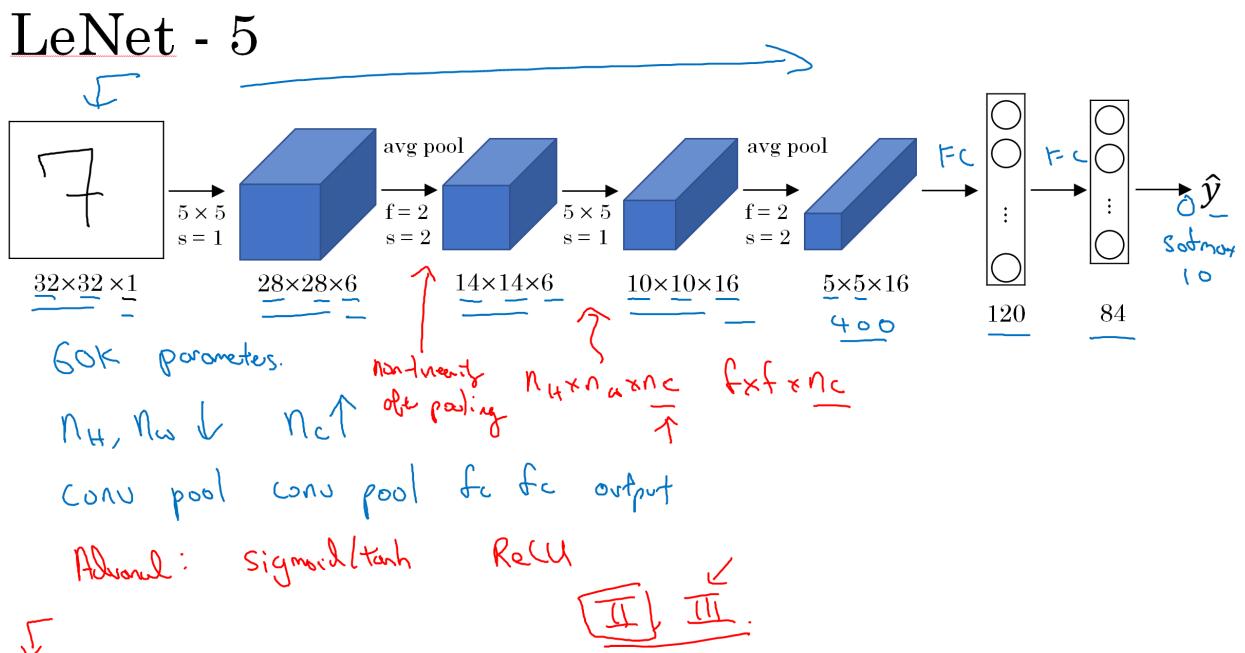


[2주차] Deep convolutional models : case studies

개요

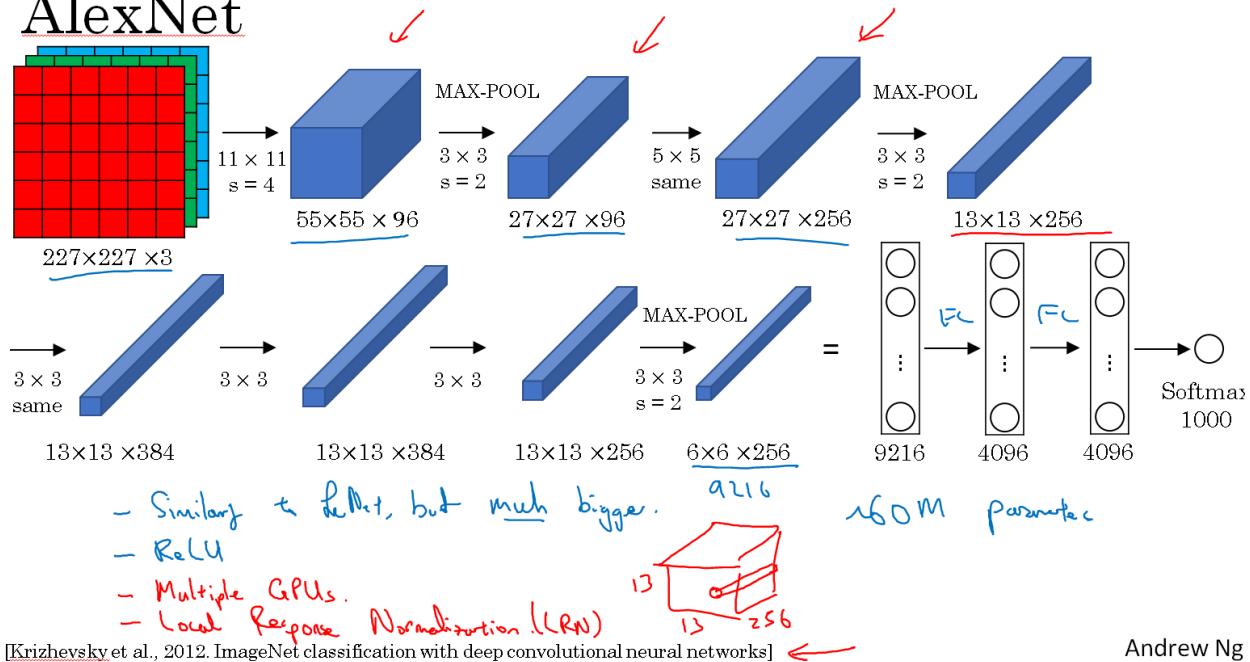
1. 몇 가지 고전 네트워크

- LeNet-5 network
- AlexNet
- VGG Network



- 왼쪽에서 오른쪽으로 갈수록 높이와 폭이 줄어드는 경향

AlexNet



[Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks]

