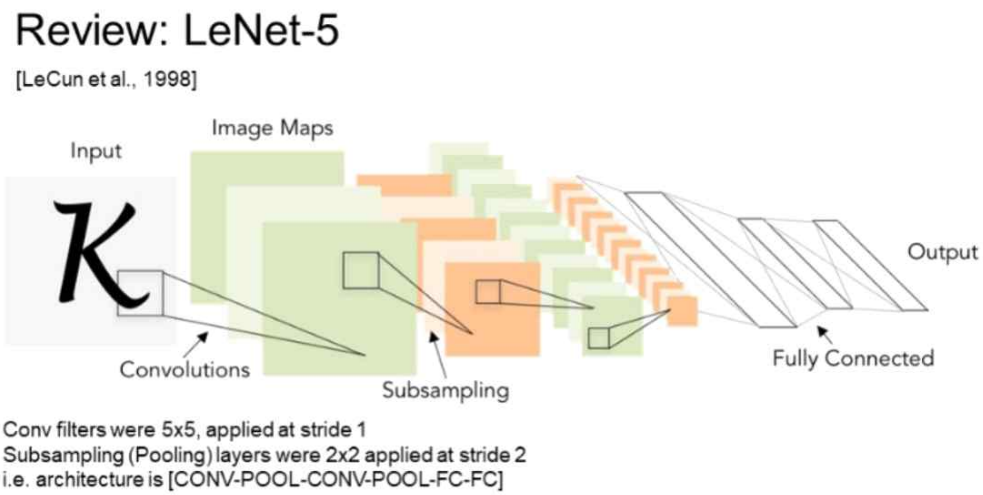
cs231n lecture 9 신지연

1. LeNet

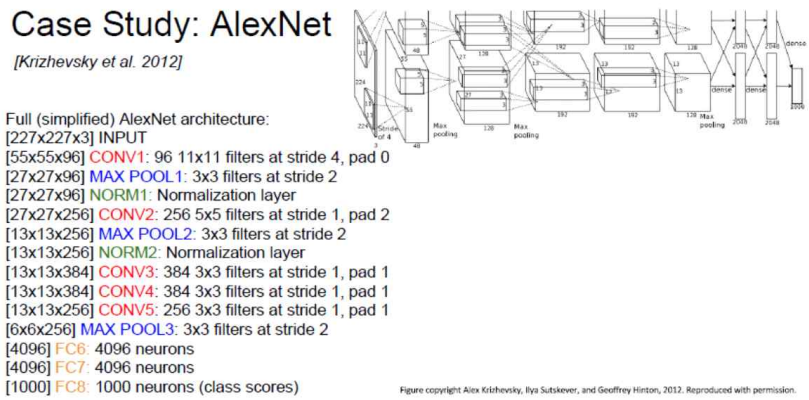
- 산업에 성공적으로 적용된 최초의 ConvNet

- 이미지를 입력으로 받아서 stride = 1인 5x5 필터를 거치고
  몇 개의 Conv Layer와 pooling layer를 거침

- Conv layer와 subsampling (average pooling layer) layer를 거쳐
  마지막에 1차원 벡터로 펴주는 FC layer가 있는 구조



2. AlexNet



- 2012년에 나온 모델, 최초의 Large scale CNN,
  ImageNet Classification Task를 잘 수행

- 기본 구조 :
[Conv - Max pooling-(normalization)] x 2 - CONV - CONV - CONV - Max pooling - [FC-FC-FC])

- 5개의 CONV layer와 3개의 FC layer 그리고 사이의 pooling layer로 LeNet과 크게 다르지 않음

- GPU를 두개로 나눠서 학습했다는 것이 큰 차이점

- Normalization 은 AlexNet 이후로 필요하지 않아서 사용X

1) Convolutional Layer

Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**

- input data = (227 * 227 * 3)

- filter = (11 * 11 * 3) & filter 96개

- Stride = 4

-> Output size = (N - F)/stride + 1 => (227 - 11)/4 + 1 = 55

-> Output size = (55 * 55 * 96) (당시 메모리 용량이 3G밖에 안돼서 filter를 48개씩 데이터를 반으로 나눠서 넣었기 때문에 (55*55*48) x 2개)

- 파라미터의 개수 : (업데이트해나갈 값, 가중치같은 것) (11*11*3) * 96

2) Maxpooling Layer

Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

- input data = 1st layer의 Conv layer ouput = (55*55*96)

