19기 정규세션

ToBig's 18기 이기정

Vision Advanced

Contents

Unit 01 I Convolutional Neural Networks Review

Convolutions

지난시간 Review

- 합성곱의 개념
- Stride
- Max Pooling
- Padding

…

Why Convolutions?

Convolution을 사용하면,

👉 **변수를 적게 사용할 수 있다.**

Ex) 32 x 32 x 3 이미지를 5 x 5 filter 6개를 통해 28 x 28 x 6 의 이미지로 합성곱 연산을 했을 경우, 필요한 변수의 개수는 5 x 5 x 3 x 6 + 6 = 456개가 필요하다. 하지만 일반적인 신경망으로는 3,072 x 4,704 + 4,704, 약 1400 만 개의 변수가 필요하다.

👉 **변수 공유**

👉 **희소 연결**

👉 **이동 불변성 포착하는데 용이**

Unit 02 | Case Studies

Classic networks

- 👉 LeNet-5
- 👉 AlexNet
- 👉 VGG

GoogleNet(Inception Module)

ResNet

Unit 02 I Case Studies

Classic networks

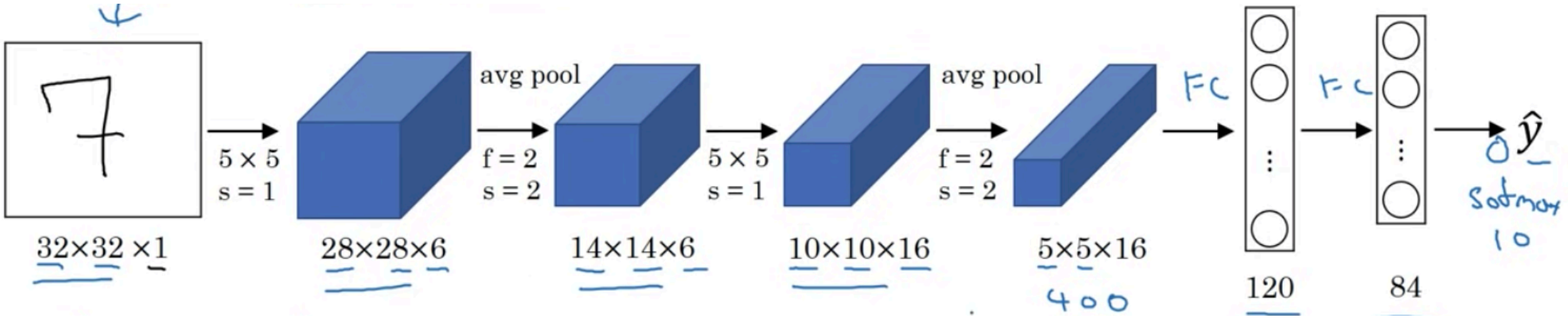
- 👉 LeNet-5
- 👉 AlexNet
- 👉 VGG

GoogleNet(Inception Module)

ResNet

ImageNet Classification Challenge

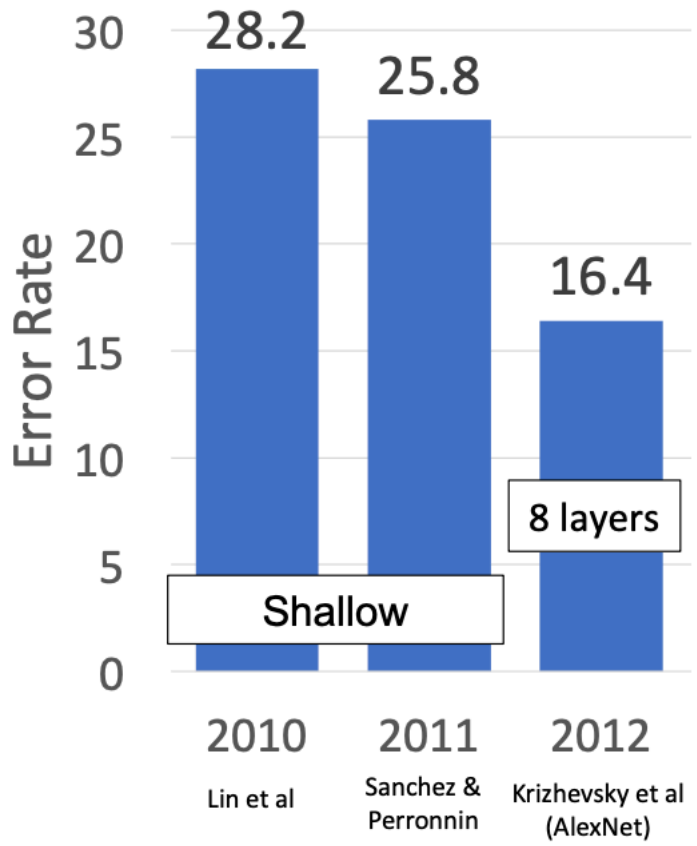
LeNet -5



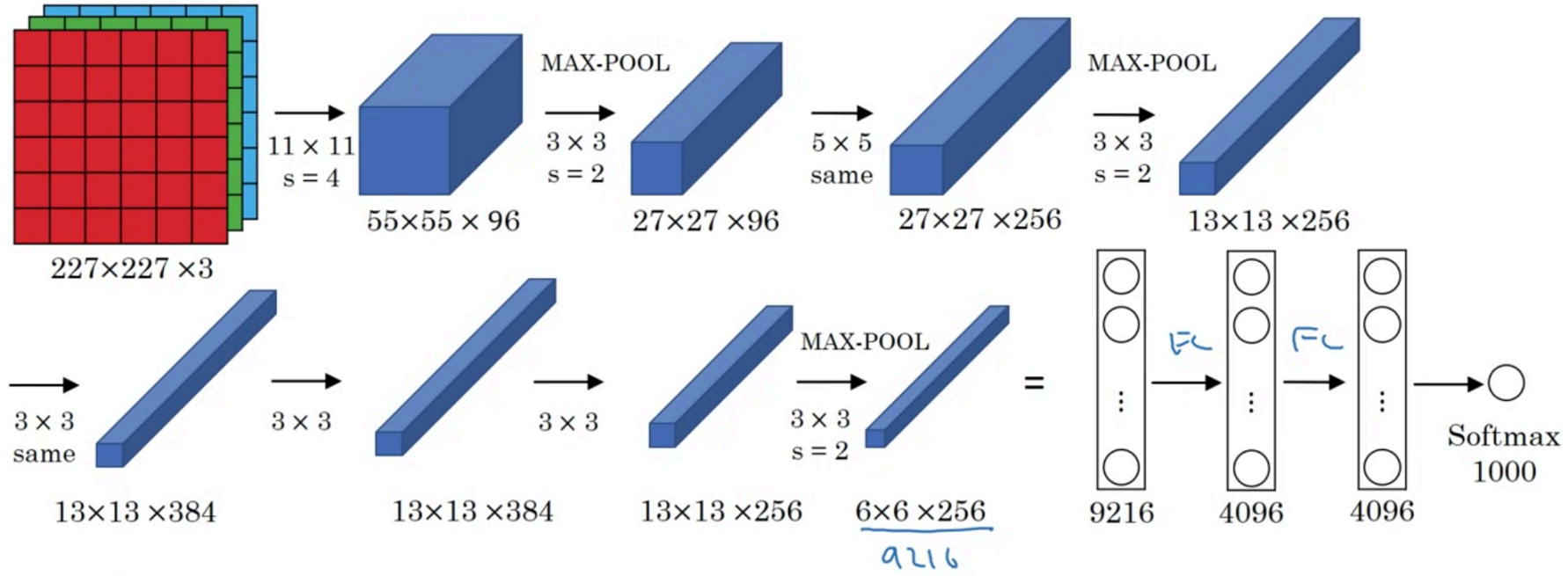
👉 목적: 흑백으로 된 손글씨 인식

👉 옛날 논문이라 상대적으로 변수 적고, pooling층 뒤에 비선형함수 적용 비선형함수도 ReLU가 아닌 Sigmoid

ImageNet Classification Challenge



AlexNet 👉 목적 : 이미지를 1000개에 해당하는 클래스로 분류



AlexNet

	Input size		Layer				Output size	
Layer	C	H / W	filters	kernel	stride	pad	C	H / W
conv1	3	227	64	11	4	2	?	

AlexNet

	Input size		Layer				Output size	
Layer	C	H / W	filters	kernel	stride	pad	C	H / W
conv1	3	227	64	11	4	2	64	?

Output channels = number of filters

AlexNet

	Input size		Layer				Output size	
Layer	C	H / W	filters	kernel	stride	pad	C	H / W
conv1	3	227	64	11	4	2	64	56

Unit 02 l Case Studies – (1) Classic networks (AlexNet)

AlexNet

	Input size		Layer				Output size		
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)
conv1	3	227	64	11	4	2	64	56	?

AlexNet

	Input size		Layer				Output size		
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)
conv1	3	227	64	11	4	2	64	56	784

Number of output elements = C * H' * W'
= 64*56*56 = 200,704

Bytes per element = 4 (for 32-bit floating point)

KB = (number of elements) * (bytes per elem) / 1024
= 200704 * 4 / 1024
= **784**

Unit 02 l Case Studies – (1) Classic networks (AlexNet)

AlexNet

	Input size		Layer				Output size			
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)
conv1	3	227	64	11	4	2	64	56	784	?

AlexNet

	Input size		Layer				Output size			
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)
conv1	3	227	64	11	4	2	64	56	784	23

Weight shape = C_{out} x C_{in} x K x K

= 64 x 3 x 11 x 11

Bias shape = C_{out} = 64

Number of weights = 64*3*11*11 + 64

= **23,296**

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	?

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

Number of floating point operations (multiply+add)

= (number of output elements) * (ops per output elem)

= (C_{out} x H' x W') * (C_{in} x K x K)

= (64 * 56 * 56) * (3 * 11 * 11)

= 200,704 * 363

= **72,855,552**

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		?			

