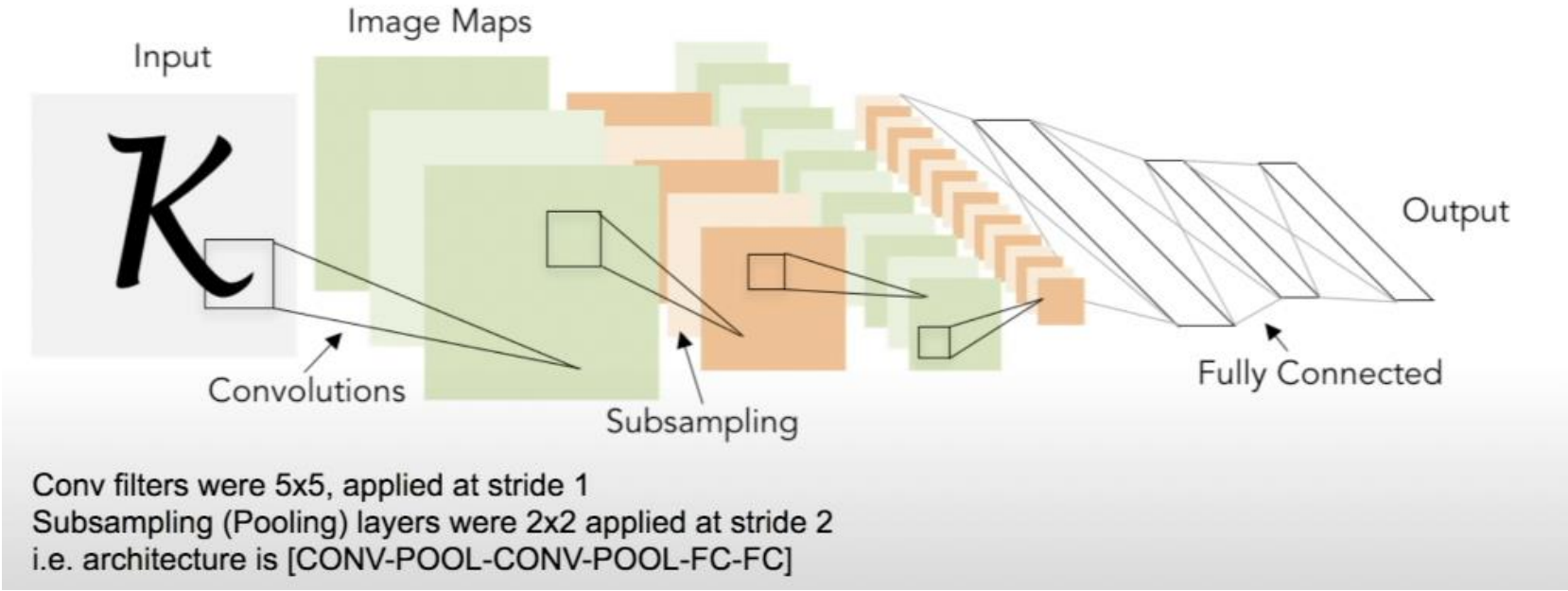
9주차
CNN Architectures

1기 박가현
1기 이선민

EURON

목차
(선택)

0. LeNet



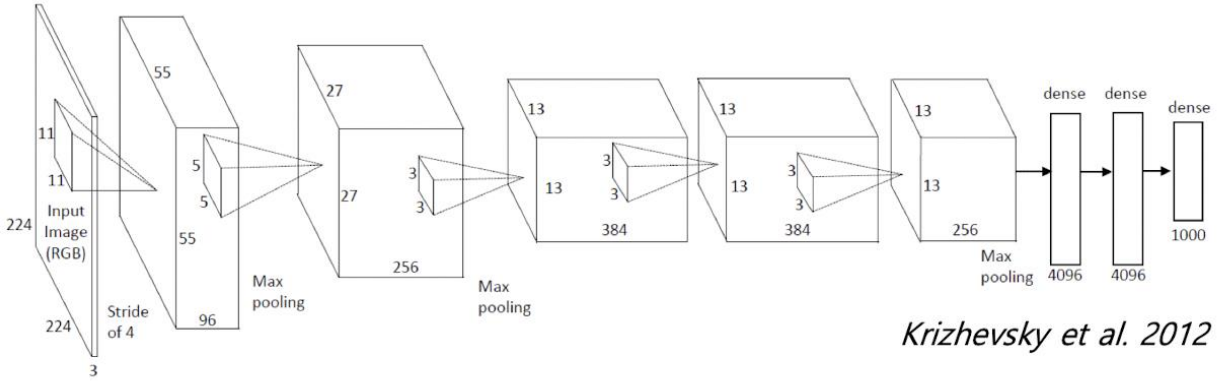
Digit recognition에서 좋은 성능을 보임

1. AlexNet

- 2012년에 발표
- ImageNet Classification에서 최초로 좋은 성능을 보인 CNN 모델
- Non-deep learning 모델을 월등히 뛰어넘는 성능

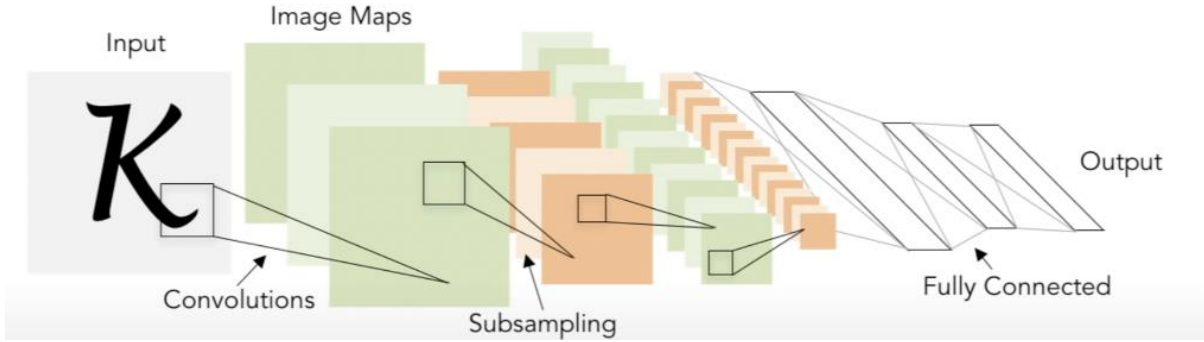
1. AlexNet

CONV1
MAXPOOL1
NORM1
CONV2
MAXPOOL2
NORM2
CONV3
CONV4
CONV5
MAXPOOL3
FC6
FC7
FC8



Krizhevsky et al. 2012

(위) AlexNet 구조 (아래) LeNet 구조



1. AlexNet

Convolution layer의 output tensor size

- 각각 기호를 아래와 같이 정의
  - O: Size(width) of output image
  - I: Size(width) of input image
  - K: Size(width) of kernels used in the Conv layer
  - N: Number of kernels
  - S: Stride of the convolution operation
  - P: Padding size
- O(Size(width) of output image)는 다음과 같이 정의 됨

$$O = \frac{I - K + 2P}{S} + 1$$

- 출력 이미지의 채널 수는 커널의 갯수(N)와 같음

1. AlexNet

$$O = \frac{I - K + 2P}{S} + 1$$

Input : 227 x 227 x 3 이미지
Conv1: 96 11x11 (stride 4)
Output size?

** Width / Height
(Input Width – Filter width) / stride +1
227 - 11 / 4 +1 = 55

** Depth
Filter의 개수 :96

=> 최종 output : 55 x 55 x 96

1. AlexNet

Conv1 의 파라미터 개수?

Input depth : 3

-> Filter는 11 x 11 x 3 씩 봄

->Filter 개수 96개

∴ (11 * 11 * 3) * 96 개 = 35K

1. AlexNet

Conv1의 output : 96 55x55
Pool1: 3x3 filters (stride 2)
Output size?

** Width / Height
(Input Width – Filter width) / stride +1
55 - 3 / 2 +1 = 27

** Depth
Pooling 층에서 depth 는 동일하게 유지

output : 27 x 27 x 96

1. AlexNet

Pool1 의 파라미터 개수?

0

Why?

Parameter란, 학습하려고 하는 weights
Pooling 층은 각 구역의 max를 뽑아오는 “규칙”

1. AlexNet

$$O = \frac{I - K + 2P}{S} + 1$$

Input: 227x227x3

	Size (Stride/Padding)	W	H	C	# of Parameter
Conv1	11x11x96 (S:4 / P:0)	(227- 11+ 0)/4 +1 = 55	(227- 11+ 0)/4 +1 = 55	96	11x11x3x96
MaxPool1	3x3 (S:2)	(55- 3+ 0)/2 +1 =27	(55- 3+ 0)/2 +1 =27	96	0
Conv2	5x5x256 (S:1 / P:2)				
MaxPool2	3x3 (S:2)				
Conv3	3x3x384 (S:1 / P:1)				
Conv4	3x3x384 (S:1 / P:1)				
Conv5	3x3x256 (S:1 / P:1)				
MaxPool3	3x3 (S:2)				

1. AlexNet

$$O = \frac{I - K + 2P}{S} + 1$$

Input: 227x227x3

	Size (Stride/Padding)	W	H	C	# of Parameter
Conv1	11x11x96 (S:4 / P:0)	(227- 11+ 0)/4 +1 = 55	(227- 11+ 0)/4 +1 = 55	96	11x11x3x96 =34,848
MaxPool1	3x3 (S:2)	(55- 3+ 0)/2 +1 =27	(55- 3+ 0)/2 +1 =27	96	0
Conv2	5x5x256 (S:1 / P:2)	(27- 5+ 2*2)/1+1 =27	(27- 5+ 2*2)/1 +1 =27	256	5x5x96x256 =614,400
MaxPool2	3x3 (S:2)	(27- 3)/2+1 =13	(27- 3)/2+1 =13	256	0
Conv3	3x3x384 (S:1 / P:1)	(13- 3+2)/1+1 =13	(13- 3+2)/1+1 =13	384	3x3x256x384 =86,400
Conv4	3x3x384 (S:1 / P:1)	(13- 3+2)/1+1 =13	(13- 3+2)/1+1 =13	384	3x3x384x384 =1,327,104
Conv5	3x3x256 (S:1 / P:1)	(13- 3+2)/1+1 =13	(13- 3+2)/1+1 =13	256	3x3x384x256 =86,400
MaxPool3	3x3 (S:2)	(13- 3)/2+1 =6	(13- 3)/2+1 =6	256	0

1. 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)

논문 상에서 input 224로 되어있으나 계산해보려면 패딩등으로 고려해서 227로 계산해야 맞음

https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

1. AlexNet

- 처음으로 ReLu 사용
- 이웃한 Channel 끼리 Normalization 적용
  -> 영향이 미미해서 사용 하지 않음
- Data augmentation을 많이 함
- Dropout 0.5
- Batch size 128
- SGD momentum 0.9
- Base Learning Rate : 1e-2
  Val accuracy 가 일정 상태를 유지하면 1/10 시킴

