DATA
SCIENCE LAB

CNN 2

23.02.16 / 8기 정건우

0. INTRO

CNN1

Computer Vision이란? CNN은 왜 등장했을까? CNN 기본 개념

Convolutional Layer / Padding, Stride / Batch Normalization, ···

CNN2

지금까지는 **어떤 CNN 모델**들이 있었지? CNN 모델의 발전과정

CNN Architecture / Comparing Model Complexity

CNN3

그래서 **성능 좋은 (CNN) 모델**을 만들려면 어떻게 해야 되는데?

CNN을 이용하는 이유 / **전이학습(Transfer Learning)**

0. Review

CNN overview

Feature Extractor는 아래 과정의 반복

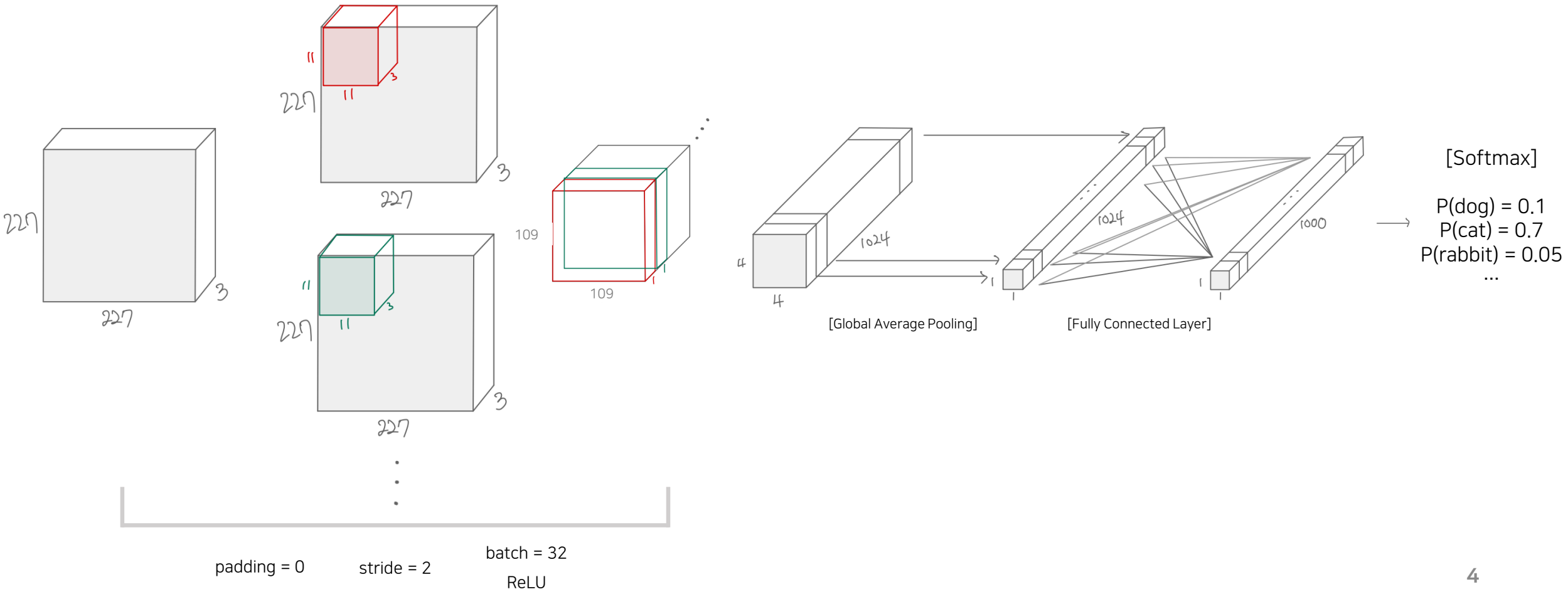
Conv Layer → Batch Normalization → Activation func (ReLU, Sigmoid, ⋯) → (Dropout) → (Pooling)

Classifier는 Feature Extractor 과정의 반복이 끝나면

Global Average Pooling → FC Layer → Softmax → Classification

0. Review

CNN overview



0. Review

What is 'Good' CNN model?

무작정 Feature Extractor 쌓고, Classifier 써서 CNN 모델 만들면 모든 이미지를 분류할 수 있을까?

→ NO. 유의미한 Feature를 만들어낼 수 있도록 모델을 구성해야 함

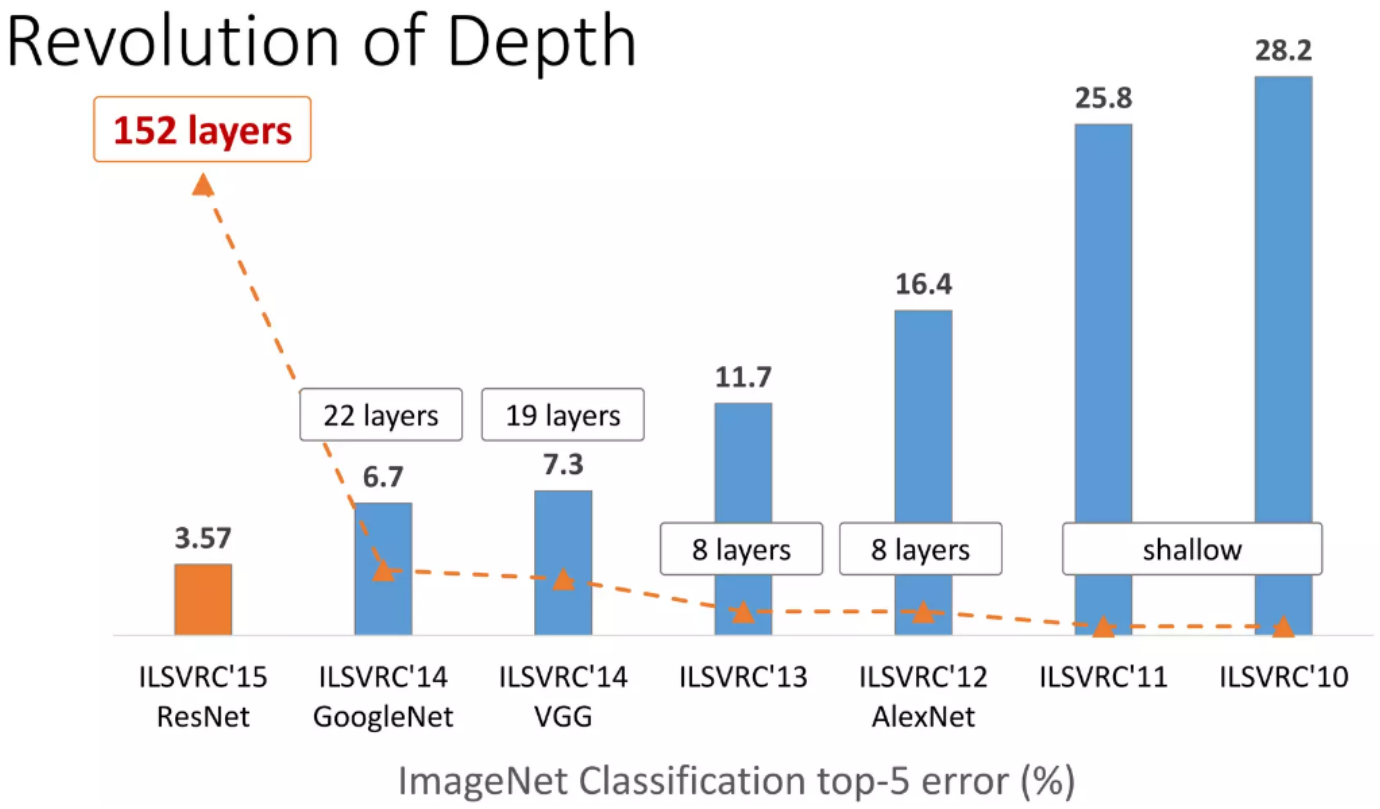
→ Filter의 크기, Network의 깊이, Activation func, 언제 어디서 Padding, Stride, Pooling 할지, …

1998년 Yann LeCun이 Convolutional Neural Network를 최초로 개발한 이후,

Image Recognition의 성능(accuracy, time complexity, …)을 높이기 위한 다양한 모델이 등장

→ 유의미하다고 평가받는 모델들을 살펴보면서, CNN 모델의 발전을 알아보고 배운 개념을 이해해보자.