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			

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	?	

Unit 02 l Case Studies – (1) Classic networks (AlexNet)

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	?

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

Floating-point ops for pooling layer
= (number of output positions) * (flops per output position)
= (C_{out} * H' * W') * (K * K)
= (64 * 27 * 27) * (3 * 3)
= 419,904
= **0.4 MFLOP**

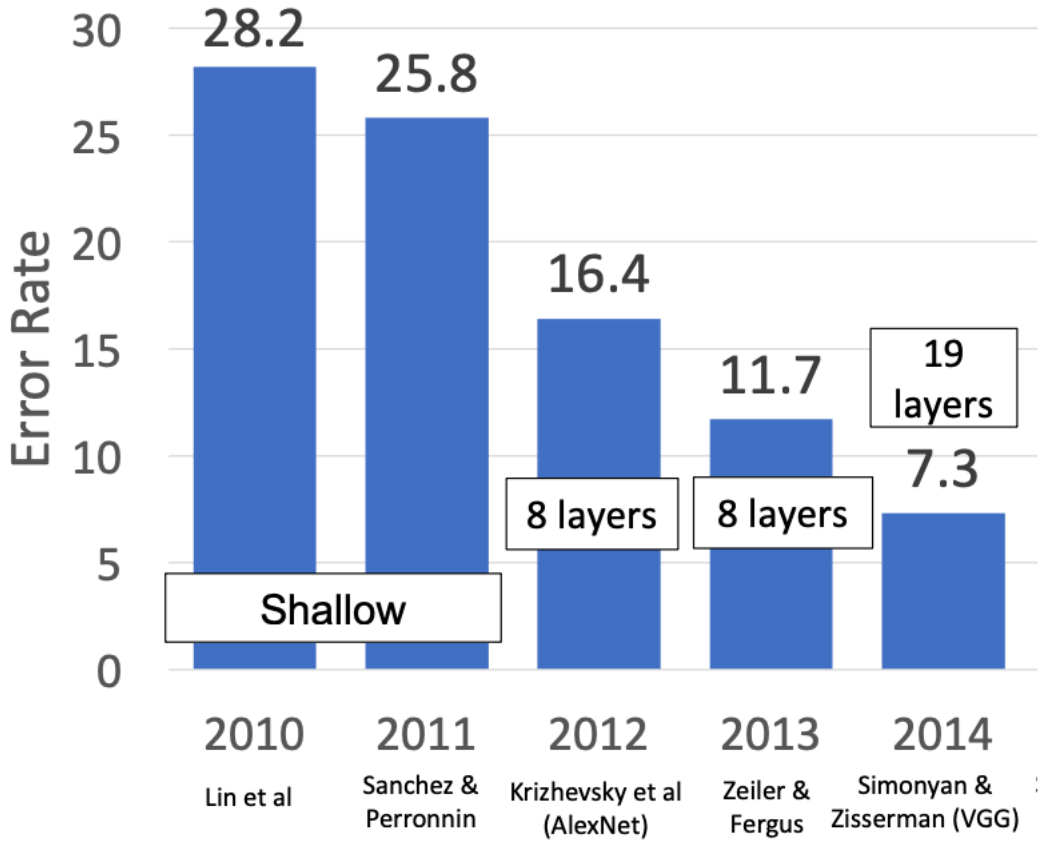
AlexNet

	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

AlexNet

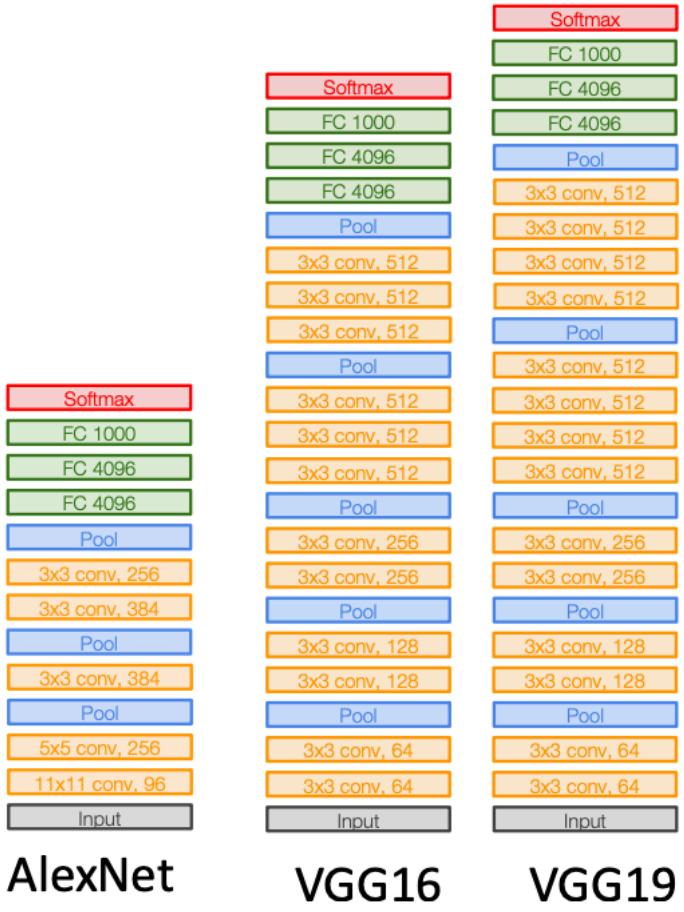
	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664	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

ImageNet Classification Challenge



Unit 02 l Case Studies - (1) Classic networks (VGG -16)

VGG -16



👉 목적: CNN의 깊이가 image classification 에

어떤 영향을 끼치는지 알아내기 위함

👉 모든 합성곱 연산은

Filter 3x3, Padding 2, Stride 1,

Max Pooling 2 x 2 픽셀씩

👉 산출값의 높이와 넓이는 각 max Pooling마다

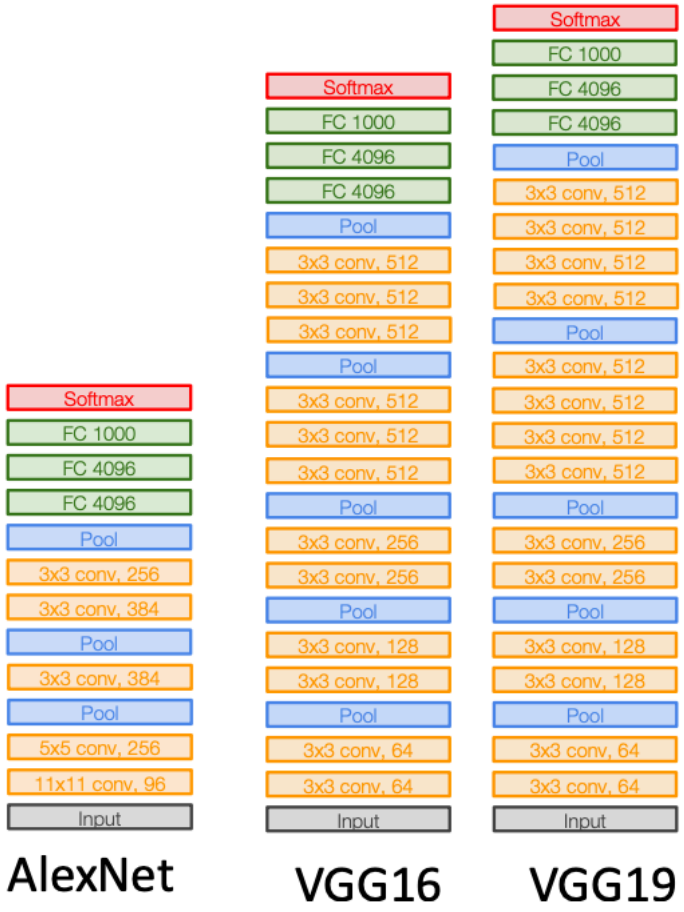
1/2씩 줄어들며, Channel의 수는 두 배 혹은 세 배로

늘어난다. (간결함, 수학적으로 깔끔)

👉 훈련시킬 변수의 개수가 많아

네트워크의 크기가 커진다는 단점

VGG -16



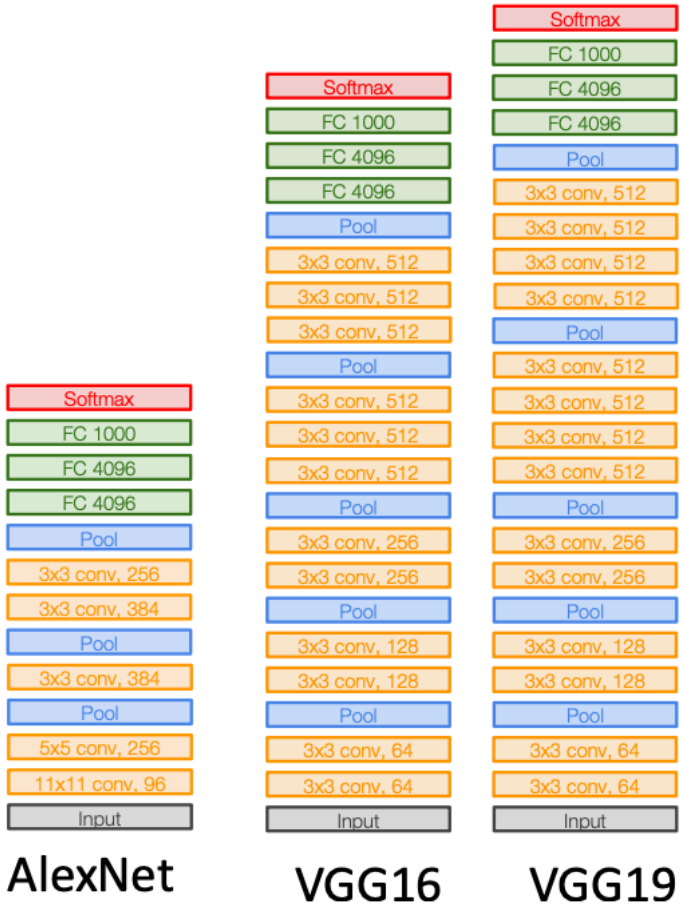
Option 1:
Conv(5x5, C -> C)

