 크기를 정하지 않고 합성곱 또는 풀링층을 모두 사용하는 것이다. 1*1 합성곱을 이용하면 28*28*64의 크기를 가진다. 그리고 만약 3*3를 사용하게 된다면 28*28*128이 될 것이고 이 두 번째 볼륨을 첫 번째 볼륨 위에 쌓는 것이다. 크기를 맞추기 위해 동일 합성곱을 사용해서 출력이 28*28로 유지되게 해준다. 입력 크기인 28*28과 같게 유지가 되는 것이다. 또한 5*5 필터가 나아 보여서 해보면 28*28*32가 된다. 역시 동일 합성곱을 사용하여 크기를 유지한다. 합성곱 층을 원하지 않는다면 풀링 층을 적용한다. 또 다른 출력이 나올텐데 역시 같이 쌓아준다. 풀링의 출력 크기는 28*28*32가 된다. 단 최대 풀링에서 크기를 맞추려면 패딩을 사용해야 하는데, 이는 보기 드문 풀링 형태이다. 만약 높이와 너비를 28*28

로 하고 다른 출력들의 크기와 맞추려면 패딩과 1의 스트라이드를 사용해야 한다. 이 세세한 것이 지금은 우스워 보일 수 있지만 일단은 넘어가자. 이후 모든 것이 합쳐질 것이다. 이러한 인셉션 모듈에서는 특정한 크기를 입력하면 출력은 이 숫자들(32+32+128+64)을 전부 더한 크기이다. 이 값은 256이다. 이 인셉션 모듈의 입력은 28*28**192이고 출력은 28*28*256이 되는 것이다. 이것이 바로 인셉션 네트워크의 핵심이다.

기본적인 개념은 필터의 크기나 풀링을 결정하는 대신 그것들을 전부 다 적용해서 출력들을 다 엮어낸 뒤 네트워크가 스스로 원하는 변수나 필터 크기의 조합을 학습하는 것이다. 그리고 여기 인셉션에 문제가 조금 있는데, 바로 계산 비용이다.



5*5 필터로 만들어진 블록의 계산 비용을 알아보도록 하자. 이 부분만 집중해서 보면, 일단 입력으로는 28*28*192 블록이 있고 5*5 동일 합성곱의 필터 32개로 28*28*32의 출력이 나오게 된다. 앞에서는 이를 얇은 보라색 조각으로 표현했는데 여기서는 파란 블록으로 나타냈다. 이제 이 28*28*32 출력의 계산 비용을 알아보자. 우선 32개의 필터가 있다. 출력값이 32개의 채널이기 때문이다. 그리고 각 필터는 5*5*192이다. 출력의 크기가 28*28*32라서 23*23*32개의 숫자를 계산해야 한다. 각각의 수에 5*5*192의 곱셈을 해야하며 총 필요한 곱셈의 수는 각각의 출력값을 계산하기 위한 곱셈의 수에다가 출력값의 개수를 곱한 수가 되는 것이다. 이 모든 수를 곱하면 1억 2천만 정도가 된다. 현대의 컴퓨터에서 1억 2천만의 곱셈을 할 수는 있지만 여전히 비용이 큰 계산이다. 여기서 1*1 필터를 사용해 계산 비용을 줄일 수 있다. 1억 2천만의 10분의1 정도로 말이다.