1. AlexNet

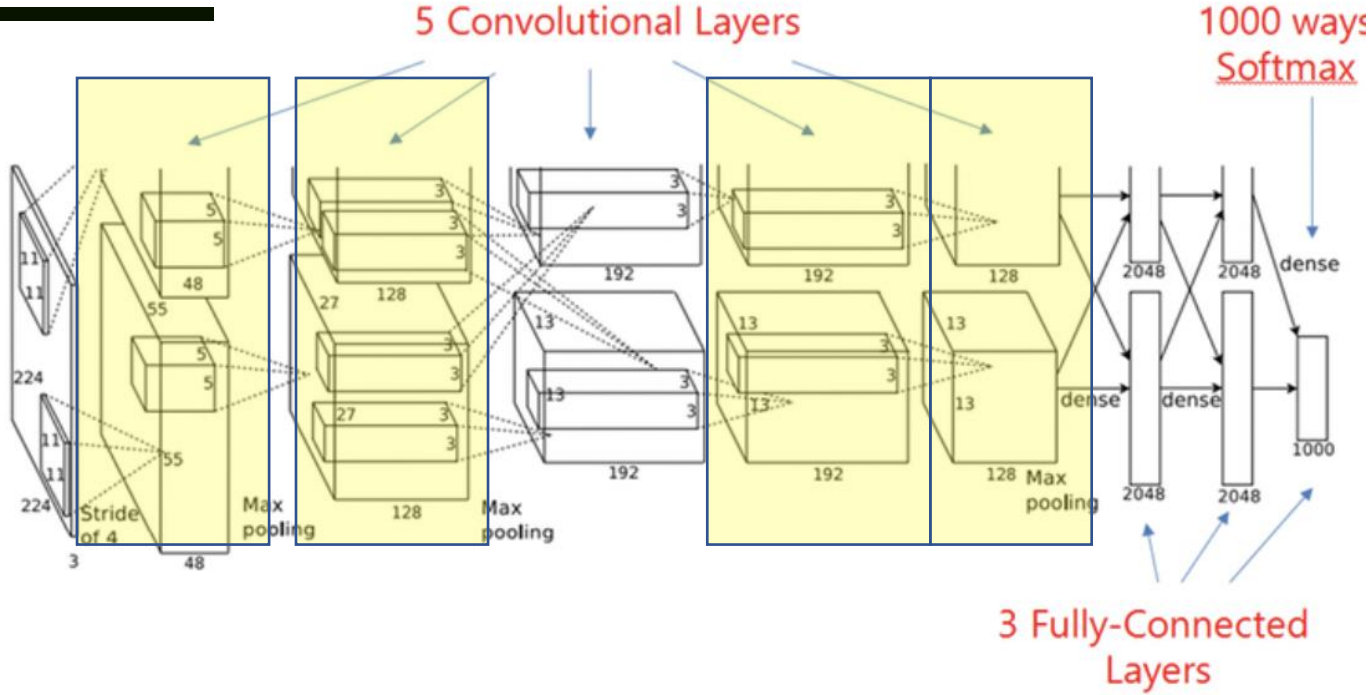
- L2 Weight decay : 5e-4

  #Weight decay?

  overfitting을 방지하기 위한 방법 중 하나.
  Loss function을 줄이기 위한 방향으로 단순 학습되는 것 방지
  모델의 weight가 너무 큰 값을 갖지 않도록 패널티 항목 추가
  L1 Regularization과 L2 Regularizationdl 대표적인 항목

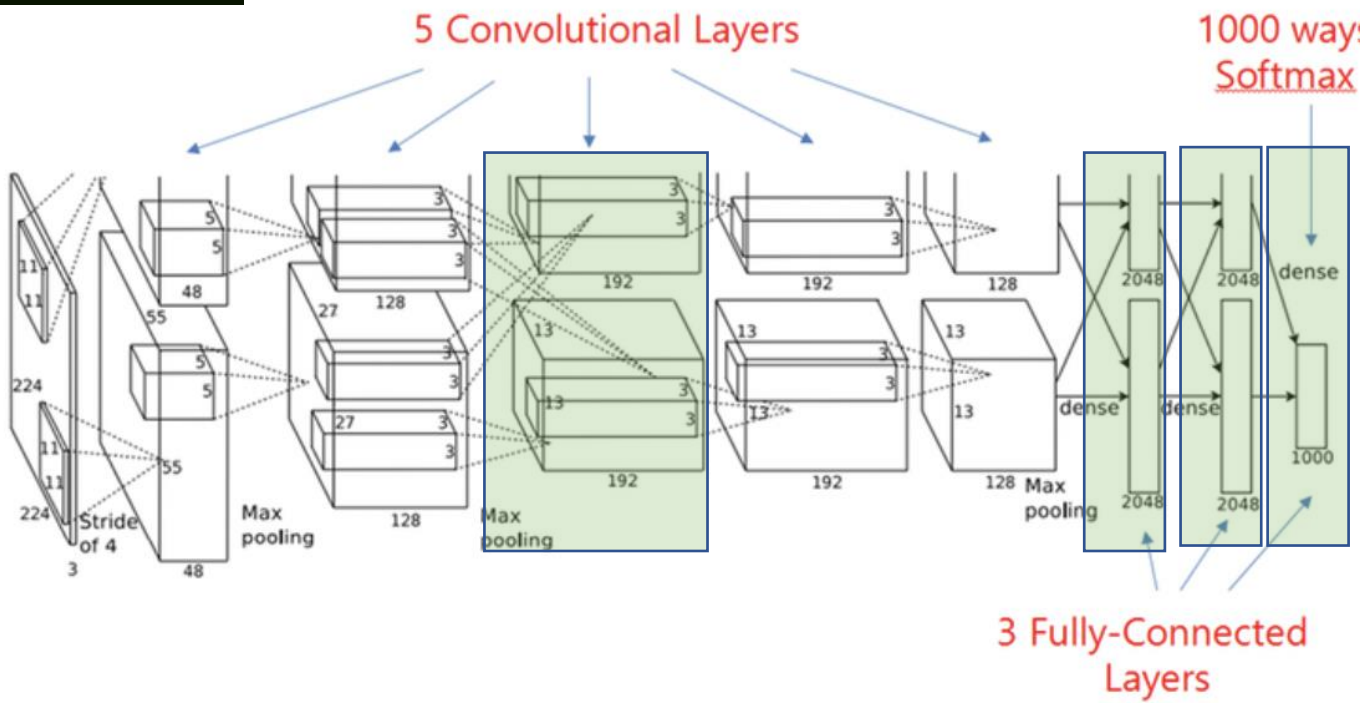
- 7 CNN Model Ensemble을 통해 성능 향상

1. AlexNet



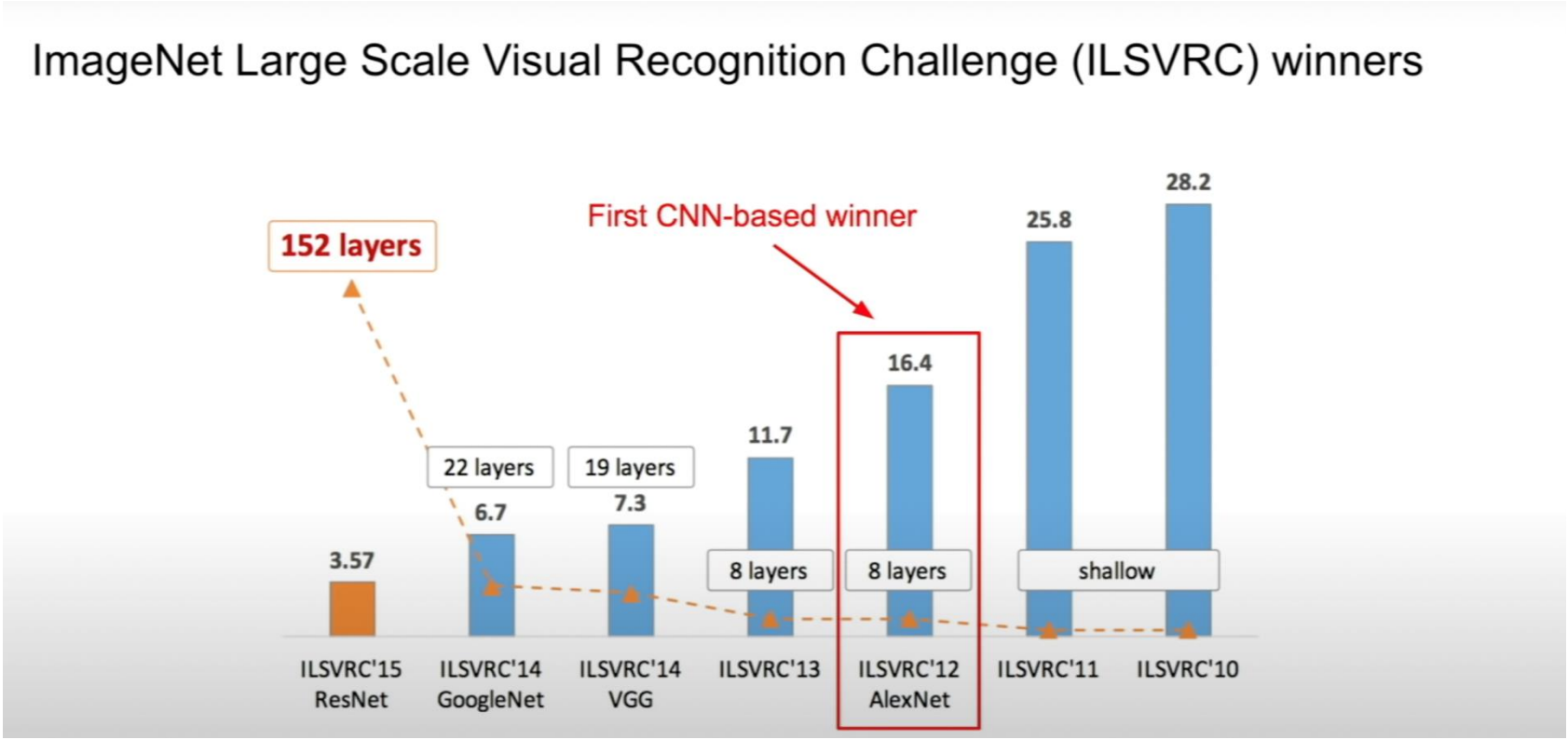
CONV1, CONV2, CONV4, CONV5
같은 GPU상의 feature map 만 사용

1. AlexNet



CONV3, FC6, FC7, FC8
이전 Layer feature map과 연결. 다른 GPU와도 communicate

1. AlexNet



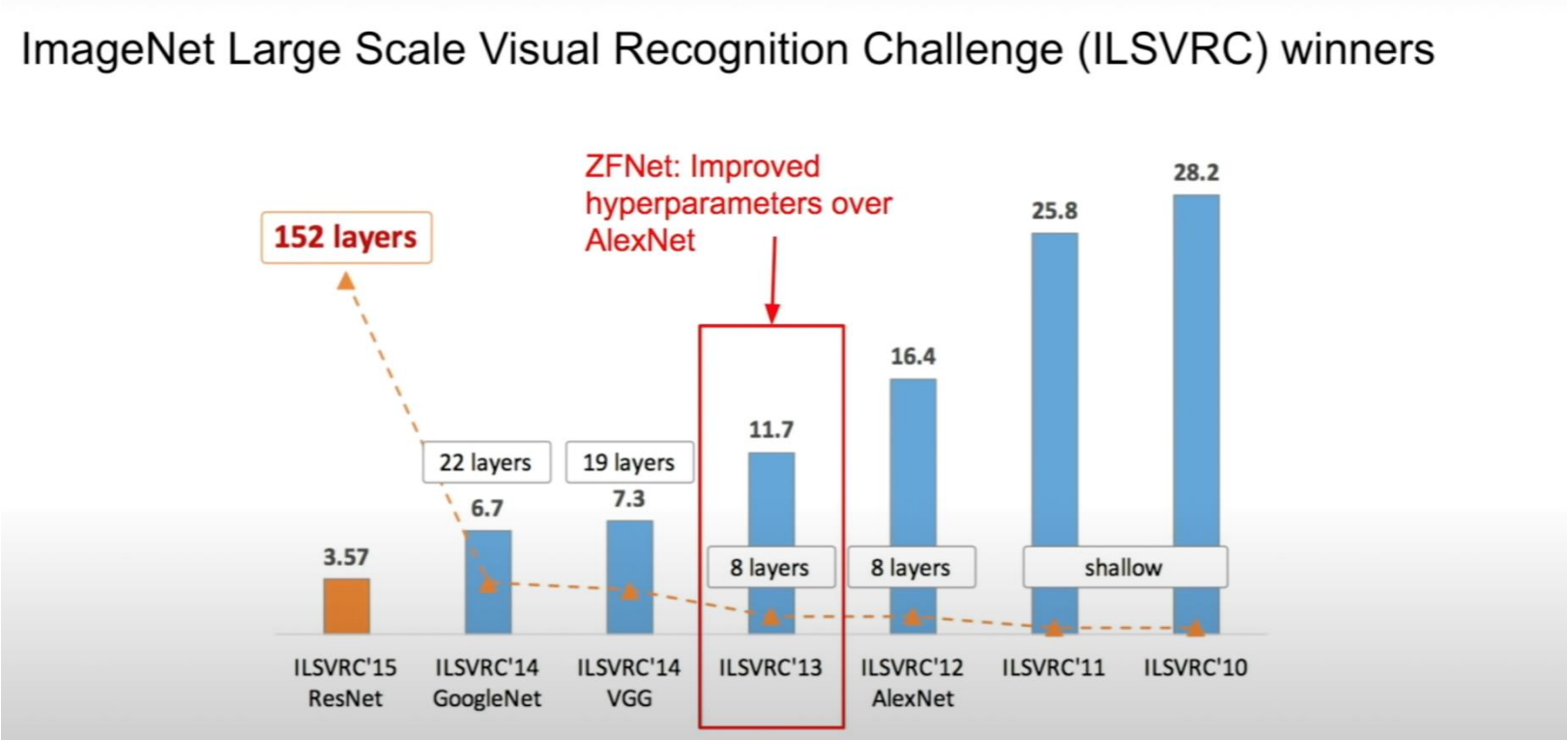
최초의 CNN 기반 우승. 최근엔 많이 쓰이지 않음

1. AlexNet

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	—	—	*26.2%*
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	**15.3%**