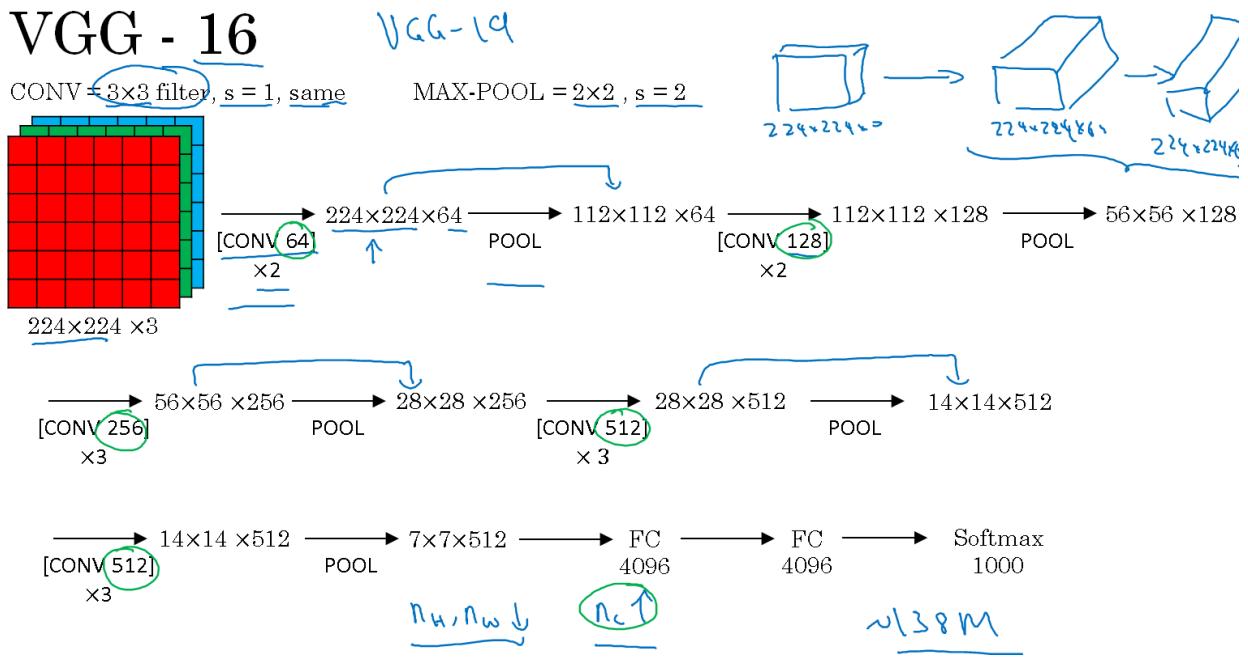
Andrew Ng

- 알렉스넷 입력은 $227 \times 227 \times 3$ 이미지로 시작
- 큰 스트라이드 4를 사용하기 때문에. 디멘션은 55×55 로 줄어듬
- 이 신경망은 실제로 LeNet과 많은 유사점을 가졌지만 훨씬 더 커짐
- LeNet-5에는 약 60,000 개의 파라미터가 있었지만 알렉스넷에는 약 6 천만 개의 파라미터
- 매우 많은 히든 유닛을 가지고 있고, 더 많은 데이터 상에서 학습
- LeNet보다 훨씬 나은 면은 **ReLU** 액티베이션 기능을 사용하고 있다는 것
- 원래의 알렉스넷 아키텍처에도 LRN(Local Response Normalization)이라는 레이어 세트가 추가

▼ LRN

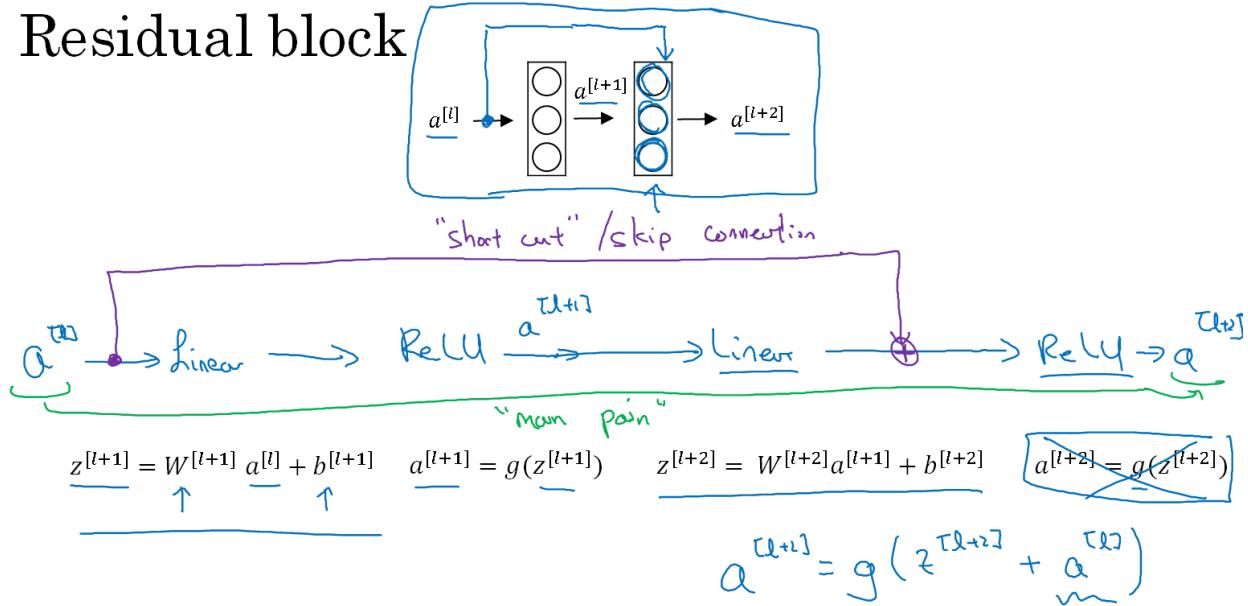
- LRN (로컬 응답 정규화)가하는 일은 한 위치를 보는 것
- 그 하나의 위치의 높이와 너비를 보는 것
- 그리고 이 모든 채널에 걸쳐 내려다 보고 이 모든 256 숫자를 보고 모두를 정상화시키는 것
- 이 로컬 응답 정규화의 동기는 이 13×13 이미지의 각 위치에 대해 매우 높은 액티베이션을 가진 너무 많은 뉴런을 갖지 않게 하기 위함

→ 그러나 이게 많은 도움이 되지는 않음



- 많은 하이퍼 파라미터를 가지고 있는 대신에, 더 단순한 네트워크를 사용
- 단지 스트라이드 1, 언제나 동일한 패딩의 3×3 필터인 컨볼레이어를 사용하는 것에 초점을 맞추고 그리고 스트라이드 2 를 가진 2×2 , max pooling레이어를 만든다는 점
- 아주 좋은 점 중 하나는 이 신경망 아키텍처를 단순화 하는 것
- 이 네트워크는 약 1 억 3 천 8 백만 개의 파라미터
- 더 깊게 높이와 너비가 내려감에 따라 채널의 수는 증가하지만 pooling레이어가 매번 2 배씩 내려간다는 것

Residual block

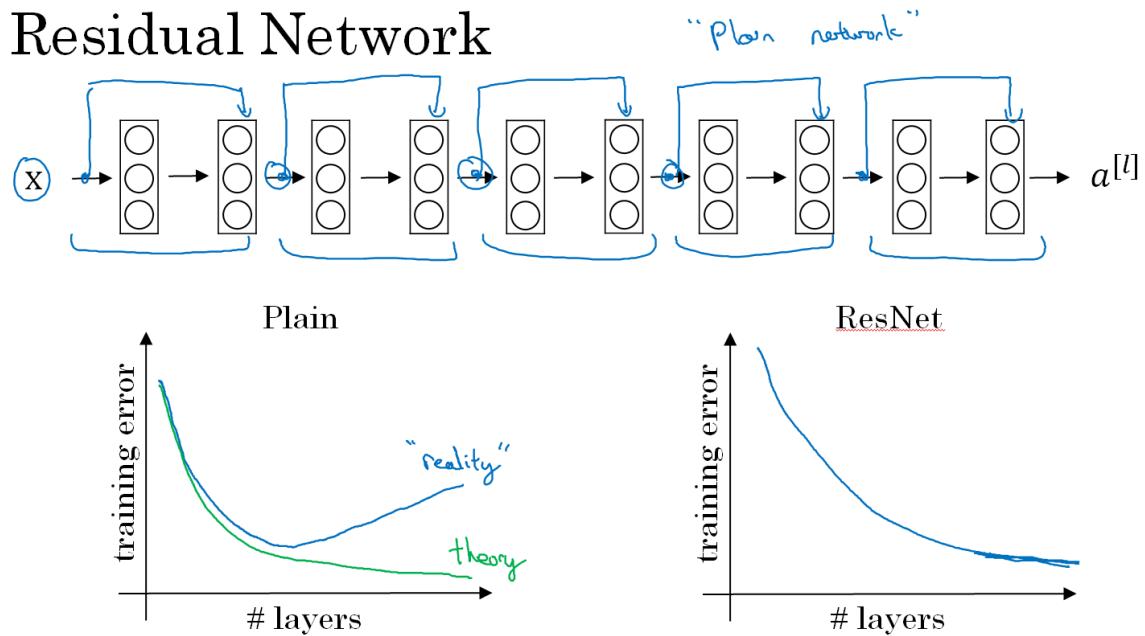


[He et al., 2015. Deep residual networks for image recognition]