- filter = 3*3*96

- stride = 2

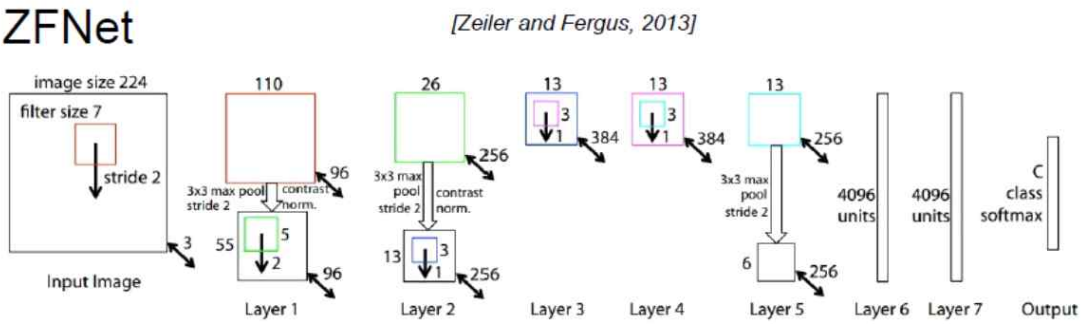
- Output size = (N - F)/stride + 1 => (55 - 3)/2 + 1 = 27

- Output size = (27 * 27 * 96)

- 파라미터 개수 : 0

  -> 파라미터는 우리가 학습시키는 가중치.
  Conv Layer에는 학습할 수 있는 가중치가 있지만
  pooling의 경우 가중치가 없고 특정 지역에서만 큰 값을 뽑아내서 학습 시킬 파라미터 x.

3. ZFNet



TODO: remake figure

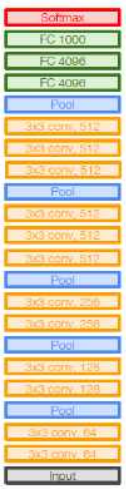
AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

- ZFNet은 AlexNet의 하이퍼파라미터를 개선한 모델
- AlexNet과 같은 layer 수이고 기본적인 구조도 같지만 stride size, filter 수와 같은 하이퍼파라미터를 조절하여 Error rate를 더 개선

4. VGGNet

INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters



VGG16

- 훨씬 더 네트워크가 깊어졌고, 더 작은 필터를 사용.
  항상 3 x 3 필터만 사용, (이웃하는 픽셀을 포함시킬 수 있는 최소 필터)
  작은 필터를 유지, 주기적으로 Pooling을 수행해 전체 네트워크를 구성

