Convolutional Neural Networks

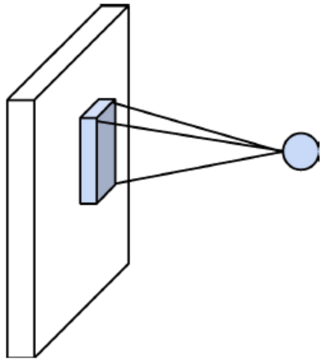
Longbin Jin

Artificial Inteligence
& Computer Vision
Laboratory

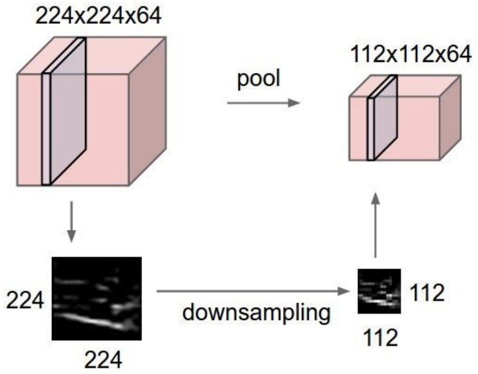
Slide reference: Stanford University cs231n, Fei-Fei Li

Components of CNNs

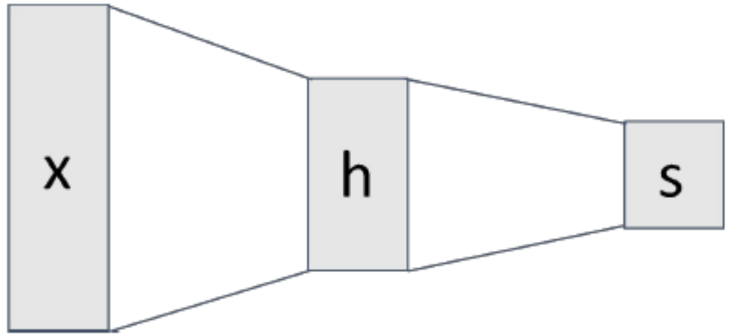
Convolution Layers



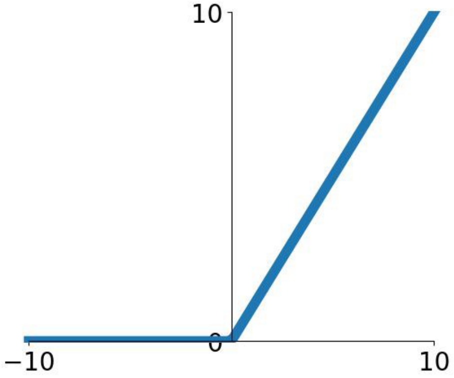
Pooling Layers



Fully-Connected Layers



Activation Function



Normalization

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

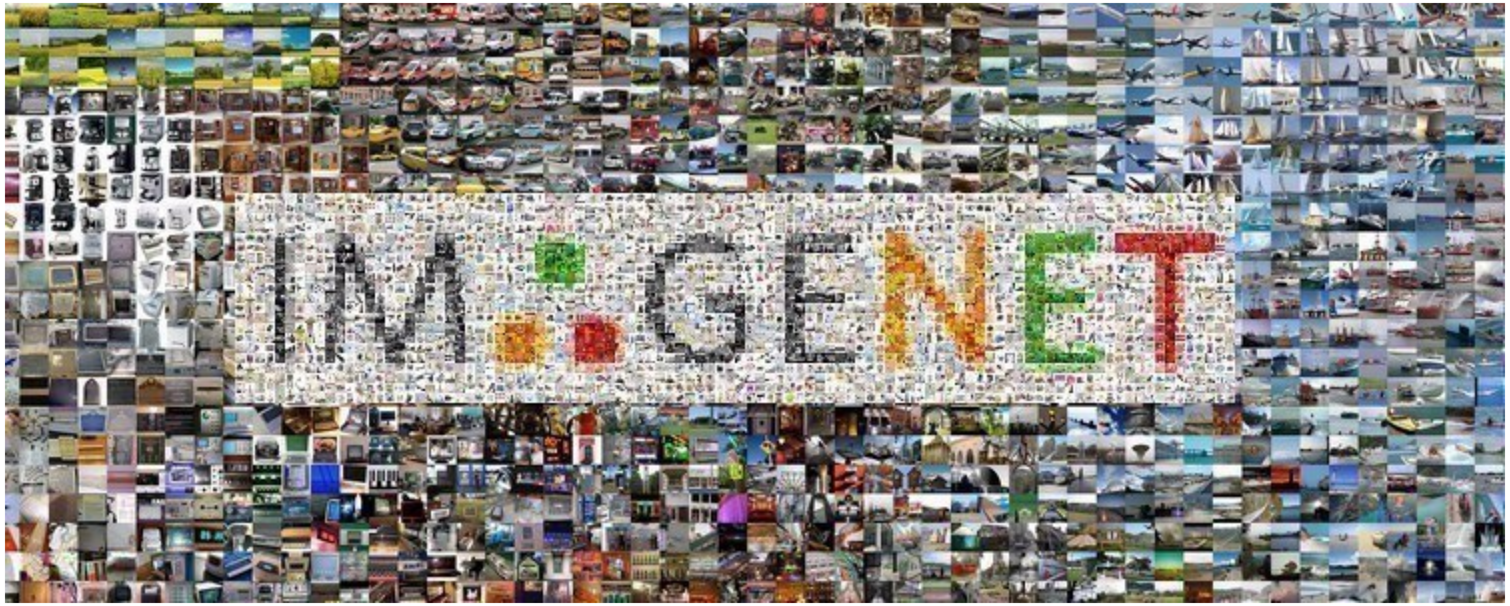
Question: How should we put them together?

CNN Architectures

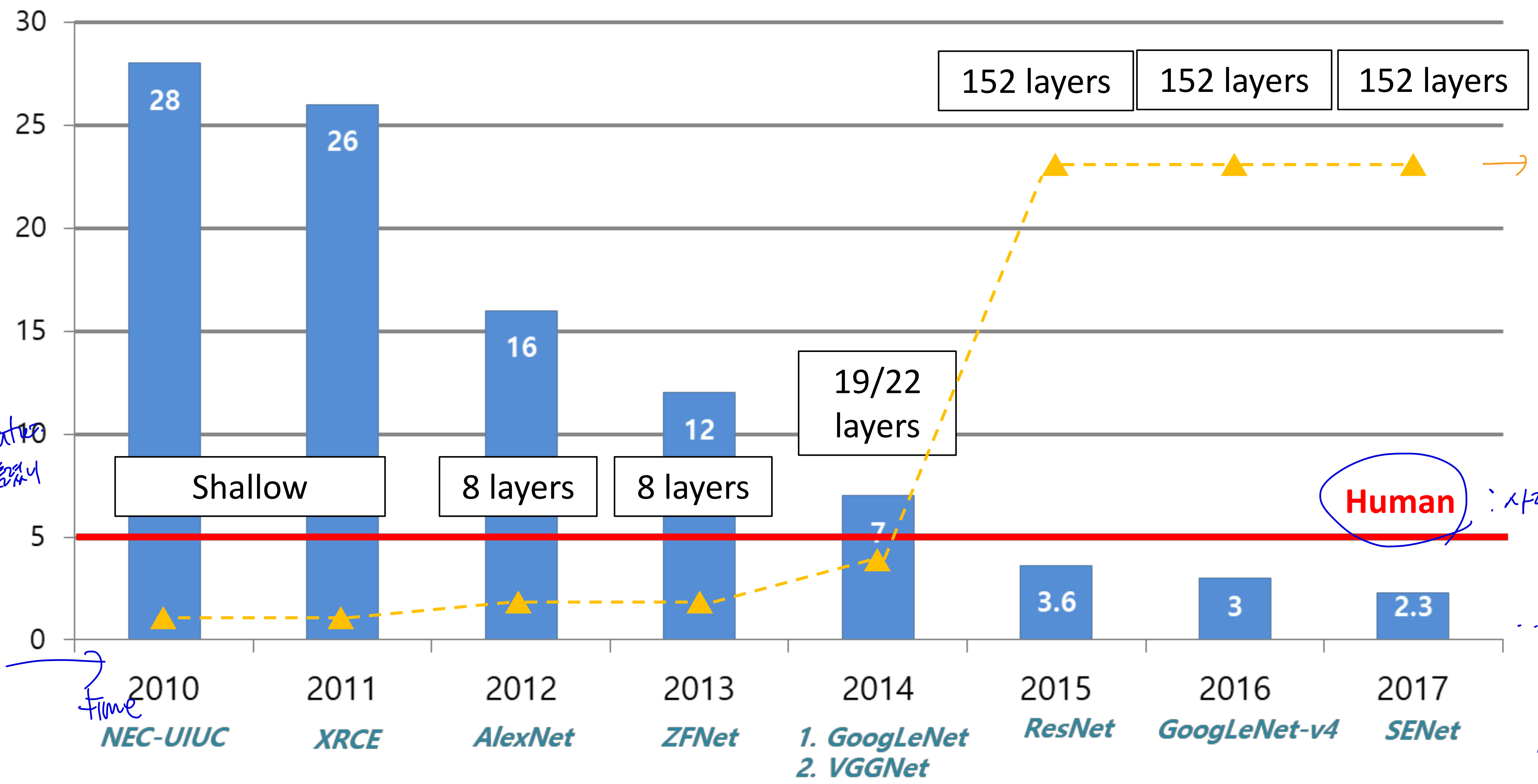
- AlexNet
- ZFNet
- VGG
- GoogleNet
- ResNet
- SENet
- MobileNets

ImageNet (dataset)

- ImageNet-1k
  - 1000 classes
  - 1000 images/class
- ImageNet-21k
  - 21841 classes
  - 14197122 images



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



error rate
: 몇 퍼센트 틀렸니

time

▲ : layer 개수
cnn layer + fully connected layer.
(cnn만 아님!)

: 사람.

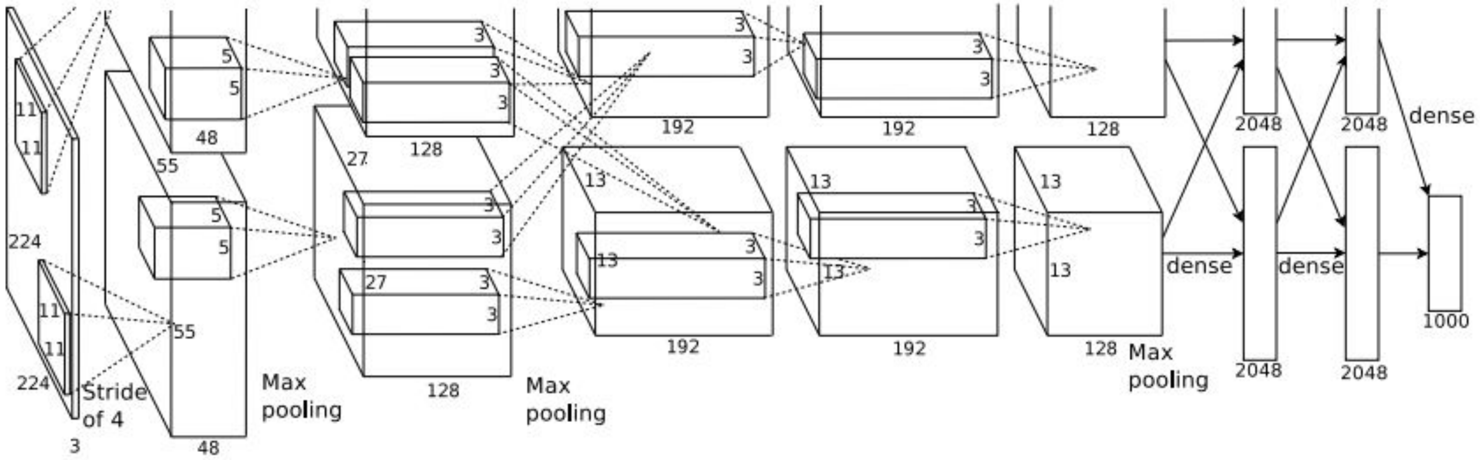
... 정점
Layer↑.
error↓

AlexNet

8개 layer

모든 layer 사이마다

relu 존재.

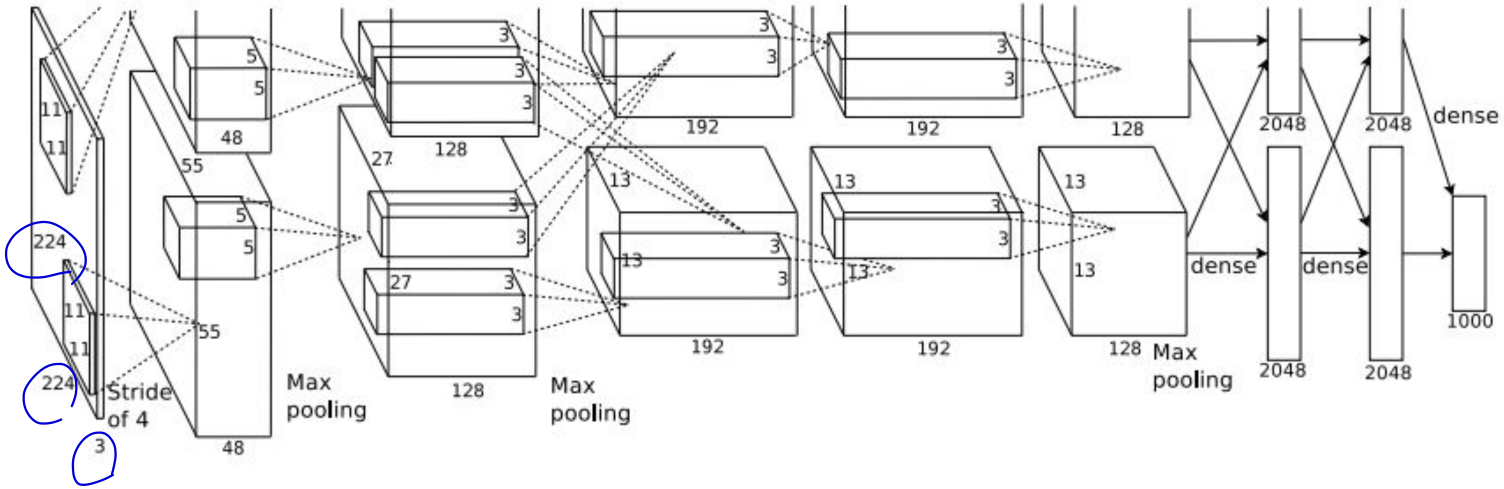


Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

Krizhevsky et al. 2012

AlexNet

<첫번째 Layer>



Input: 227x227x3 images → 원래 Image는 224×224×3 인데, 정보가 버려지니까? 227×227 ×3.

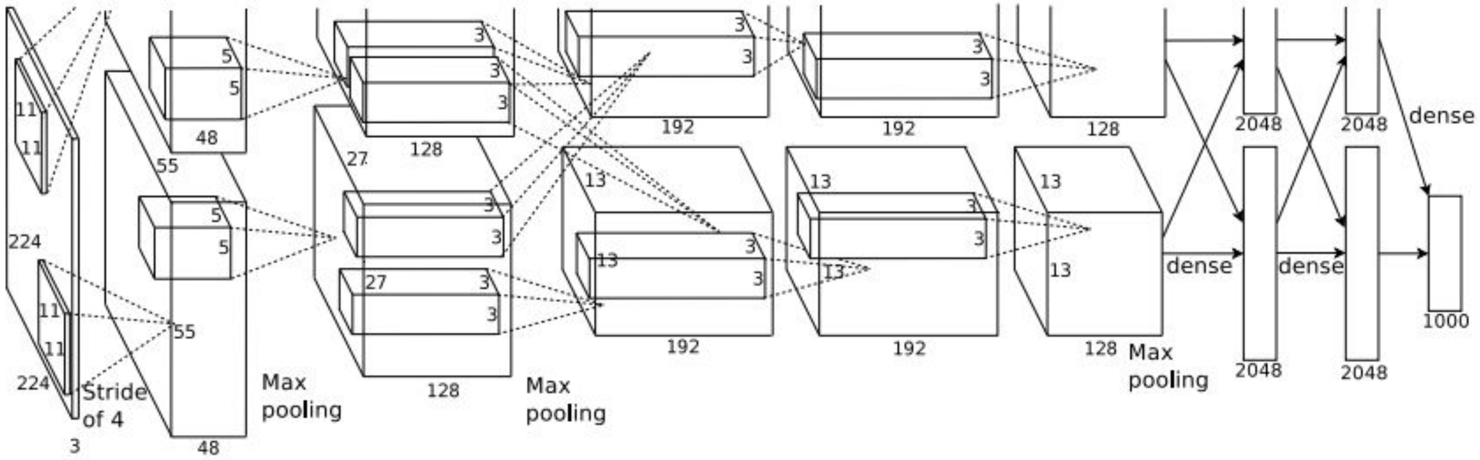
W' = (W - F + 2P) / S + 1

First layer (CONV1): 96 11x11 filters applied at stride 4 , padding = 0.

Q: what is the output volume size?

Krizhevsky et al. 2012

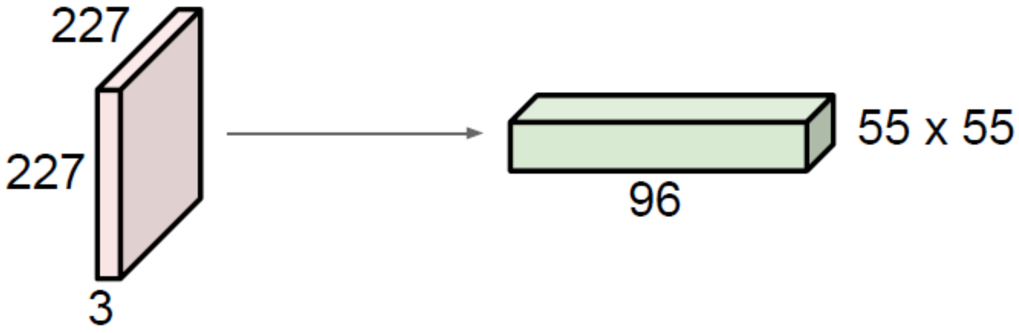
AlexNet



Input: 227x227x3 images

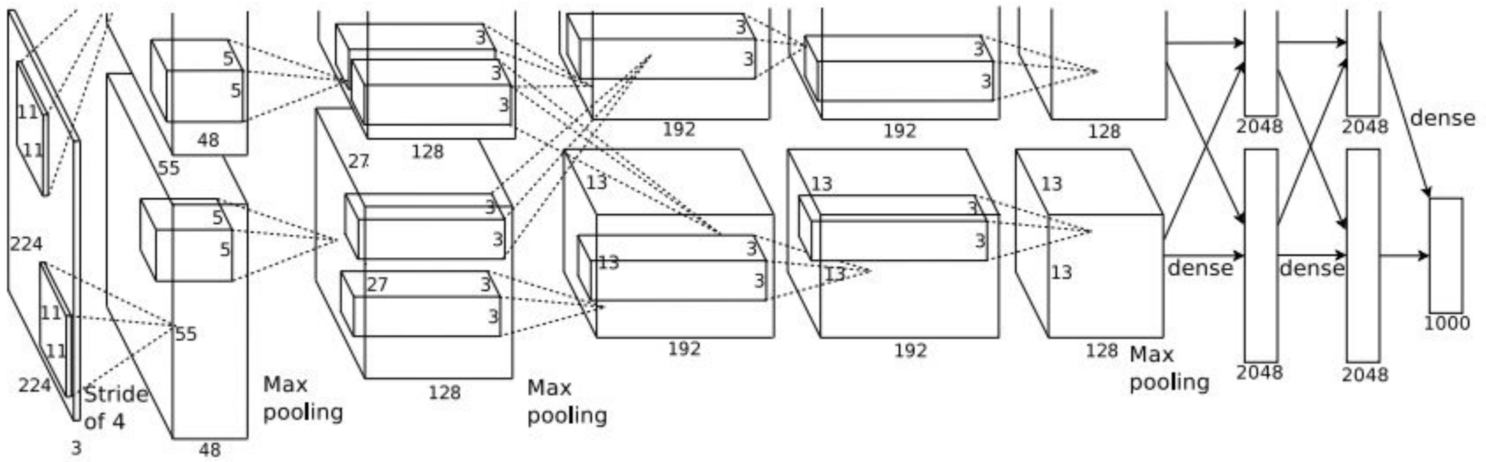
W’ = (W - F + 2P) / S + 1

First layer (CONV1): 96 11x11 filters applied at stride 4
Output volume **55x55x96**



Krizhevsky et al. 2012

AlexNet



Input: 227x227x3 images

∴ 11×11×3.

