이화여자대학교
EWHA WOMANS UNIVERSITY

LECTURE 9
CNN Architectures

2070070 정보경

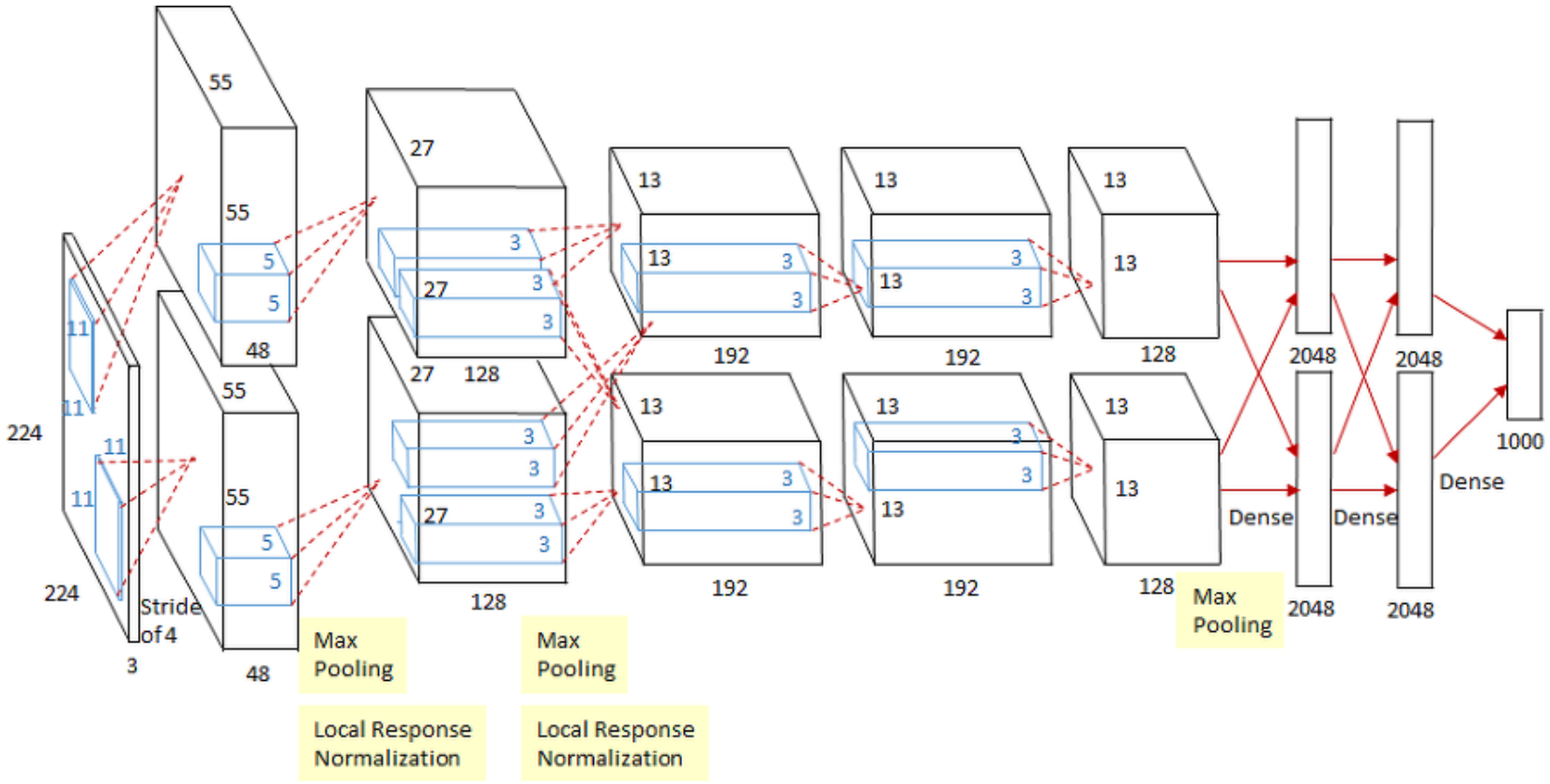
EWHA,
THE FUTURE
WE CREATE

CONTENTS

1. Alexnet

<최초의 Large scale CNN>

✓ Architecture



CONV1
MAX POOL1
NORM1

CONV2
MAX POOL2
NORM2

CONV3
CONV4
CONV5
MAX POOL3

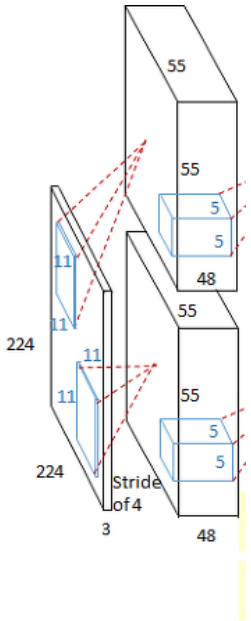
FC6
FC7
FC8

1. Alexnet

<최초의 Large scale CNN>

✓ Architecture

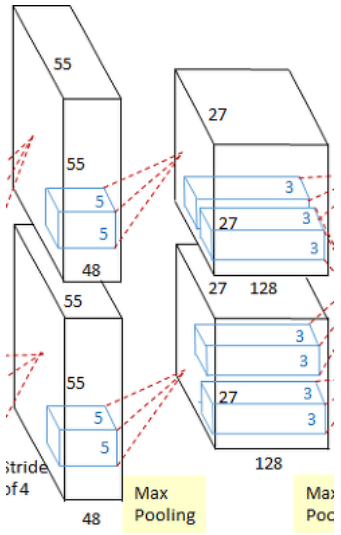
Input
Image: 227*227*3 (사진과 다름)



1) CONV1
Filter: 11*11*3 96개 / Stride: 4

Output 크기
(227-11)/4 +1 = 55
55*55*96

Parameter 개수
(11*11*3)*96개
(Filter size요소 * Filter 개수)



2) POOL 1
Filter: 3*3*3 96개 / Stride: 2

Output 크기
(55-3)/2 +1 = 27
27*27*96

Parameter 개수
0개
(학습의 가중치=Parameter이므로)

1. Alexnet

<최초의 Large scale CNN>

✓ Architecture

위와 같은 과정으로 계산할 시,
[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)

1. Alexnet

<최초의 Large scale CNN>

✓ Detail

최초의 ReLU사용

Norm layer 사용 (현재는 잘 쓰이지 않음)

