



Chapter

3

CNN Architectures

- 1. CNN Architectures**
- 2. Case Studies**
 - AlexNet
 - VGG
 - GoogLeNet
 - ResNet
- 3. Keggle**

강의에 앞서서..

❖ 본 문서는 아래의 자료들을 활용하여 만들어 졌음을 알립니다

❖ 모두를 위한 딥러닝 강좌

- 네이버 Search & Clova AI 부분 리더 김성훈 교수님
- https://www.youtube.com/playlist?list=PLIMkM4tgfjnLSOjrEJN31gZATbcj_MpUm
- <https://www.edwith.org/boostcourse-dl-tensorflow/lecture/43739/>

❖ 스탠포드 대학 CNN 강좌

- Fei-Fei Li & Andrej Karpathy & Justin Johnson
- <http://cs231n.stanford.edu/slides/2020/>

CS231n: Convolutional Neural Networks for Visual Recognition

- This course, Prof. Fei-Fei Li & Justin Johnson & Serena Yeung
- Focusing on applications of deep learning to computer vision

강의에 앞서서..

❖ 밑바닥부터 시작하는 딥러닝

- <https://github.com/ExcelsiorCJH/DLFromScratch>

❖ DeepLearning Getting Started with TensorFlow

- <https://github.com/Jpub/TensorflowDeeplearning>

❖ TensorFlow Image Classification

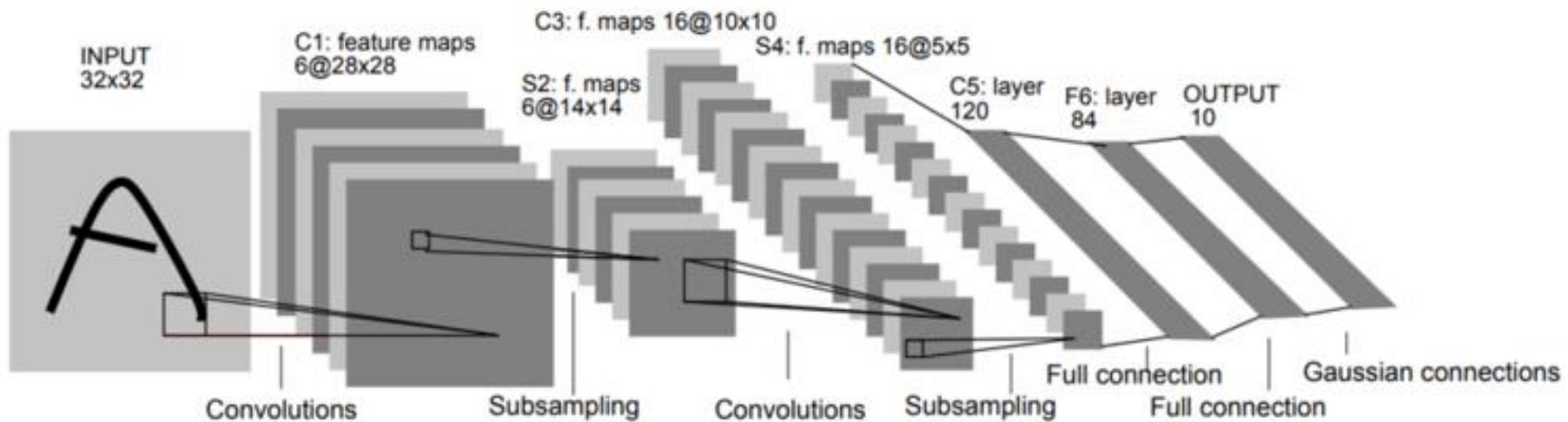
- <https://www.tensorflow.org/tutorials/images/classification>

