대표적인 CNN 모델 분석

김준영

분석 모델

- AlexNet (ILSVRC2012)
- VGGNet (ILSVRC2014)
- GoogLeNet (ILSVRC2014)
- ResNet (ILSVRC2015)
- MobileNet (2017)
- ShuffleNet (2018)

모델의 발전 과정

- AlexNet -> GoogLeNet, VGG -> ResNet
  - 분류 정확도를 높이기 위해 모델을 대형화 하는 방향으로 연구

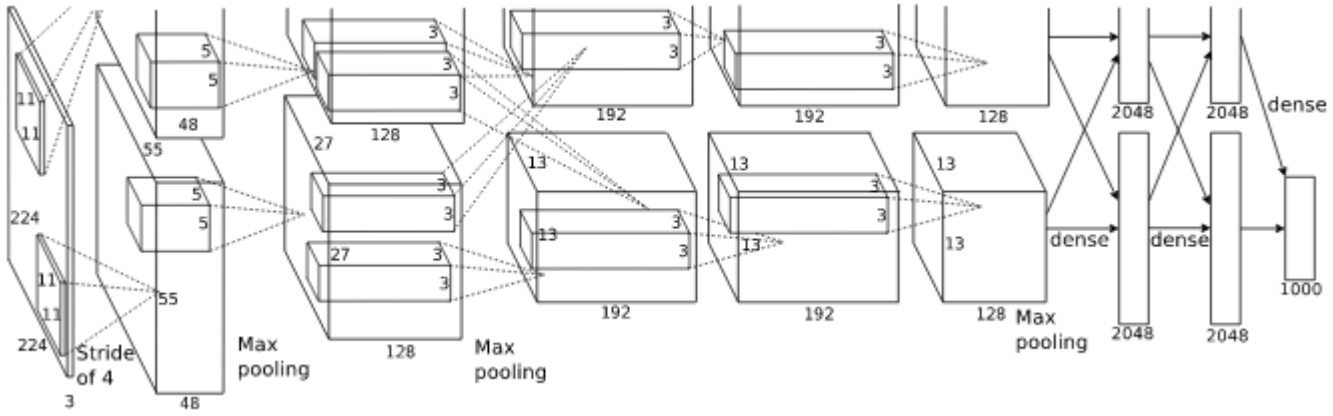
- MobileNet -> ShuffleNet
  - 충분한 정확도를 확보하면서도 연산 복잡도를 줄이는 방향으로 연구

AlexNet (1/6)

- ILSVRC2012에서 압도적인 성능으로 1위를 차지한 모델
  - Top-5 classification error
    - SuperVision(AlexNet): 15.3%
    - ISI: 26.1% / OXFORD_VGG: 26% / XRCE/INRIA: 27% / Univ. Amsterdam: 29.5%
- 대회 참가 팀 중 유일하게 CNN을 이용한 모델을 사용
- 뿐만 아니라 GPU를 연산에 활용한 유일한 모델

AlexNet (2/6)

- 5개의 Convolutional layer와 3개의 FC layer로 구성
  - 당시 GPU(GTX580)의 한계로 레이어의 채널을 둘로 나누어 연산



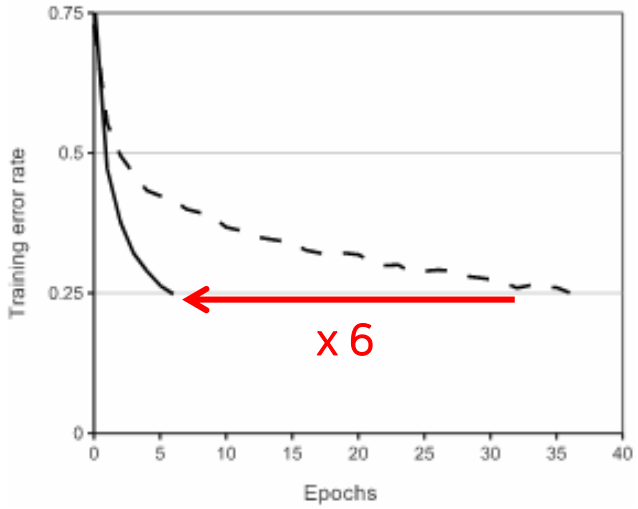
  - Conv. 1 레이어와 Conv. 2 레이어에서
    11x11, 5x5 라는 비교적 큰 커널을 사용해 컨볼루션 연산을 진행

AlexNet (3/6)

- AlexNet은 모델의 성능을 향상시키기 위해 여러 방법을 사용
  - ReLU activation function
  - Overrapped pooling
  - Local Response Normalization
  - Data augmentation
  - Dropout
  - Using GPU (GTX580 x2)

AlexNet (4/6)

- 활성화 함수로 ReLU 함수를 사용
  - 당시 보편화된 활성화 함수는 tanh 함수와 sigmoid 함수
  - 두 함수 역시 잘 작동한다는 주장이 있었지만
    이러한 주장은 대개 학습 속도보다는 overfitting 방지에 초점이 맞춰져 있음
  - ReLU 함수는 기존 활성화함수보다 빠른 학습 속도를 보장
  - 따라서 ReLU 함수를 사용하면 기존에는 불가능했던
    대형 데이터셋을 이용한 대형 모델을 학습하는 것을 가능하게 함



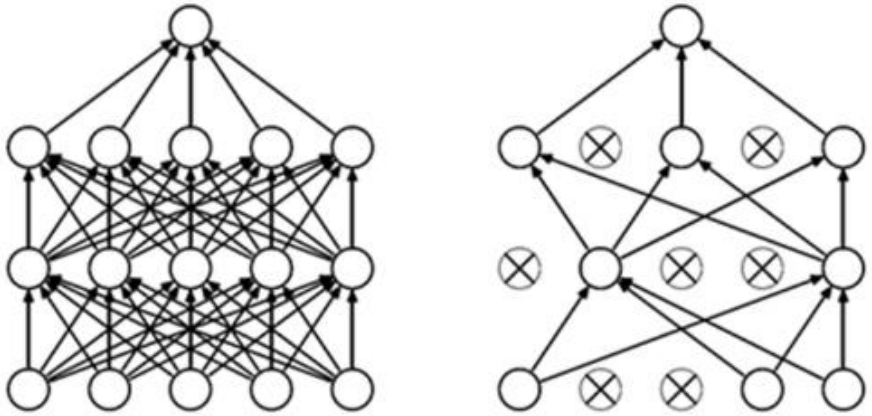
AlexNet (5/6)

- Data augmentation
  - Overfitting 문제를 해결하기 위한 대표적인 방법은
    학습에 사용할 데이터의 양을 늘리는 것
  - AlexNet은 이미지를 256x256 크기로 축소한 후,
    무작위로 224x224 이미지를 추출
  - 추출된 이미지를 상하 반전하거나 RGB 채널의 값을 변화시켜
    한정된 이미지에서 다양한 데이터를 확보

AlexNet (6/6)

- 히든 레이어의 개수가 많으면 일반적으로 학습 능력이 좋아짐
- 그러나 망의 크기가 커지면 커질 수록 Overfitting에 빠질 가능성이 높아짐

- Dropout
  - 망의 크기가 커져 Overfitting에 빠지는 문제를 피하기 위해
    무작위로 일부 뉴런을 생략한 후 학습을 수행
  - 무작위로 뉴런이 생략되기 때문에 가중치가 큰 뉴런에
    신경망이 받는 영향이 줄어 뉴런들이 동조화되는 경향을 피할 수 있음
  - 이 과정을 무작위로 반복하면 여러 신경망을 통한 평균 효과를 얻을 수 있음



(a) Standard Neural Net (b) After applying dropout.