Data Augmentation 많이 사용

Dropout: 0.5

SGD Momentum: 0.9

Learning Rate: 초기: 1e-2 / 1e -10까지 감소

L2 Weight Decay: 5e-4

7개의 CNN ensemble, 성능 향상

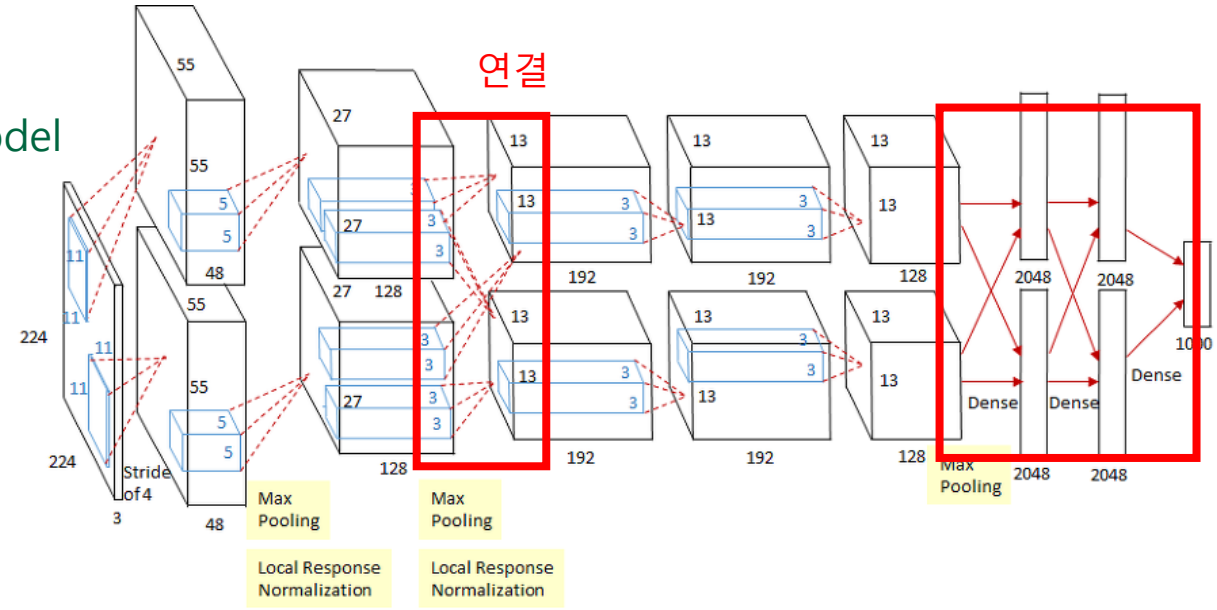
1. Alexnet

<최초의 Large scale CNN>

✓ Detail

과거 GTX 580의 memory: 3GB
한 GPU로 모든 데이터 처리가 불가해 2개의 모델로 나누어 연결하는 방식 사용

2개의 Model

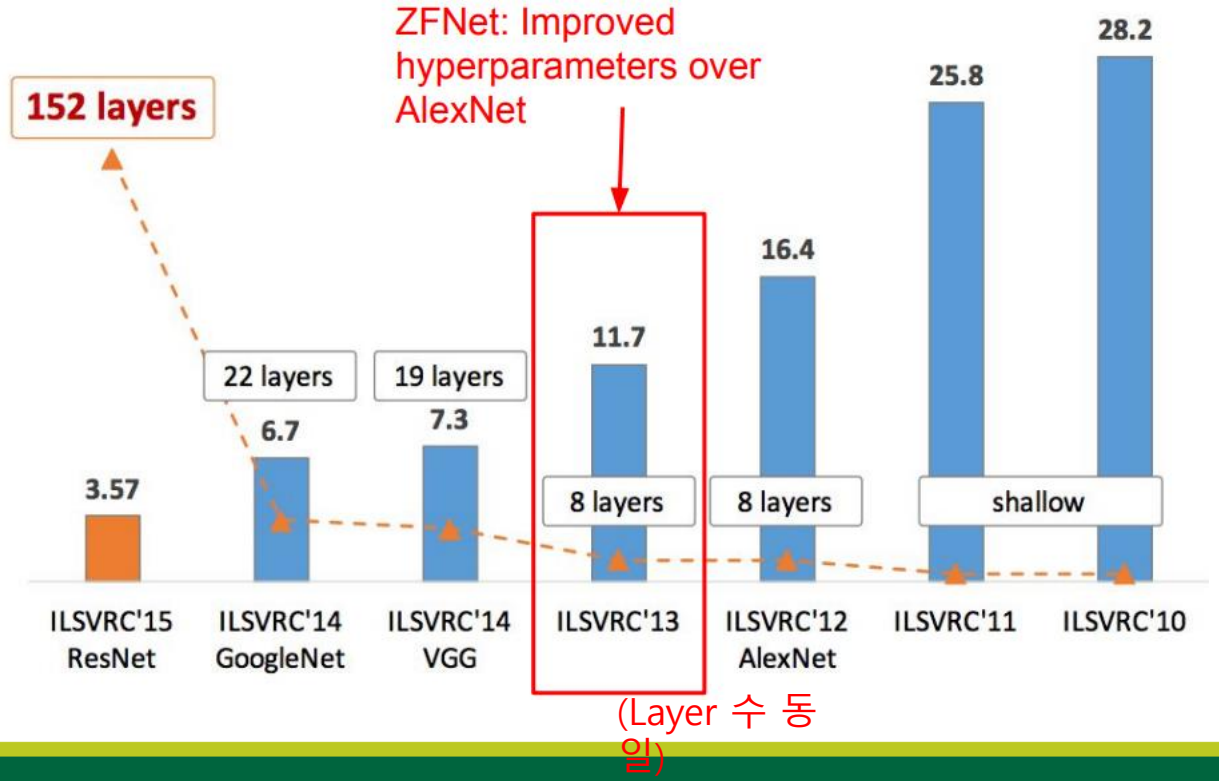


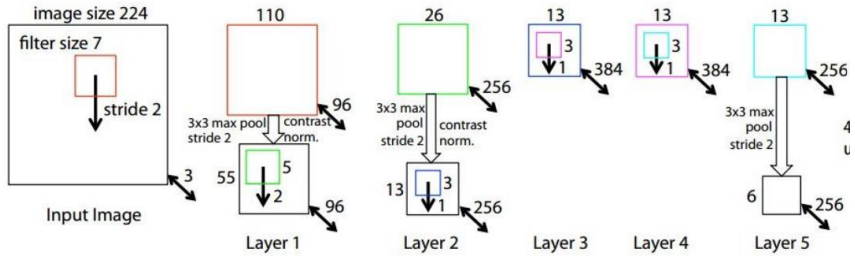
1. Alexnet

<최초의 Large scale CNN>

✓ ZFNet

2013 ImageNet 우승
AlexNet의 Hyperparameter 개선





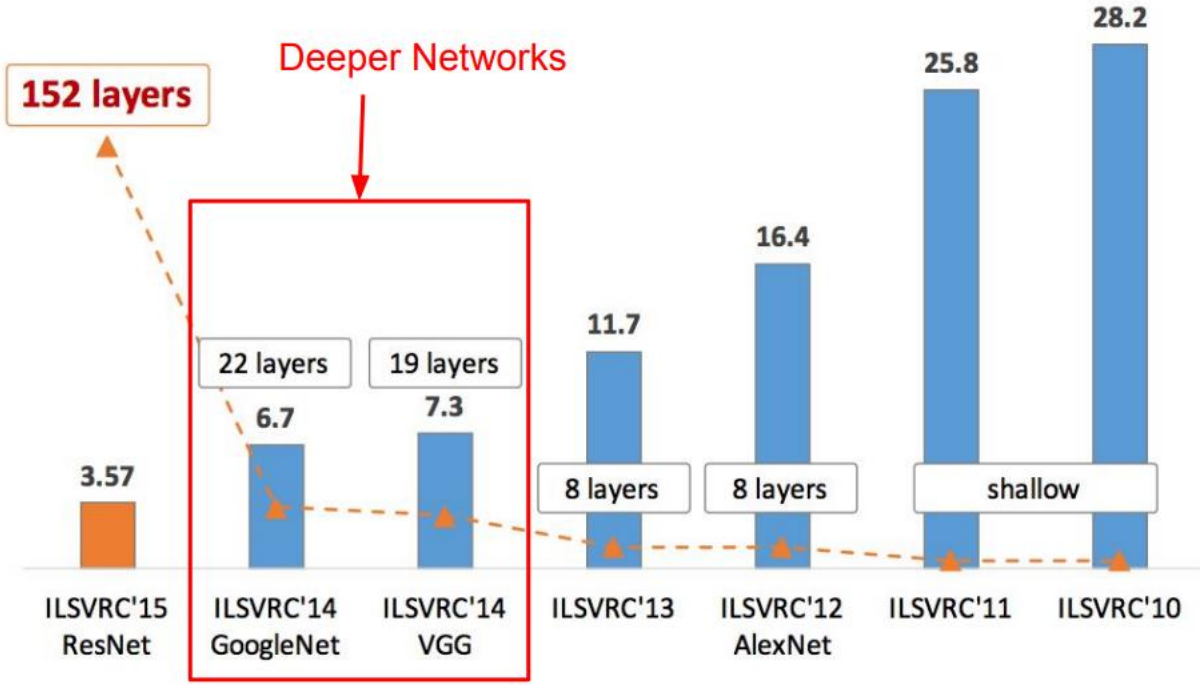
<AlexNet 개선점>
CONV1 Filter: 7*7*3 96개 / Stride: 2
CONV 3, 4, 5 Filter: 512, 1024, 512

2. VGGNet

<Deeper Network 사용>

✓ Deeper Network

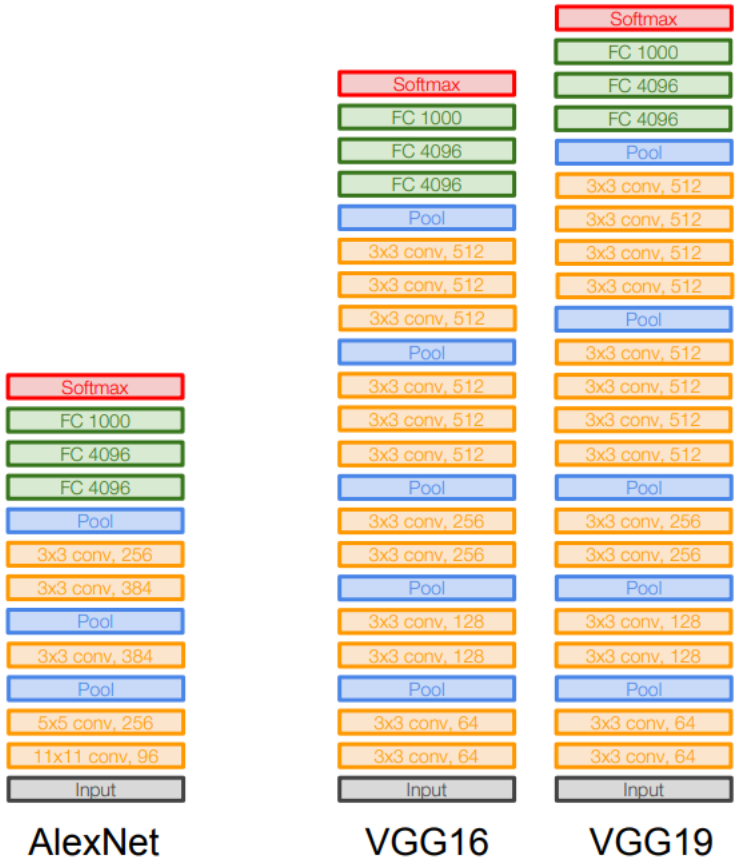
2014-GoogleNet 1위, VGGNet 2위
Deeper Networks 사용