❖ Hands-On Machine Learning

- <https://github.com/ExcelsiorCJH/Hands-On-ML>

Review: LeNet-5

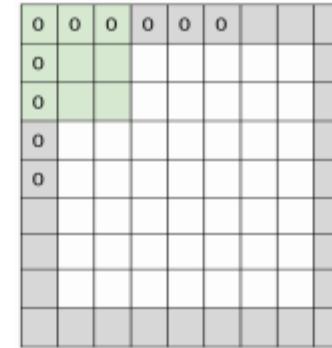
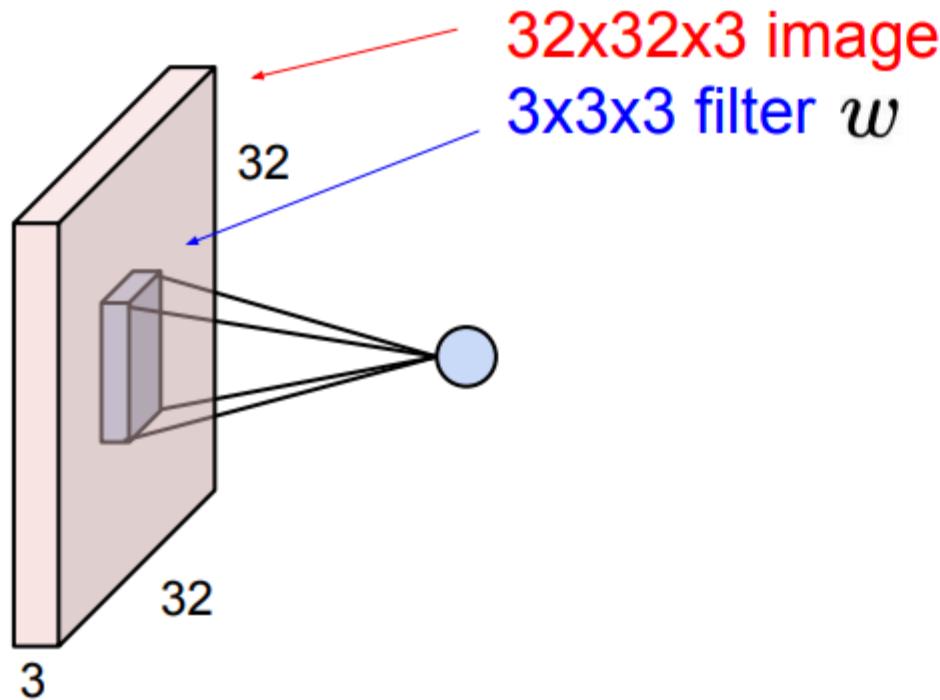
[LeCun et al., 1998]



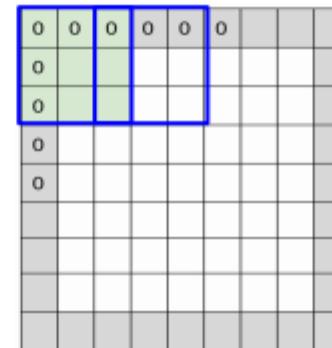
Conv filters were 5×5 , applied at stride 1

Subsampling (Pooling) layers were 2×2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