VGG (1/4)

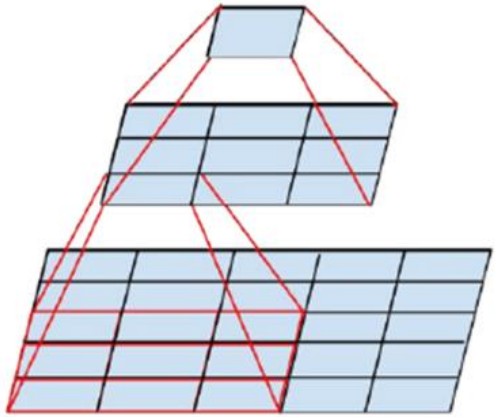
- ILSVRC2014에서는 AlexNet보다 더 깊어진 모델이 나타나기 시작
- GoogLeNet은 22개의 레이어 층으로, VGG는 19개의 레이어 층으로 구성
  - Error rate: GoogLeNet – 6.65% / VGG19 - 7.32%
- CNN의 성능을 향상시키는 가장 직접적인 방법은 망의 크기를 키우는 것
  - 하지만 레이어의 수가 증가할 수록 파라미터의 수가 증가하게 되면서 Overfitting에 빠질 가능성이 높아짐
  - 또한 망의 크기가 커질 수록 연산량이 증가함
    - *8개의 레이어를 사용한 AlexNet도 학습에 일주일을 소모*

VGG (2/4)

- VGG는 다른 모델들이 사용한 11x11, 7x7, 5x5와 같은 컨볼루션 연산 대신 3x3 컨볼루션 연산을 중첩해 이를 구현
  - 3x3 컨볼루션 연산 두 번은 5x5 영역에 컨볼루션을 한 것과 같은 효과
  - 3x3 컨볼루션 연산 세 번은 7x7 영역에 컨볼루션을 한 것과 같은 효과

- (3x3 Conv.)x2 : 5x5 Conv. = 18 : 25
- (3x3 Conv.)x3 : 7x7 Conv. = 27 : 49

- 3x3 컨볼루션의 중첩이 큰 컨볼루션 연산 한 번보다 더 높은 효율을 얻음



VGG (3/4)

- VGG는 Conv. Layer에 패딩을 적용했고
  max pooling을 이용해 feature map의 크기를 줄임

- VGG는 비교적 간단한 방식으로 깊은 신경망을 구성할 수 있지만
- 파라미터의 수가 매우 많은(144M) 문제가 있음

VGG16 VGG19

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					

VGG (4/4)

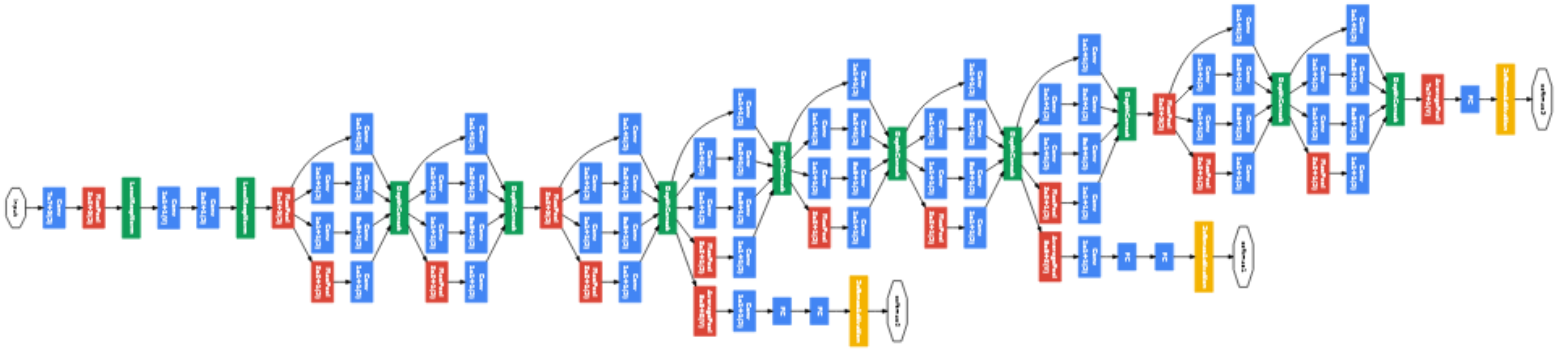
- VGG는 입력 feature를 생성하기 위해
  single-scale, 또는 multi-scale 기법을 이용

- Multi-scale
  - 이미지를 384x384로 축소시킨 후 224x224 영역을 추출하여 학습
  - 256~512 범위 내에서 무작위로 축소시킨 이미지에서
    feature를 추출해 fine tuning을 수행

GoogLeNet (1/6)

- AlexNet과 VGG는 단순한 레이어 구조로 구성

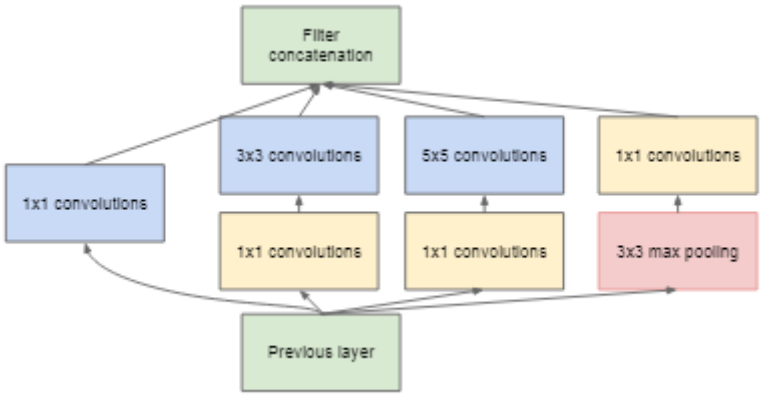
- GoogLeNet은 더 많은 레이어를 이용하기 위해 복잡한 모듈 구조로 구성
  - 망의 깊이는 훨씬 깊지만 파라미터의 수는 훨씬 적음
  - AlexNet (8 layers) : 60M
  - VGG19 (19 layers) : 144M
  - GoogLeNet (22 layers): 5M



GoogLeNet (2/6)