Params: $25C^2$
FLOPs: $25C^2HW$

Option 2:
Conv(3x3, C -> C)
Conv(3x3, C -> C)

Params: $18C^2$
FLOPs: $18C^2HW$

VGG -16



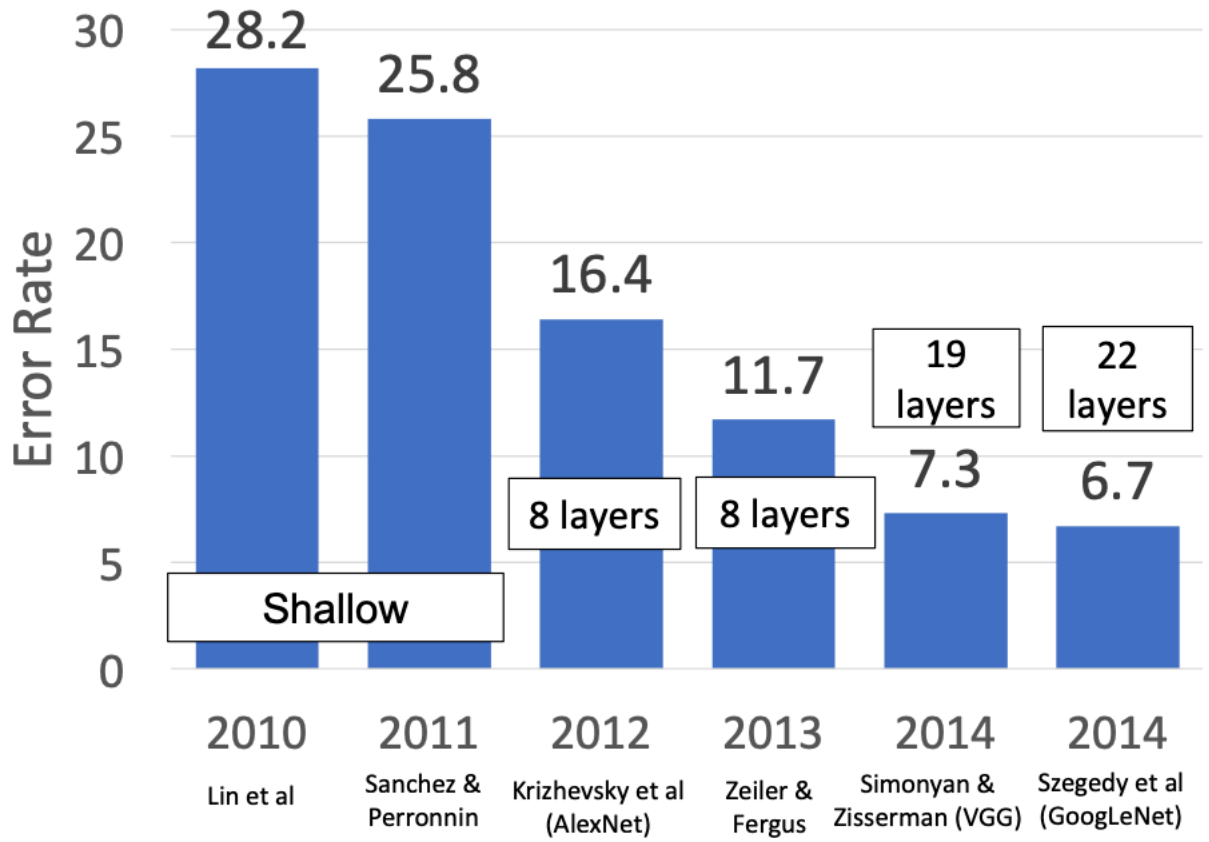
Input: C x 2H x 2W
Layer: Conv(3x3, C->C)

Memory: 4HWC
Params: $9C^2$
FLOPs: $36HWC^2$

Input: 2C x H x W
Conv(3x3, 2C -> 2C)

Memory: 2HWC
Params: $36C^2$
FLOPs: $36HWC^2$

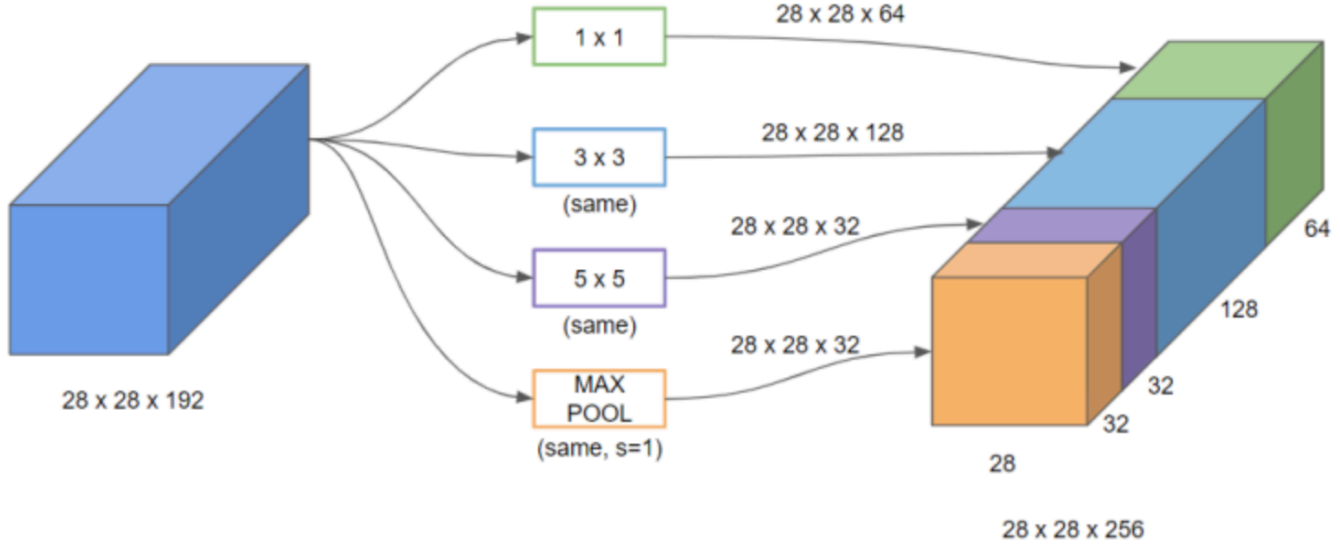
ImageNet Classification Challenge



Inception Network Motivation

👉 Filter의 크기나 pooling을 결정하는 대신 전부다 적용해서 출력들을 합친뒤 네트워크로

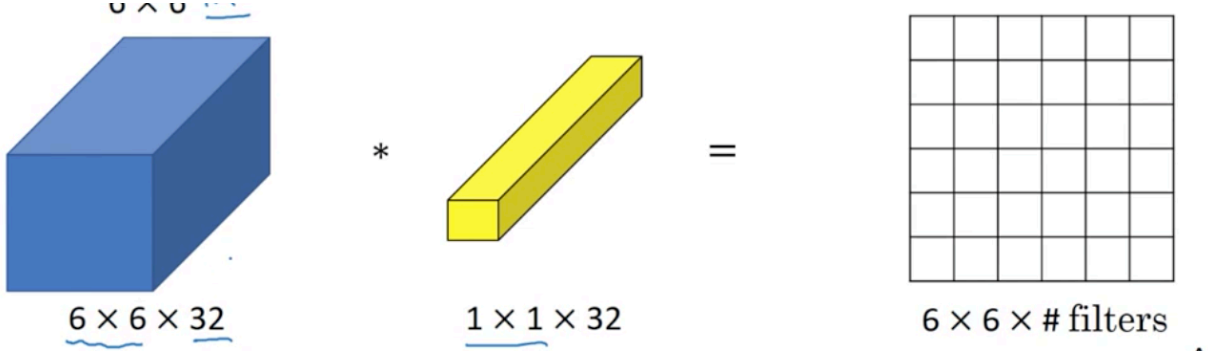
하여금 스스로 변수나 Filter 크기의 조합을 학습하게 만드는 것



계산 비용이 너무 크다! > 이를 해결하기 위해 1x1 합성곱 사용

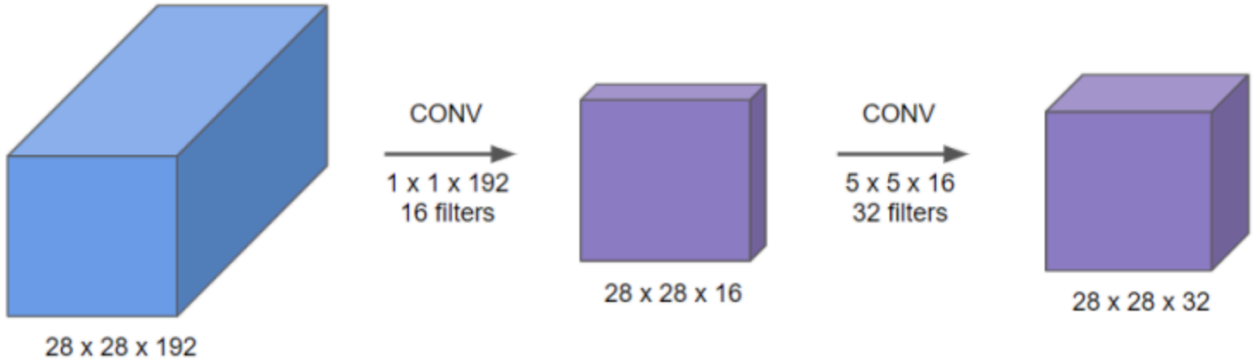
Network In Network

숫자 하나 곱하는 거 아닌가?