Review: Convolution

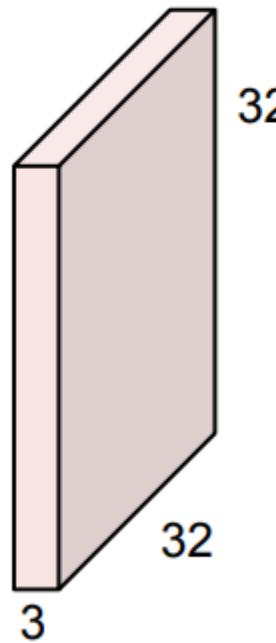


Padding:
Preserve
input spatial
dimensions in
output activations



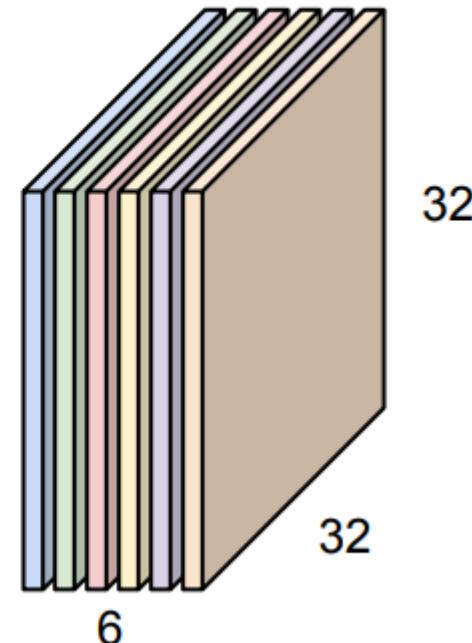
Stride:
Downsample
output activations

Review: Convolution



Convolution Layer

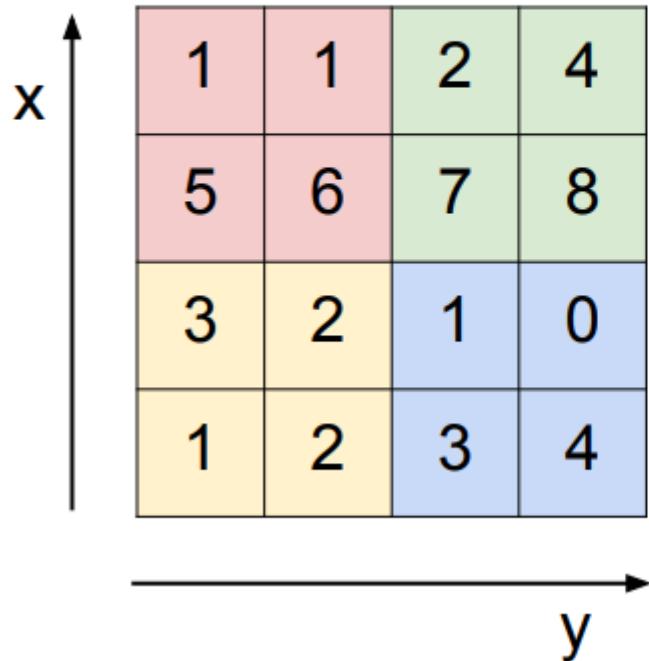
activation maps



Each conv filter outputs a “slice” in the activation

Review: Pooling

Single depth slice



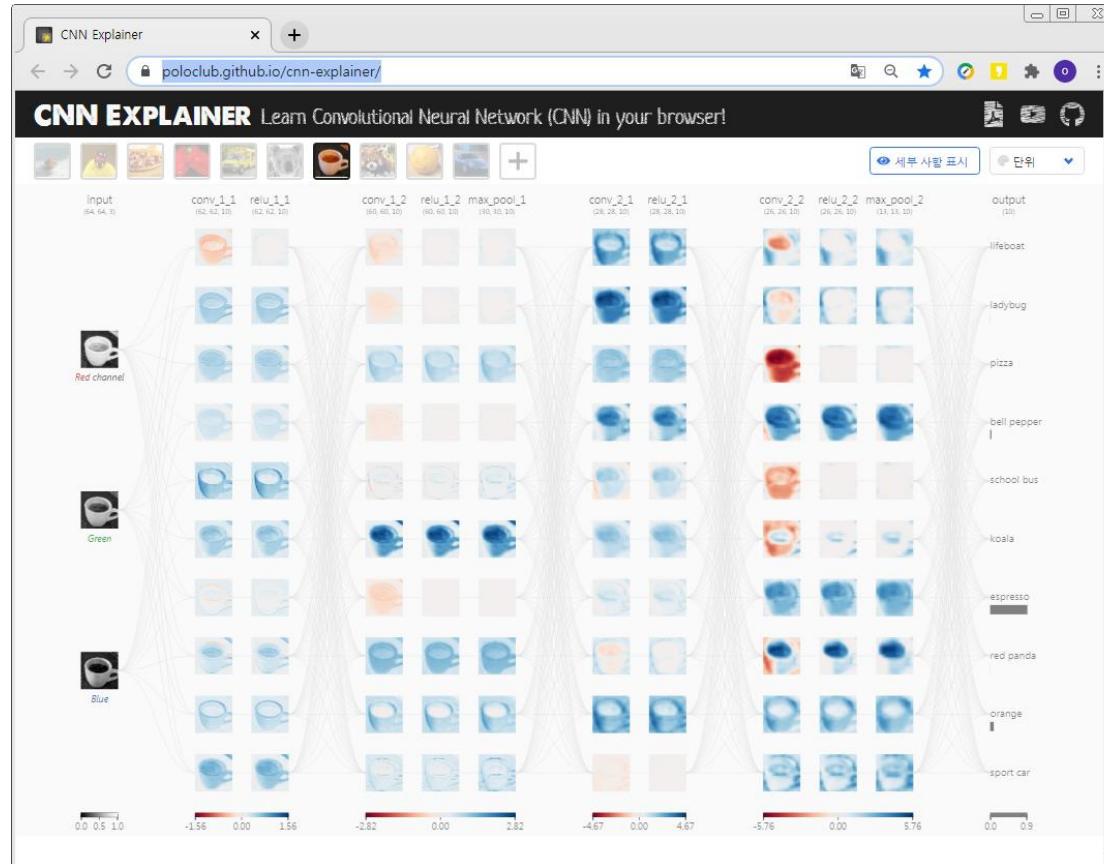
max pool with 2x2 filters
and stride 2

A 2x2 grid representing the output of max pooling. The values are 6 (top-left), 8 (top-right), 3 (bottom-left), and 4 (bottom-right). The grid is color-coded: pink for 6, light green for 8, yellow for 3, and light blue for 4.