채널 차원 X ⇒ 입력 채널 차원과 같게 설정하면됨.

W' = (W - F + 2P) / S + 1

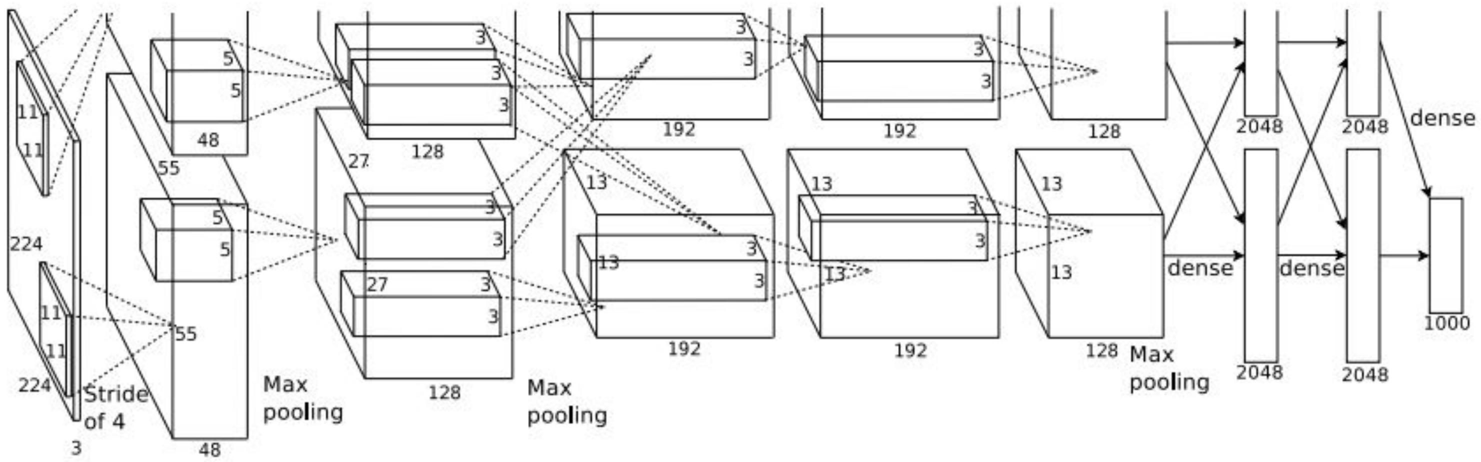
First layer (CONV1): 96 11x11 filters applied at stride 4

Output volume **55x55x96**

Q: What is the total number of parameters in this layer?

Krizhevsky et al. 2012

AlexNet



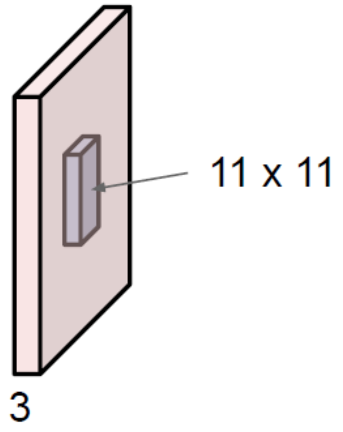
Input: 227x227x3 images

$$W' = (W - F + 2P) / S + 1$$

First layer (CONV1): 96 11x11 filters applied at stride 4

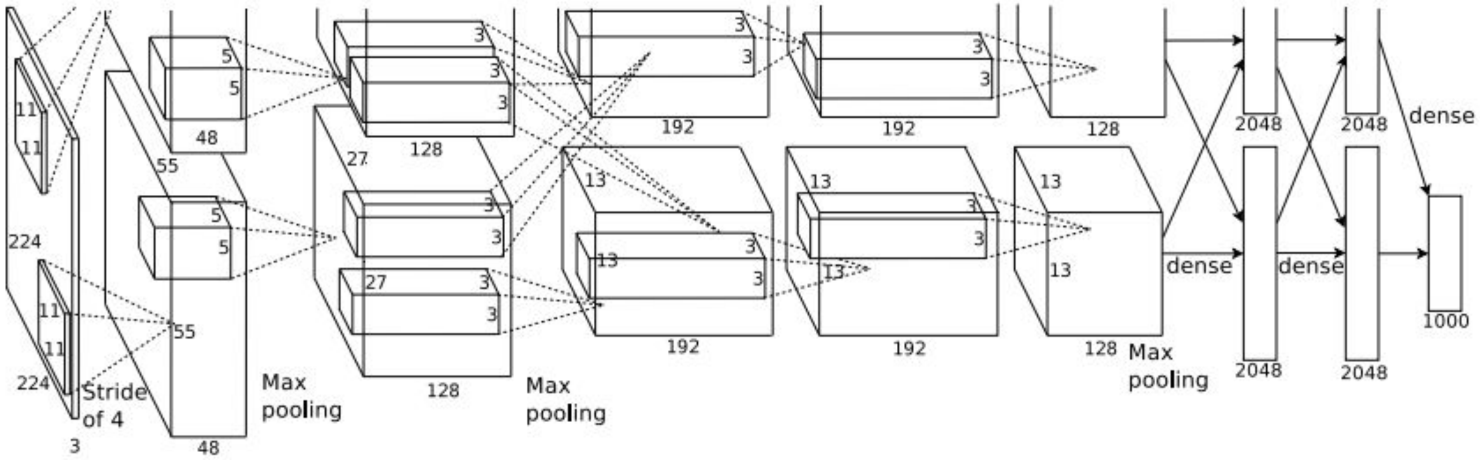
Output volume **55x55x96**

Parameters: (11*11*3 + 1)*96 = 35K



Krizhevsky et al. 2012

AlexNet



Input: 227x227x3 images
output

After CONV1: 55x55x96

input

Second layer (POOL1): 3x3 filters applied at stride 2

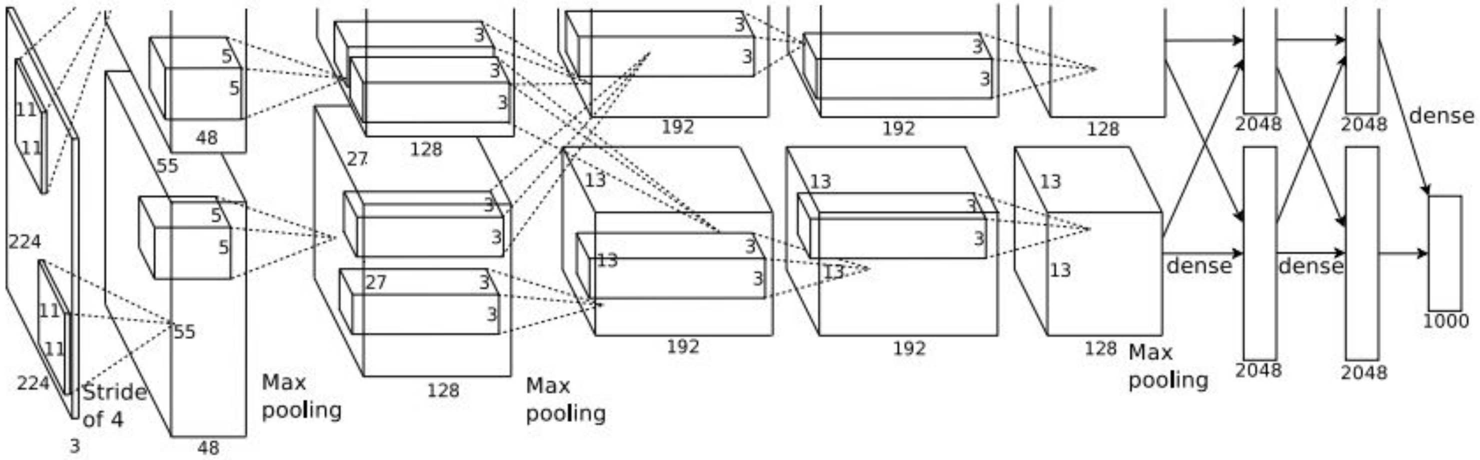
Q: what is the output volume size?

conv 계산과 똑같이

W' = (W - F + 2P) / S + 1

Krizhevsky et al. 2012

AlexNet



Input: 227x227x3 images

After CONV1: 55x55x96

W’ = (W - F + 2P) / S + 1

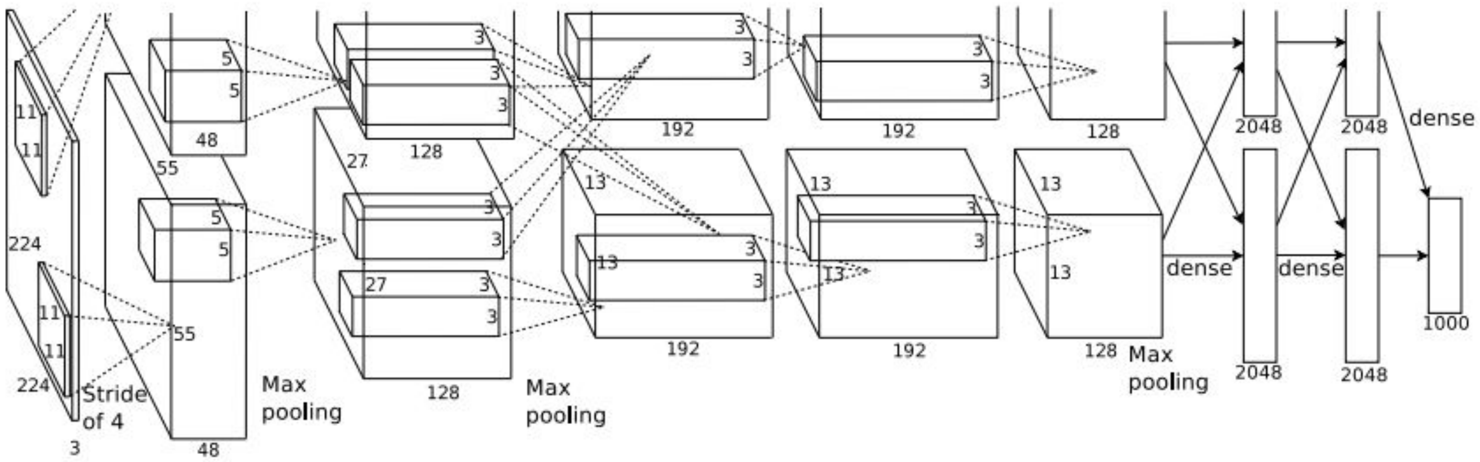
Second layer (POOL1): 3x3 filters applied at stride 2

Output volume: 27x27x96 → Pooling은 채널 차원을 건드리지 X

Q: what is the number of parameters in this layer? ⇒ Pooling layer에는 parameter 존재 XX!

Krizhevsky et al. 2012

AlexNet



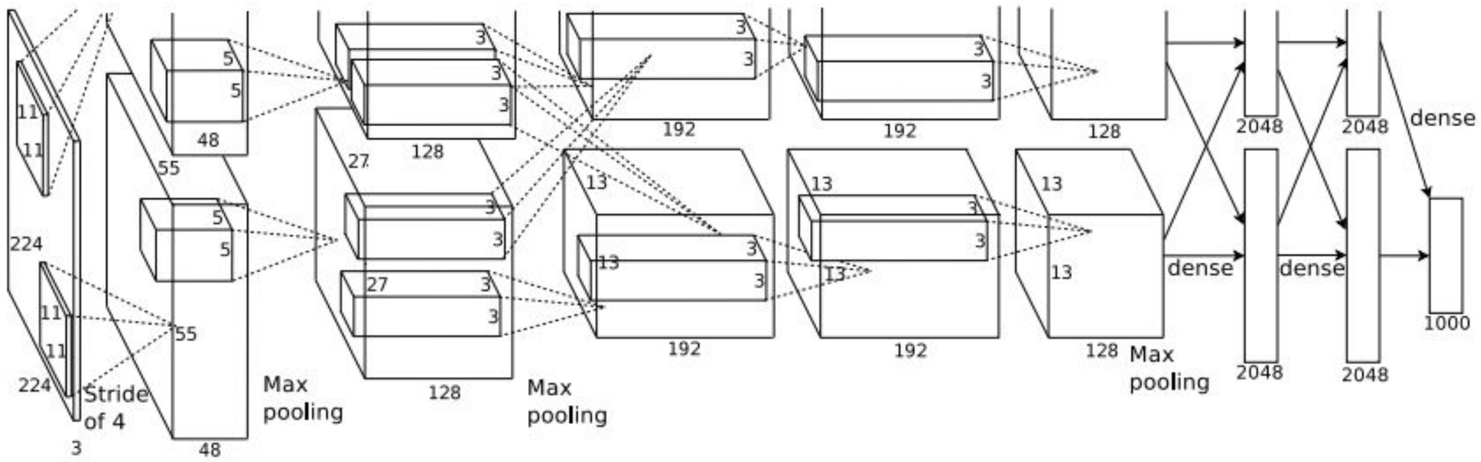
Input: 227x227x3 images
After CONV1: 55x55x96

W' = (W - F + 2P) / S + 1

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

Krizhevsky et al. 2012

AlexNet



Input: 227x227x3 images

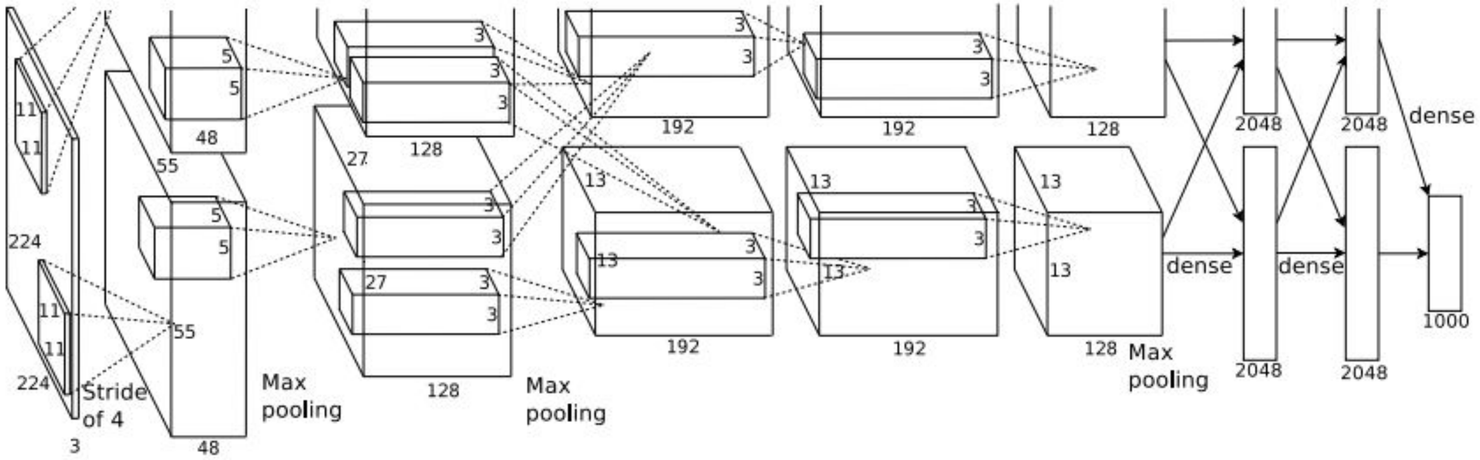
After CONV1: 55x55x96

After POOL1: 27x27x96

...

Krizhevsky et al. 2012

AlexNet



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0 → 인접한 픽셀 사이에 중복되는 특징이 있어서, 큰 영역에서 추출함.
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

filter size ↓↓.
→ 고급 특징이 아래에서 나오니까 영역을 줄여나감.

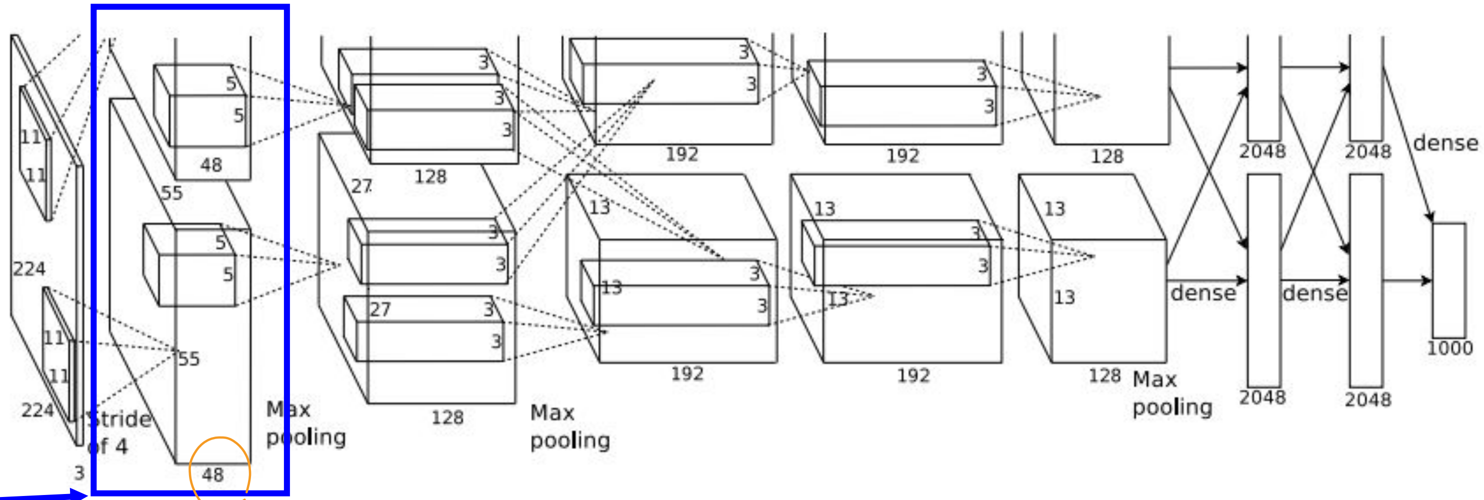
* stride 1, pad 1 이면
입력 size == 출력 size.

6x6x256을 넣을 때,
9천 몇 차원을 4천 몇 차원으로,
다시 1000 차원으로 넣음.
(3차원 아님!)

출력값, parameter 값 계산할 수 있어야 함!