2. VGGNet

<Deeper Network 사용>

✓ Architecture



Layer
16~19개 → 증가

CONV Filter
3*3,Stride 1 → Filter 크기 감소, Stride 감소

MAX POOL
2*2, Stride 2 → POOL 크기 감소

2. VGGNet

<Deeper Network 사용>

✓ Architecture

<Filter 크기를 줄인 이유>

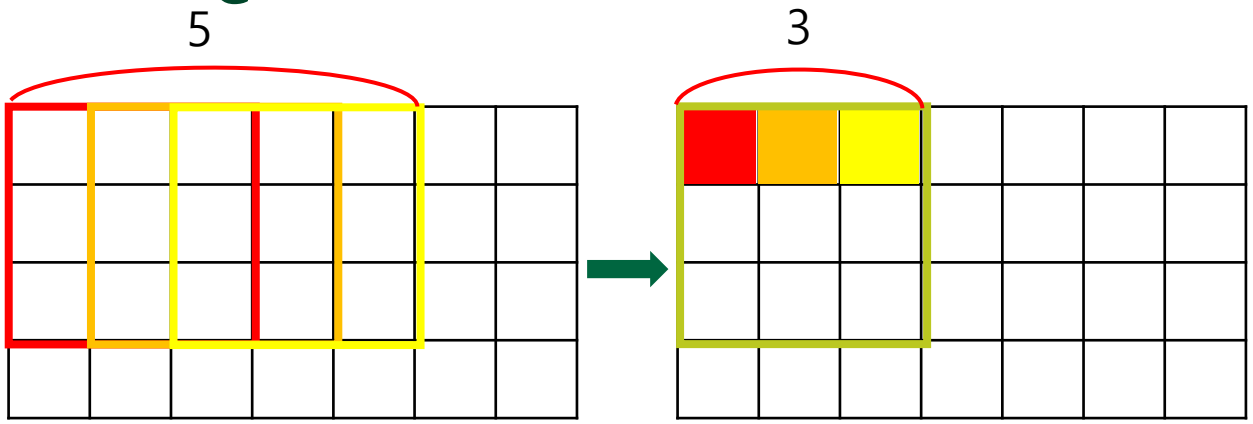
3*3 Filter, Stride 1 이용 시 7*7 Filter와 동일한 Receptive Field를 얻을 수 있음.
7*7 Filter보다 Deeper, 비선형적, 적은 Parameter
∴ 성능이 더 좋음

2. VGGNet

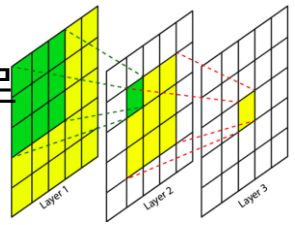
<Deeper Network 사용>

✓ Architectur e

*Receptive Field



1) Layer 1의 3*3 내 요소가 한 칸에 집약
2) Layer 2에 3*3 Filter 적용 시 , Stride 1이므로
   Layer 1에서의 5*5 범위가 파악됨
3) Layer 3에 3*3 Filter 적용 시, Stride 1이므로
   Layer 2에서의 5*5 범위
   =Layer1에서의 7*7 범위가 파악됨



*Parameter

Filter 개수가 3개/ Depth가 d일 때
7*7 Parameter: 7*7*d = 49d
3*3 Parameter: 3*3*3*d = 27d

2. VGGNet

<Deeper Network 사용>

✓ Architectur

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

1) 초기 CONV에 가장 많은 메모리 사용
2) FC에 가장 많은 Parameter 사용 (Dense connection)

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

2. VGGNet

<Deeper Network 사용>

✓ Detail

AlexNet (2012 Krizhevsky) 와 유사한 training process

Local Response Normalization (LRN) 사용X

VGG16 (16 Layers), VGG19 (19 Layers)

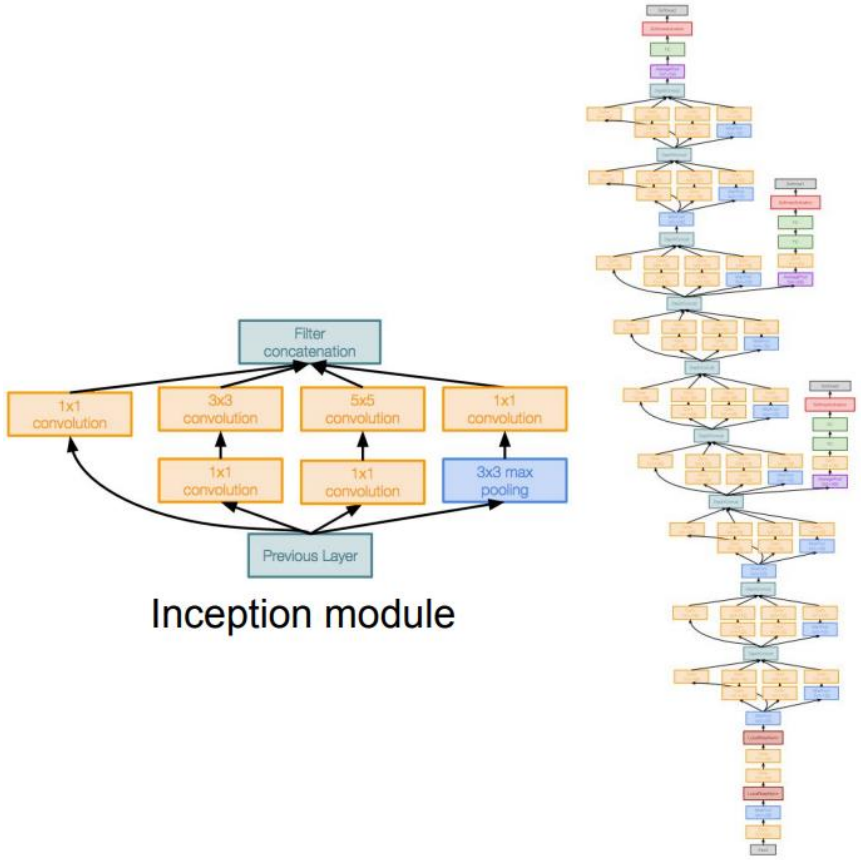
ensemble, 성능 향상

FC7: 특징 잘 추출, 일반화 능력이 좋음

3. GoogleNet

<Inception Module>

✓ Architecture



Inception module

Layer
22개

Inception Module

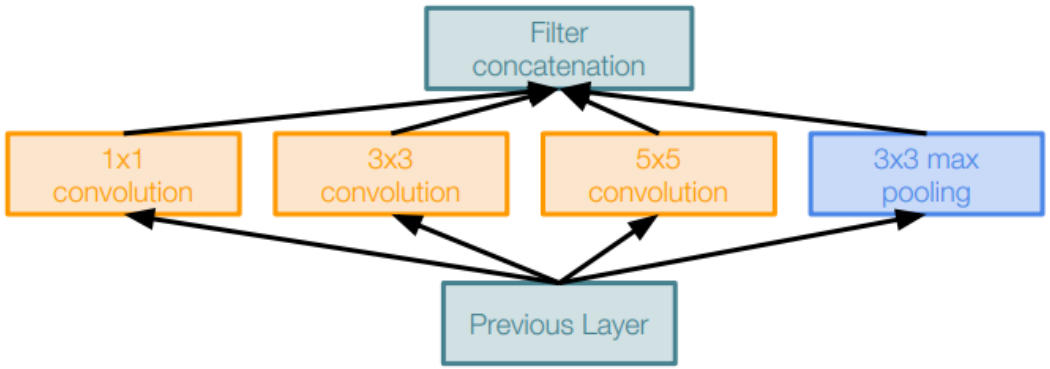
FC layer 없음
↓
Parameter
5M (VGG: 138M)

