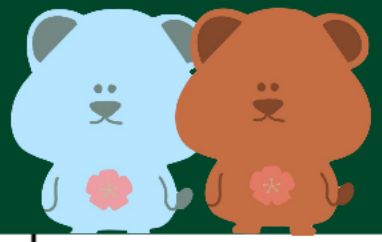


CNN Architectures

Week9_발표자: 하수민, 안서연

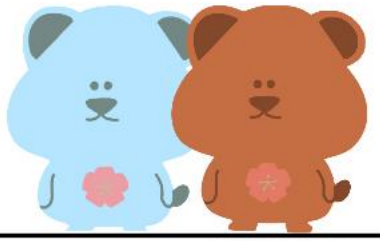
EWHA EURON

목차

01 Review

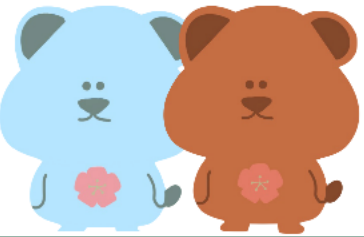
02 Case Studies

03 Other Architectures



EWHA EURON

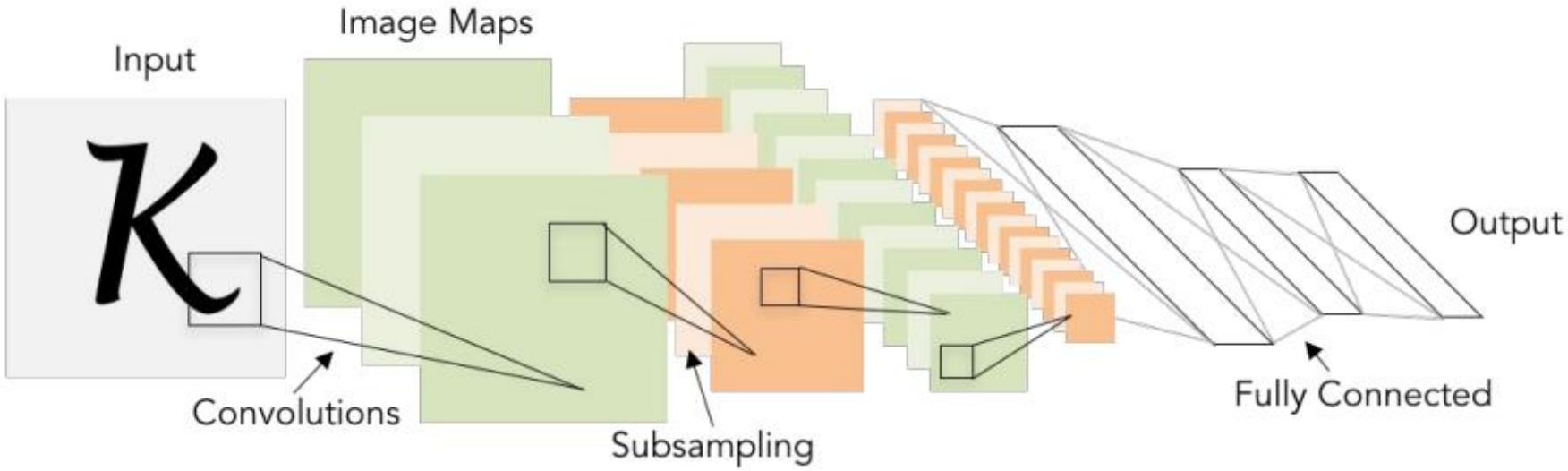
Review



EWHA EURON

1. Review: LeNet-5

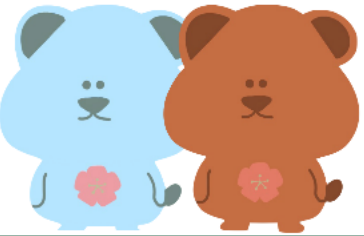
[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

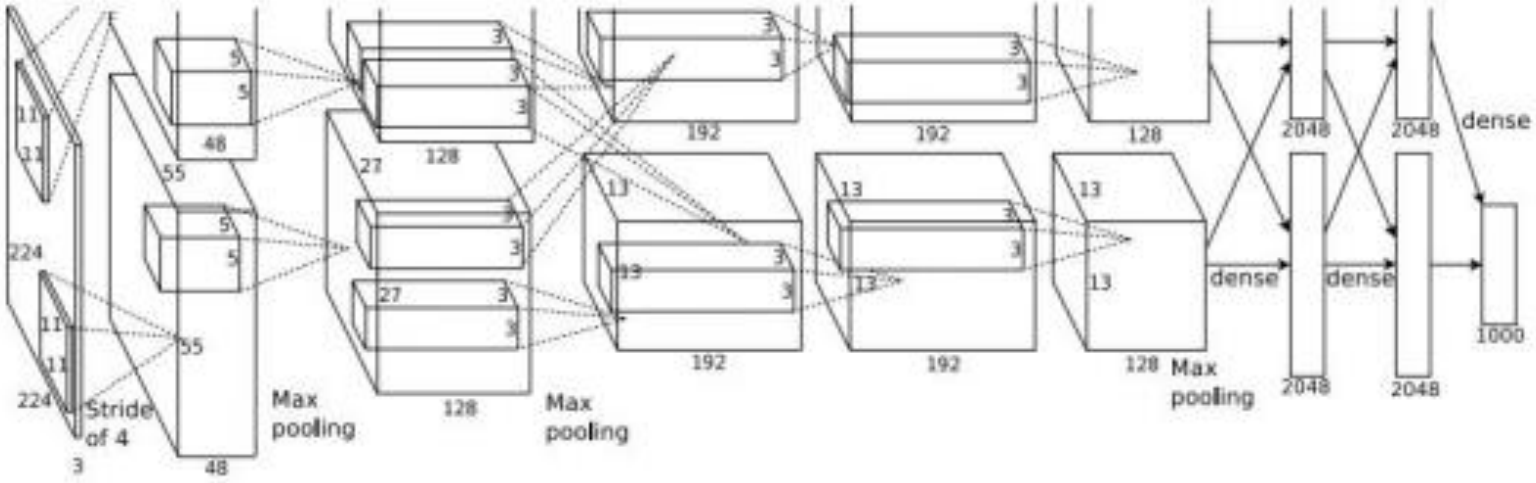
- 처음으로 CNN이란 개념을 개발한 네트워크
- Conv layer와 Subsampling layer를 거쳐 마지막에 1차원 벡터로 펴주는 Fully Connected layer가 있는 구조

Case Studies



EWHA EURON

2. Case Studies: AlexNet



Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

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

5개의 Conv layer와 3개의 FC layer, 사이에 pooling layer가 있는 구조로 LeNet과 비슷
요즘엔 Normalization 쓰지 않음.

2. Case Studies: AlexNet

Layer의 output volume/파라미터 개수 변화

1) Conv layer

Input: 227x227x3 images
Filter = 11x11, 96개
Stride = 4

출력차원 = (전체 이미지 크기 – 필터 크기) / stride + 1
= (227 – 11) / 4 + 1
= 55
➞ output volume(출력 사이즈): [55x55x96]

파라미터 개수 = (11 x 11 x 3) x 96 = 35K

2. Case Studies: AlexNet

Layer의 output volume/파라미터 개수 변화

2) Pooling layer

Input: Conv layer output (55x55x96)
Filter = 3x3, 96개
Stride = 2

출력차원 = (전체 이미지 크기 – 필터 크기) / stride + 1
= (55 – 3) / 2 + 1
= 27
➞ output volume(출력 사이즈): [27x27x96]

파라미터 개수 = 0
➞ Pooling layer는 위치한 필터 안의 데이터 값 중 큰 값을 뽑아내는 등 데이터 사이즈만 변환시키기 때문에 가중치가 없다. (파라미터는 우리가 학습시키는 가중치)

2. Case Studies: AlexNet

AlexNet 특징

1. 처음으로 비선형함수로 ReLU 사용
2. Normalization layer 사용 (이제는 잘 사용하지 않음)
3. Data augmentation: flipping, jittering, clipping, color normalization
4. Dropout: 0.5
5. Batch size: 128
6. Optimization: SGD Momentum 사용
7. Learning rate 1e-2에서 시작해 val accuracy가 더 이상 올라가지 않는 지점에서부터 학습이 종료되는 지점까지 1e-10으로 내림
8. Weight decay: L2
9. 7-CNN ensemble 사용 (에러 18.2% -> 15.4%)

2. Case Studies: AlexNet

전체 layer 특징

AlexNet은 모델이 두 개로 나누어져 서로 교차
(당시 GPU 메모리가 3GB 뿐이라 전체 레이어를 하나의 GPU에 넣을 수 없어 네트워크를 분산시켜 넣었다.)

각 GPU가 모델의 뉴런과 feature map을 절반씩 나누어 가진다.
➞ 첫 번째 레이어의 출력이 [55x55x96]이지만 각 GPU에서의 depth는 48

2. Case Studies: AlexNet

전체 layer 특징

AlexNet은 모델이 두 개로 나누어져 서로 교차
(당시 GPU 메모리가 3GB 뿐이라 전체 레이어를 하나의 GPU에 넣을 수 없어 네트워크를 분산시켜 넣었다.)

각 GPU가 모델의 뉴런과 feature map을 절반씩 나누어 가진다.
➞ 첫 번째 레이어의 출력이 [55x55x96]이지만 각 GPU에서의 depth는 48

1. Conv1, Conv2, Conv4, Conv5
: 같은 GPU 내에 있는 48개의 feature maps만 보고 학습한다.

2. Conv3, FC6, FC7, FC8
: GPU간 통신을 하기 때문에 이전 layer의 전체 depth를 가져올 수 있다.
➞ 이전 layer에서 들어오는 모든 input의 feature maps를 참고할 수 있다.