비선형성을 하나 더 추가해 복잡한 함수 학습 가능!
채널 수 조절 가능!

Unit 02 l Case Studies - (2) Inception



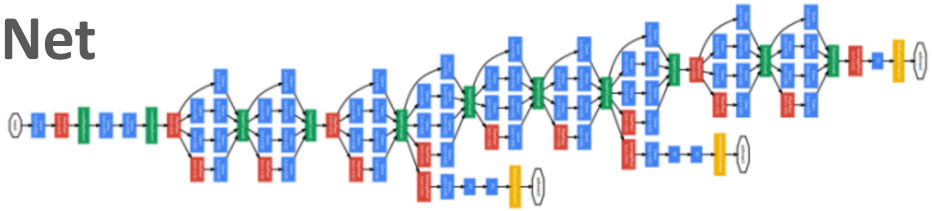
계산 비용이 1/10수준으로 줄어든다!

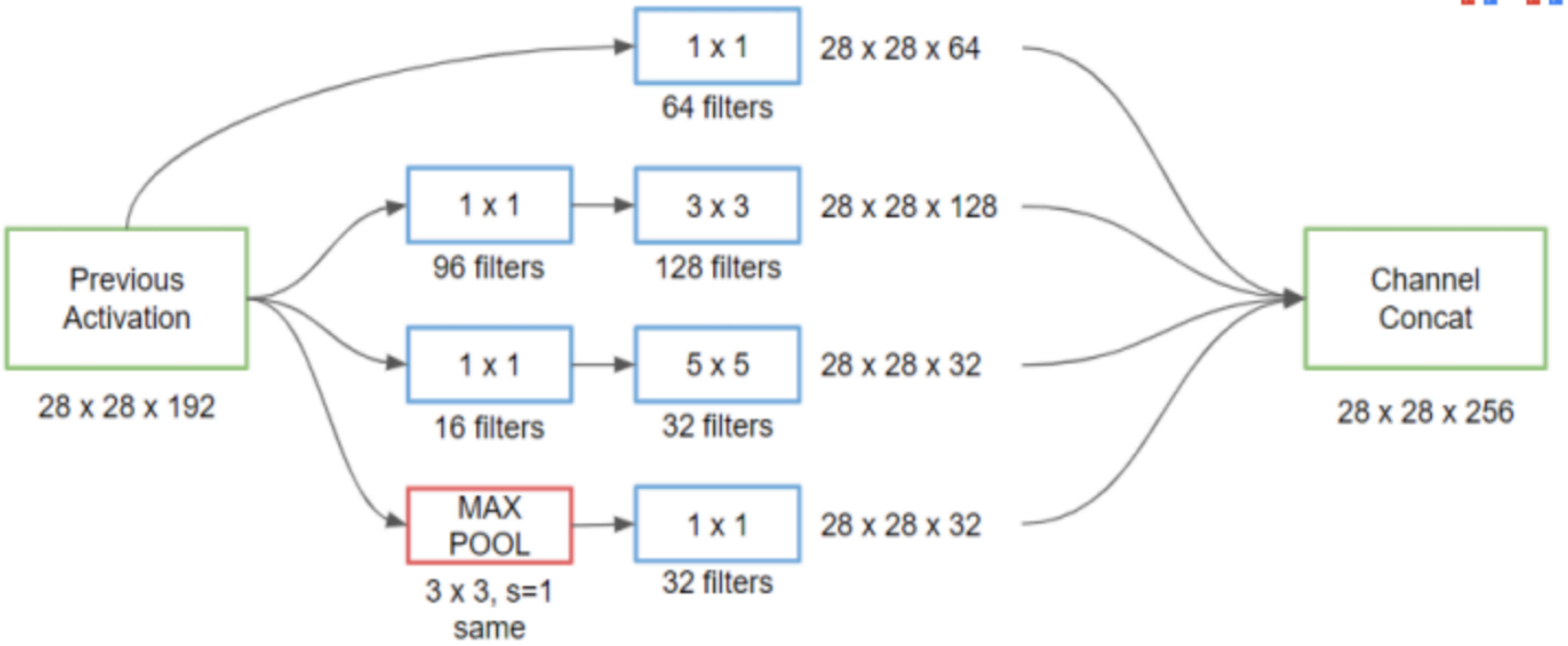
여기서 사용된 1x1합성곱 층을 “Bottle neck layer”라고 부른다.

적절히 구현하면 성능에 큰 지장없이 많은 수의 계산 줄일 수 있다.

Inception Module

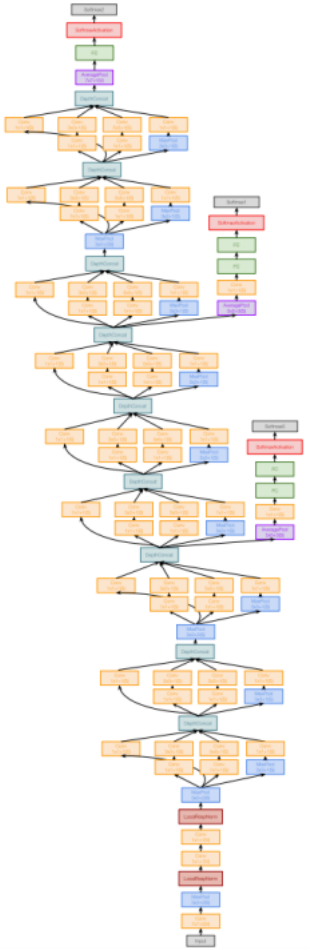
Inception Network = GoogleNet





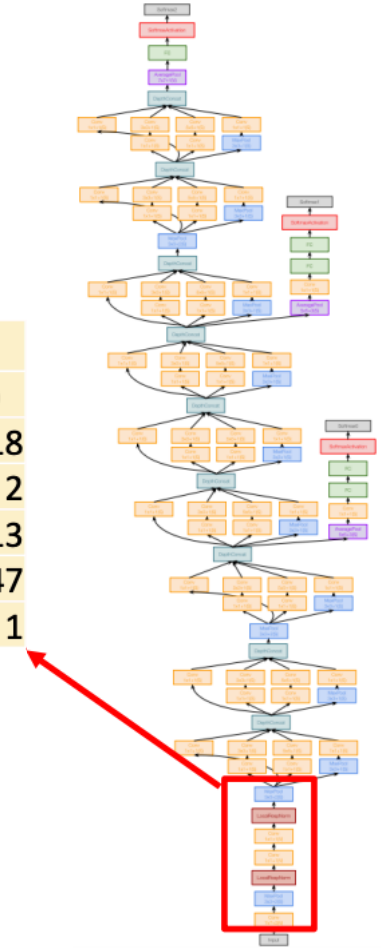
Unit 02 l Case Studies - (2) Inception

GoogLeNet

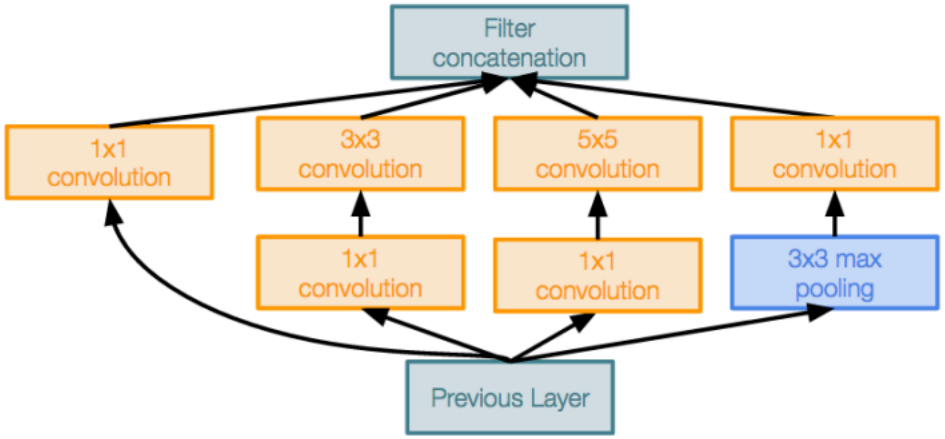


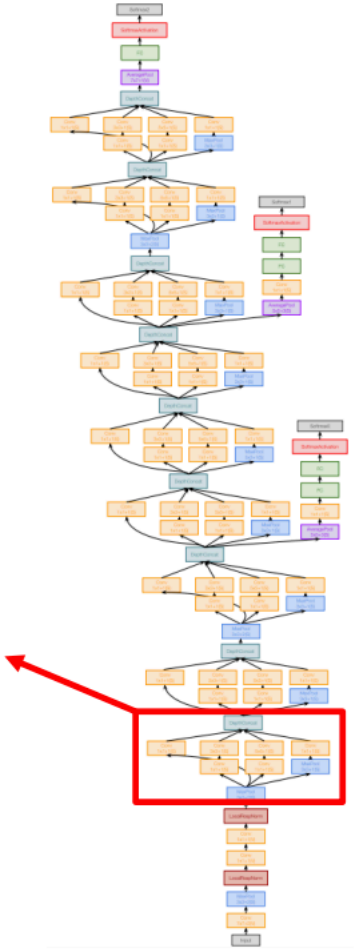
GoogLeNet

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (K)	flop (M)
conv	3	224	64	7	2	3	64	112	3136	9	118
max-pool	64	112		3	2	1	64	56	784	0	2
conv	64	56	64	1	1	0	64	56	784	4	13
conv	64	56	192	3	1	1	192	56	2352	111	347
max-pool	192	56		3	2	1	192	28	588	0	1