3. GoogleNet

< Inception Module >

✓ Architectur

- *Naïve Inception Module*



Local Network Topology를 제작, 이를 쌓은 형태

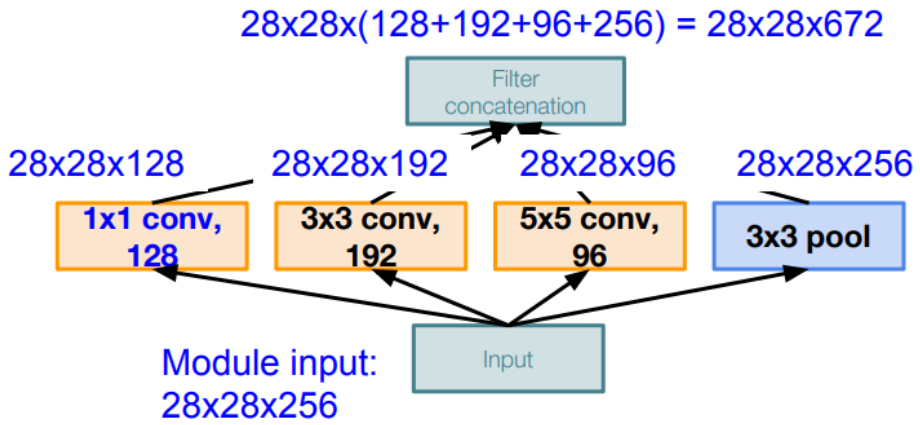
-동일 Input 받는 Filter가 병렬로 존재
-이후 출력을 Depth 방향으로 합하여 1개의 Tensor로 만듦

3. GoogleNet

< Inception Module >

✓ Architectur

*e Naïve Inception Module



Naïve Inception Module의 단점
<계산량 증가>
동일 Input
+
다수의 Layer를 최종적으로 1개의 Tensor로 만듦
= Depth 매우 크게 증가, 계산량 매우 많아짐

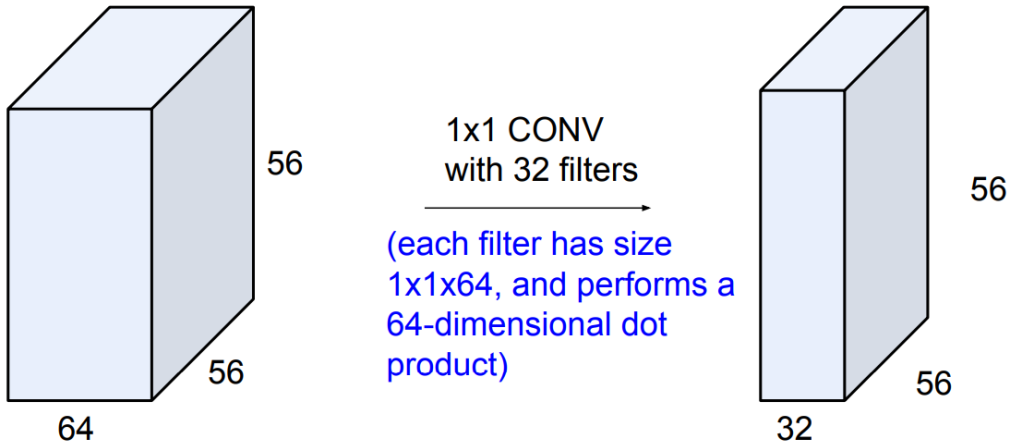
3. GoogleNet

< Inception Module >

✓ Architectur

*e Bottleneck Layer

Naïve module의 계산량 증가에 대한 해법
1*1 Convolution을 사용하여 병합 전 Layer들을 더 낮은 차원으로 변환



56*56*64 에 32개의 1*1 Filter CONV 처리
(각 요소를 내적하여 Depth만 줄임, 선형 결합)

=56*56*32의 Output

차원 감소

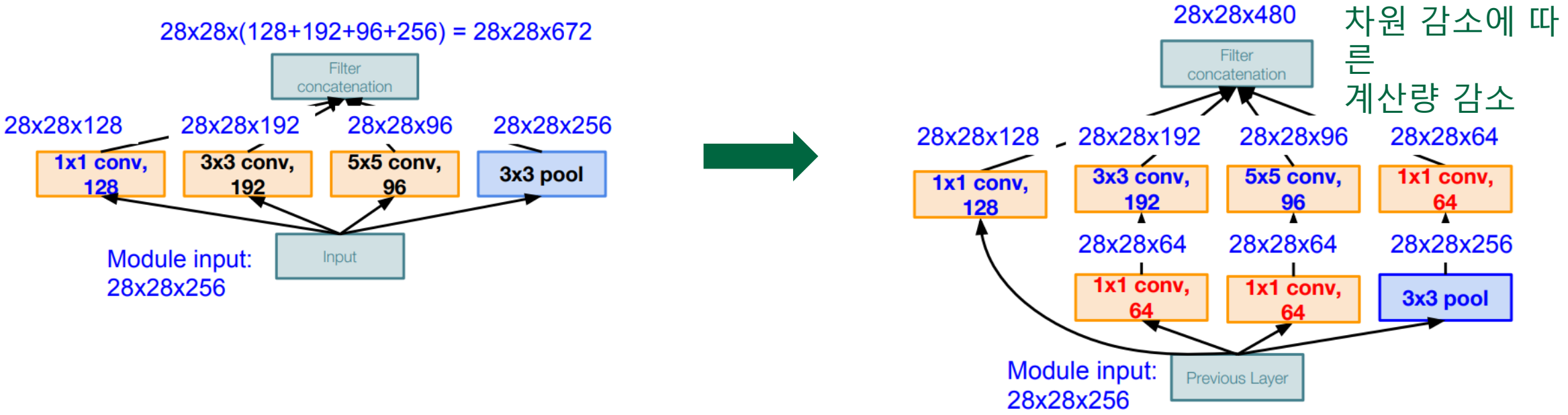
3. GoogleNet

< Inception Module >

✓ Architectur

e

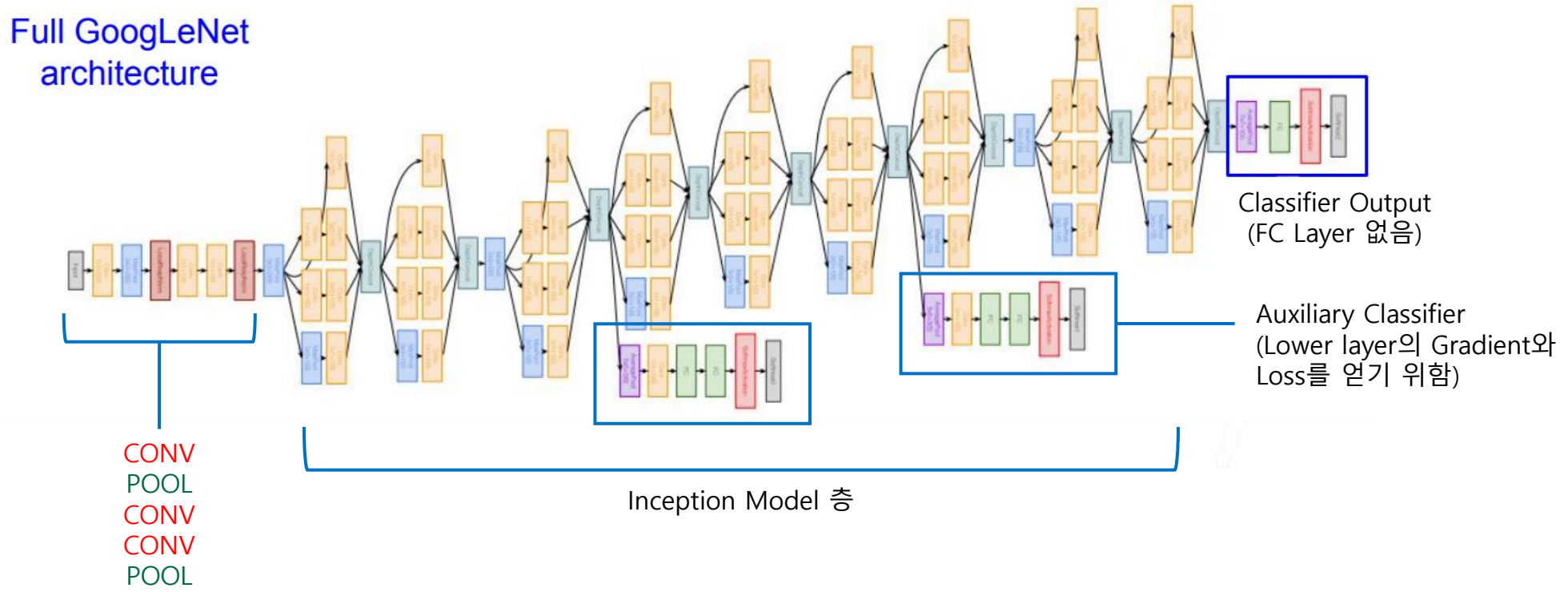
*Inception module with Bottleneck Layer



3. GoogleNet

< Inception Module >

✓ Architectur



가중치를 가진 Layer: 22개

3. GoogleNet

< Inception Module >

✓ Detail

Deeper Network / Computation 효율 향상

효율적인 Inception Module

FC Layer 없음

Parameter 적음 (Alexnet보다 12배 적음)

