

# Week11: CNN Architectures

예습	미완료
복습	미완료
복습과제 날짜	
예습과제 날짜	
내용	cs231n Lecture09

개요

AlexNet

ZFNet

VGGNet

GoogLeNet

naive inception module

ResNet

## 개요

Case studies : Alexnet, VGG, GoogleLeNet, ResNet

Also NiN, WideResNet, ResNeXT, Stochastic Depth, DenseNet, FractalNet, SqueezeNet

## AlexNet

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0

[27x27x96] **MAX POOL1**: 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2

[13x13x256] **MAX POOL2**: 3x3 filters at stride 2

[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1

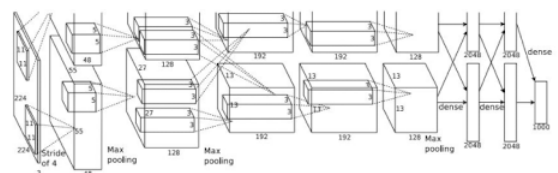
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

[6x6x256] **MAX POOL3**: 3x3 filters at stride 2

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)



2012년에 나온 모델, 최초의 large scale CNN

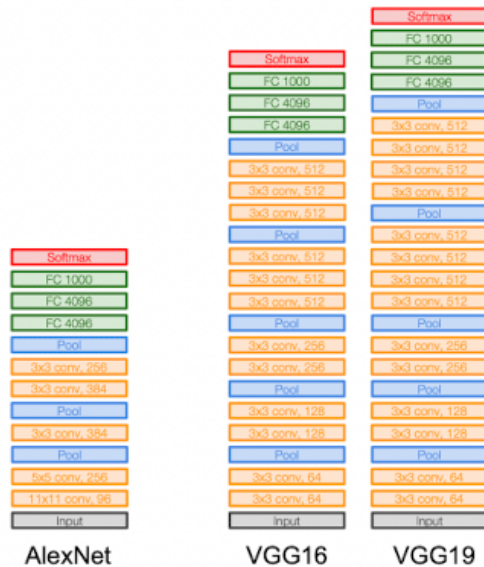
- conv- pool- normalizaion 구조가 2번 반복, 뒤에 conv layer, pooling, FC-layer가 붙는다.
- ImageNet을 학습 시키는 경우
  - input: 227x227x3
  - first layer:
    - 11x11 filter, 4 stride → 96
    - after 1st layer : 55x55x96 (  $(227-11)/4 + 1$  )
    - parameter : 11x11x3x96 : 35K
  - 2nd layer: pooling layer
    - 3x3 filter, 2 stride
    - after 2nd layer : 27x27x96 (depth doesn't chg)
    - parameter: X ⇒ 풀링 레이어에는 파라미터가 없음.
      - pooling은 가중치가 따로 없고, 특정 부분에서 값을 뽑는 역할이기 때문에 파라미터 존재하지 않음
      -

## ZFNet

- ZFNet은 AlexNet의 하이퍼파라미터 값을 개선한 모델.

## VGGNet

- 깊은 네트워크, 작은 필터(3x3)사용.
- 작은 필터를 유지하면서 주기적으로 pooling 해준다.
- 작은 필터를 사용하면서 파라미터의 수를 줄일 수 있고, 큰 필터에 비해 레이어를 좀 더 많이 쌓을 수 있다. (depth



를 늘릴 수 있다) AlexNet에서 8레이어였던 것이 16-19레이어가 되었다.

- 3x3필터를 3개 쌓은 것은 7x7 conv layer와 동일한 effective receptive filed를 가진다. 3x3 filter의 effective receptive filed는 15이다.
  - receptive filed : filter가 한 번에 볼 수 있는 입력의 'special area'. 필터가 서로 겹치기 때문에 7x7과 동일한 필드를 가진다.