0. Review


Revolution of Depth
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
ImageNet Classification top-5 error (%)

CONTENTS

1. AlexNet

Architecture

"First CNN-based winner"

Input : 227 * 227 *3

5 Conv layer + 3 FC layer

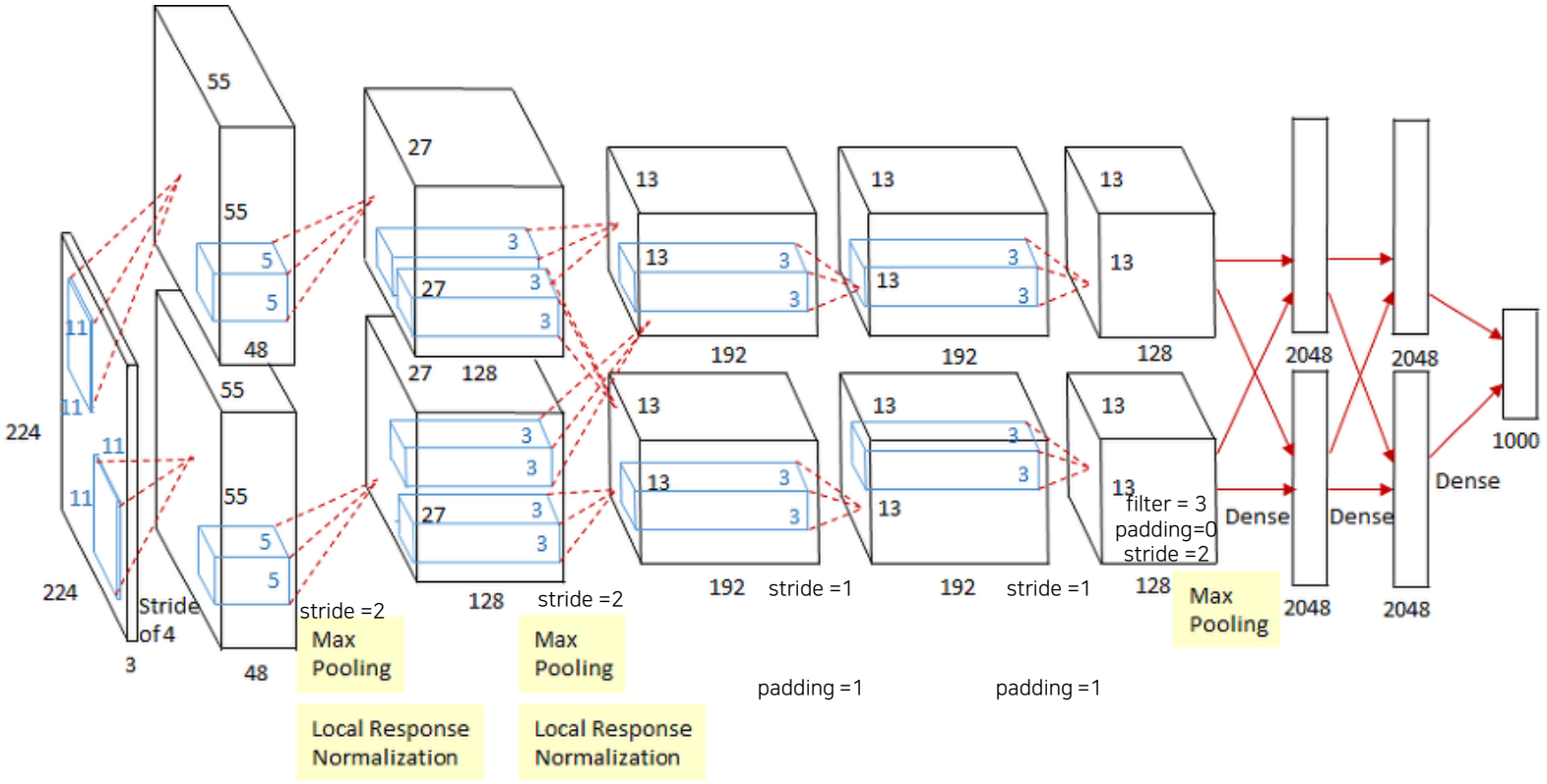
ReLU Nonlinearities

Max pooling (Overlapping)

Local Response Normalization

Data Augmentation

Dropout (in FC layer 1,2)



1. AlexNet

Complexity

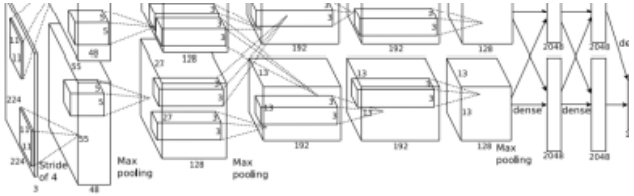
of parameters가 약 6,000만 개

(Ex. Conv1에서 # of parameters는

3 * 11 * 11 * 64 = 23,232)

대부분의 parameter가 FC Layer에 있다 !

AlexNet



	Input size			Layer				Output size					
Layer	C	H / W		filters	kernel	stride	pad	C	H / W		memory (KB)	params (k)	flop (M)
conv1	3	227		64	11	4	2	64	56		784	23	73
pool1	64	56			3	2	0	64	27		182	0	0
conv2	64	27		192	5	1	2	192	27		547	307	224
pool2	192	27			3	2	0	192	13		127	0	0
conv3	192	13		384	3	1	1	384	13		254	664	112
conv4	384	13		256	3	1	1	256	13		169	885	145
conv5	256	13		256	3	1	1	256	13		169	590	100
pool5	256	13			3	2	0	256	6		36	0	0
flatten	256	6						9216			36	0	0
fc6	9216			4096				4096			16	37,749	38
fc7	4096			4096				4096			16	16,777	17
fc8	4096			1000				1000			4	4,096	4

1. AlexNet

Overlaping Pooling

우리가 흔히 아는 Max pooling 은 Non-overlapping 방식

AlexNet은 window=3, stride=2 인 overlapping 방식. 왜?
→Image의 해상도가 낮았기 때문에, pixel 하나하나가 중요했음
→Overfitting을 막기 위해, 한 번의 pooling이어도 window 크게 !

[0 0 5 0 0 6 0 0 3 0 0 4 0 0]

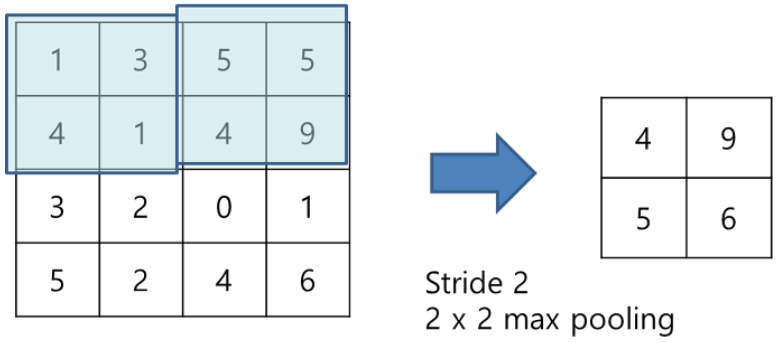
[0 0 0 5 0 6 0 0 0 3 0 4 0 0]

[0 0 5 0 0 6 0 0 3 0 4 0 0 0]

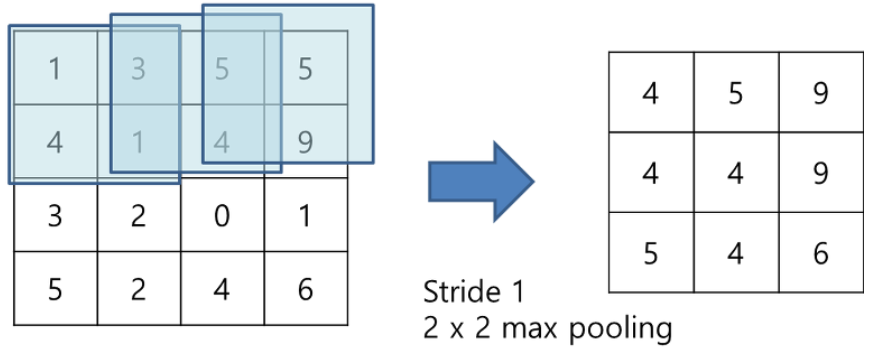
ex. stride 2 / width 2
⇒ 모든 결과가 0 5 6 0 3 4 0

ex. stride 2 / width 3
⇒ 5 5 6 3 3 4 0
⇒ 0 5 6 0 3 4 0
⇒ 5 5 6 3 4 4 0
서로 다른 결과를 보인다.