- GoogLeNet은 단순히 Conv. Layer를 이어 붙인 구조가 모델을 확장하기에 적합하지 않다는 가정을 세움

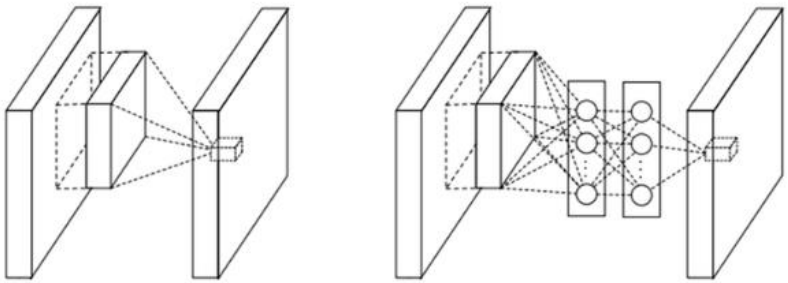
- 다양한 크기의 Conv. Layer가 병렬적으로 연결된 inception 모듈을 제작



  - Min Lin 등이 발표한 Network in Network 구조를 발전시킨 형태

GoogLeNet (3/6)

- NiN
  - 컨볼루션 연산이 feature 추출 능력은 우수하지만
  - 컨볼루션의 linear한 특성 때문에
    non-linear한 성질을 갖는 feature까지 추출하기에는 어려움이 있음
  - 그래서 micro neural network를 제안



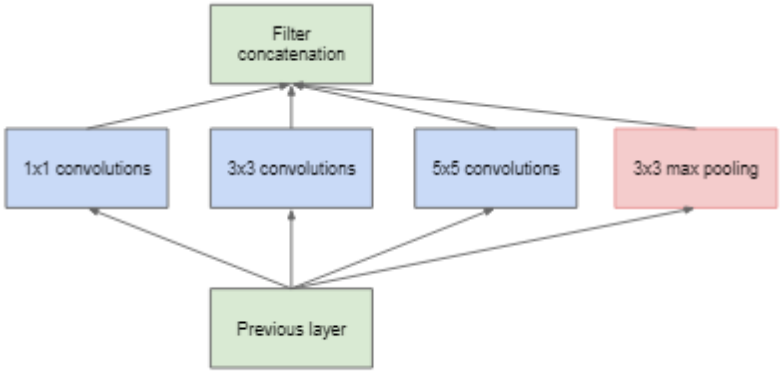
(a) Linear convolution layer (b) Mlpconv layer

  - 컨볼루션 대신 MLP를 사용해 feature를 추출한 다음
    1x1 컨볼루션을 이용해 channel 크기를 축소
  - 또, 효과적으로 feature를 추출했기 때문에
    fc layer 없이 global average pooling만으로도 분류를 할 수 있음

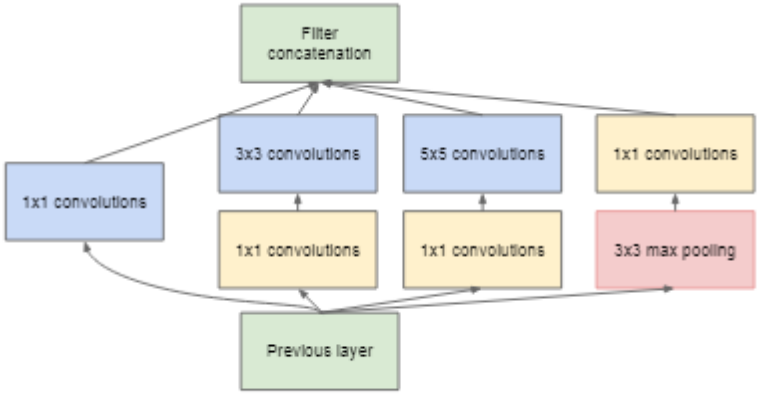
GoogLeNet (4/6)

- Inception Module
  - GoogLeNet은 MLP 대신 다양한 크기의 컨볼루션 연산을 병렬적으로 두어 다양한 feature를 추출하는 방법을 선택
  - 이는 컨볼루션 연산의 막대한 비용 문제로 인해 신경망을 확장하기 힘듬

  - 연산량을 줄이기 위해 1x1 컨볼루션 연산으로 feature map의 차원의 줄인 후
  - 3x3, 5x5 컨볼루션 연산을 진행



(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

GoogLeNet (5/6)

- 효과적으로 feature를 추출하였기 때문에
  별도의 fc layer 연산 없이 average pooling 후 이미지 분류

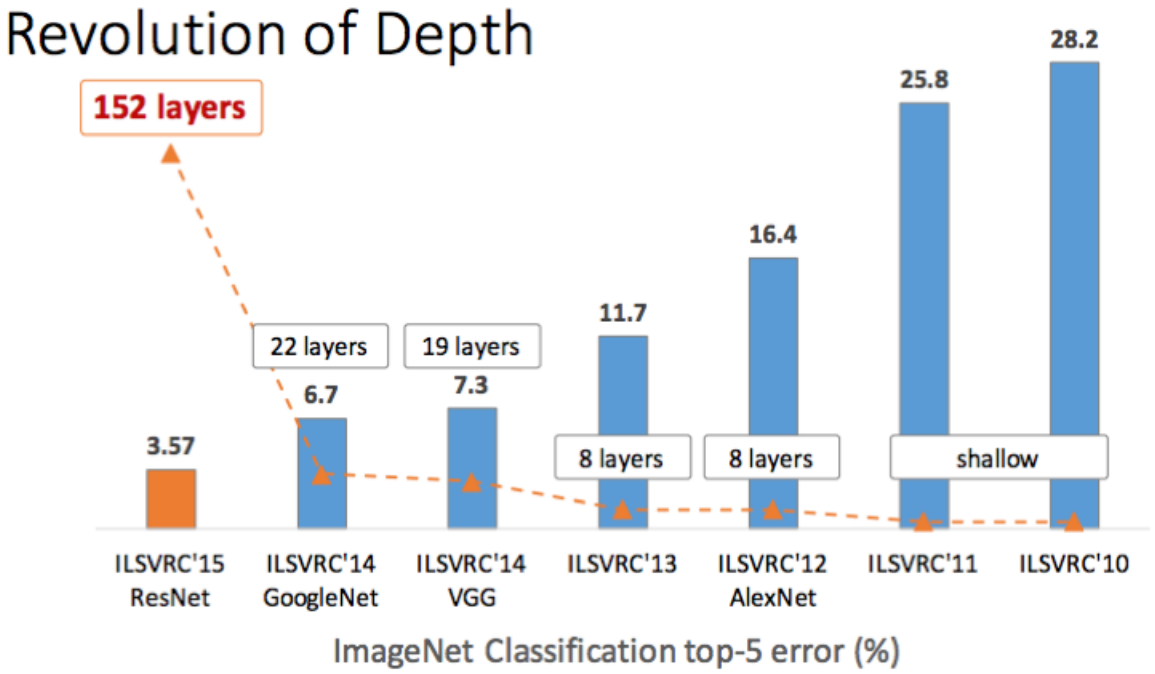
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	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 (6/6)

- Auxiliary Classifier
  - Vanishing gradient 문제를 해결하기 위해
    두 개의 auxiliary classifier를 학습 과정에서 사용
  - Auxiliary classifier와의 분기점에서는
    auxiliary classifier와 최종 출력의 back-propagation 결과를 결합해 학습

