BE530 – Medical Deep Learning

– Representative CNNs –

Byoung-Dai Lee

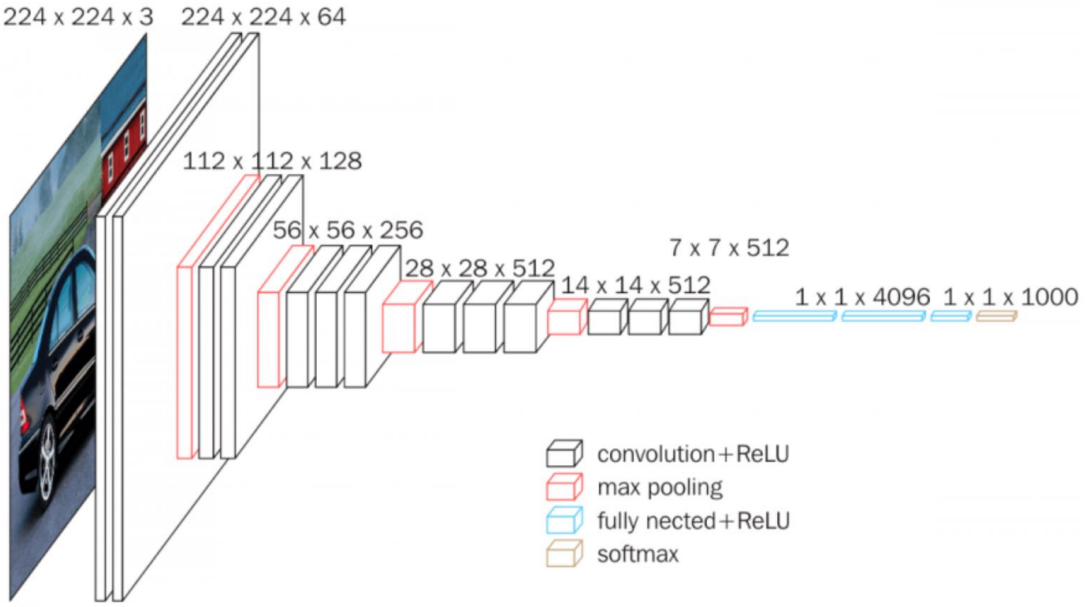
Division of AI Computer Science and Engineering

Kyonggi University

KYONGGI UNIVERSITY

VGGNet (1)

- 2nd place in ILSVRC
  - 7.3% top 5 error in ILSVRC 2014

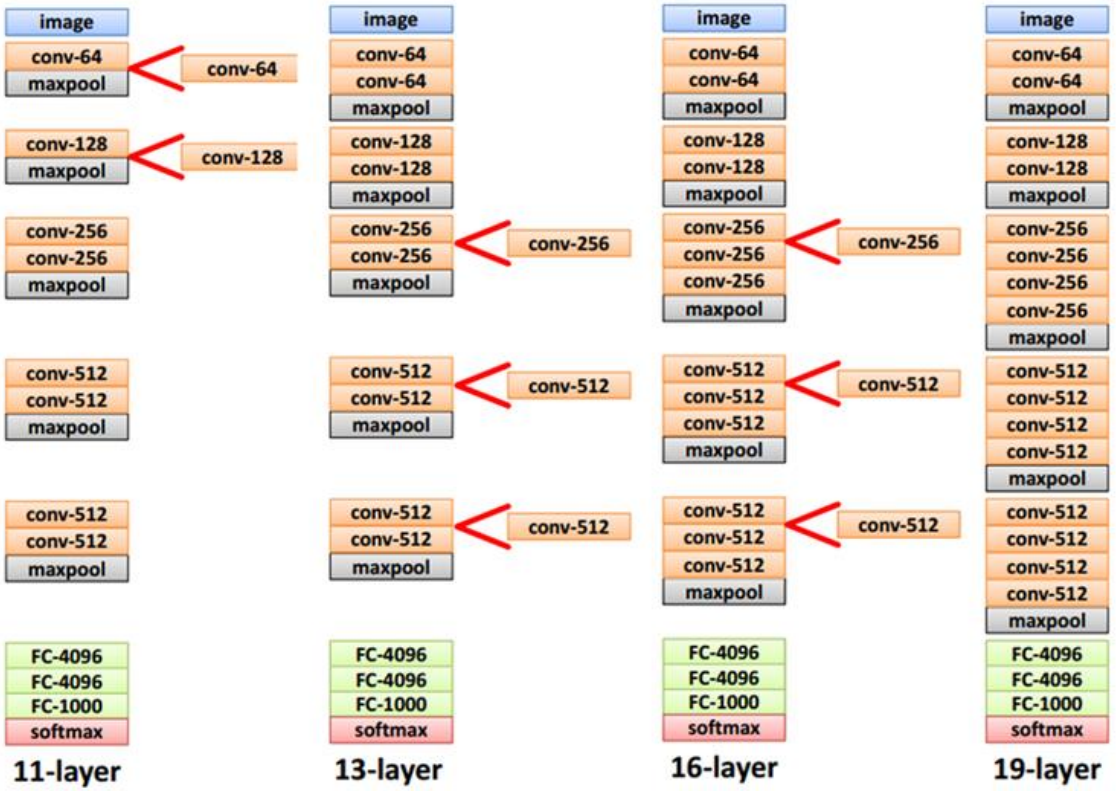


- Small filters & deep networks
  - Only 3×3 filters with stride 1, pad 1

VGG16

VGGNet (2)

- Why use smaller filters? (3×3 conv)
  - One 7×7 conv layer vs. Stack of three 3×3 conv (stride 1) layers
    - The same effective receptive field
    - But, deeper, more non-linearities
    - Fewer parameters: 3×(3×3×C) vs. 7×7×C



VGGNet (3)

- Memory & Parameters for VGG16

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
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

Note:

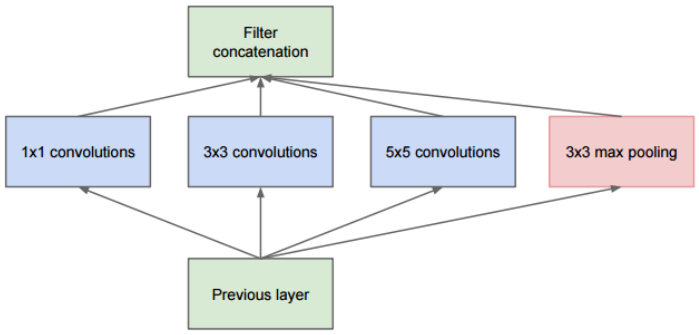
Most memory is in early CONV

Most params are in late FC

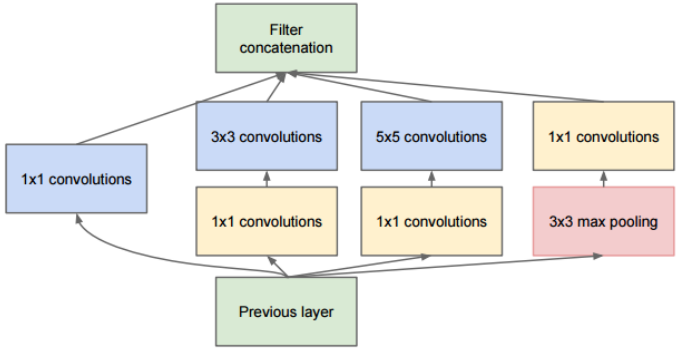
TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

GoogLeNet (1)

- As DNN goes deeper
  - The number of parameters increases ⇨ possibility of overfitting ↑
  - The amount of computations increases
    - Not suitable for mobile devices or embedded systems
- Key features of GoogLeNet
  - Inception Module
    - 같은 Layer에서 서로 다른 크기의 filter 적용 ⇨ 다른 scale의 feature 추출
    - 1×1 convolution ⇨ dimension reducton
  - 망의 깊이는 훨씬 깊으나 AlexNet 대비 parameters의 수는 ½