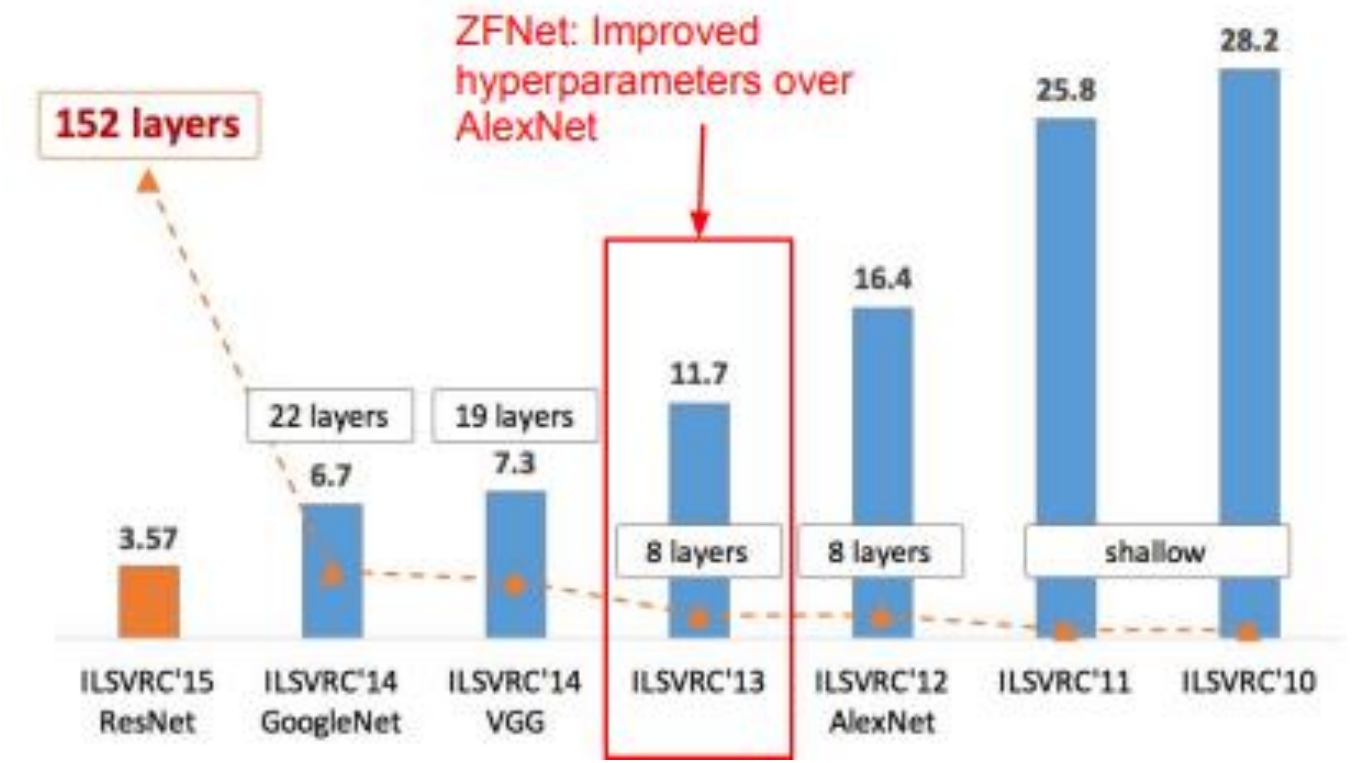
2. Case Studies: AlexNet

ImageNet Large Scale Visual Recognition Challenge(ILSVRC)

2012년에 ImageNet에서 우승
최초의 CNN 기반 우승 모델
➞ 이후로 CNN이 발전하면서 transfer learning의 기반 모델로 많이 사용

2. Case Studies: ZFNet

ImageNet Large Scale Visual Recognition Challenge(ILSVRC)

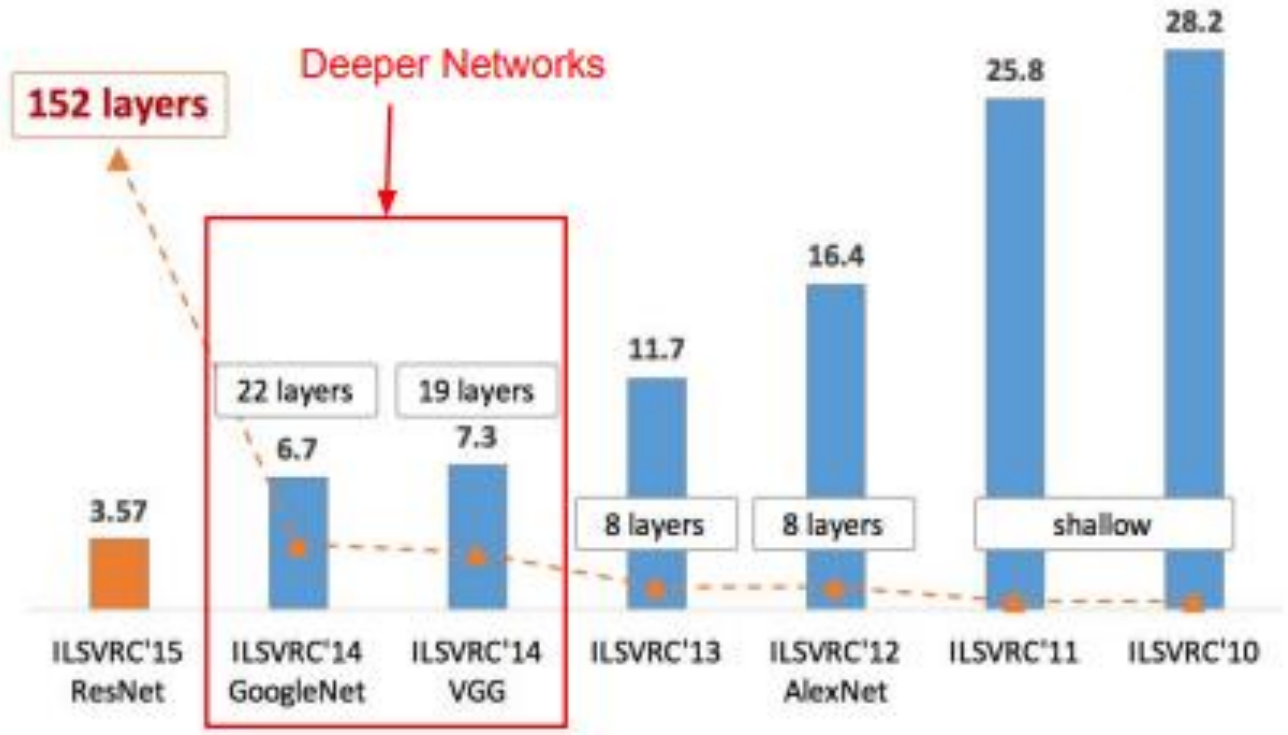


2013년에 ImageNet에서 우승

AlexNet에서 하이퍼파라미터 값만 변경

2. Case Studies: VGGNet

ImageNet Large Scale Visual Recognition Challenge(ILSVRC)



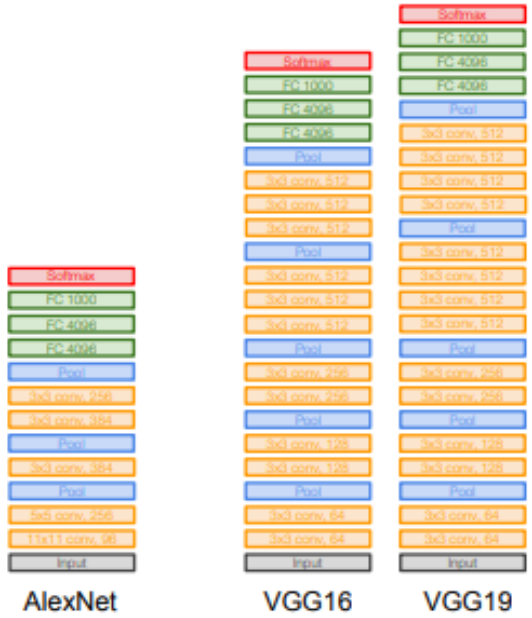
2014년에 많은 변화: “훨씬 깊어진 네트워크“, 성능 향상

1등: Google의 GoogleNet, 2등: Oxford의 VGGNet

2. Case Studies: VGGNet

작은 필터, 깊어진 네트워크

Layer: AlexNet 8개 -> VGGNet 16-19개
3x3 Conv(stride=1, pad=1), 2x2 max pool(stride=2)만 사용



🤔 왜 작은 필터를 사용했을까?
큰 필터를 사용한 것과 성능은 비슷하나 파라미터 수가 작아지기 때문!
레이어를 더 많이 쌓을 수 있다(Depth를 키울 수 있다).

➡ 3x3 필터를 3개 쌓은 것은 하나의 7x7 필터를 사용하는 것과 같은 receptive field
receptive field: filter가 한번에 볼 수 있는 입력의 영역

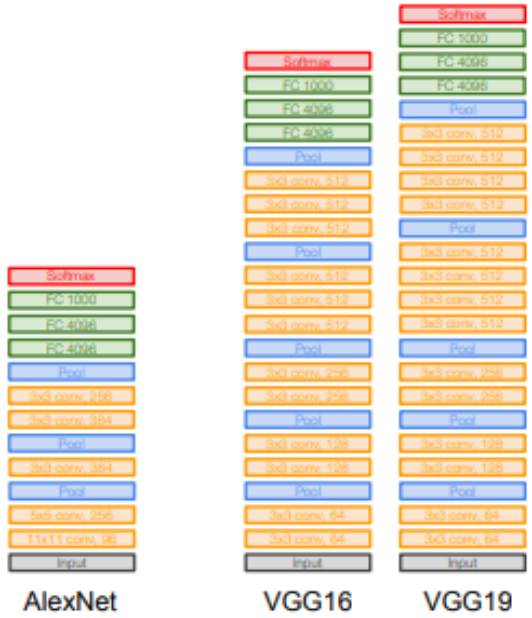
2. Case Studies: VGGNet

VGGNet의 총 파라미터 개수

- 파라미터의 총 개수는 138M(1억3800만)으로 많다. *(AlexNet은 6M개)*
- FC layer에도 1억 개가 넘는 파라미터가 있다.

VGGNet의 메모리 용량

- 메모리 용량이 크다.
- Forward pass만 계산해도 용량이 커서 무겁다. 96MB/image



메모리는 대부분 앞쪽 Conv layer에, 파라미터는 대부분 뒤쪽 FC layer에 집중되어 있다.

➡ 최근에는 특정 층을 버리는 방법을 사용하기도 한다.