ResNet (1/7)

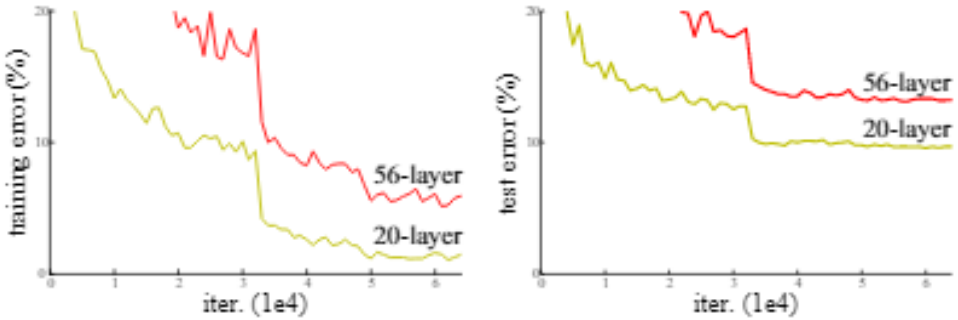
- ResNet은 152개의 레이어로 구성
- Top-5 error rate 3.56%라는 높은 정확도 달성



ResNet (2/7)

- 망이 깊어질 수록 모델의 정확도가 증가하는 것은
  VGG와 GoogLeNet이 보여줌

- 하지만 망이 깊어지면 깊어질 수록 제대로 학습시키는 것은 어려움
  - 56개의 레이어를 이용한 모델의 정확도가
    20개의 레이어를 이용한 모델보다 낮음

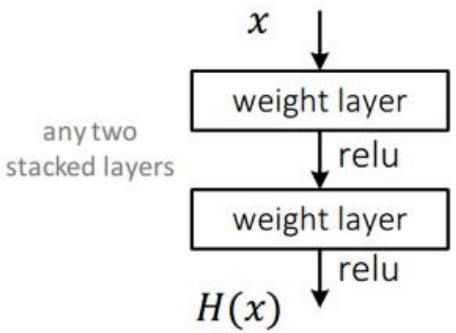


  - Vanishing/exploding gradient 문제가 발생함

ResNet (3/7)

- ResNet은 Residual learning 기법을 통해 이러한 문제를 해결

- 기존 CNN 모델은 입력 x가 레이어를 거치면서 H(x)를 얻어내는 방식
- Residual learning은
  레이어를 거친 결과를 F(x)라고 하고 이에 입력 x를 더해
  "H(x) = F(x) + x"를 얻어내는 방식으로 구조를 변경



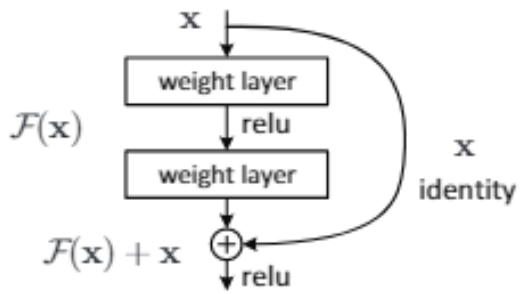


Figure 2. Residual learning: a building block.

ResNet (4/7)

- 기존의 학습 과정이 H(x)를 얻는 과정이었다면
  Residual learning의 학습 과정은 F(x) = H(x) - x 를 얻는 과정

- 최적의 경우 F(x) = 0이 되어야 하기 때문에
  학습 방향이 미리 결정되어 있는 상태에서 학습이 진행됨

- 입력과 출력의 잔차(residual)를 학습하고 나면,
  입력 값의 작은 변화에도 민감하게 반응할 수 있음

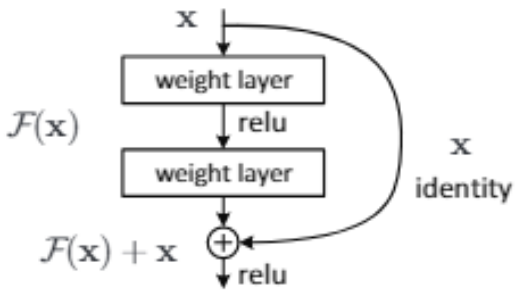
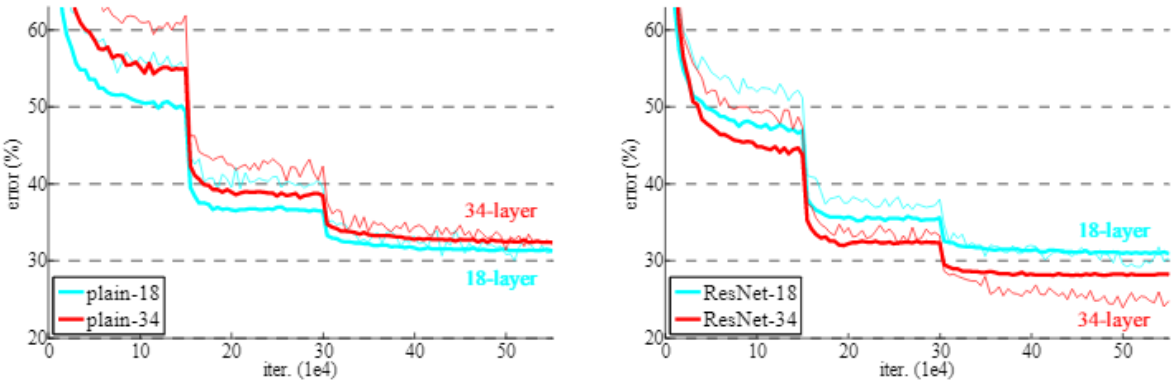


Figure 2. Residual learning: a building block.

ResNet (5/7)

- ResNet은 연산 복잡도를 줄이기 위해
  fc layer와 dropout 등은 사용하지 않고 max pooling 또한 한 번만 사용
  - Feature map의 크기를 줄일 때는 stride를 2로 바꾼 컨볼루션 연산을 수행

- 덕분에 신경망이 깊어져도 기존 모델처럼 error가 증가하지 않음



ResNet (6/7)

- 레이어가 깊어질 수록 학습 시간이 증가하는 문제를 해결하기 위해
  50개 이상의 레이어를 사용하는 대형 모델을 구성할 때는
  연산량을 줄이기 위해 1x1 컨볼루션을 이용

  - 1x1 컨볼루션을 통해 차원을 축소하고
    3x3 컨볼루션을 수행한 후
    1x1 컨볼루션으로 차원을 증가시키는 방법을 사용

ResNet (7/7)

- ResNet 레이어 구성

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

MobileNet (1/8)