Krizhevsky et al. 2012

AlexNet



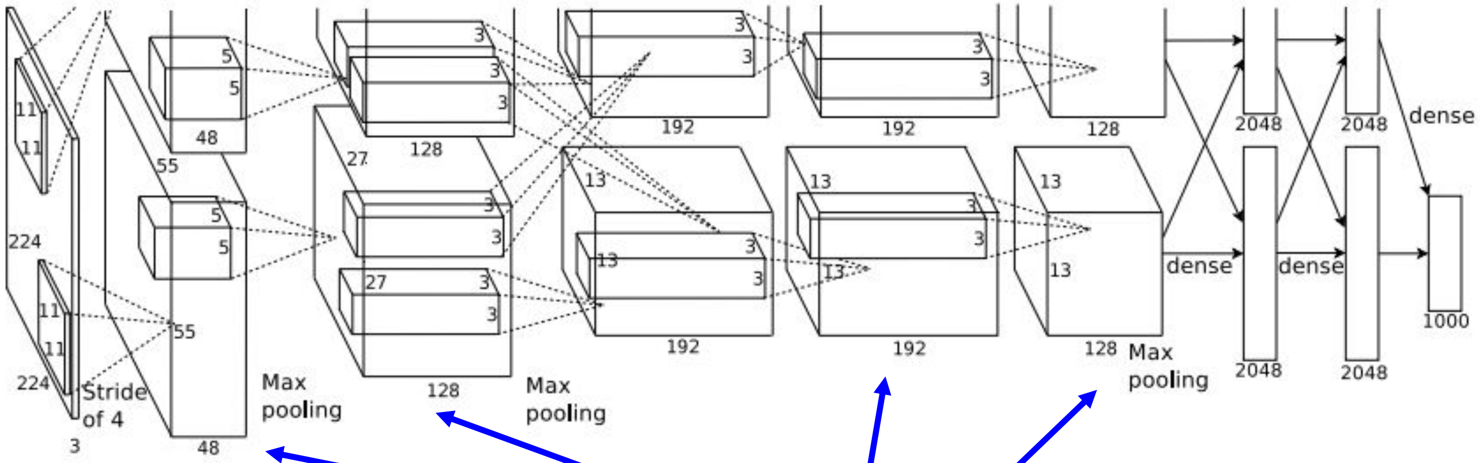
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
$\boxed{\text{[55x55x96]}}$ CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

[55x55x48] x 2 → 예전에는 GPU 용량이 적어서 나눠서 계산함. (Group Conv라고 생각해도 무방)

Historical note: Trained on GTX 580 GPU with only 3 GB of memory. Network spread across 2 GPUs, half the neurons (feature maps) on each GPU.

Krizhevsky et al. 2012

AlexNet



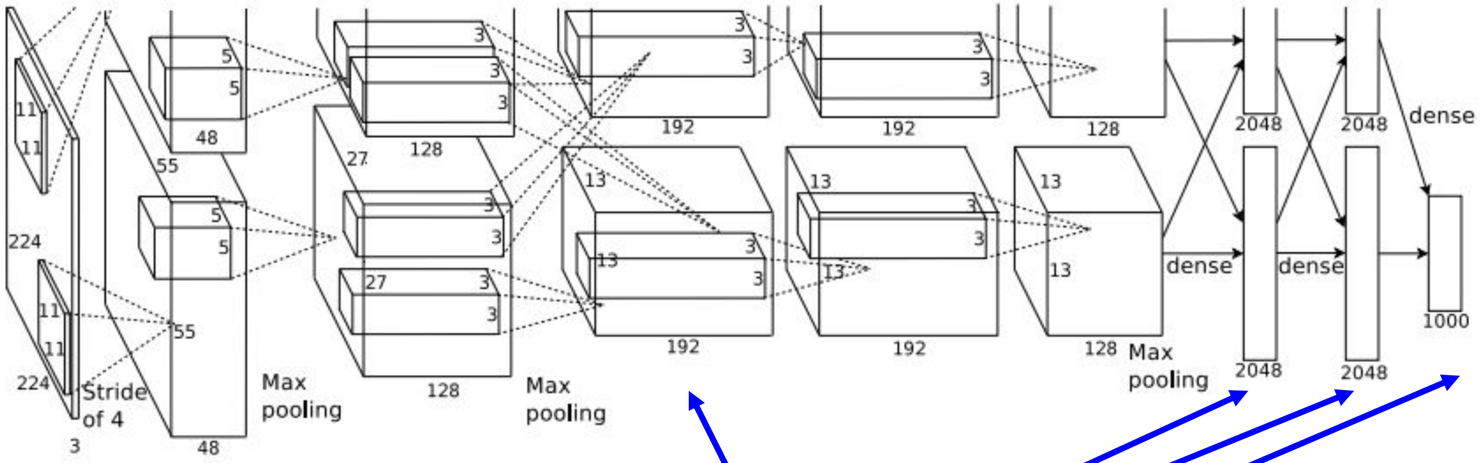
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

CONV1, CONV2, CONV4, CONV5:
Connections only with feature maps on same GPU

group conv?

Krizhevsky et al. 2012

AlexNet



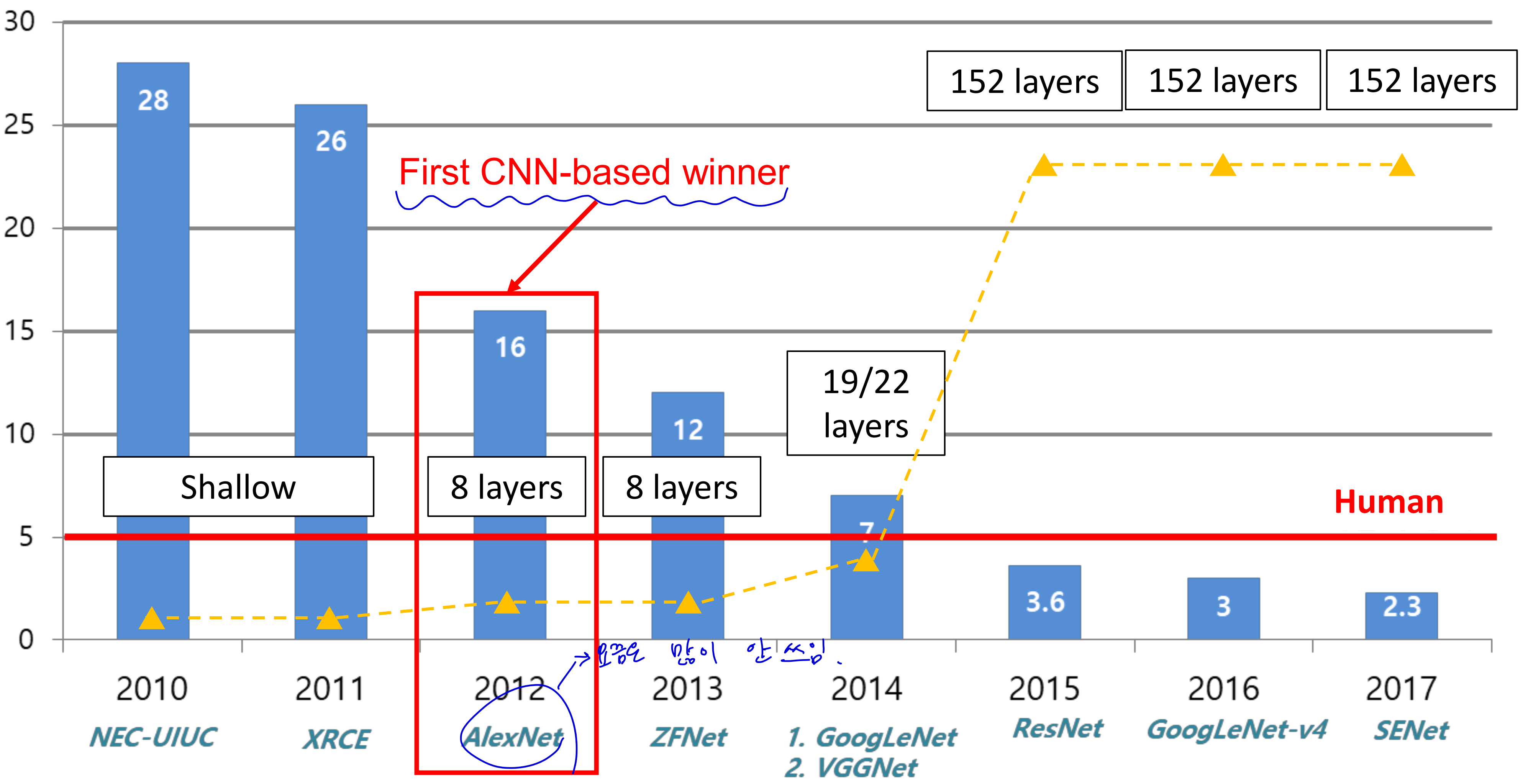
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

CONV3, FC6, FC7, FC8:
Connections with all feature maps in preceding layer, communication across GPUs

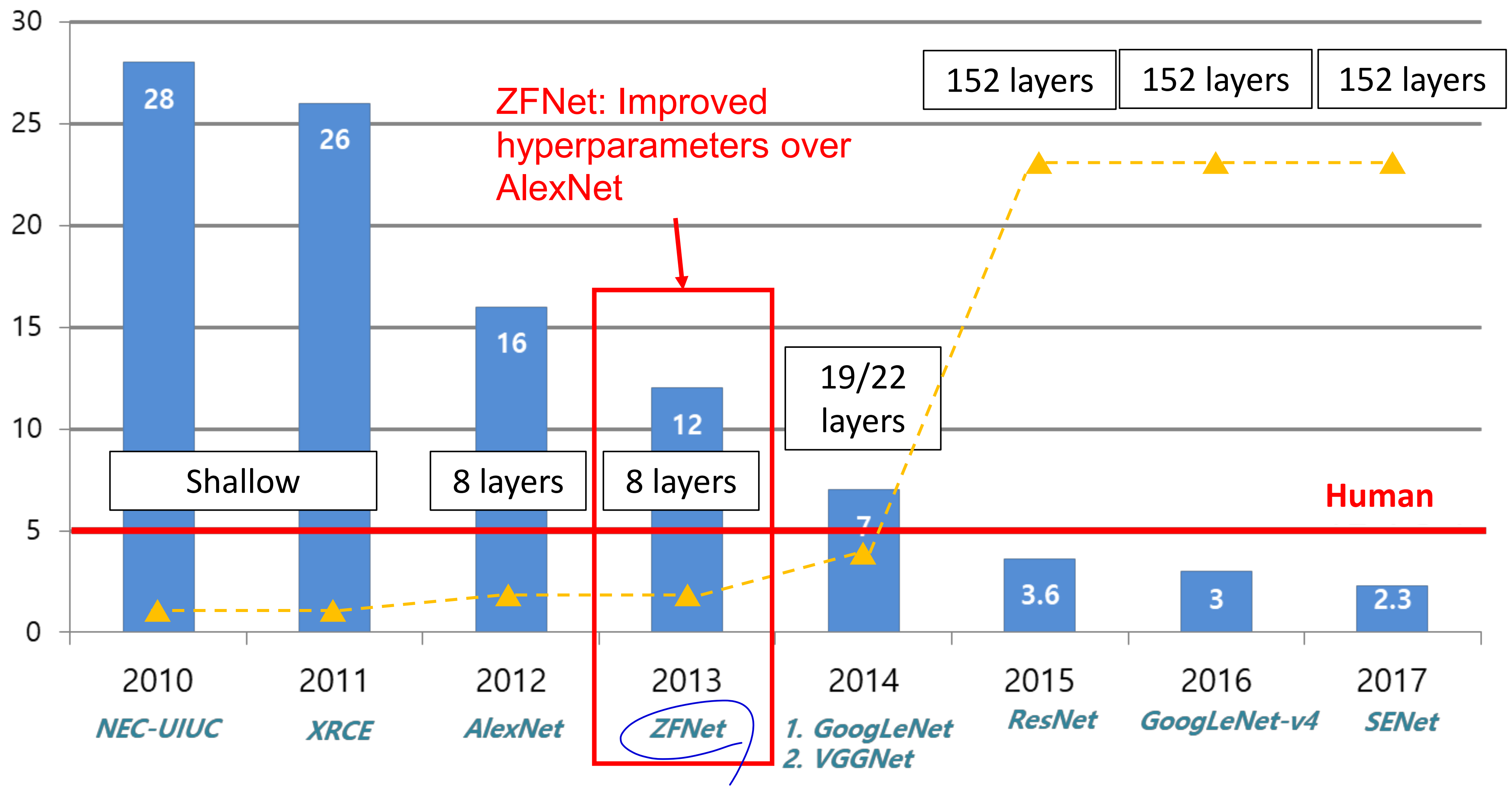
跨组 group ×

Krizhevsky et al. 2012

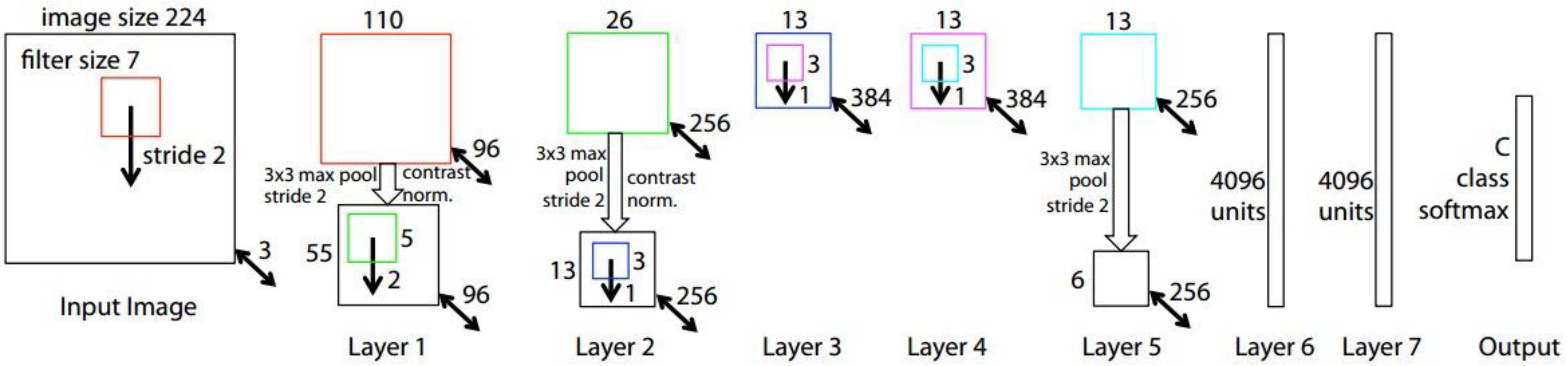
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ZFNet



AlexNet but:

AlexNet ZFNet.

CONV1: change from (11x11 stride 4) to (7x7 stride 2)

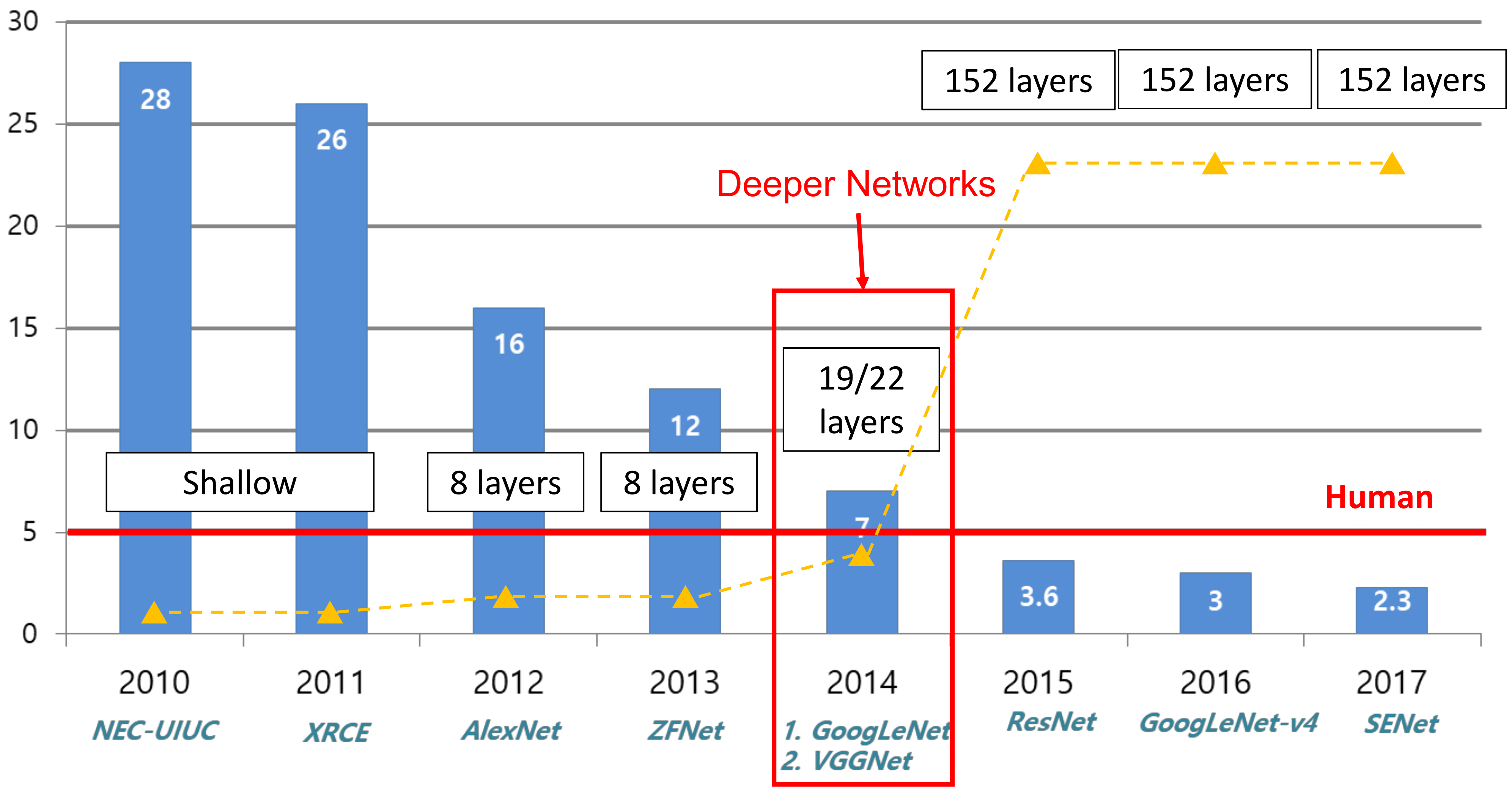
CONV2: change from (5x5 stride 1) to (5x5 stride 2)

⇒ 두 개의 다른 점은
이거 2개뿐.