Andrew Ng

- [l]에서 [l + 2]로 이동하기 위해서는 이 모든 단계를 거쳐야 하는데요, 이걸 레이어 세트의 주요 경로
- 주 통로를 따라갈 필요 없이 $a^{[l]}$ 의 정보는 이제 신경망로 훨씬 더 깊숙이 들어가는 지름길을 따라갈 수 있습니다. → short cut
- 이 마지막 방정식은 없애버리고, 대신에 아웃풋 $a^{[l + 2]}$ 는 ReLU 비선형성 $g(z^{[l + 2]})$ 라고 앞에 했던 것처럼 똑같이 쓰고 이제는 $a^{[l]}$ 을 더하면 됩니다. 따라서, 여기서 이 $a^{[l]}$ 을 덧셈하면 이게 잔여 블럭을 만들게 되는 거죠.
- 따라서 $a^{[l]}$ 은 선형 부분이면서, ReLU 부분 앞쪽인 곳에 들어가게 됩니다. 때로는 shortcut이라는 용어 대신, skip connection이라는 용어를 듣게 될 텐데요, 이것은 신경망 속으로 더 깊숙이 들어가 정보를 처리하기 위해 $a^{[l]}$ 이 레이어를 하나씩 혹은 두 개씩 건너뛰는 것을 가리키는 것
- ResNet을 구축하는 방법은 이러한 잔여 블럭을 많이 사용하여 심층 신경망을 형성하기 위해 이렇게 겹쳐 쌓아 올리는 것

Residual Network



[He et al., 2015. Deep residual networks for image recognition]

Andrew Ng

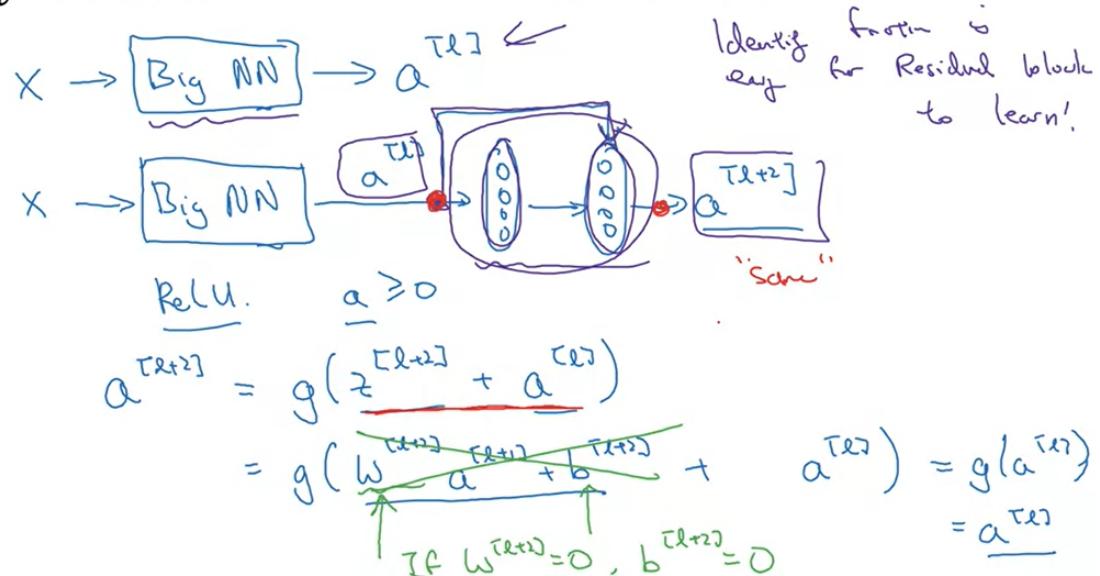
- 이것은 잔류블럭이 아니라 plain network (평범한 망)
- 이것을 ResNet으로 바꾸려면 모든 skip connection을 추가해야 함
- 사라지거나 폭발적으로 증가하는 gradient 문제점들을 처리하도록 도와주고, 수행 시 눈에 띄는 손실 없이 신경망을 훨씬 깊게 훈련하도록 해줍니다.
- 하지만 ResNet은 심층 신경망 깊이 훈련하도록 도와주는 데에는 그리 효과적이지는 않습니다.

ResNet은 왜 잘 작동되는 걸까?

- 트레이닝 세트에서 잘 수행하는 것이 그 다음에 있는 홀드아웃 와 테스트 세트를 잘 수행 할 수 있는 조건

예시

Why do residual networks work?



Andrew Ng

- 네트워크를 더 깊이 만들면 트레이닝 세트에서 잘 수행할 수 있게 하려다가 네트워크를 학습시키는 능력에 손상을 줄 수 있다

→ 그런 이유로 때로는 지나치게 깊은 네트워크는 원치 않게 되는 것이죠.

→ 그러나 여러분이 ResNet를 학습시킬 때에는 이 말은 사실이 아니거나 적어도 사실일 가능성이 매우 적습니다!

- X 가 큰 신경망에 입력되어 있고 아웃풋으로 액티베이션 $a[1]$ 을 가지고 있다고 가정
- 이 예제에서 신경망을 **조금 더 깊게** 만들고자 수정한다고 가정
- 따라서 동일한 큰 NN을 사용하면 이 아웃풋은 $a[1]$ 이 되고, 여기에 레이어를 하나 더 추가하고, 또 다른 레이어는 여기에 둡시다. 그럼 이제 아웃풋은 $a[l + 2]$ 가 되는군요.
- 이걸 추가된 shortcut을 이용해서 ResNet 블럭, 즉 잔류블럭으로 만들어봅시다.
- 이 네트워크 전반에 걸쳐 ReLU 액티베이션 함수를 사용한다고 가정

→ 모든 액티베이션 0보다 크거나 같을 것

- 만약 여러분이 K의 방법으로서 L2 일반화를 사용하고 있다면 그것은 $w[l + 2]$ 값이 줄어드는 경향이 있을 것이다, K에서 B로 적용하고 있다면, 이것 또한 이것을 줄어들게 할 것입니다. 실제로 여러분이 K에서 B로 적용을 해보기도 하고 하지 않기도 할 테지만, W는 정말로 우리가 주목해야 할 핵심포인트입니다.

- $w[l + 2] = 0$ 이고, 토론을 위해, $B[l + 2] = 0$ 라고 가정
- 그러면 이것들은 0이 되기 때문에, 이 부분은 없어질 것
- $g(a[l])$ 은 $a[l]$ 과 같아질 것입니다.

→ 왜냐하면 ReLU 액티베이션 함수를 사용하고 있다고 가정했기 때문이죠.