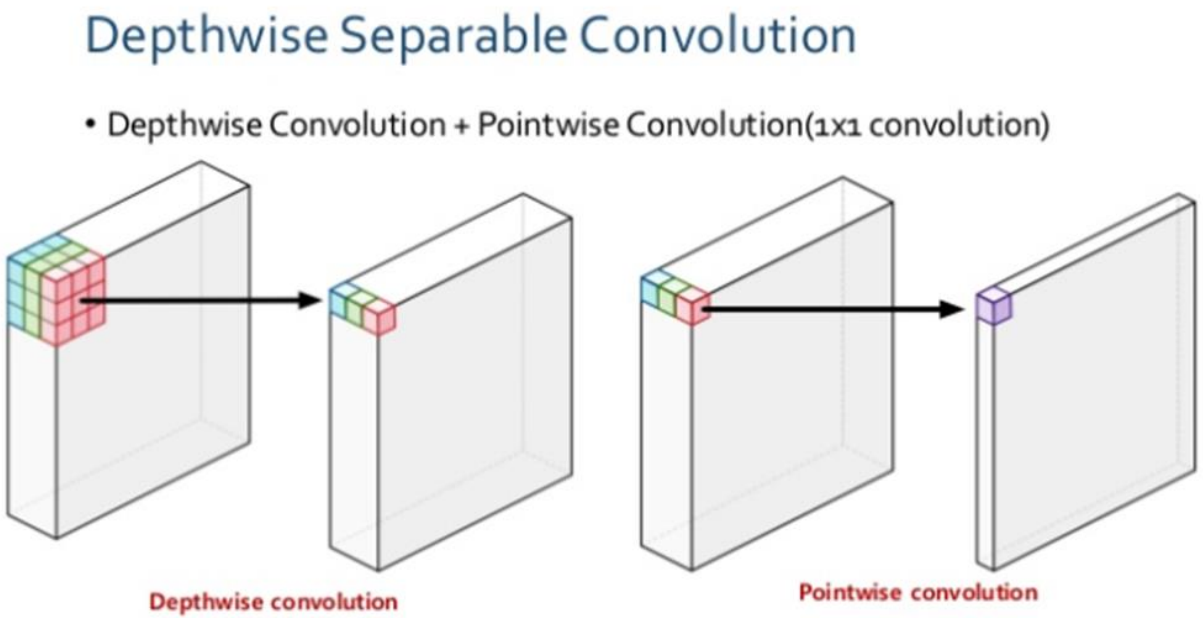
- 지금까지 CNN 모델은 정확도 향상을 위해
  모델의 크기를 키우는 방향으로 연구가 진행됨

- 이러한 모델은 모바일 장치에서 사용할 수 없음

- 모바일 장치에서 사용하기 위한 모델은
  - 충분한 정확도를 확보하면서
  - 낮은 연산 복잡도와
  - 낮은 에너지 소모량을 만족시키는
  - 작은 모델이어야 함

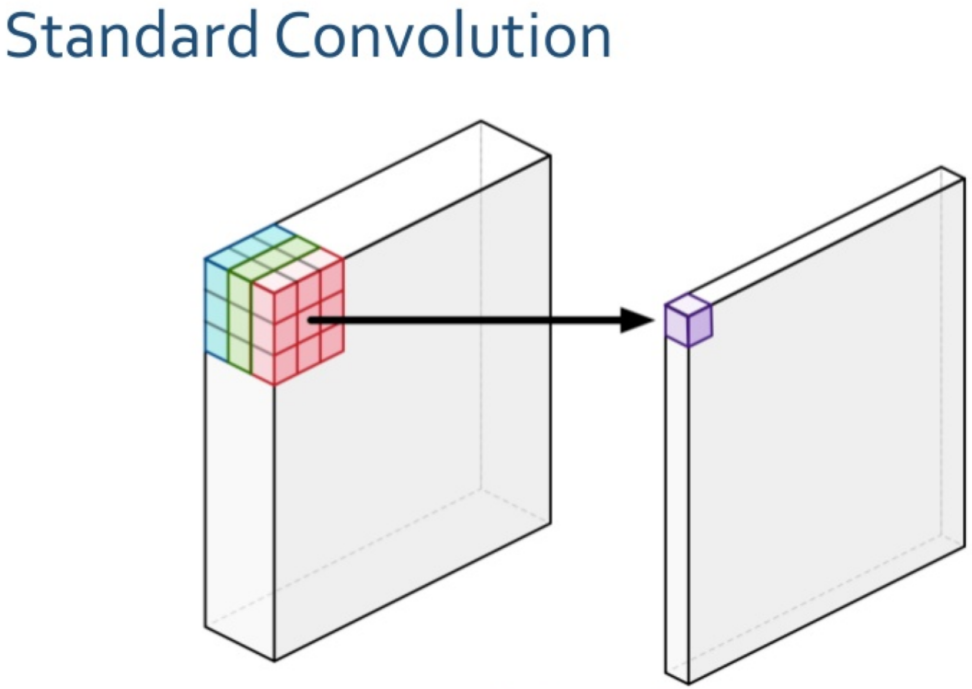
MobileNet (2/8)

- MobileNet은 작은 모델을 만들기 위해
  Depthwise Separable Convolution 연산을 이용



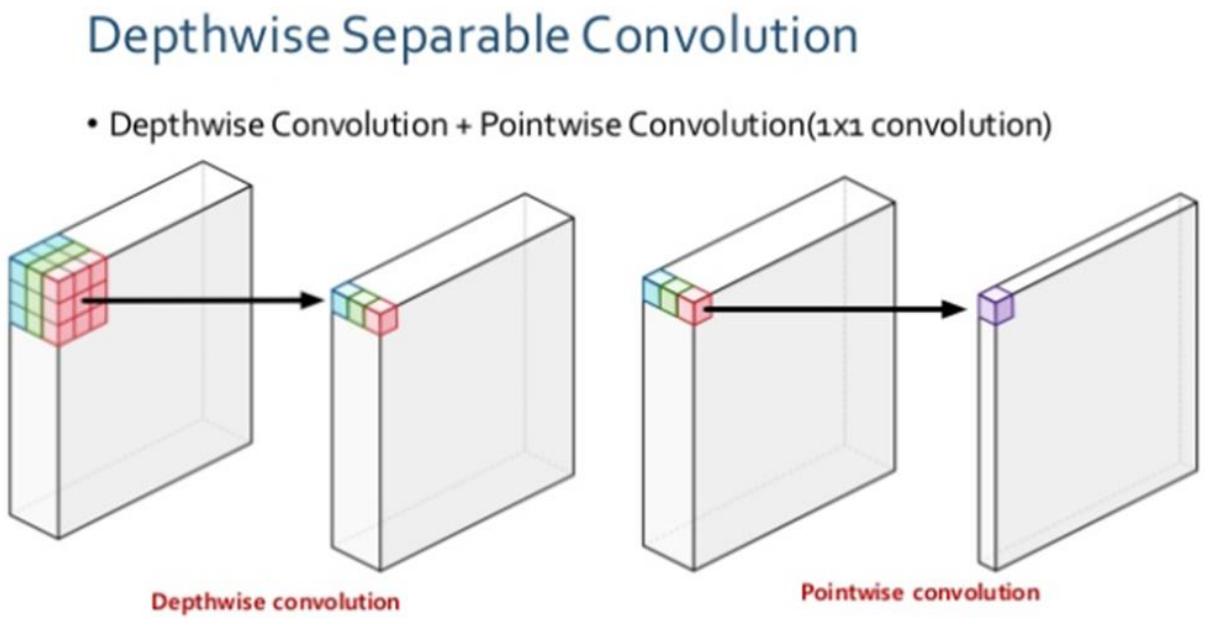
MobileNet (3/8)