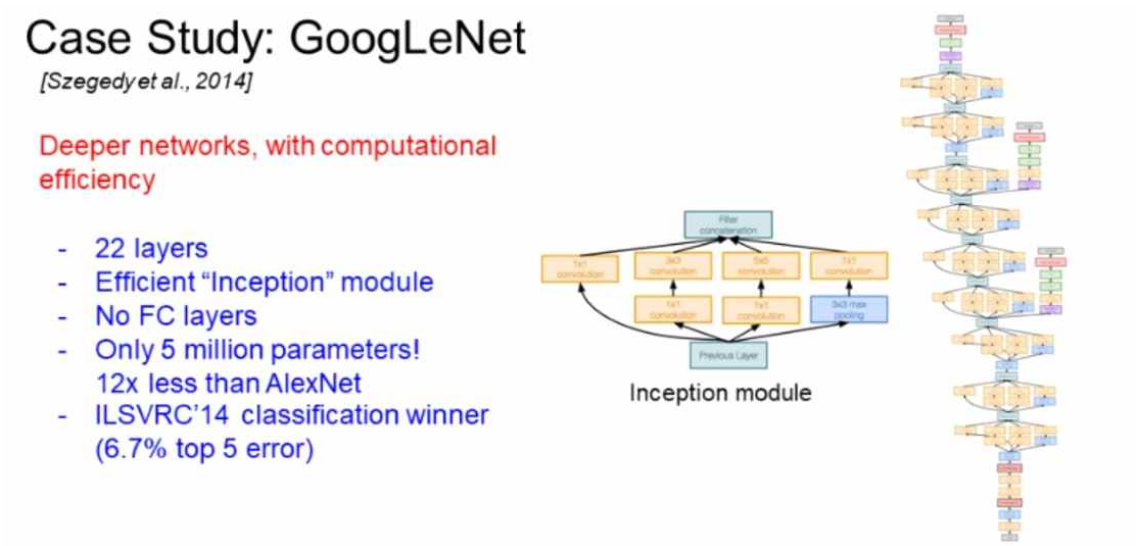
- AlexNet과 비슷한 절차로 Training을 거침

- Local Response Normalization (LRN)이 없음

- Ensemble을 사용하여 최고의 결과를 뽑아냄

- 맨 마지막에서 두 번째 FC (fc7)은 다른 task들을 가지고 잘 일반화를 함.

5. GoogLeNet



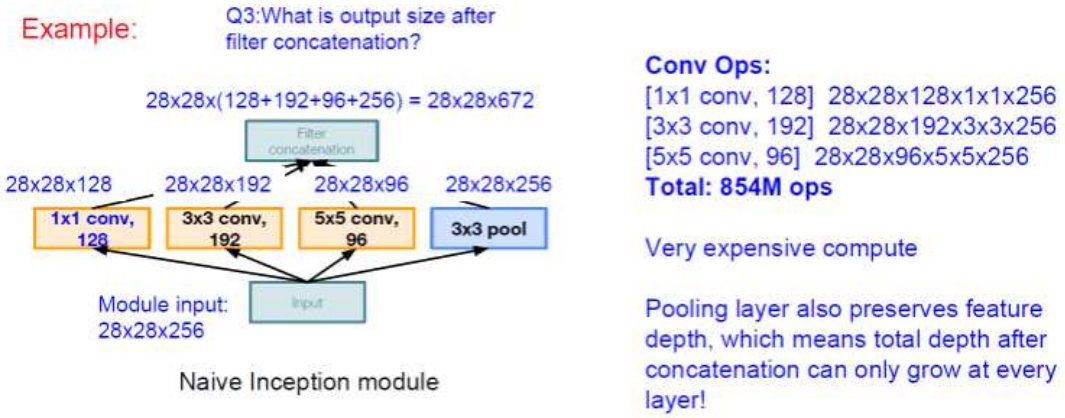
- 더 깊어진 layer (22 layer로 구성)

- Inception module을 사용

- FC layer가 사라짐
-> 파라미터 개수가 집중돼있는 layer를 없애 파라미터 개수를 줄임

- 파라미터가 5M개밖에 없음 (AlexNet은 6M개)

1. Inception module

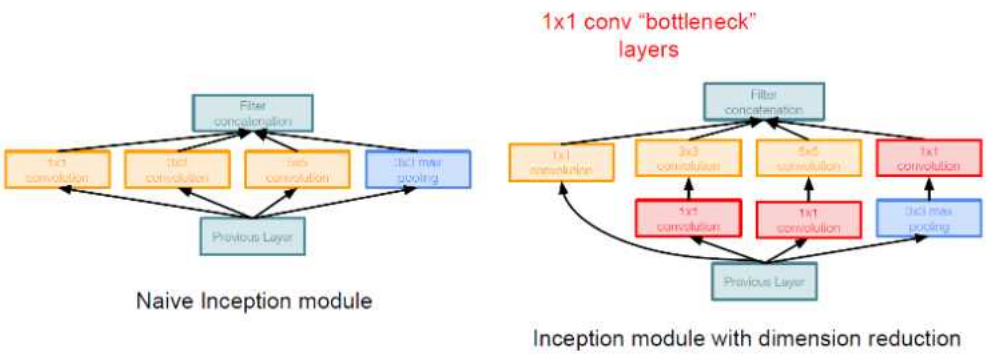


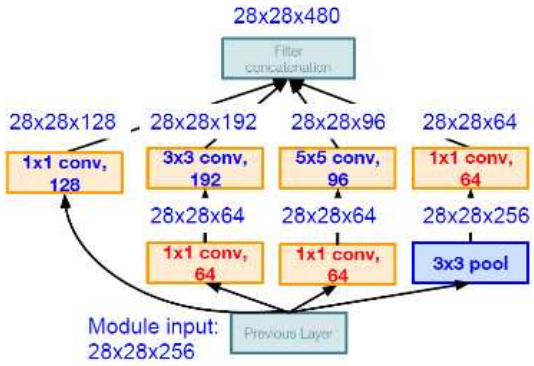
- 동일한 입력을 받는 서로 작은 다양한 필터들이 병렬적으로 존재

- 총 28x28x672개의 output data가 생성되고, 이를 위해 854M(8억 5400만)개의
  합성곱연산이 필요 → Inception module 안에서의 계산량이 많다.