→ 이는 이 두 추가적인 레이어가 없는 더 단순한 네트워크에서든 신경망에서 있는 **이 두 레이어를 더하던 순간에 이것이 신경망의 능력을 저하시키지 않는다는 것을 의미합니다.**

- 왜냐하면 항등 함수를 학습하는 것을 매우 쉽기 때문

→ 여기 이 두 레이어를 더해서 $a[l]$ 를 $a[l + 2]$ 으로 똑같이 복사해주만 하면 되니까요. 이게 바로 두 개의 추가 레이어를 더하고 이 잔류블럭을 여기 큰 신경망의 중간이나 끝 어딘가에 더해도 수행능력은 저해되지 않는 이유인 것입니다.

- 하지만 물론 우리의 목표는 수행능력을 떨어뜨리지 않는 게 아니고 수행능력을 도와주는 것입니다.
- 네트워크를 점점 깊게 만들면, 이 네트워크가 항등 함수를 학습하는 파라미터를 선택하게 하기란 사실 매우 어려운 일입니다. 그리고 많은 레이어들이 결국 결과를 더 좋게 하기보다 더 악화시키는 이유이기도 합니다.
- 레지듀얼 네트워크가 작동하는 주된 이유는 이 추가적인 레이어들이 항등 함수를 학습하는 게 너무 쉽기 때문에, 수행이 저해되는 일이 없고 심지어 더 잘 수행할 것임을 보장받을 수 있기 때문입니다.
- 이 편집 본을 통해 논의할 가치가 있는 잔류 네트워크의 또 하나의 세부사항은 우리가 $z[l + 2]$ 와 $a[l]$ 가 같은 차원을 가질 거라고 가정하는 것입니다.
- ResNet에서 봐야 할 것은 같은 컨볼루션을 많이 사용해서 이 다이멘션이 이쪽 레이어나 아웃풋 레이어의 다이멘션과 같아지도록 하는 겁니다.
- 같은 컨볼루션은 다이멘션을 그래도 유지해주고, 여러분이 이 short circle을 수행하거나 두 동일한 다이멘션 벡터의 덧셈을 하는 것을 더욱 쉽게 만들어줍니다.

ReLU.

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

$$= g(w^{[l+2]} a^{[l+1]} + b^{[l+1]}) + w_s a^{[l+2]} = g(a^{[l]}) = a^{[l]}$$

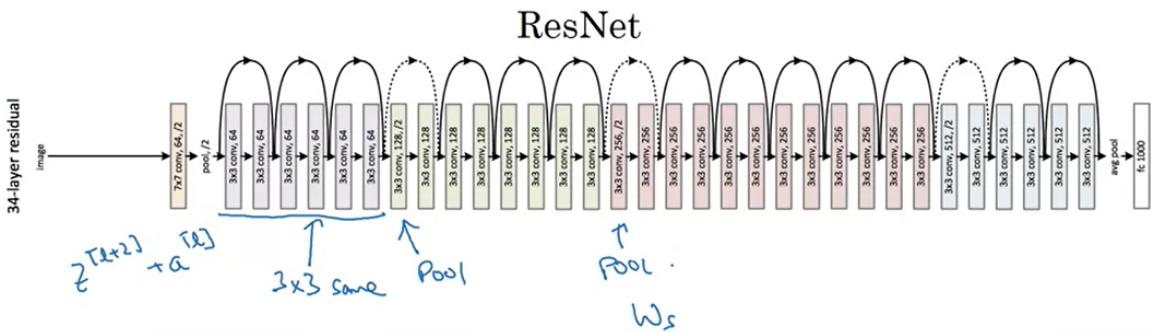
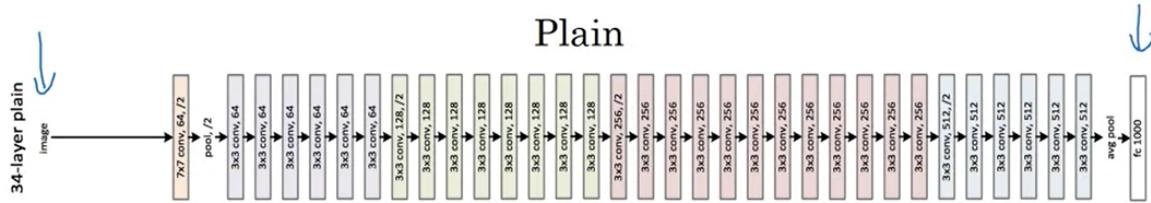
if $w^{[l+2]} = 0, b^{[l+2]} = 0$

same 256×128

- 인풋과 아웃풋이 다른 다이멘션을 가지는 경우를 예시로 들면, 만약 이게 128 차원이고, Z 니까, 이 $a[i]$ 이 256 차원이라면, 추가 매트릭스 즉 W_s 를 여기에 더해주십시오 이 예시의 W_s 는 256×128 차원의 매트릭스가 될 것입니다. 따라서, $W_s \times a[i] = 256$ 차원이 되고 이 덧셈은 이제 2개의 256 다이멘션의 벡터 사이에 있는 것입니다.

이미지 상의 ResNet

ResNet



[He et al., 2015. Deep residual networks for image recognition]

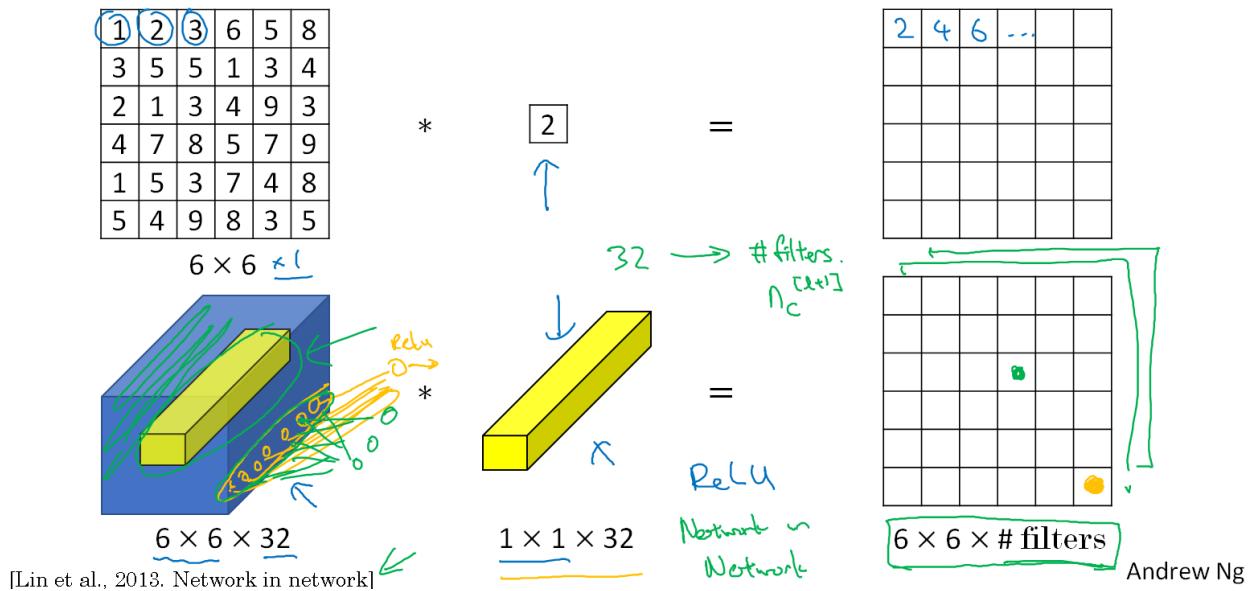
Andrew Ng

- 이걸 ResNet으로 전환시키려면, 이 추가 skip connection을 더하십시오.