ImageNet top 5 error: 16.4% -> 11.7%

Zeiler and Fergus, 2013

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



VGGNet

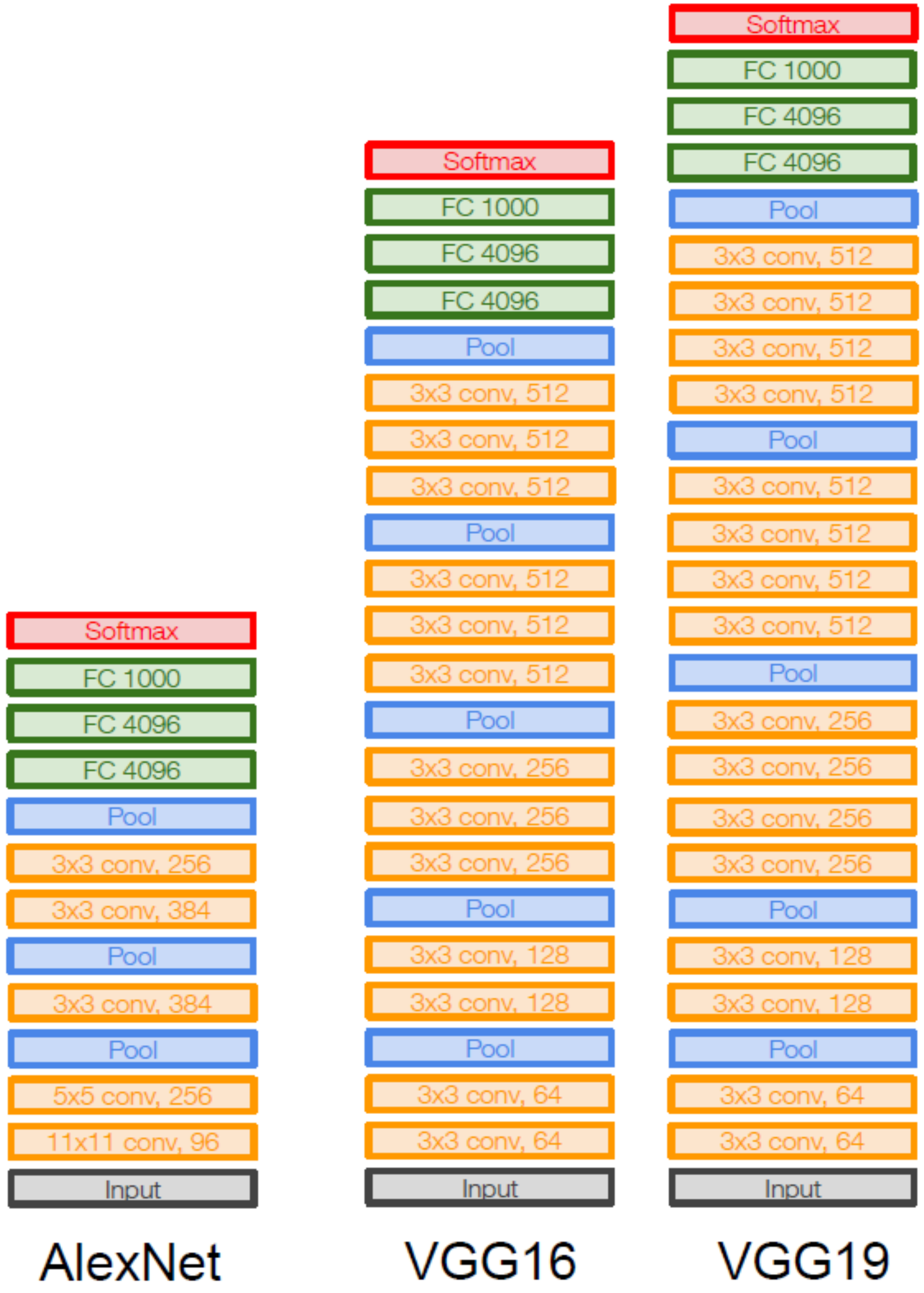
Alex 에서 사용했던 11x11 같은 큰 커널 사이즈 사용 X.

Small filters, Deeper networks

8 layers (AlexNet) 2배
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13 (ZFNet)
-> 7.3% top 5 error in ILSVRC'14



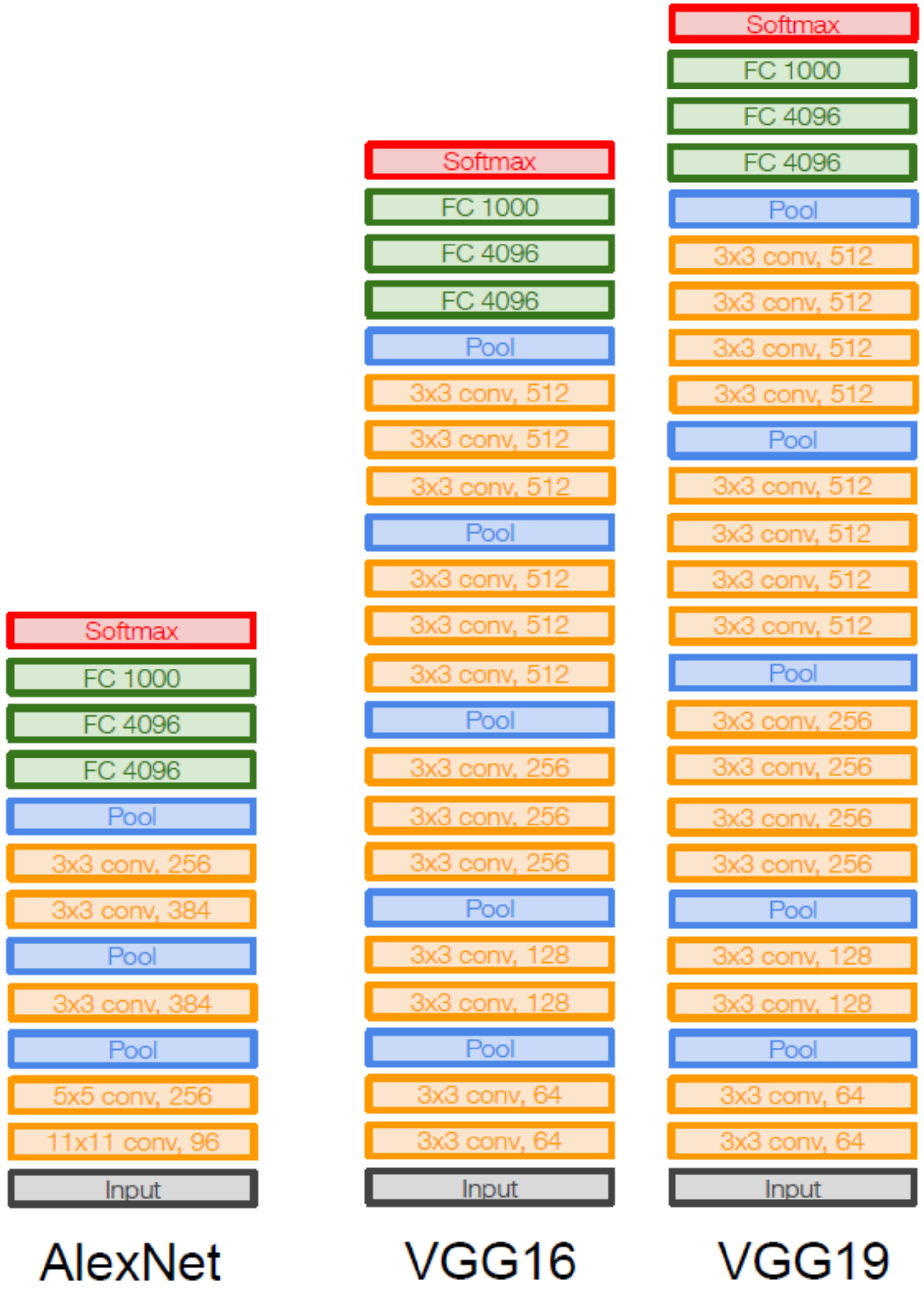
Simonyan and Zisserman, 2014

VGGNet

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same effective receptive field as one 7x7 conv layer

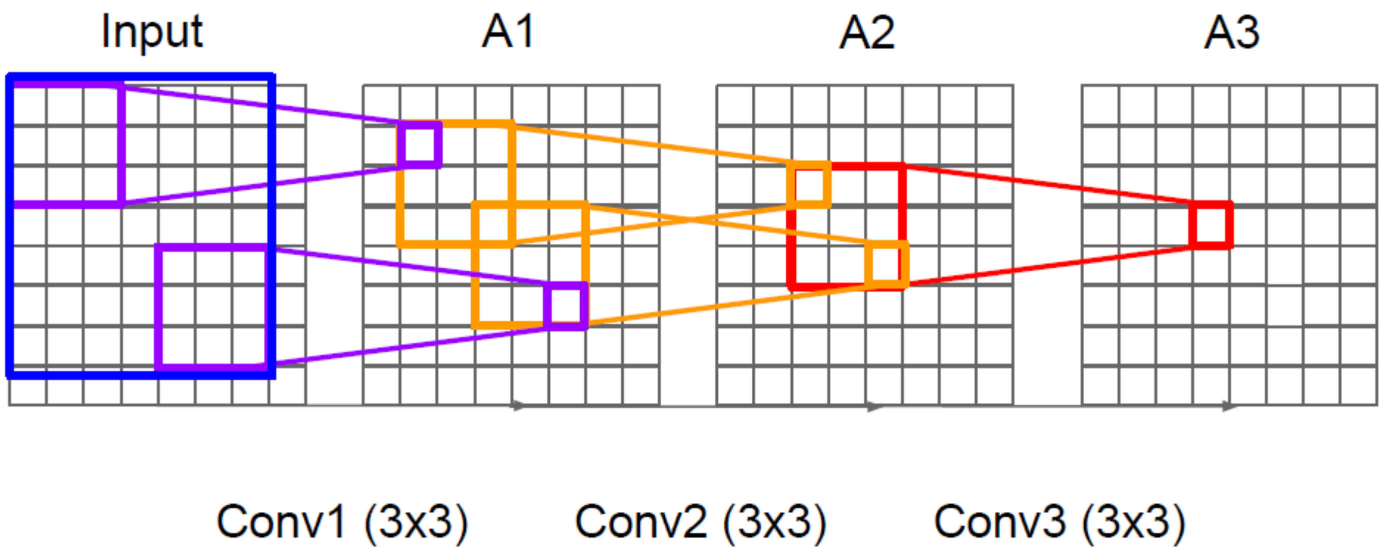
수용영역



Simonyan and Zisserman, 2014

VGGNet

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



Simonyan and Zisserman, 2014

VGGNet

Q: Why use smaller filters? (3x3 conv)

Layer들 쌓으면
성능이 좋아지는 이유.

↓

중간중간 relu(non-
linear)가 붙으니까,
7x7 한번 사용보다
3x3을 3번 쌓으면
2개 붙으니까.
그리고 param도 줄어듦.

Stack of three 3x3 conv (stride 1) layers has same effective receptive field as one 7x7 conv layer

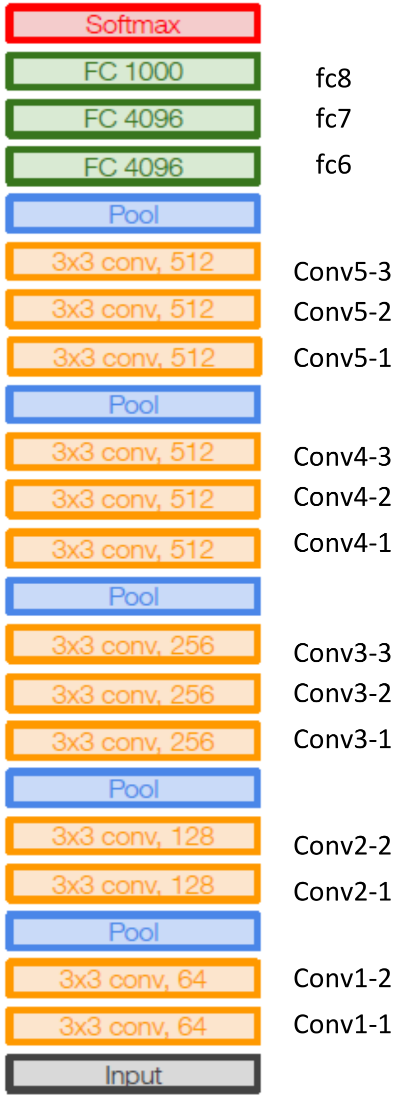
But deeper, more non-linearities

And fewer parameters: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer

Simonyan and Zisserman, 2014

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

* 출력?
* filter 크기



VGG16

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
(1억)
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
TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

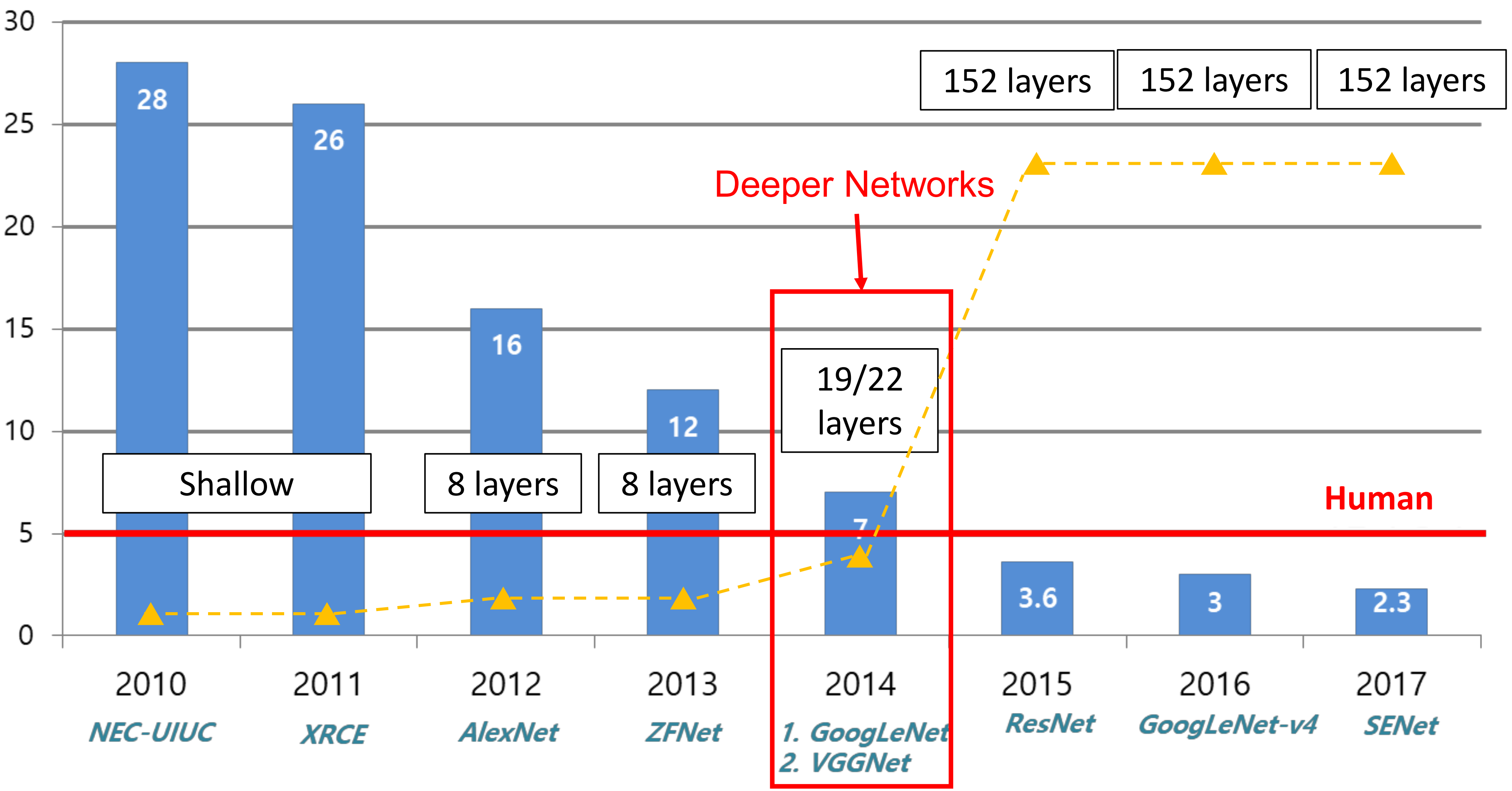
Note:
Most memory is in early CONV

여기가 제일 크다.
점점 작아짐.

Most params are in late FC

Conv param 다 합쳐도, fully connected param을 못이김.
So, 단점이다. 오래걸림.

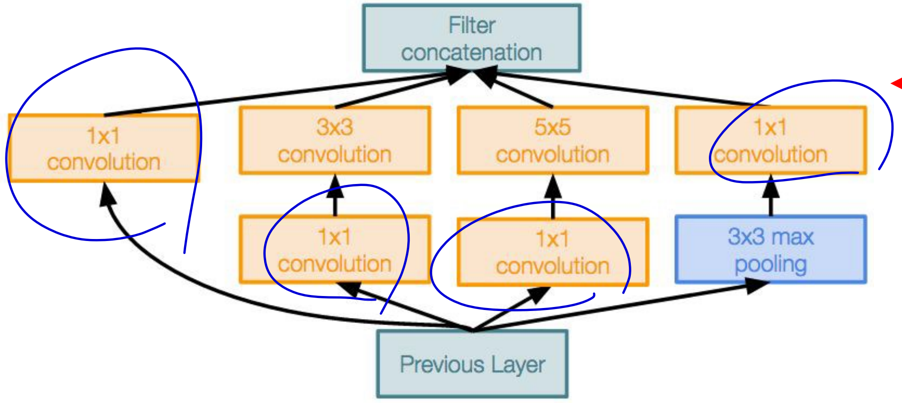
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

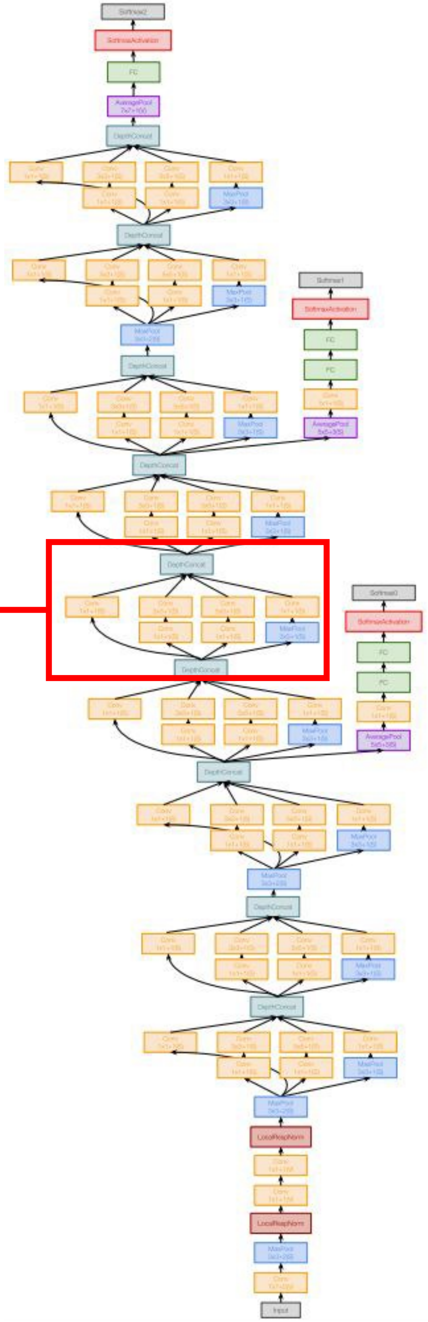


GoogLeNet

이건 계속 쌓음.