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

Note:

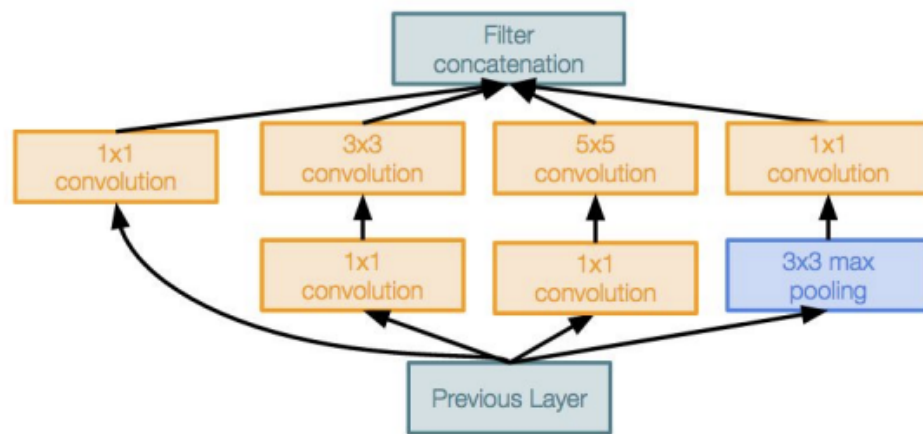
Most memory is in early CONV

Most params are in late FC

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

## GoogLeNet

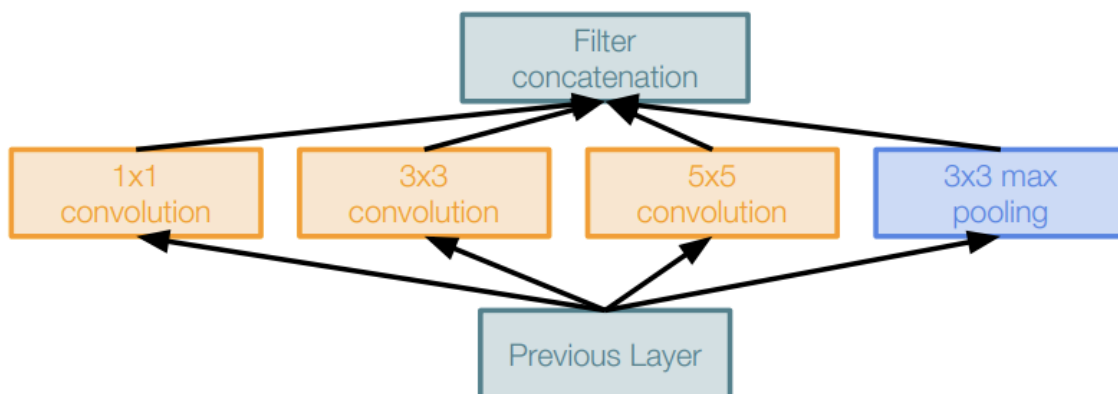
- 22개의 레이어를 가지고 있다.
- inception module 사용. 모듈을 여러 개 쌓아 모델을 만든다.



Inception module

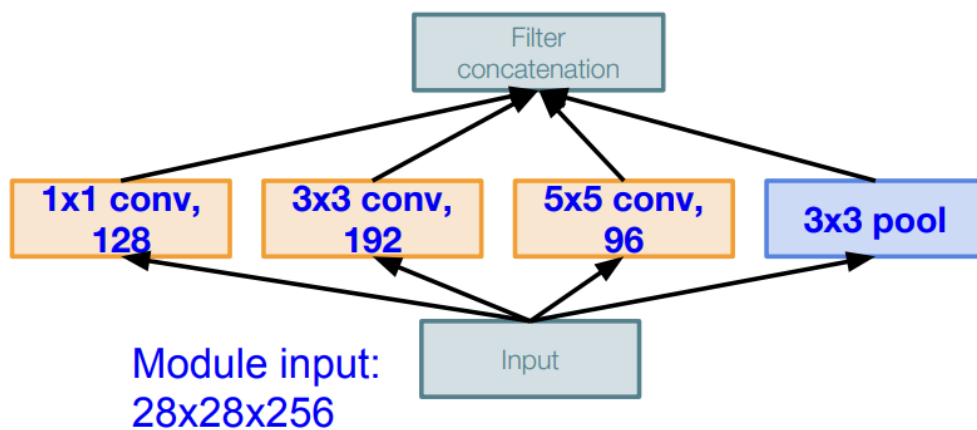
- FC layer이 없는데, 이것은 파라미터를 줄이기 위함.