- 차원들이 유지되고 있는 동일한 컨볼루션이기 때문에, $z[l + 2] + a[l]$ 에서 이 덧셈이 이 치에 맞게 되는 겁니다. 앞서 NetRes에서 보았던 것과 유사하게, 상당한 컨볼루션 레이어가 있고, 간헐적으로 pooling layer들도 보입니다, 혹은 pooling 같아 보이는 것들도 있죠. 이런 것들이 생겨날 때마다, 이전 슬라이드에서 보았던 차원을 조정해야 할 필요가 있습니다. Ws 매트릭스를 할 수 있고, 이 네트워크에 보이는 대로 컨볼, 컨볼, 컨볼, pool, 컨볼, 컨볼, 컨볼, pool 이런 식으로 되어있습니다.<unknown> 그리고 마지막에는 softmax를 이용하여 예측을 하는 완전 연결 레이어가 있습니다. 자, 여기까지가 ResNet에 대한 것입니다. 신경망을 사용하는 것에는 아주 흥미로운 아이디어가 있습니다. 바로 1×1 필터, 1×1 컨볼루션입니다. 따라서, 1×1 컨볼루션을 사용할 수 있게 될 것입니다. 다음 강의를 한 번 보시죠.

1x1 convolutions

Why does a 1×1 convolution do?



- 1×1 컨볼루션이 수행 할 작업은 여기에 있는 36 가지 위치를 각각 살펴보고 왼쪽의 32 개 숫자와 필터의 32 개 숫자 사이에 요소 간 곱셈을 하는 것
- 그리고 나서 여기에 ReLU 비선형성을 적용

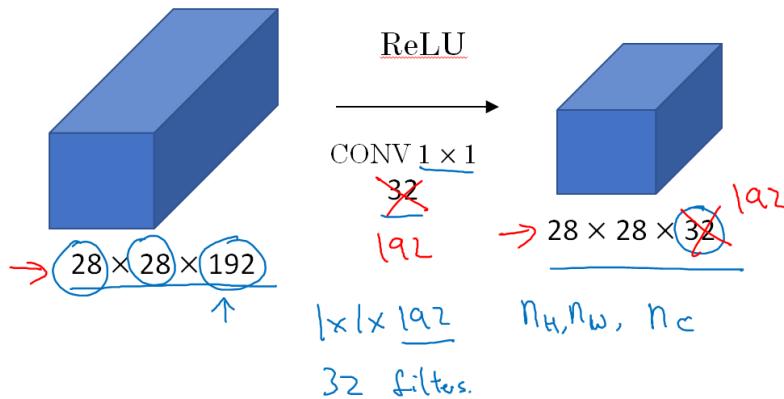
- 일반적으로, 필터가 하나가 아니라 필터가 여러 개인 경우 하나의 유닛이 아닌, 여러 개의 유닛을 가진 것과 같습니다.

→ 하나의 조각에 모든 숫자를 인풋한 다음 $6 \times 6 \times$ 필터의 개수의 아웃풋으로 이것들을 만들 어내면 되는 것!

- 1×1 컨볼루션에 대해 생각해볼 수 있는 한가지 방법은 기본적으로 완전 연결 신경망을 사용하여 62 개의 서로 다른 위치에 적용하는 것
- 완전 연결 신경망은 무엇을 할까요? 인풋이 32개고 아웃풋은 필터의 개수
- 각각의 36개의 위치, 즉 각각 6×6 위치에서 이렇게 함으로서 $6 \times 6 \times$ 필터의 개수인 아웃풋을 만들어낼 수 있습니다.

[예시]

Using 1×1 convolutions



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

Andrew Ng

Q. 많은 채널 중의 하나가 과하게 커서 이걸 줄이고 싶다면, $28 \times 28 \times 32$ 차원 볼륨으로 어떻게 줄일 수 있을까요?

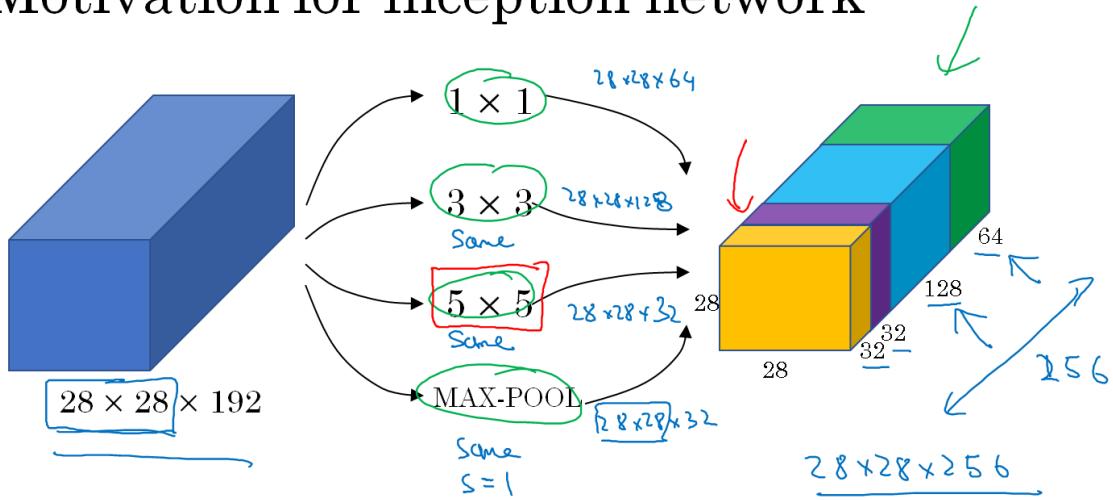
A. 1×1 인 32 개의 필터를 사용하는 것

→ 기술적으로 각각의 필터는 $1 \times 1 \times 192$ 입니다. 필터안에 있는 채널의 수는 여러분의 인풋 볼륨에 있는 채널의 숫자와 매치가 되야 하기 때문입니다. 하지만 32개의 필터를 사용하면,

이 과정의 아웃풋이 $28 \times 28 \times 32$ 볼륨이 되는 겁니다.