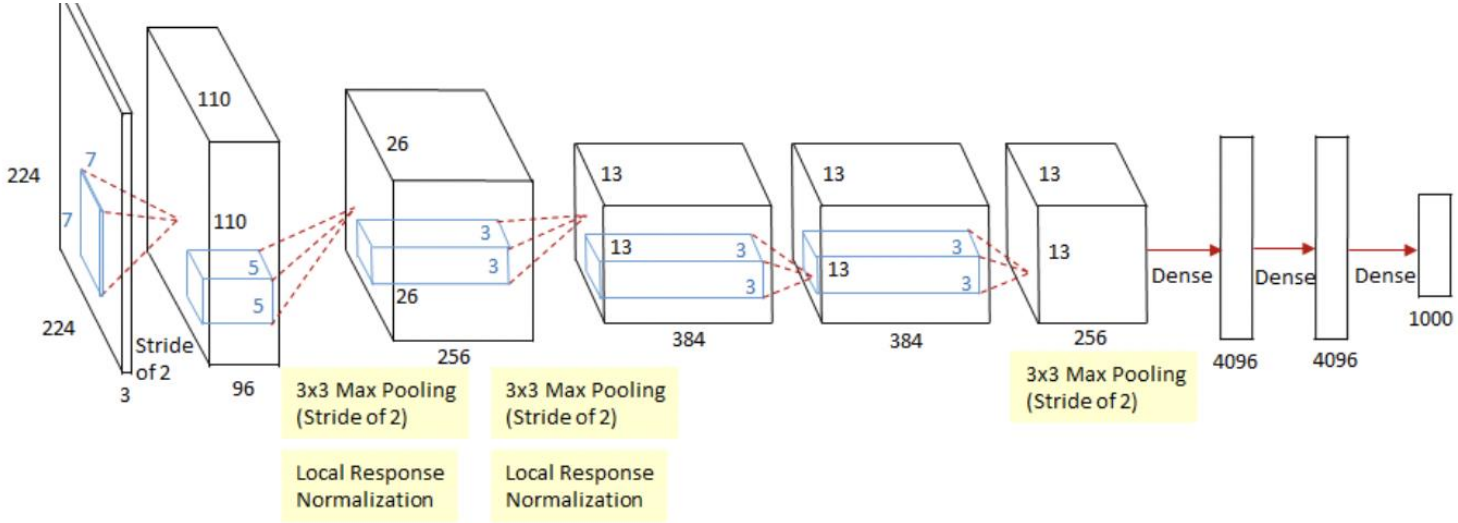
The CNN described in this paper : a top-5 error rate of 18.2%.
Averaging the predictions of five similar CNNs : error rate of 16.4%
1 CNN with an extra sixth convolutional layer over the last pooling layer +"fine-tuning" : error rate of 16.6%.
Averaging the predictions of two CNNs that were pre-trained on the entire Fall 2011 re- lease with the aforementioned five CNNs : error rate of 15.3%.

1. AlexNet


ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners
ZFNet: Improved hyperparameters over AlexNet
152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15 ResNet
ILSVRC'14 GoogleNet
ILSVRC'14 VGG
ILSVRC'13
ILSVRC'12 AlexNet
ILSVRC'11
ILSVRC'10

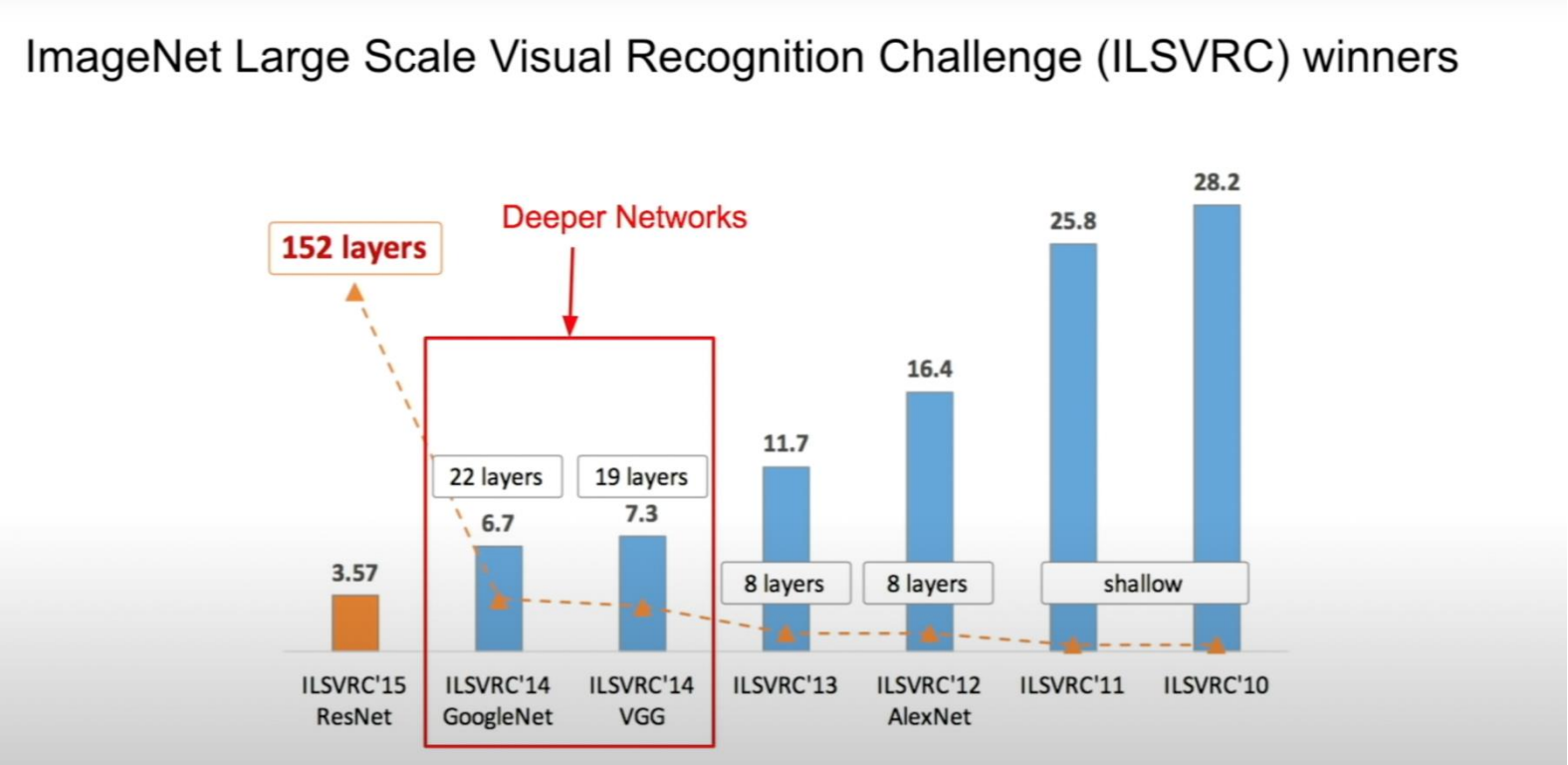
1-2. ZFNet

- 같은 구조, 같은 Layer 개수
- Conv1 의 stride, Conv 3,4,5의 filter 사이즈 변경
- Error Rate : 16.4 -> 11.7%