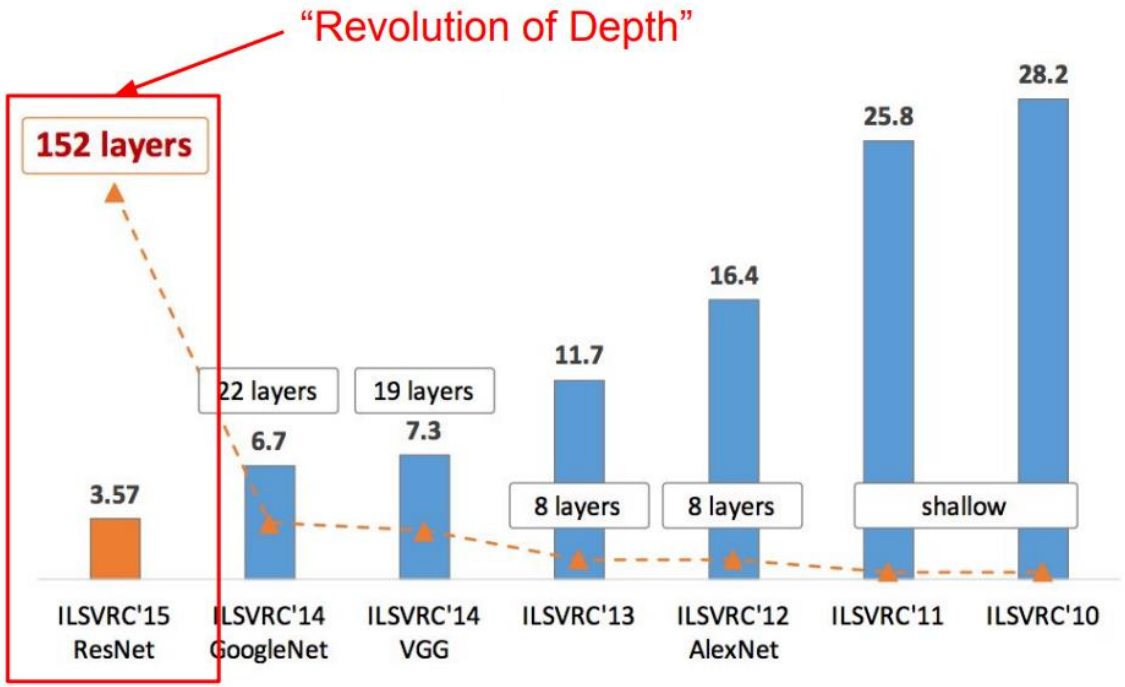
가중치를 가진 Layer: 22개

4. ResNet

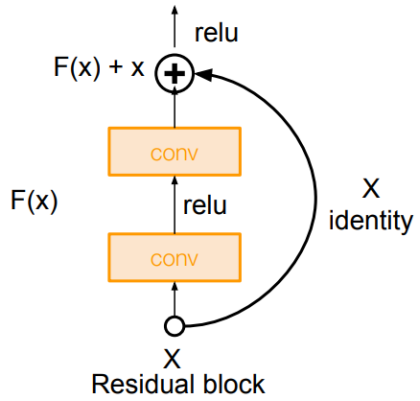
<Revolution of Depth>

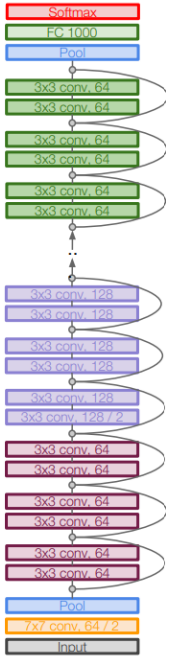
✓ Revolution of Depth

2015-ResNet 1위
152개의 Layer 사용 (GoogleNet: 22개)



Residual Connection을 사용한
Deep Network





4. ResNet

<Revolution of Depth>

✓ Hypothesis

일반 CNN의 Layer 수를 늘릴 시 성능의 변화는?

Layer가 많을 때 오히려 성능 감소
Training set에 대해서도 성능 감소, Overfitting이 이유가 아님

→ Optimization이 문제라는 가설 제기
모델이 깊을수록 최적화가 어려울 것이다

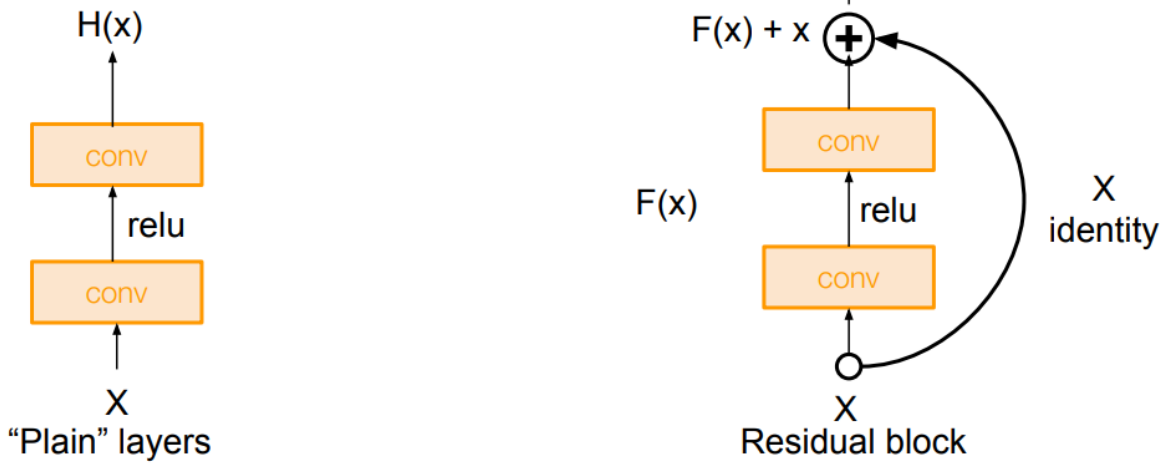
4. ResNet

<Revolution of Depth>

✓ Hypothesis

모델이 깊을수록 최적화가 어려울 것이다

많은 Layer에서도 얕은 Model과 같은 성능을 내는 방법
1) 얕은 model에서 계산된 가중치를 복사
2) identity mapping으로 input data를 바로 Output으로
= Residual Block