Non-overlapping pooling



Overlapping pooling



1. AlexNet

LRN (Local Response Normalization)

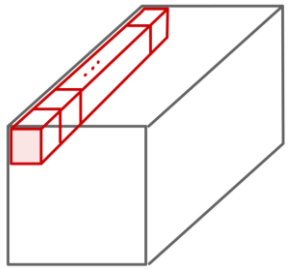
$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

$a^i_{x,y}$는 픽셀 x,y에서 i번째 채널의 값

$b^i_{x,y}$는 정규화된 결과

가까운 채널의 feature map에서 같은 위치에 있는 뉴런들을 정규화

현재에는 LRM 사용하지 않고, Batch Normalization을 사용



2. VGGNet

Architecture

"Deeper Network"

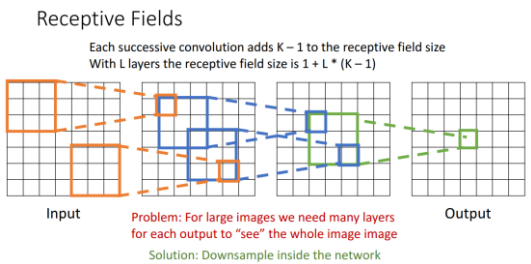
13 Conv layer + 3 FC layer

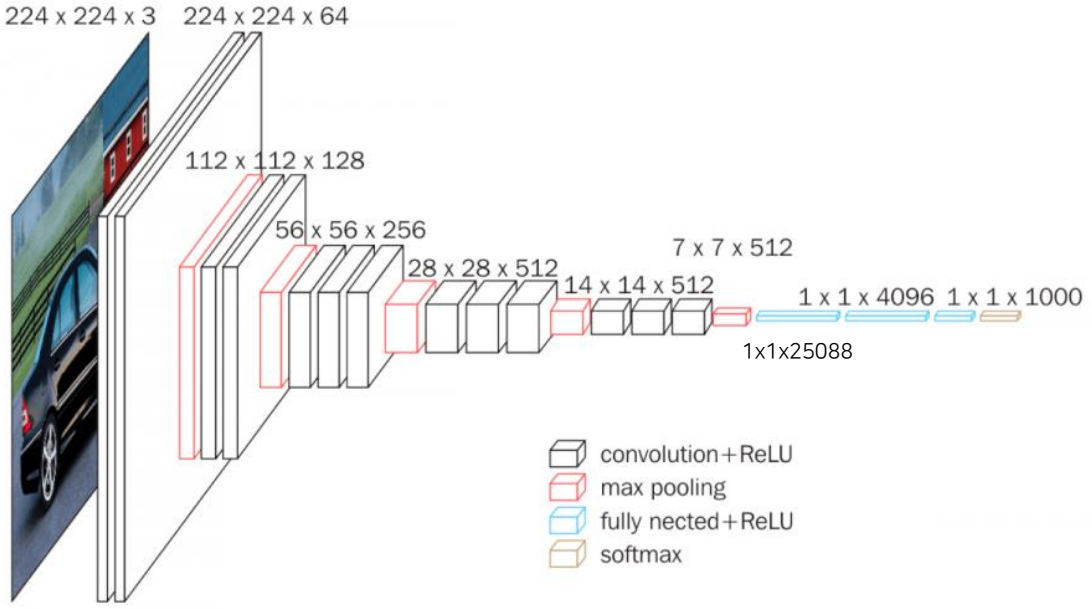
3*3 small filter(kernel)

→ Deeper Network

→ 동일한 Receptive field에 대해 More non-linearities & Less parameters

→ 풍부한 feature를 효율적으로 뽑아낼 수 있음





But 초반 feature map이 크고, FC layer 사용

→ memory도 많이 필요하고 parameter도 많음

→ 지금 관점에서 별로 효율적이지는 않지만, 참신했다 !

2. VGGNet

Complexity

VGG16

VGG19

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

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).

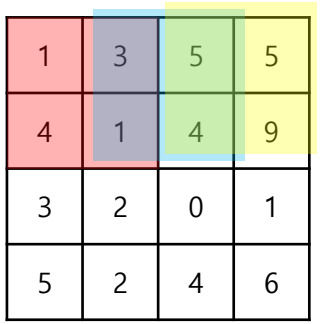
Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

# of parameters 약 1억 4천만 개

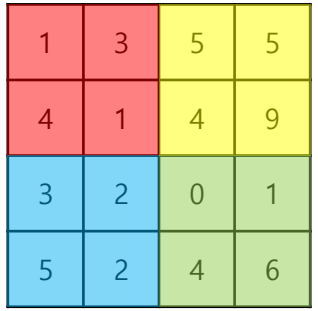
(parameters 90% 이상이 FC layer)

2. VGGNet

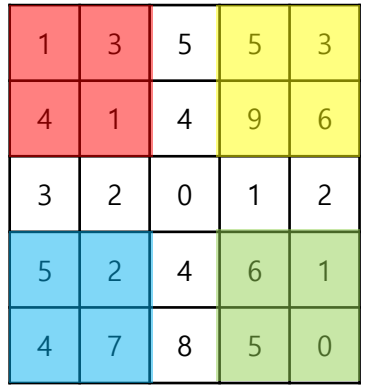
Max pooling



AlexNet의 Overlapping max pooling
(pooling size > stride)



VGGNet의 max pooling
(pooling size = stride)



Conv filter size < stride

2. VGGNet

How downsampling ?

Filter size = 3, padding =1

→ Conv layer 거쳐도 input_dim = output_dim

$$\frac{N-F+2P}{1}+1=\frac{N-3+2}{1}+1=N$$

Max pooling으로 downsampling

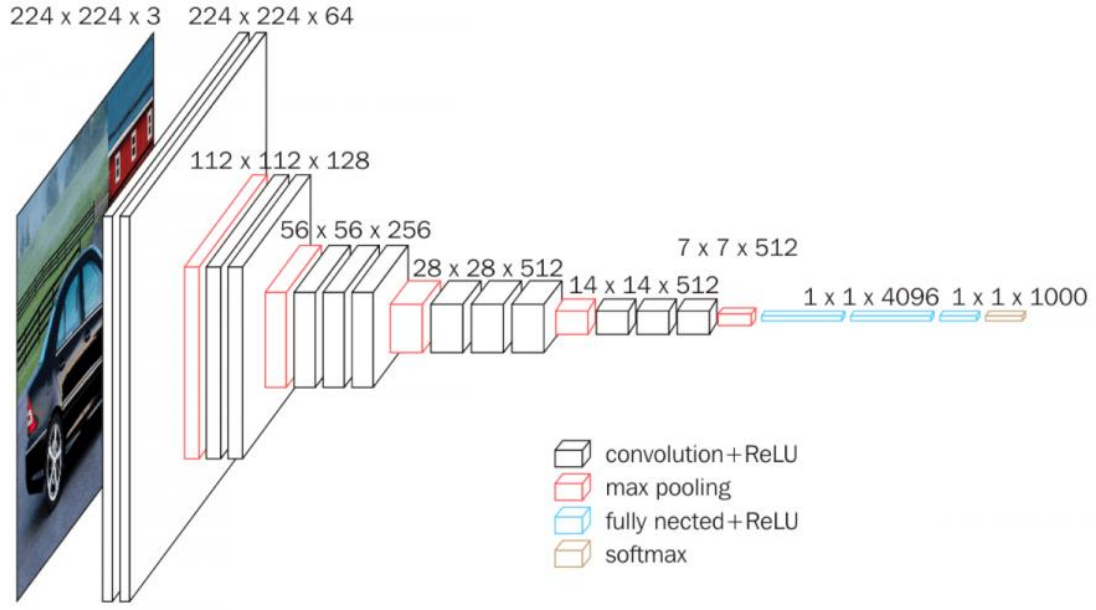
→ VGG 팀의 연구 목적이

"네트워크 깊이에 따른 성능 " 이었음

1	3	5	5
4	1	4	9
3	2	0	1
5	2	4	6

VGGNet의 max pooling

(pooling size = stride)



2. VGGNet

Performance

Table 4: **ConvNet performance at multiple test scales.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
B	256	224,256,288	28.2	9.6
C	256	224,256,288	27.7	9.2
	384	352,384,416	27.8	9.2
	[256; 512]	256,384,512	26.3	8.2
D	256	224,256,288	26.6	8.6
	384	352,384,416	26.5	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**
E	256	224,256,288	26.9	8.7
	384	352,384,416	26.7	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**

D : VGG16, E : VGG19

→ layer가 깊어져도 성능이 별로 다르지 않거나 오히려 저하하는 경우도 발생

→ 무작정 깊은 네트워크를 만든다고 error가 내려가는 것은 아님을 발견

→ 네트워크 깊어지면 # of parameter 많아지니까 Overfitting이 발생했기 때문인가?