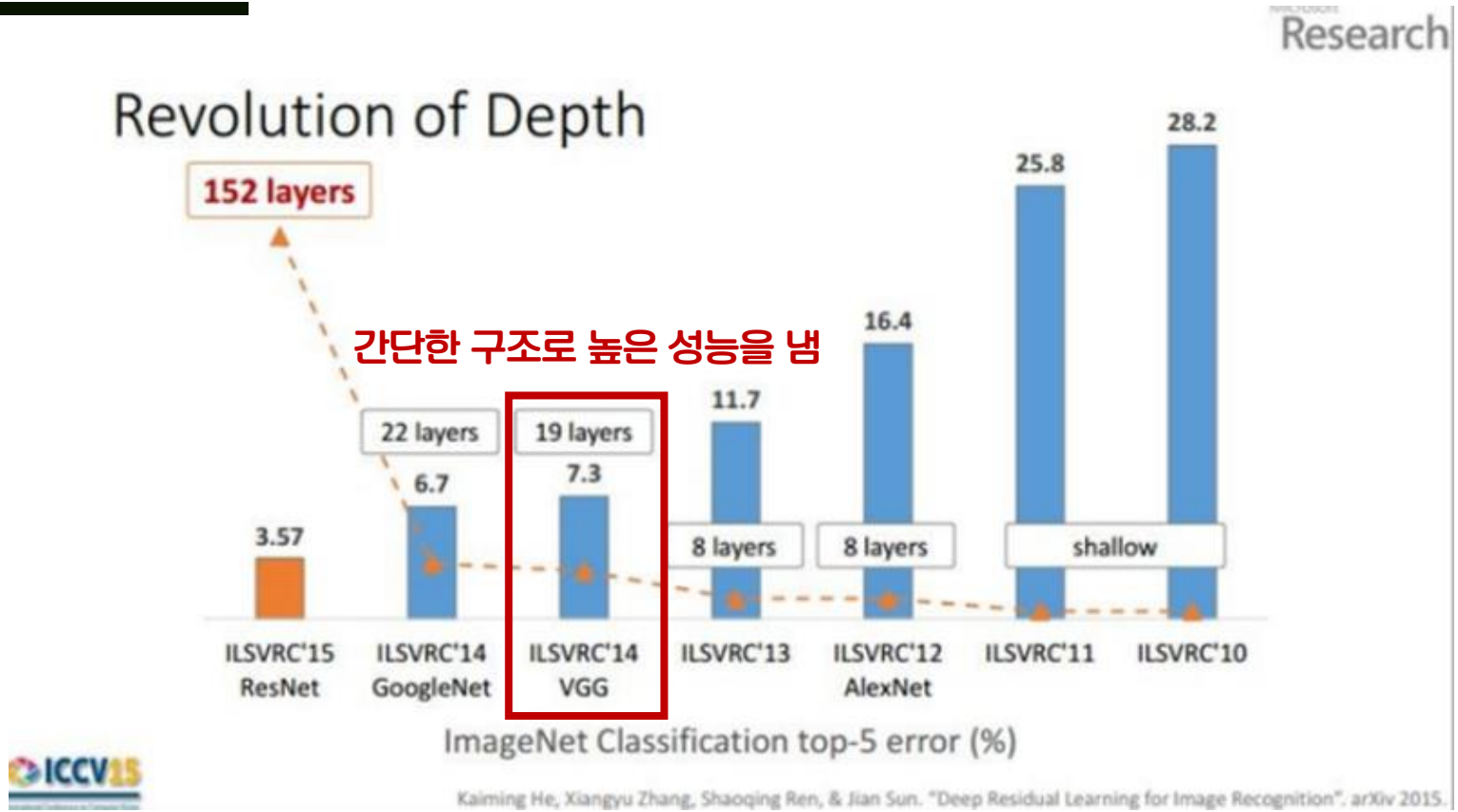
ZFNet

1-2.ZFnet



이후 우승 모델들은 훨씬 DEEP 함

2. VGGNet



신경망의 깊이가 딥러닝의 정확도에 큰 영향을 미침

2-1. VGGNet 연구

연구의 핵심

네트워크의 깊이를 깊게 만드는 것이 성능에 어떤 영향을 미치는 확인

↓

Convolutional Fillter size = 3x3

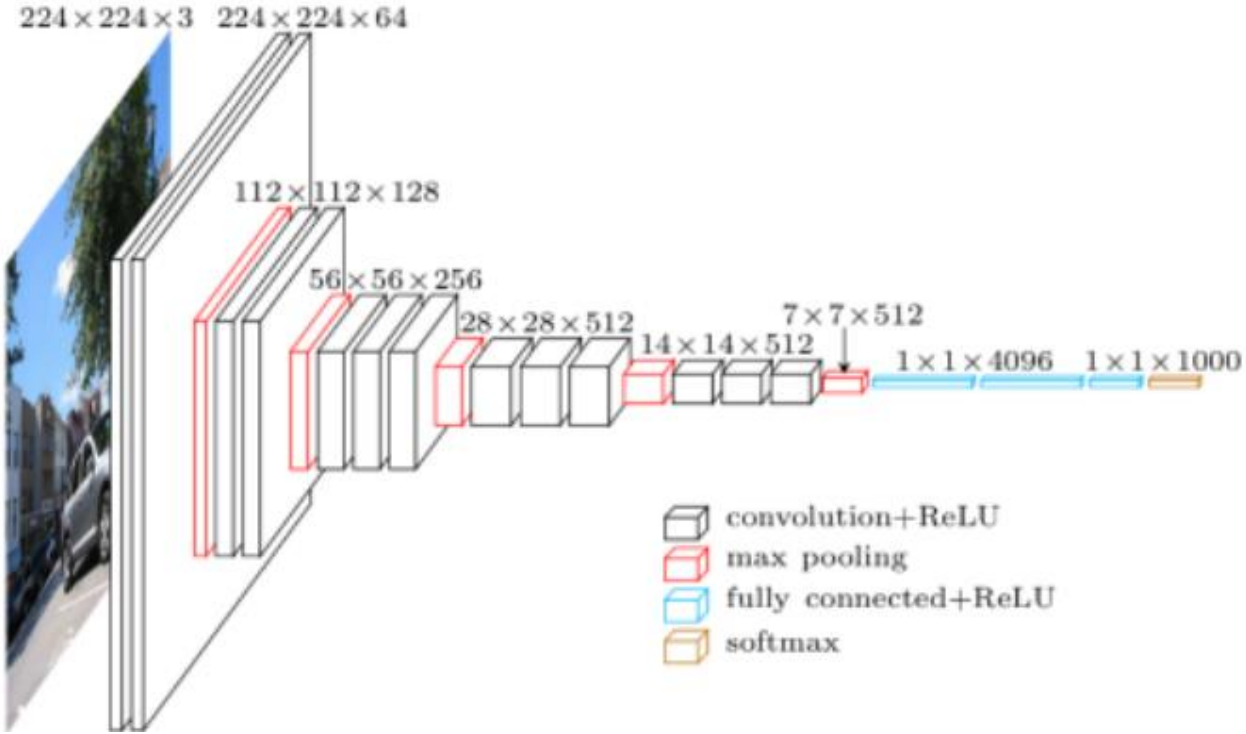
↓

네트워크의 깊이를 깊게 만듬

↓

성능이 좋아짐

2-2. VGGNet의 구조



- 16 ~ 19 Layer
- 8 ~ 16 Convolutional layer + 3 Fully-Connected Layer

2-2. VGGNet의 구조

Convolutional Layer

- 3x3 filter, stride = 1, padding = True

- 의사결정함수에 Non-linearity를 부여할 목적

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

2-2. VGGNet의 구조

Max-Pooling Layer

- 2x2 filter, stride = 2
- Conv Layer 수와 관계없이 5장 사용
- 특정 맵을 $\frac{1}{4}$로 줄여줌

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

2-2. VGGNet의 구조

Fully-Connected Layer

- 3개의 layer (4096 -> 4096 -> 1000)
- 파라미터가 많아짐
- 많은 메모리를 사용하여 연산

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

2-2. VGGNet의 구조

A < B < C,D < E

학습해야할 파라미터 수 줄어듬

= 네트워크의 깊이가 깊어짐

= 좋은 성능

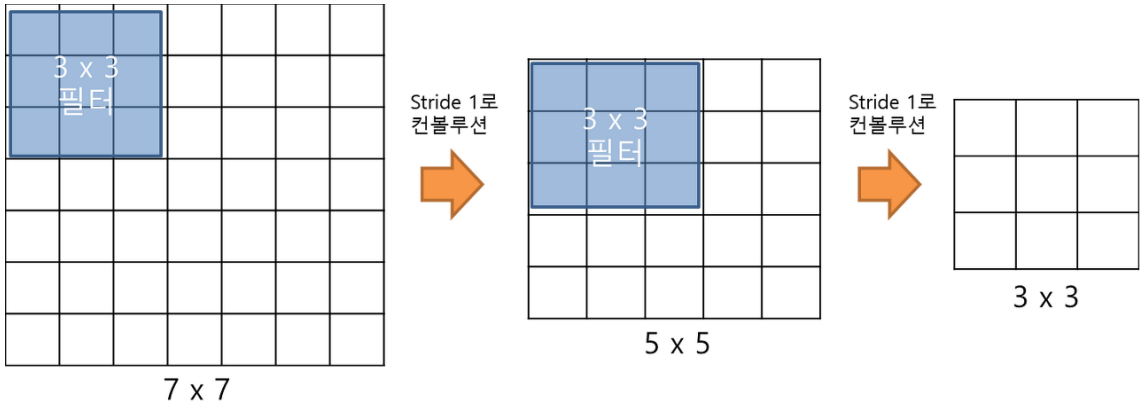
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

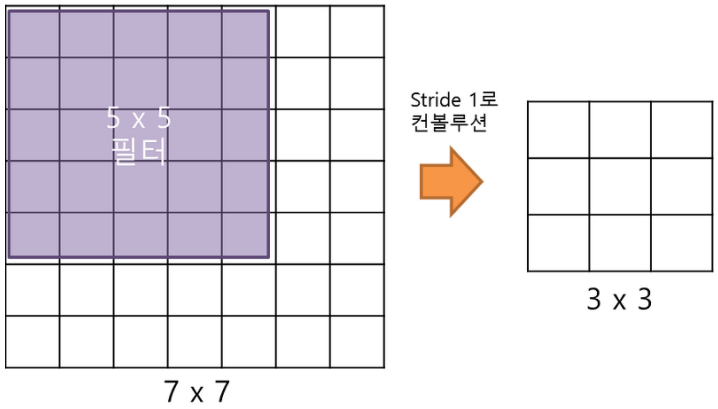
Table 2: **Number of parameters** (in millions).

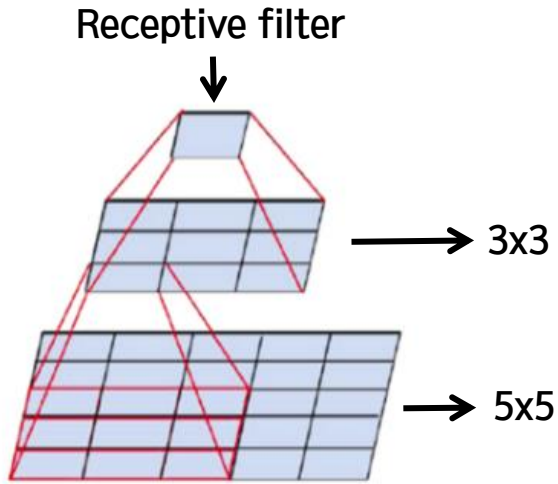
Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

2-3. VGGNet의 특징

3x3 Filters 를 사용하는 이유?