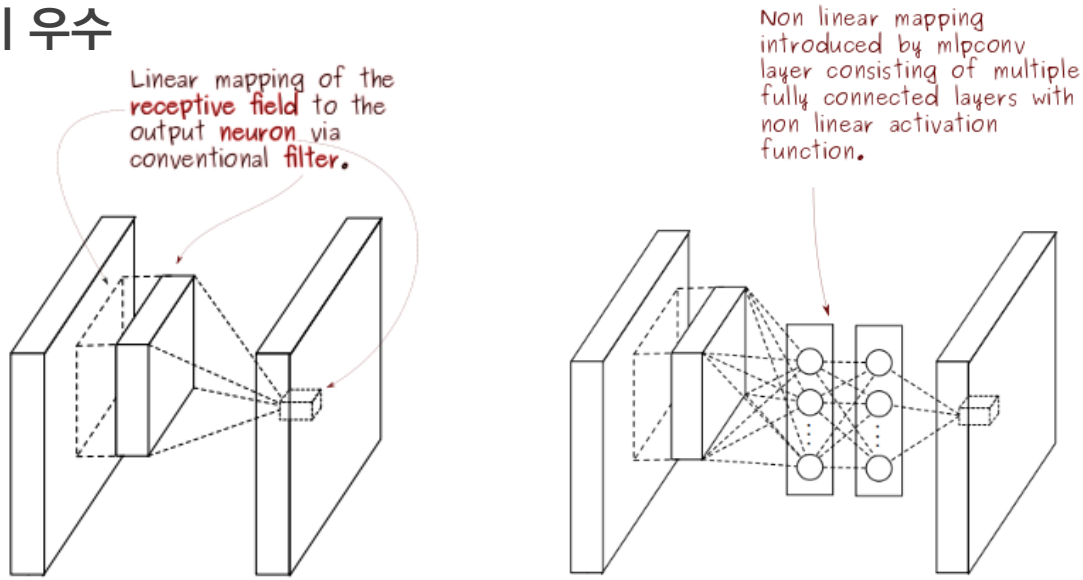
(a) Inception module, naïve version



(b) Inception module with dimension reductions

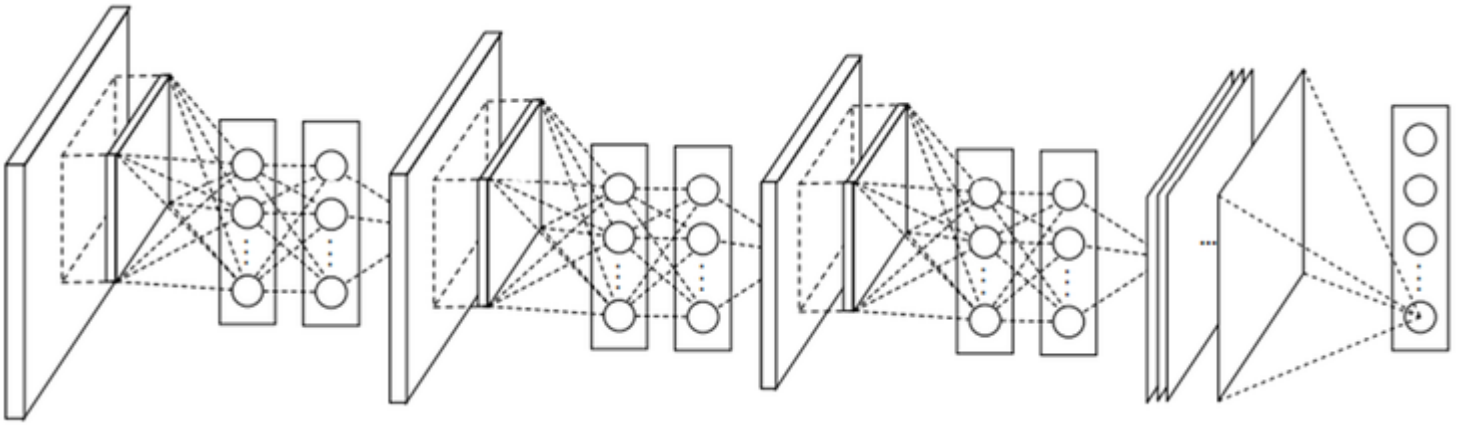
GoogLeNet (2)

- Conventional convolutional layer
  - Local receptive field에서 feature 추출 능력은 우수
  - filter의 특징이 linear하기 때문에 non-linear한 성질을 갖는 feature를 추출하기 위해 feature map의 개수를 늘려야 하는 문제 발생
    - Filter 개수 증가 ⇨연산량 증가
- NIN (Network In Network)
  - Convolution을 수행하기 위해 filter 대신 MLP(Multi-Layer Perceptron)을 사용하여 Feature 추출
  - Convolution kernel보다 non-linear한 성질을 잘 활용할 수 있기 때문에 feature 추출 능력이 우수

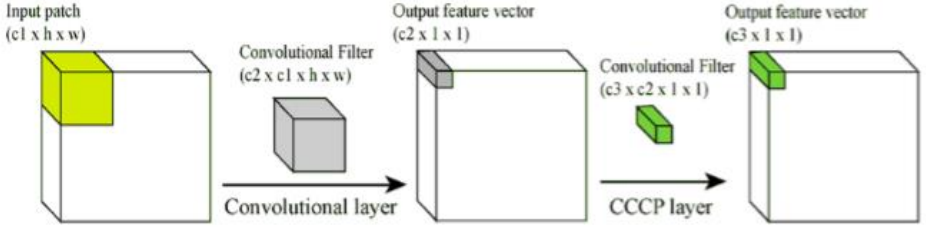


GoogLeNet (3)

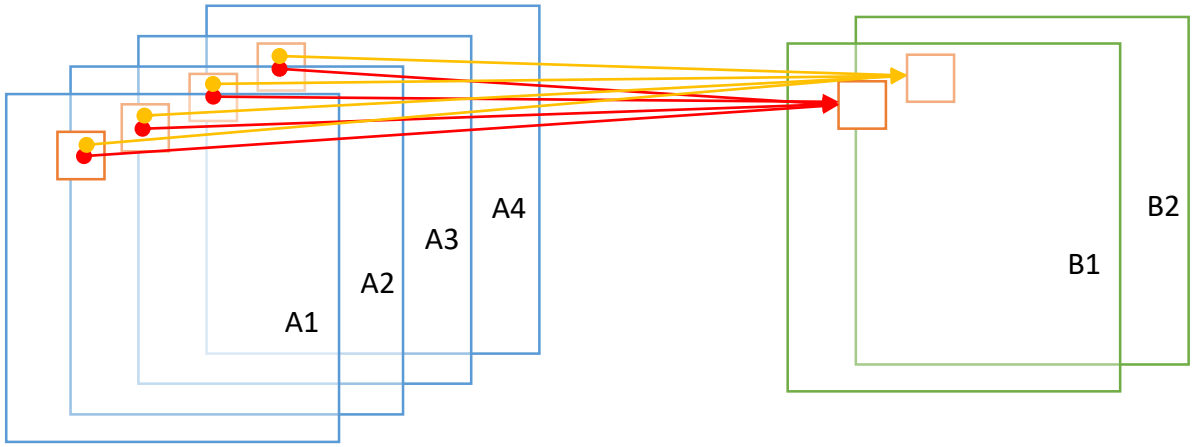
- NIN vs. CNN
  - NIN에서는 CNN의 최종단에서 흔히 사용되는 fully connected network이 없음
  - 앞 단에서 효과적으로 feature vector를 추출했기 때문에 pooling만으로도 충분히 classification이 가능하다고 주장
  - FCN 대신 최종단에 global average pooling 사용
    - Average pooling만으로 classifier 역할 수행
    - 파라메터 수 감소 → 연산량 감소 & overfitting 회피



GoogLeNet (4)



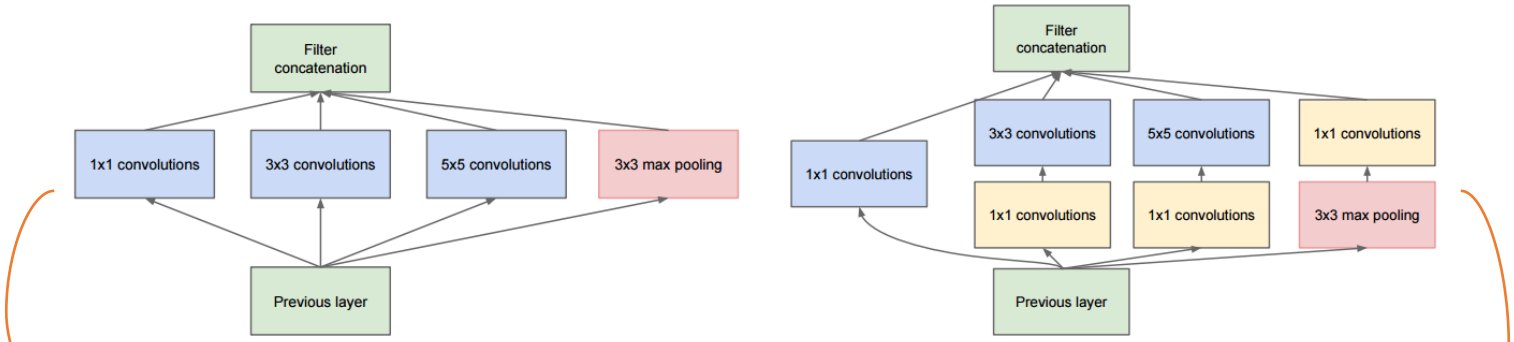
- 1×1 convolution
  - For dimension reduction
  - 여러 개의 feature map으로부터 비슷한 성질(the same location)을 갖는 것들을 묶을 수 있음



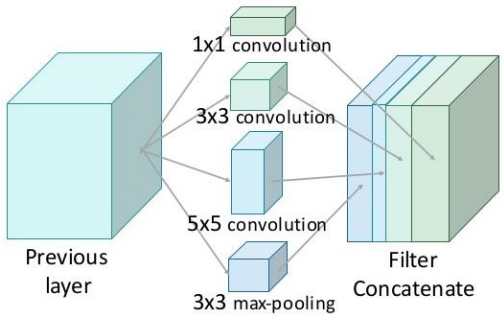
- 학습을 통해 얻어진 learned parameter를 통해 4개의 입력 feature map이 2개의 출력 feature map으로 결정됨
- Neuron에는 활성화 함수로 ReLU를 사용함으로써 추가로 non-linearity를 얻을 수 있음

GoogLeNet (5)