## naive inception module



Naive Inception module

- 내부에 동일한 입력을 받는 필터가 병렬로 존재, 이 레이어의 압력을 받아 conv 연산 수행
- 각 레이어에 각각의 출력값이 나오면, 이 출력 값을 모두 depth 방향으로.
- 이렇게 연산 수행 후 하나로 합치는 방식은 depth가 매우 불어나고, 연산량이 늘어나게 된다.
- 예를 들어,



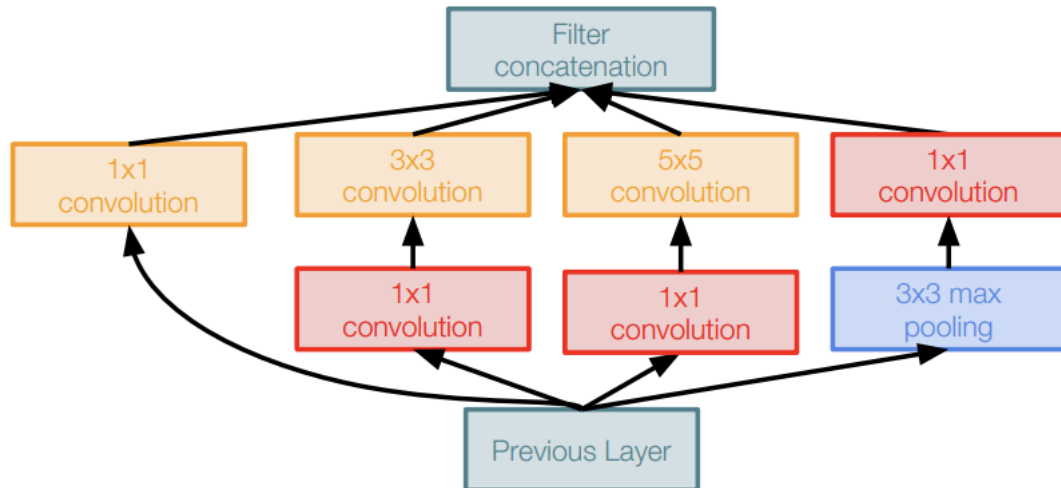
### Naive Inception module

이 경우 1x1x128 conv 출력이 28x28x128이 된다. 3x3 conv는 28x28x192, 5x5 conv는 28x28x96이 된다. pooling layer에서는 입출력의 크기가 같다.

최종적으로 28x28x672가 된다.

문제를 해결하기 위해 사용한 것이 'bottleneck layer' : conv연산 전 input의 차원을 낮춘다.

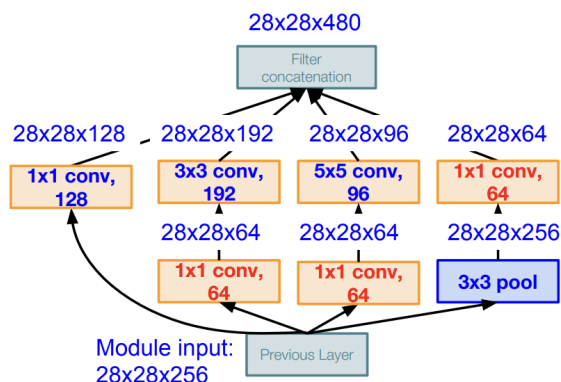
## 1x1 conv “bottleneck” layers



## Inception module with dimension reduction

### Case Study: GoogLeNet

[Szegedy et al., 2014]