6	8
3	4

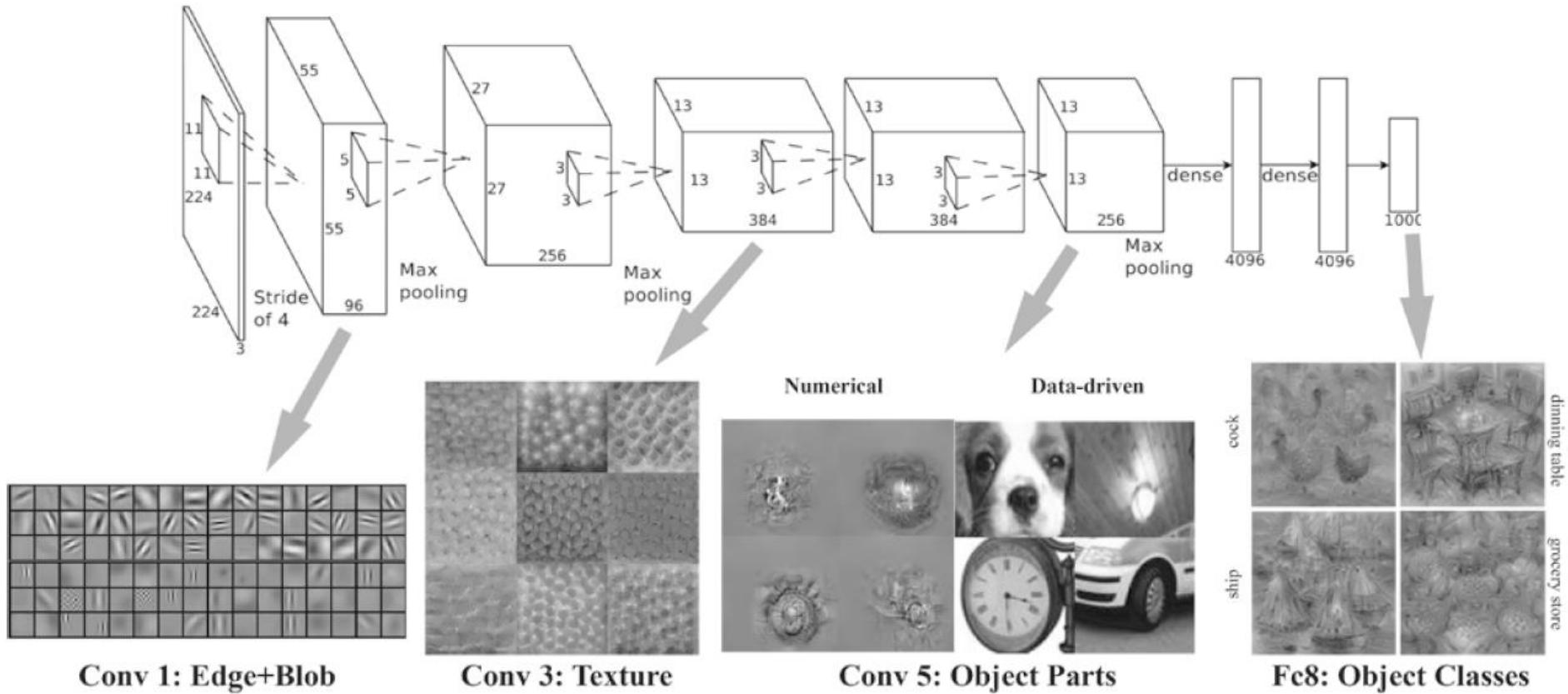
CNN Explainer

❖ CNN 이해를 위한 사이트



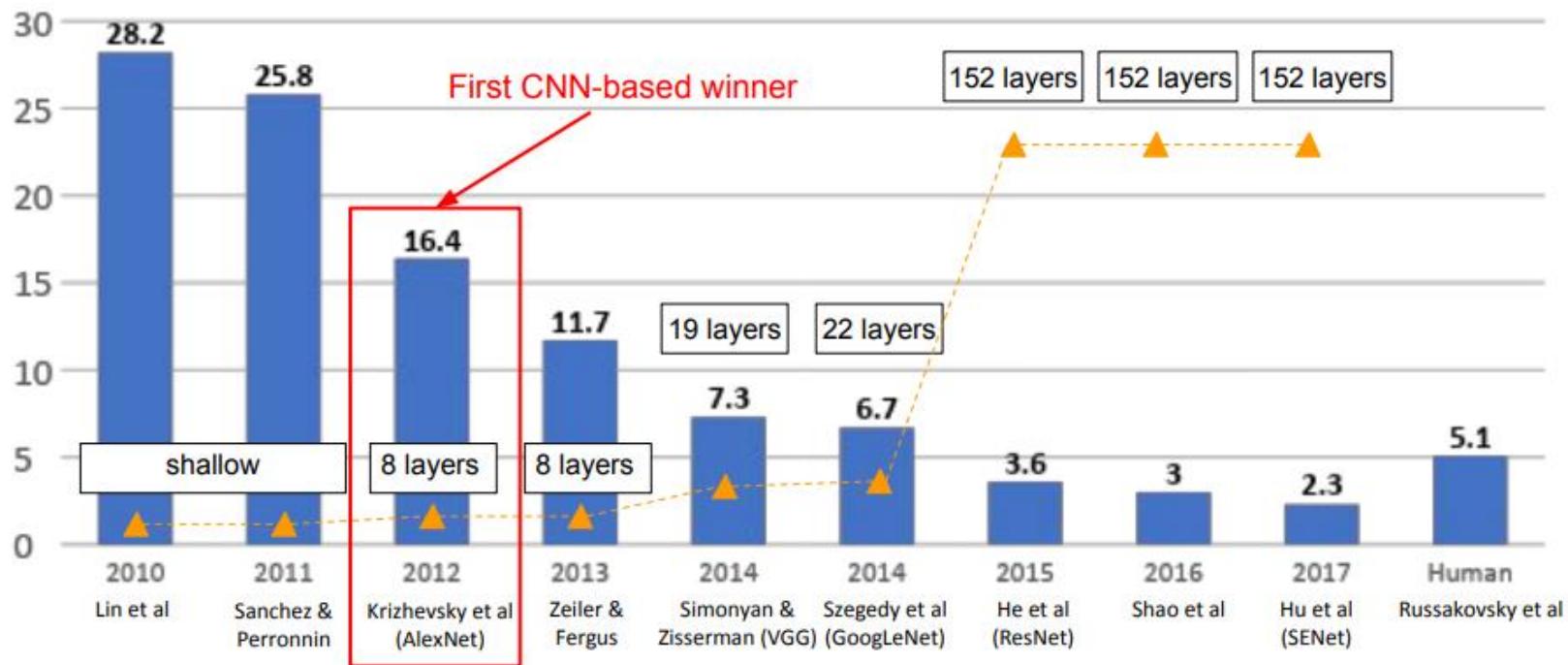
<https://poloclub.github.io/cnn-explainer/>