7x7 Filters 1번 > 3x3 Filters 3번

파라미터의 개수가 줄어듬

-> ReLu 활성화 함수가 들어갈 수 있는 공간이 많아짐

-> 망이 깊어짐

-> 더 많은 non-linearities 적용 가능

2-4. VGG16와 VGG19

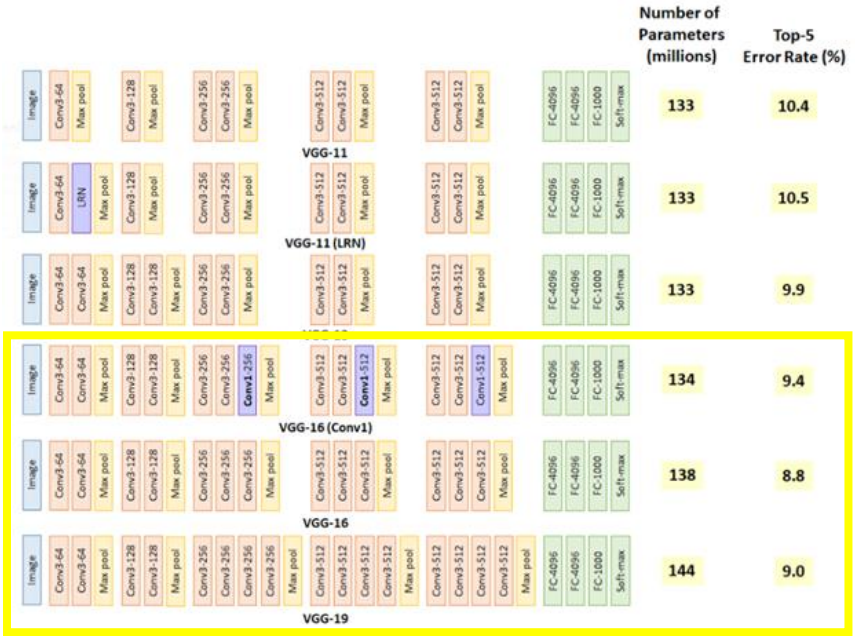
6개의 구조 개발 = 깊이에 따른 성능 변화 비교

D : VGG16

E : VGG19

- Layer 깊이 : VGG16 < VGG19
- 소모하는 메모리 수 : VGG16 < VGG19

=> VGG16 더 많이 사용

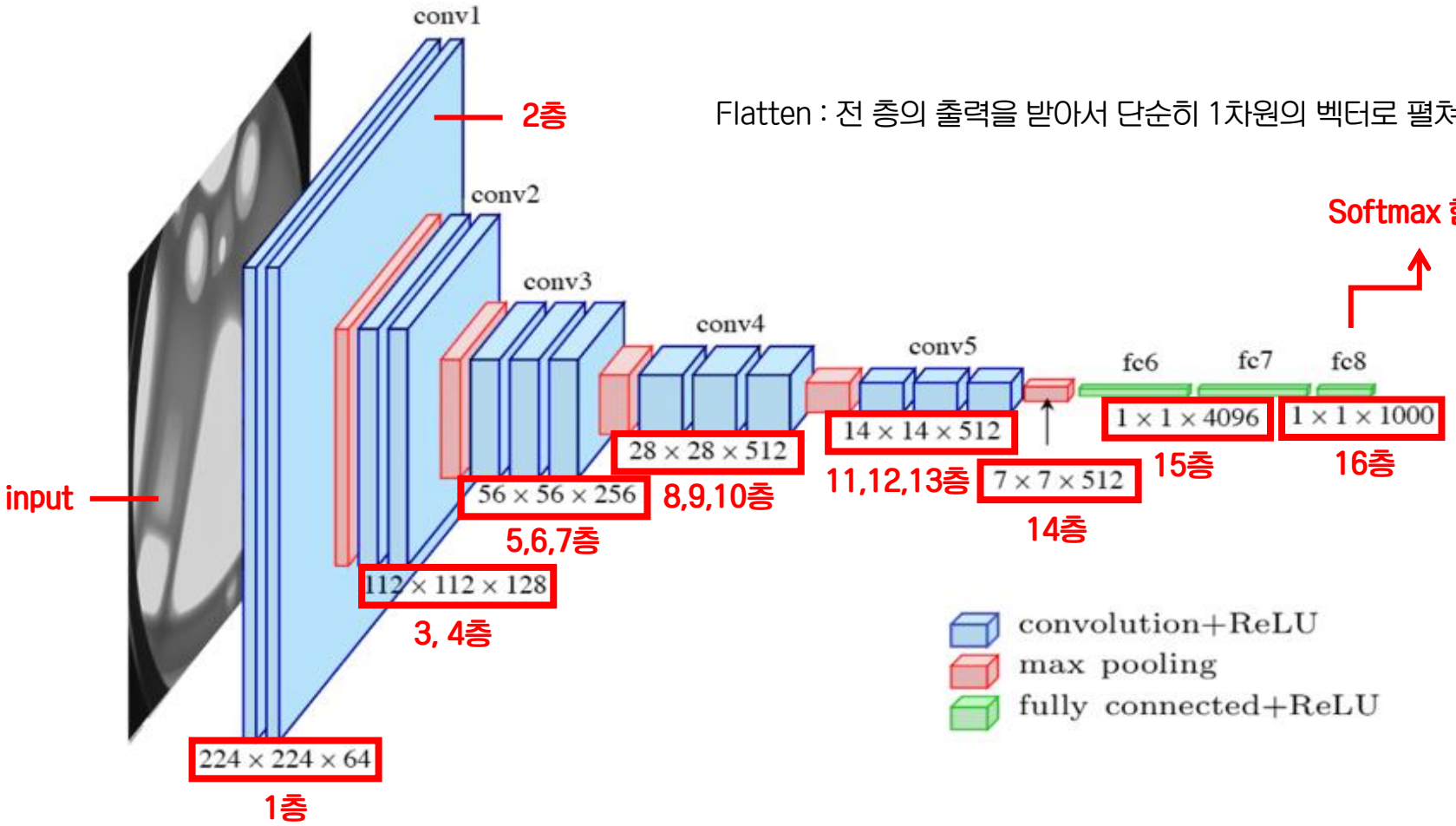


ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

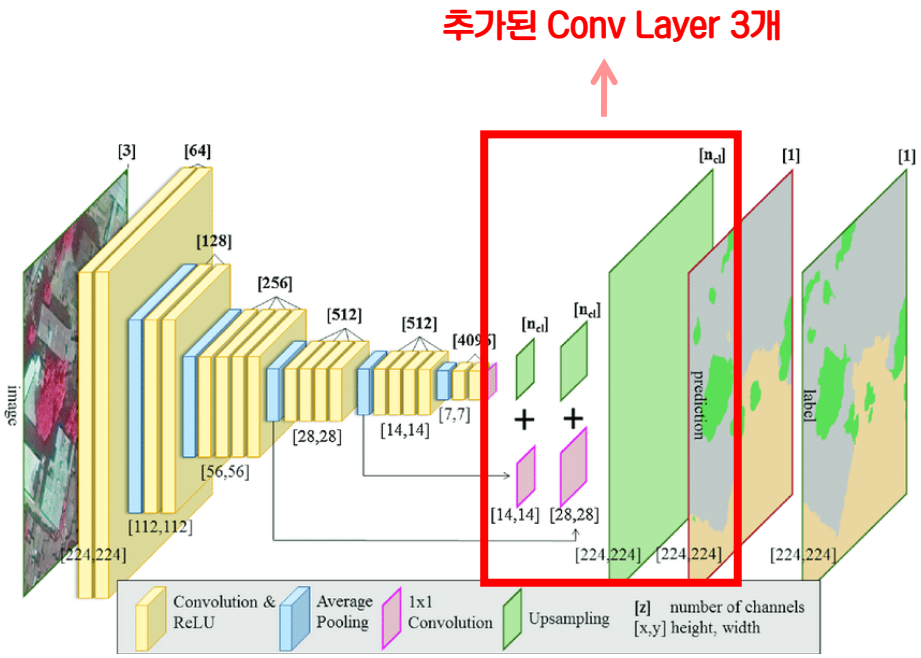
Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

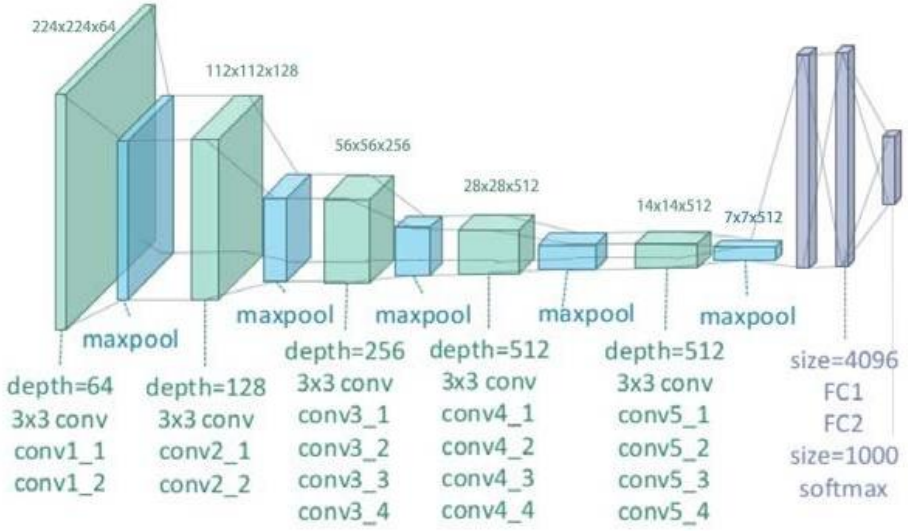
2-4.1. VGG16



Flatten : 전 층의 출력을 받아서 단순히 1차원의 벡터로 펼쳐주는 것

2-4.2. VGG19





- VGG16과 구조가 비슷함
- Layer 수 많음 = 파라미터 수 많음 = 메모리 많이 씀 = 성능 떨어짐
- Exploding Gradient / Vanishing 문제로 인한 깊은 레이어 학습 때 발생하는 문제 해결

2-5. VGGNet 요약

VGGNet

· 3x3의 작은 필터를 깊게 쌓음

-> 동일 효과 유지 = 파라미터수 줄어듬 = 많은 ReLU 함수 사용

-> 많은 non-linearity 사용 가능 = representation 능력 향상

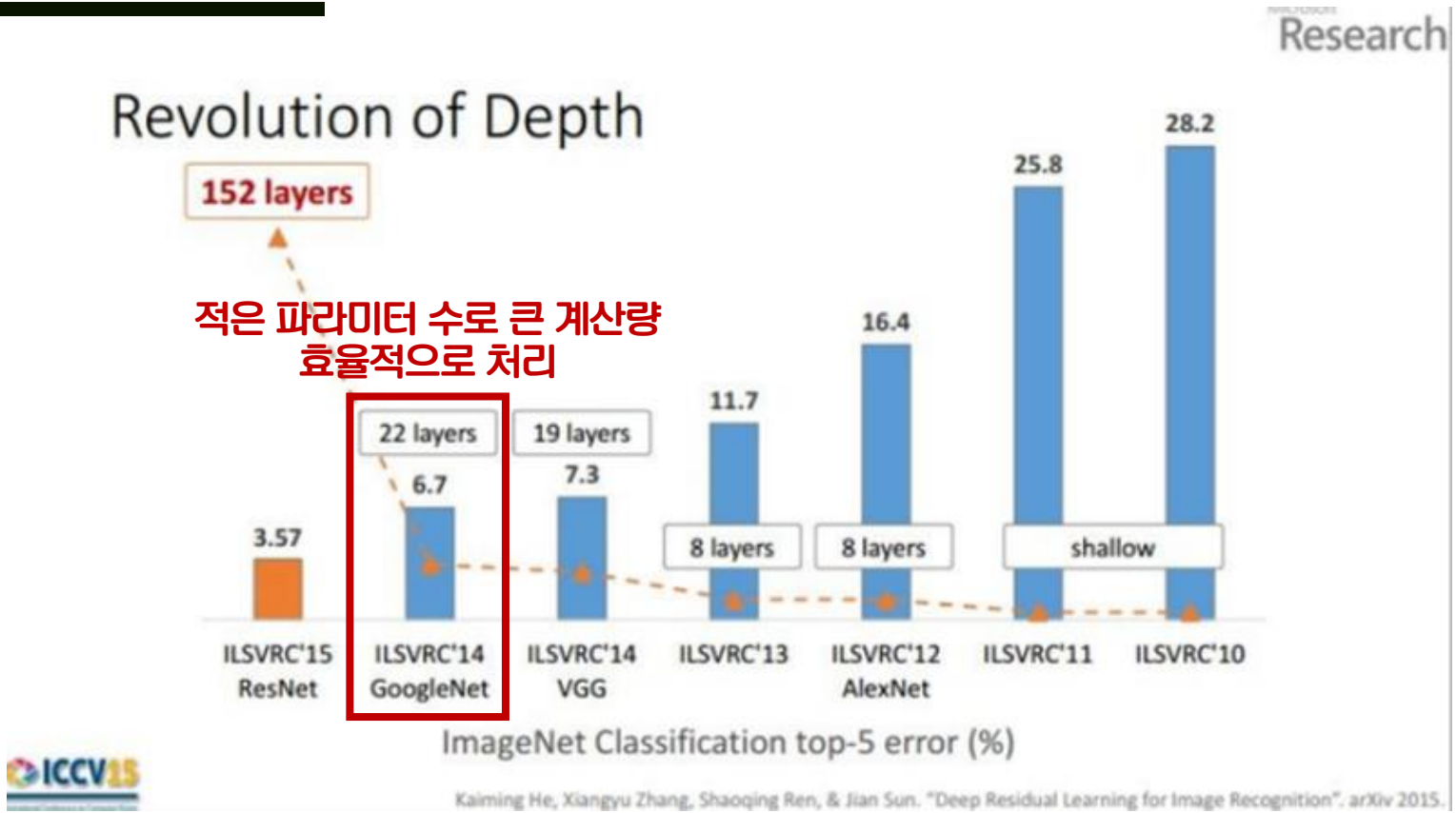
· FC7 Layer 부근이 잘 generalize 되어있음

-> 좋은 feature representation을 가짐 = feature 추출 잘됨

-> 일반화 능력이 뛰어남

· AlexNet보다 좋은 성능을 가짐

3. GoogLeNet



신경망의 깊이가 딥러닝의 정확도에 큰 영향을 미침

3-1. GoogLeNet 연구

망이 깊어지면 생기는 문제점 2가지

· 자유 파라미터의 수 증가 (overfitting에 빠질 가능성 높아짐)

· 연산량 늘어남 (학습 시간 늘어남)