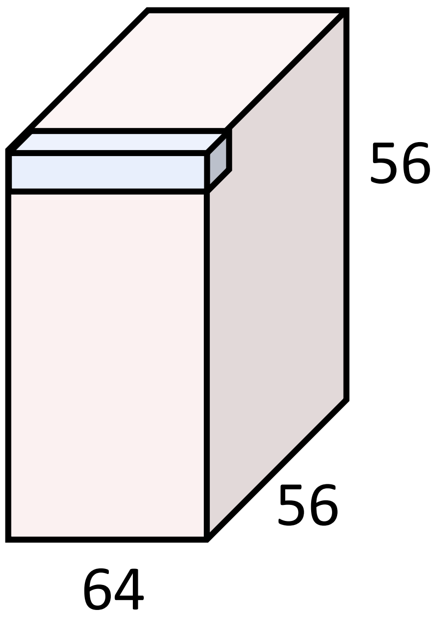
"Inception module": design a good local network topology (network within a network) and then stack these modules on top of each other





Szegedy et al., 2014

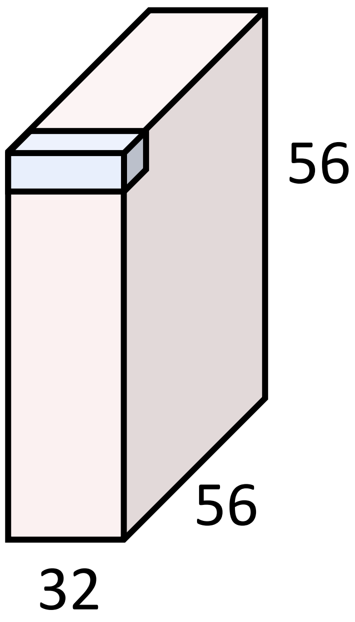
1x1 convolution layers make perfect sense



원래 채널 정보를 늘리거나 줄일 때 1x1 filter 사용

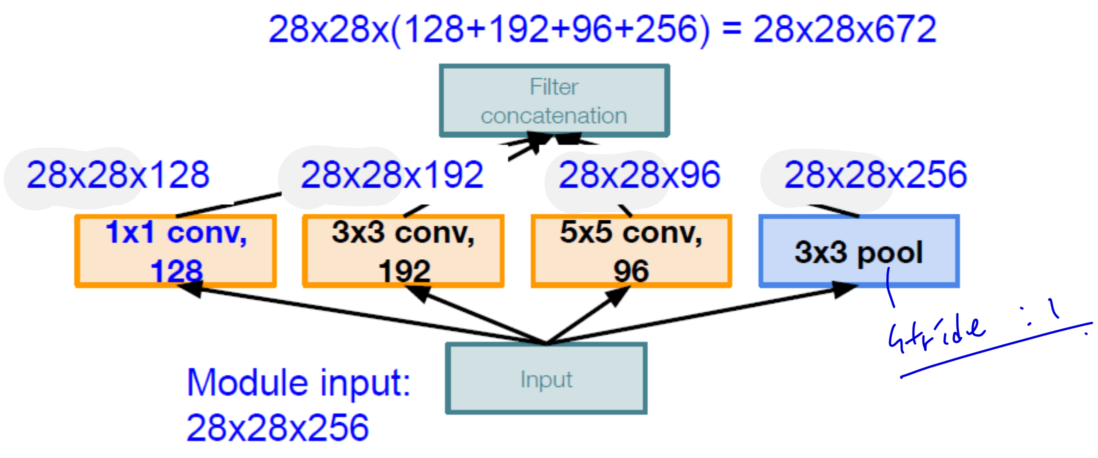
1x1 CONV
with 32 filters

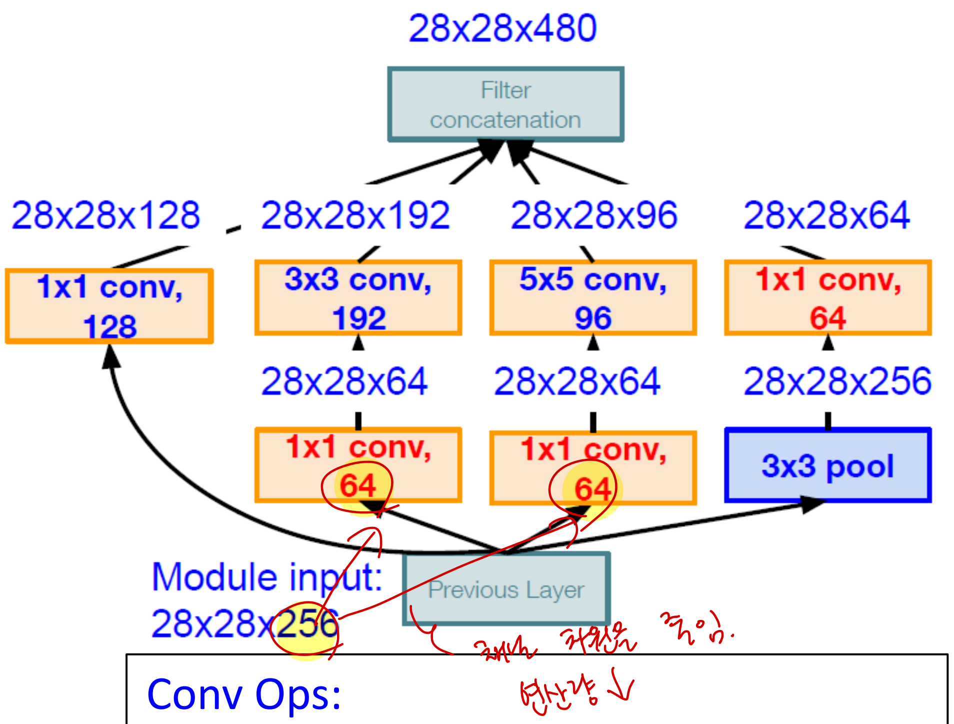
(each filter has size 1x1x64, and performs a 64-dimensional dot product)



GoogLeNet

★ 곱하기 연산이 몇번인지.





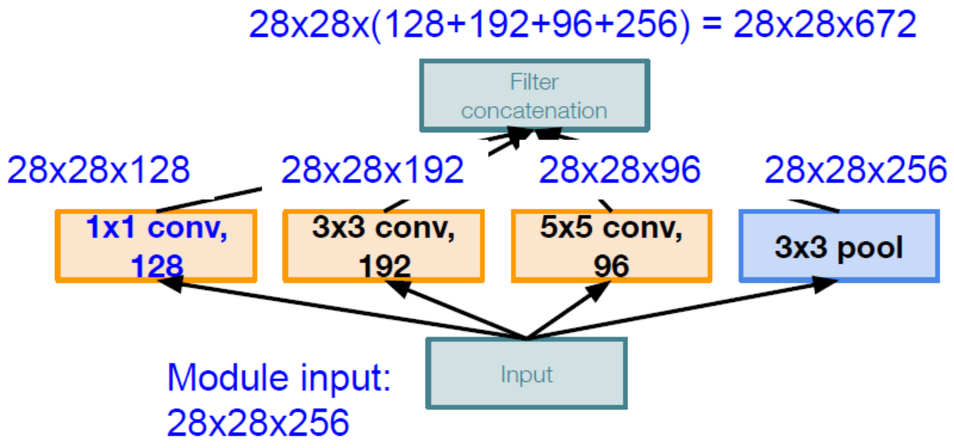
Conv Ops:
[1x1 conv, 128]
[3x3 conv, 192]
[5x5 conv, 96]
Total:

Conv Ops:
[1x1 conv, 64]
[1x1 conv, 64]
[1x1 conv, 128]
[3x3 conv, 192]
[5x5 conv, 96]
[1x1 conv, 64]
Total:

Szegedy et al., 2014

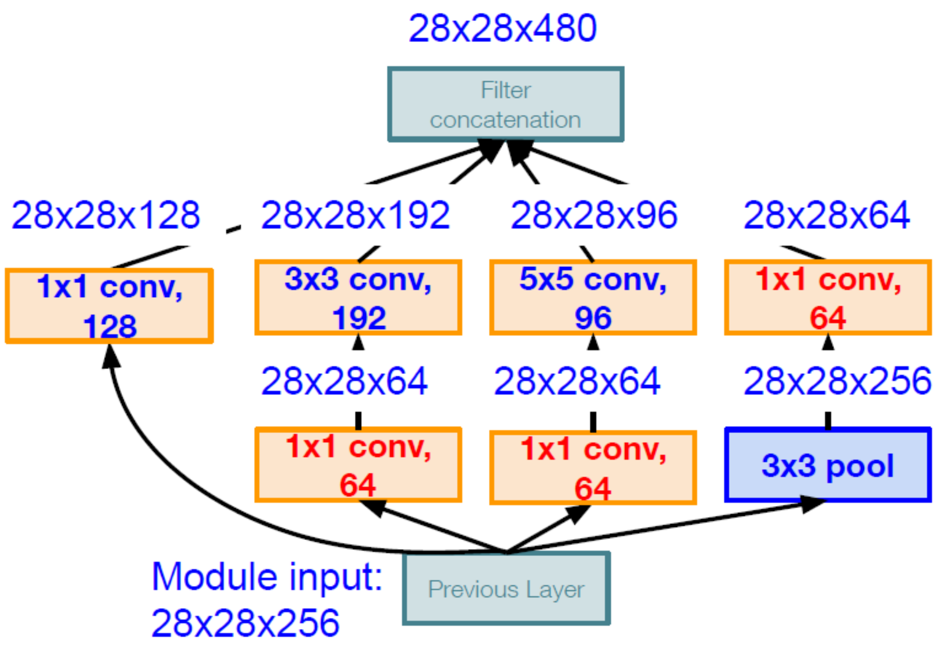
GoogLeNet

더 빨리 계산 가능.



Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: 854M ops

Very expensive compute



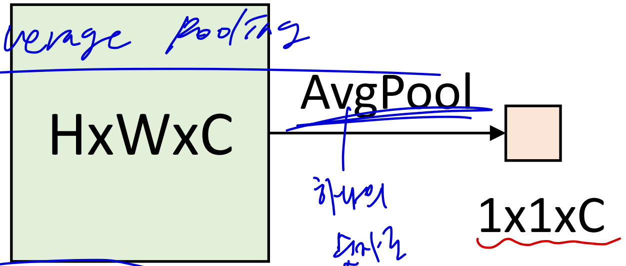
Conv Ops:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops

Szegedy et al., 2014

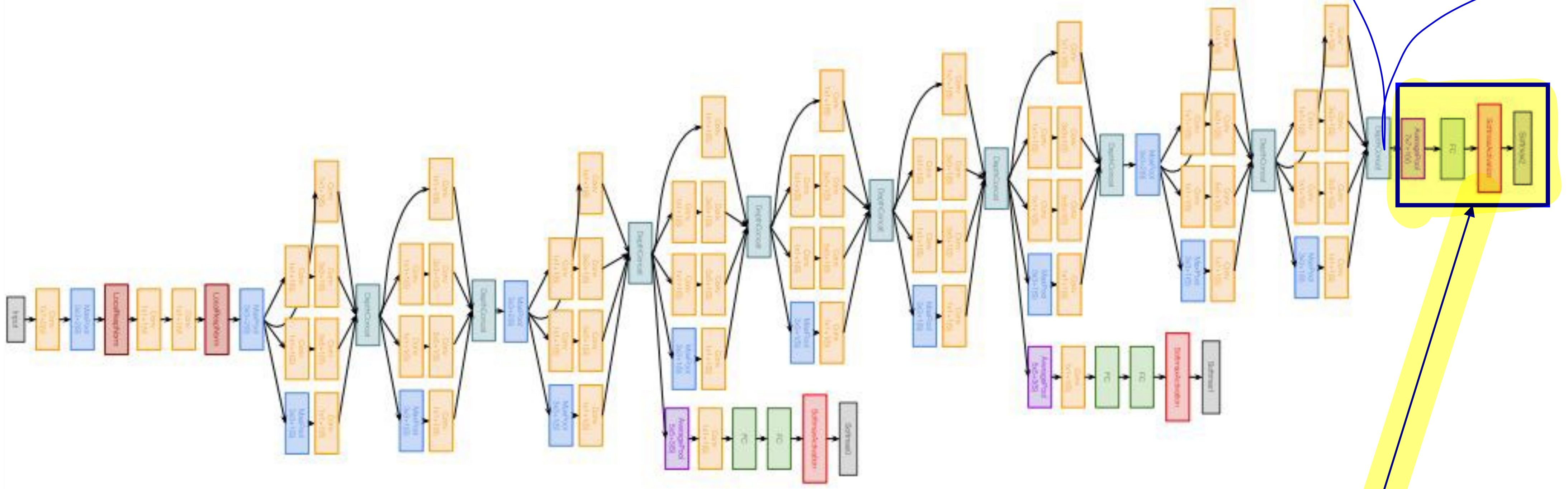
GoogLeNet

global average pooling

큰, 작은 입력 img 전부 가능.



하나의 숫자로 통일

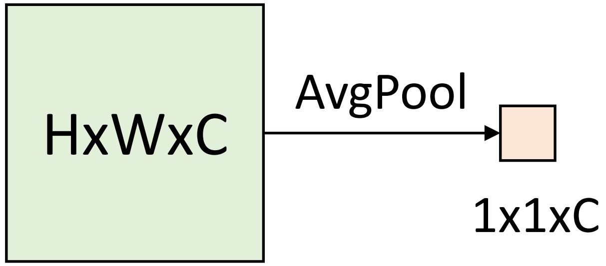


Note: after the last convolutional layer, a global average pooling layer is used that spatially averages across each feature map, before final FC layer. No longer multiple expensive FC layers!

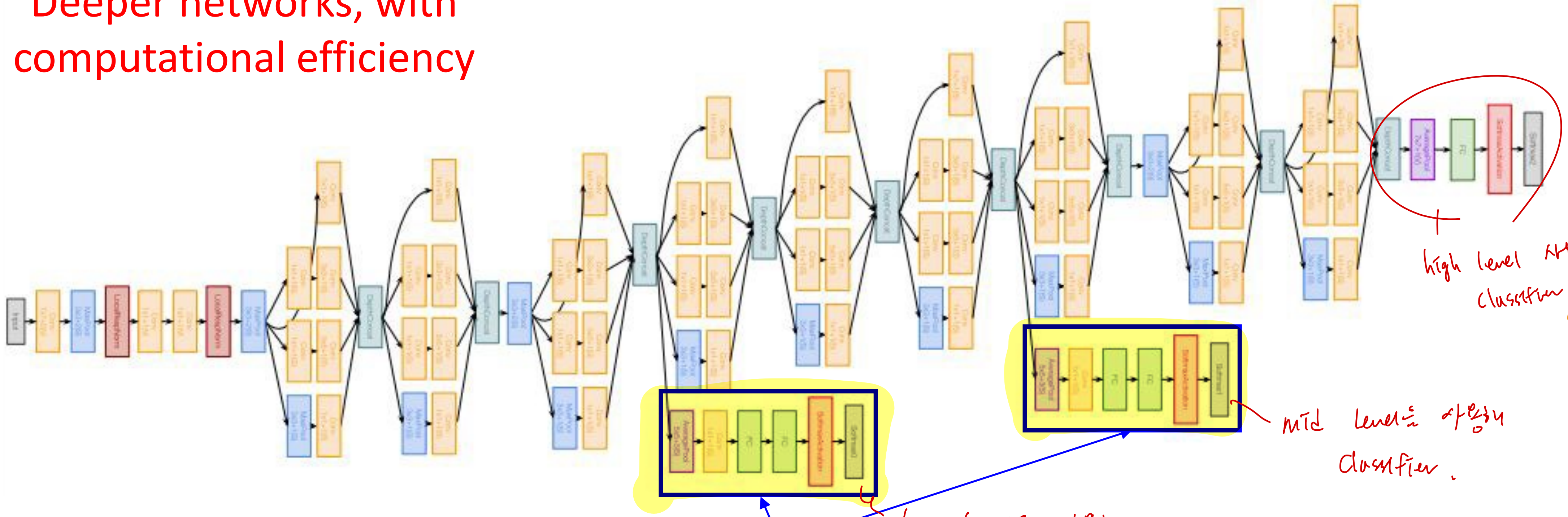
Classifier output

Szegedy et al., 2014

GoogLeNet



Deeper networks, with computational efficiency



high level 사용해 classifier.

mid level을 사용해 classifier.

low level을 사용해 classifier

당연히 이걸 사용한게 제일 성능 good.

Auxiliary classification outputs to inject additional gradient at lower layers
(AvgPool-1x1Conv-FC-FC-Softmax)

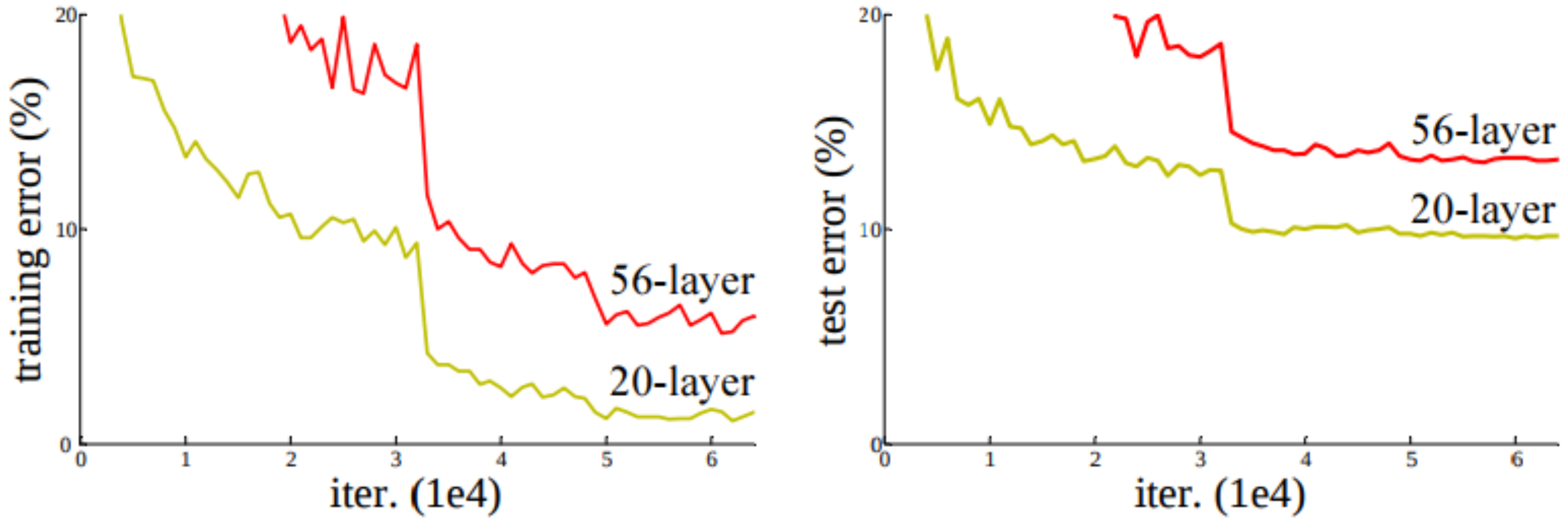
알 수가 없는 gradient를 계산하다보니, gradient를 곱하다 보면 사라지는 문제 O.

그래서 앞단에서도 gradient를 더해서 학습이 좋게.

Szegedy et al., 2014

ResNet

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



56-layer model performs worse on both test and training error

-> The deeper model performs worse, but it’s not caused by overfitting!

He et al., 2015

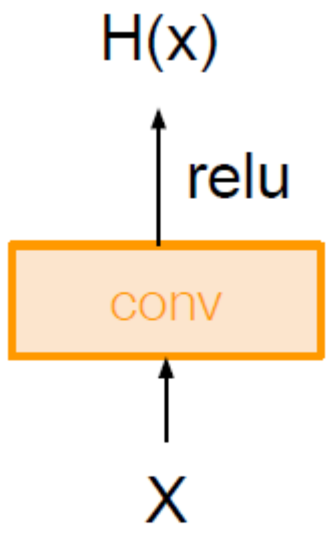
ResNet

Fact: Deep models have more representation power (more parameters) than shallower models.

Hypothesis: the problem is an optimization problem, deeper models are harder to optimize

What should the deeper model learn to be at least as good as the shallower model?

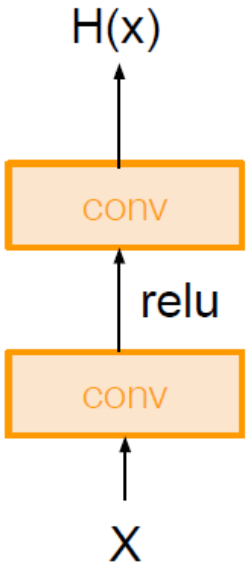
A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.