GoogLeNet: Inception Module



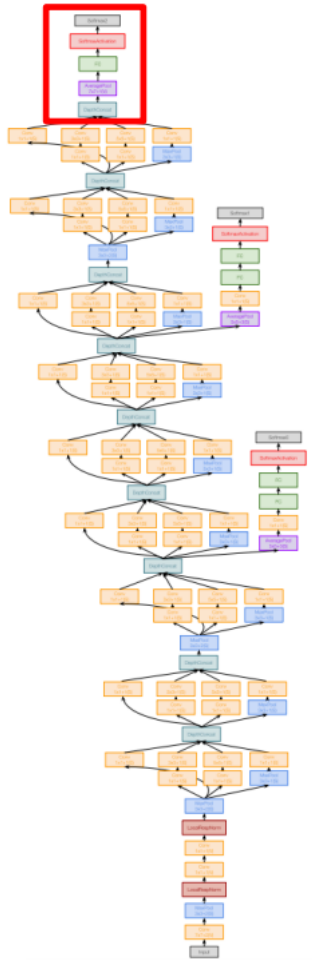


GoogLeNet: Global Average Pooling

	Input size		Layer				Output size				
Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (k)	flop (M)
avg-pool	1024	7		7	1	0	1024	1	4	0	0
fc	1024		1000				1000		0	1025	1

Compare with VGG-16:

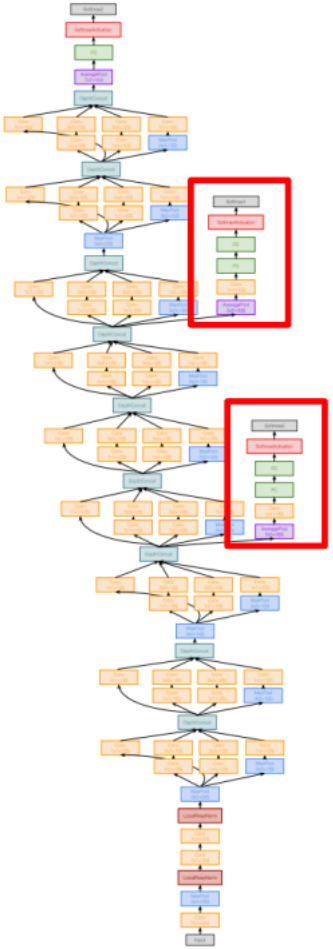
Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (K)	flop (M)
flatten	512	7					25088		98		
fc6	25088			4096			4096		16	102760	103
fc7	4096			4096			4096		16	16777	17
fc8	4096			1000			1000		4	4096	4



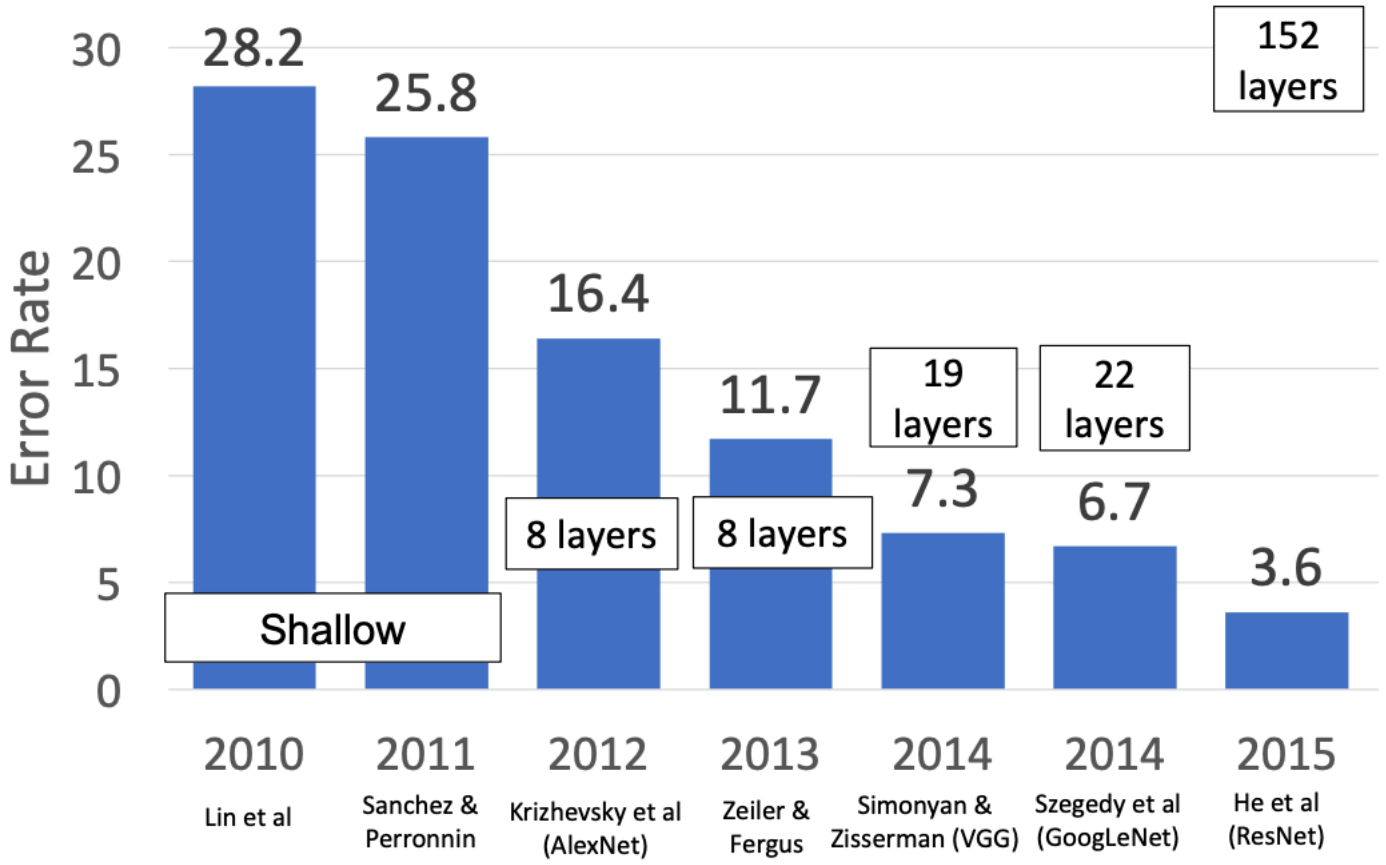
GoogLeNet: Auxiliary Classifiers

네트워크가 너무 깊어서 gradients가 잘 전달안된다.