Inception module with dimension reduction

Using same parallel layers as naive example, and adding “1x1 conv, 64 filter” bottlenecks:

#### 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**

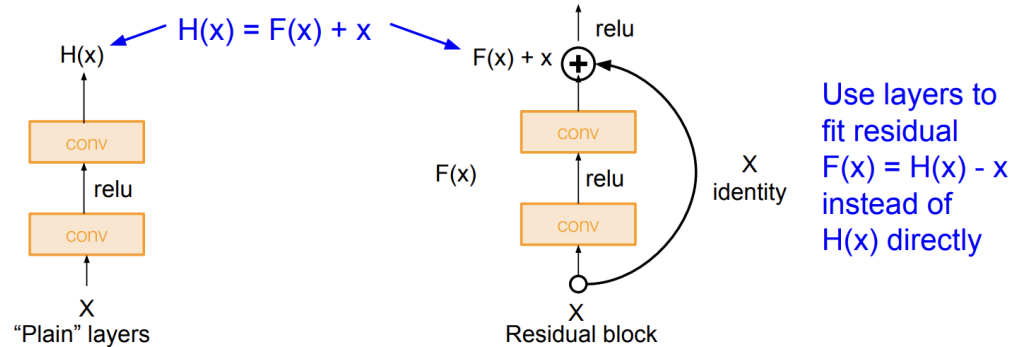
Compared to 854M ops for naive version  
Bottleneck can also reduce depth after pooling layer

## ResNet

- residual connections method

- 네트워크가 깊어질 수록 좋아지는 것이 아니다. optimization에 문제가 나올 수 있다. (가설)

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



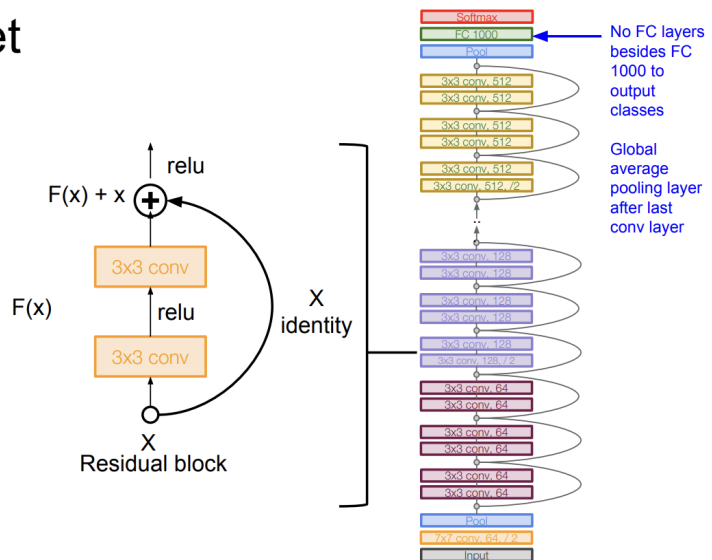
- 레이어가 직접  $H(x)$ 를 학습 하는 것이 아니라  $H(x) - x$  를 학습 할 수 있도록 한다. 고리 형태로 skip connection을 사용.
- 최종 출력값은 input+ residual이다.

## Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)



- 전체 ResNet 아키텍처이다. 하나의 residual block 은 2 3x3 conv layers로 이루어져 있고, 이 블록을 매우 많이 쌓아 올린다. ResNetdms 150레이어까지 쌓아올릴 수 있다.
- Depth가 50이상일 경우, 구글넷처럼 bottleneck layer을 사용한다. 1x1 conv를 도입해 필터의 depth를 줄여준다.
- 실제로 ResNet은 모든 Conv layer에 batch norm을 사용한다. 초기화는 Xavier