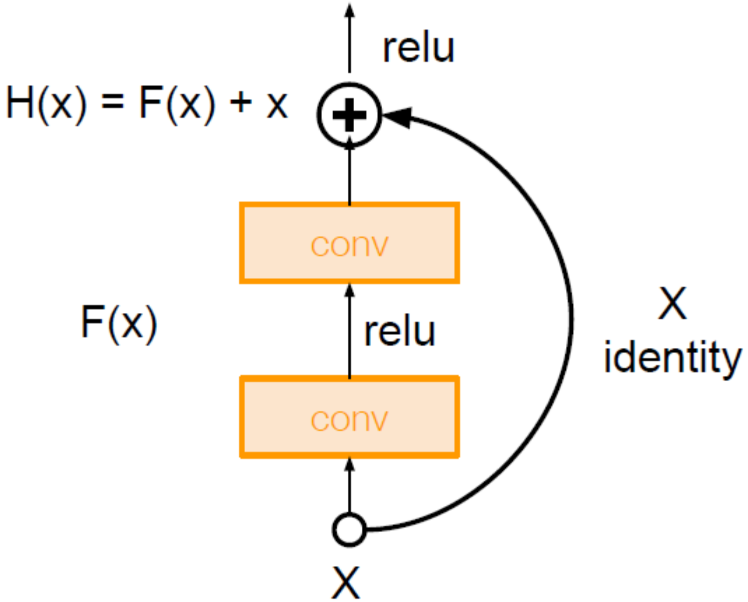
He et al., 2015

ResNet

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



"Plain" layers



Residual block

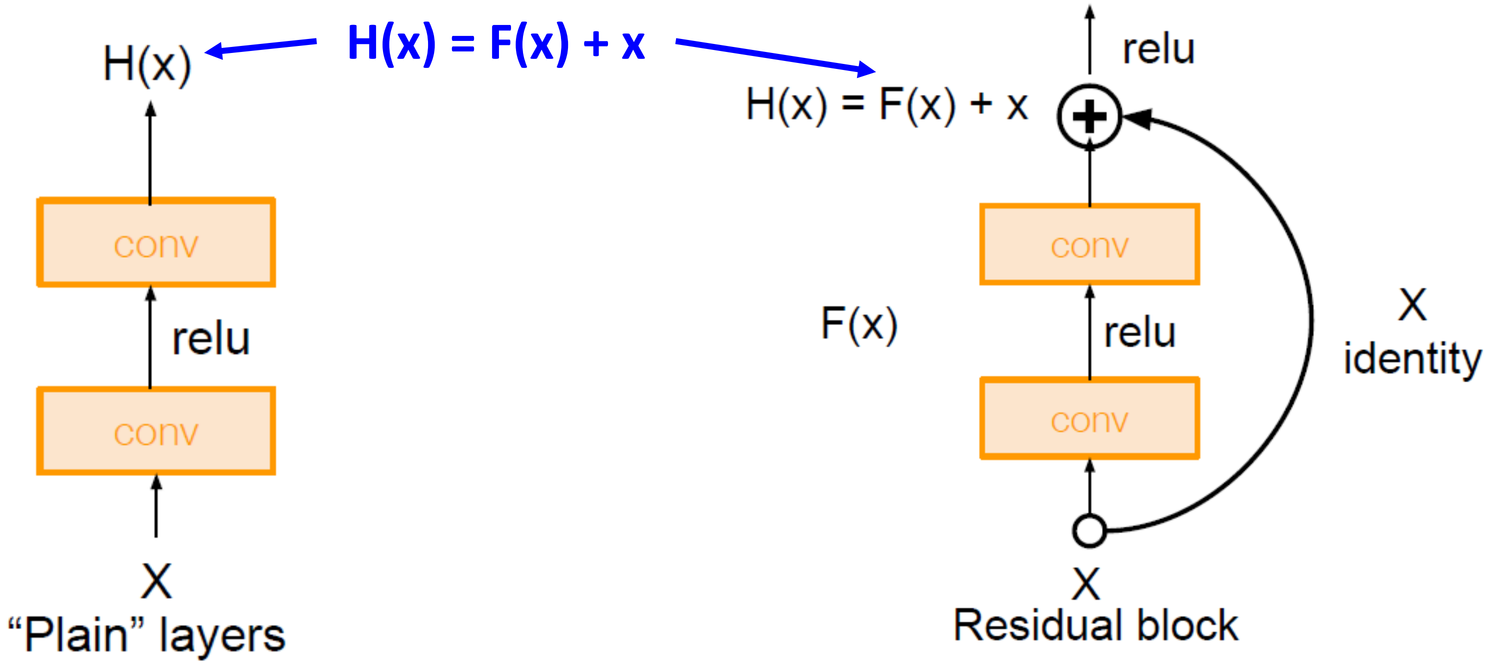
Identity mapping:
H(x) = x if F(x) = 0

차이 : 어떤걸 학습하는지

He et al., 2015

ResNet

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Identity mapping:
H(x) = x if F(x) = 0

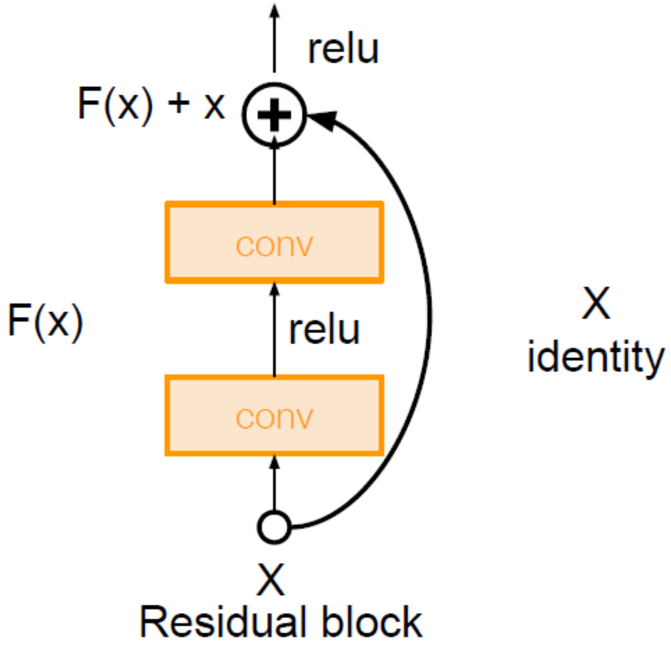
Use layers to
fit **residual**
F(x) = H(x) - x
instead of
H(x) directly

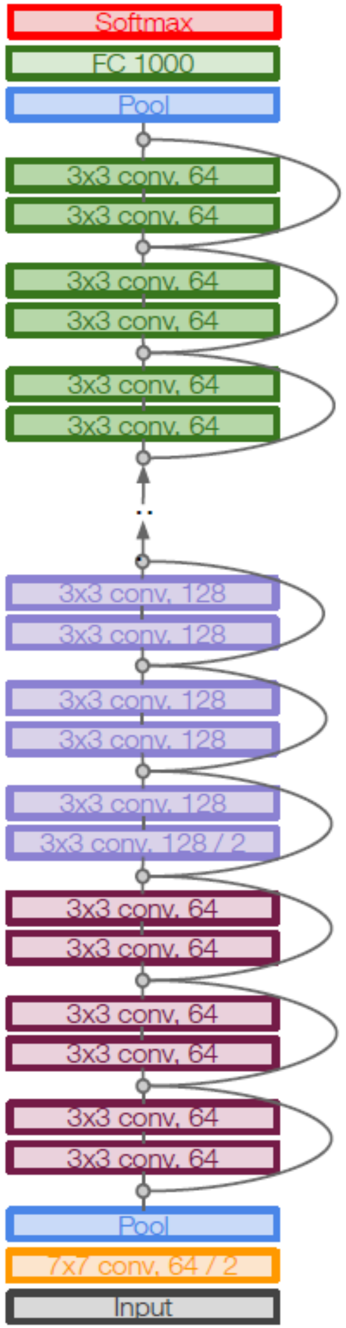
He et al., 2015

ResNet

Very deep networks using residual Connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!





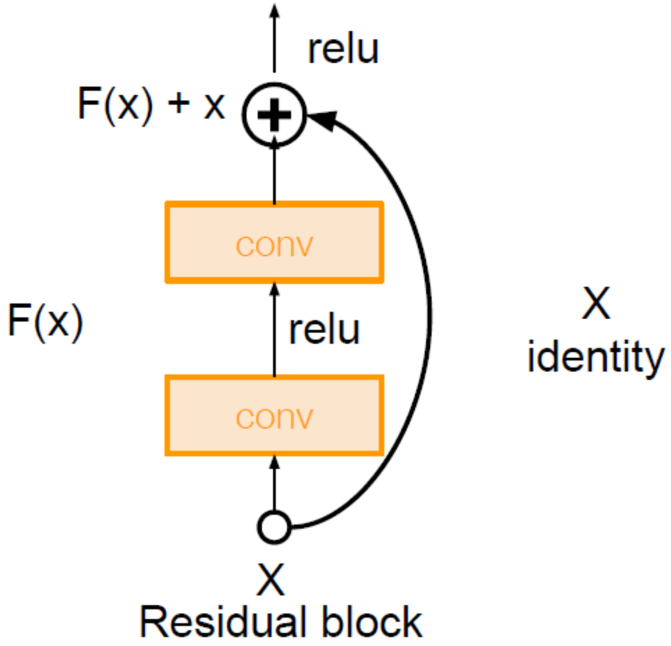
He et al., 2015

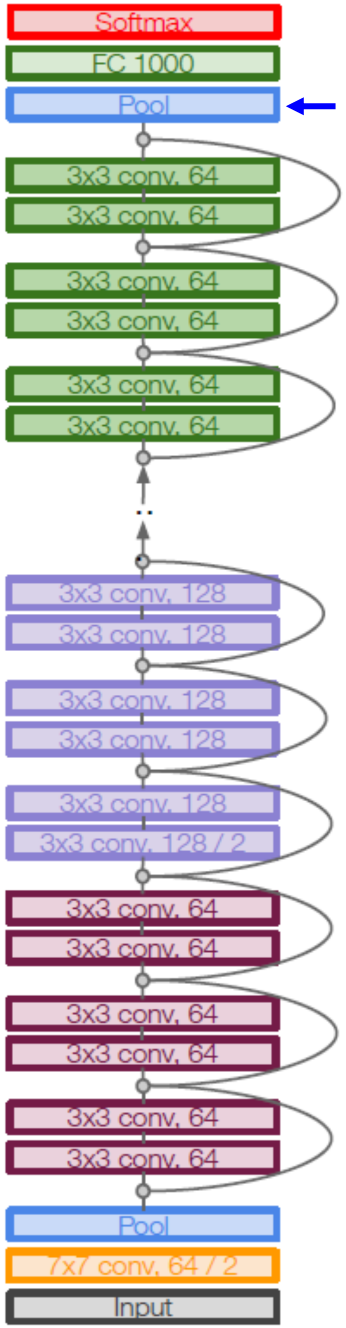
ResNet

Conv layer : 3 × 3

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2
- (/2 in each dimension)
- Additional conv layer at the beginning (stem)
- No FC layers at the end (only FC 1000 to output classes)
- (In theory, you can train a ResNet with input image of variable sizes)





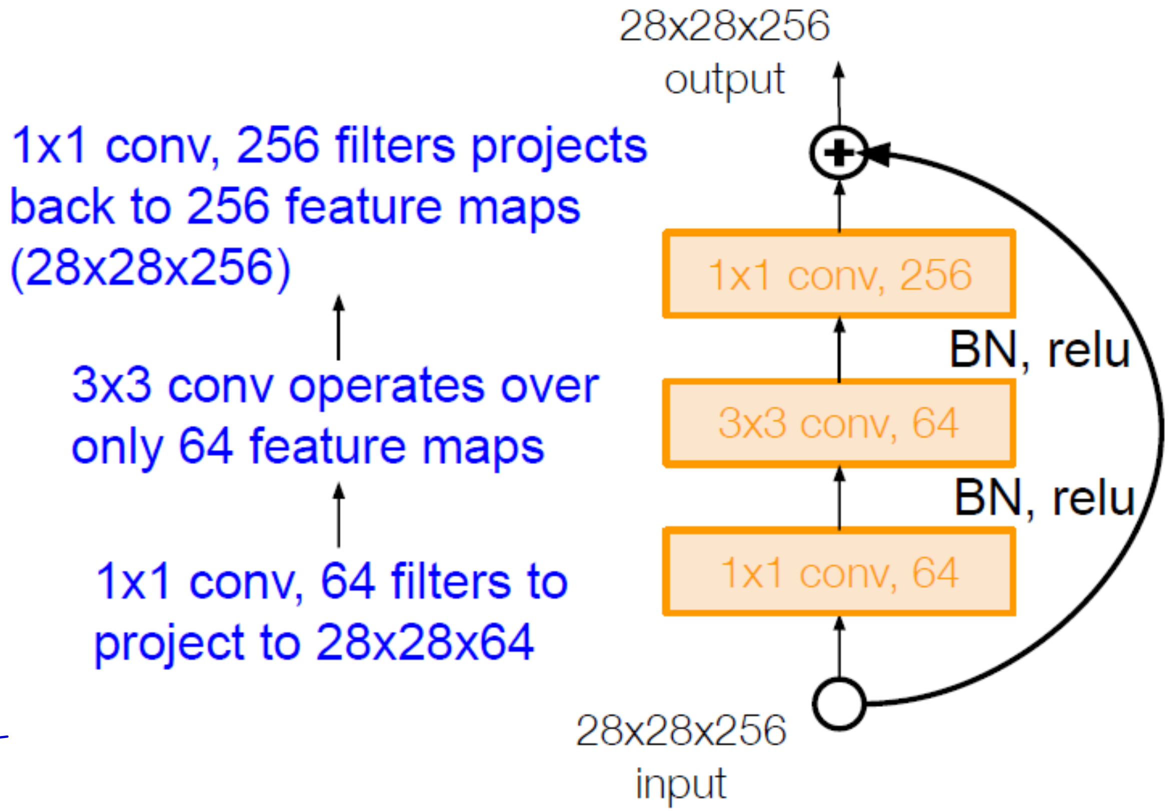
No FC layers besides FC 1000 to output classes

Global average pooling layer after last conv layer

He et al., 2015

ResNet

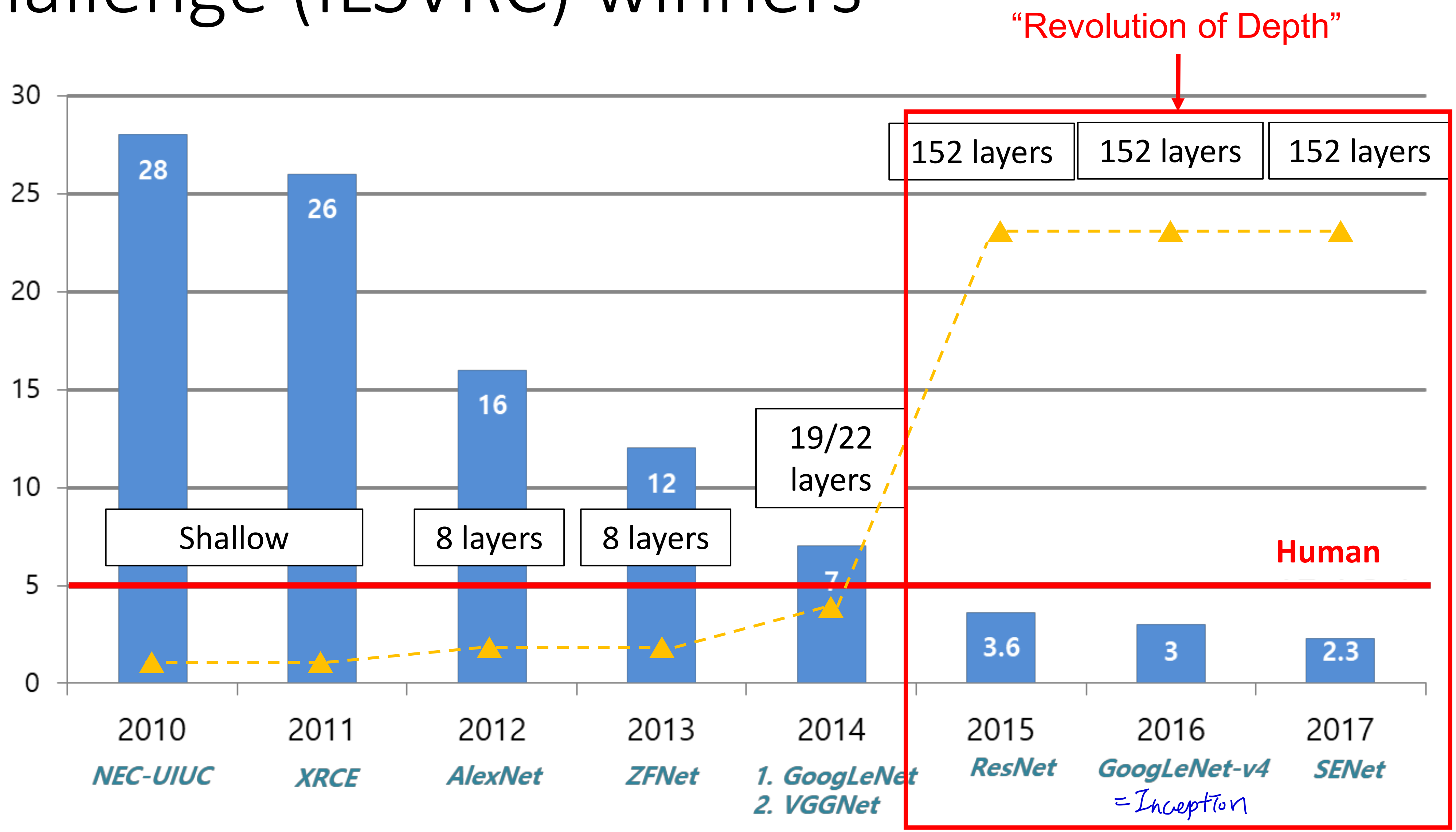
For deeper networks (ResNet-50+), use "bottleneck" layer to improve efficiency (similar to GoogLeNet)



계산량이
몇 배 줄어드냐는거 ?