↓

GoogLeNet 문제 해결 방법

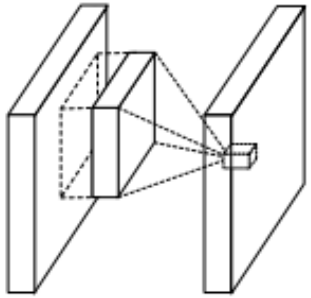
↓

신경망 깊게 유지하여 성능을 올리면서 파라미터 수 줄임

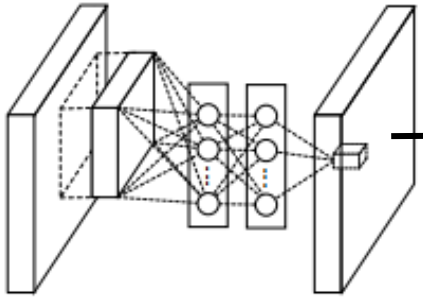
↓

효율성을 높일 수 있는 architecture 제안

3-2. Network in Network(NIN) 연구

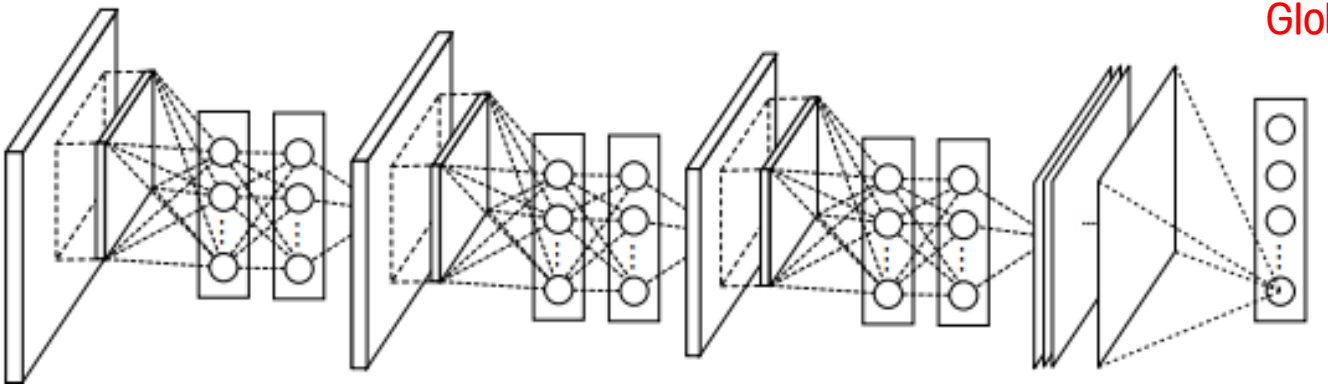


(a) Linear convolution layer



(b) Mlpconv layer

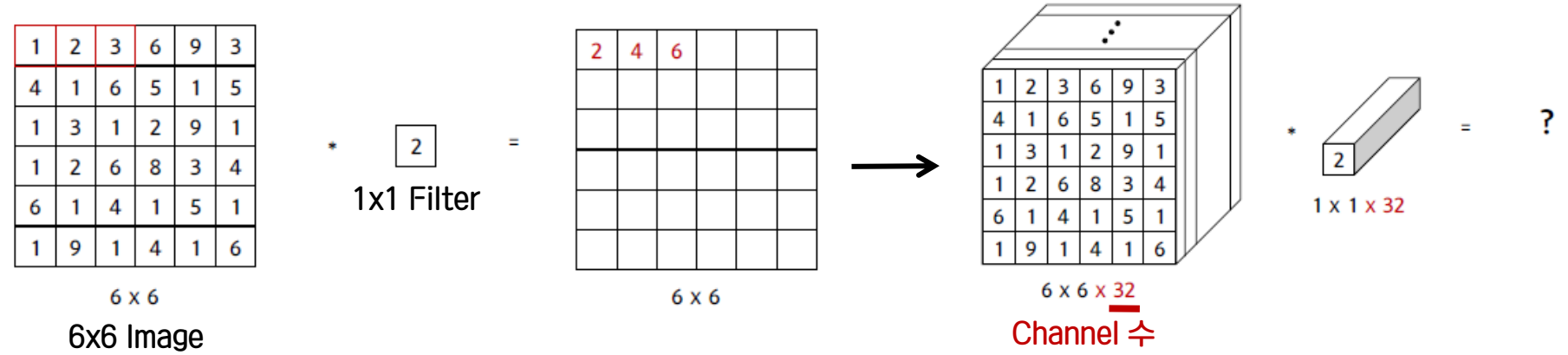
→ 비선형적인 특징들 추출

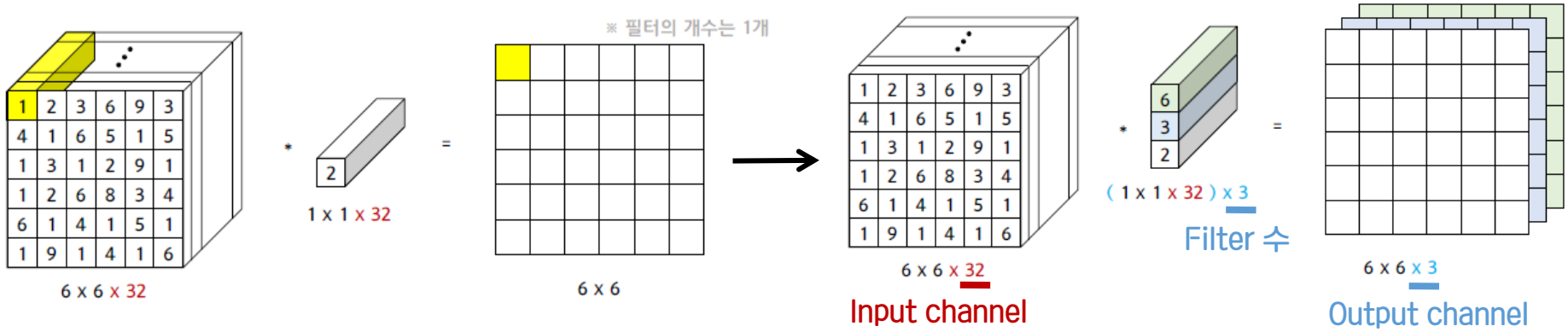


Global average pooling 적용

- 일반 filter보다 데이터의 non-linear한 성질을 잘 표현
- 1x1 convolution을 이용하여 feature map의 크기를 줄일 수 있음

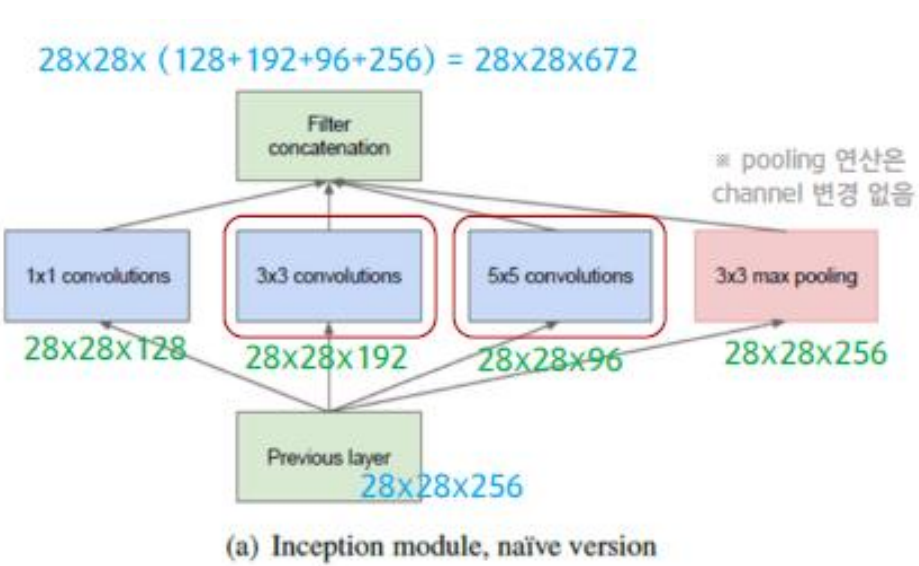
3-3. 1x1 Convolution



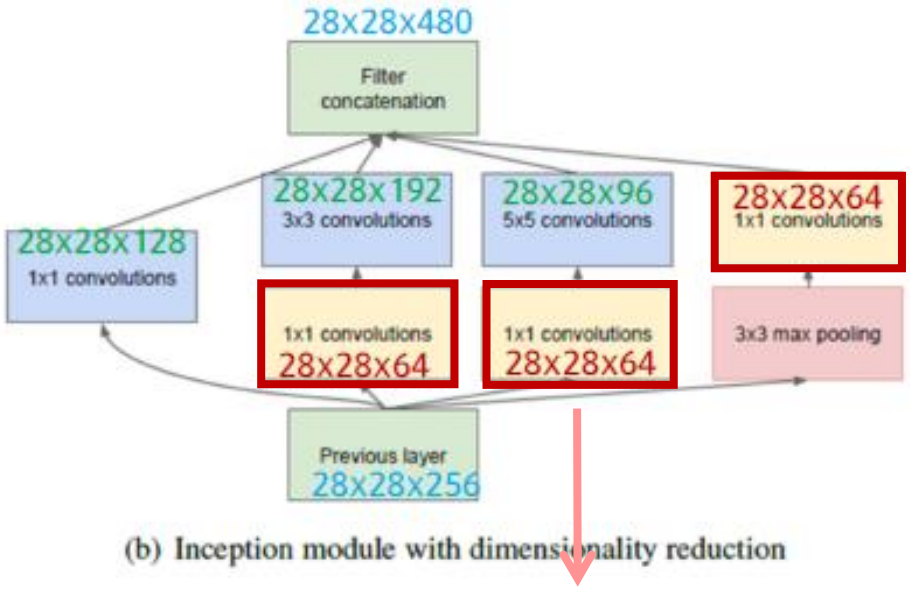


1x1 Convolution을 통해 비선형성 해결, 깊이 축소, 성능 개선

3-4. Inception



(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

1x1 Filter 64개 도입

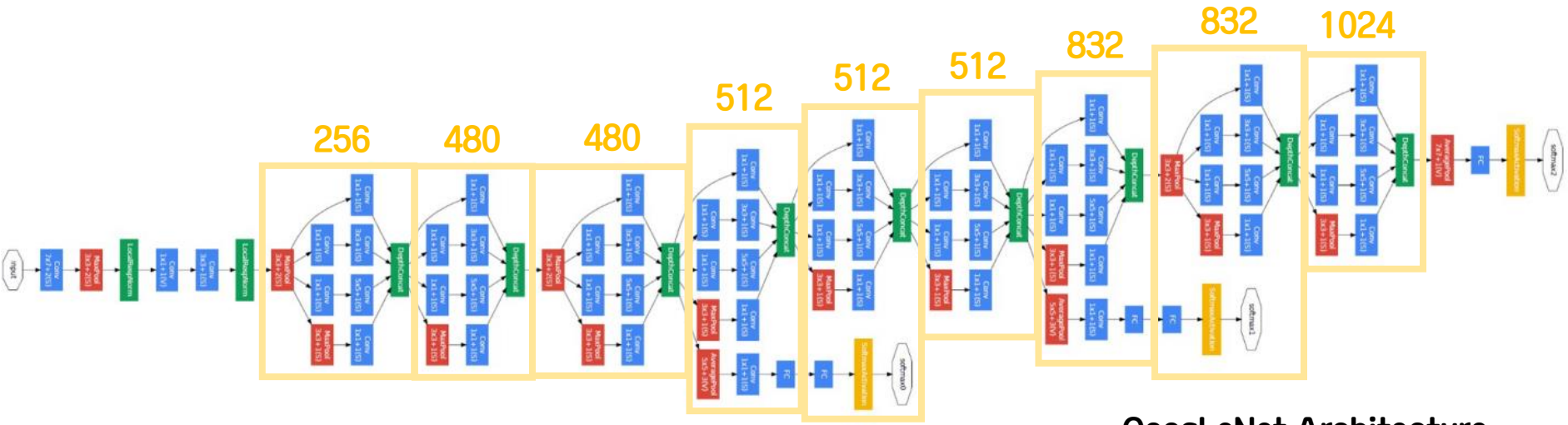
· 다양한 scale의 피쳐를 추출하기 위해 다양한 convolution으로 연산 시도