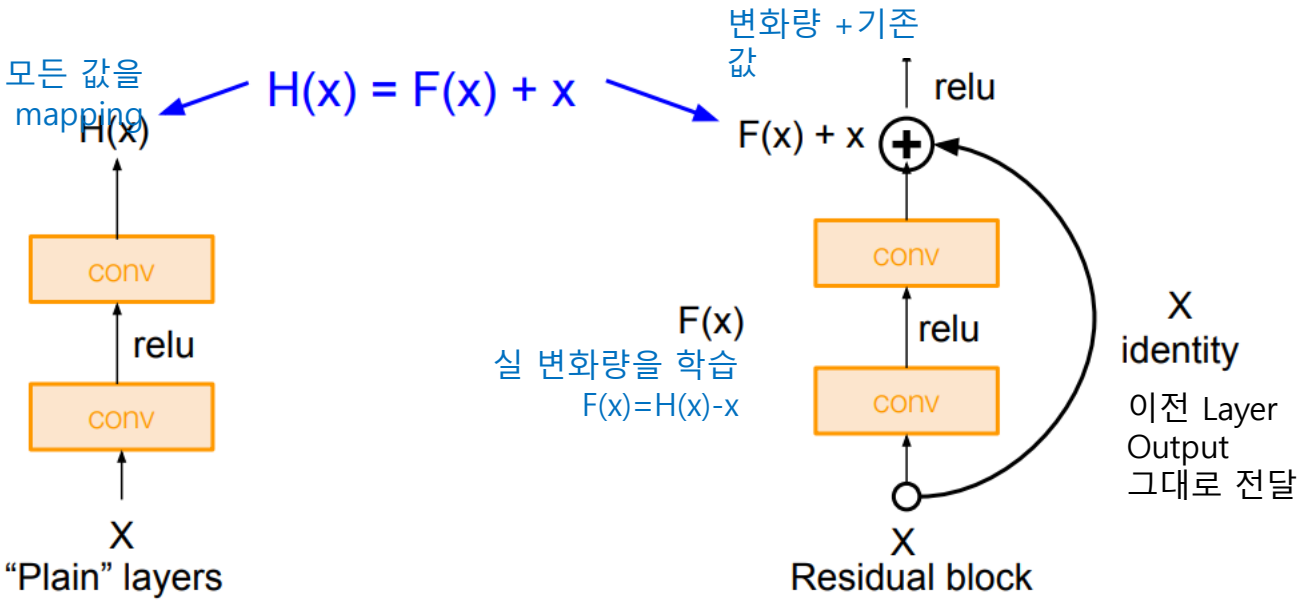


4. ResNet

<Revolution of Depth>

✓ Architecture

*Residual Block



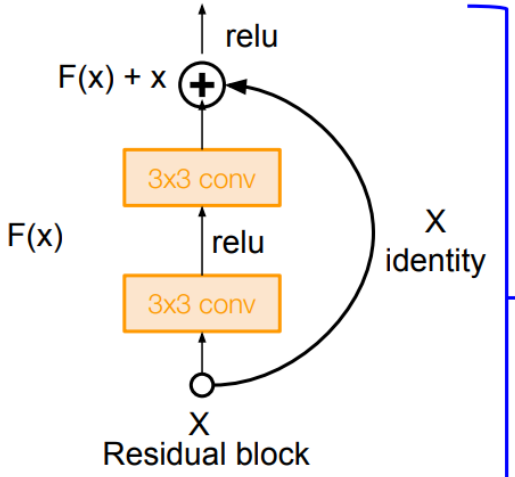
모든 mappin에 대해 fit하는 대신,
Residual mappin에 Fit하도록 함 → 관점의 변화

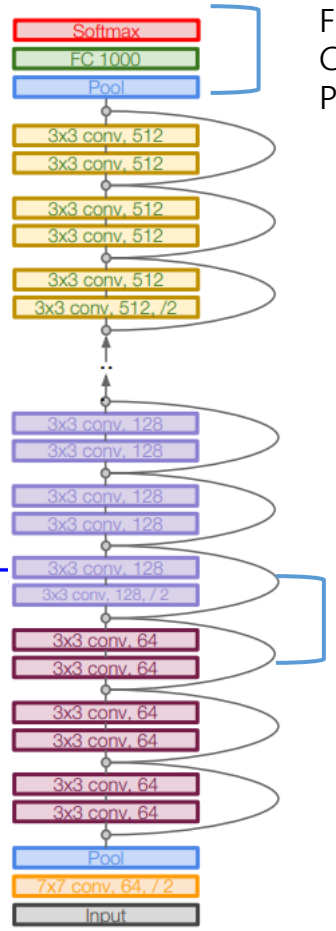
" 기존 값에서 얼마를 변화시키는 것이 효율적인가를 학습"

4. ResNet

<Revolution of Depth>

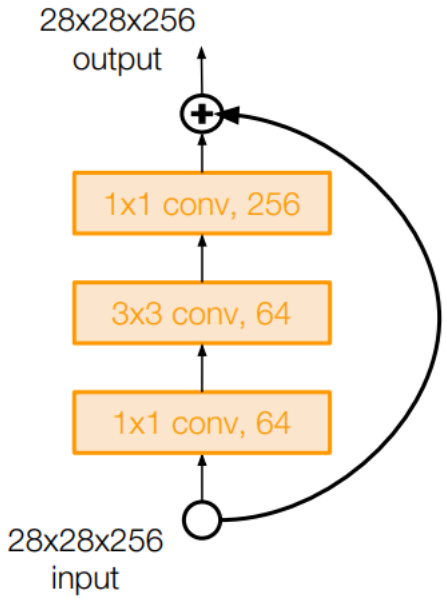
✓ Architecture





FC 1000 외 기타 FC 없음
CONV Layer 완료 후 Global Average Pooling Layer 로 Pooling

Filter 수 2배,
Stride를 2로 증가
= Depth 증가, Downsampling



(+) Model Depth가 50 이상일 때만 Bottleneck Layer 사용

2개의 3*3 CONV Layer를 가진 Residual Block을 쌓음

4. ResNet

<Revolution of Depth>

✓ Detail

모든 CONV Layer 이후 Batch Normalization

Xavier/2 initialization의 초기화 성능이 좋음

SGD Momentum: 0.9

Learning Rate: 초기 0.1
/ Validation Error가 줄지 않는 시점부터 기존 rate/10

Mini-batch의 size: 256

Weight Decay 1e-5

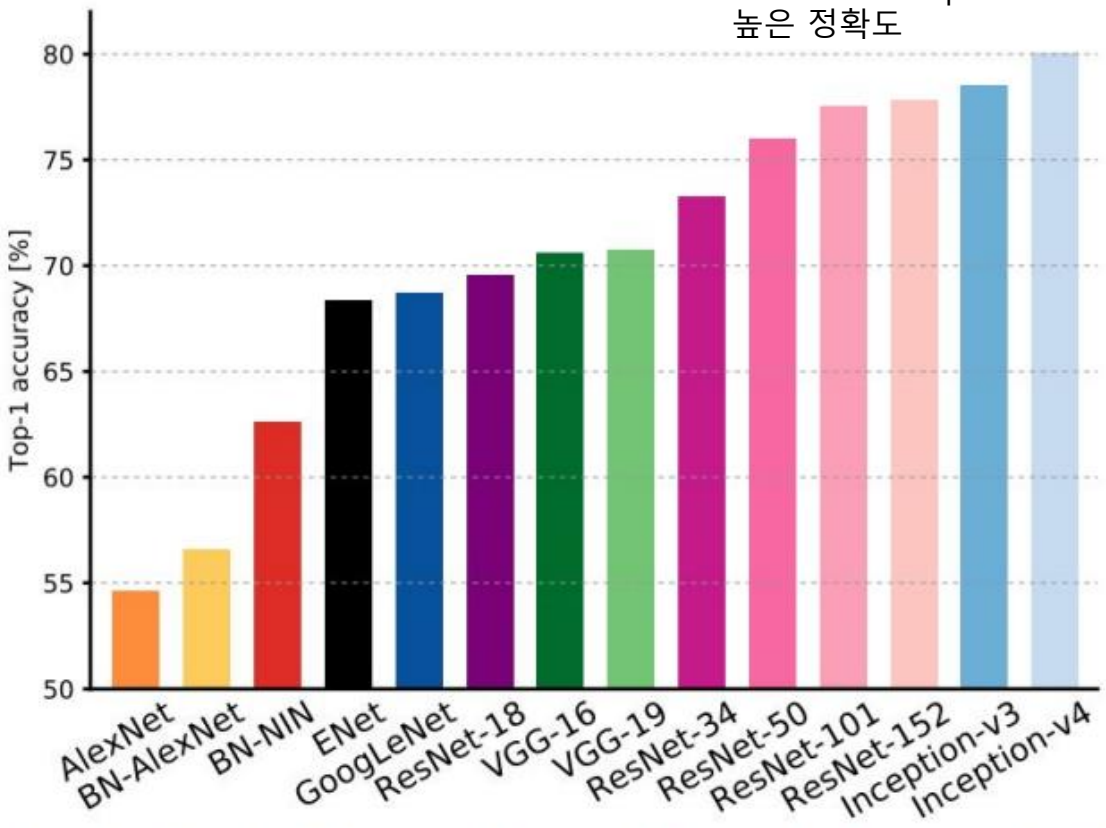
Dropout 사용하지 않음

Degrading 없이 Very Deep Network 학습 가능

Deeper Network 의 Train error Lowing에 성공
(ResNet에서 얕은 Network의 Error가 더 작은 경우 X)

