C4W2L01 Why look at case studies?

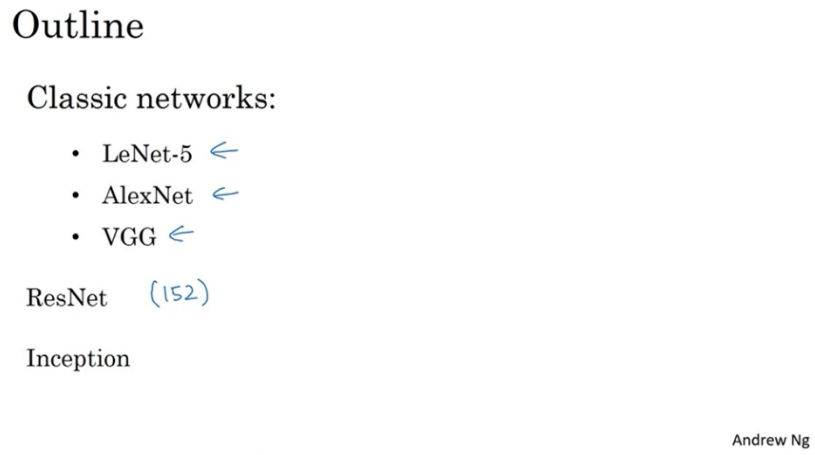
- 이번 시간에는 지난 주에 배운 기초 블록들로 만든 신경망 케이스 연구들을 살펴보자.

- 누군가 개발한 개, 고양이 분류 신경망을 자율 주행이나 다른 분야에도 활용 할 수도 있음.

- 다음 강의에서 볼 고전 신경망으로 LeNet-5, Alexnet, VGG 등

- 그 다음으로 resnet에 대해서 살펴볼 것임. resnet은 깊게 학습 가능함.

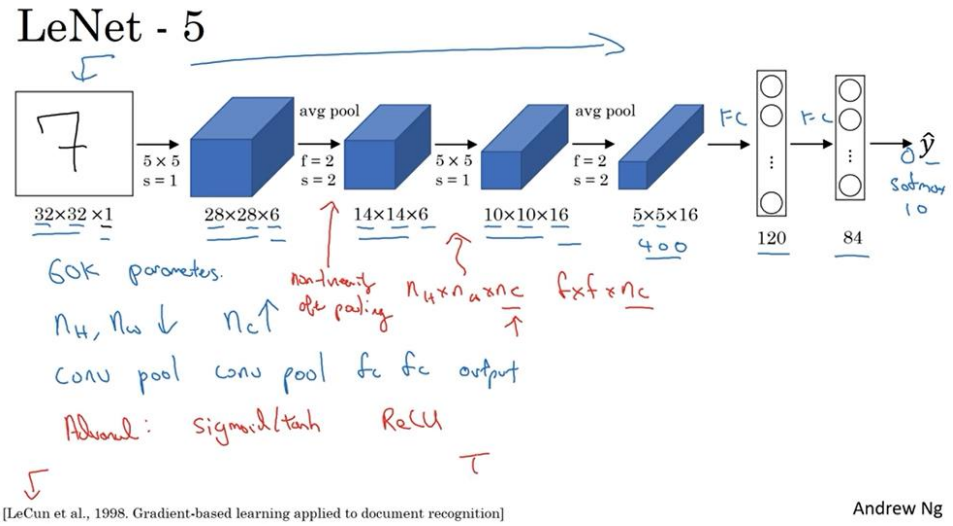
- 마지막 케이스 스터디로 inception에 대해서 배우자.



C4W2L02 Classic Network

LeNet-5

- LeNet-5는 32 x 32 x 1 크기의 흑백 손글씨 이미지를 인식하는 모델
- conv2d, 5 x 5, s=1, 필터 6개 사용, no padding => 28 x 28 x 6
- avg pooling, f = 2, s = 2 => 14 x 14 x 6
- conv2d, 5 x 5, s = 1, 16 filters, no padding => 10 x 10 x 16
- avg pooling, f=2, s=2 => 5 x 5 x 16 = 400
- flatten 후 fully connected layer, hidden unit 120
- fc2, 84 => y_hat, 10 categories, softmax
- 60k parameter로 작은편, 요즘은 10m, 100M등 수천배만음.
- N_h, N_w가 줄어들고, N_c가 늘어남.
- conv - pool -> conv - pool -> fc -> fc -> output
- tahn/sigmoid를 활성화 함수로 사용하고, 당시에는 relu는 잘 안씀 * 풀링 뒤에 비선형 활성화 함수 사용.