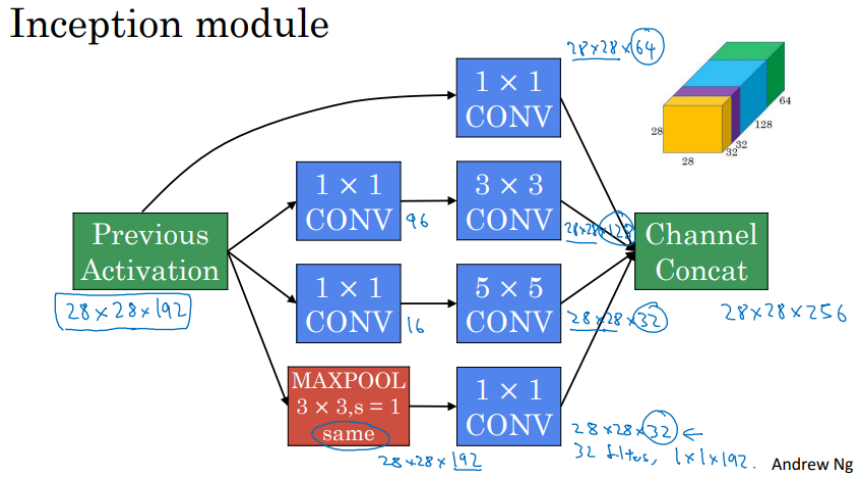


Andrew Ng

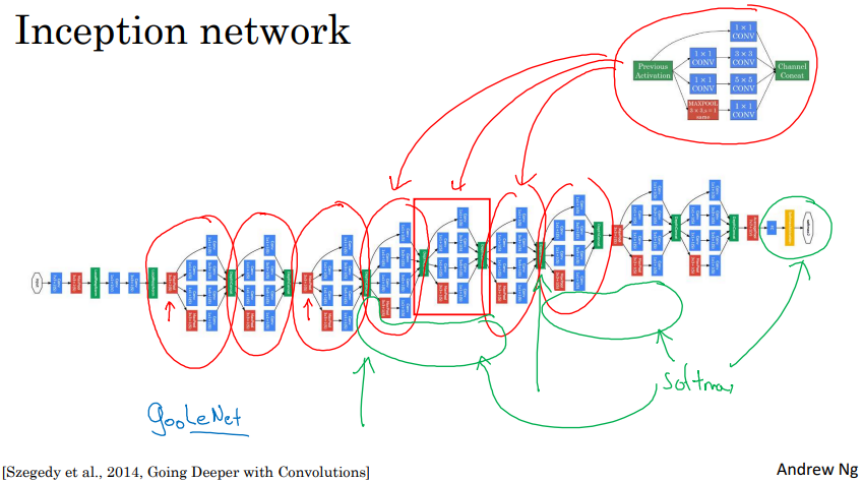
여기 또 다른 방식으로 28*28*192의 입력으로 28*28*32를 출력하는 구조가 있는데 다음과 같다. 볼륨을 입력받고 1*1 합성곱을 사용해서 192개의 채널을 16개로 줄이고 이 볼륨에 5*5 합성곱을 해주면 최종 출력을 얻게 된다. 입력과 출력 크기는 전과 같다. 입력은 28*28*192이고 출력은 28*28*32이다. 여기서 한 일은 왼쪽의 큰 볼륨을 가지고 여기 중간 크기의 볼륨으로 줄인 것이다. 192개 대신 16개의 채널만 가지고 있다. 이것은 병목 층이라고도 불린다. 일반적으로 병의 목은 어떤 것의 가장 작은 부분을 가리키기 때문이다. 병목층도 네트워크에서 가장 작은 부분을 나타낸다. 크기를 다시 늘이기 전에 이미지를 줄이는 것이다. 이것의 계산 비용을 한 번 알아보자. 1*1 합성곱을 사용하기 위해서는 16개의 1*1*192의 필터가 필요하다. 그래서 이 28*28*16의 출력을 계산하기 위한 비용은 28*28*16만큼의 출력이 있고 각각 192번의 곱셈이 필요하다. 이를 계산하면 240만 정도이다. 이것이 첫 번째 합성곱 층의 비용이다. 그럼 두 번째 계산을 보자. 두 번째 합성곱 층의 비용은 우선 28*28*32만큼의 출력값이 있고 각 출력값마다 5*5*16의 필터를 적용해줘야 한다. 이를 계산하면 1천만 저도이다. 그래서 총 필요한 곱셈의 수는 240만과 1천만의 합이다. 이전 슬라이드와 비교하면 계산 비용을 1억 2천만개의 곱셈에서 10분의 1 정도인 1천 2백만 곱셈으로 줄인 셈이 된다. 필요한 덧셈의 수는 곱셈의 수와 비슷하기 때문에 곱셈의 수만 계산하는 것이다. 정리하자면 신경망을 구축할 때 1*1, 3*3, 5*5 또는 풀링 층인지 고민하기 원하지 않는다면 인셉션 모델은 그것들을 전부 다 실행해서 함께 엮는 것이다. 그리고는 계산 비용 문제에 대해 1*1 합성곱을 이용해서 병목 층을 만들어 계산 비용을 상당히 많이 줄일 수 있다. 표현 크기를 줄이는 것이 성능에 지장을 줄지 걱정될 수 있는데 만약 이 병목 층을 적절하게 구현할 수 있다면 표현 크기를 줄이는 동시에 성능에 큰 지장 없이 많은 수의계산을 줄일 수 있다.