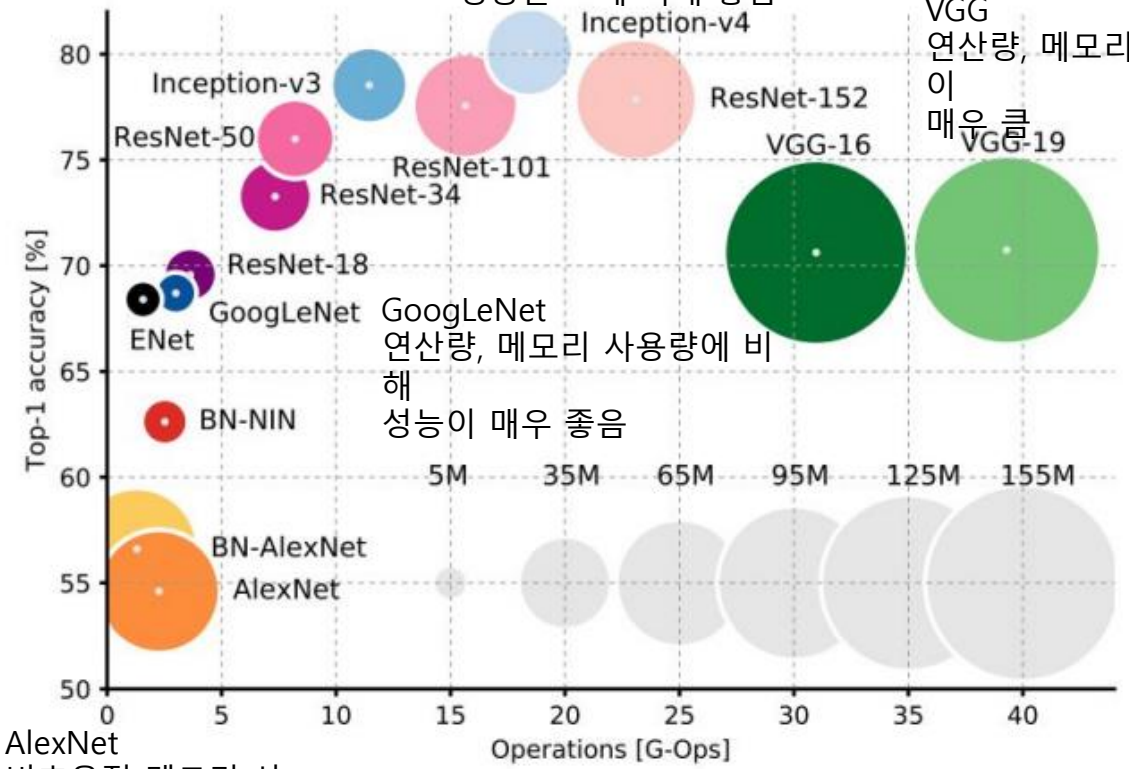
Compare complexity

Inception V4
: ResNet+ Inception
높은 정확도



ResNet
적당한 연산량, 메모리 사용량
성능은 그에 비해 좋음

VGG
연산량, 메모리 사용량이
매우 큼



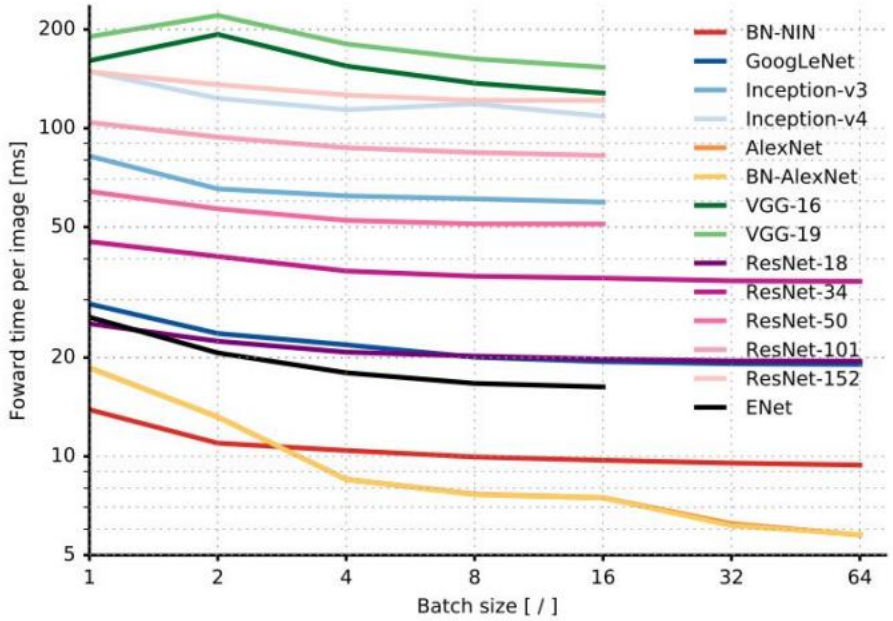
GoogLeNet
연산량, 메모리 사용량에 비해
성능이 매우 좋음

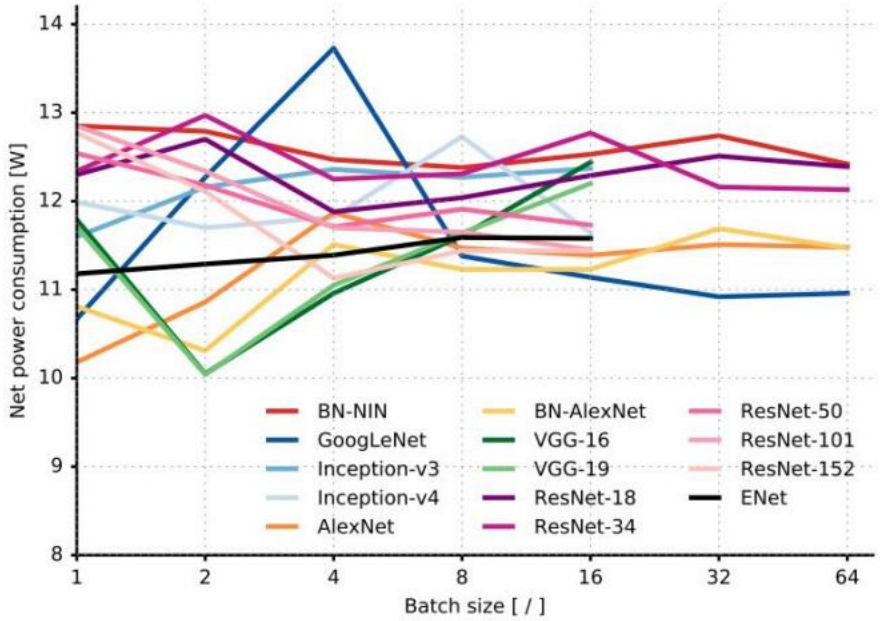
AlexNet
비효율적 메모리 사용

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

Compare complexity

VGG
계산 시간 오래걸림

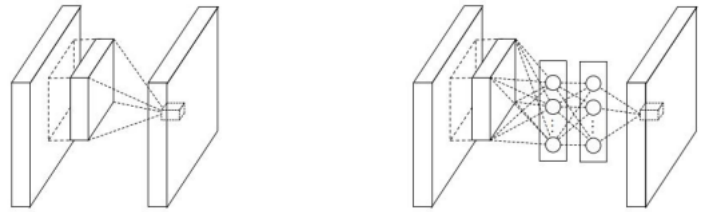




5. Another Net

✓ Network in Network (NiN)

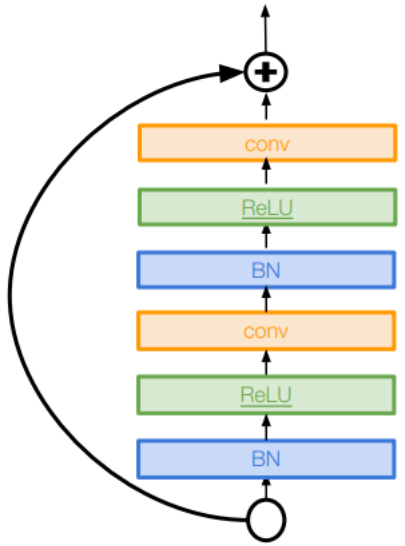
-Mipconv layer로 구성된 Micronetwork
-Multilayer Perceptron을 이용
-GoogLeNet보다 먼저 Bottleneck 아이디어 제시



(a) Linear convolution layer (b) Mlpconv layer

✓ Identity Mapping in Deep Residual Network

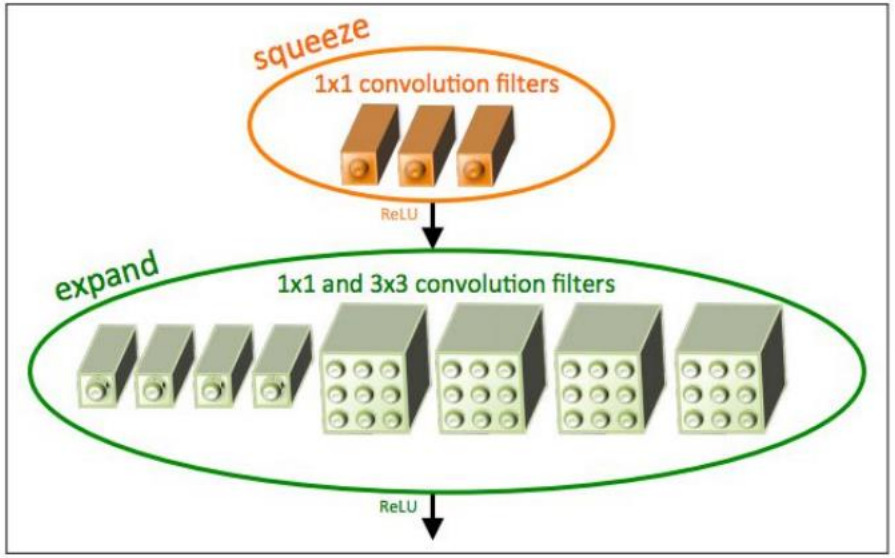
-ResNet Block Design 보완
-ResNet의 Block Path를 조절하여 Direct path를 증가시킴
-정보 전달/Backprop 성능 향상