- Inception
  - 여러 개의 convolution을 병렬적으로 활용 ⇨ 다양한 scale의 feature 추출

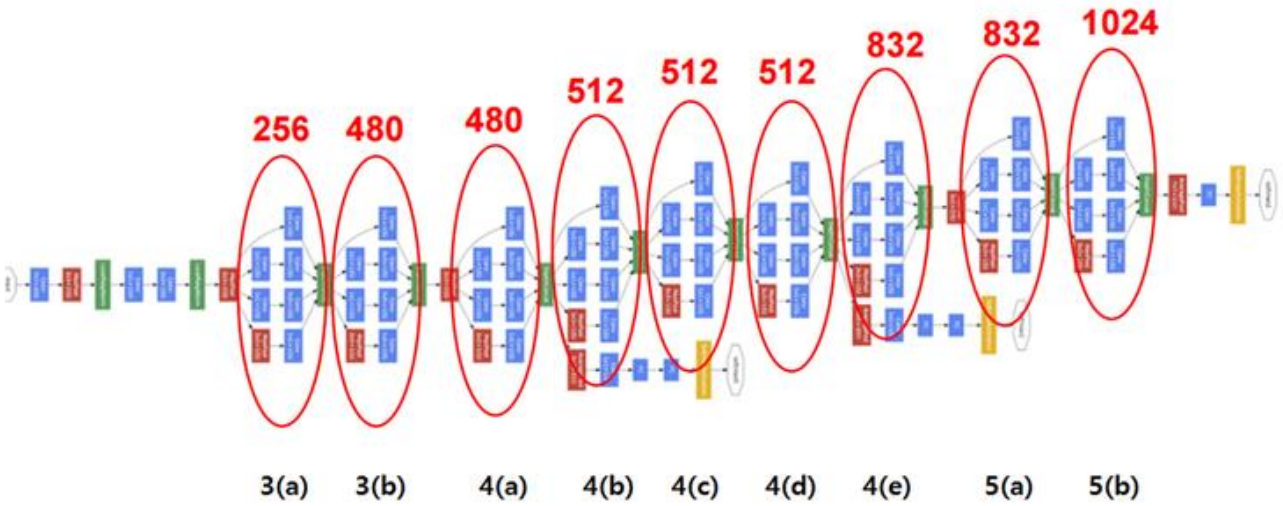


3×3과 5×5과 convolution의 경우 연산량 관점에서 볼 때 expensive unit



3×3과 5×5과 convolution 앞에 1×1 convolution을 cascade구조로 위치
➔ 1×1 convolution을 통해 feature map의 차원 축소
➔ 다양한 scale의 feature 추출과 동시에 연산량의 균형을 맞출 수 있음

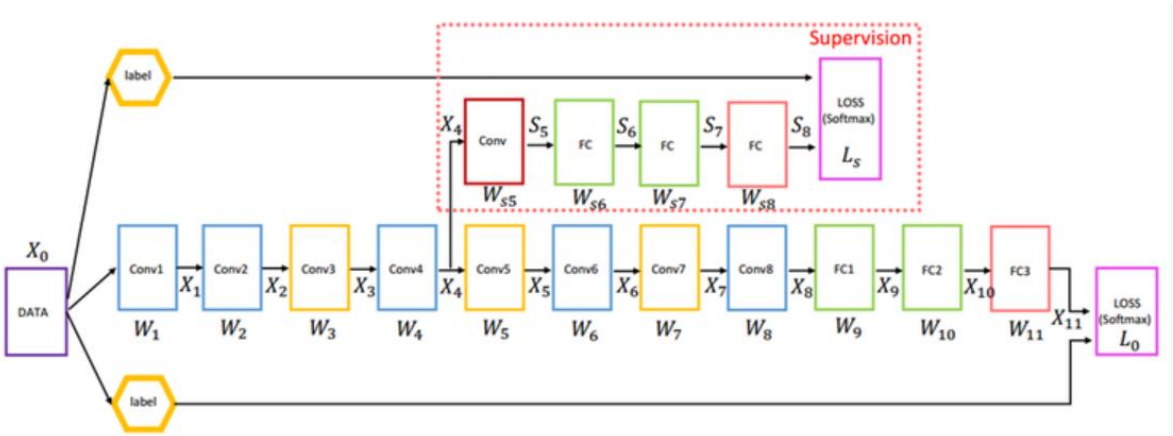
GoogLeNet (6)



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								

GoogLeNet (7)

- Auxiliary Classifier
  - Backpropagation시에 X4 위치에서는 Auxiliary Classifier와 최종 출력으로부터 정상적인 Backpropagation 결과를 결합
    - Auxiliary Classifier의 Backpropagation 결과가 더해지기 때문에 X4 위치에서 Gradient가 작아지는 문제 해결



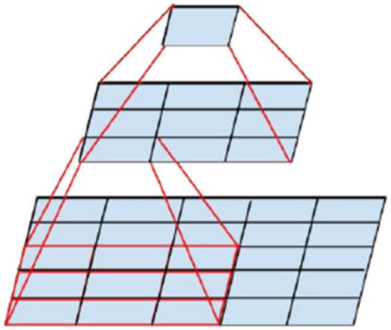
  - Auxiliary Classifier의 위치
    - 초기 10~50번 정도의 Iteration을 통해 Gradient가 어떻게 움직이는지 확인하고 그 위치에 Auxiliary Classifier를 붙이는 것이 좋음
  - 학습된 DNN을 사용할 때는 Auxiliary Classifier를 삭제 후 사용

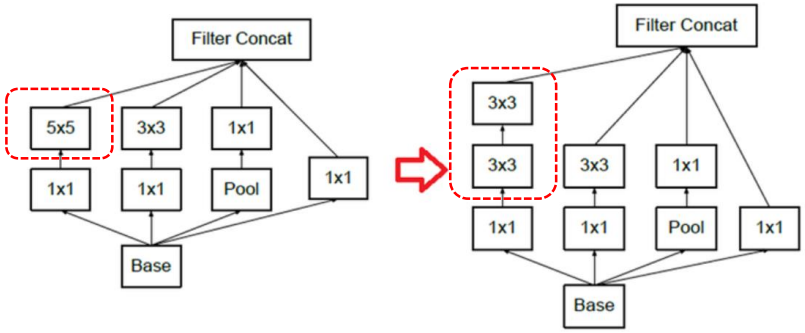
GoogLeNet (8)

- Training 방법
  - 영상의 가로/세로 비율을 [3/4, 4/3]의 범위를 유지하면서 원본 영상의 8%에서 100%까지 포함할 수 있도록 다양한 크기의 patch를 학습에 사용 (Data augmentation)
  - Photometric distortion을 이용한 학습 데이터 생성 (Data augmentation)
- Test 방법
  - Test 영상을 256×256 1개의 scale만을 사용하는 것이 아니라 256, 288, 320, 352로 총 4개의 scale로 만들고, 각각의 scale로부터 3장의 정사각형 이미지를 선택함
  - 선택된 이미지로부터 4개의 코너 및 중앙 2개의 취해 총 6장의 224×224 크기 영상을 취하고 각각을 좌우 반전시켜 총 144개의 Test 영상 생성 후 CNN에 입력으로 제공
  - 144개 softmax값의 평균값 또는 Voting을 이용하여 최종 class 결정

GoogLeNet (9)

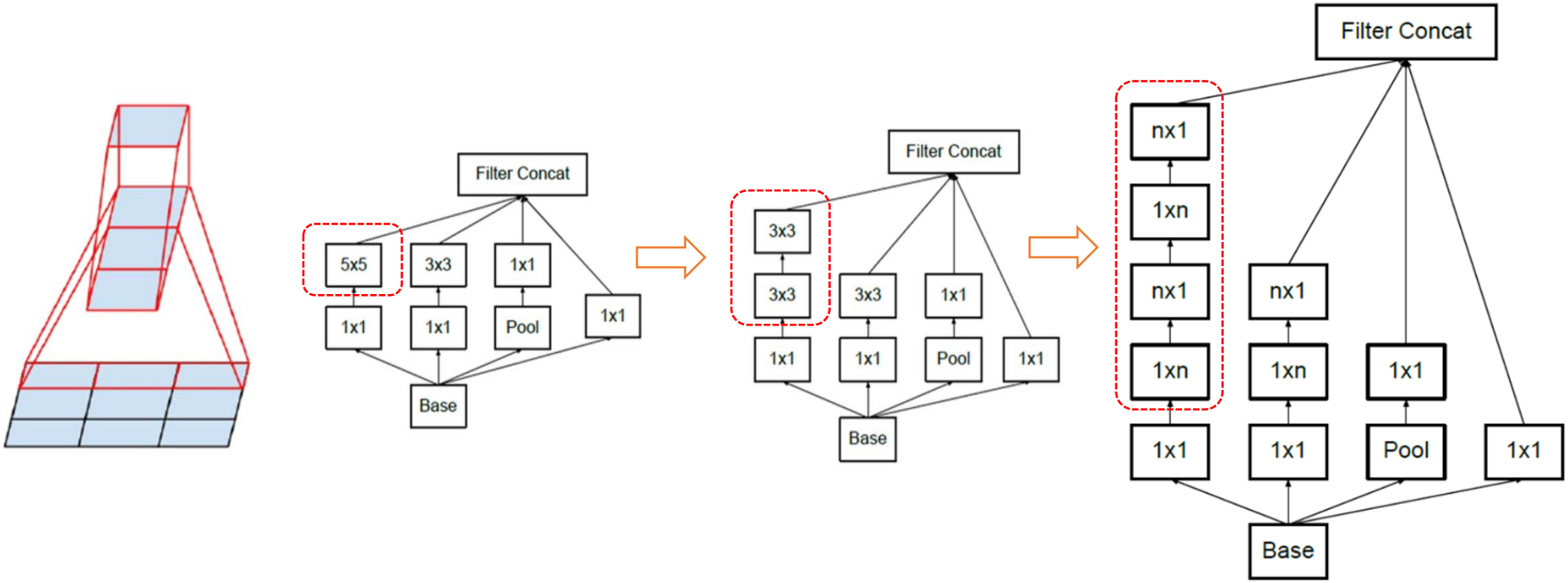
- Factorizing Convolutions
  - 큰 필터를 복수개의 작은 필터로 인수분해 ⇨ Parameter의 수를 감소시키면서 망은 깊어지는 효과 발휘
  - 1개의 5×5 filter vs. 2개의 3×3 filter
    - 5×5 convolution은 3×3 convolution에 비해 넓은 영역에 걸쳐 있는 특징을 1번에 추출 가능
    - Free parameter: (9+9)/25 = 72 ➔ 28% 절감 효과