(라고 당시에는 생각했지만… 나중에 알고 보니 아니었음)

3. GoogleNet

Architecture

"Inception-v1"

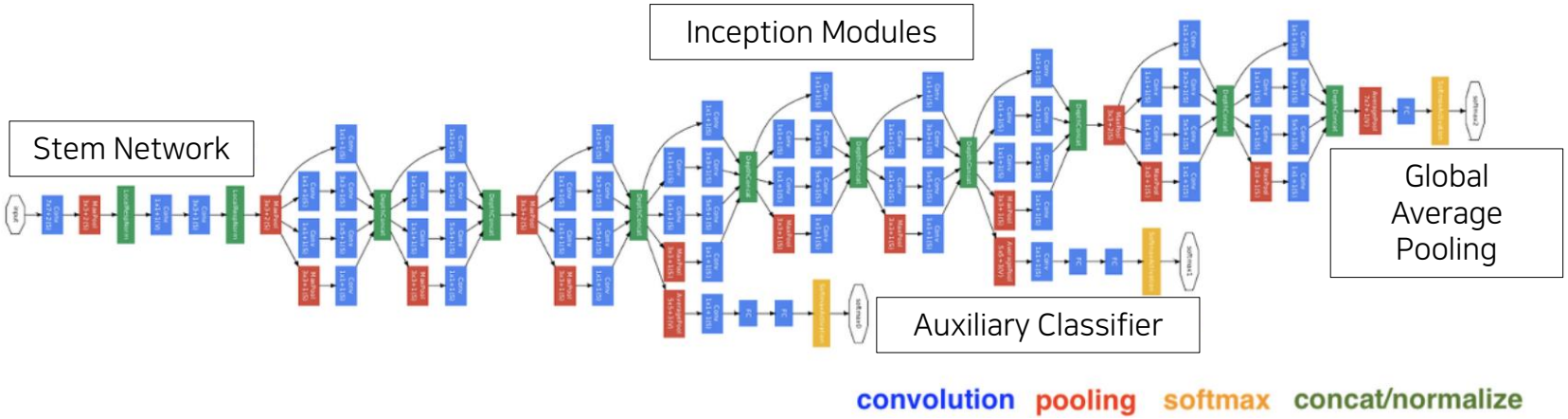
9 Inception Modules (22 layer)

Stem Network : 7*7 conv filter

Inception Modules : 1*1 conv filter

Auxiliary Classifier : increase the gradient signal

Global Average Pooling : fewer parameter



3. GoogleNet

Stem Network

Network 초반에 큰 filter 하나를 통과시켜 빠르게 downsampling !

그러면 VGG가 3*3 conv 여러 번 써가면서 뽑은 Good Feature를 GoogleNet은 어떻게 뽑을까?

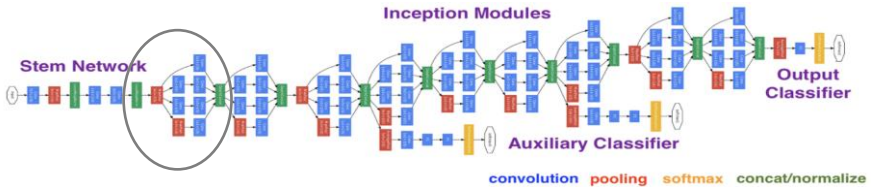
→ Inception module !

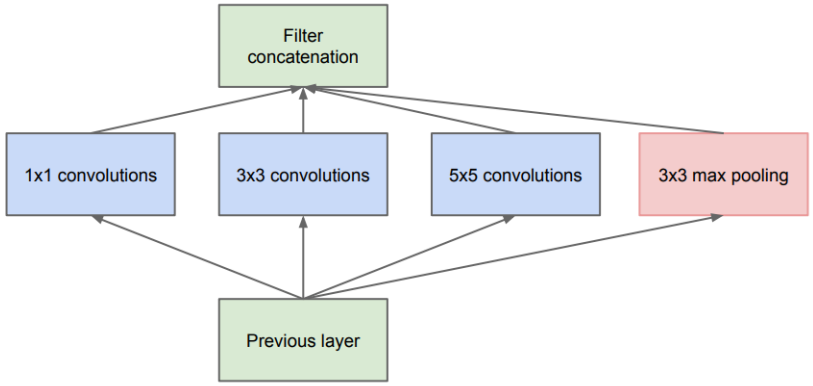
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	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								

Table 1: GoogLeNet incarnation of the Inception architecture

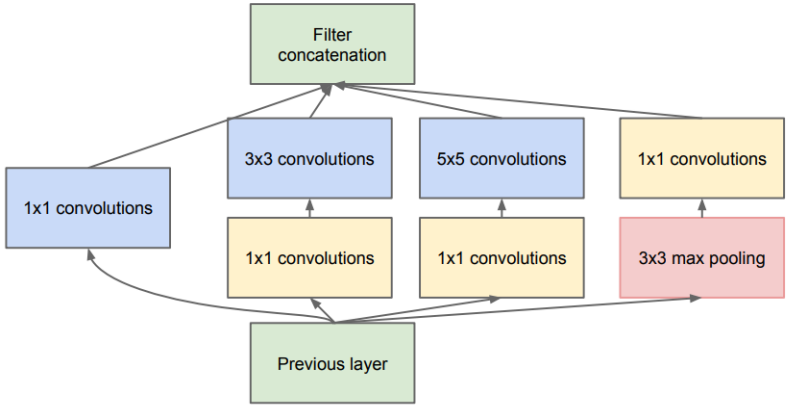
3. GoogleNet

Inception Modules

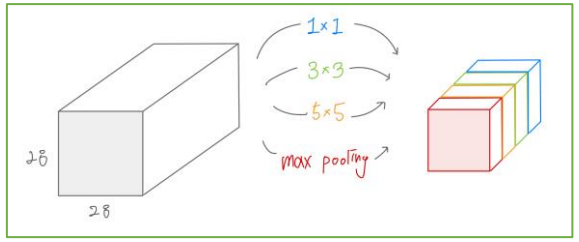




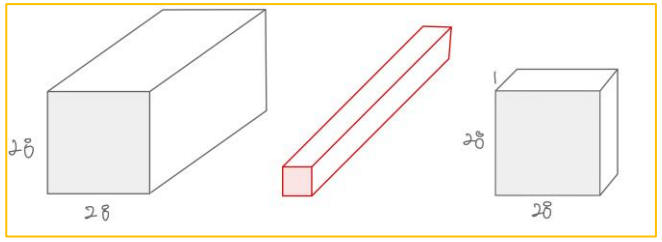
(a) Inception module, naïve version



(b) Inception module with dimension reductions

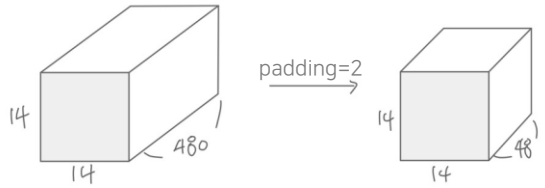


다양한 size의 filter를 씌워서
나온 결과를 Ensemble
(다양한 종류의 feature를 extract)



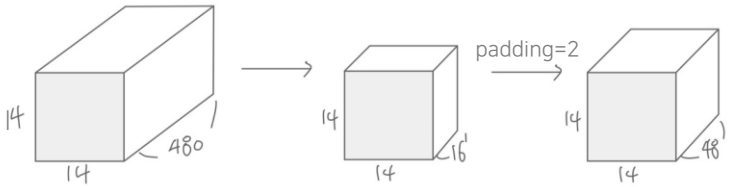
다양한 size의 filter를 씌우려면
parameters가 너무 많아지지 않나?
→ 1*1 conv filter로 해결 !

1*1 Conv filter가 뭘 할 수 있지?