2. Case Studies: VGGNet

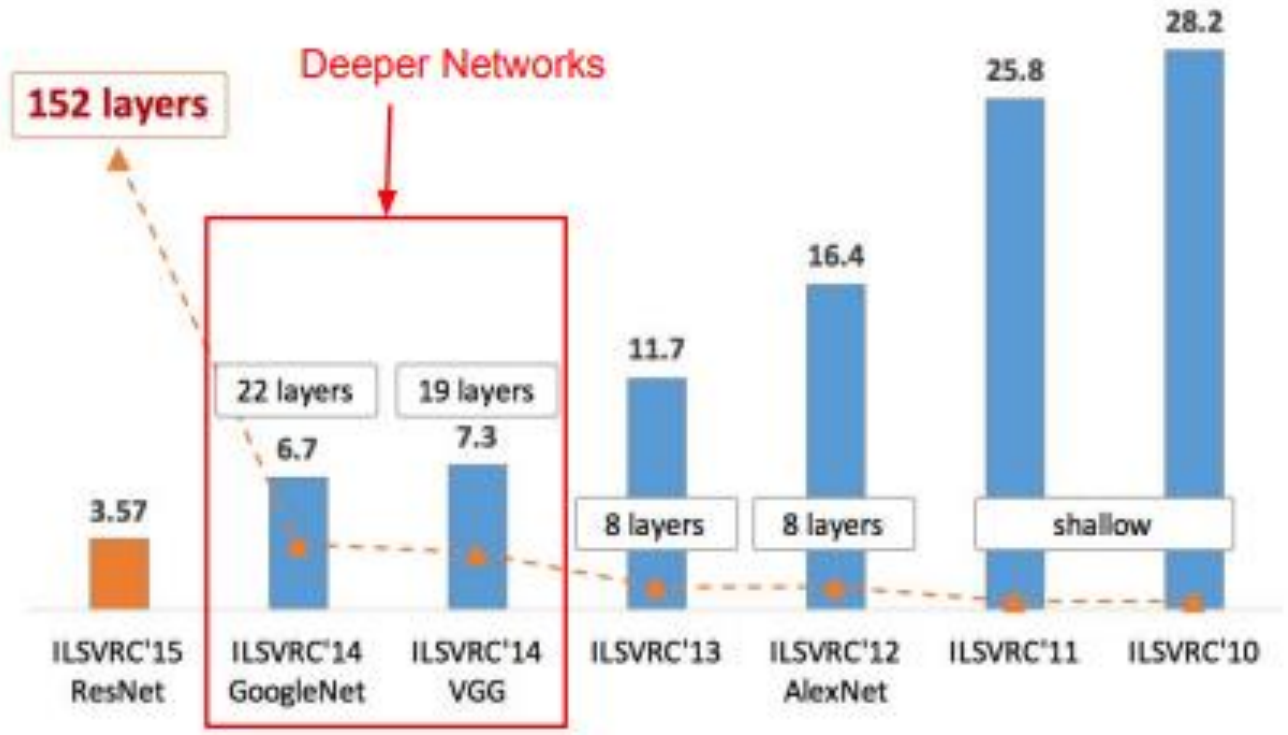
VGGNet의 특징

- ILSVRC' 14에서 classification에서는 2등, localization에서는 1등
- 학습 과정은 AlexNet과 유사하나 LRN(Local Response Normalization)은 사용 X
- VGG16보다 VGG19가 성능이 약간 더 좋으나 더 많은 메모리를 소모
- AlexNet처럼 모델 성능을 위해 ensembles 기법 사용
- FC7 layer(마지막 FC layer)는 아주 좋은 feature representation을 가진다.

  ➞ 다른 데이터에서도 특징 추출이 잘 되고, 다른 task에서도 일반화 능력이 뛰어나다.

2. Case Studies: GoogleNet

ImageNet Large Scale Visual Recognition Challenge(ILSVRC)



2014년에 많은 변화: “훨씬 깊어진 네트워크“, 성능 향상

1등: Google의 GoogleNet, 2등: Oxford의 VGGNet

2. Case Studies: GoogleNet

계산 효율성이 좋은, 깊은 네트워크

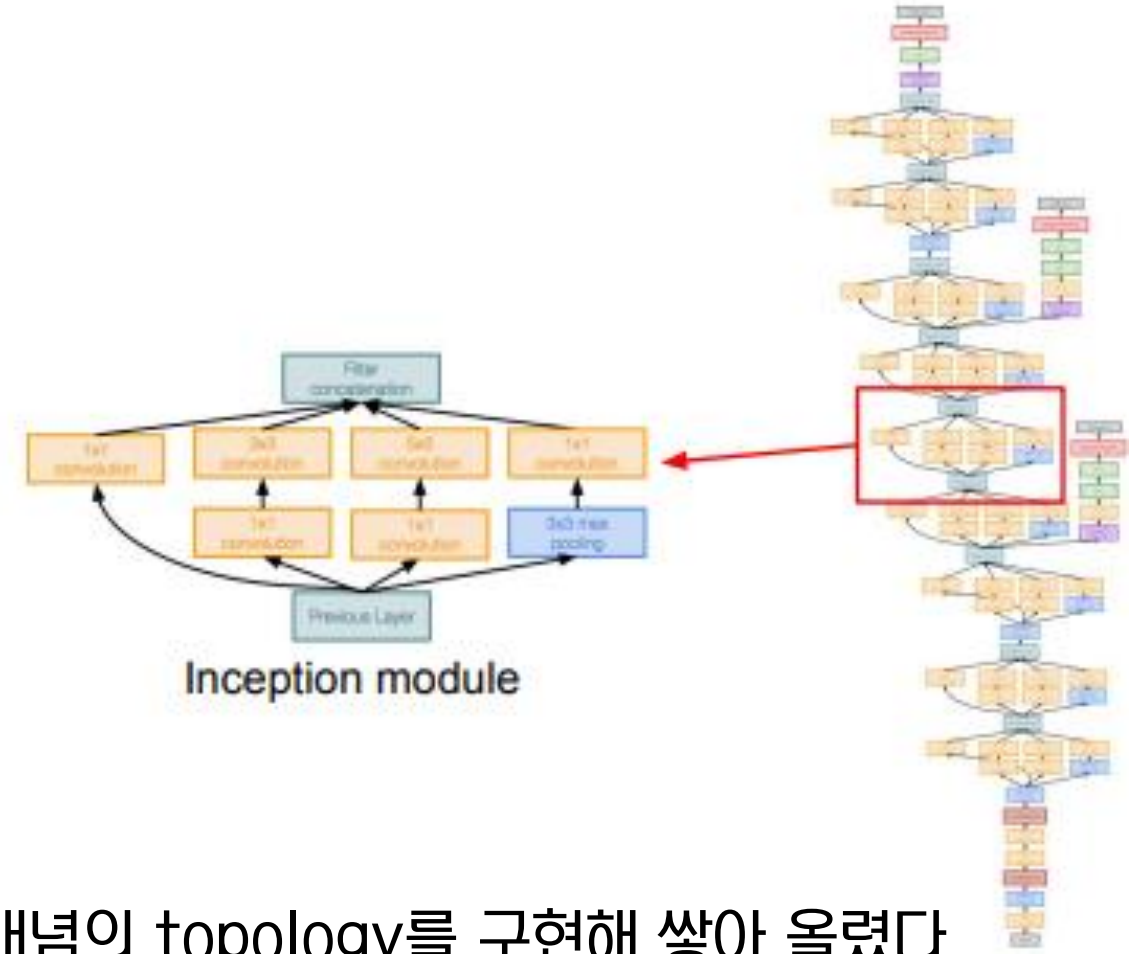
Layer 22개 (AlexNet 8개, VGGNet 16-19개)

파라미터가 고작 5M개! (AlexNet 6M, VGGNet 138M)

➡ 이는 파라미터 개수가 집중되어 있는 FC layer를 없앴기 때문!

2. Case Studies: GoogleNet

효율적인 “Inception” 모듈을 사용!

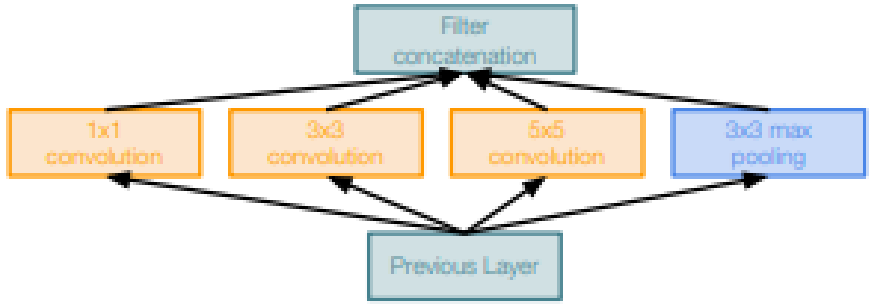


🤔 Inception module이란?

네트워크와 네트워크 사이에 있는 네트워크라는 개념의 topology를 구현해 쌓아 올렸다.

이 local network를 inception module이라고 한다.

2. Case Studies: GoogleNet



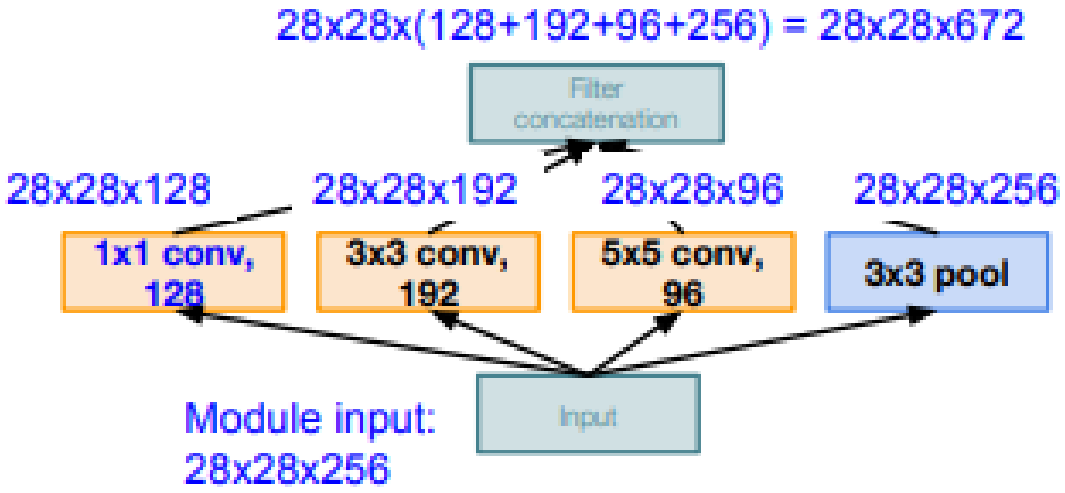
Naive Inception module

원래 하나의 layer에는 하나의 filter를 적용했다.