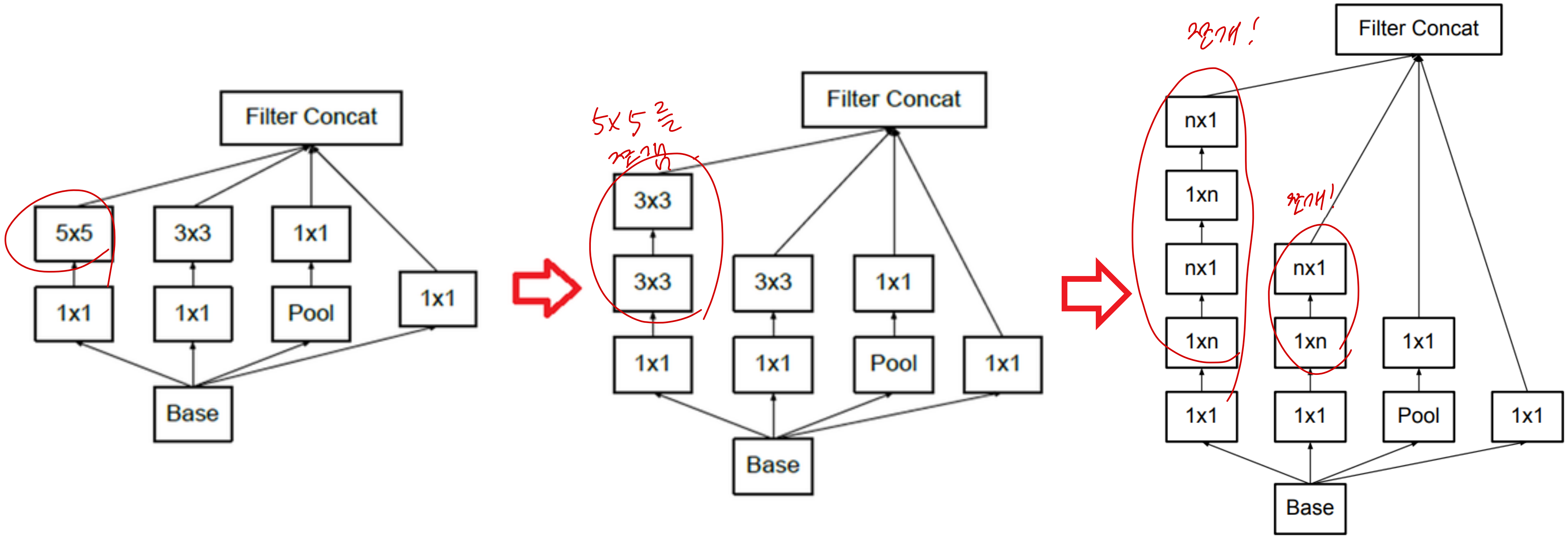
He et al., 2015

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



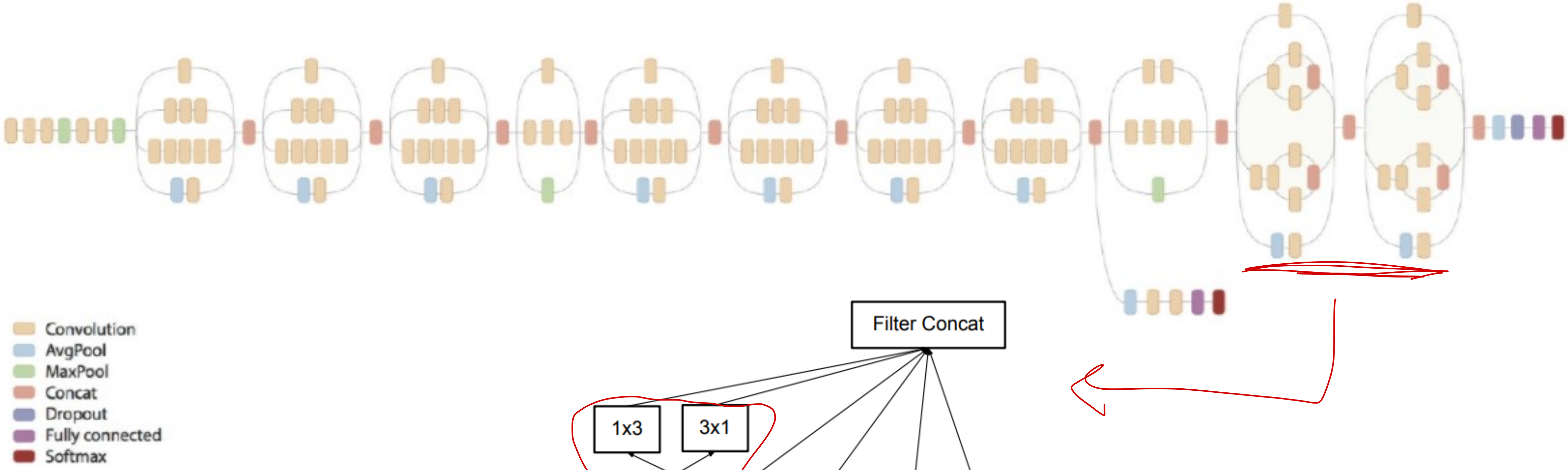
googlenet '발전 ver.

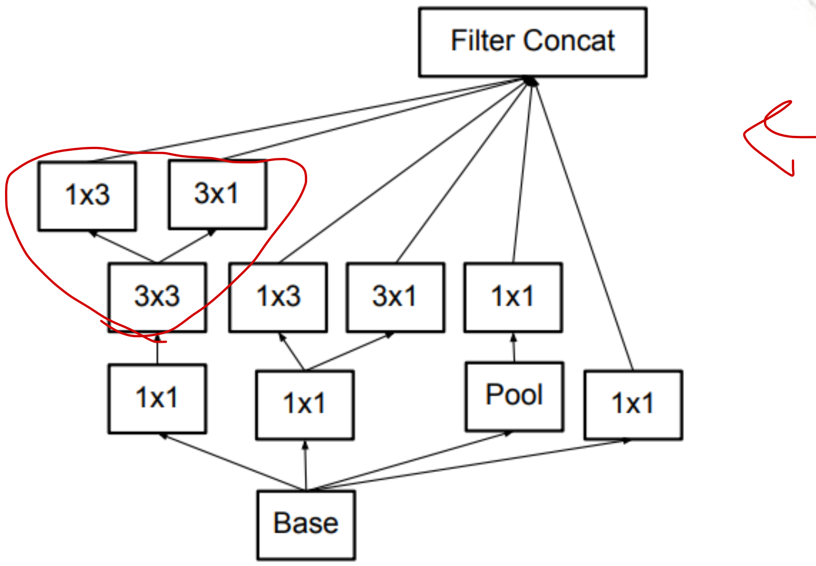
Inception v2



Reduce the model size and computational cost

Inception v2





Inception v3

오로지 학습 parameter 학습량? 만 줄임.

- Optimizer
  - RMSProp
- Label Smoothing → 하나의 정답인 레이블을 0 0 0 0 1 이런 식이 아니고, (벡터로 만듦?) , 0.8%는 치타다, 0.2% [illegible] 를 닮음. 이런식으로
- Add BatchNorm after last Fully Connected layer

Inception v4

단계를 나눠서
(A, B, C) 진행.

모듈을 만들어서
사용.

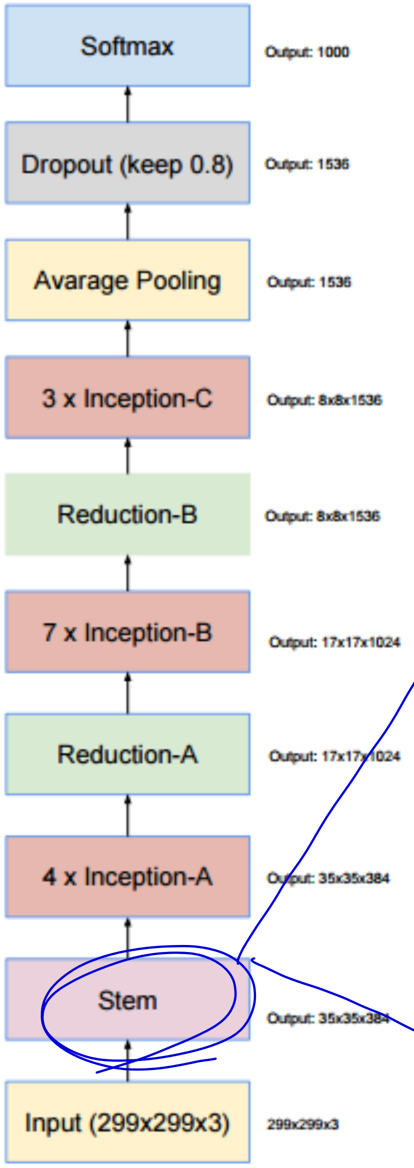


Figure 9. The overall schema of the Inception-v4 network. For the detailed modules, please refer to Figures 3, 4, 5, 6, 7 and 8 for the detailed structure of the various components.

차원을 줄임

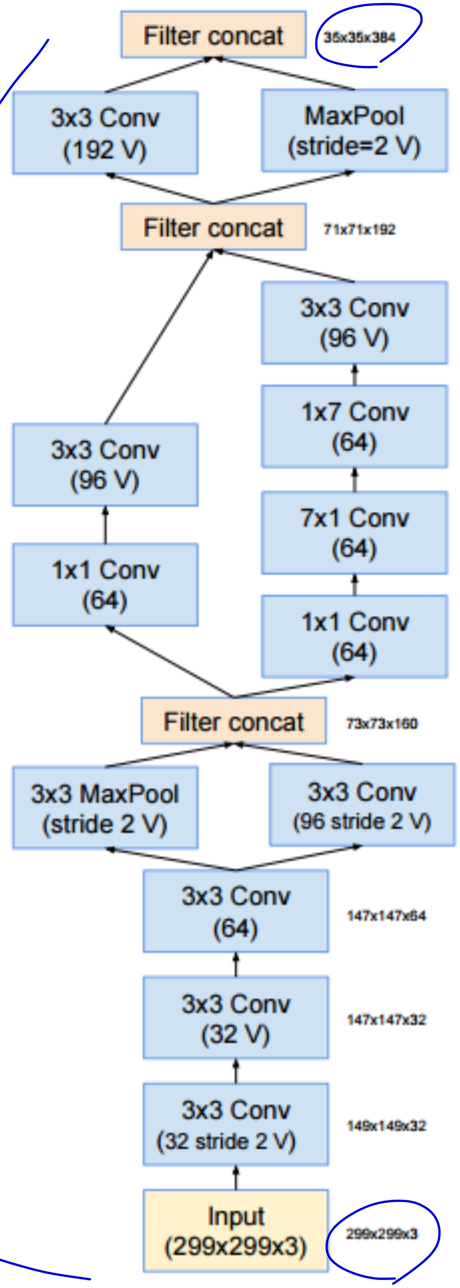


Figure 3. The schema for stem of the pure Inception-v4 and Inception-ResNet-v2 networks. This is the input part of those networks. Cf. Figures 9 and 15

Inception v4

Inception-A

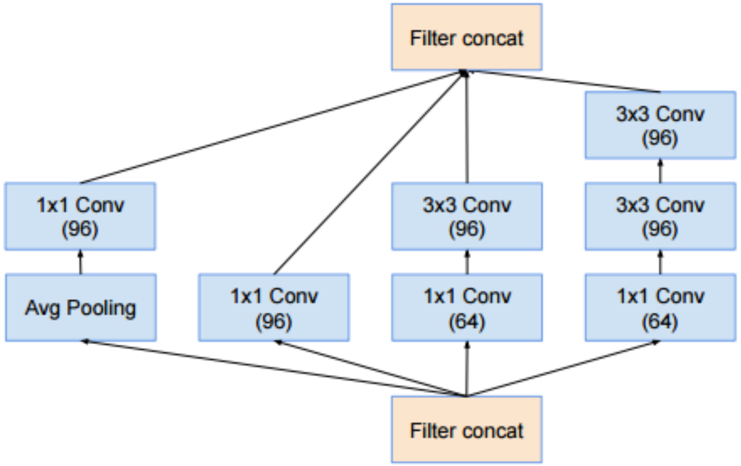


Figure 4. The schema for 35 × 35 grid modules of the pure Inception-v4 network. This is the Inception-A block of Figure 9.

low level

Inception-B

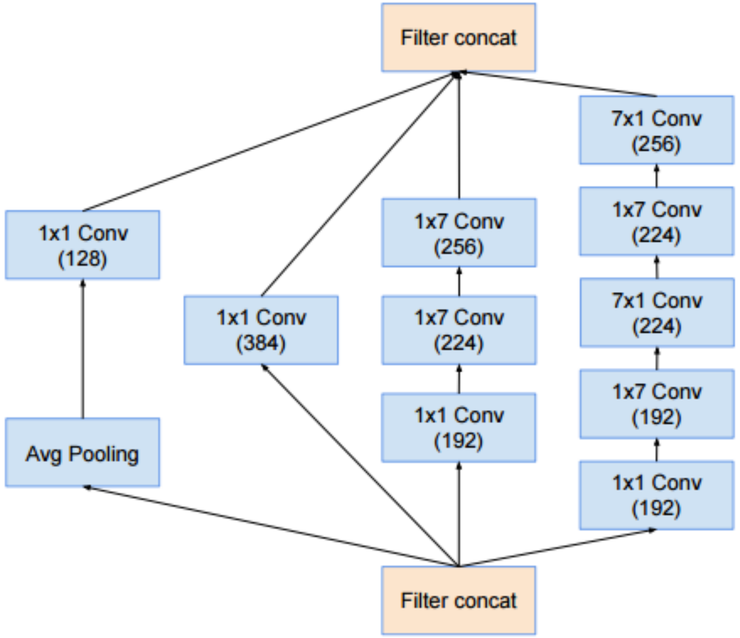


Figure 5. The schema for 17 × 17 grid modules of the pure Inception-v4 network. This is the Inception-B block of Figure 9.

mid level

Inception-C

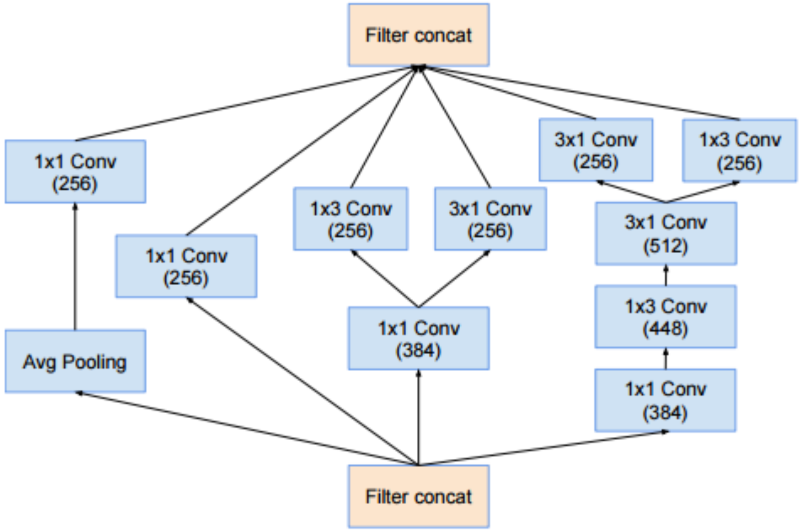


Figure 6. The schema for 8 × 8 grid modules of the pure Inception-v4 network. This is the Inception-C block of Figure 9.

high level

Inception v4

Pooling

Reduction-A

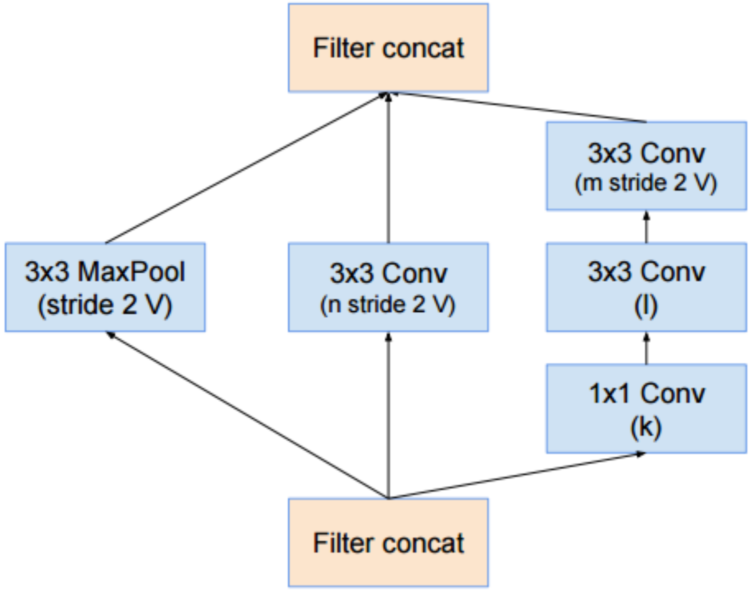


Figure 7. The schema for 35×35 to 17×17 reduction module. Different variants of this blocks (with various number of filters) are used in Figure 9, and 15 in each of the new Inception(-v4, -ResNet-v1, -ResNet-v2) variants presented in this paper. The k, l, m, n numbers represent filter bank sizes which can be looked up in Table 1.

Reduction-B

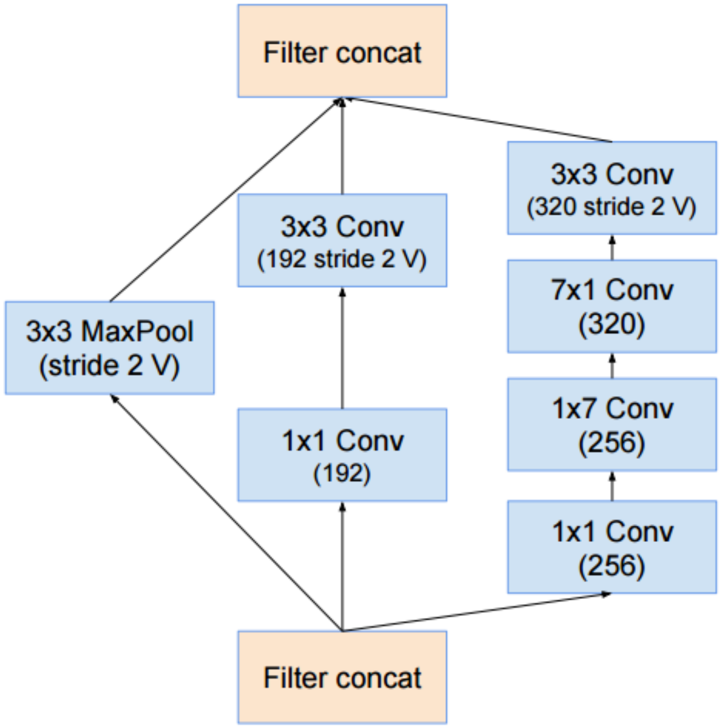


Figure 8. The schema for 17×17 to 8×8 grid-reduction module. This is the reduction module used by the pure Inception-v4 network in Figure 9.