Filter (5*5*480)
48개

→ # of operation
= (5*5*480)*(14*14*48) = 112.9M

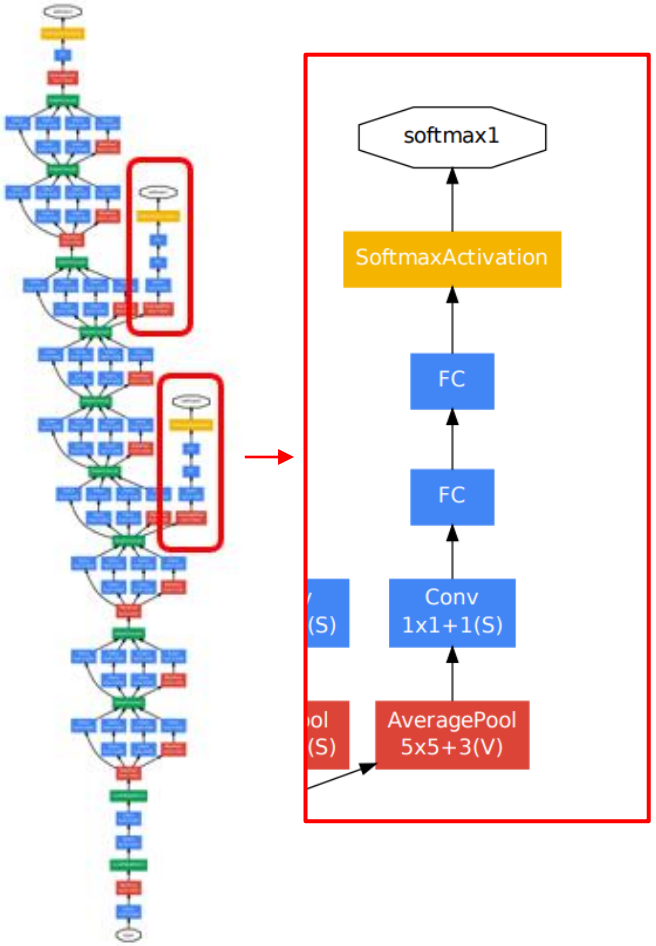


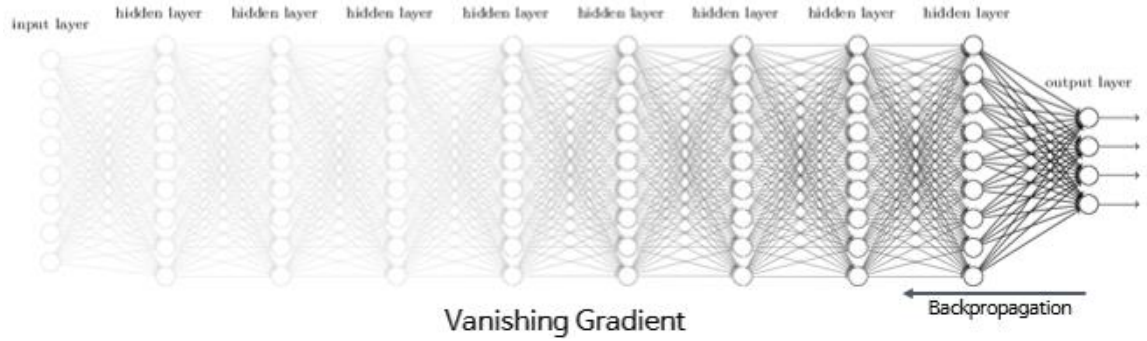
Filter (1*1*480) Filter (5*5*16)
16개 48개

→ # of operation
= (1*1*480)*(14*14*16) + (5*5*16)*(14*14*48) = 5.3M

3. GoogleNet

Auxiliary Classifier





모델의 layer가 깊어지면, backpropagation 할 때

Input쪽에는 vanishing gradient 발생

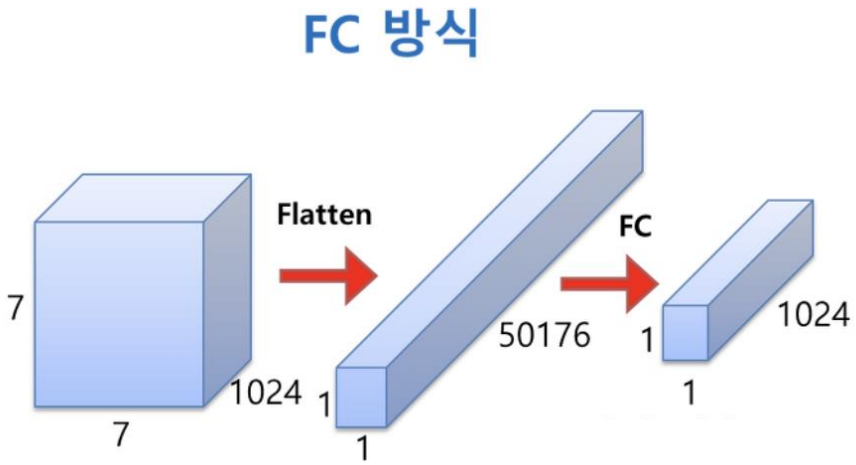
→ 지금까지 학습한 정보에 기반한 softmax classifier 붙여놓고 backpropagation 중간에 gradient를 보충 !

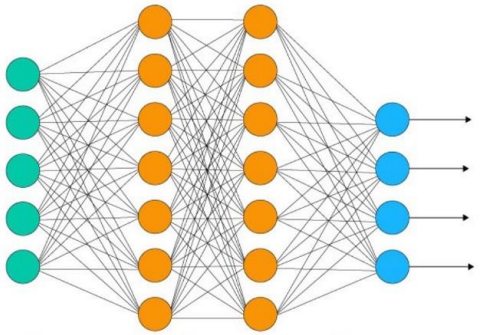
→ vanishing gradient 방지

→ Auxiliary classifier는 train에서만 사용하고, 모델을 사용할 땐 제거

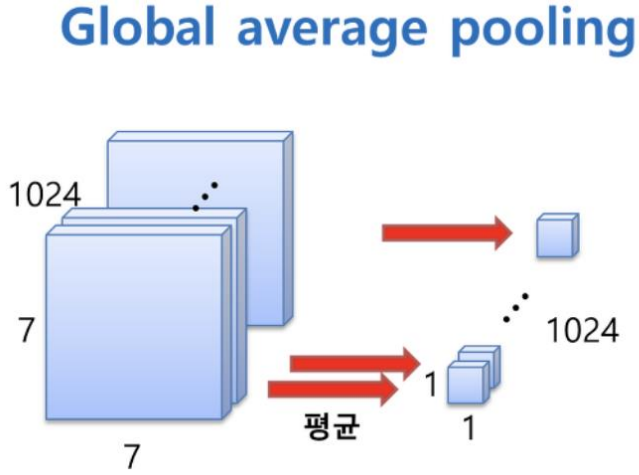
3. GoogleNet

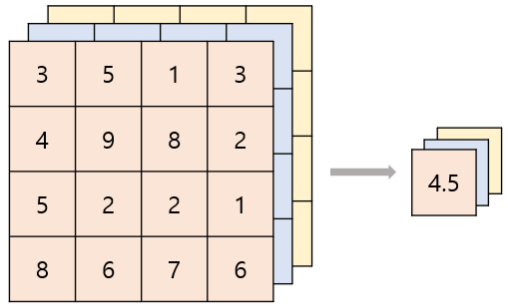
Global Average Pooling





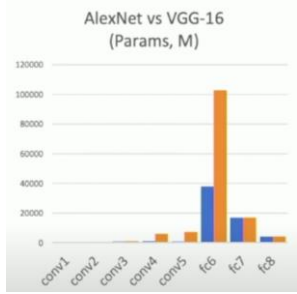
# of parameters
= 7 * 7 * 1024 * 1024
= 51,380,224