📌Inception 네트워크(C4W2L07)

핵심어: 인셉션 네트워크(Inception Network), 구글넷(GoogLeNet)



이전에 인셉션 네트워크의 기본 구성 요소들을 살펴봤는데, 이번에는 이러한 구성 요소들을 조합해서 인셉션 네트워크를 구축해 볼 것이다. 인셉션 모듈은 활성값이나 이전 층의 출력을 입력값으로 받는다. 이를 이전 영상과 같이 28*28*192라고 해보자. 전에 살펴봤던 예시는 1*1과 5*5의 층을 사용한 것인데 1*1의 경우에는 16개의 채널이 있고 5*5의 경우에는 28*28*32의 출력 크기를 가졌다. 3*3 합성곱의 계산을 줄이기 위해 동일한 것을 수행할 수 있었고, 3*3의 경우에는 28*28*128의 출력이 있다. 또 1*1 합성곱을 해줄 수도 있다. 1*1 합성곱에는 또 다른 1*1 합성곱이 필요하지는 않다. 그리고 출력으로는 28*28*64가 나온다고 한다. 마지막으로는 풀링 층이 있는데, 여기서 조금 우스운 것은 이 결과들을 엮어주기 위해 풀링에 동일 패딩을 적용해서 높이와 너비가 28*28로 유지도어서 다른 결과와 엮을 수 있다. 하지만 동일 패딩을 가진 최대 풀링을 사용한다면 3*3 필터에 스트라이드를 1로 놓는다면 그 결과는 28*28*192의 크기를 가진다. 입력과 동일한 수의 채널을 가지게 되는 것이다. 딱 보기에 채널이 너무 많아 보이는데, 여기 1*1의 합성곱 층을 추가해줘서 채널의 수를 줄여 28*28*32로 만드는 것이다. 그러기 위해서는 32개의 1*1*192 필터가 필요하며 출력의 채널 수가 32가 줄어들게 된다. 풀링 층이 채널을 모두 차지하지 않게 말이다. 마지막으로 이 블록들을 모아 연결해준다. 64+128+32+32개를 하나로 연결하게 되면 28*28*256 크기가 된다. 채널 연결은 이전 블록들을 연결해주는 것이다. 이것이 하나의 인셉션 모듈이고 ,인셉션 네트워크는 이런 모듈들을 하나로 모아놓은 것이다.



위는 논문에서 가져온 인셉션 네트워크 그림이다. 많은 블록들이 반복되고 보기에 복잡해 보이지만 하나의 블록을 살펴보면 아까 보았던 인셉션 모듈임을 알 수 있다. 여기에는 차원을 바꾸기 위한 최대 풀링 층이 추가적으로 있고, 여러 인셉션 블록들이 있다. 인셉션 네트워크는 방금 보았던 이런 블록들이 네트워크의 서로 다른 곳에 반복되는 것이다.