Inception Network Motivation

Motivation for inception network

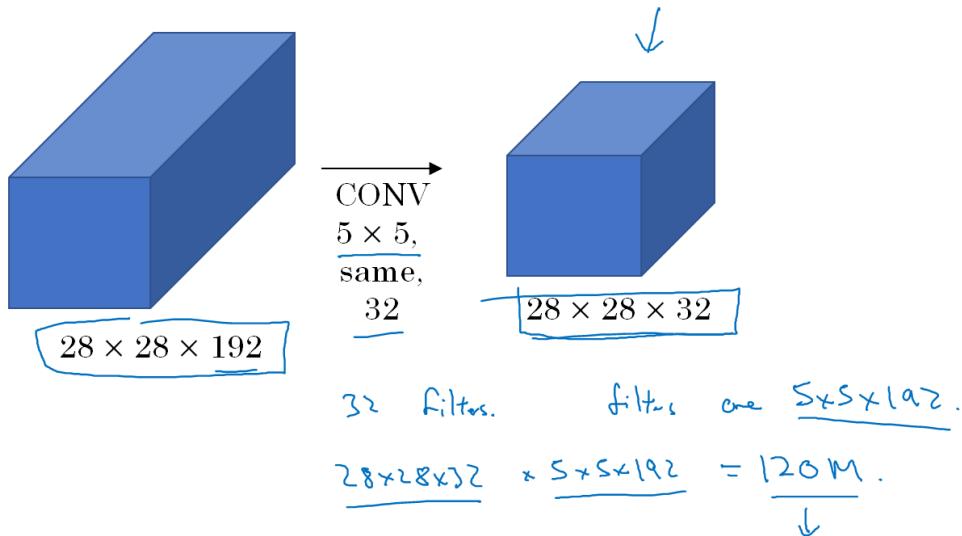


[Szegedy et al. 2014. Going deeper with convolutions]

Andrew Ng

- 인셉션망 또는 인셉션 레이어가 말하고자 하는 것은 컨볼 레이어에서 여러분이 원하는 필터사이즈를 고르는 대신에 여러분은 컨볼루션 레이어와 pooling 레이어 중에 어떤 것은 원하는지 입니다.
- 3×3 을 하고싶고, output차원이 28×28 이 되고싶다면 same padding 사용
- 5×5 도 마찬가지
- 그 다음 max-pooling 사용 (동일 아웃풋을 위해 padding=same, stride=1)
- 결과를 차례로 쌓고 모두 더하면 채널은 256이 됨

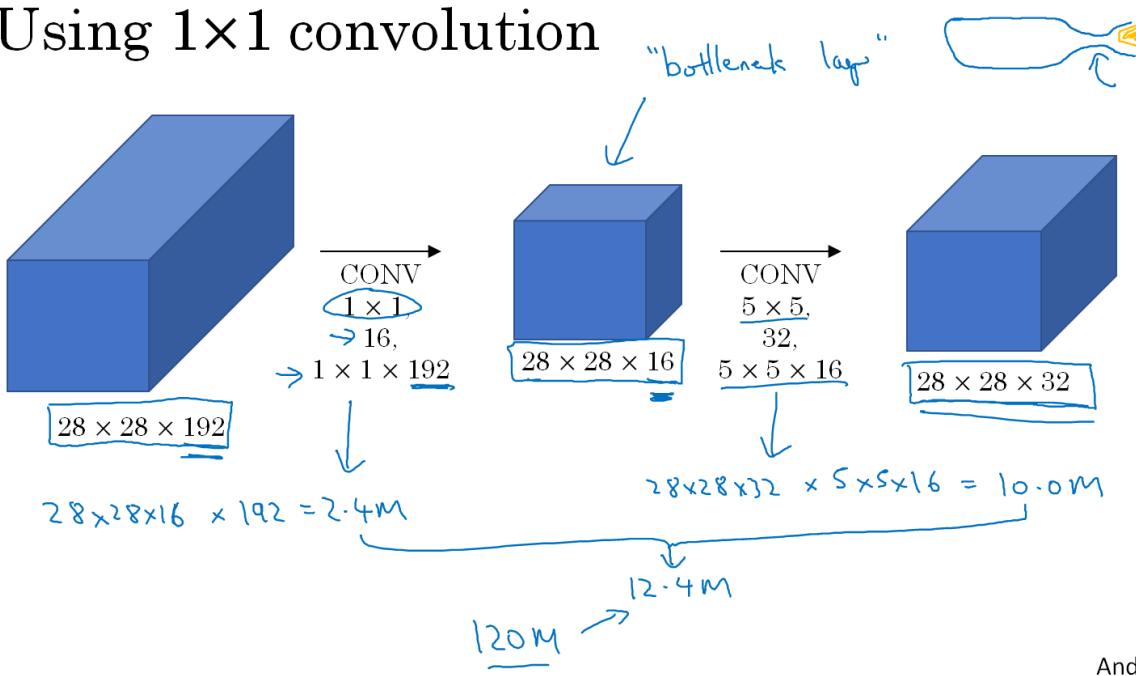
The problem of computational cost



Andrew Ng

- 이전 슬라이드의 5×5 포트에 초점을 맞추기 위해 $28 \times 28 \times 192$ 블록을 인풋하고 32 개 필터를 가진 5×5 동일한 컨볼루션을 $28 \times 28 \times 32$ 아웃풋으로 실행
- 곱셈의 총액은 1억 2천만 → 비싼 작업임

Using 1×1 convolution



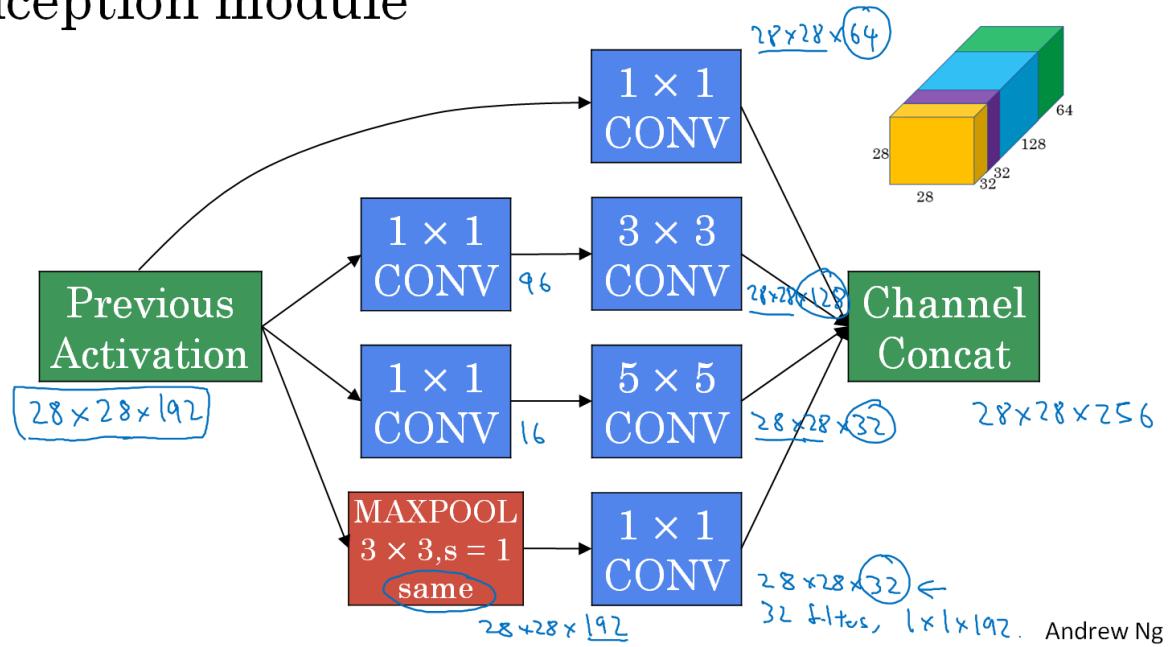
Andrew Ng

- 1×1 컨볼루션을 16개 사용 $\rightarrow 28 \times 28 \times 16$ 이 됨
- 192채널 대신에 16개 채널만 가지고 있습니다. 때때로 이것을 병목 레이어라고 부름
- 첫 번째 레이어 계산 비용 : 192번의 곱셈을 곱하면, 이것은 240만
- 두 번째 레이어 계산 비용 : $28 \times 32 \times 32$ 에 $5 \times 5 \times 16$ 차원 필터를 적용을 곱하면 10.0 million