반면, GoogleNet은 여러 종류의 filter를 병렬적으로 input layer에 적용했다.
이렇게 나온 모든 filter의 output을 depth 방향으로 concatenate
→ 층층이 쌓아 올려 1개의 tensor output을 생성

하지만 이 방식엔 문제점이 있다! → 계산 비용

2. Case Studies: GoogleNet



총 28x28x672개의 output data

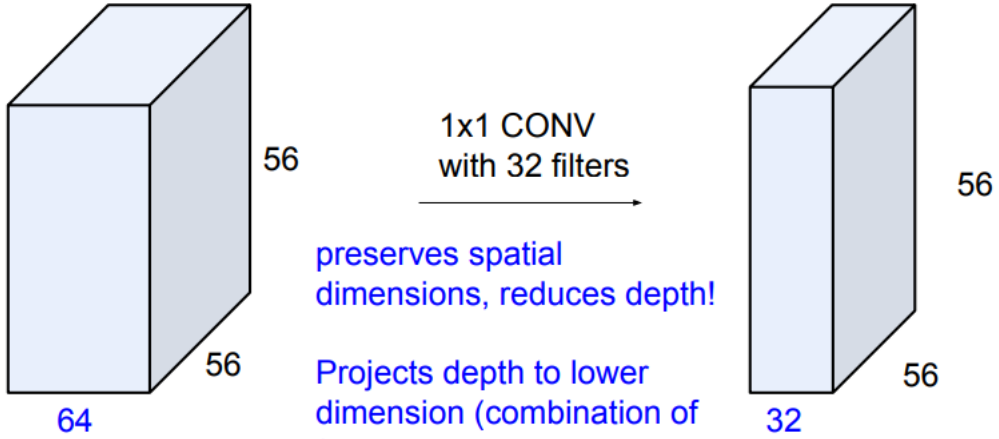
이를 위해 854M개의 합성곱 연산

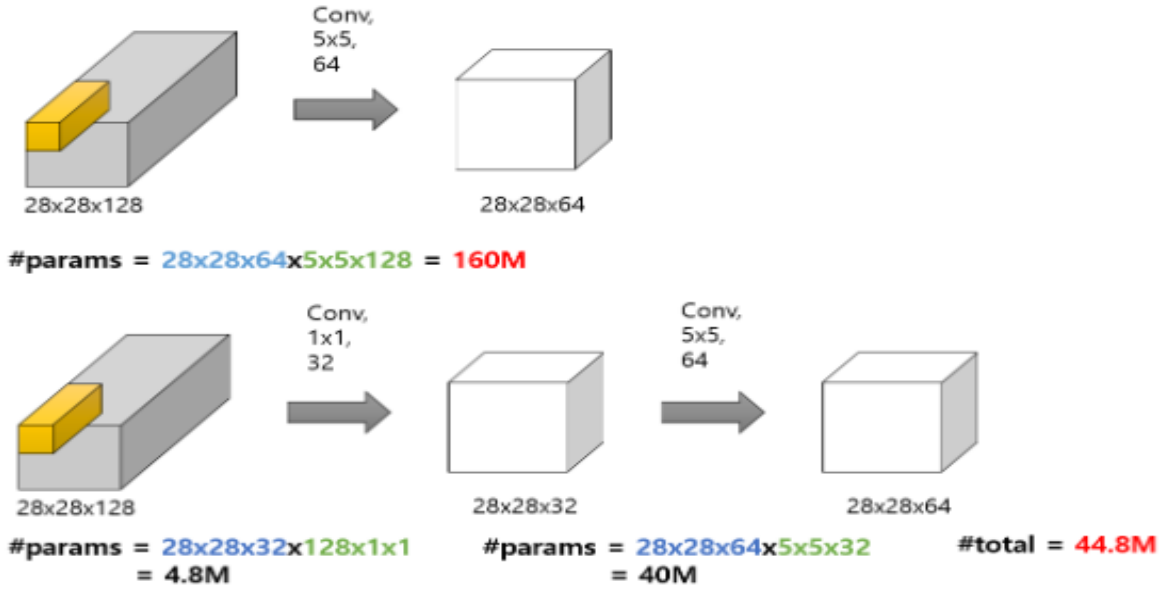
➞ Inception module 안에서의 계산량이 많다.

Pooling layer는 feature depth를 유지하기 때문에 concatenation은 항상 증가!

계산량을 줄이고 depth를 줄이기 위해 bottle neck layer를 추가하자!

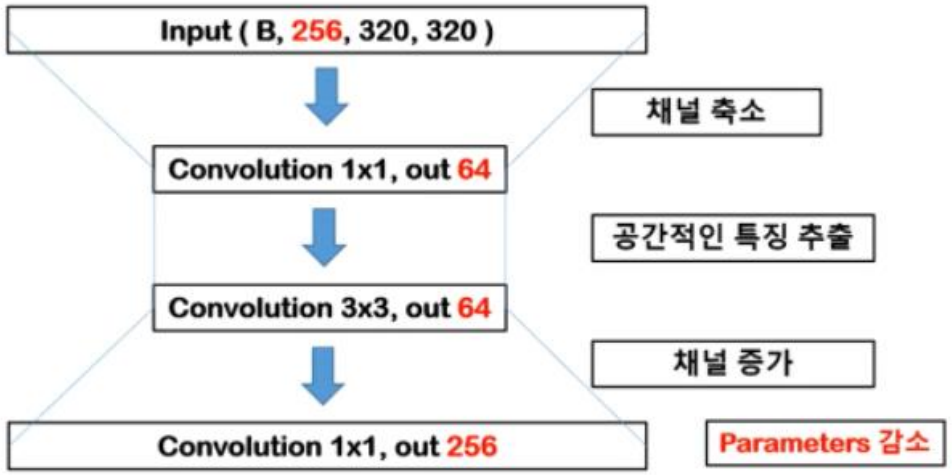
1X1 Convolution

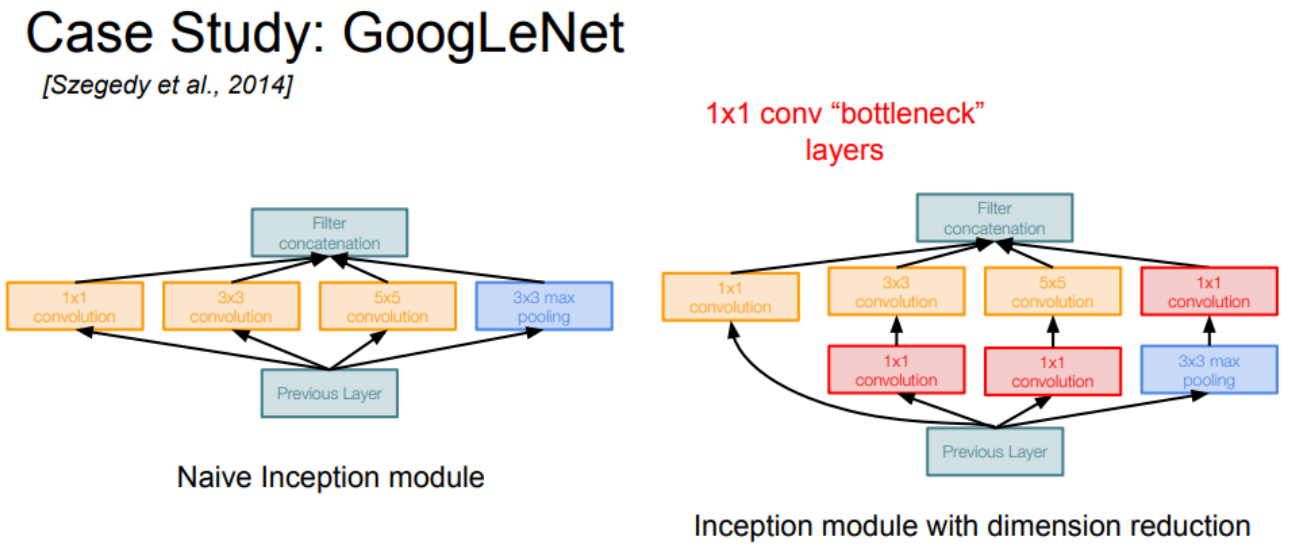




- Spatial dimensions 보존 및 depth 감소
  ex) 56x56x64 → 56x56x32
- 연산량 감소
  : 1x1 convolution 적용하여 channel(depth) 조절 후
  필터 적용 → parameter 감소, 계산량 감소

2. Case Studies: GoogLeNet





Bottleneck 구조

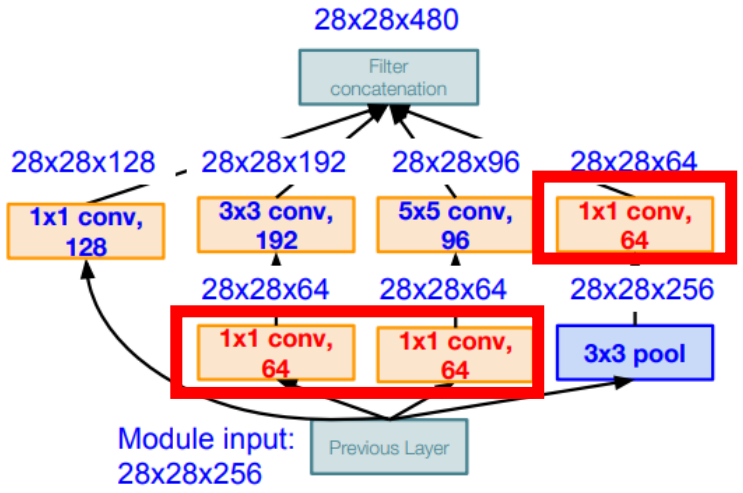
: 연산량이 많을때 1x1 convolution으로 channel을 줄였다가 channel을 다시 늘려 연산량을 최소화