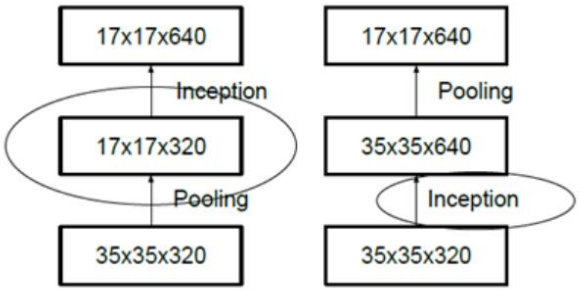
GoogLeNet (10)

- Asymmetric Factorizing
  - n×n convolution을 1×n convolution과 n×1 convolution으로 분해
  - Example
    - 3×3 convolution을 1×3 convolution과 3×1 convolution으로 분해
    - Parameter: (3+3)/9 = 66.6 ➔ 33% 절감



GoogLeNet (11)

Reduction of grid size



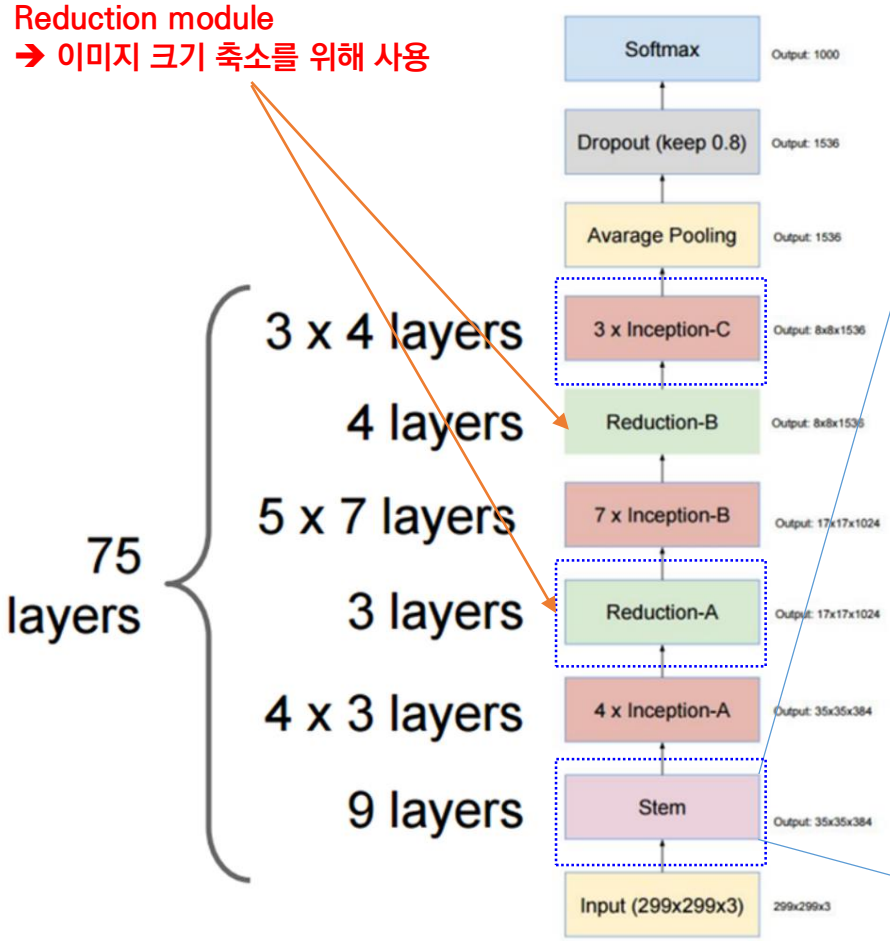
- 35×35×320 feature map에 pooling을 적용하여 크기를 반으로 줄인 후 Inception을 적용하여 feature 추출
  ➔연산량 관점에서는 효율적
  ➔Pooling 단계를 거치면서 원본 feature map에 숨어 있는 정보가 사라지게 될 가능성이 높음

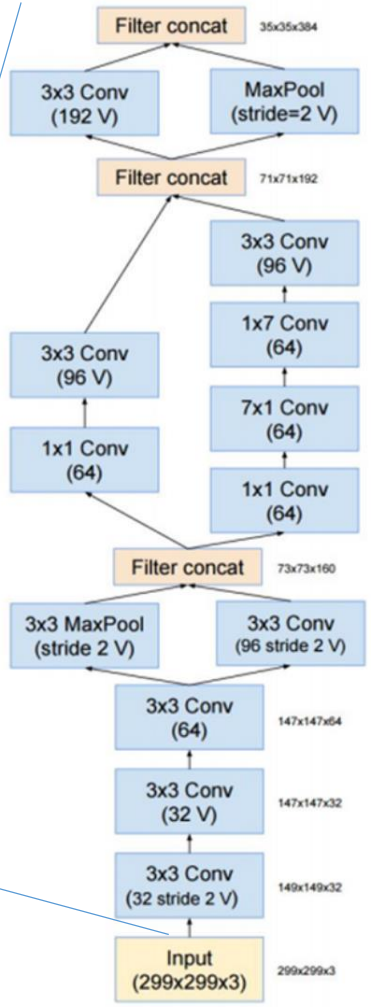
- Inception을 먼저 적용하여 640개의 feature map을 얻은 후 pooling을 적용하여 크기 축소
  ➔연산량 관점에서는 4배 증가
  ➔feature map의 크기를 줄이기 전에 Inception을 적용했기 때문에 숨은 특징을 더 잘 찾아낼 수 있음

GoogLeNet (12)

- Inception-V4

Reduction module
➔ 이미지 크기 축소를 위해 사용





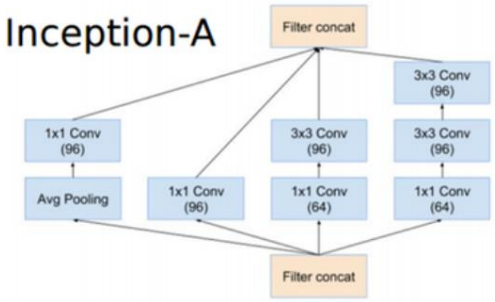
1x7, 7x1 filters

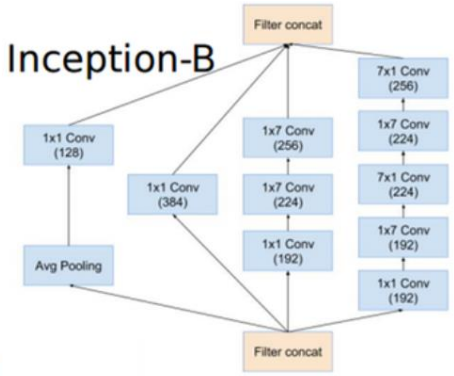
Strided convolution AND max pooling

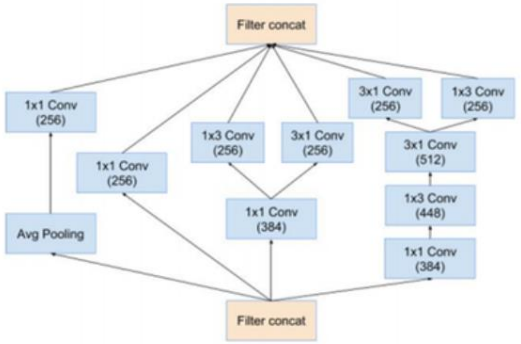
V = Valid convolutions (no padding)

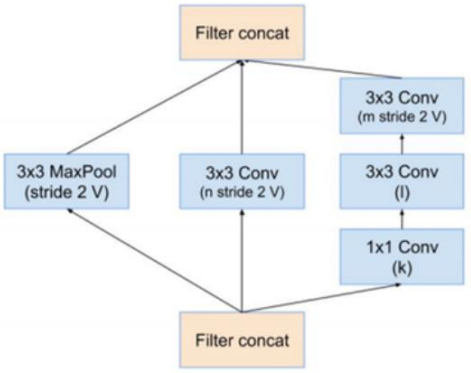
GoogLeNet (13)

- Inception-V4 (cont.)

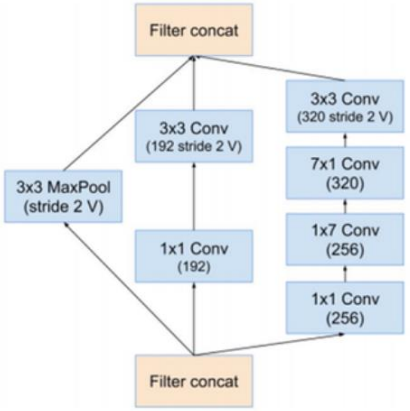








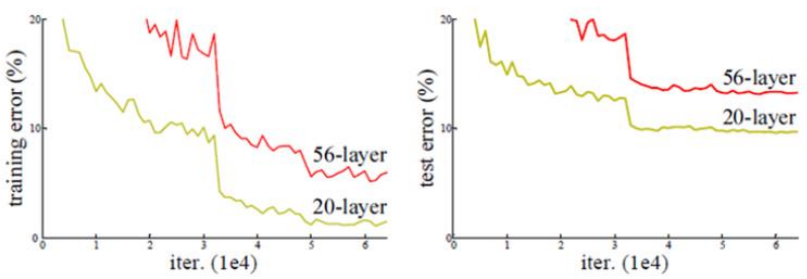
Reduction-A
K=192, l=224, m=256, n=384



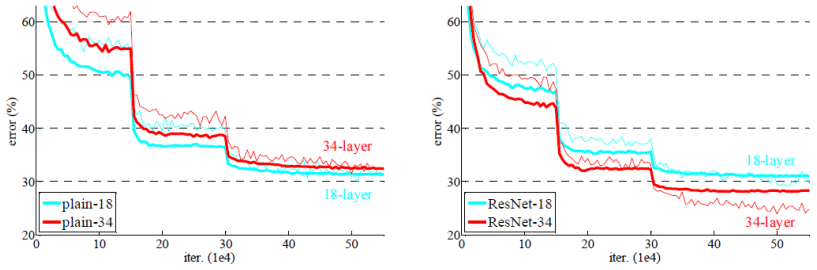
Reduction-B

ResNet (1)