- 기존 Convolution 연산은
  Spatial한 부분과 depth 부분을 한 번에 처리



MobileNet (4/8)

- Depthwise Separable Convolution 연산은
  Spatial한 부분과 depth 부분을 나누어 처리함



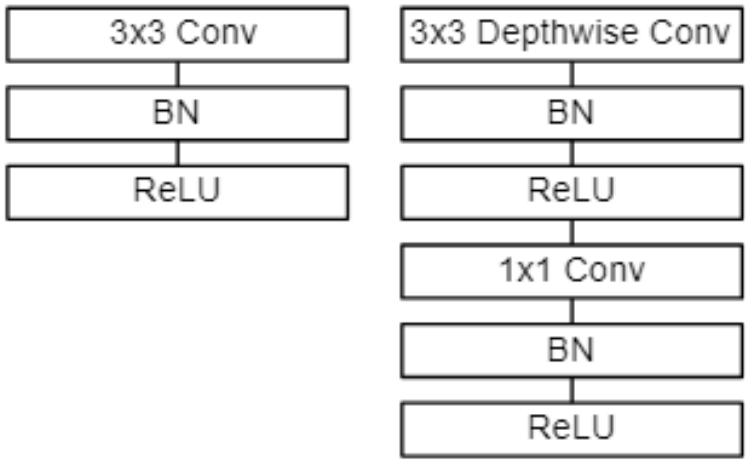
MobileNet (5/8)

- 기존 Convolution 연산이
  *"D_k x D_k x M x N x D_f x D_f"*의 연산 복잡도를 갖는다면
- Depthwise Separable Convolution 연산은
  "D_k x D_k x M x D_f x D_f + M x N x D_f x D_f" 의 연산 복잡도를 가짐
  - D_k: 커널 크기 / D_f: feature map 크기 / M: 입력 채널 수 / N: 출력 채널 수

- Depthwise Separable Convolution 연산이 동일한 영역을 처리하면서도
  기존 연산의 *1/N + 1/(Dk^2)* 만큼의 연산 복잡도를 가짐

MobileNet (6/8)

- Depthwise Separable Convolution은
  컨볼루션 연산 중간에 Batch normalization과 ReLU를 넣을 수 있음



MobileNet (7/8)

- MobileNet 레이어 구성

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

MobileNet (8/8)

- Depthwise Separable convolution 모델은 convolution 모델보다 정확도는 1% 떨어지지만 연산량은 8.5배, 파라미터의 수는 7배 감소함

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

- MobileNet은 연산량과 파라미터의 수가 작은 모델이지만 VGG, GoogLeNet과 비교해도 떨어지지 않는 정확도를 보여줌

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

ShuffleNet (1/10)

- MobileNet은 Depthwise Separable Convolution을 이용해 연산량과 파라미터의 수를 대폭 줄임

- MobileNet을 분석해보면 1x1 컨볼루션이 연산량과 파라미터의 대부분을 차지함
  - 이를 줄이기 위한 연구가 시작

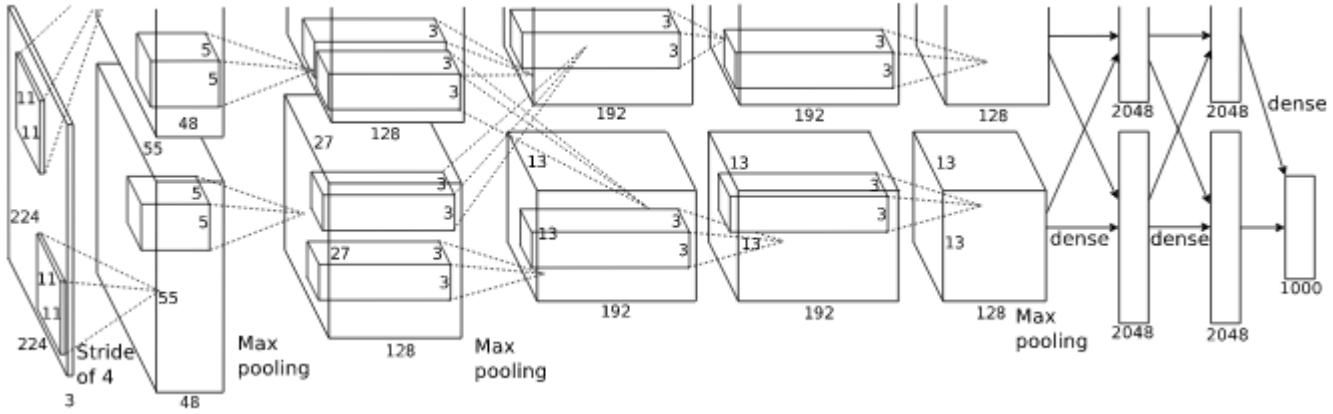
Table 2. Resource Per Layer Type

Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%

- ShuffleNet은 1x1 컨볼루션의 부하를 줄이기 위해 Grouped Convolution 기법을 사용

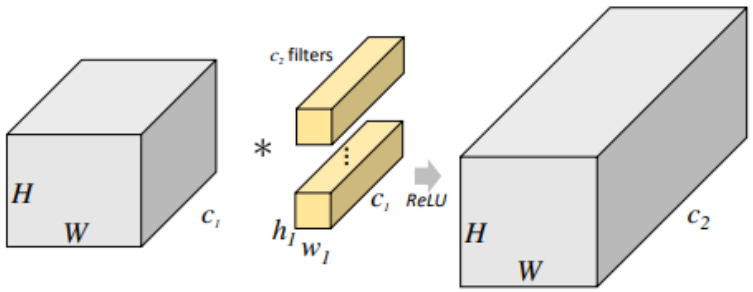
ShuffleNet (2/10)

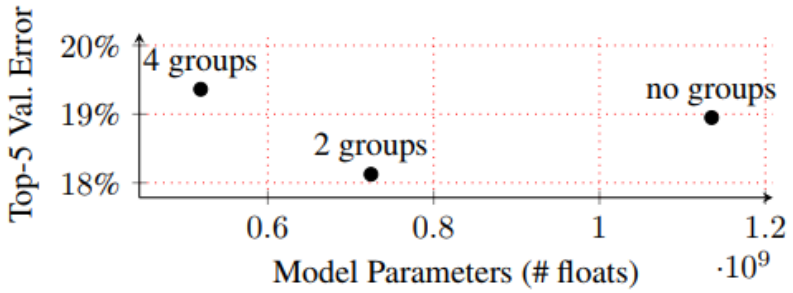
- Grouped Convolution은 AlexNet에서 처음 사용
  - AlexNet은 당시 GPU(GTX580) 성능의 한계로 인해 layer의 채널을 각 GPU가 절반씩 나누어 처리함



ShuffleNet (3/10)