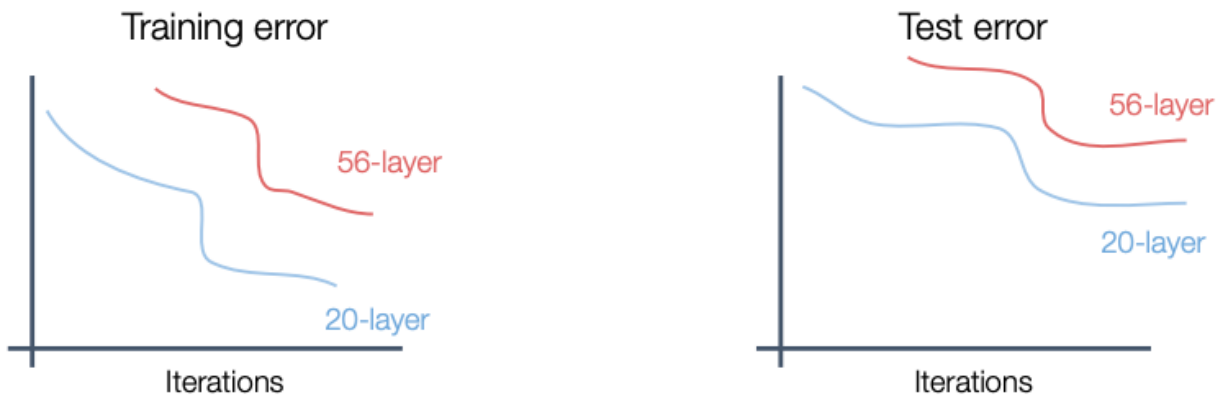
-> “auxiliary classifers”달아서 해결



ImageNet Classification Challenge



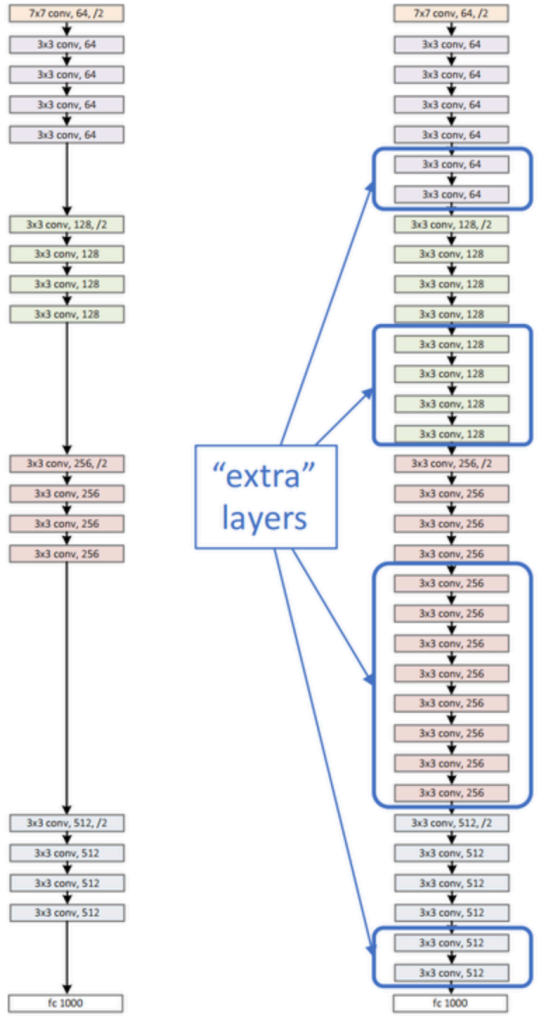
Residual Networks



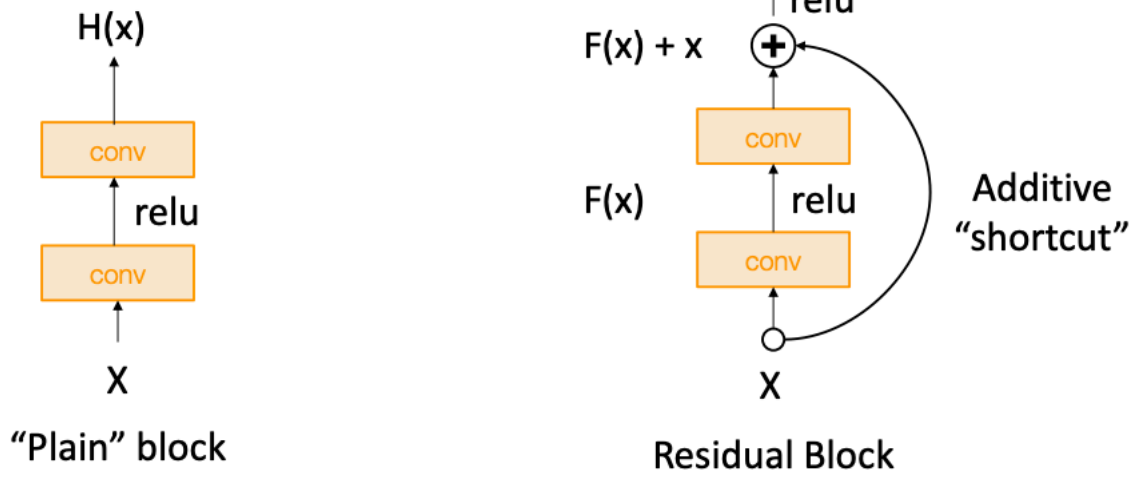
가설: 최적화 문제. 모델이 깊어질수록 최적화가 어렵기 때문에 underfitting발생,

특히 indentity function을 학습하기 어렵다.

Identity function 학습 이유?

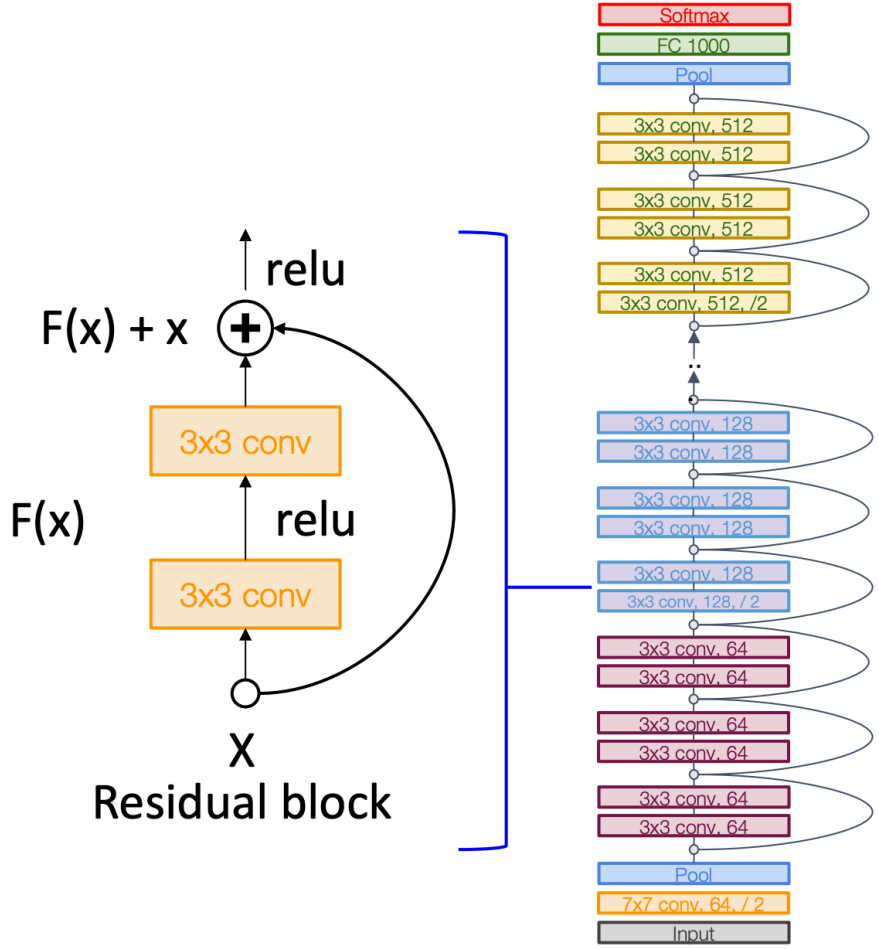


Shortcut or Skip connection



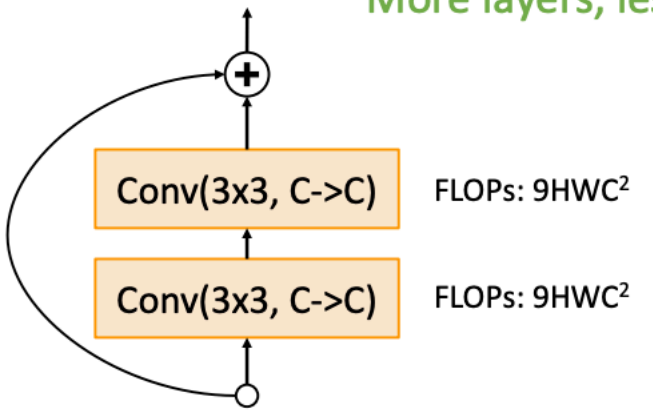
H(x) = x로 학습시키는 것(identity function)이 어렵기 때문에 "shorcut"을 추가해 F(x)=H(x)-x이 되는 것을 학습시키자.

Residual Networks



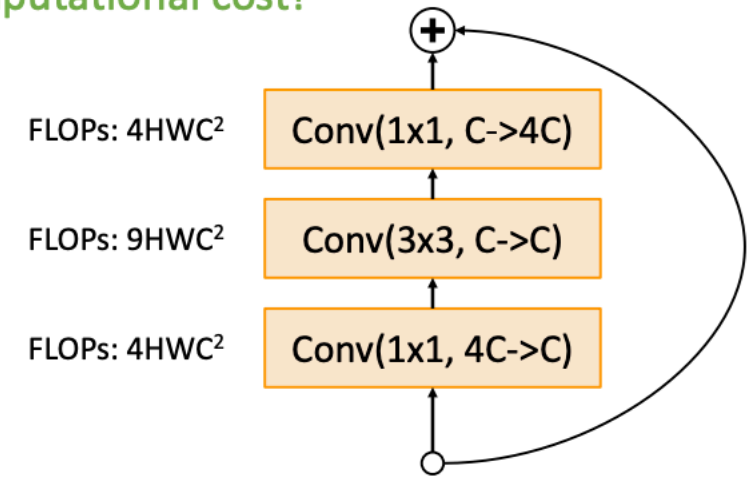
Residual Networks: Bottleneck Block

More layers, less computational cost!