- 망이 깊어질수록 학습시키기가 어려워짐
  - Vanishing/Exploding gradients
    - Solutions: normalized initialization, normalization layers, …
  - Overfitting
    - 망이 깊어지면 parameter 수가 비례적으로 증가 ➔ overfitting & error ↑
  - Degradation problem
    - ResNet에서 해결하고자 하는 문제
    - Different from overfitting
    - Adding more layers ➔ higher training error



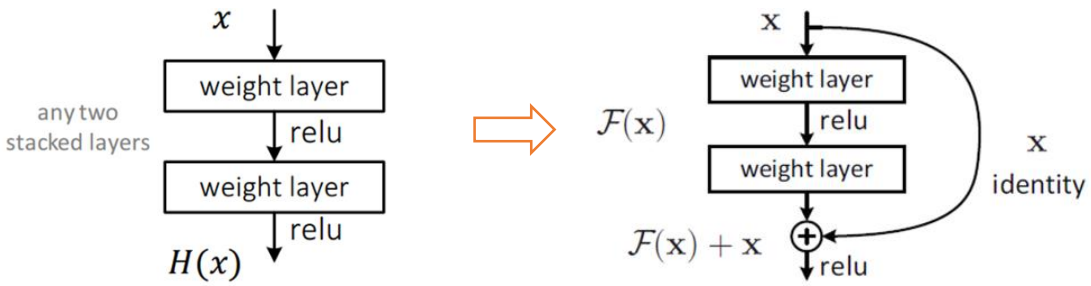
망이 깊은 경우 (56-layer) 성능이 더 나쁨



PlainNet: 망이 깊은 경우 (34-layer) 성능이 더 나쁨
ResNet: 망이 깊은 경우 (34-layer) 성능이 더 좋음

ResNet (2)

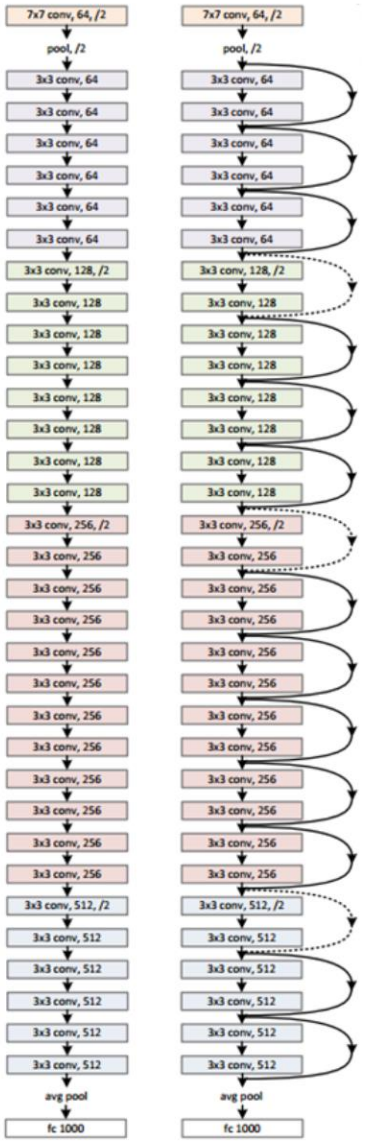
- Residual Learning
  - Depth가 깊어지더라도 degradation 문제가 발생하지 않도록 하기 위함
  - Key idea
    - Learning the residual function (e.g., H(x)−x), instead of the actual function (e.g., H(x))



ResNet (3)

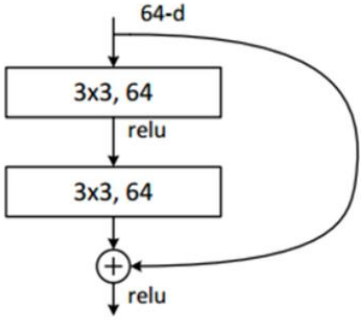
- Network architecture
  - Output feature map의 크기가 같은 경우 해당 모든 layer는 동일한 filter수를 가짐
  - Feature map의 크기가 절반으로 작아지는 경우 연산량의 균형을 맞추기 위해 filter 채널 수를 2배로 증가
    - Feature map의 크기를 줄일 때는 pooling을 사용하는 대신 convolution을 수행할 때 stride=2로 설정
  - 연산량을 줄이기 위해 max-pooling (1곳 제외), hidden fc, drop-out 제거

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[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



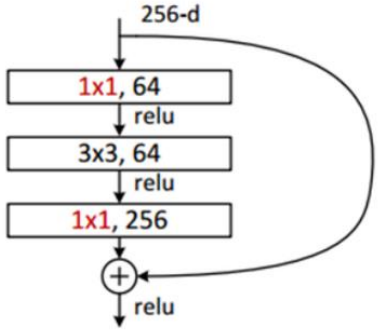
ResNet (4)

- Building block



all-3x3

- Bottleneck Building block

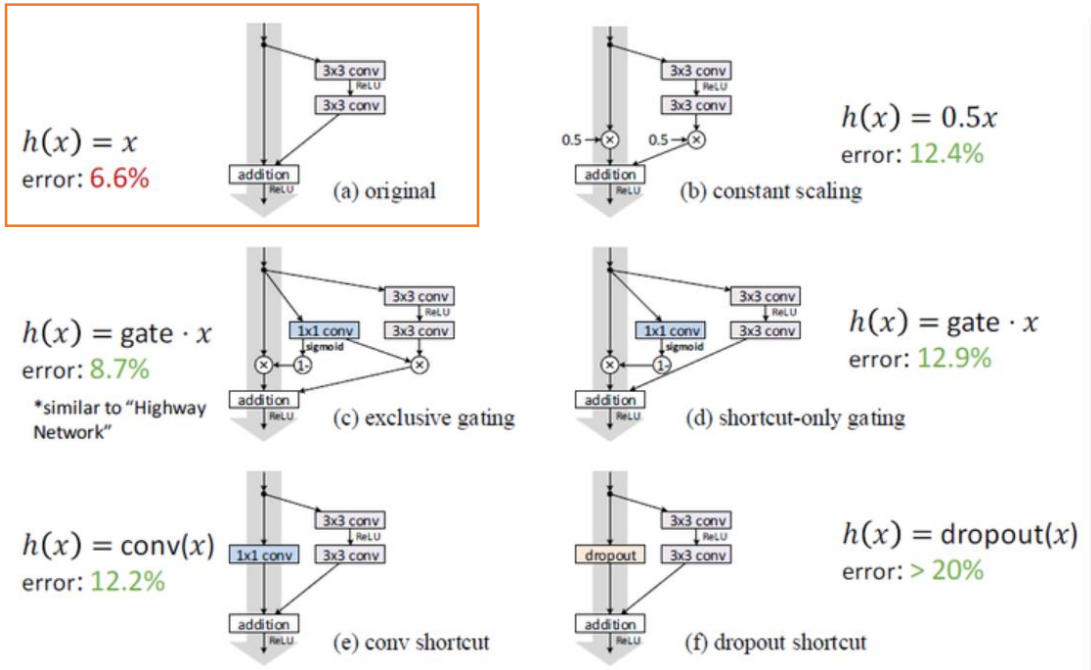


bottleneck
(for ResNet-50/101/152)

1×1 convolution을 통해 차원 축소 후 3×3 convolution을 수행.
마지막으로 1×1 convolution을 통해 차원 확대
➔ 3×3 convolution을 곧바로 연결시킨 구조보다 연산량 감소

ResNet (5)