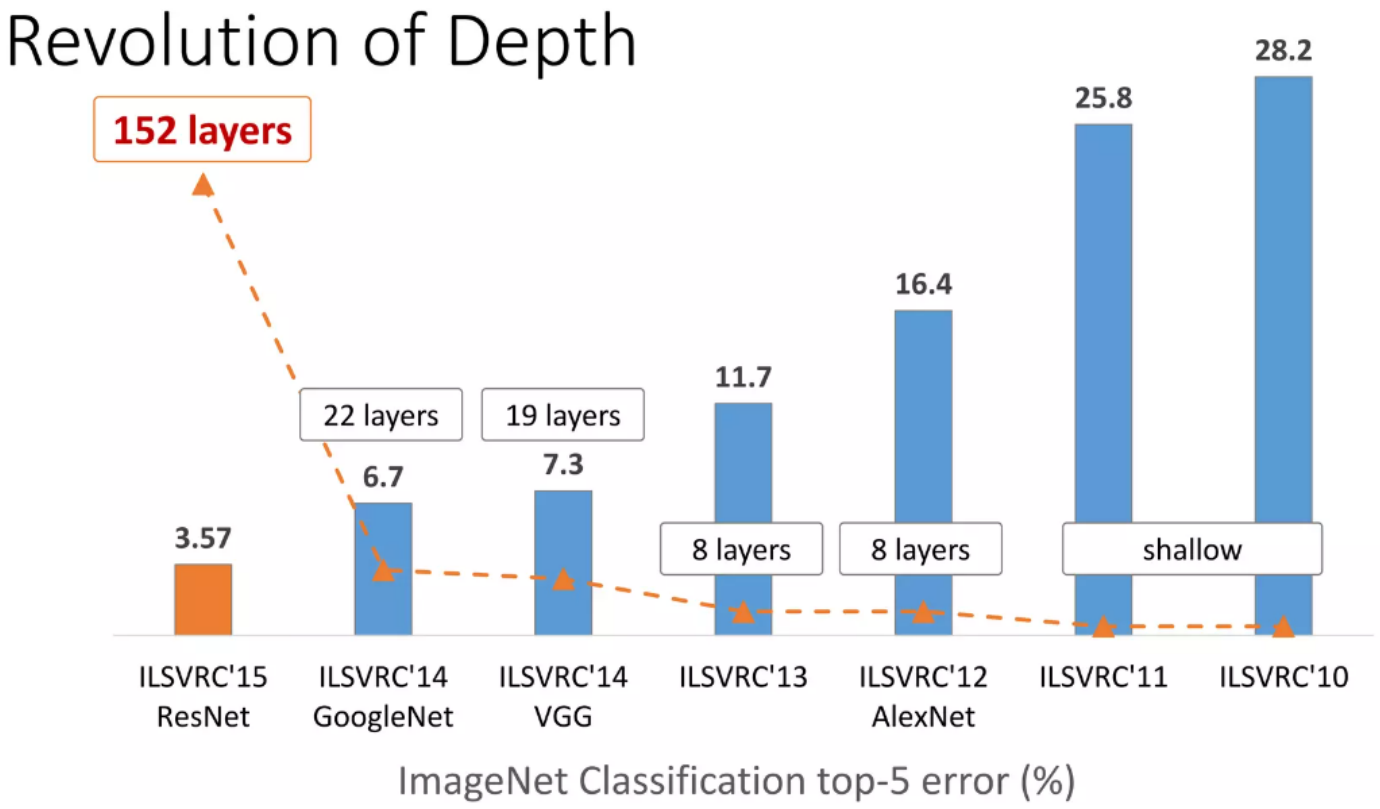


Global Average Pooling

# of parameters
= 0

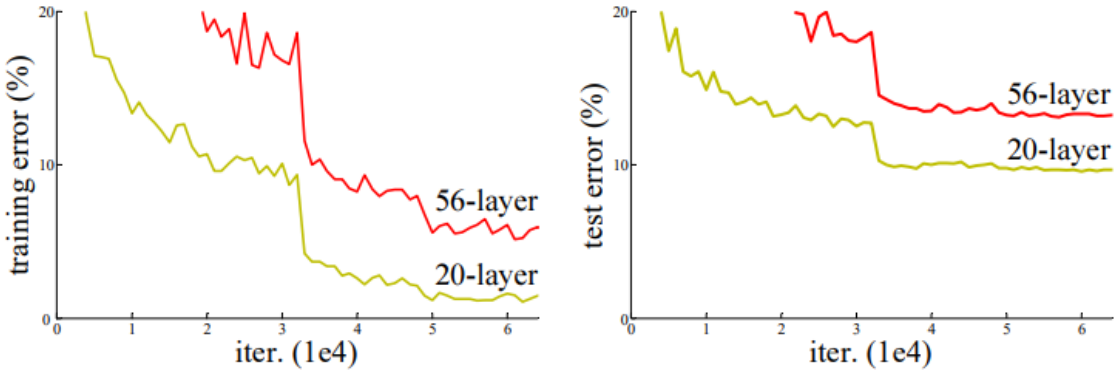


4. ResNet


Revolution of Depth
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
ImageNet Classification top-5 error (%)

4. ResNet

Background



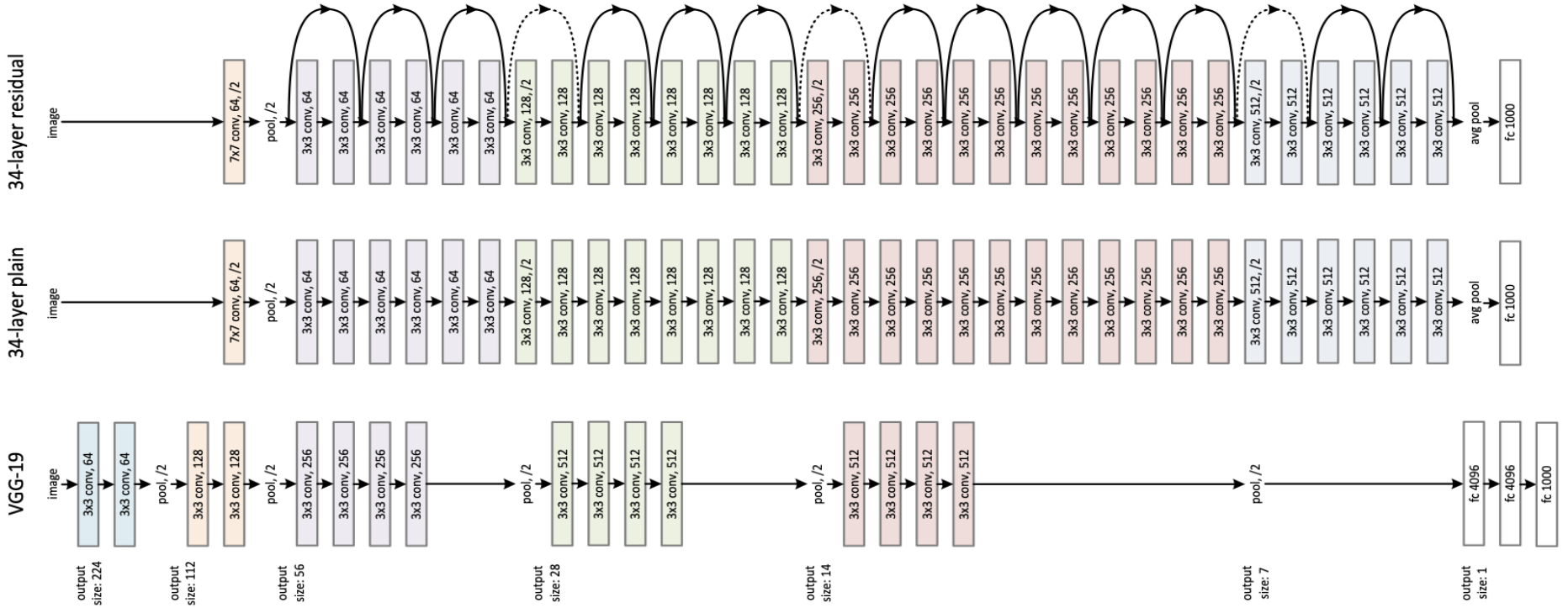
ResNet 팀 “VGGNet은 layer가 깊을수록 test error 뿐만 아니라 training error도 이 더 높다”

→ overfitting이 아니라, underfitting이었다 ?

→ 오히려 “더 깊은 네트워크”가 “덜 깊은 네트워크”를 emulate(모방) 하더라 !

4. ResNet

Architecture



4. ResNet

Architecture

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				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	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

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

처음에 64개의 7*7 conv filter로 downsampling 해주고 (GoogleNet처럼)

2~3개의 3*3 conv filter를 Residual block로 묶어서 conv2_x, conv3_x, …라 하고 conv_x 사이사이 BN & ReLU

마지막에는 Global Average Pooling → FC layer → Softmax

4. ResNet

Residual Block

기존 CNN 모델처럼 conv layer를 거친 결과 H(x)를 학습하기보다는

학습 결과 H(x)에서 input된 x를 제외한 residual F(x)=H(x)-x를 학습하자 !

H(x)를 학습할 때에는

→ 지금까지 학습된 feature인 x를 기반으로 H(x)를 새롭게 학습해야 함

F(x)를 학습할 때에는

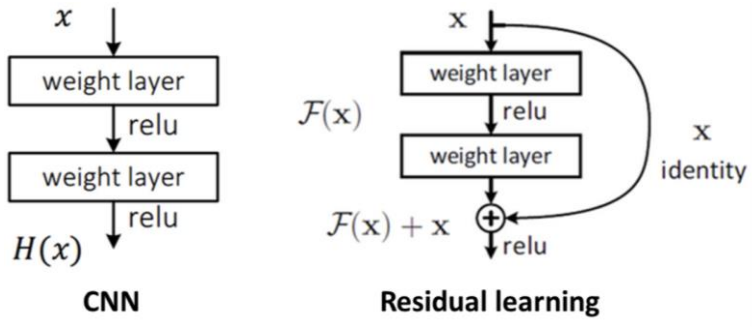
→ Input되는 x가 optimal이 되도록

=가중치가 더 이상 업데이트 되지 않도록

= F(x)가 0이 되도록 학습

→ 추가적인 parameter가 없기 때문에 학습 난이도가 쉽고 (easy to optimize)

네트워크가 깊어질수록 높은 accuracy



Skip-connection
Shortcut-connection

x는 이번 layer의 input이기도 하지만, 이전 layer의 output이라는 점에서, x는 이미 image의 feature를 반영하고 있다.

x를 통해 완전히 새로운 H(x)를 학습하기보다는, x에 있는 feature 정보는 그대로 가져오고 나머지(residual, F(x))만 학습하면 더 빠르고 정확하다 !