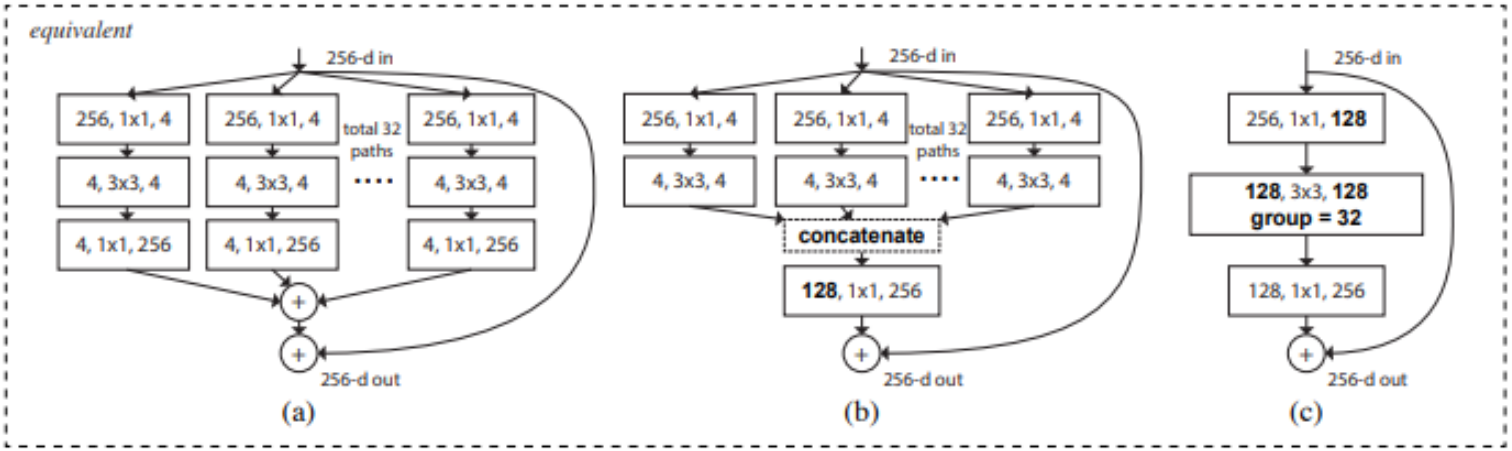
- 2016년 Yani Ioannou 등은 Grouped Convolution이 단순히 성능이 떨어지는 장치에서 대형 컨볼루션 연산을 진행하기 위한 Engineering Hack이 아님을 보여줌





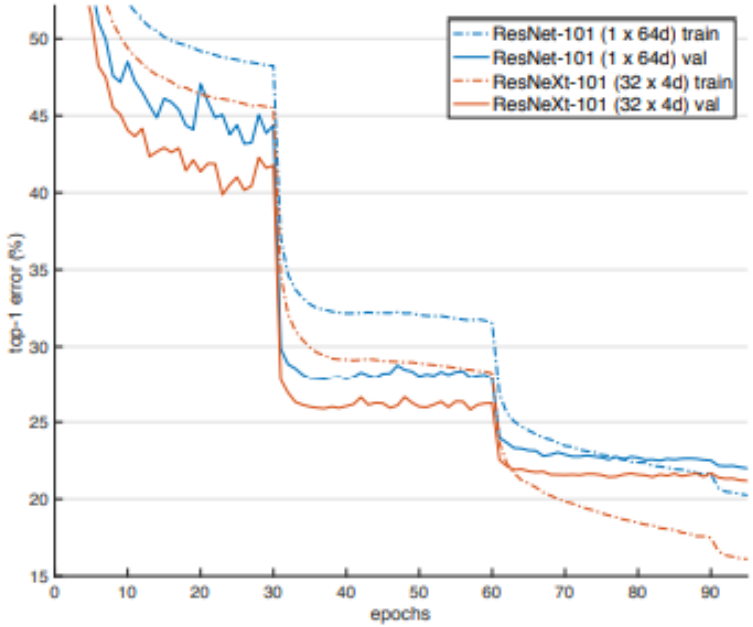
ShuffleNet (4/10)

- Grouped Convolution을 적극적으로 이용한 모델로 ResNeXt가 있음
  - ResNet과 Inception을 조합해 모델을 만든 후,
    Grouped Convolution을 사용



ShuffleNet (5/10)

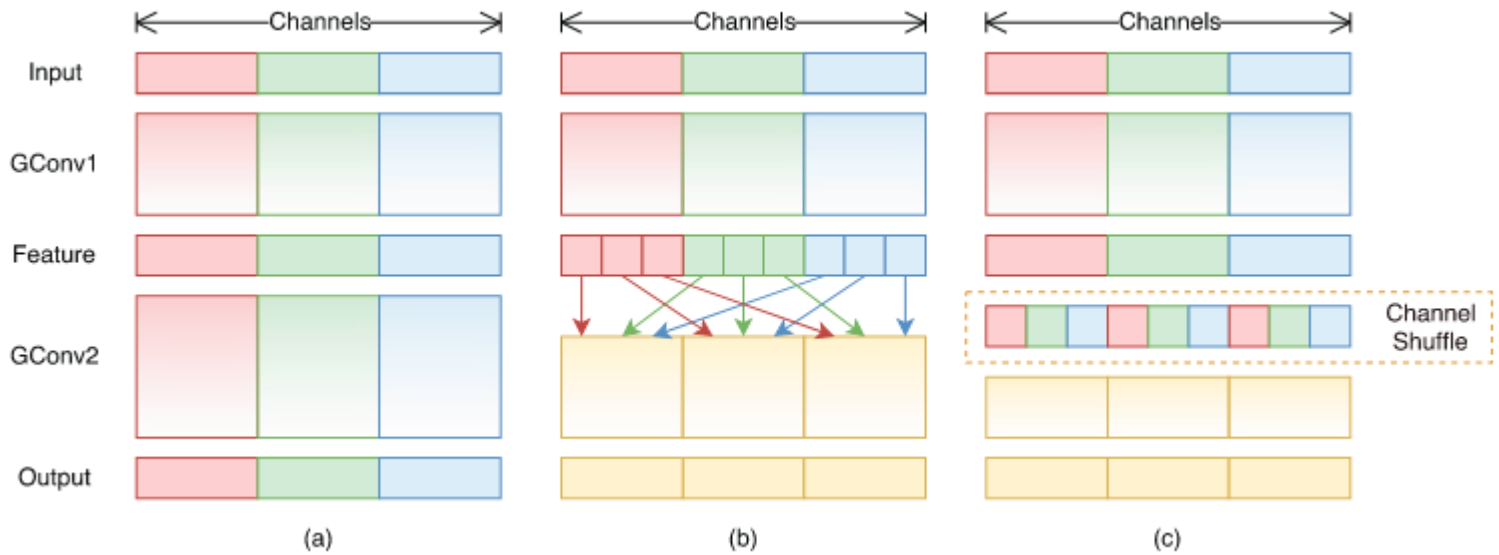
- ResNeXt는 이를 통해 효율적으로 ResNet보다 높은 성능을 확보



ShuffleNet (6/10)