- Identity skip connection
  - Short-cut connection의 구성에 따른 성능 비교

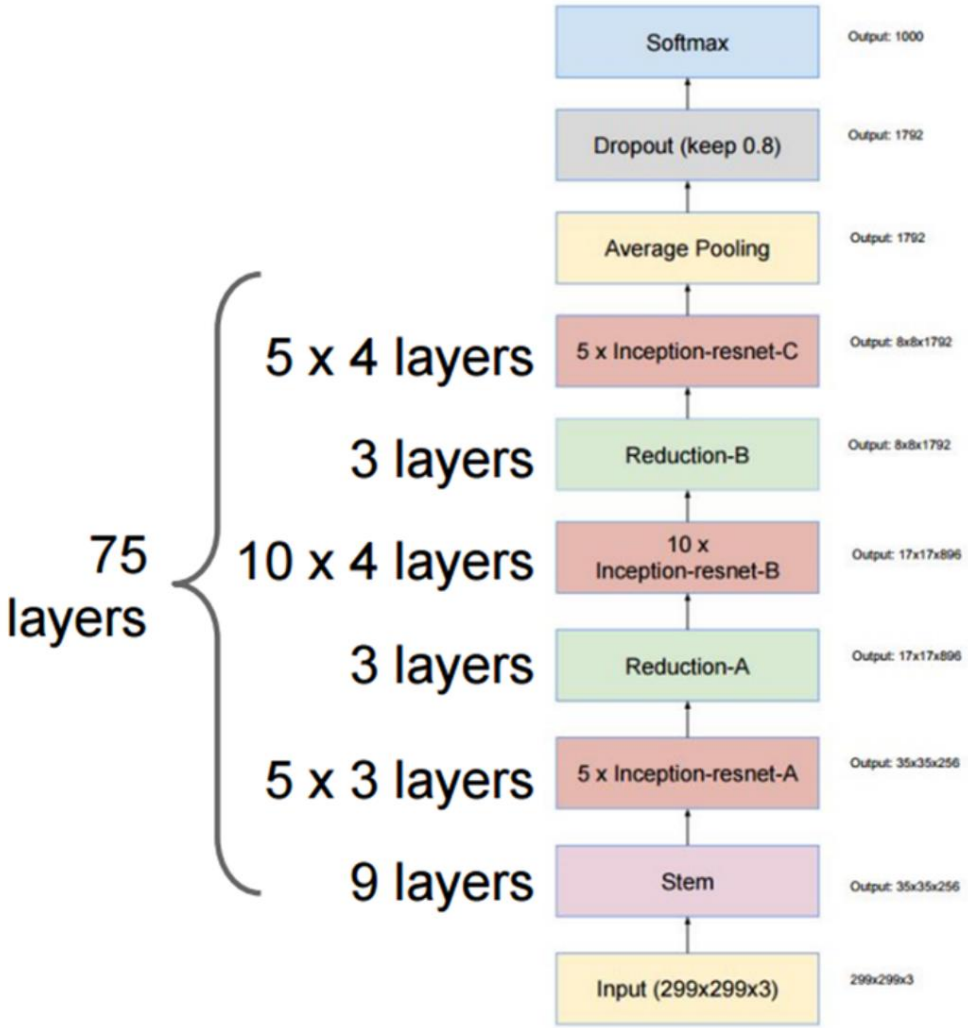


(b)에서 (f)까지는 x_l의 정보가 변경 없이 "clean" 상태로 전달되는 것이 아니라 곱셈에 의한 변형된 형태로 전달되기 때문에 최적화 문제 발생 가능

➔(a)는 short-cut connection에 어떤 것도 추가하지 않고 identity connection으로 연결시킨 경우로 성능이 가장 우수함

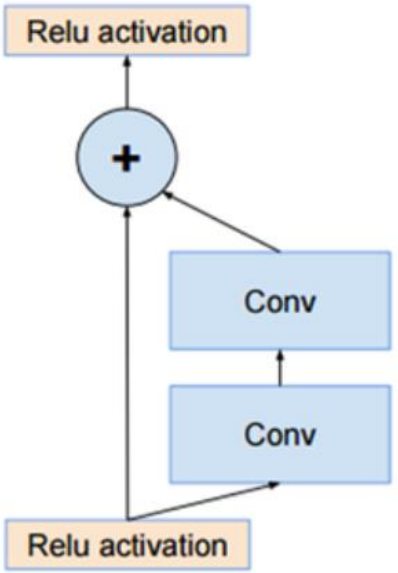
ResNet (6)

- Inception–ResNet

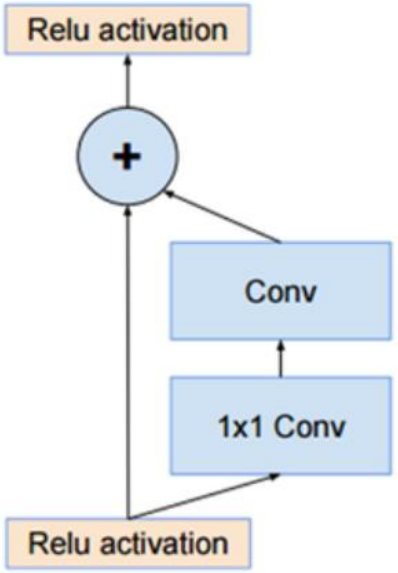


ResNet (7)

- Inception-ResNet (cont.)
  - 기존 ResNet 모듈 변경
    - 1×1 convolution을 통한 차원 감소 ➔ 연산량 감소



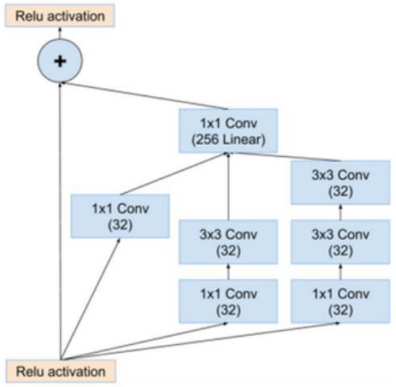
Residual connections as introduced in He et al. [5]



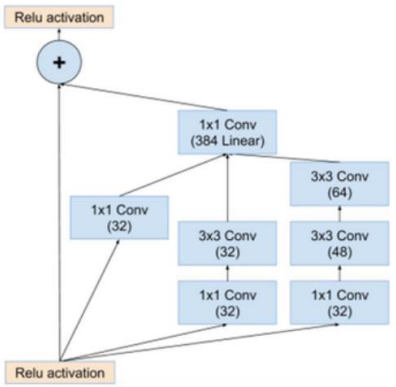
Optimized version of ResNet connections by [5] to shield computation.

ResNet (8)

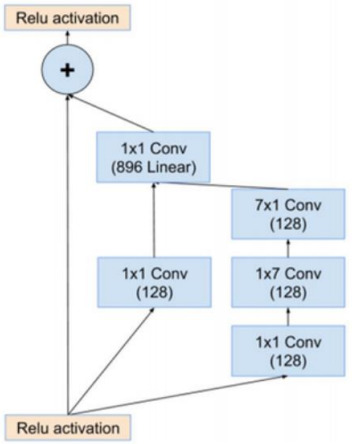
- Inception–ResNet (cont.)



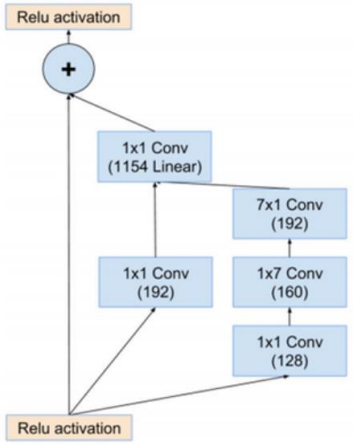
Inception-ResNet-A in v1



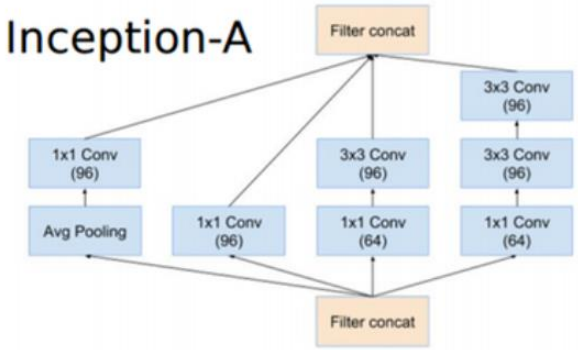
Inception-ResNet-A in v2

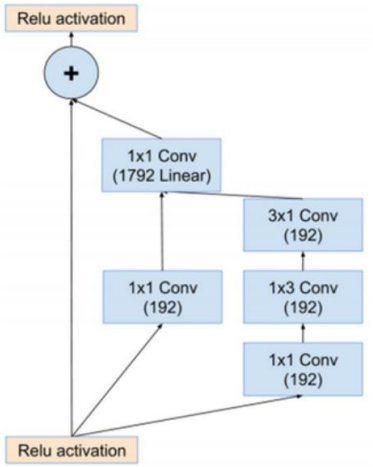


Inception-ResNet-B in v1

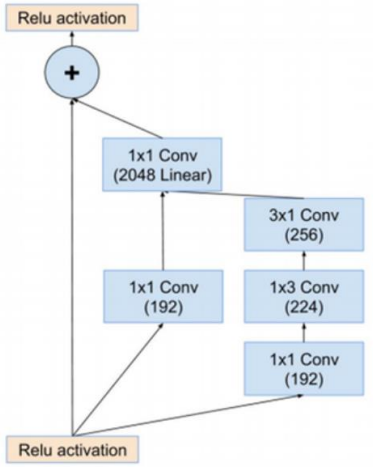


Inception-ResNet-B in v2





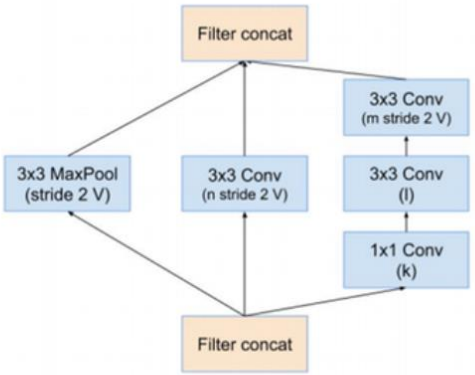
Inception-ResNet-C in v1



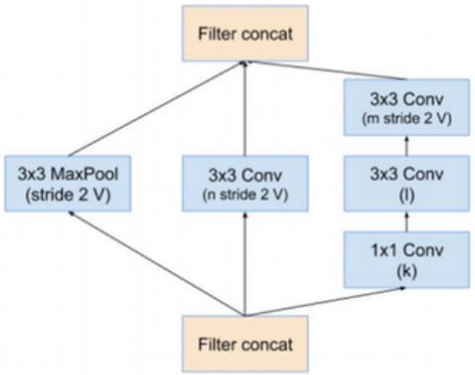
Inception-ResNet-C in v2

ResNet (9)

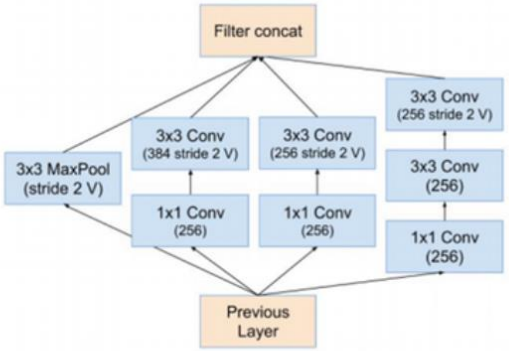
- Inception–ResNet (cont.)