4. ResNet

Bottleneck residual block

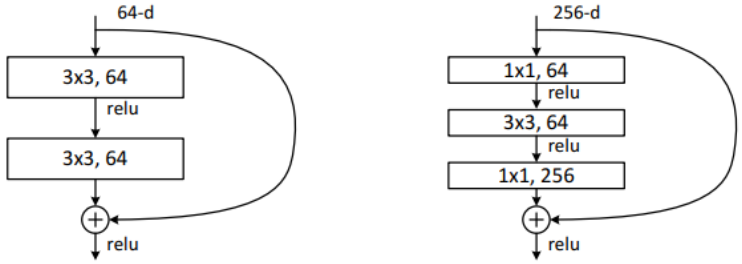


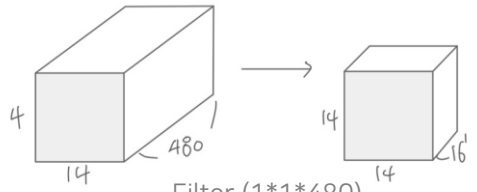
Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

ResNet의 네트워크가 깊어지면(ResNet-50), operation을 줄이기 위해 residual block 변형

Input의 channel이 256이면 1*1*256 conv filter 64개 사용해서 channel을 64로 만들어준 뒤

3*3*64 conv filter를 64개만 사용 (256개 안 써도 됨)

그리고 다시 1*1*64 conv filter 256개 사용해서 channel을 256으로 되돌려준 뒤 skip-connection



4. ResNet

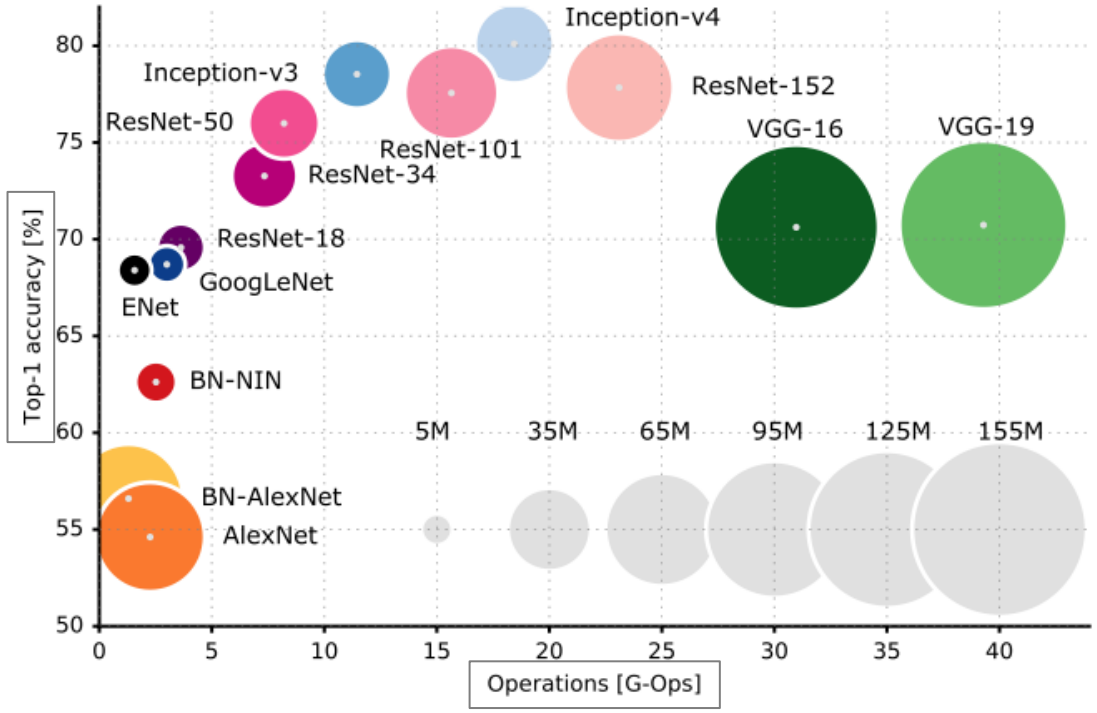
MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: "*Ultra-deep*" (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

ResNet은 Image classification 뿐만 아니라 Detection, Segmentation에서도 압도적인 1위를 기록

4. ResNet

Comparing model complexity



VGGNet : Accuracy에서는 우수하지만 parameter가 많아 큰 메모리 요구, operation이 많아 학습 느림

GoogleNet : Accuracy는 VGG보다 살짝 낮지만, parameter와 operation이 적은 efficient한 모델

ResNet : Accuracy가 굉장히 높은데도 GAP로 fewer parameter를, residual block으로 fewer operation 유지

5. After ResNet

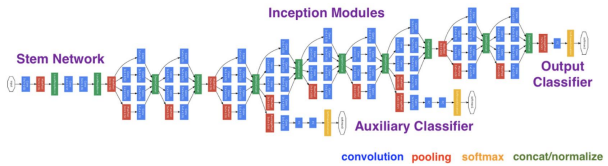
2016 : ResNet era

2016년에는 VGG(2014), GoogleNet(2014), ResNet(2015)처럼 새로운 아이디어가 등장하거나, 성능이 크게 향상된 모델은 없었음

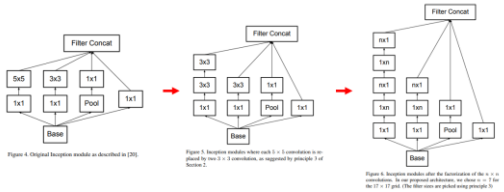
하지만 Inception-v2, v3를 내던 Google팀이 Inception-v4에서

Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning를 발표하면서 Inception에 ResNet을 도입

ILSVRC 2016 우승도 Inception, ResNet 등을 앙상블한 팀이었음

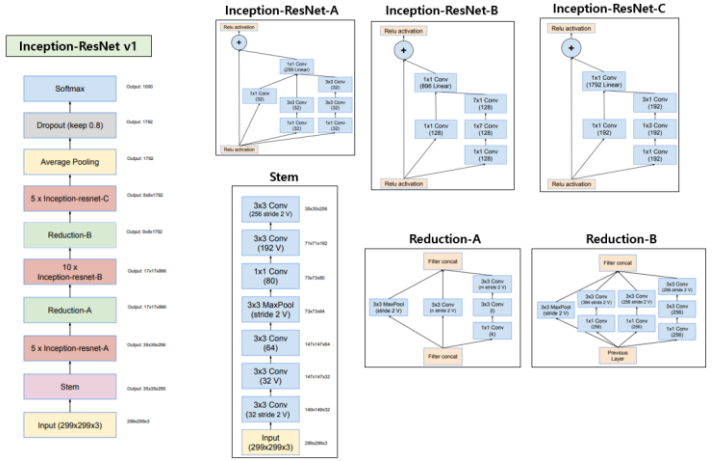


GoogleNet (Inception-v1)



Inception-v2

Inception-v3



Inception-v4

Inception 구조 / Layer 순서 / 정규화 방식 / …

5. After ResNet

2017 : SENet

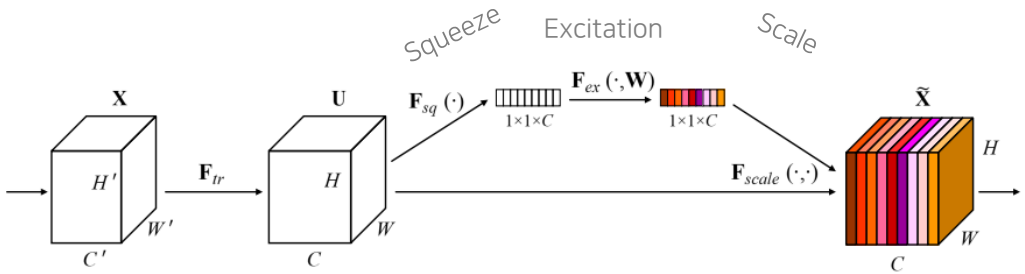
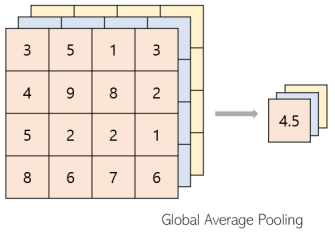


Fig. 1. A Squeeze-and-Excitation block.