\rightarrow 총 곱셈의 수는 131 1240 만 배수의 곱셈의 합계 \rightarrow 약 1 억 2 천만에서 약 10분의 1 배까지 천이백만 곱셈으로 줄어듦

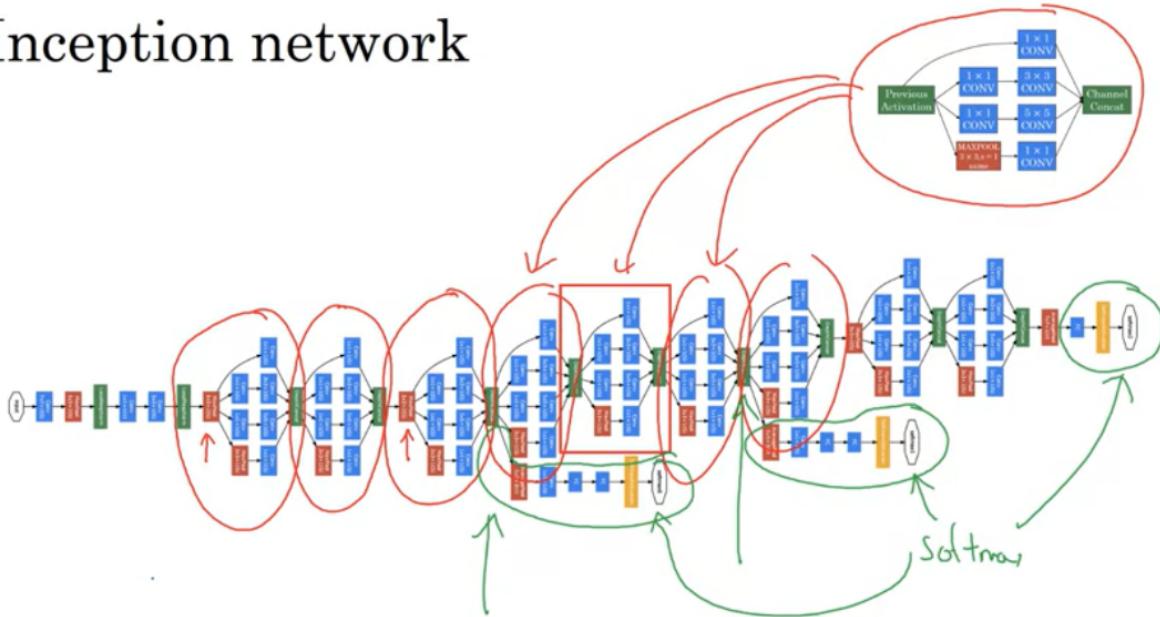
병목 레이어를 구현하면 representation의 크기를 상당히 줄일 수 있고 성능을 해치는 것처럼 보이지는 않지만 많은 계산량을 절약 할 수 있습니다. 따라서 이것이 인셉션 모듈의 핵심 아이디어입니다.

Inception module



- 이것은 하나의 인셉션 모듈입니다. 그리고 인셉션망이 하는 일은 이 모듈들을 함께 모으는 것입니다.

Inception network



[Szegedy et al., 2014, Going Deeper with Convolutions]

Andrew Ng

- 이 곁가지들은 숨겨진 레이어를 가져 와서 예측하는 데에 그 레이어를 사용하는 것입니다. 그래서 이것은 실제로 softmax 아웃풋이며, 이것 역시 그렇습니다.

인셉션망은 대체로 네트워크 전체에 걸쳐 여러 번 반복되는 인셉션모듈입니다.

오픈 소스 실행

Clone or Download → URL복사 → cmd에서 git clone 옆에 URL복붙 → 다운로드됨

⇒ 이러한 네트워크를 사용해서 전이 학습(transfer learning)을 할 수 있습니다.

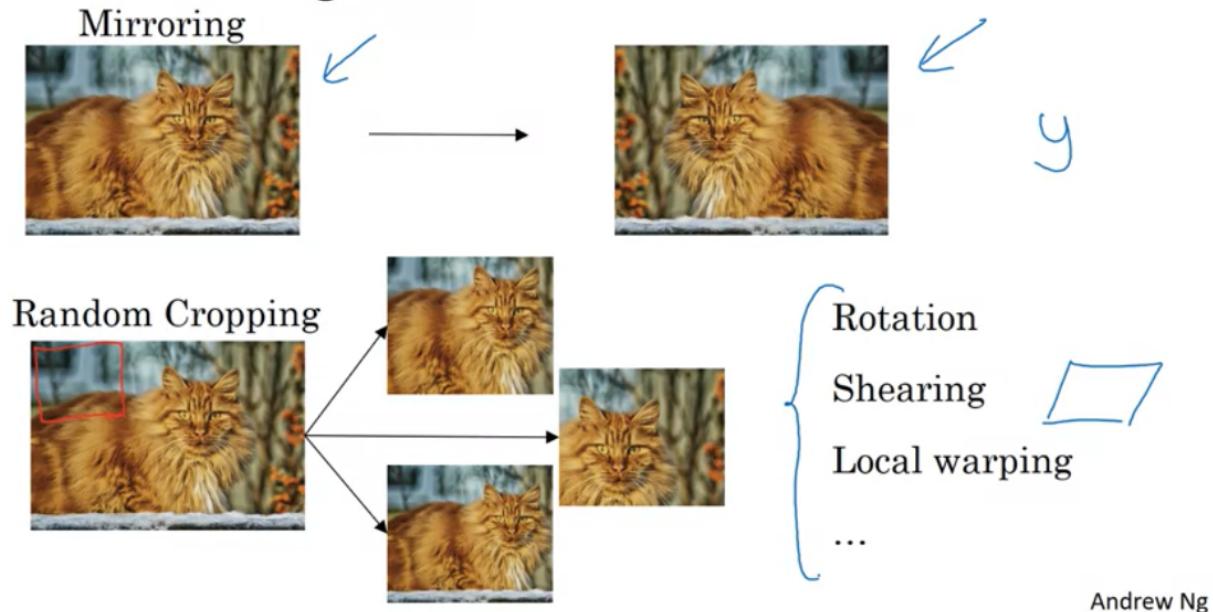
전이 학습

세 클래스에 대한 분류 문제가 있습니다.

- 다른 사람이 이미 트레이닝 시켜놓은 네트워크 아키텍처를 다운받아서, 사전 훈련으로 사용하고 그것을 여러분이 관심있는 새로운 작업으로 전환하면 훨씬 더 빠른 진전
- 코드뿐만 아니라 가중치도 다운로드하는 것이 좋습니다.
- 이미 트레이닝 된 다운로드 받을 수 있는 네트워크들이 많이 있습니다. 예를 들자면, 1000 개의 다른 클래스들이 있는 Init Net 데이터 세트죠. 그러면 그 네트워크는 가능한 1000개 중의 하나를 아웃풋하는 Softmax 유닛을 가지고 있을 수도 → **softmax 레이어를 없애고 Tigger 나 Misty를 출력하는 softmax 유닛을 생성!**
- 여러분이 더 큰 레이블 데이터 세트를 가지고 있고, 티거와 미스티, 그리고 둘 다 아님의 아주 많은 사진을 가지고 있다면, 여러분이 할 일은 더 적은 레이어를 고정하고, 나중에 훈련시키는 것입니다.
- **아웃풋 레이어에 다른 클래스가 있는 경우라면, 여러분의 티거, 미스티, 또는 둘 다 아님의 아웃풋 단위가 필요합니다.** 이 작업에는 몇 가지 방법이 있습니다. 마지막 몇 개의 레이어 방법을 사용하여 초기화(initialization)로 사용하고, 그곳에서 기울기 강하를 수행하거나 마지막 몇 개의 레이어를 날려 버리는겁니다. 그리고 자신만의 새로운 숨겨진 유닛과 자신의 최종 softmax 아웃풋을 사용할 수 있습니다.