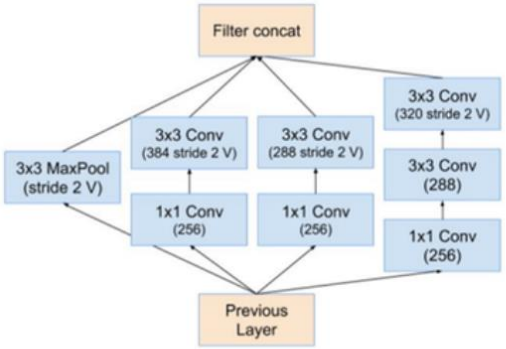
Reduction-A v1
K=192, l=192, m=256, n=384



Reduction-A v2
K=256, l=256, m=384, n=384

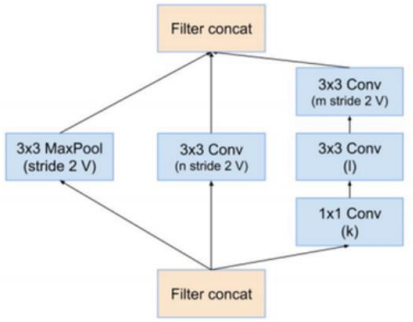


Reduction-B v1

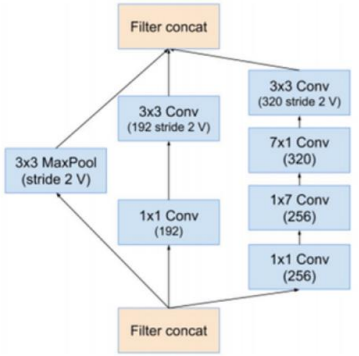


Reduction-B v2

Inception-V4



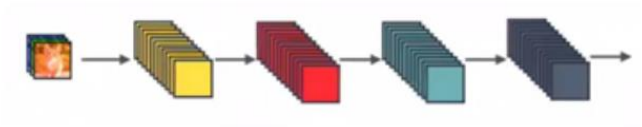
Reduction-A
K=192, l=224, m=256, n=384



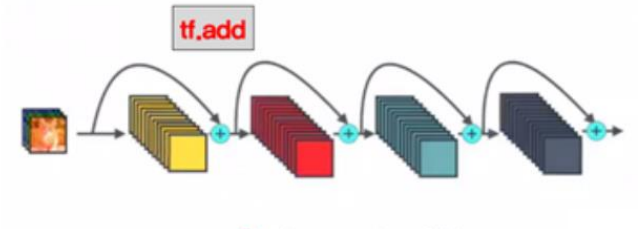
Reduction-B

DenseNet (1)

- ConvNet vs. ResNet vs. DenseNet

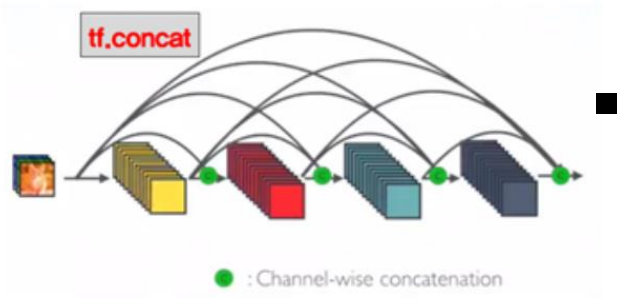


Standard Connectivity



ResNet Connectivity

$$\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1}) + \mathbf{x}_{\ell-1}$$



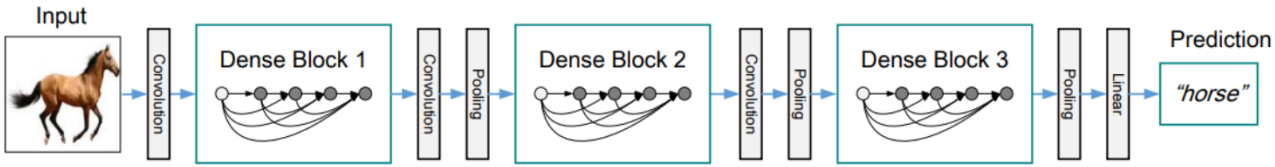
Dense Connectivity

$$\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}])$$

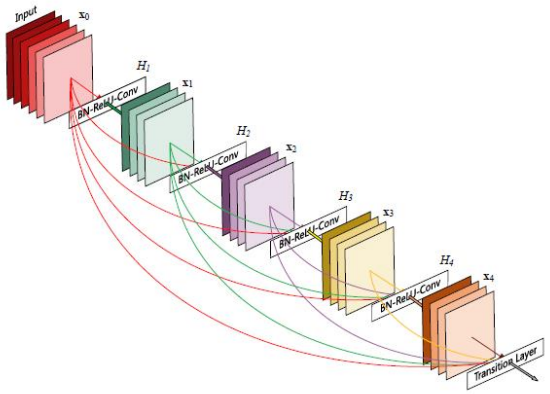
$H_\ell(\cdot)$
1. Batch normalization (BN)
2. Rectified linear unit (ReLU)
3. 3*3 convolution

- Advantages
  - They alleviate the vanishing-gradient problem
  - Strengthen feature propagation
  - Encourage feature reuse
  - Substantially reduce the number of parameters (less complexity)
  - Reduce overfitting on tasks with smaller training set sizes.

DenseNet (2)

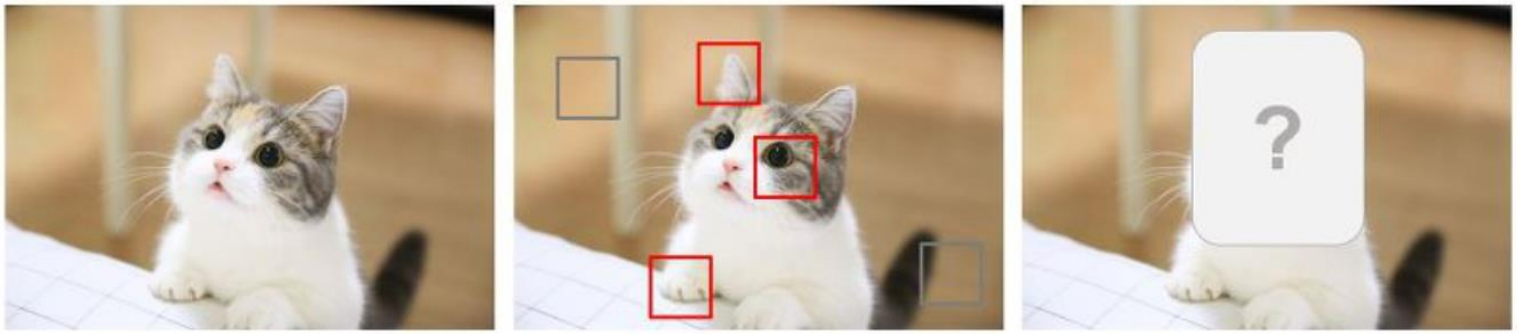


Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			



Attentions (1)

- DNN의 깊이를 더 깊게 구성함으로써 표현 능력 향상
  - Inception 모듈, Residual 모듈, Dense 모듈 등
- 최근에는 쓸모없는 정보는 제거하고 중요한 특징은 더욱 부각시키는 방법 적용 ➔ Attention

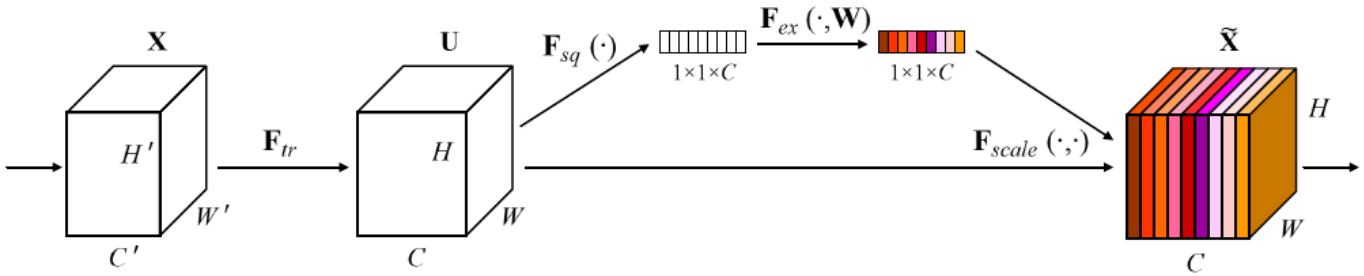


Attentions (2)

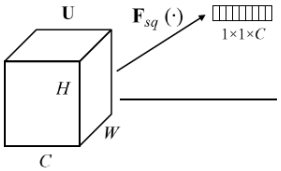
- Squeeze-and-Excitation Network
  - 기존의 Conv 연산은 filter 크기가 H×W×C 형태이기 때문에 입력 feature map의 channel간 상관 관계를 암묵적으로 활용한다고 볼 수 있음
    - 그러나 filter의 spatial resolution이 H×W이기 때문에 해당 영역의 channel 상관 관계만을 고려하기 때문에 channel간 상관 관계에 대한 global information을 사용한다고 볼 수 없음
  - 각 channel간의 global한 상관 관계를 추출해내기 위한 SE block 제안
    - 입력 feature map이 주어졌을 때, SE block을 통과시킴으로써 입력 feature를 구성하는 channel 중 중요한 feature를 포함하는 channel은 더욱 부각시키고 그렇지 않은 channel은 상대적으로 억제시킴
    - Early layer에서는 class-agnostic한 특징들이 부각되는 반면, later layer로 갈수록 class-specific한 특징들이 부각됨