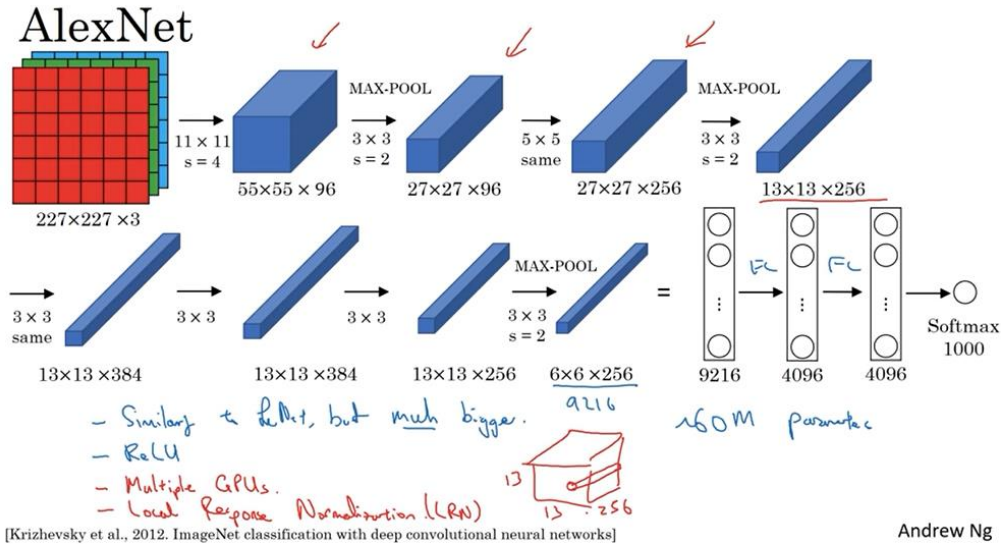
[LeCun et al., 1998. Gradient-based learning applied to document recognition]

Andrew Ng

AlexNet

- Alex krizhesky et al.
- 227 x 227 x 3 크기 이미지를 사용.
- 1. 11 x 11, s = 4, N_c = 96 => 55 x 55 x 96, max pooling 3 x 3, s= 2 => 27 x 27 x 96
- 2. 5 x 5, same, N_c = 256 => 27 x 27 x 256, max pooling 3 x 3, s = 2 => 13 x 13 x 256
- 3. conv + max_pooling => 6 x 6 x 256 = 9216
- 4. fc 4096 -> fc 1000 -> softmax, 1000
- LeNet-5와 유사하나 훨씬 큼. (LeNet - 60k 개의 파라미터였다면, AlexNet은 60M개의 파라미터를 가짐)
- LeNet-5와 달리 ReLU를 활성화 함수를 사용.
- 당시 GPU 성능이 나빠 두 개의 GPU에서 작업하도록 나눠서 사용.
- Local Response Normalization layer 도 있었으나 자주 사용되지 않음.

* 13 x 13 x 256 LRN은 h,w 한지점의 모든 채널을 보고 정규화 수행. 높은 활성 값을 가진 경우가 많아지는걸 방지하기 위함. 성능 상 유용하지는 않았음.

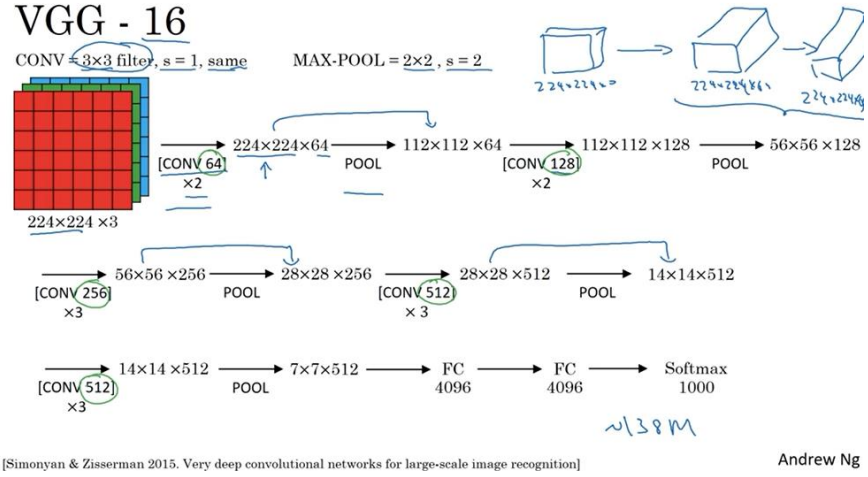


[Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks]