"Basic" Residual block

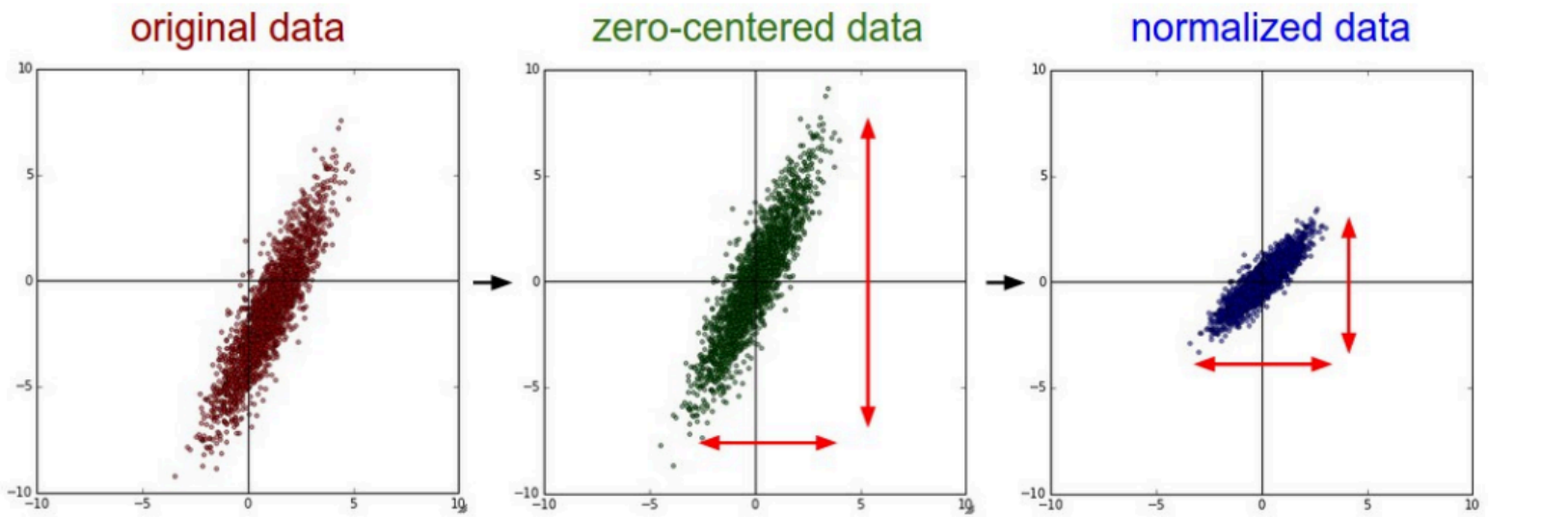
Total FLOPs: $18HWC^2$



Total FLOPs: $17HWC^2$

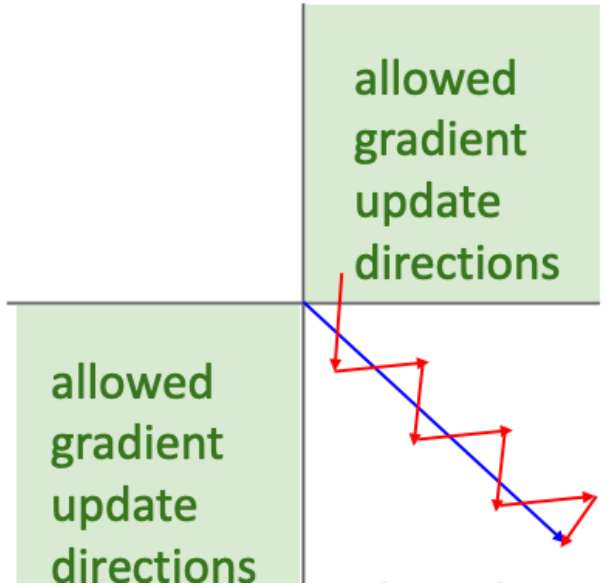
"Bottleneck" Residual block

Data Preprocessing



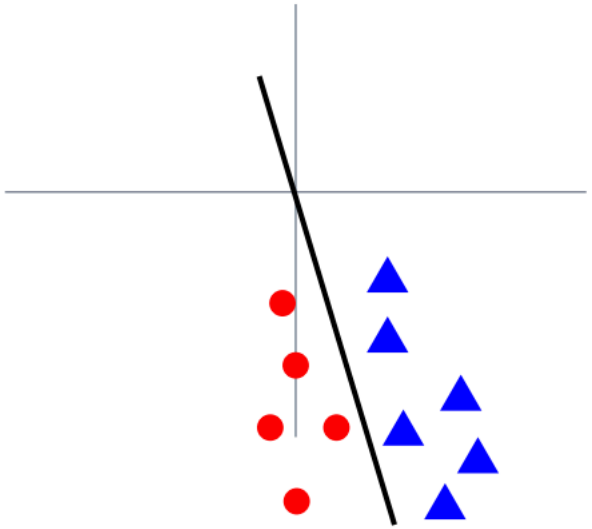
Why?

$$f\left(\sum_i w_i x_i + b\right)$$

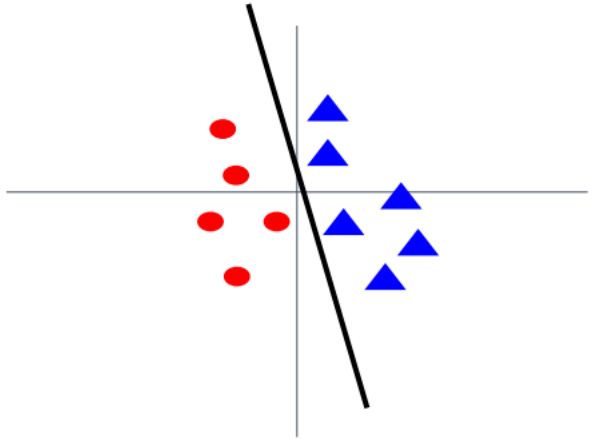


Why?

Before normalization: classification loss very sensitive to changes in weight matrix; hard to optimize



After normalization: less sensitive to small changes in weights; easier to optimize

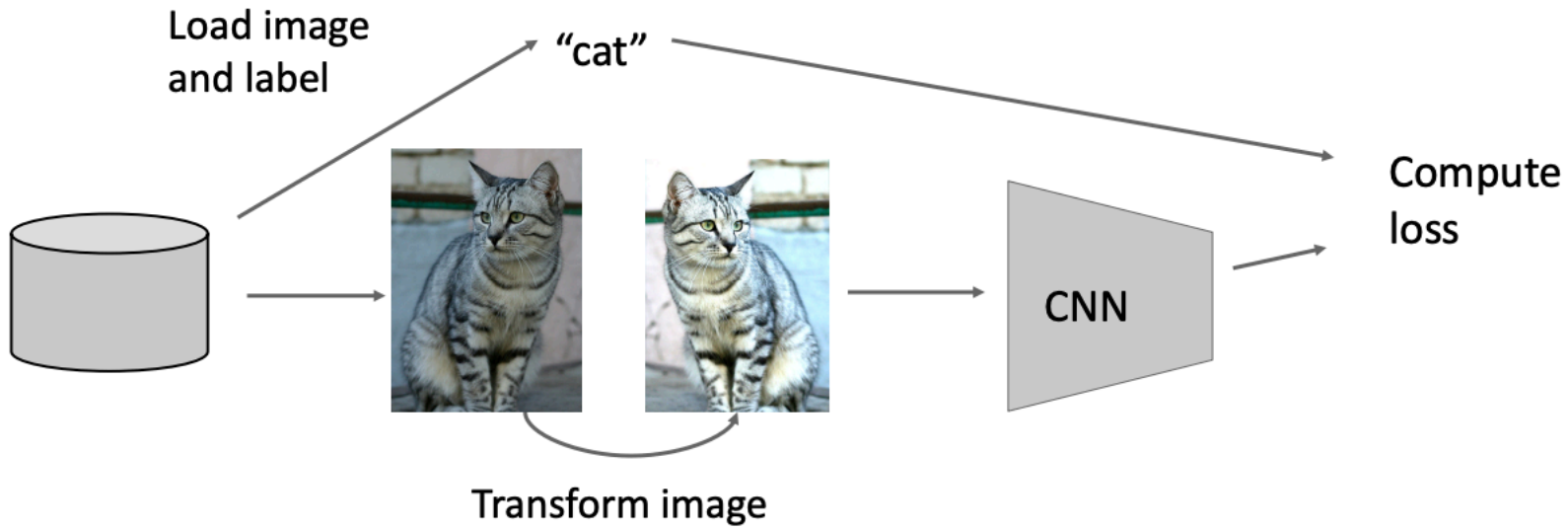


Data Preprocessing for Images

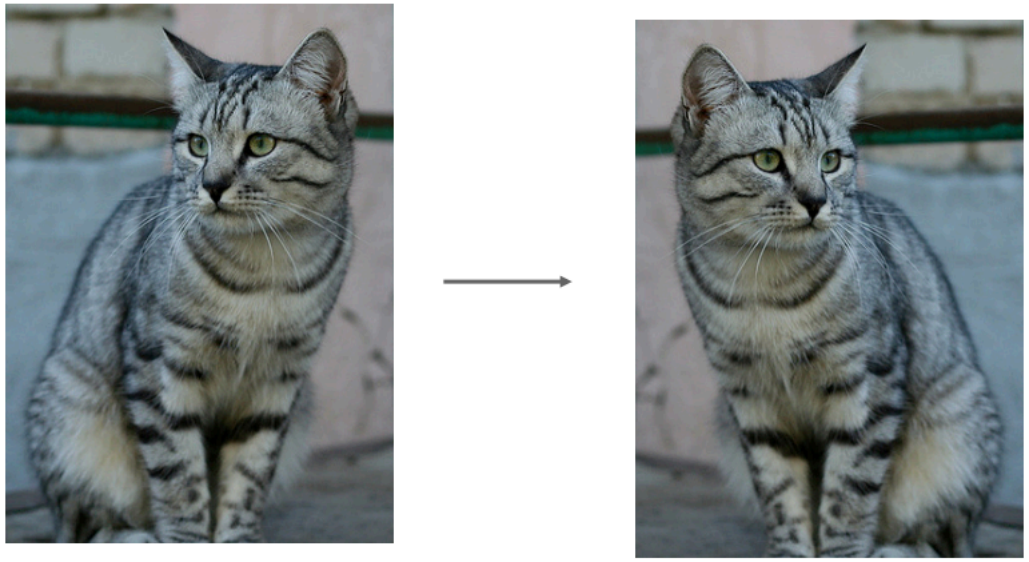
e.g. consider CIFAR-10 example with [32,32,3] images

- Subtract the mean image (e.g. AlexNet)
  (mean image = [32,32,3] array)
- Subtract per-channel mean (e.g. VGGNet)
  (mean along each channel = 3 numbers)
- Subtract per-channel mean and
  Divide by per-channel std (e.g. ResNet)
  (mean along each channel = 3 numbers)

Data Augmentation



Data Augmentation: Horizontal Flips



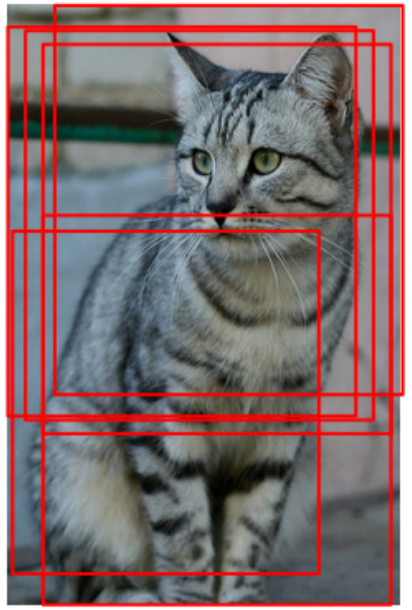
Data Augmentation: Random Crops and Scales

Training:

1. [256, 480] 범위 안에서 랜덤하게 하나의 size 뽑는다.
2. 뽑힌 size로 이미지를 resize 시킨다.
3. 224x224 패치를 랜덤하게 샘플링한다.

Testing:

1. 고정된 5개의 scale(224, 256, 384, 480, 640)로 이미지를 resize 시킨다.
2. 각 사이즈에서, 224x224 패치를 10개 뽑는다.(4 corners + 1 center)*2 flips
3. 샘플링된 50개의 패치를 모델에 넣어서 예측값을 평균낸다.