Attentions (3)

- SE block architecture



- Squeeze – global information embedding
  - 채널 기반의 global 정보 추출을 위해 Global Average pooling 사용



$$\mathbf{z} \in \mathbb{R}^C$$

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

- Excitation – adaptive recalibration
  - Input-specific descriptor z를 channel-specific weights로 변경
  - 2개의 Fully Connected Layer 사용

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

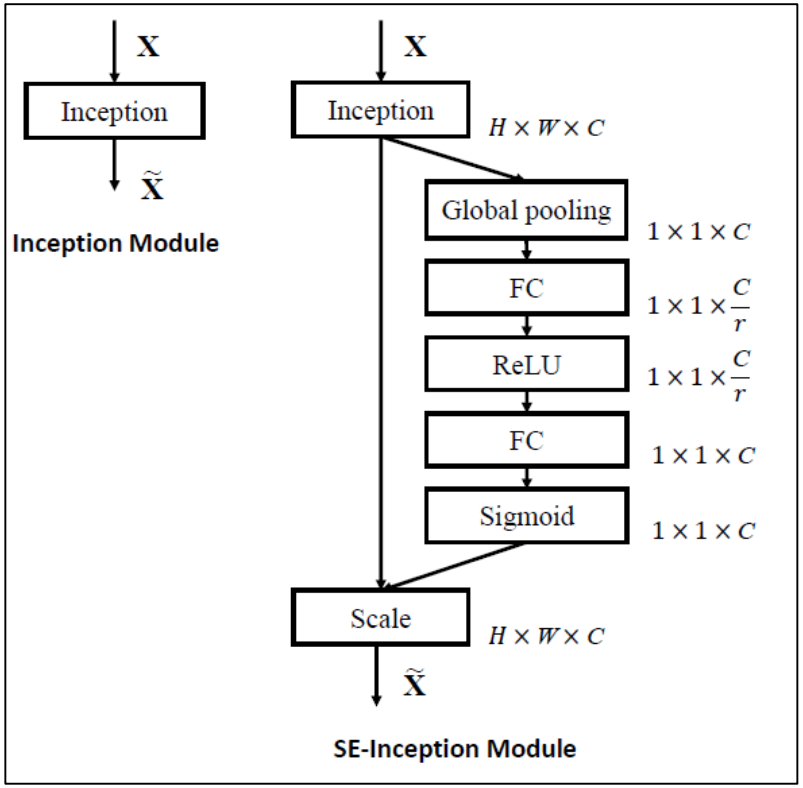
$$\mathbf{W}_1 \in \mathbb{R}^{\frac{C}{r} \times C}$$

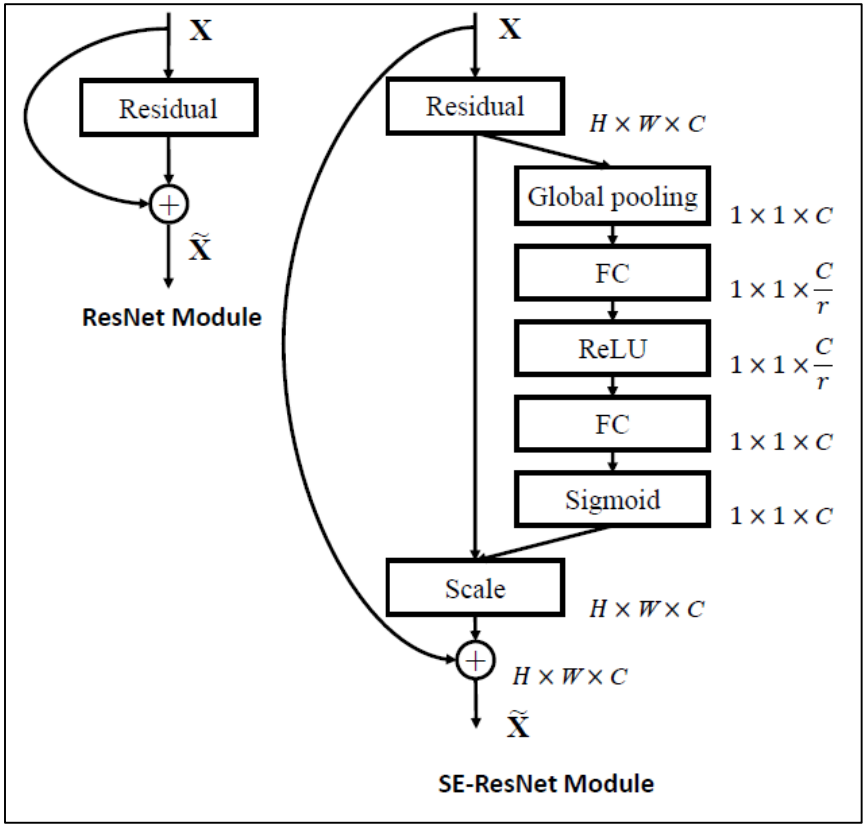
$$\mathbf{W}_2 \in \mathbb{R}^{C \times \frac{C}{r}}$$

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

Attentions (4)

- SE block instantiations





Attentions (5)

- Performance Comparision

Single-crop error rates (%) of state-of-the-art CNNs on ImageNet validation set with crop sizes 224×224 and 320×320 / 299×299.

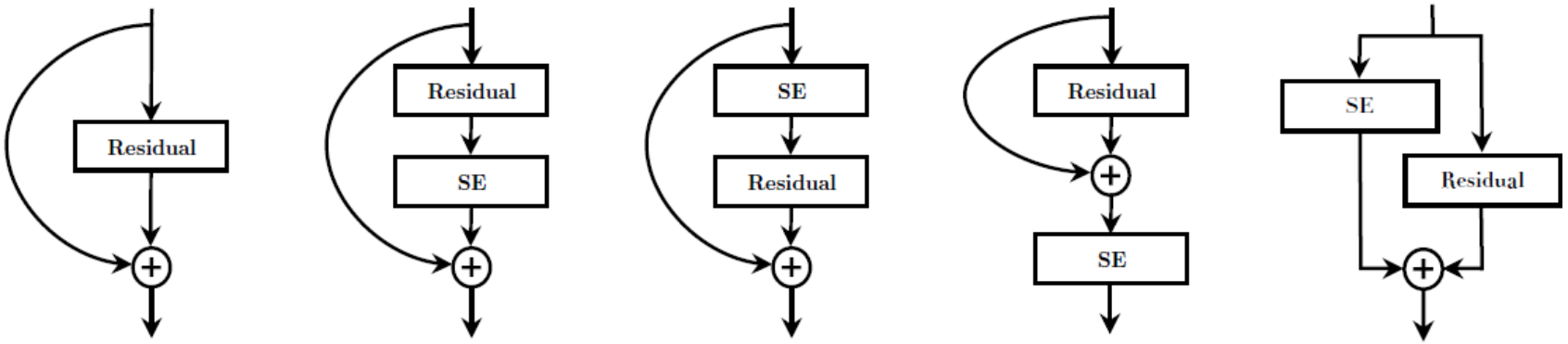
	224×224		320×320 / 299×299	
	top-1 err.	top-5 err.	top-1 err.	top-5 err.
ResNet-152 [13]	23.0	6.7	21.3	5.5
ResNet-200 [14]	21.7	5.8	20.1	4.8
Inception-v3 [20]	-	-	21.2	5.6
Inception-v4 [21]	-	-	20.0	5.0
Inception-ResNet-v2 [21]	-	-	19.9	4.9
ResNeXt-101 (64×4d) [19]	20.4	5.3	19.1	4.4
DenseNet-264 [17]	22.15	6.12	-	-
Attention-92 [58]	-	-	19.5	4.8
PyramidNet-200 [77]	20.1	5.4	19.2	4.7
DPN-131 [16]	19.93	5.12	18.55	4.16
SENet-154	**18.68**	**4.47**	**17.28**	**3.79**

Single-crop error rates (%) on ImageNet and parameter sizes for SE-ResNet-50 at different reduction ratios. Here, *original* refers to ResNet-50.

Ratio r	top-1 err.	top-5 err.	Params
2	22.29	6.00	45.7M
4	22.25	6.09	35.7M
8	22.26	5.99	30.7M
16	22.28	6.03	28.1M
32	22.72	6.20	26.9M
original	23.30	6.55	25.6M

Attentions (6)

- SE block integration design



(a) Residual block (b) Standard SE block (c) SE-PRE block (d) SE-POST block (e) SE-Identity block

Design	top-1 err.	top-5 err.
SE	22.28	6.03
SE-PRE	22.23	6.00
SE-POST	22.78	6.35
SE-Identity	22.20	6.15

References

- Very Deep Convolutional Networks for Large-Scale Image Recognition, https://arxiv.org/pdf/1409.1556.pdf
- Going Deeper with Convolutions, https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Szegedy_Going_Deeper_With_2015_CVPR_paper.pdf
- Deep Residual Learning for Image Recognition, https://arxiv.org/pdf/1512.03385.pdf
- Densely Connected Convolutional Networks, https://arxiv.org/pdf/1608.06993.pdf
- Squeeze-and-Excitation Networks, https://arxiv.org/pdf/1709.01507.pdf
- CS231n: Convolutional Neural Networks for Visual Recognition, http://cs231n.stanford.edu/
- 라온피플 머신러닝 아카데미, https://laonple.blog.me/


ANY
QUESTIONS?