Andrew Ng

VGG - 16

- 단순한 신경망으로 간결함이 장점. conv = 3 x 3 filter, s = 1, same, max pool = 2 x 2, s = 2
- 1. 224 x 224 x 3 -> (conv 64) x 2-> 224 x 224 x 64 -> (max pooling) -> 112 x 112 x 64
- 2. 112 x 112 x 64 -> (conv 128) x 2 -> (max pooling) -> 56 x 56 x 128 => (여러번 반복)
- 3. 7 x 7 x 512 ->. (fc 4096) -> (fc 4096) -> softmax, 1000
- vgg-16인 이유는 16개의 레이어/층을 가지기 때문
- 138M 파라미터로 상당히 크나, 일관적임
- 다음 단계로 넘어갈 때마다 필터 개수가 규칙적으로 2배가 됨.
- 더 큰 버전으로 VGG - 19도 있음.

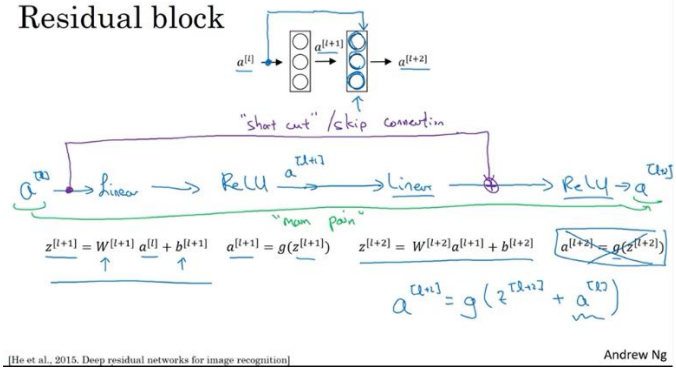


[Simonyan & Zisserman 2015. Very deep convolutional networks for large-scale image recognition] Andrew Ng

C4W2L03 Resnets

Residaual block

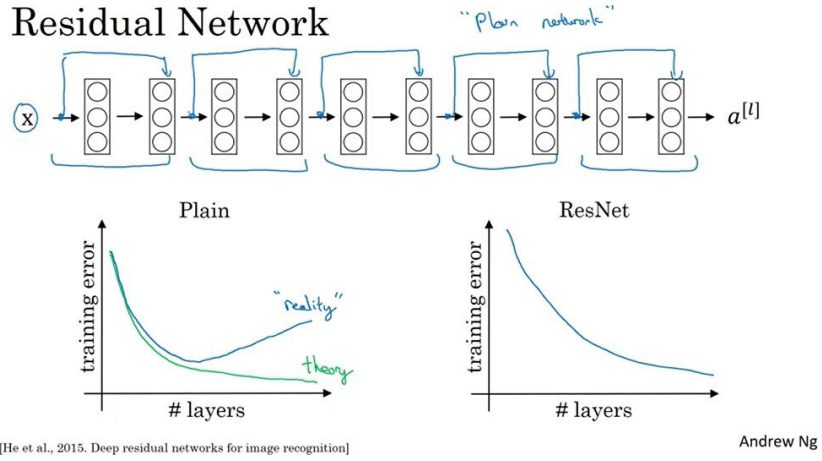
- 아주 깊은 신경망이 훈련하기 힘든 이유는 그라디언트 소실이나 폭증 때문임.
- 스킵 커낵션은 한 레이어의 활성 결과를 더 깊은 레이어로 전달함. -> 100층 넘게 훨씬 깊게 학습 가능해짐.
- main path : a_l -> linear operator -> relu -> a_l+1 -> linear op -> relu -> a_l+2
- resnet에서 main path를 수정하여 a_l를 a_l+1를 선형 연산 후 relu 적용전에 덧샘 연산함 => 이를 short cut이라 부름.
  * a_l의 정보는 더 깊이 전달된다. => a_l+2 = g(z_l+2 + a_l)
  * short cut을 skip connection이라고도 부름.
- residual block을 사용하면 훨씬 깊은 신경망을 학습할 수 있게 됨.