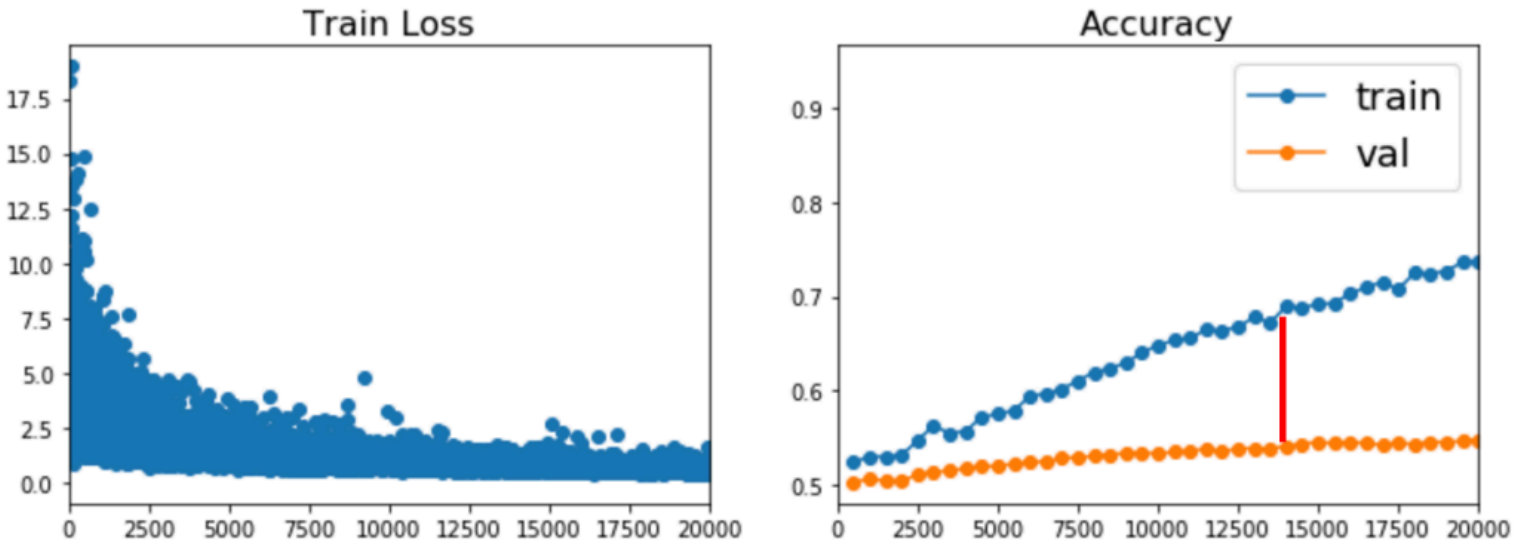


Data Augmentation

Random mix/combinations of :

- translation
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

Unit 03 I Training Neural Networks - (3) Regularization



Regularization

Regularization: Add term to the loss

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

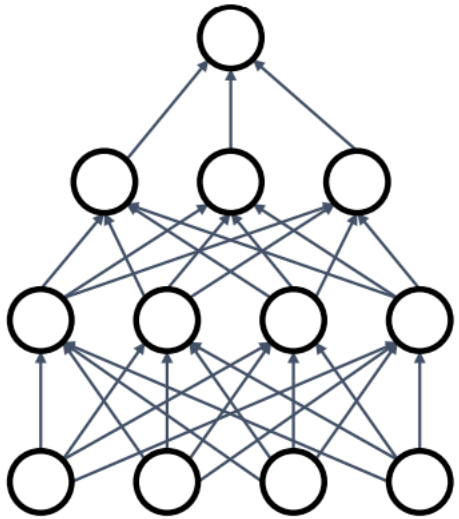
L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay)

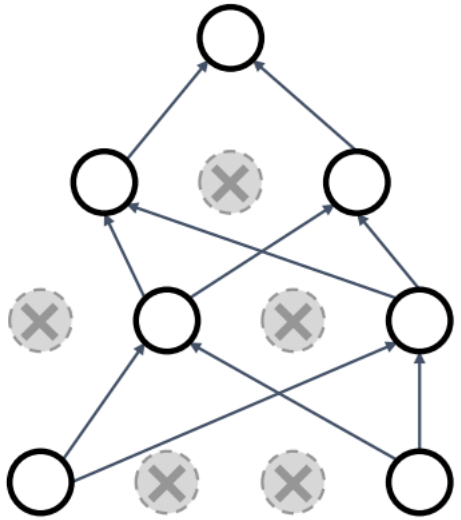
L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Regularization: Dropout

각 forward pass에서, 랜덤하게 몇몇 neurons의 값을 0으로 dropping할 확률을 설정해줍니다.

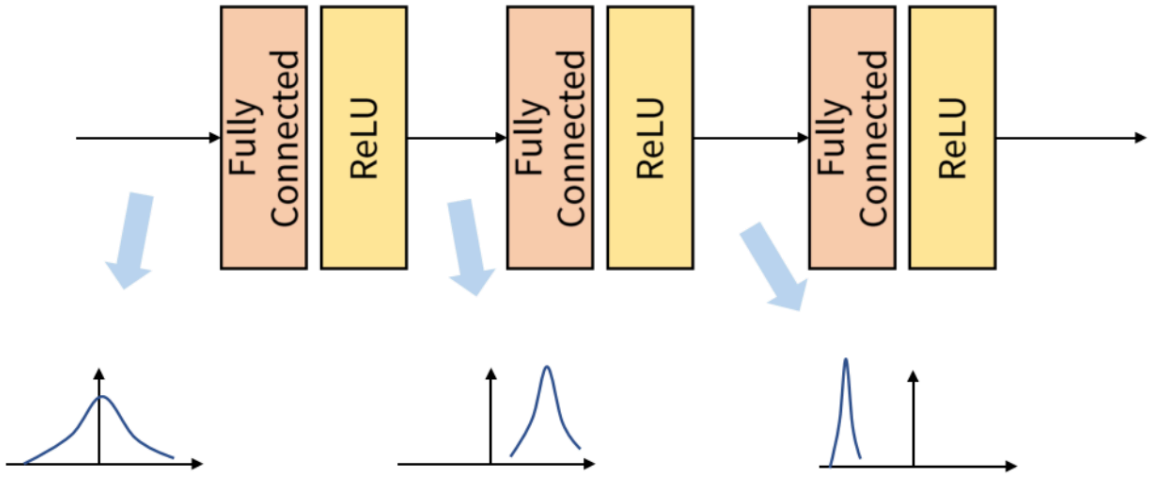




Batch Normalization

Batch 단위로 학습하기 때문에 학습 과정에서 계층 별로 입력의 데이터 분포가 달라지는 현상인

Internal Covariant Shift 문제가 발생 -> batch단위로 normalization 함으로써 해결



Batch Normalization

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

Per-channel std, shape is D

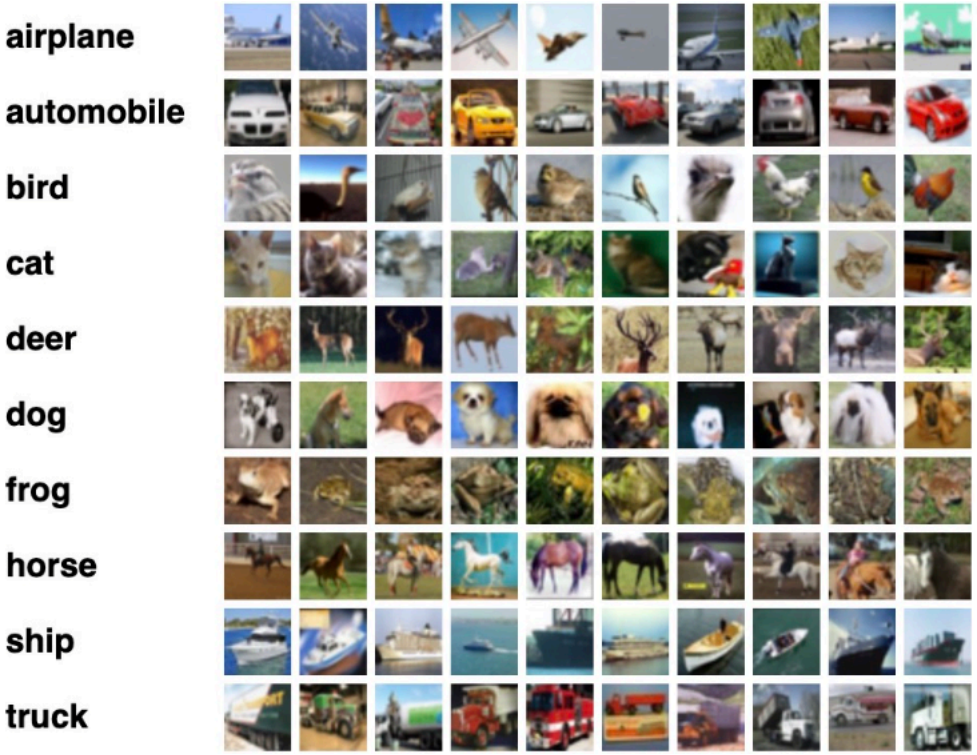
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

Unit 04 l Assignments



과제 - Cifar100 데이터셋을 분류

Validation set에 대한 정확도 최소 60%이상 도출

참고자료

•https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (ImageNet Classification with Deep Convolutional Neural Network)

•https://bkshin.tistory.com/entry/%EB%85%BC%EB%AC%B8-%EB%A6%AC%EB%B7%B0-YOLOYou-Only-Look-Once

• https://www.youtube.com/watch?v=IGbQIr1Ts7w&list=PL5-TkQAfAZFbzxjBHtzdVCWE0Zbhomg7r&index=10(미시간 주립 대학교 강의자료)

• Tobigs 16기 김경민님 강의자료

들어주셔서 감사합니다.