CNN Architectures



CNN Architectures

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Case Study: AlexNet

[Krizhevsky et al. 2012]

Architecture:

CONV1

MAX POOL1

NORM1

CONV2

MAX POOL2

NORM2

CONV3

CONV4

CONV5

Max POOL3

FC6

FC7

FC8

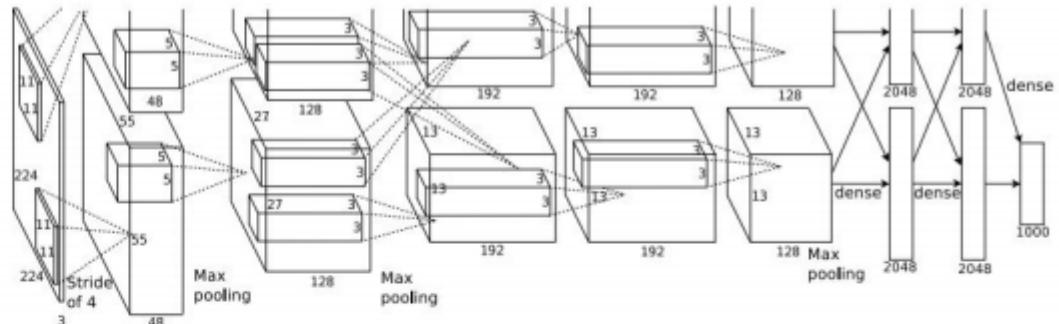
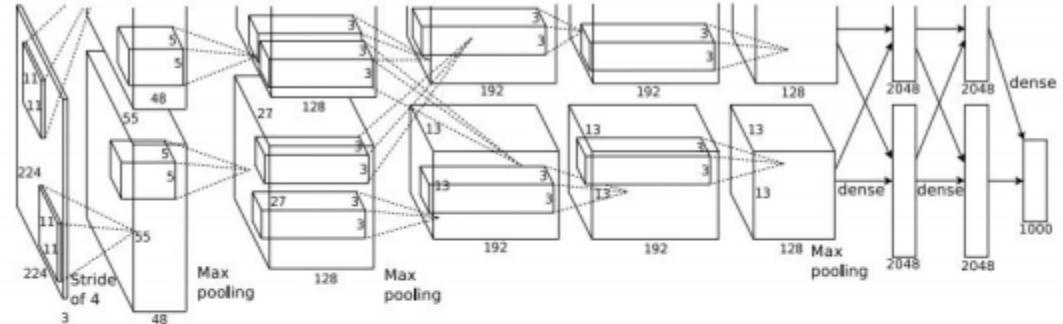


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

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

=>

Q: what is the output volume size? Hint: $(227-11)/4+1 = 55$

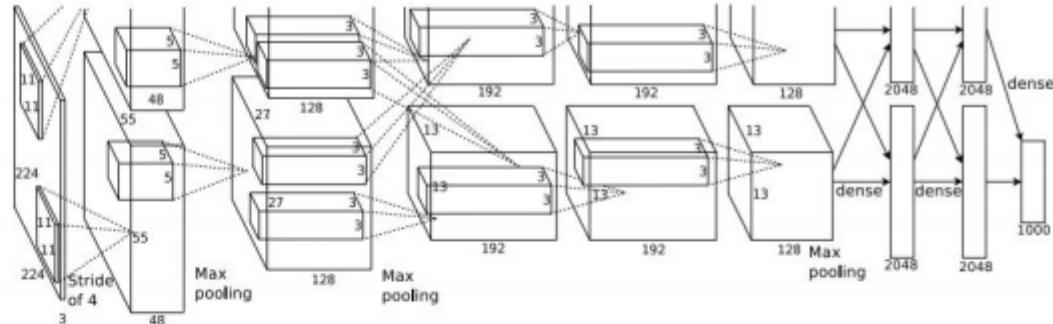
* 입력 크기 (H,W), 필터 크기 (FH, FW), 출력 크기를 (OG, OW), 패딩 P, 스트라이드 S일 때 출력의 크기는 ?

$$OH = (H + 2P - FH) / S + 1$$

$$OW = (W + 2P - FW) / S + 1$$

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

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

=>

Output volume [55x55x96]