→ Naïve Inception Module에 Bottleneck layer 추가

2. Case Studies: GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]



Inception module with dimension reduction

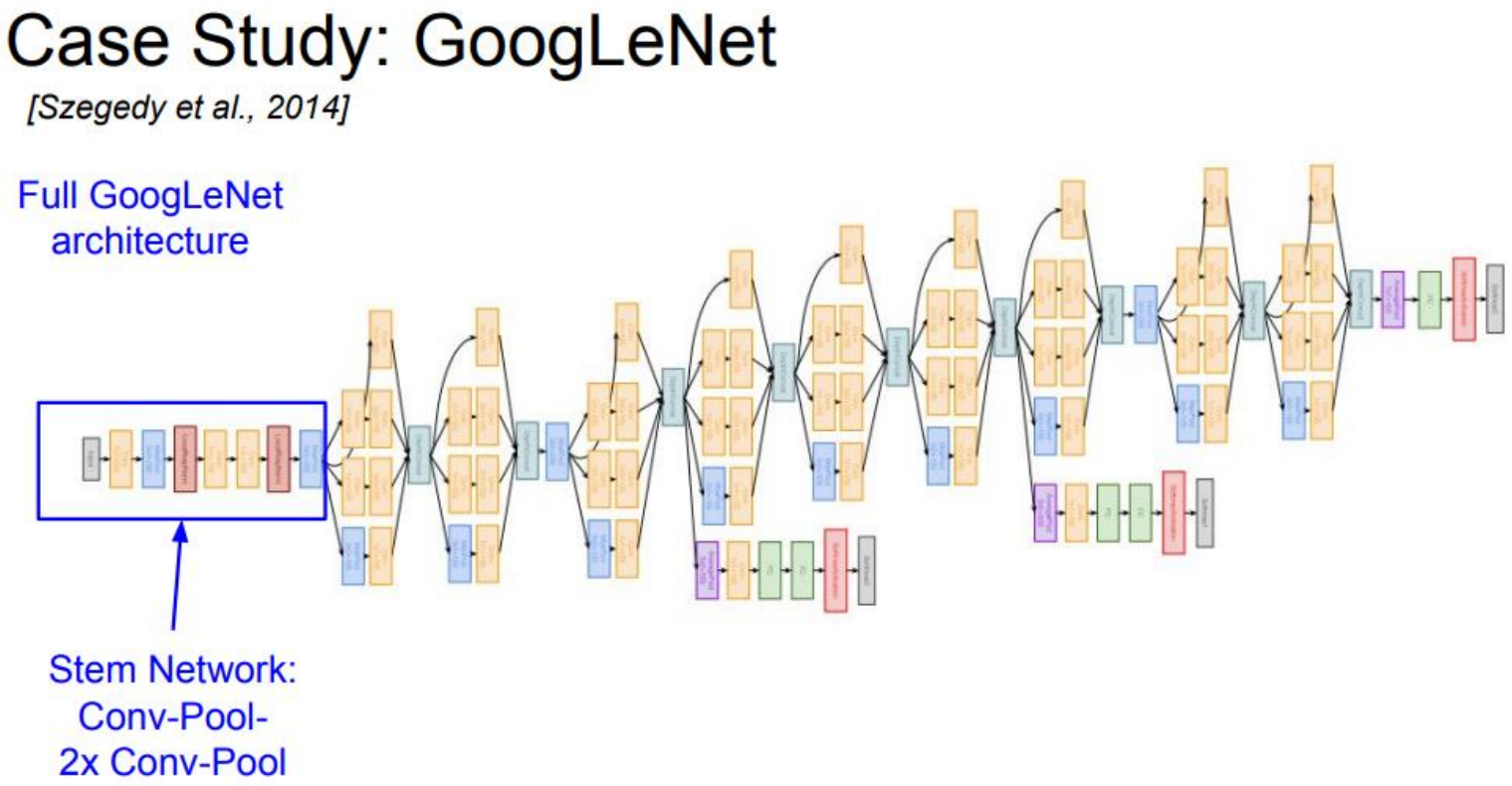
Using same parallel layers as naive example, and adding "1x1 conv, 64 filter" bottlenecks:

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

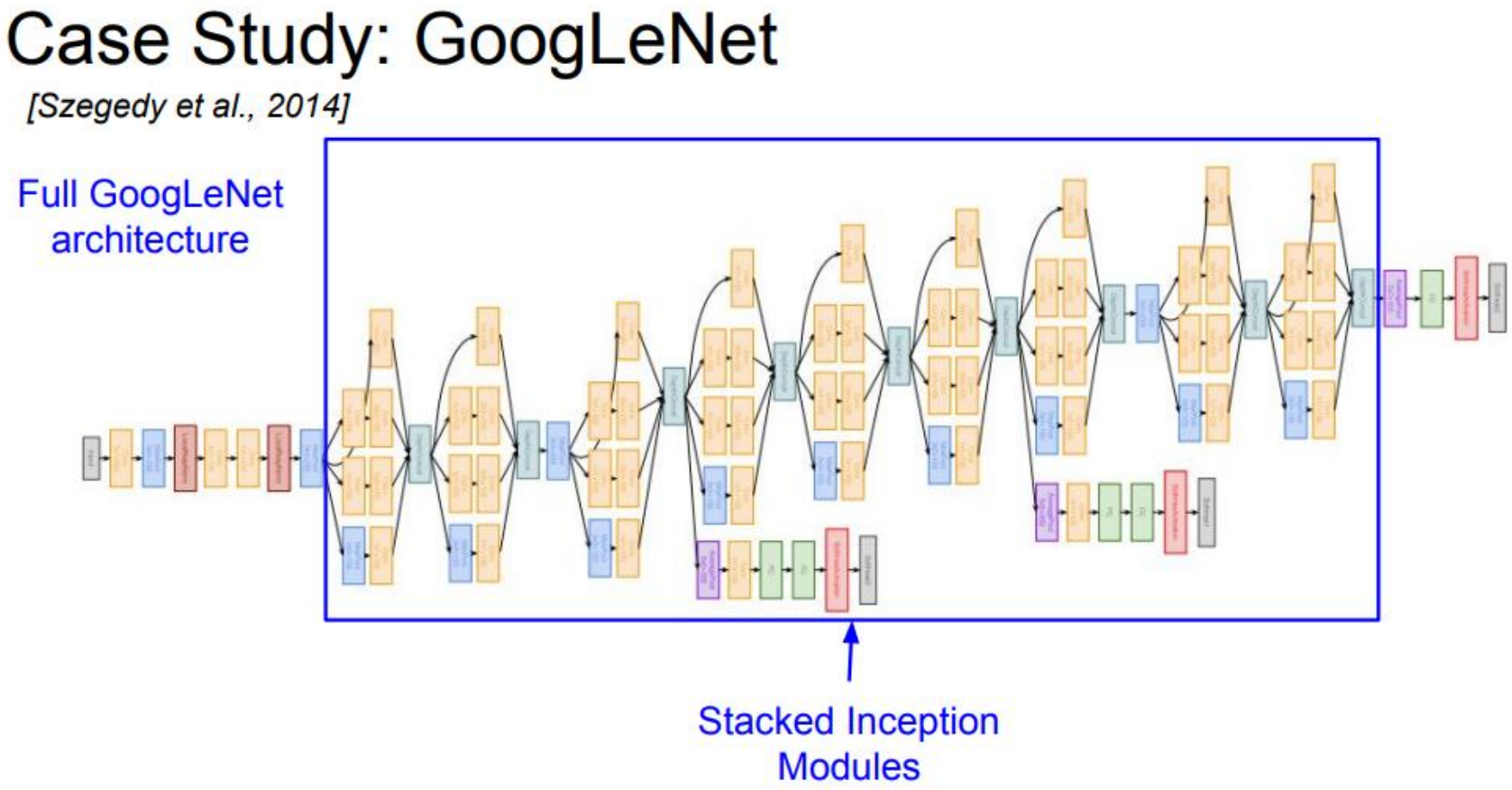
854M → 358M

2. Case Studies: GoogLeNet



1)Stem Network: 일반적인 convolutional Network layer과 유사

2. Case Studies: GoogLeNet



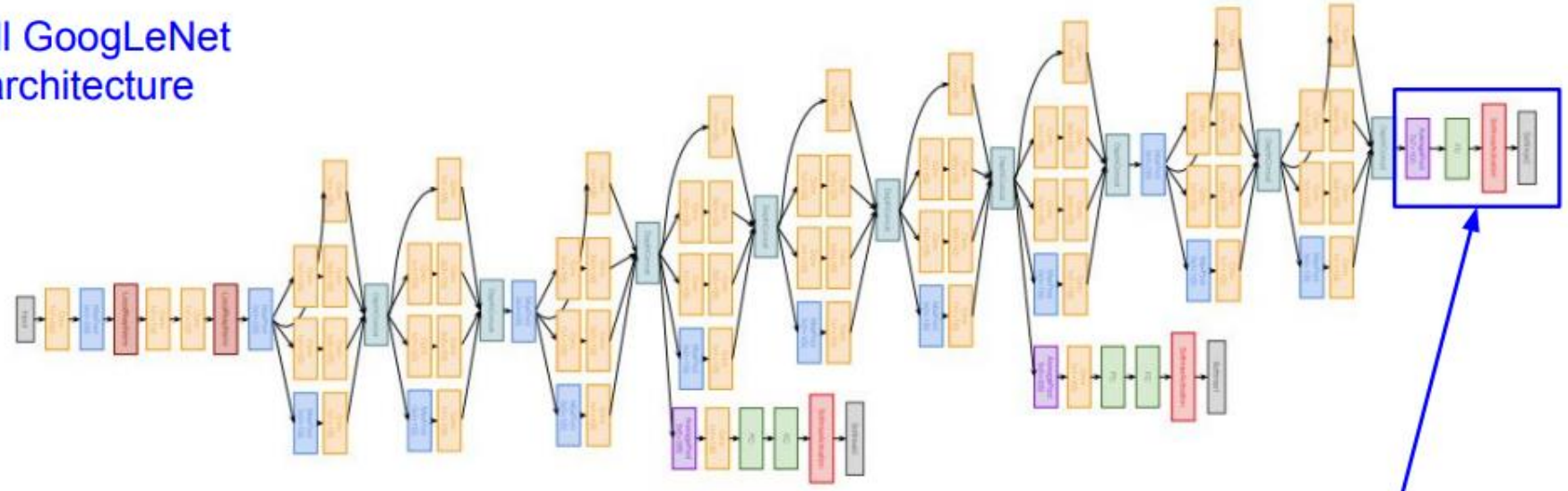
2) Stacked Inception Modules

2. Case Studies: GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture



Classifier output
(removed expensive FC layers!)