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

Residual Network

- resnet 논문에서 스킵 커넥션이 없는 신경망을 plain network라 함.
- 아래의 신경망은 5개의 residual block으로 이루어진 신경망.
- plain network는 layer가 깊어질수록 계속 에러가 내려가야 하나 실제로는 증가함.
  * 최적화 알고리즘이 학습하기 힘들기 때문.
- resnet의 경우 층이 늘어나도, 100층 이상이더라도 에러도 계속 줄어듦



C4W2L04 Why ResNets Work

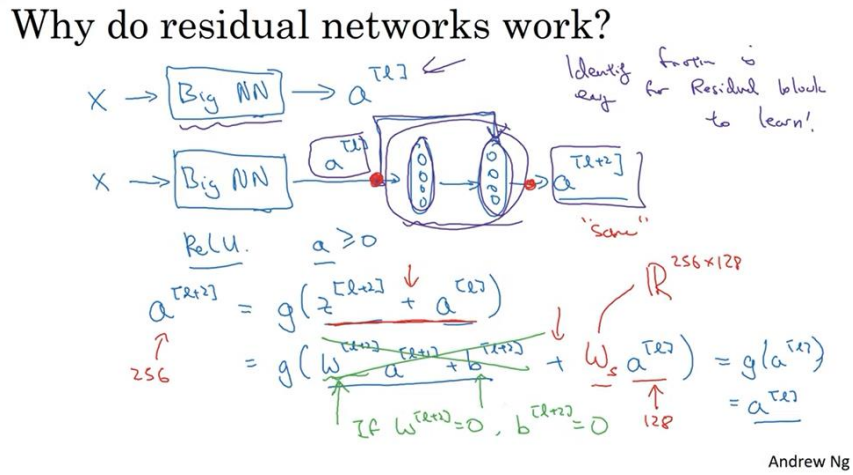
왜 잔차 신경망이 동작할까?

- 신경망이 깊어질 수록 훈련 셋을 잘 학습하기 힘들어 지므로 잘 깊게 만들지 않음. 하지만 resnet은 다름.
- a_l+2 = g(z_l+2 + a_l) = g(w_l+2 a_l+1 + b_l+1 + a_l)

* 여기서 a_l은 스킵 커넥션으로 전달받음.
* 여기서 w_l+2 = 0이면 a_l+2 = a_l가 되며, relu로 인해 양수값만 존재.

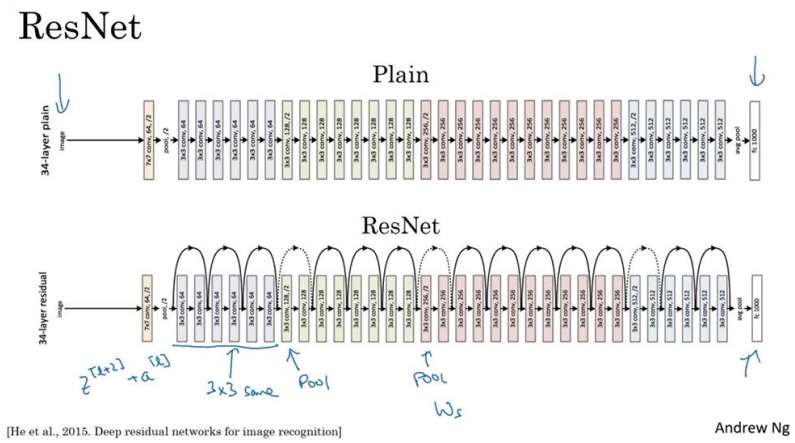
- 이와 같이 항등 함수로 학습이 되면, a_l+2 = a_l이므로 적어도 성능이 떨어질 일이 없으며 훈련이 더 용이해짐
- 이런 은닉 유닛을 많이 학습하면 성능 향상을 이룰 수 있음. 허나 스킵 커넥션이 없으면 항등 함수 물론 학습하기 힘듦.

* 항등 함수를 학습한다 = 스킵 커넥션은 적어도 성능 저하는 없다!



ResNet

- Plain Network에서 이미지를 입력으로받아 마지막에 softmax 출력을 받음.
- ResNet은 Plain에 skip connection이 추가됨.
- 이때 3 x 3 same conv 연산이 수행되어 공간적 크기가 줄어들지 않음. 대신 중간 중간에 풀링 수행