5. Another Net

✓ SqueezeNet

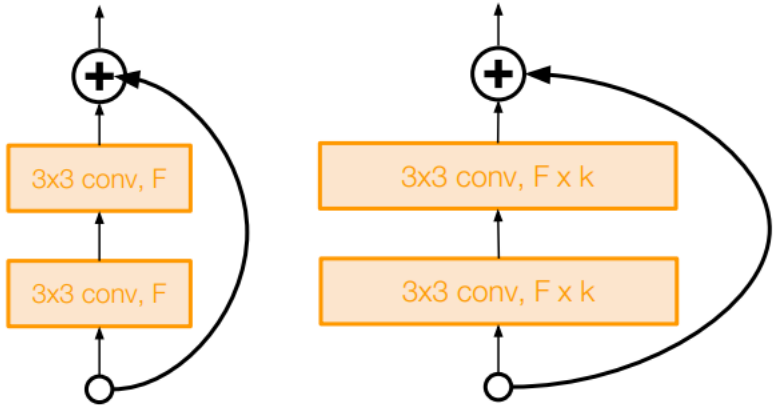
-Squeeze layer에서 시작해 expand layer로 전달
-AlexNet과 유사한 정확도, 50배 적은 parameter, 510배 작은 size



5. Another Net

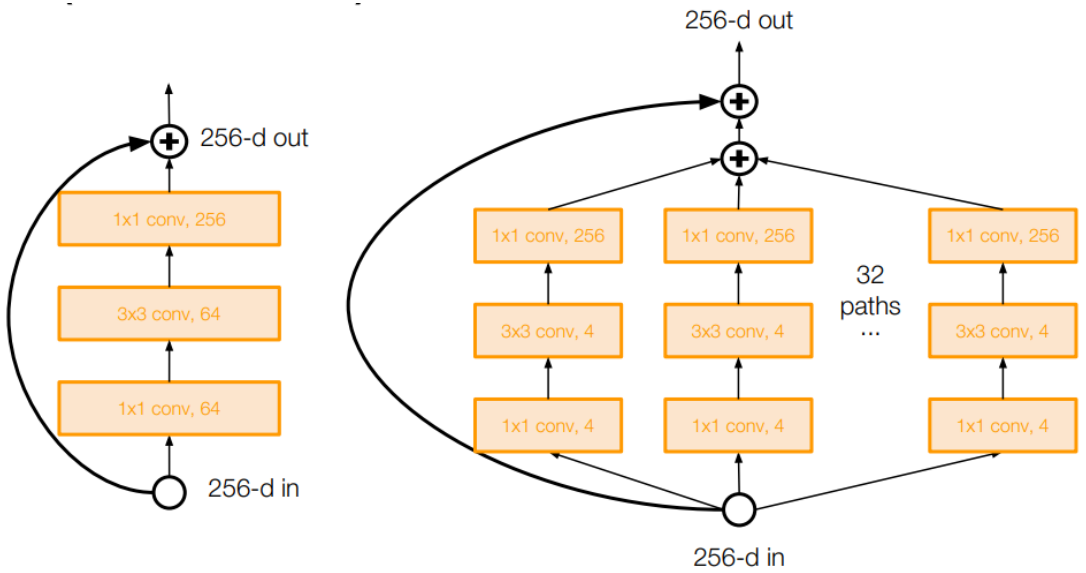
✓ Wide Residual Network <improve>

-Residual이 Depth보다 중요하다는 관점
-Residual Block 크기 확장
(Filter 개수 ↑)
-기존 152개 Layer보다 50 Layer의 Wide Residual Network 성능이 좋음을 확인 (계산 효율의 증가)



✓ ResNeXt <improve>

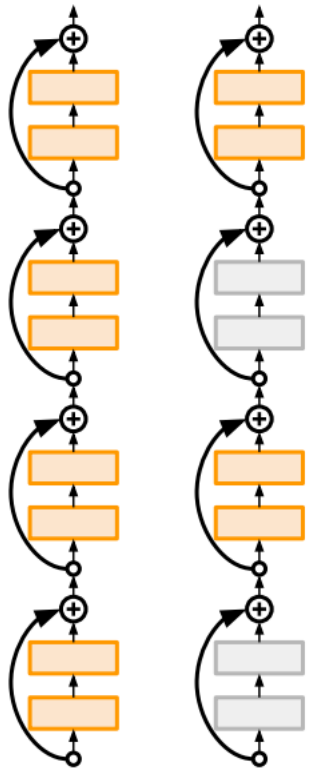
-ResNet에 multiple parallel pathways 추가
-inception Module 방식과 유사



5. Another Net

- **Deep Networks with Stochastic Depth** <improve>

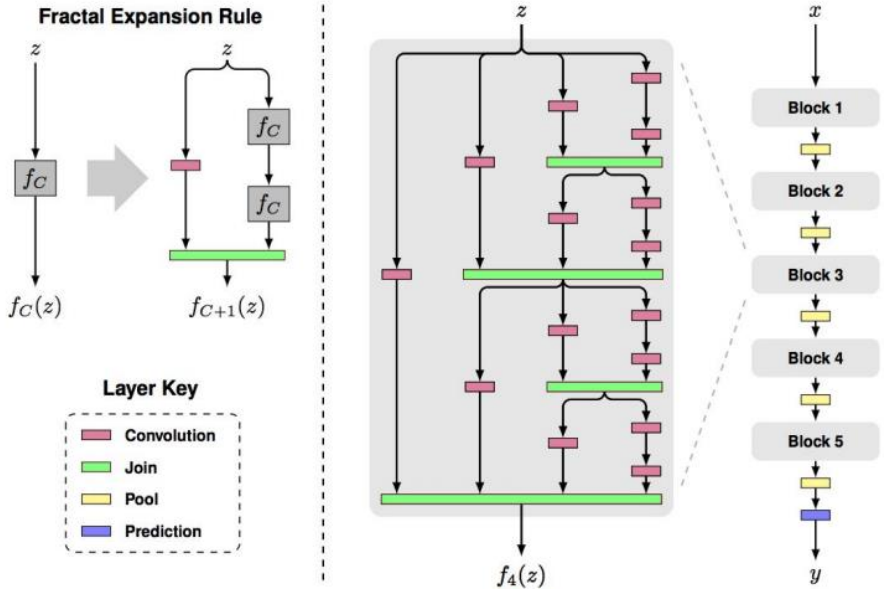
-Network가 깊어질수록 gradient가 사라지는 문제
-Train time의Network 일부를 identity connection 으로 변환
-Dropout과 유사



5. Another Net

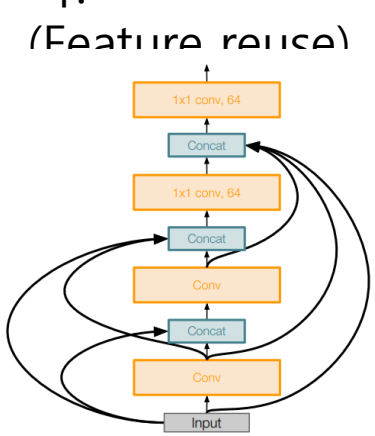
✓ FractalNet <Beyond>

-Residual connection과 shallow-deep 정보전달이 무의미하다는 관점
-shallow, deep 정보 모두 원활하게 전달하고자 함
-프랙탈 형태의 network

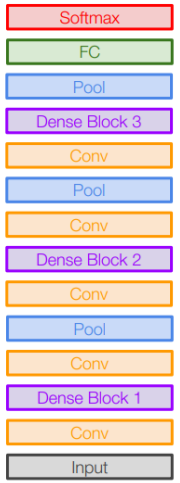


✓ Densely Connected Convolutional Network <beyond>

-Dense block으로 각 layer를 다른 모든 layer에 연결
-Network의 입력 이미지=모든 Network 입력+아래 network 출력
-Feature를 모두 전달되므로 잘 사용될 수 있다.
(Feature reuse)



Dense Block



이화여자대학교
EWHA WOMANS UNIVERSITY

감사합니다.

EWHA,
THE FUTURE
WE CREATE