· 망이 넓어지고 깊어질때, 3x3 5x5는 연산량이 너무 많음

· 1x1 convolution 사용 = 연산량 줄임

· 깊이 256 -> 64 -> 192 -> bottleneck 구조

3-5. GoogLeNet Architecture



GoogLeNet Architecture

파란색 : Convolution Layer

빨간색 : Max-Pooling Layer

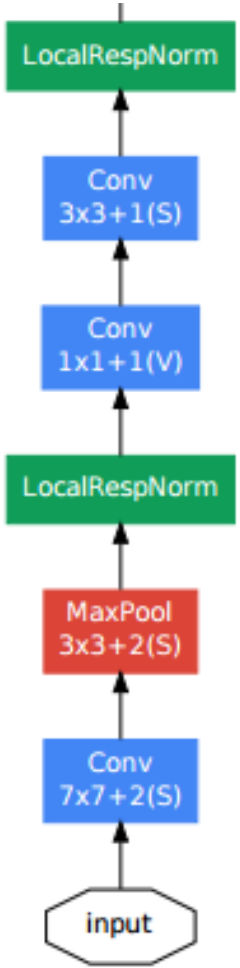
노란색 : Softmax Layer

녹색 : 기타 Function

노란색 박스 : Inception Module

박스 위 숫자 : Feature-map

3-5.1. GoogLeNet Architecture



< Architecture의 첫 부분 >

(S) : STRIDE

Stem Network

: 픽셀 레벨의 특징을 잡아내는 용도

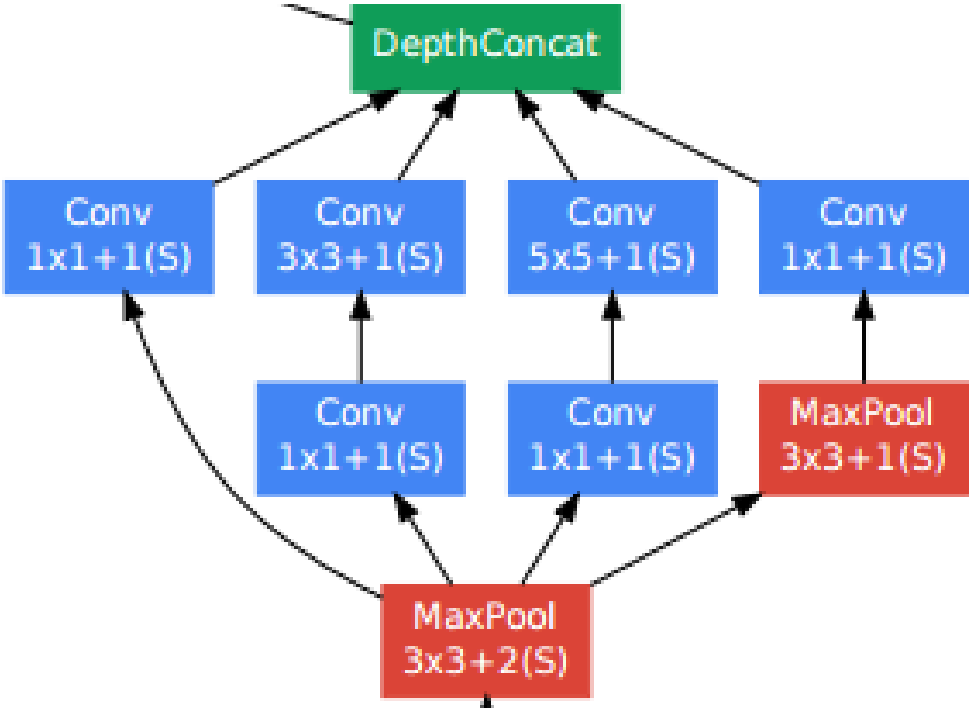
: 가장 단순한 Conv-pool 스타일

LRN 수행

Pooling

3-5.2. GoogLeNet Architecture

< Inception Module >



3-5.3. GoogLeNet Architecture

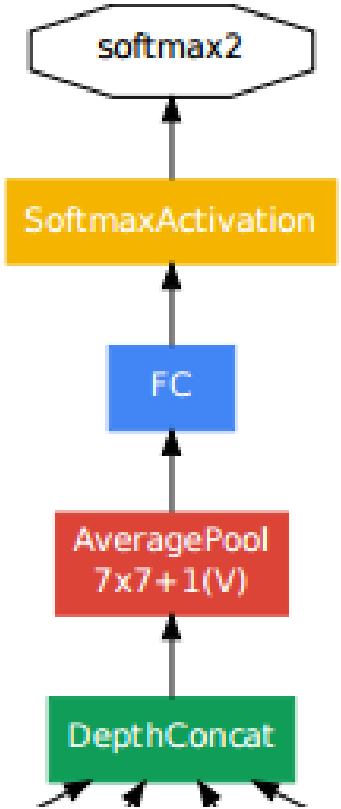
< Architecture의 중간 부분 >

· 분류기 형식 부분

· 해당 두 부분은 auxiliary classifiers로써 network가 깊어짐에 따라 vanishing gradient가 생기는 것을 방지

3-5.4. GoogLeNet Architecture

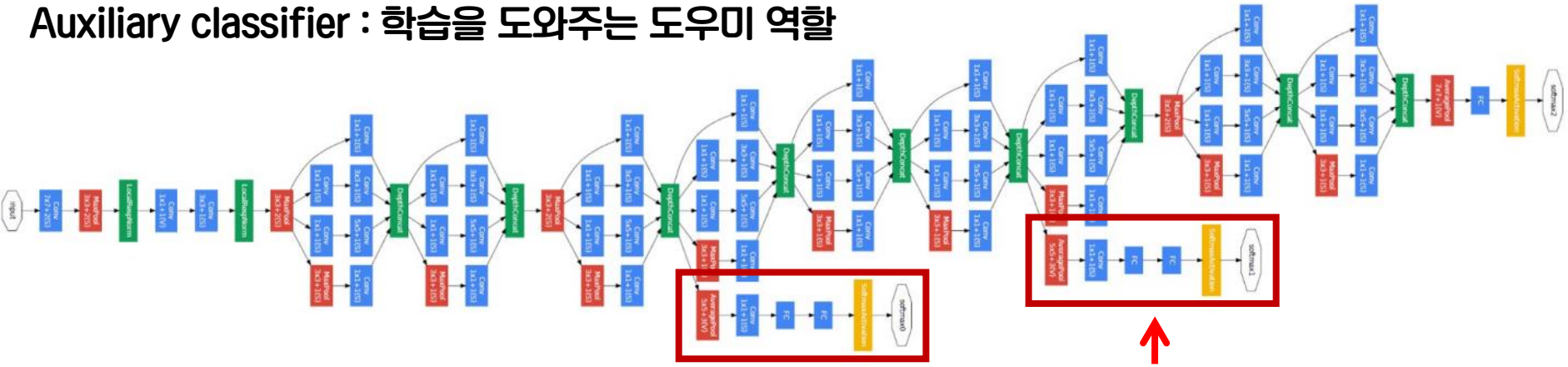
< Architecture의 마지막 부분 >



· 최종적인 분류를 하기 위한 output layer 부분

3-6. Auxiliary classifier

Auxiliary classifier : 학습을 도와주는 도우미 역할



문제점 : Vanishing Gradient

-> 학습 속도가 느려지거나 Overfitting 문제 발생

해결책 : Auxiliary classifier를 중간 2곳에 배치

3-7. GoogLeNet Layer

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Patch size / Stride

: 커널의 크기, stride 간격

Output size

: feature-map의 크기 및 개수

Depth

: 연속적인 convolution layer 개수

#1x1

: 1x1 convolution을 수행하고 얻는 Feature-map 개수

#3x3

: 3x3 convolution을 적용하고 얻는 Feature-map 개수

#5x5

: 5x5 convolution을 적용하고 얻는 Feature-map 개수

Pool / proj

: max-pooling 뒤에오는 1x1 convolution을 적용한 것

Params

: 해당 layer에 있는 free parameter 개수

Ops

: 연산의 수

3-8. GoogLeNet 요약

GoogLeNet

· Inception Module 사용

-> 더 깊고 넓은 네트워크 구성

· 1x1 Convolution Layer

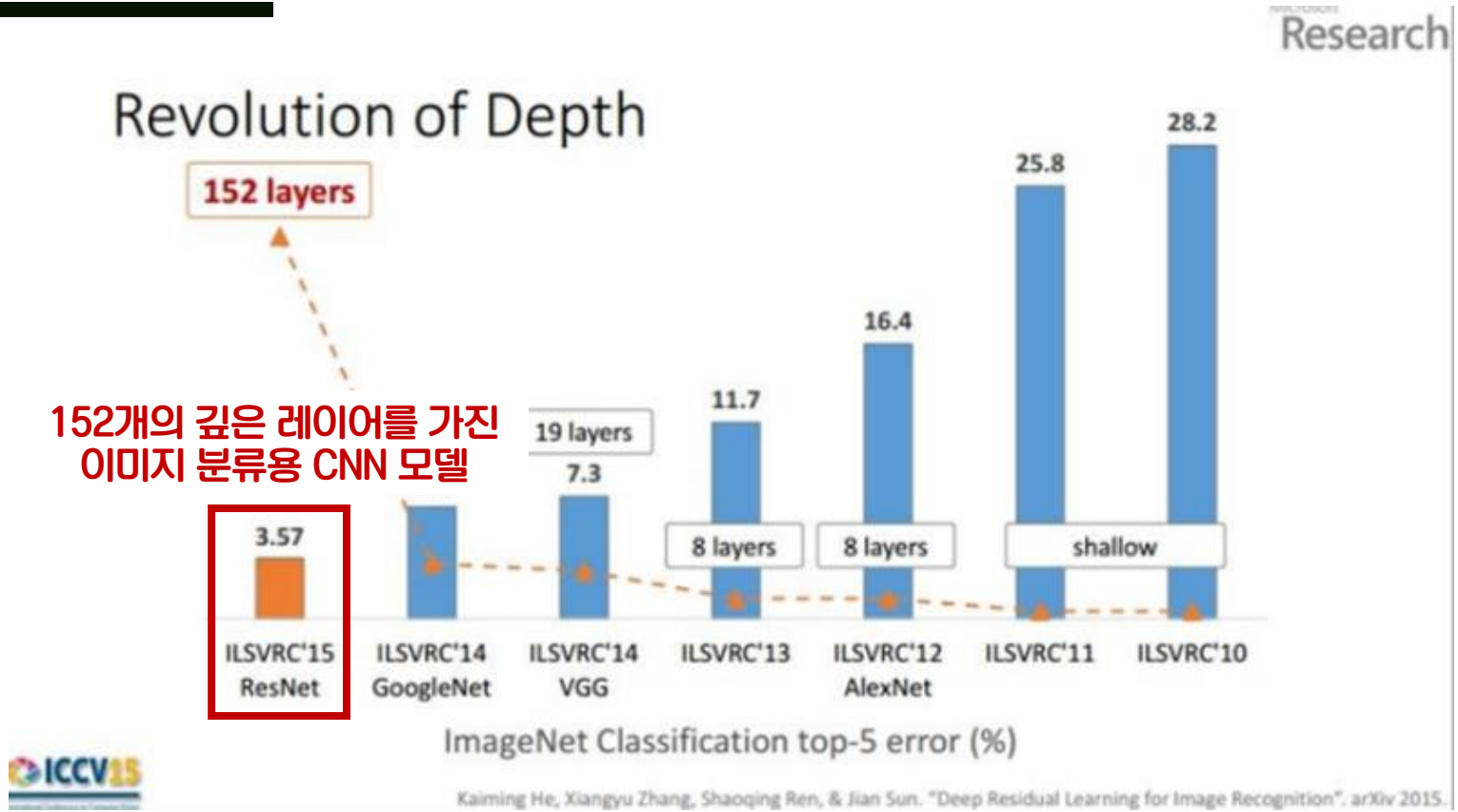
-> 연산량 줄임

· Bottleneck Layer 추가

-> 유연하게 대응하기 위해 여러가지 필터 사이즈 (1x1, 3x3, 5x5) 사용

-> 이로 인한 계산량 증가 방지

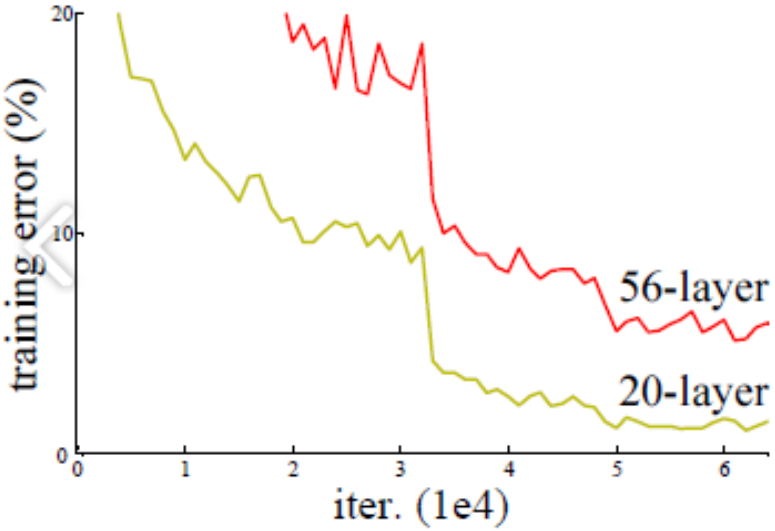
4. ResNet

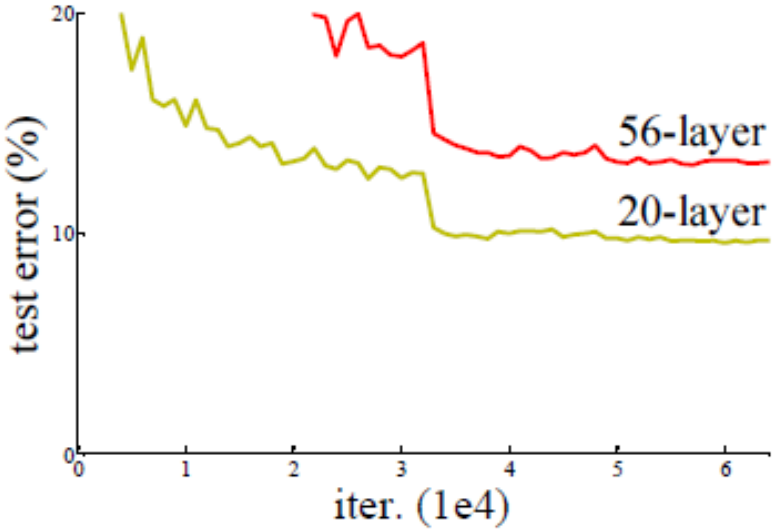


네트워크가 깊어지면서 top-5 error가 낮아짐 = 성능 좋아짐

4-1. ResNet 연구

망을 깊게 하면 무조건 성능이 좋아질까요?





더 깊은 구조를 갖는 56층의 네트워크가 20층의 네트워크보다

더 나쁜 성능을 보인다.