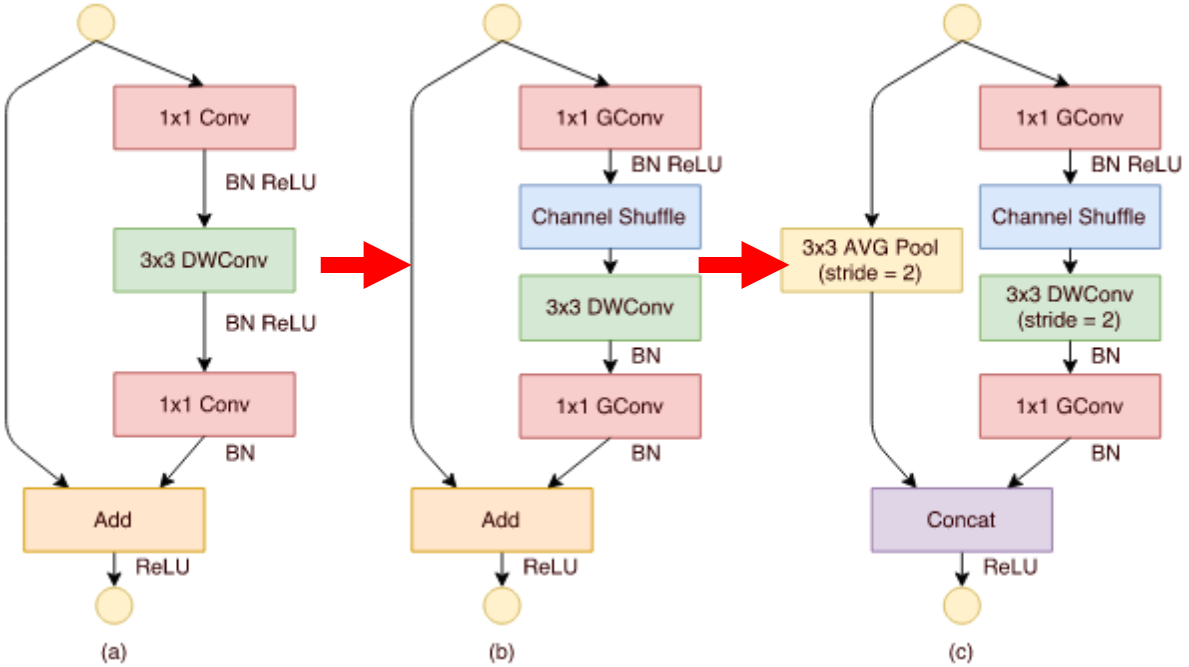
- ShuffleNet은 ResNeXt에 영감을 얻어
  Grouped Convolution을 MobileNet에 적용

- 동시에 연산 중간에 Group된 채널을 섞어주어
  각 Grouped Convolution이 줄어든 채널에 overfitting 되는 것을 방지



ShuffleNet (7/10)

- Grouped Convolution 연산 구조



ShuffleNet (8/10)

- ShuffleNet 레이어 구조

Layer	Output size	KSize	Stride	Repeat	Output channels (g groups)				
					$g = 1$	$g = 2$	$g = 3$	$g = 4$	$g = 8$
Image	224×224				3	3	3	3	3
Conv1	112×112	3×3	2	1	24	24	24	24	24
MaxPool	56×56	3×3	2						
Stage2	28×28		2	1	144	200	240	272	384
	28×28		1	3	144	200	240	272	384
Stage3	14×14		2	1	288	400	480	544	768
	14×14		1	7	288	400	480	544	768
Stage4	7×7		2	1	576	800	960	1088	1536
	7×7		1	3	576	800	960	1088	1536
GlobalPool	1×1	7×7							
FC					1000	1000	1000	1000	1000
Complexity					143M	140M	137M	133M	137M

ShuffleNet (9/10)

- Channel Shuffle은 Grouped Convolution의 성능을 향상시킴

Model	Cls err. (%, no shuffle)	Cls err. (%, shuffle)	Δ err. (%)
ShuffleNet 1x ($g = 3$)	34.5	**32.6**	1.9
ShuffleNet 1x ($g = 8$)	37.6	**32.4**	5.2
ShuffleNet 0.5x ($g = 3$)	45.7	**43.2**	2.5
ShuffleNet 0.5x ($g = 8$)	48.1	**42.3**	5.8
ShuffleNet 0.25x ($g = 3$)	56.3	**55.0**	1.3
ShuffleNet 0.25x ($g = 8$)	56.5	**52.7**	3.8

- 동일한 연산 복잡도를 갖는 MobileNet보다 높은 정확도를 보임

Model	Complexity (MFLOPs)	Cls err. (%)	Δ err. (%)
1.0 MobileNet-224	569	29.4	-
ShuffleNet 2× ($g = 3$)	524	**26.3**	3.1
ShuffleNet 2× (with *SE*[13], $g = 3$)	527	**24.7**	4.7
0.75 MobileNet-224	325	31.6	-
ShuffleNet 1.5× ($g = 3$)	292	**28.5**	3.1
0.5 MobileNet-224	149	36.3	-
ShuffleNet 1× ($g = 8$)	140	**32.4**	3.9
0.25 MobileNet-224	41	49.4	-
ShuffleNet 0.5× ($g = 4$)	38	**41.6**	7.8
ShuffleNet 0.5× (shallow, $g = 3$)	40	42.8	6.6

ShuffleNet (10/10)

- Snapdragon 820 싱글코어를 사용한 성능 비교
  - ShuffleNet이 더 적은 연산을 수행해 더 빠른 처리 속도를 보임

Model	Cls err. (%)	FLOPs	224×224	480×640	720×1280
ShuffleNet $0.5\times$ ($g = 3$)	43.2	38M	15.2ms	87.4ms	260.1ms
ShuffleNet $1\times$ ($g = 3$)	32.6	140M	37.8ms	222.2ms	684.5ms
ShuffleNet $2\times$ ($g = 3$)	26.3	524M	108.8ms	617.0ms	1857.6ms
AlexNet [22]	42.8	720M	184.0ms	1156.7ms	3633.9ms
1.0 MobileNet-224 [12]	29.4	569M	110.0ms	612.0ms	1879.2ms

결론

- 2012년 AlexNet의 등장 이후 CNN에 많은 발전이 일어남

- 성능을 높이기 위해 다양한 Convolution 연산과
  다양한 모델 구조가 나타남
  - Inception module
  - Residual Learning
  - Depthwise Separable Convolution
  - Grouped Convolution

- 각 모델에 적합한 경량화, 분할 기법을 고민하고
  계속되는 모델의 변화에 대응할 수 있는 방법에 대해 생각해야 함

References (1/2)

- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.
- Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).
- Szegedy, Christian, et al. "Going deeper with convolutions." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.
- Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *arXiv preprint arXiv:1312.4400* (2013).
- He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

References (2/2)

- Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." *arXiv preprint arXiv:1704.04861* (2017).

- Zhang, Xiangyu, et al. "Shufflenet: An extremely efficient convolutional neural network for mobile devices." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.

- Ioannou, Yani, et al. "Deep roots: Improving cnn efficiency with hierarchical filter groups." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017.

- Xie, Saining, et al. "Aggregated residual transformations for deep neural networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.