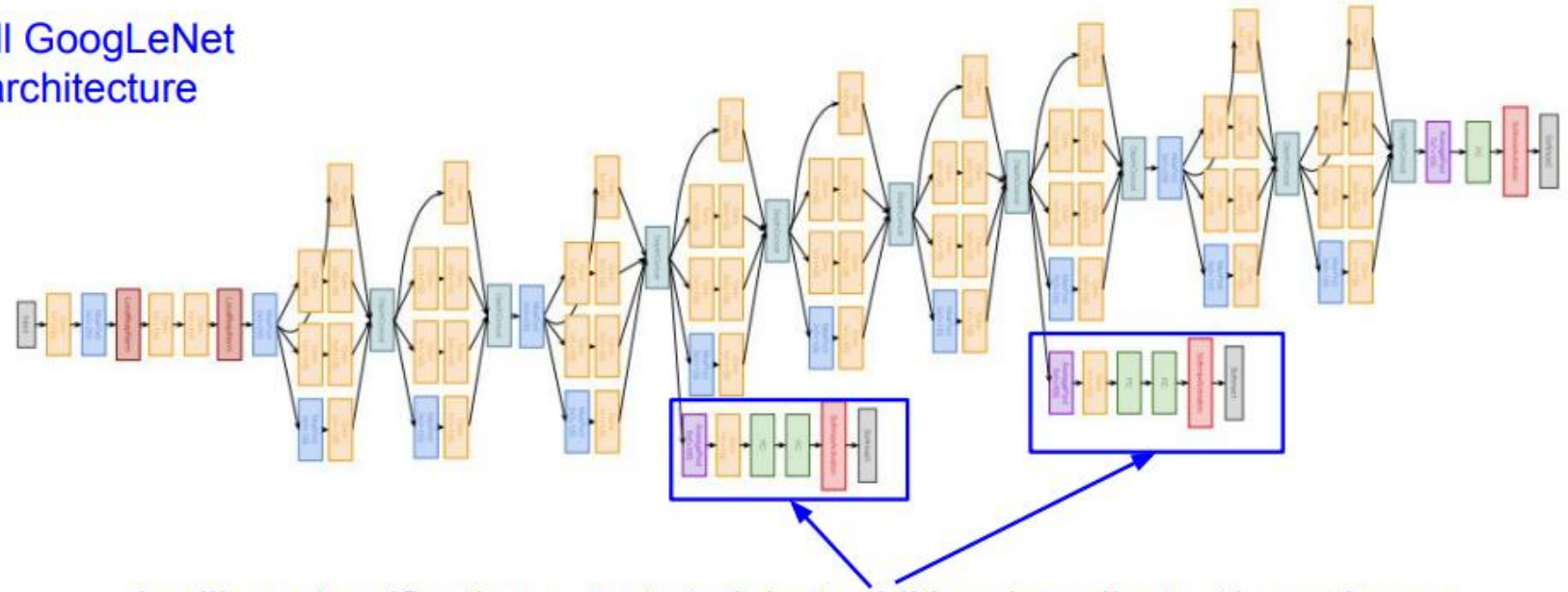
3) Classifier output (FC layer X)

2. Case Studies: GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture



Auxiliary classification outputs to inject additional gradient at lower layers
(AvgPool-1x1Conv-FC-FC-Softmax)

4) Auxiliary classifer : 중간에서 loss 구하여 역전파 적절하게 되도록 함

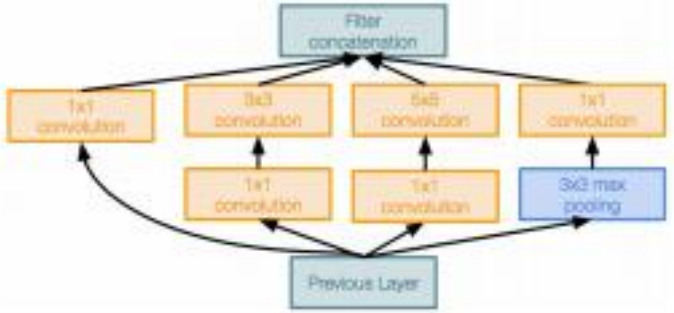
2. Case Studies: GoogLeNet

Case Study: GoogLeNet

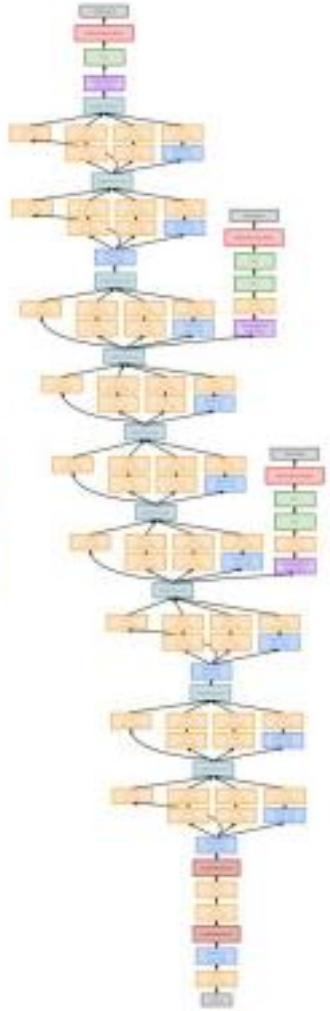
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- No FC layers
- 12x less params than AlexNet
- ILSVRC’14 classification winner (6.7% top 5 error)

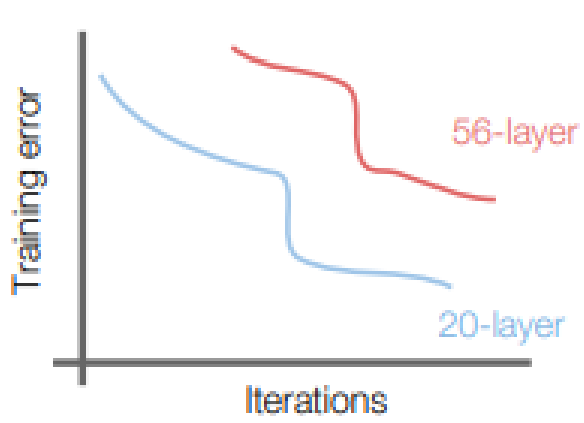


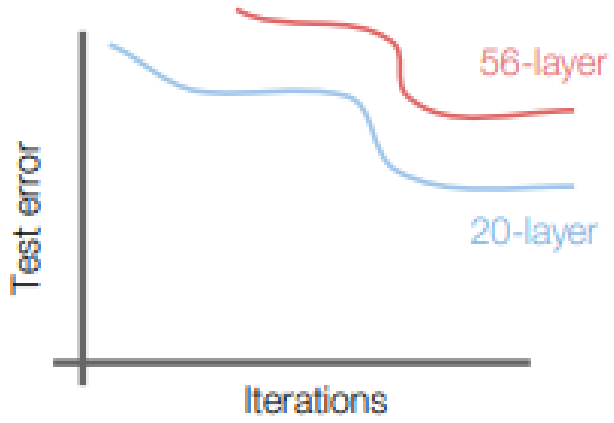
Inception module



2. Case Studies: ResNet

Plain CNN(skip connection 사용x)을 깊이 쌓으면 성능이 반드시 향상?
→ X





- Training error: 20-layer< 56-layer
- Test error: 20-layer<56-layer

Overfitting: Training error ↓, Test error ↑

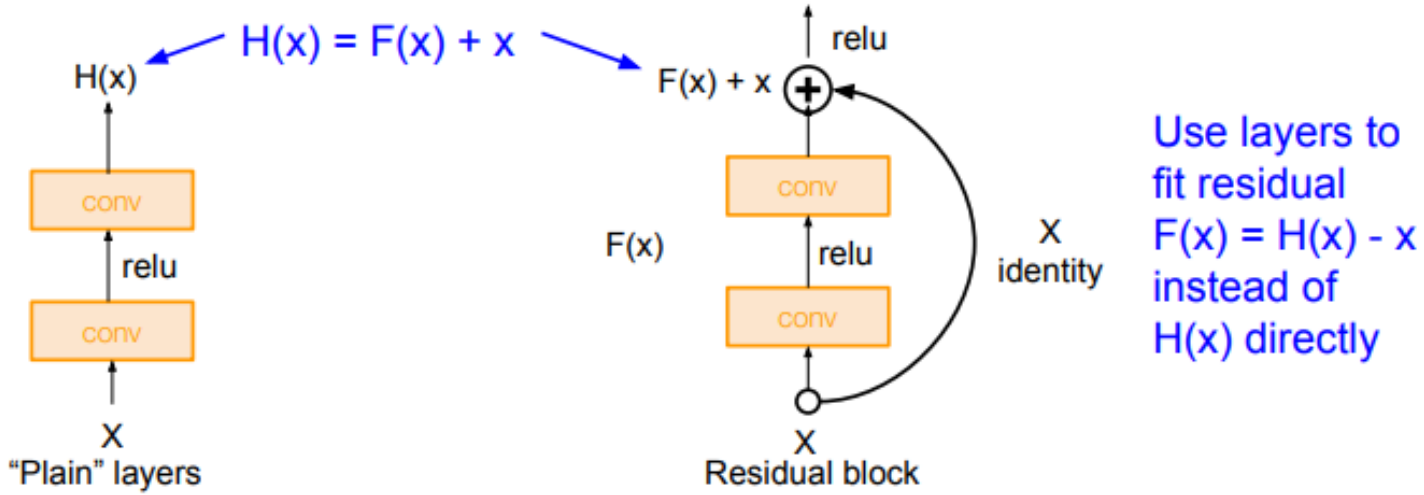
→ Overfitting의 문제가 아니라 **optimization 문제!**

2. Case Studies: ResNet

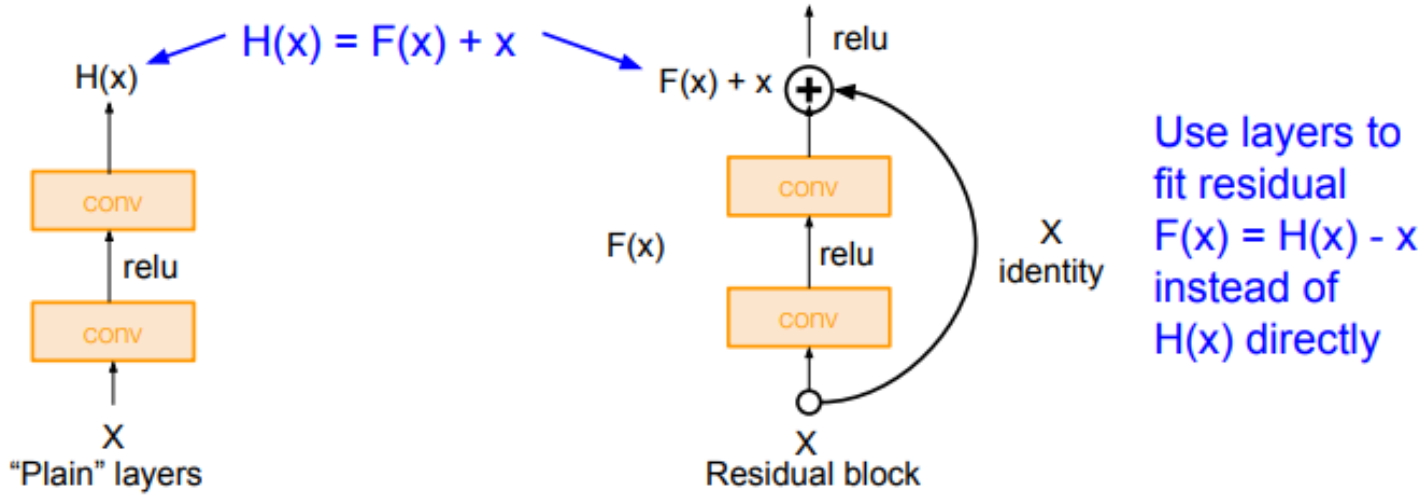
Hypothesis: layer을 깊게 쌓으려면 Overfitting의 문제가 아니라 **optimization 문제**로 접근