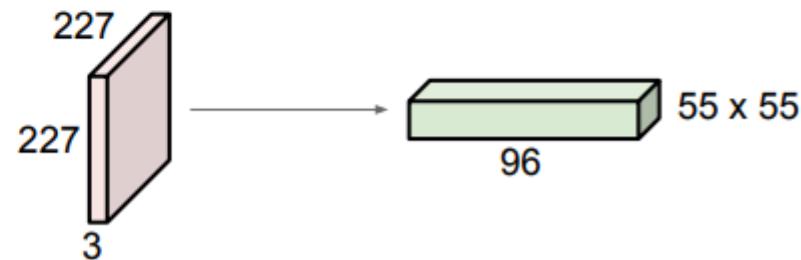
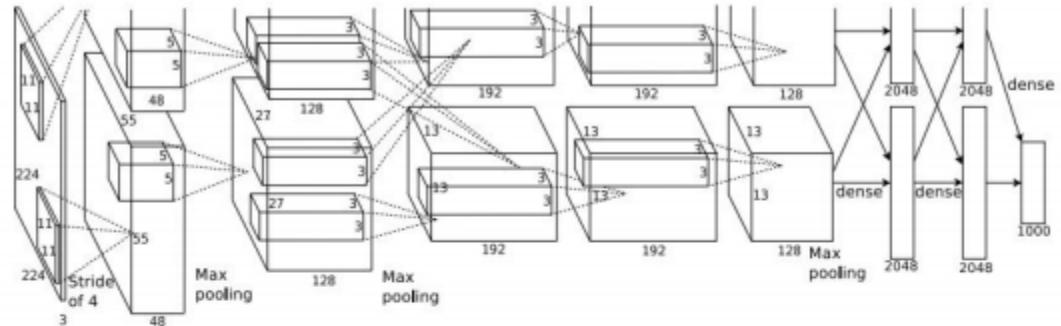


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

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

=>

Output volume [55x55x96]

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

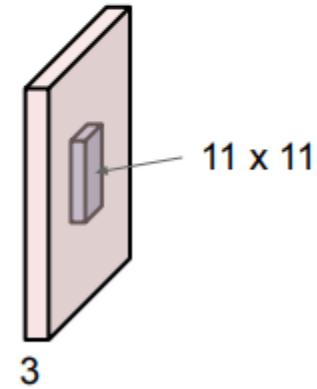
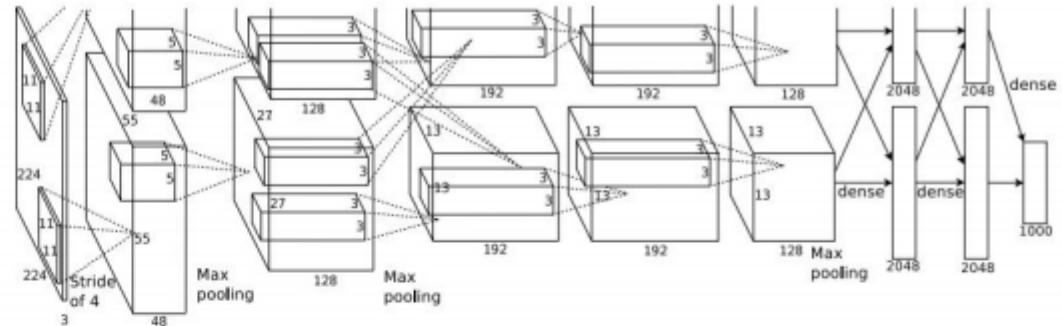


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

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

=>

Output volume **[55x55x96]**

Parameters: $(11 \times 11 \times 3) \times 96 = 35K$

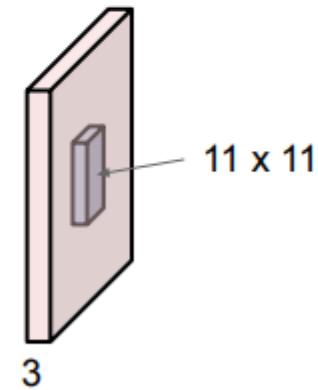
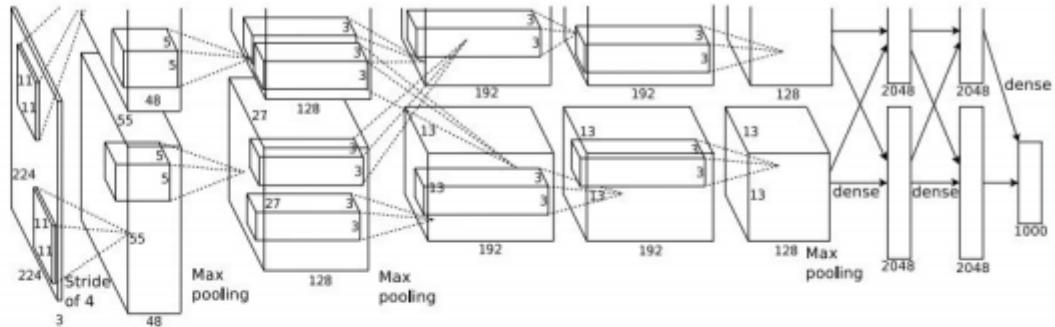