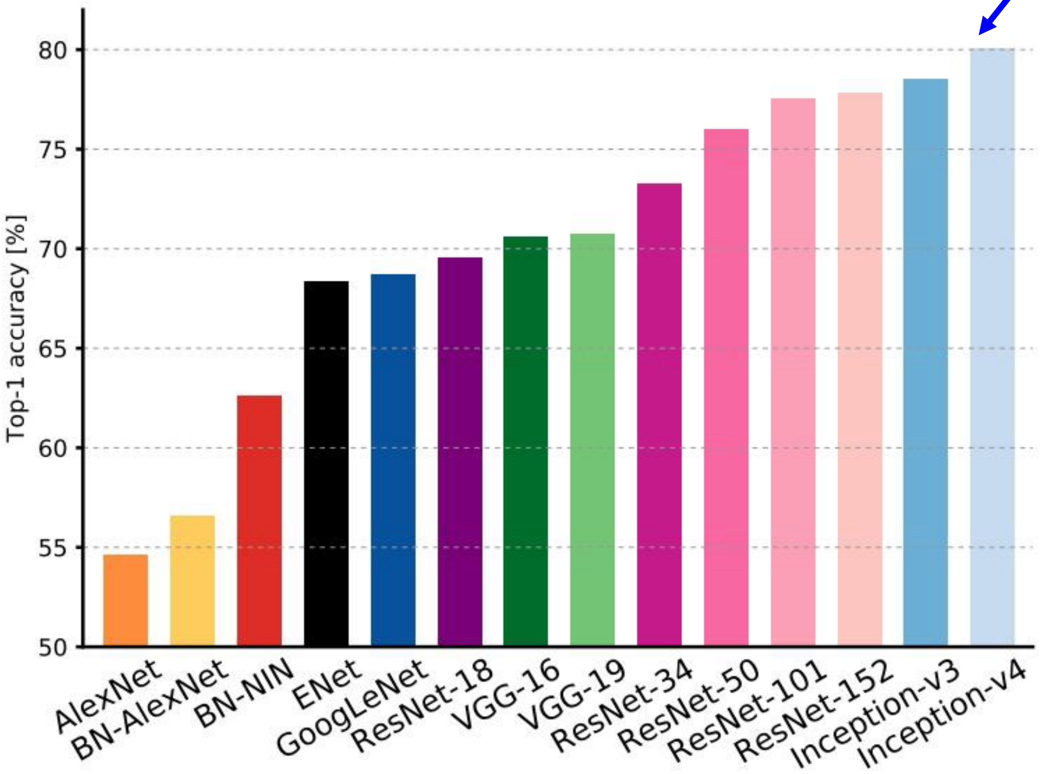
Comparing complexity

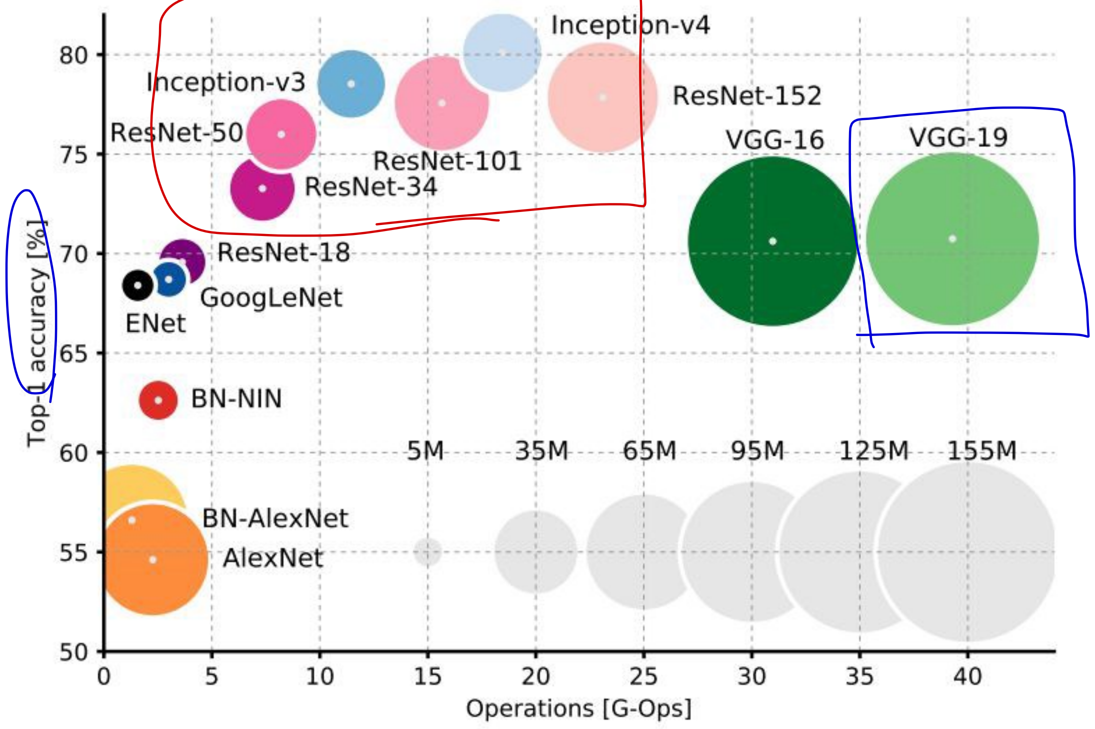
싱글에서는 ResNet50
15 epoch → 10분 소요.

챌린지 제한 시간에
다 못 돌릴 수 O.

Inception-v4: Resnet + Inception!

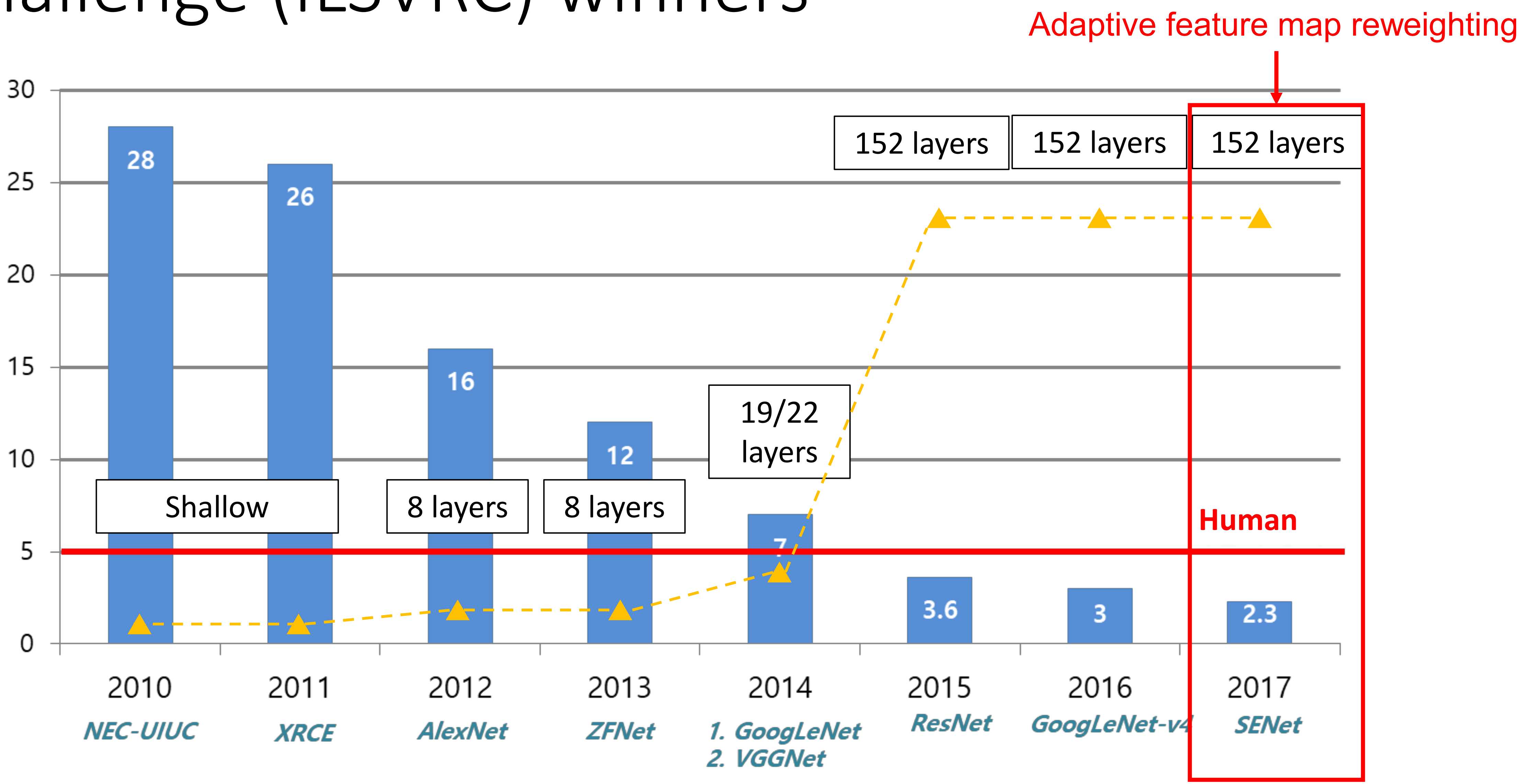
빠르고, 정확. 가성비갑.





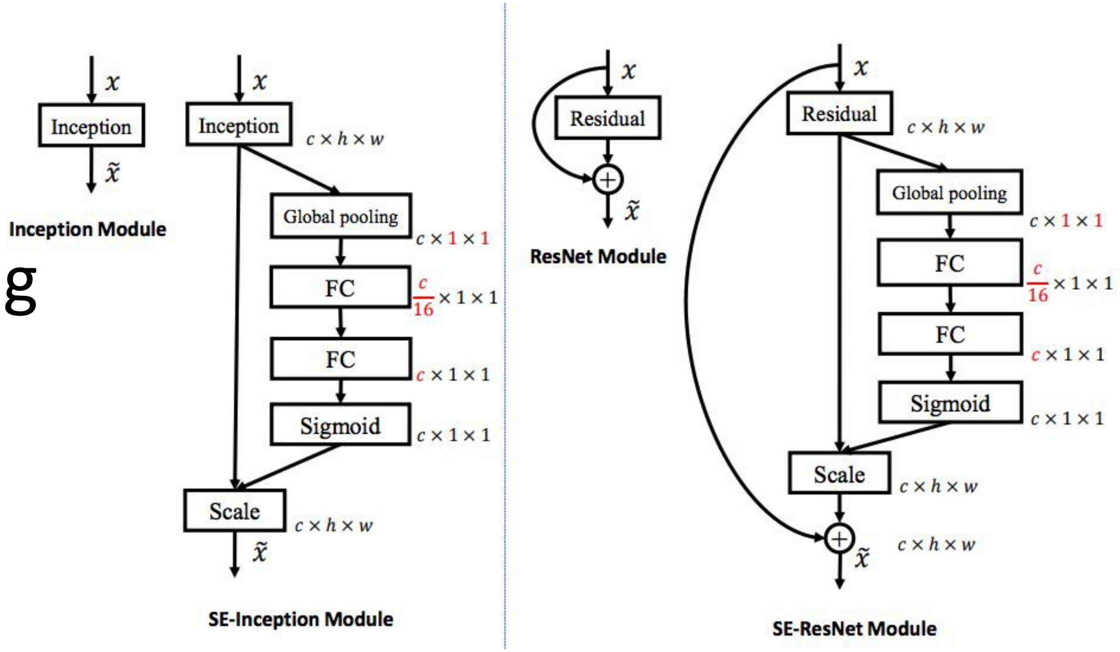
Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

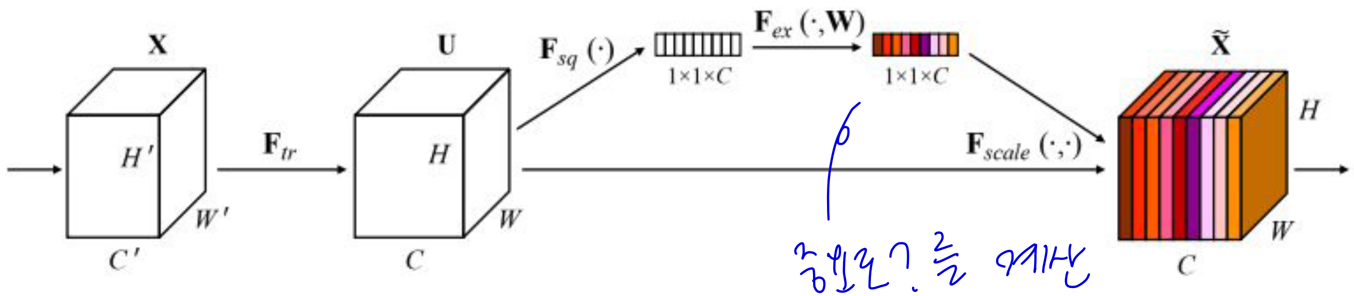


Squeeze-and-Excitation Networks (SENet)

- Add a “feature recalibration” module thatlearns to adaptively reweight feature maps
- Global information (global avg. pooling layer) + 2 FC layers used to determine feature map weights
- ILSVRC’17 classification winner (using ResNeXt-152 as a base architecture)



공간에 모듈을 제한?



중요도를 계산

Hu et al. 2017

Efficient networks

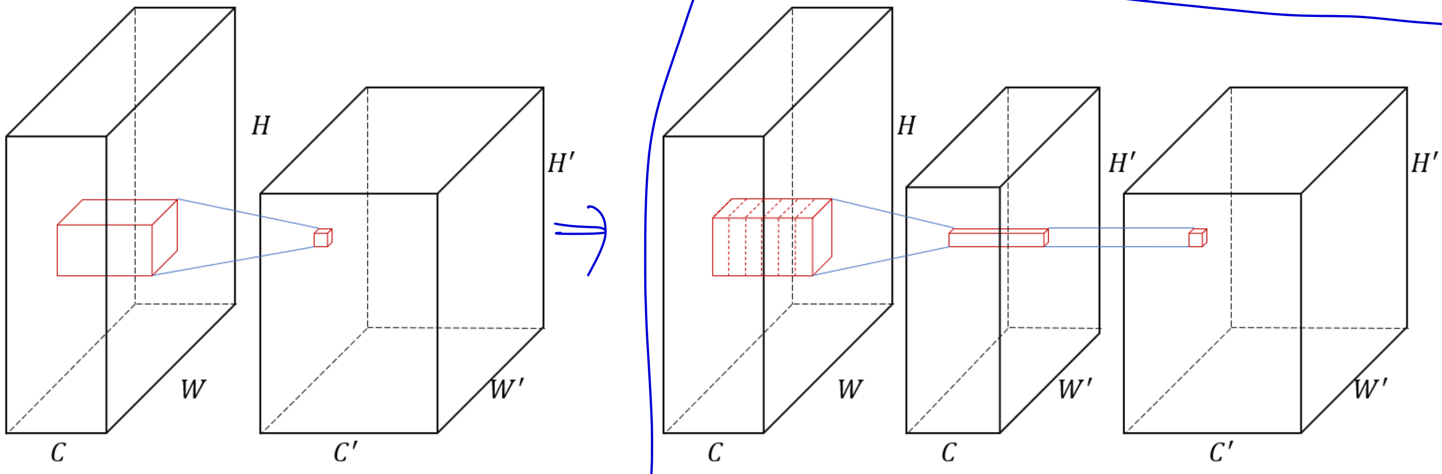
모델 size를 성능↑.

차이점 다시 듣기

- MobileNets

group conv를 잘 사용.

  - Depthwise separable convolutions replace standard convolutions by factorizing them into a depthwise convolution and a 1x1 convolution
  - Much more efficient, with little loss in accuracy



1x1으로 서로의 group을 연결

C개의 그룹 Conv 진행.

$k \times k$ Convolution

#Parameters: $C \times k \times k \times C'$

$k \times k$ Depthwise Convolution → 1x1 Pointwise Convolution

#Parameters: $C \times (k \times k + C')$

Howard et al. 2017

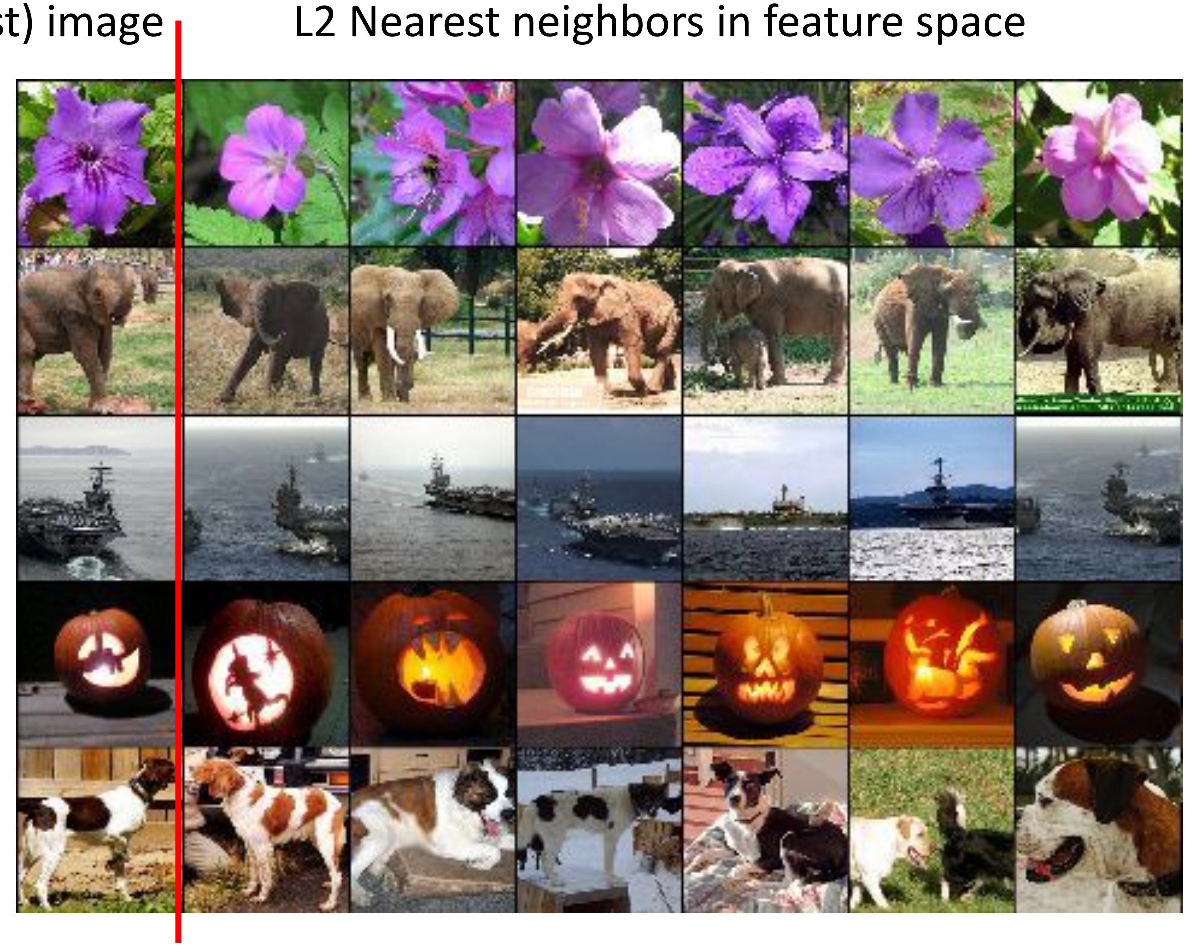
Main takeaways

AlexNet showed that you can use CNNs to train Computer Vision models.
ZFNet, **VGG** shows that bigger networks work better
GoogLeNet is one of the first to focus on efficiency using 1x1 bottleneck convolutions and global avg pool instead of FC layers
ResNet showed us how to train extremely deep networks
- Limited only by GPU & memory!
- Showed diminishing returns as networks got bigger

After ResNet: CNNs were better than the human metric and focus shifted to Efficient networks:
- Lots of tiny networks aimed at mobile devices: **MobileNet**

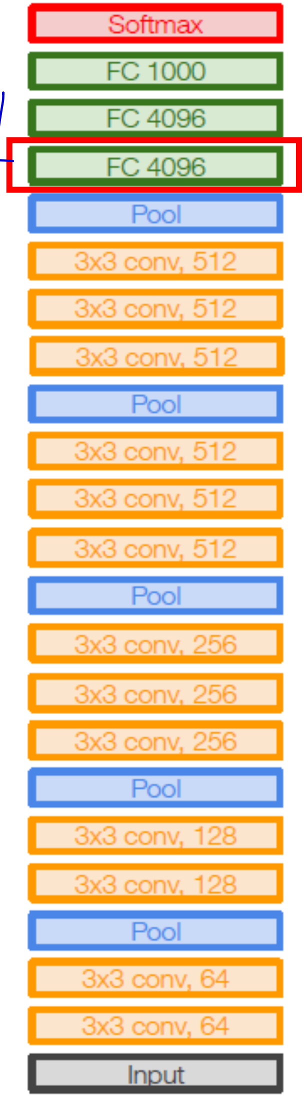
Transfer Learning with CNNs

Query(Test) image | L2 Nearest neighbors in feature space



다 학습해도 되고.
앞은 고정하고,
뒤만 학습해봤다.
이래도 가능.

10개 class정리



VGG16