원래의 논문을 읽으면 인셉션 네트워크에 대한 또 다른 점이 있는데 곁가지들이 있다는 것이다. 네트워크의 마지막 몇 개의 층은 완전연결 층이고 그 뒤에는 예측을 위한 소프트맥스 층이 있는데, 이 곁가지가 하는 일은 은닉 층을 가지고 예측을 하는 것이다. 그리고 또 다른 곁가지도 은닉층을 가지고 완전 연결층을 지나서 소프트맥스로 결과를 예측한다. 이것을 인셉션 네트워크의 또 다른 세부사항이라 할 수 있지만, 은닉층이나 중간 층에서 계산된 특성들이라도 이미지의 결과를 예측하는 데 아주 나쁘지는 않다는 것이다. 인셉션 네트워크에 정규화 효과를 주고 네트워크의 과대적합을 방지해준다. 그리고 이 인셉션 네트워크는 Google의 일원에 의해 개발되어 GoogLeNet으로 나타낸다.

정리하자면, 인셉션 모듈을 이해하면 인셉션 네트워크를 이해할 수 있다. 인셉션 모듈이 네트워크 상에서 반복되는 것이기 때문이다. 기존의 인셉션 모듈이 개발되던 때부터 계속 새롭게 만들고 다른 버전을 내왔다. 새로운 인셉션 알고리즘에 대한 논문도 있다. 이후에 나온 인셉션 알고리즘들 중 하나는 ResNet의 스킵 연결을 활용하는데, 훨씬 좋은 성능을 보인다. 그러나 이 모든 것들이 기초를 두고 있는 것은 다수의 인셉션 모듈을 모아서 쌓는다는 개념이다.

<인셉션 네트워크>

1. 먼저 인셉션 모듈은 활성값 또는 이전 층의 출력을 입력으로 받는다. 이 예에서는 입력의 크기를 28x28x192라고 가정한다.
2. 이 입력에 대해 1x1, 3x3, 5x5 크기의 합성곱 필터와 3x3 크기의 최대 풀링이 동시에 적용된다. 이때 각 크기의 합성곱 필터는 입력의 모든 위치에 적용되어 그 결과를 새로운 특징 맵으로 만든다.

3. 각 합성곱 필터와 풀링은 다른 개수의 채널을 생성한다. 예를 들어, 1x1 합성곱은 16개의 채널을, 5x5 합성곱은 32개의 채널을, 3x3 합성곱은 128개의 채널을 생성하고, 풀링은 입력과 동일한 192개의 채널을 유지한다.
4. 풀링은 일반적으로 출력의 크기를 줄이지만 여기서는 동일 패딩을 사용하여 출력 크기를 입력과 동일하게 유지한다. 그러나 이 경우 채널 수가 너무 많아지므로, 풀링 후에 1x1 합성곱을 추가적으로 적용하여 채널 수를 32로 줄인다.
5. 마지막으로 모든 합성곱과 풀링의 결과를 채널 방향으로 합치며 이를 '채널 연결'이라고 한다. 이렇게 하면 최종 출력은 28x28 크기의 특징 맵이고, 채널 수는 각 결과의 채널 수를 모두 더한 256이 된다.
6. 이렇게 생성된 인셉션 모듈이 인셉션 네트워크의 기본 구성 단위가 된다. 인셉션 네트워크는 이런 모듈들을 여러 개 쌓아서 구성한다.

결국 인셉션 네트워크는 서로 다른 크기의 합성곱 필터를 동시에 적용하여 다양한 종류의 특징을 동시에 잡아내는 능력을 가지고 있다. 이런 특징이 인셉션 네트워크를 대규모 이미지 분류 문제에 효과적으로 만든다.

'DL 스터디&프로젝트' 카테고리의 다른 글

[Euron 중급 세션 16주차] 딥러닝 4단계 1. 합성곱 신경망 (1)	2023.12.25
[Euron 중급 세션 15주차] 딥러닝 2단계 7. 다중 클래스 분류 ~ 8. 프로그래밍 프레임워크 소개 (1)	2023.12.18
[Euron 중급 세션 12주차] 딥러닝 2단계 6. 배치 정규화 (1)	2023.11.27
[Euron 중급 세션 11주차] 딥러닝 2단계 5. 하이퍼파라미터 튜닝 (1)	2023.11.20
[Euron 중급 세션 10주차] 캐글 필사 과제 - Pytorch Tutorial for Deep Learning Lovers (2)	2023.11.20

관련글