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

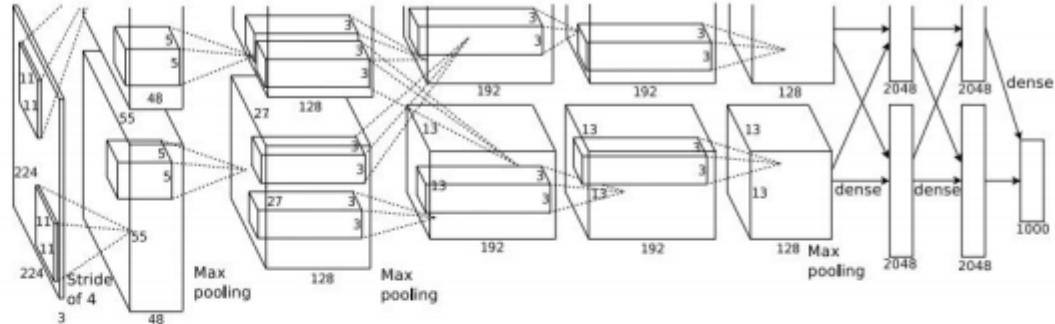
Second layer (POOL1): 3x3 filters applied at stride 2

Q: what is the output volume size? Hint: $(55-3)/2+1 = 27$

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

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

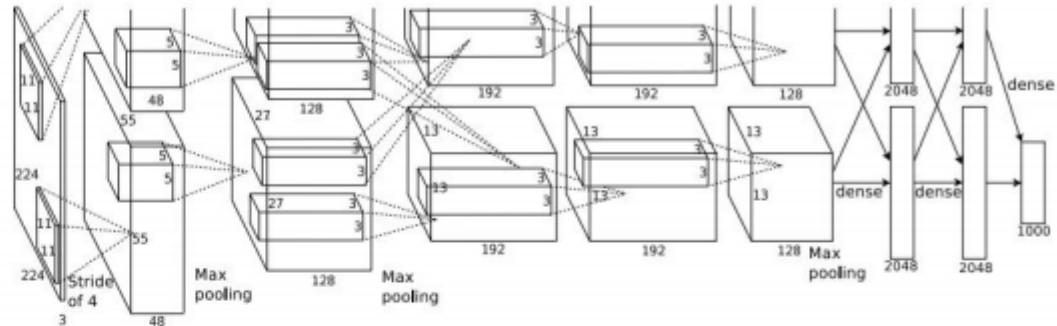
Output volume: 27x27x96

Q: what is the number of parameters in this layer?

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

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

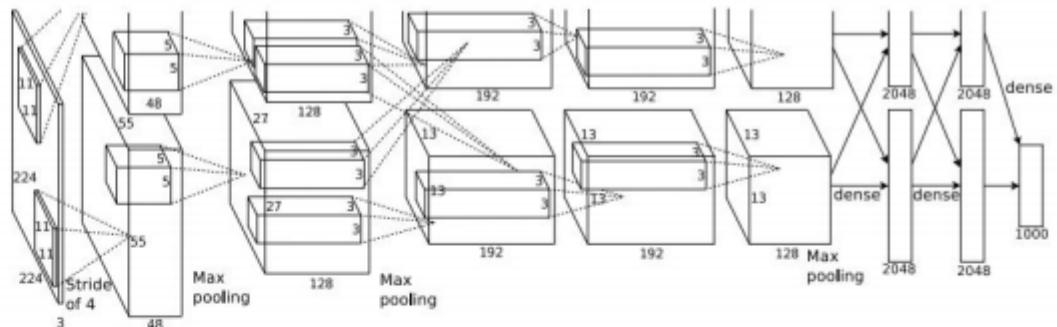
Output volume: 27x27x96

Parameters: 0!

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

After POOL1: 27x27x96

...

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Case Study: AlexNet

[Krizhevsky et al. 2012]

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)

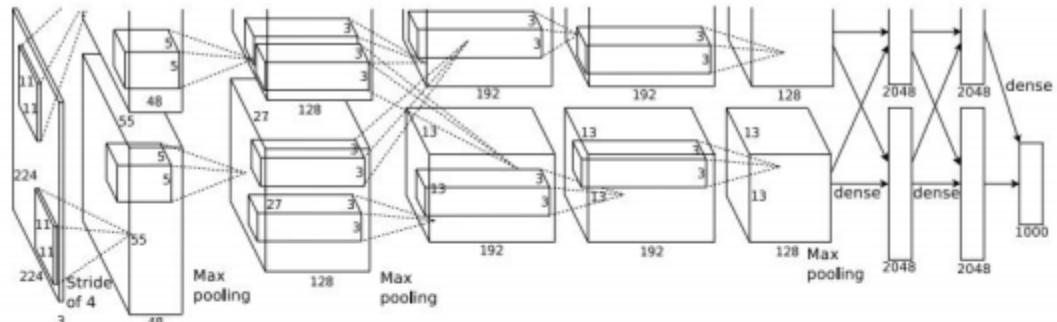


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Case Study: AlexNet

[Krizhevsky et al. 2012]