4-1.1. ResNet 연구 가설

ResNet 연구의 가설

더 깊은 모델 학습 시 optimization에 문제가 생긴다

(모델이 깊어질수록 최적화가 어렵다)

↓

모델이 더 깊다면 적어도 더 얕은 모델만큼 성능이 나와야 한다.

↓

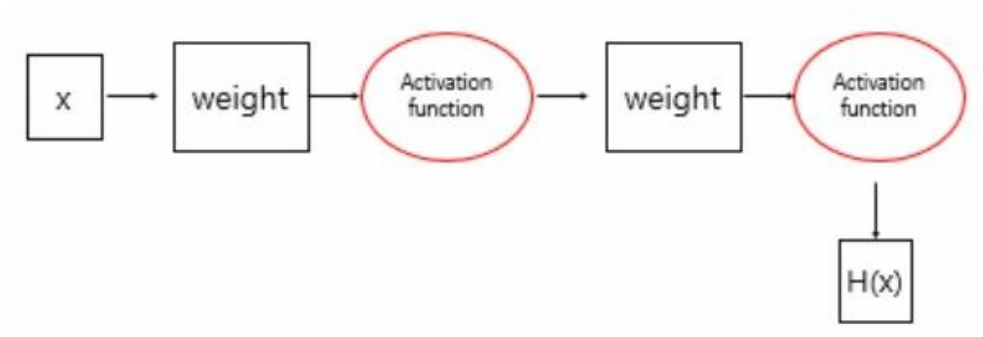
더 얕은 모델의 가중치를 깊은 모델의 일부 레이어에 복사, 나머지 레이어는 identity mapping

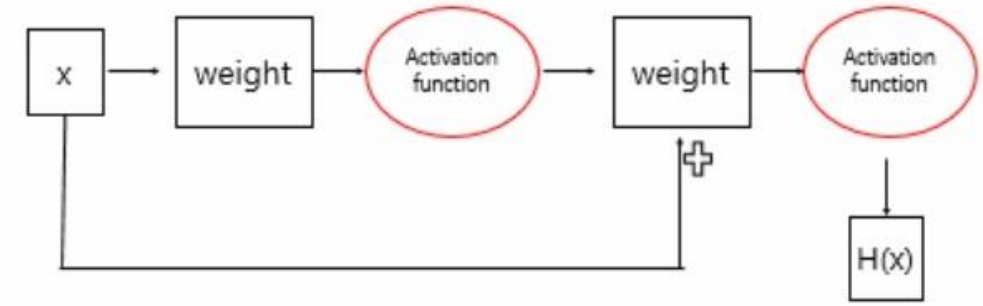
↓

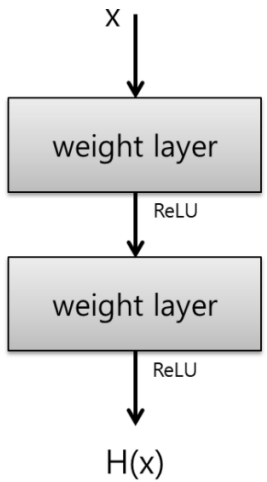
이렇게 구성하면 shallower layer 만큼 성능이 나온다

4-1.2. ResNet 모델 비교

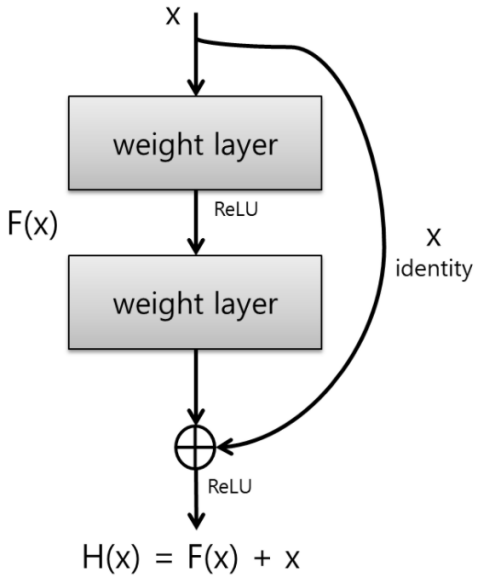
〈 기존 모델 〉







기존 방식



Residual block

ResNet 목적

: F(x) + x 를 최소화하는 것

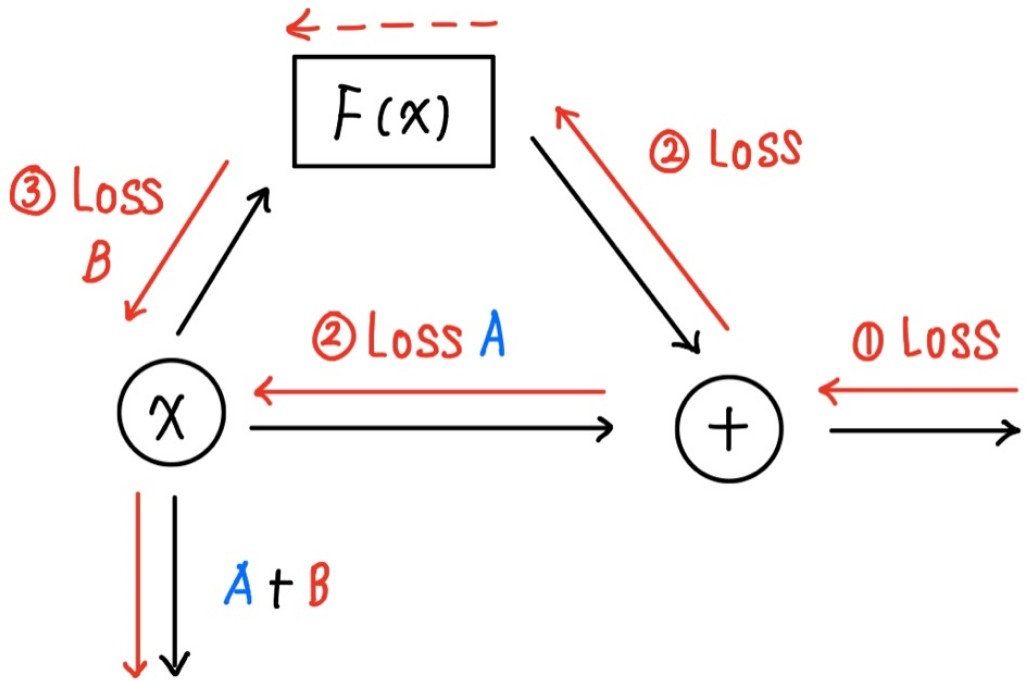
-> forward / backward가 단순해지는 효과

ResNet Effect

- 깊은 망도 쉽게 최적화 가능
- 늘어난 깊이로 인해 정확도 개선

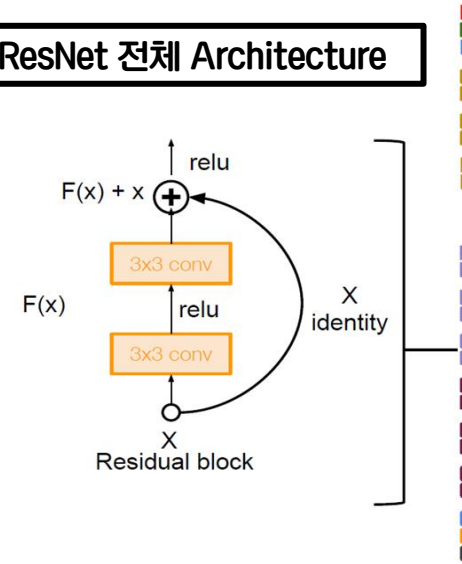
4-1.3. ResNet Computation Graph

< Computation graph >



4-2. ResNet Architecture

ResNet 전체 Architecture



Plain Net 구조

: (뼈대) VGG19 + 컨볼루션 층들

ResNet 구조

: (뼈대) VGG19 + 컨볼루션 층들 + shortcut들

- 균일하게 3x3 사이즈의 컨볼루션 필터 사용
- 특성맵의 사이즈 $\frac{1}{2}$ = 특성맵의 뎁스 2배

4-3. ResNet Layer

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

가장 성능 뛰어남

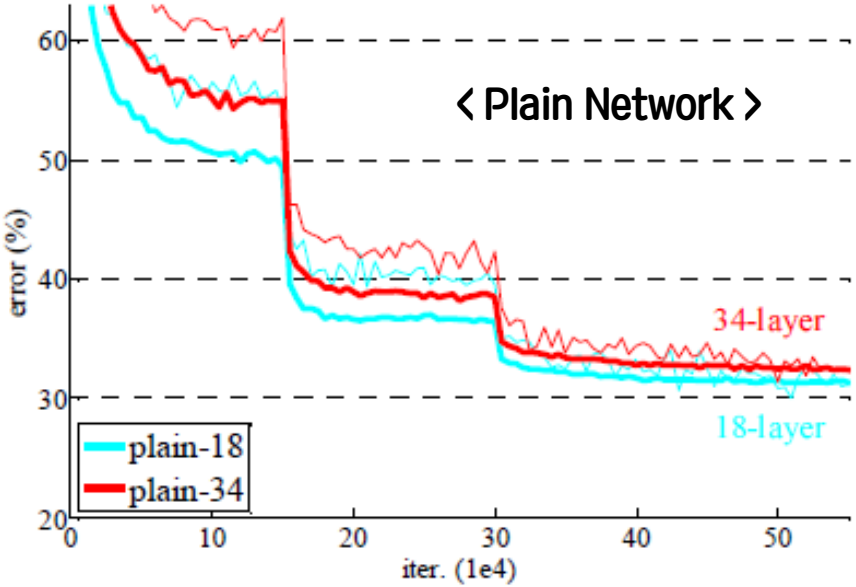
18층, 34층, 50층, 101층, 152층의 ResNet Layer 구성

4-4. ResNet Experiment

실험 원칙

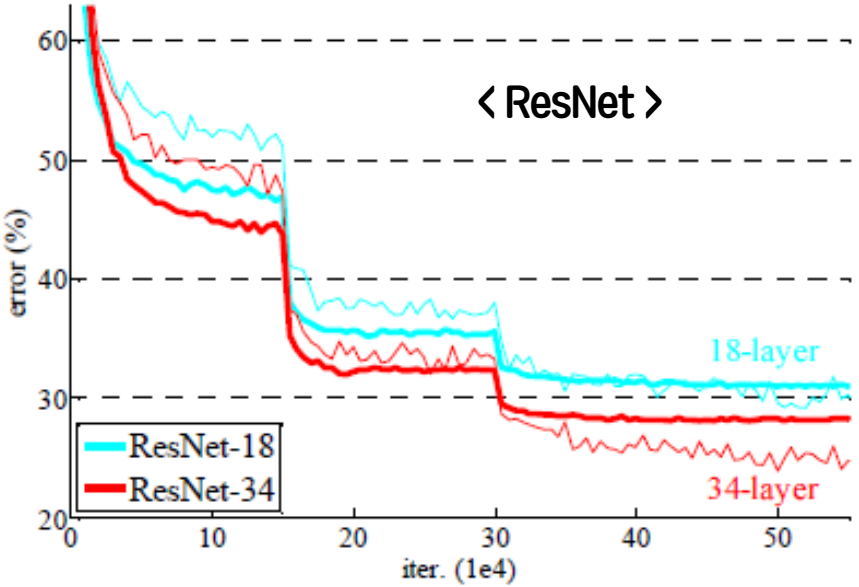
- 출력 feature-map 크기가 같은 경우, 해당 모든 layer는 모두 동일한 수의 fillter를 갖음
- Feature-map의 크기가 절반으로 작아지는 경우는 연산량의 균형을 맞추기 위해 필터의 수를 두 배로 늘림

실험 결과



Plain Network : 망이 깊어지면서 에러가 커졌음

-> 34층의 Plain Network가 18층의 Plain Network보다 성능이 나쁨



ResNet : 망이 깊어지면서 에러가 작아짐

-> Shortcut을 연결해서 residual를 최소가 되게 학습

4-5. ResNet 요약

ResNet

· 이미지 분류에 쓰이는 CNN

· 문제점 : 망의 깊이가 늘어날수록 무조건 성능이 좋아지는 것은 아님

-> Over-fitting 때문이 아님

· 해결책 : Residual Block 사용 = 지름길 만들기

-> Gradient vanishing 문제 해결

감사합니다

Q&A