- 깊은 network일 때 기울기 소실문제가 원인
- 최소한 더 얕은 layer의 network만큼 성능이 있어야함
  → Identity mapping을 하고, layer을 추가하는 방식

*Identity mapping이란 입력으로 들어간 값 x가 어떠한 함수를 통과하더라도 x가 출력,항등함수의 개념



2. Case Studies: ResNet



Residual Block

- 깊은 layer에서는 H(x)를 바로 찾는 것이 어려움
- F(x)+x 형태
  (x:기존 layer or 복사할 얕은 layer 가중치 전달하는 skip connection
  F(x): 추가로 학습할 conv가중치)
- 얕은 layer 복사하려면 F(x)=0 설정, 즉 가중치 0으로 하여 그대로 통과시키기
- H(x)-x, 즉 F(x)가 최소화하는 방향 = F(x) 변화가 생길 때마다 결과 확인하기 쉬워 확인하며 최적화하는 과정
- F(x)+x 미분 → F' (x)+1 으로 최소 1 이상의 gradient 가져 기울기 소실문제 해결

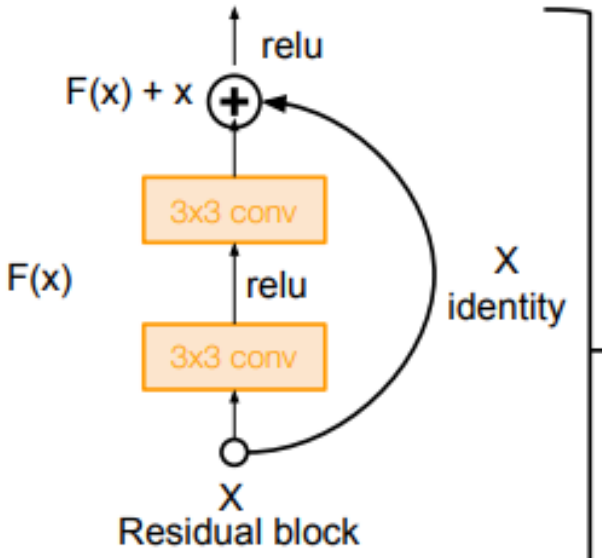
2. Case Studies: ResNet

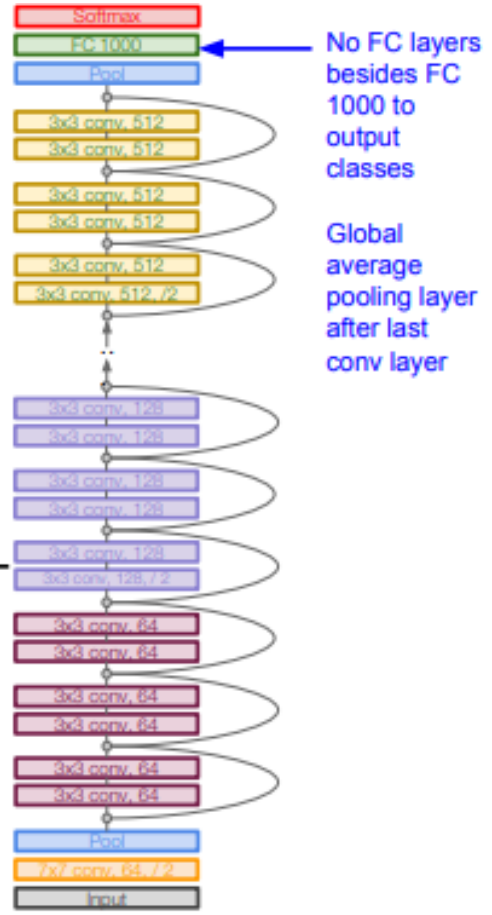
Case Study: ResNet

[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)



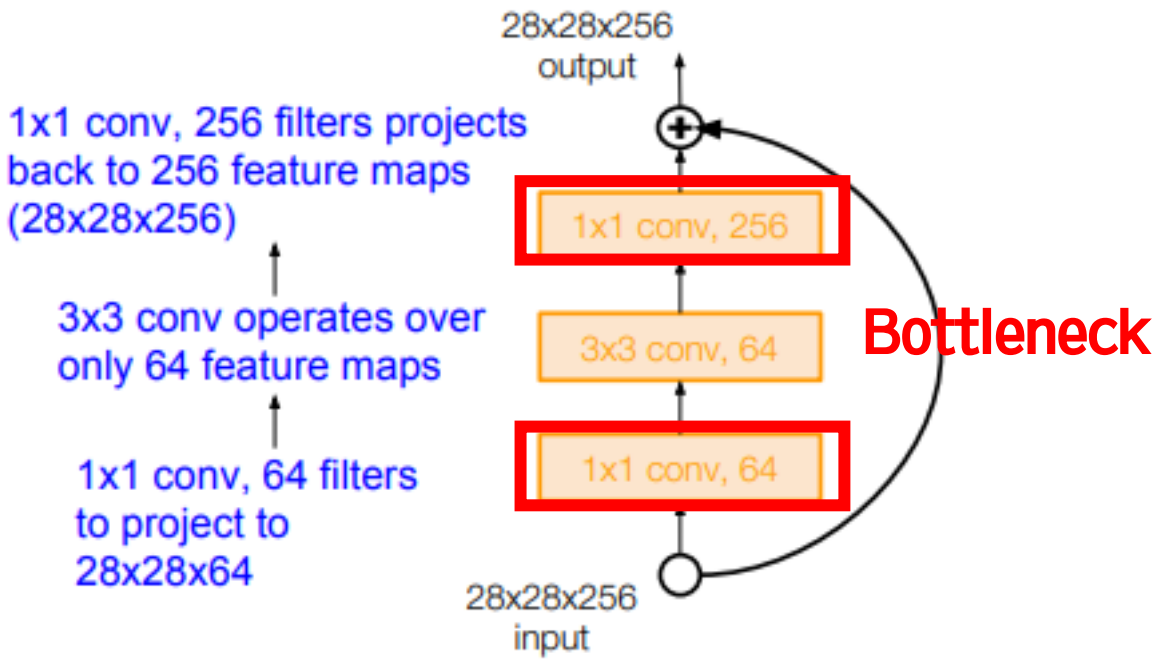


No FC layers besides FC 1000 to output classes

Global average pooling layer after last conv layer

2. Case Studies: ResNet

For deeper networks (ResNet-50+), use "bottleneck" layer to improve efficiency (similar to GoogLeNet)



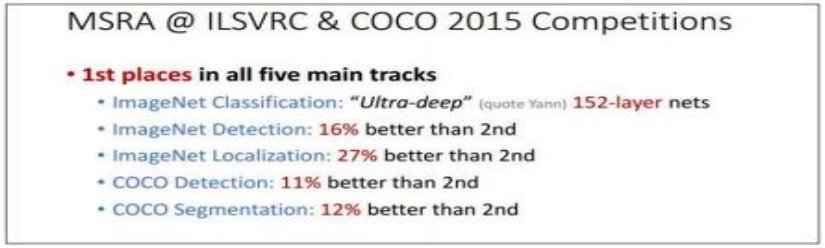
2. Case Studies: ResNet

Training ResNet in practice:

- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

ResNet의 성과

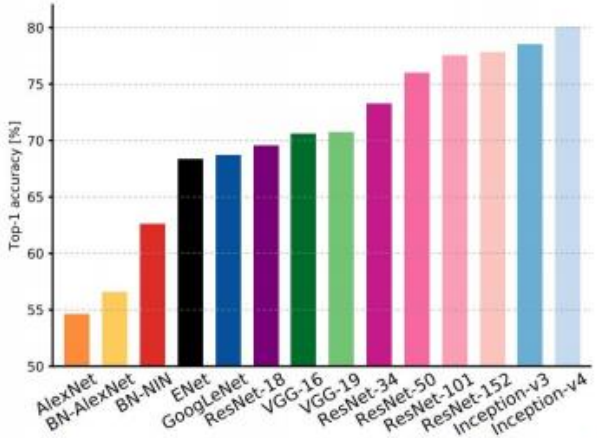
- 152 layer까지 깊은 layer 가능
- Training error ↓
- 2015 all ILSVRC, COCO competitions 1st place
- 3.6 % error로 사람보다 좋은 performance

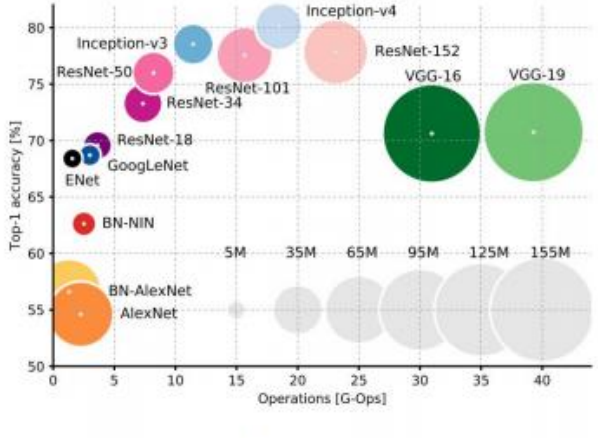


ILSVRC 2015 classification winner (3.6% top 5 error) -- better than "human performance"! (Russakovsky 2014)

Comparison

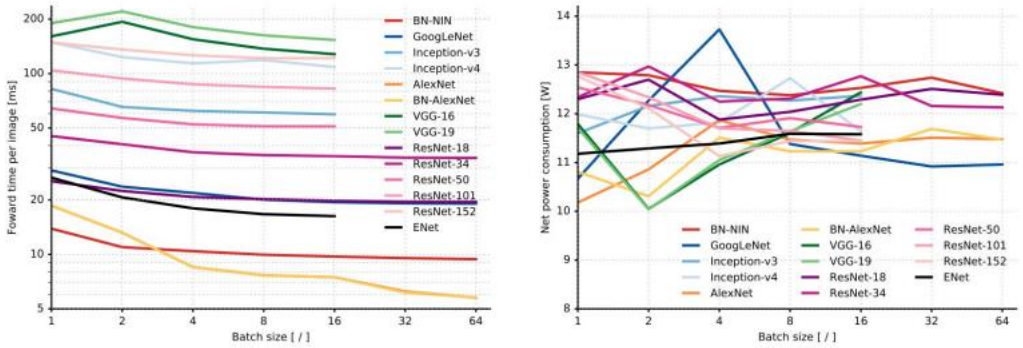
Comparing complexity...