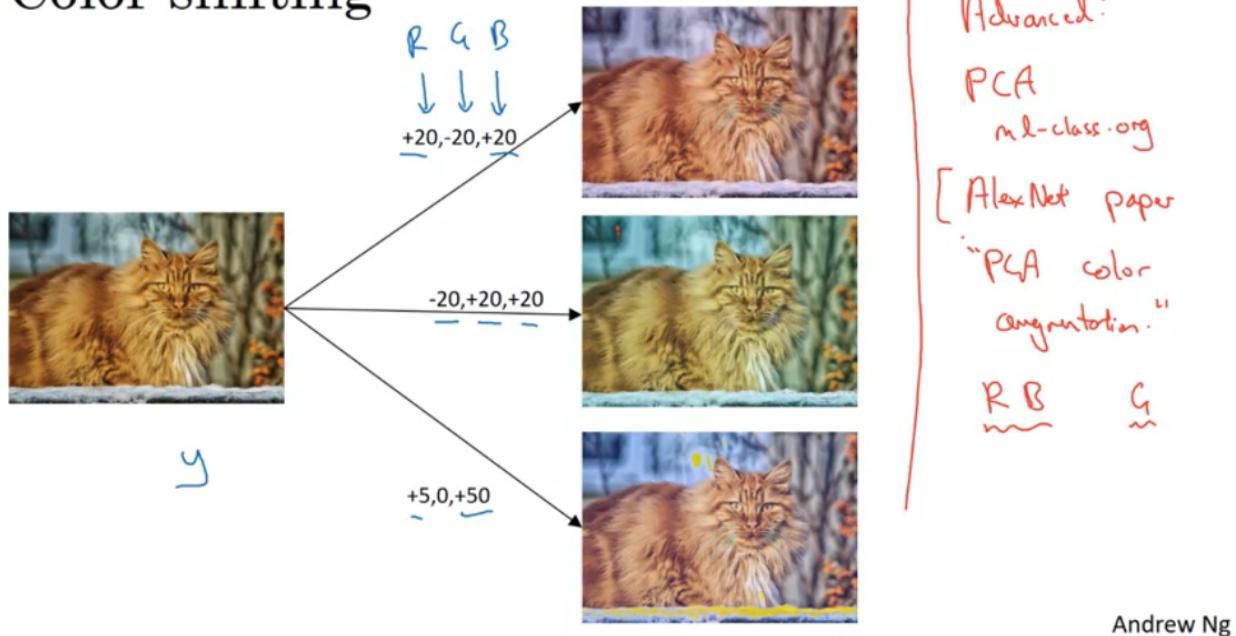
데이터 증식

Common augmentation method



- 미러링
- 무작위 크롭 crop
- 회전
- 이미지 깎기
- 색상 이동

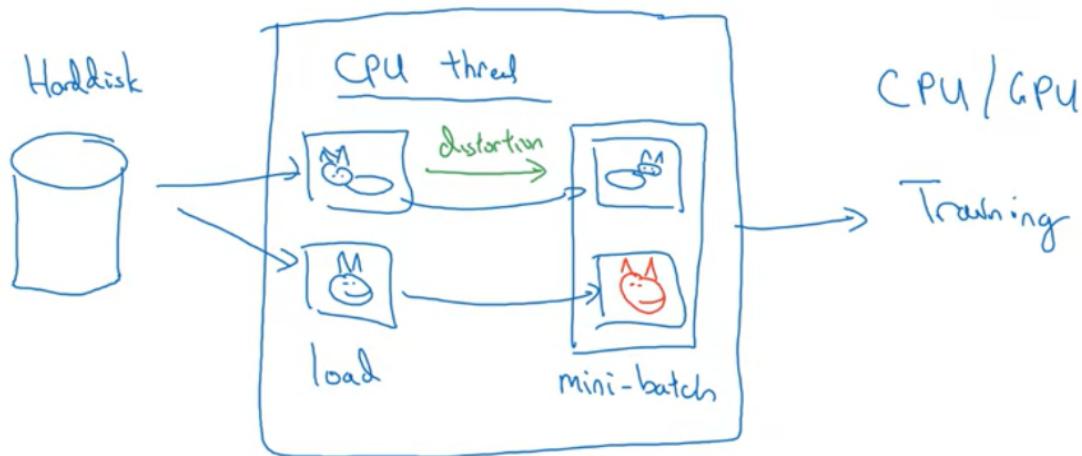
Color shifting



색상 왜곡을 구현하는 방법 중 하나는 PCA라는 알고리즘을 사용하는 것

PCA 색상 증강의 대략적인 아이디어는 이미지가 주로 자주색인 경우, 주로 빨간색과 파란색 색조가 있고 녹색은 거의 없습니다. 그러면 PCA 색상 증강으로 빨간색과 파란색으로 많이 추가 및 제거할 것입니다 이렇게 되면 모든 녹색과 균형을 이루기 때문에 색조의 전체 색상이 동일하게 유지됩니다.

Implementing distortions during training



Andrew Ng

CPU 스레드를 사용하여 왜곡을 구현

- 데이터를 로딩하고 왜곡을 실행하는 역할을 하는 하나의 스레드, 거의 네 개의 스레드를 가지는 것
- 그리고 나서, 다른 스레드나 다른 프로세스를 전달해서 트레이닝을 하는 것

컴퓨터 비전의 상태

- 더 많은 데이터가 없는 상황에서 좋은 성능을 얻는 방법은 아키텍팅에 더 많은 시간을 할애하고 네트워크 아키텍처와 더 많이 작업해보는 것
- 하나는 앙상블입니다. 이것이 의미하는 바는, 여러분이 원하는 신경망을 알아 낸 후에 여러 신경망을 독립적으로 훈련시키고 아웃풋을 평균화하는 것
- 작동하지 않을 가중치를 평균화하지 마십시오. 7 가지 다른 예측을 가진 7가지 신경망을 가지고 평균을 내십시오. 그리고 이것으로 1 % 또는 2 % 더 나아지게 할 것입니다.
- 벤치 마크에서 실제로 도움이 되는 논문에서 볼 수 있는 또 다른 사항은 테스트 시에 하는 다중 크롭(multi-crop)
- 앙상블의 가장 큰 문제점 중 하나는 이러한 모든 서로 다른 네트워크를 유지해야한다는 것입니다. 그래서 컴퓨터 메모리가 더 많이 필요합니다. 멀티 크롭의 경우, 최소한 하나의

네트워크만 유지하면됩니다. 따라서 이것이 많은 메모리를 낭비하지는 않지만, 실행 시간은 상당히 느려집니다.