- Pooling layer는 feature depth를 유지하기 때문에 concatenation은
  항상 증가할 수밖에 없음

- 계산량을 줄이기 위해서 1x1 convolution layer를 사용 -> Input의 depth가 줄어드는 효과, Bottleneck layer



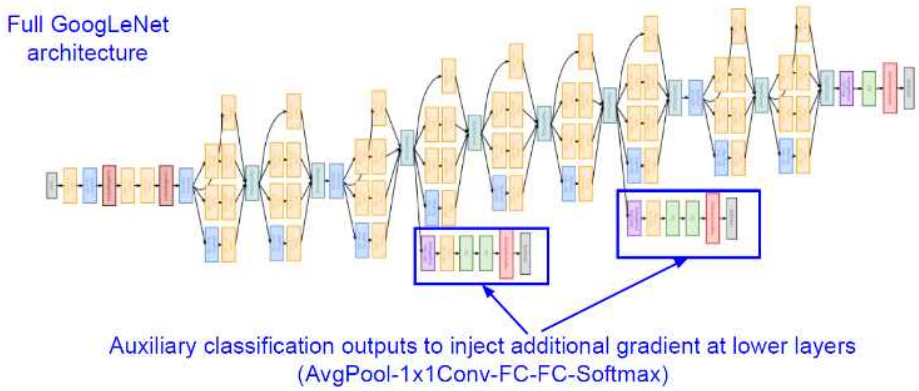


Inception module with dimension reduction

Conv Ops:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops

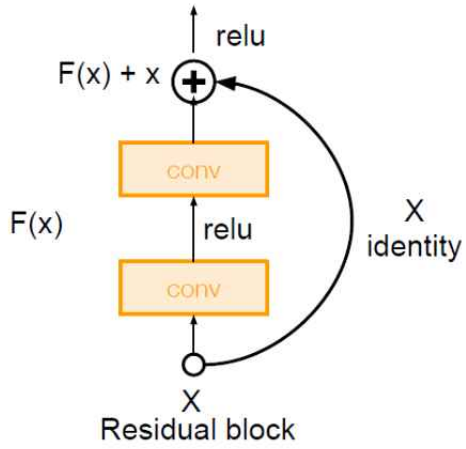
Compared to 854M ops for naive version
Bottleneck can also reduce depth after pooling layer

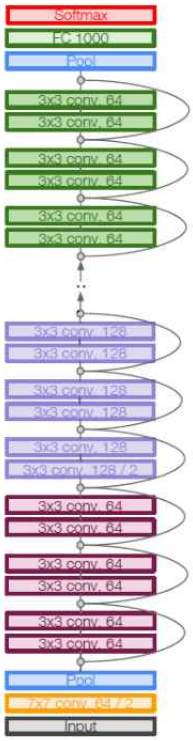
2. GoogLeNet architecture



- 보조 분류기O :
  네트워크의 깊이가 깊어서 중간 Layer의 학습을 돕기 위해서 설계

6. ResNet





- 152층으로 획기적으로 깊어졌음

- ResNet 이름은 Residual network

- layer가 깊어질 때 생기는 gradient vanishing 문제를
  residual block이라는 방식을 통해 해결하고자 한 데에서 착안한 이름