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

Forward pass time and power consumption



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

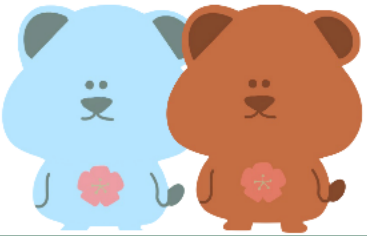
Accurracy 👍 Inception-v4:Resnet +Inception

Efficiency 👍 GoogLeNet

☺ ResNet (depending on model)

▼ VGG, AlexNet

Other Architectures



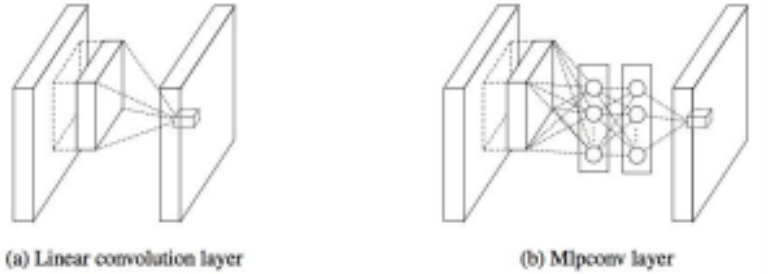
EWHA EURON

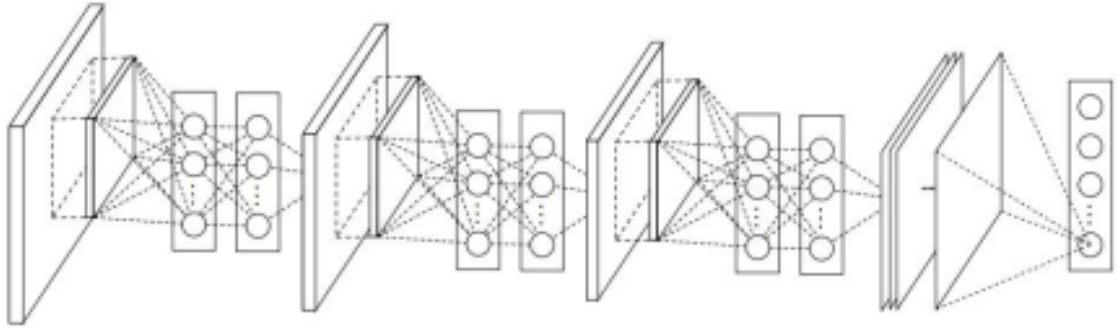
3. Other Architectures

Network in Network (NiN)

[Lin et al. 2014]

- Mlpconv layer with "micronetwork" within each conv layer to compute more abstract features for local patches
- Micronetwork uses multilayer perceptron (FC, i.e. 1x1 conv layers)
- Precursor to GoogLeNet and ResNet "bottleneck" layers
- Philosophical inspiration for GoogLeNet

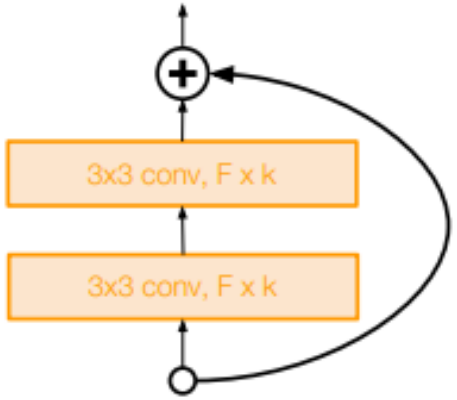




3. Other Architectures

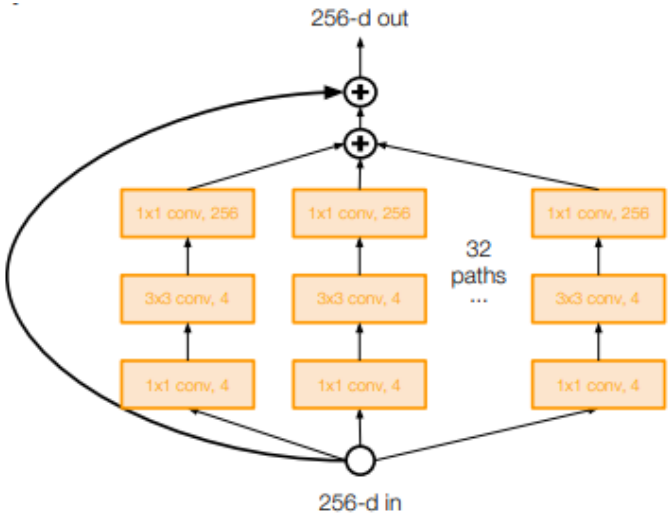
ResNet 개선

Direct path▲

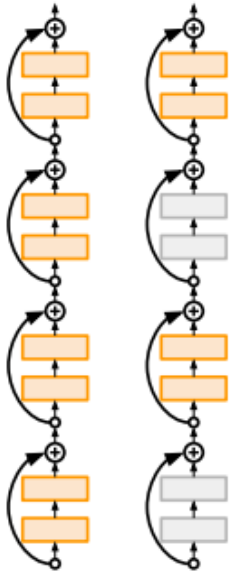


Wide residual block

Wider residual block
Fxk filter



multiple parallel pathways
(ResNext)

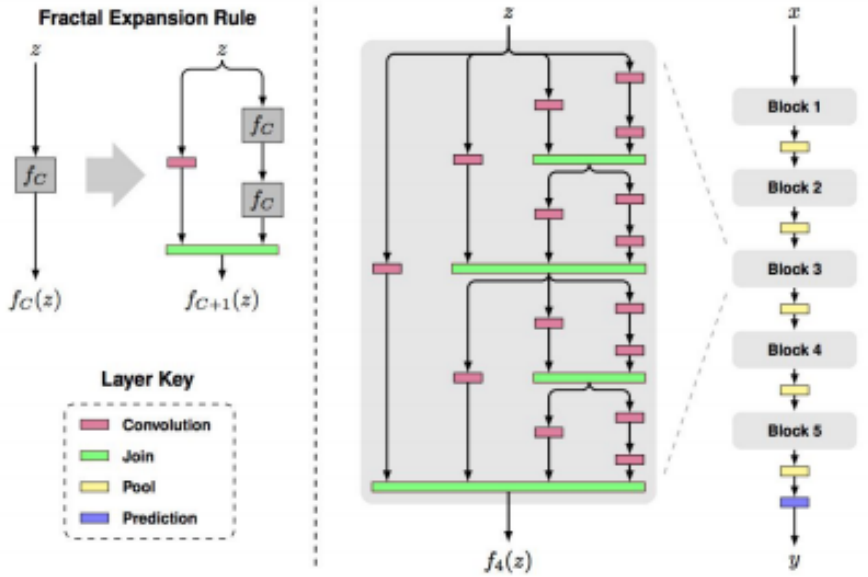


Deep Networks with
Stochastic Depth

3. Other Architectures

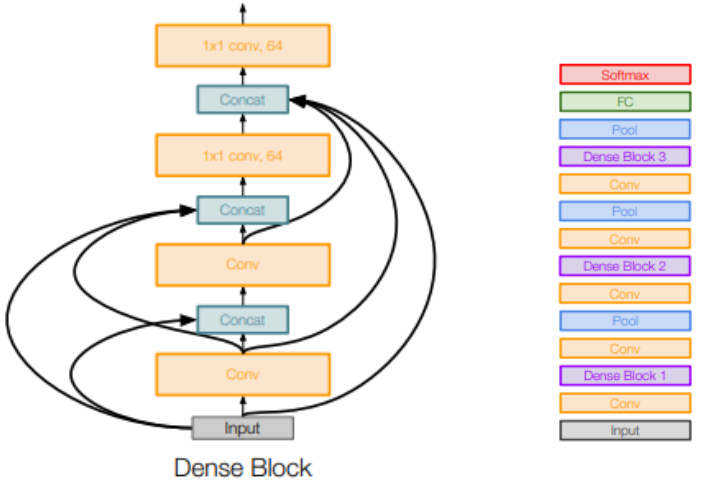
FractalNet

- key: Residual (x), transitioning shallow to deep layer
- Fractal block
- 일부 path에서만 training
- 모든 network에서 test



Densely Connected Convolutional networks

- Dense Block
- Vanishing gradient ▼, feature propagation ▲

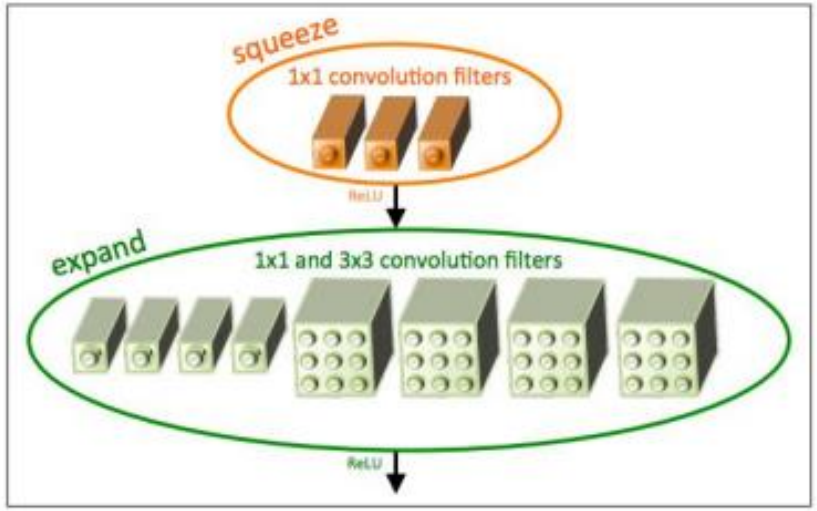


Dense Block

3. Other Architectures

SqueezeNet

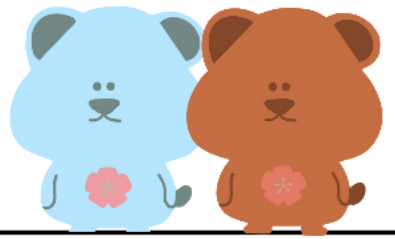
- 적은 수의 parameter 가진 small CNN
- 50배 적은 파라미터로 ImageNet AlexNet-level 달성
- Model compression (AlexNet보다 510배 축소)
- Fire modules
  :1x1 filter squeeze convolution layer + 1x1,3x3 filter expand layer



작은 CNN 모델의 장점

- 분산처리로 학습 시 서버간의 통신 적게 필요
- Cloud부터 자율주행차까지 새로운 모델 적용 용이
- 제한된 메모리가 있는 곳에서 적용하기 좋음

THANK YOU



EWHA EURON