C4W2L05 Network In Network

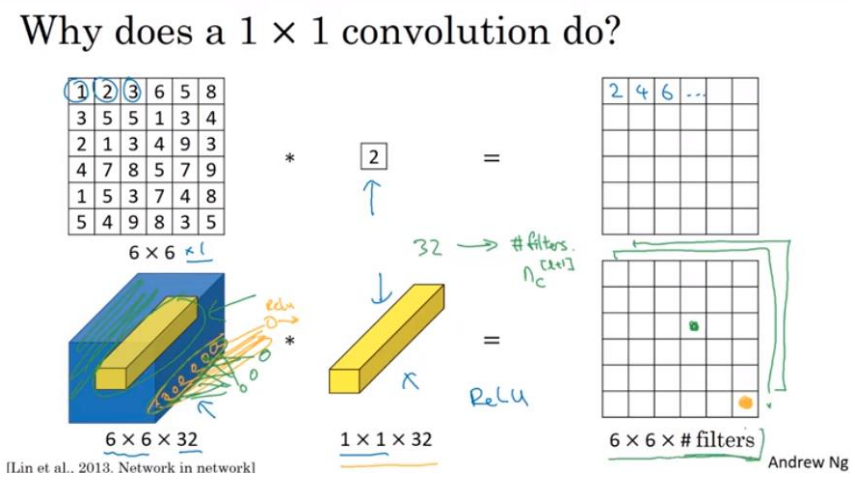
- 1 x 1 합성곱의 유용함을 알아보자

왜 1 x 1 합성곱 연산을 하는가?

- 입력 텐서의 공간적 모든 지점들을 각각 1 x 1 합성곱 연산하고, 비선형 함수를 적용하는데 필터의 개수만큼 출력한다

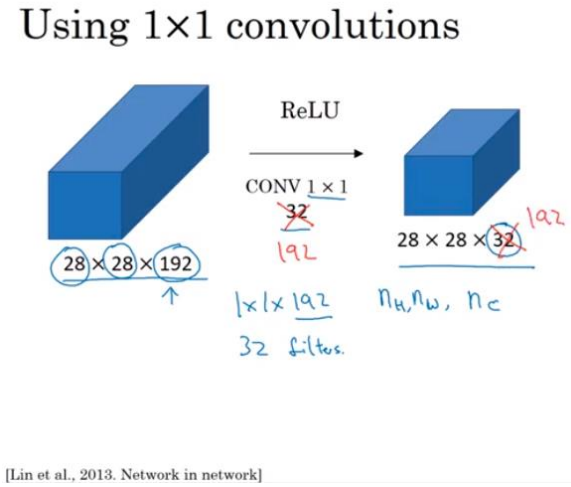
=> 6 x 6 x # filters

- 1 x 1 합성곱 연산을 네트워크 안의 네트워크라고도 부름.
- 많은 합성곱 신경망에 이용됨.



1 x 1 합성곱 연산의 예시

- 28 x 28 x 192의 입력 텐서가 들어올 때 공간적 크기를 줄이려면 풀링을 하면 됨
- 채널을 줄이려면 1 x 1 합성곱 연산을 통해 1 x 1 x 192 x #filter 크기의 텐서를 합성곱연산하여 28 x 28 x #filter로 차원을 줄임



C4W2L06 Inception Network Motivation

- 합성곱 신경망을 설계할 때 1 x 3, 3 x 3, 5 x 5 등 필터 크기를 지정해주어야 함.

=> 인셉션 신경망에서는 모든 필터 크기를 다 사용함. 복잡하지만 성능은 더 좋아짐.

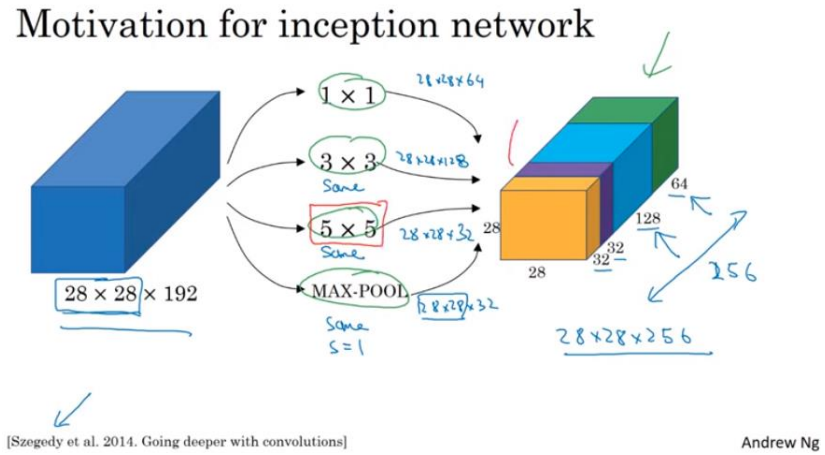
인셉션 신경망의 개요

- 필터 크기 지정없이 다양한 합성곱, 풀링 층을 사용함.
- 28 x 28 x 192 -> 1 x 1 x 192 x 64 => 28 x 28 x 64
- 28 x 28 x 192 -> 3 x 3 x 192 x 128 => 28 x 28 x 128
- 28 x 28 x 192 -> 5 x 5 x 192 x 32 => 28 x 28 x 32
- 28 x 28 x 192 -> Max Pool 28 x 28 x 32 => 28 x 28 x 32