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

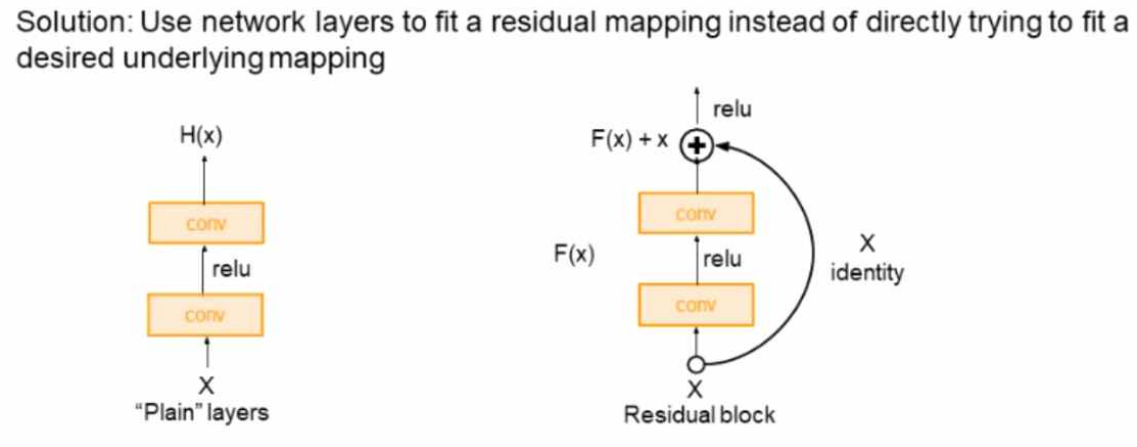
56-layer model performs worse on both training and test error
-> The deeper model performs worse, but it’s not caused by overfitting!

- 네트워크가 깊어지면 깊어질수록 풍부한 특징들을 추출할 수 있지만 무작정 깊은 층을 쌓으면 성능이 저하되는 문제

- 20층과 56층 CNN을 비교했을때, 56층이 train, test error 모두 높음

- overfitting 때문에 test error가 높은 것이라 생각할 수 있으나, train error도 높은 것으로 보아 overfitting의 문제만은 아니었다.

-> CNN을 깊게 쌓았을 때 성능이 저하되는 것이 Optimization의 문제라면, " 얕은 네트워크의 구조를 그대로 가져와서, 추가로 쌓는 층은 input을 output으로 그대로 내보내는 identity mapping을 한다면, 적어도 얕은 네트워크만큼의 성능은 나와야될 것이다" 라는 가정을 세움
여기서 ResNet의 가장 큰 특징인 Residual learning이 등장



- 일반적으로 Layer를 쌓아 올리는 방식 대신 Skip connection 구조 이용

- 가중치 Layer는 H(x)-x 에 대한 값이 0에 수렴하도록 학습을 진행

- VGGNet의 철학을 담고 있어 대부분의 convolution layer를 3x3으로 설계

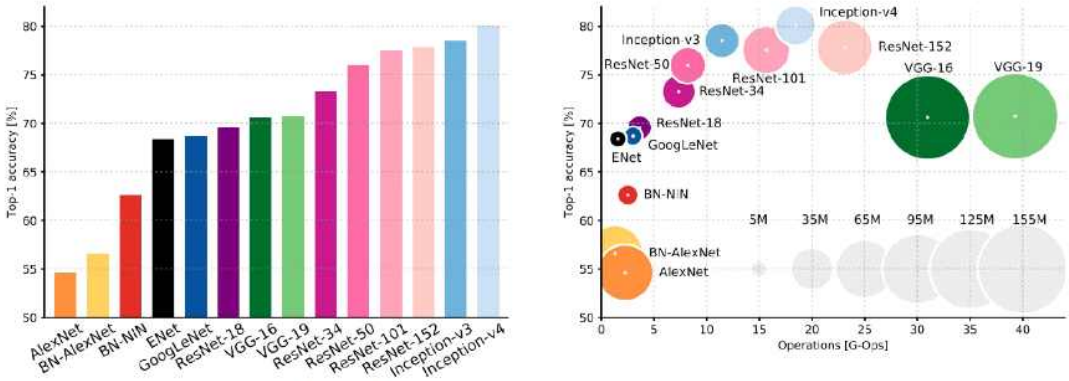
- 복잡도를 낮추기 위해 dropout, hidden fc를 사용하지 않음

- 출력 feature-map 크기가 같은 경우, 해당 모든 layer는
  모두 동일한 수의 filter를 갖음

- Feature-map의 크기를 줄일 때는 pooling을 사용하는 대신
  convolution을 수행할 때, stride의 크기를 "2"로 하는 방식

-네트워크를 깊게 설계할 때는 GooLeNet과 비슷하게 Bottleneck layer 추가

7. CNN Architecture 정리 그래프



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

- VGG Network가 가장 많은 메모리가 필요하고 Operation이 최대

- GooLeNet이 가장 효과적인 네트워크 구조

- AlexNet은 계산량은 적지만 정확도 많이 떨어짐

- ResNet은 깊이에 따라 정확도가 달라지고 가장 큰 정확도를 가진 네트워크