Squeeze : Feature map → (Global Average Pooling) → 1*1*C

$$z_c = \mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i,j).$$



Excitation : squeezed 1*1*C → FC layer → ReLU → FC layer → Sigmoid

$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \sigma(\mathbf{W}_2 \delta(\mathbf{W}_1 \mathbf{z})),$$

Scale : channel-wise multiplication

$$\tilde{\mathbf{x}}_c = \mathbf{F}_{scale}(\mathbf{u}_c, s_c) = s_c \, \mathbf{u}_c,$$

5. After ResNet

2017 : SENet

W*H*C feature map을 input으로 받아, squeeze – excitation – scale 해서 W*H*C feature map이 output 으로 나온다

→ 기존CNN architecture (VGGNet, GoogleNet, ResNet, …)에 붙여서 사용할 수 있다

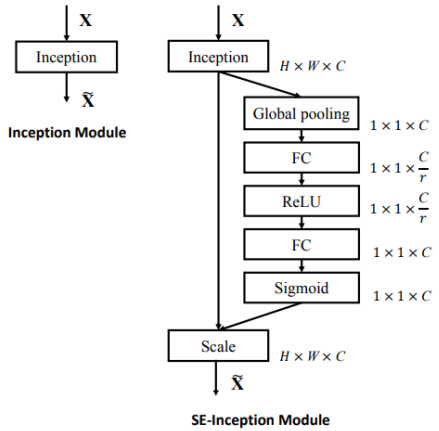


Fig. 2. The schema of the original Inception module (left) and the SE-Inception module (right).

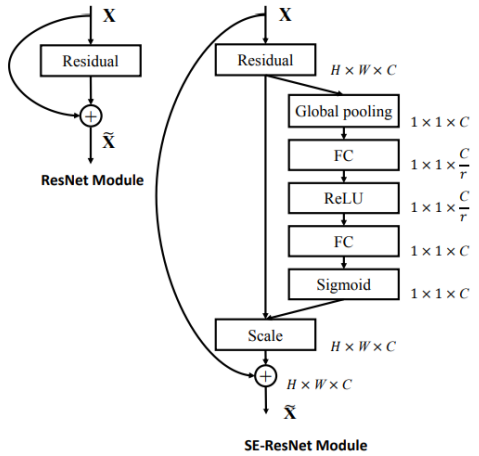
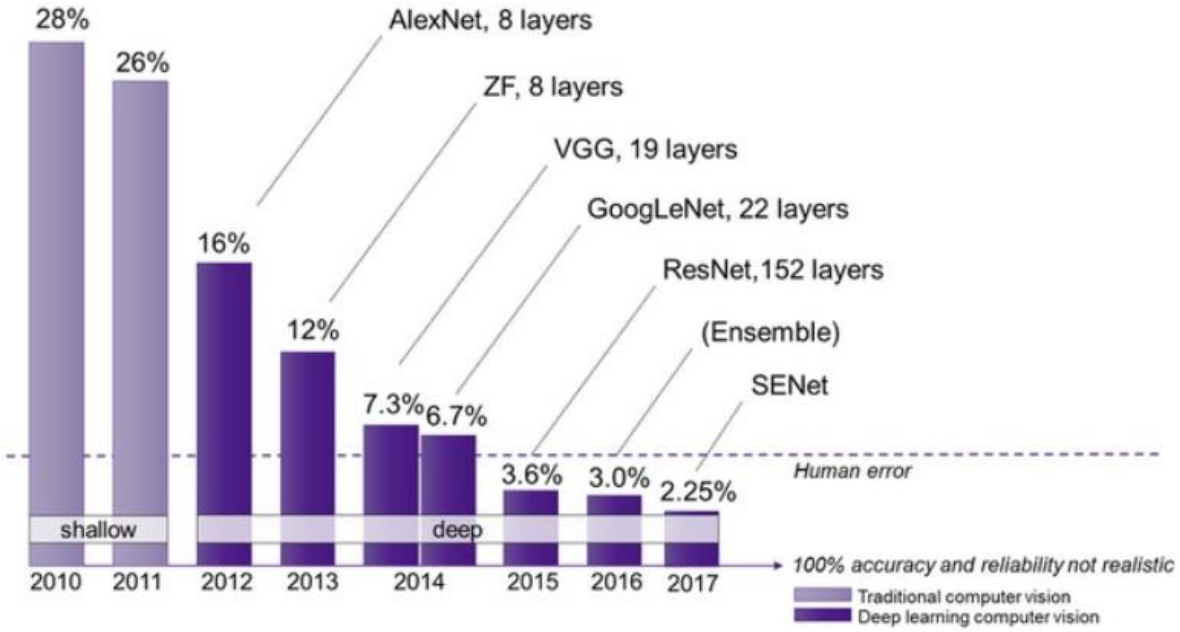


Fig. 3. The schema of the original Residual module (left) and the SE-ResNet module (right).

	original		re-implementation			SENet		
	top-1 err.	top-5 err.	top-1 err.	top-5 err.	GFLOPs	top-1 err.	top-5 err.	GFLOPs
ResNet-50 [13]	24.7	7.8	24.80	7.48	3.86	$23.29_{(1.51)}$	$6.62_{(0.86)}$	3.87
ResNet-101 [13]	23.6	7.1	23.17	6.52	7.58	$22.38_{(0.79)}$	$6.07_{(0.45)}$	7.60
ResNet-152 [13]	23.0	6.7	22.42	6.34	11.30	$21.57_{(0.85)}$	$5.73_{(0.61)}$	11.32
ResNeXt-50 [19]	22.2	-	22.11	5.90	4.24	$21.10_{(1.01)}$	$5.49_{(0.41)}$	4.25
ResNeXt-101 [19]	21.2	5.6	21.18	5.57	7.99	$20.70_{(0.48)}$	$5.01_{(0.56)}$	8.00
VGG-16 [11]	-	-	27.02	8.81	15.47	$25.22_{(1.80)}$	$7.70_{(1.11)}$	15.48
BN-Inception [6]	25.2	7.82	25.38	7.89	2.03	$24.23_{(1.15)}$	$7.14_{(0.75)}$	2.04
Inception-ResNet-v2 [21]	$19.9^{\dagger}$	$4.9^{\dagger}$	20.37	5.21	11.75	$19.80_{(0.57)}$	$4.79_{(0.42)}$	11.76

5. After ResNet



6. Summary

1. ReLU를 처음 사용한 AlexNet (2012)을 시작으로 CNN에 많은 관심이 집중되기 시작했다.
2. VGGNet은 3*3 conv filter를 사용하여 네트워크의 깊이 깊어질수록 성능이 좋아지는지 확인하였다.

   → 네트워크가 깊다고 해서 항상 error가 낮아지는 것은 아니었다.
3. GoogleNet은 Inception module을 통해 다양한 feature를 extract 하였고,

   1*1 conv filter와 Global Average Pooling으로 Complexity(parameters, operation)를 완화하였다.
4. ResNet은 Residual block에서 x를 입력 받아 H(x)를 출력하는 함수 H를 학습하는 대신, F(x) = H(x)-x를 학습하였다.

   VGGNet의 3*3 conv filter, 그리고 GoogleNet의 1*1 conv filter와 GAP를 적절히 활용하여 성능을 크게 향상시켰다.
5. ResNet이 등장한 이후, ResNet을 기반으로 한 다양한 모델들이 등장했고, 좋은 성능을 보여주었다.

   Ex. ResNext, DenseNet, Residual Attention Network, …
6. 2018년 이후에는 CNN 모델의 경량화를 다룬 많은 연구가 이루어졌다.

   Ex. ShuffleNet, MobileNet, …

6. Summary

Reference

Michigan Univ. <Deep Learning for Computer Vision>

-Lecture 7 : Convolutional Networks

-Lecture 8 : CNN Architectures

Stanford Univ. <Convolutional Neural Networks for Visual Recognition>

-Lecture 9 : CNN Architectures

https://hoya012.github.io/blog/deeplearning-classification-guidebook-1/

Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner (1998) *Gradient-Based Learning Applied to Document Recognition*

Alex Krizhevsky , Ilya Sutskever, and Geoffrey E. Hinton (2017) *ImageNet Classification with Deep Convolutional Neural Networks*

Visual Geometry Group, Department of Engineering Science, University of Oxford (2015) *VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION*

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich (2015) *Going deeper with convolutions*

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun (2015) *Deep Residual Learning for Image Recognition*

Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke (2016) *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*

Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu (2018) *Squeeze-and-Excitation Networks*

DATA SCIENCE LAB

발표자 정건우 010-6473-3938
E-mail: wjdrjsdn3938@naver.com