* 다른 합성곱 연산 결과와 맞추기 위해서 최대 풀링의 경우 same, s=1 지정 필요

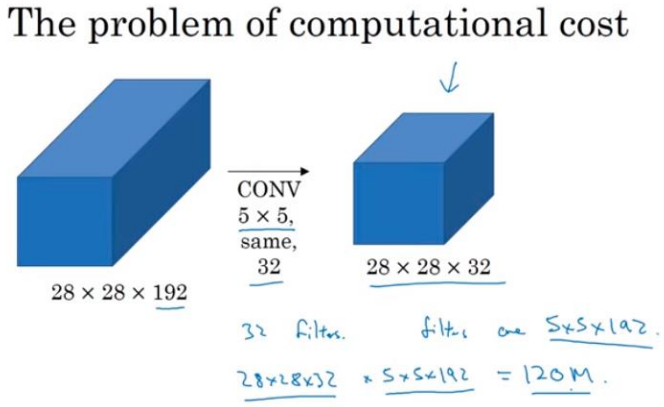
=> 아래의 인셉션 모듈의 출력은 28 x 28 x 256

- 필터나 풀링을 지정하기 보다 모두 사용해서 합한 것을 학습함.
- 인셉션 모듈의 문제로 계산 비용이 있음.



계산 비용 문제

- conv2d 5 x 5, same, 32 연산 시 출력은 28 x 28 x 32
- 28 x 28 x 32 (output) x 5 x 5 x 192 (각 출력을 계산하는데 필요한 연산수) = 120M
- 1 x 1 convolution으로 계산 비용을 크게 줄일 수 있음.



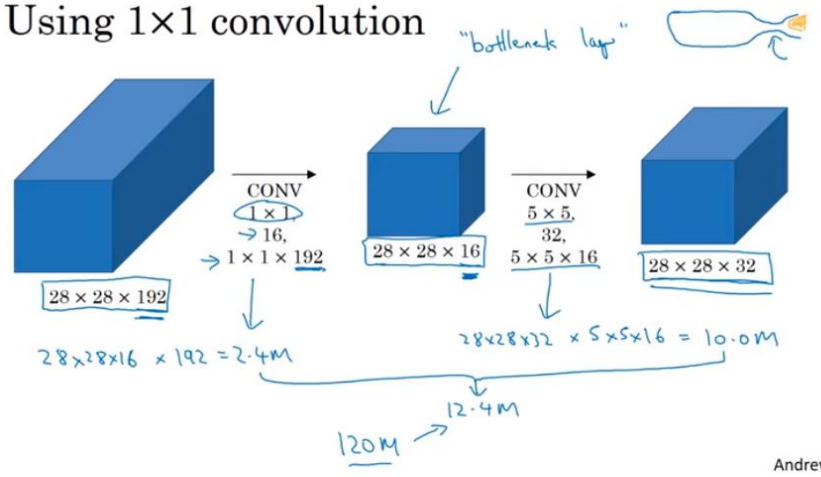
Andrew Ng

1 x 1 합성곱 연산 사용하기

- 28 x 28 x 192 입력에 1 x 1 x 192 x 16 연산 후, 5 x 5 x 16 x 32 합성곱 연산 수행
- 중간의 1 x 1 convolution으로 채널 수를 크게 줄임. 이를 bottleneck layer 병목층이라 부름.
- 28 x 28 x 16의 출력을 구하기 위한 계산량 : 28 x 28 x 1 x 1 x 16 x 192 = 2.4M
- 28 x 28 x 32 출력을 구하기 위한 계산량 : 28 x 28 x 32 x 5 x 5 x 16 = 10.0M
- 총 계산량 = 2.4M + 10.0M = 12.4M

* 1 x 1 합성곱 연산을 하지 않았을 때 120M 연산량보다 1/10으로 줄어듦.

- 표현 크기를 줄이면 성능 저하가 일어날 수 있으나 보틀넥 레이어를 잘 구현하면 계산량을 잘 줄이면서 성능에 지장 주지않을 수 있음.



Andrew Ng