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

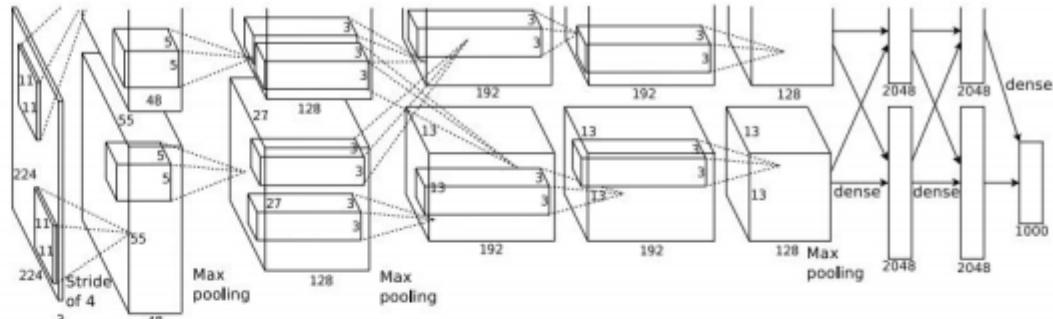
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)



Details/Retrospectives:

- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Case Study: AlexNet

[Krizhevsky et al. 2012]

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

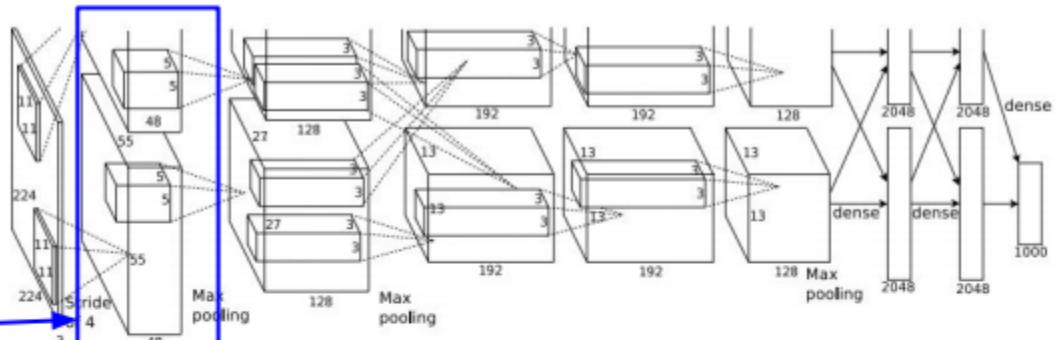
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)



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.

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Case Study: AlexNet

[Krizhevsky et al. 2012]

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

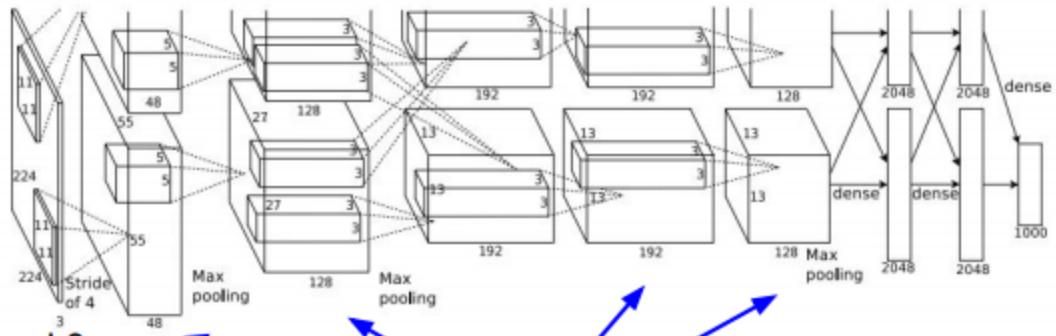
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[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

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Case Study: AlexNet

[Krizhevsky et al. 2012]

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

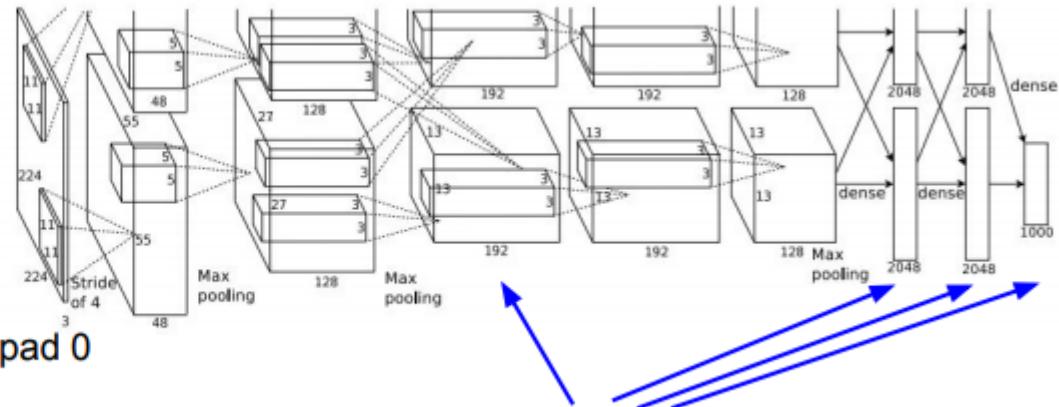
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

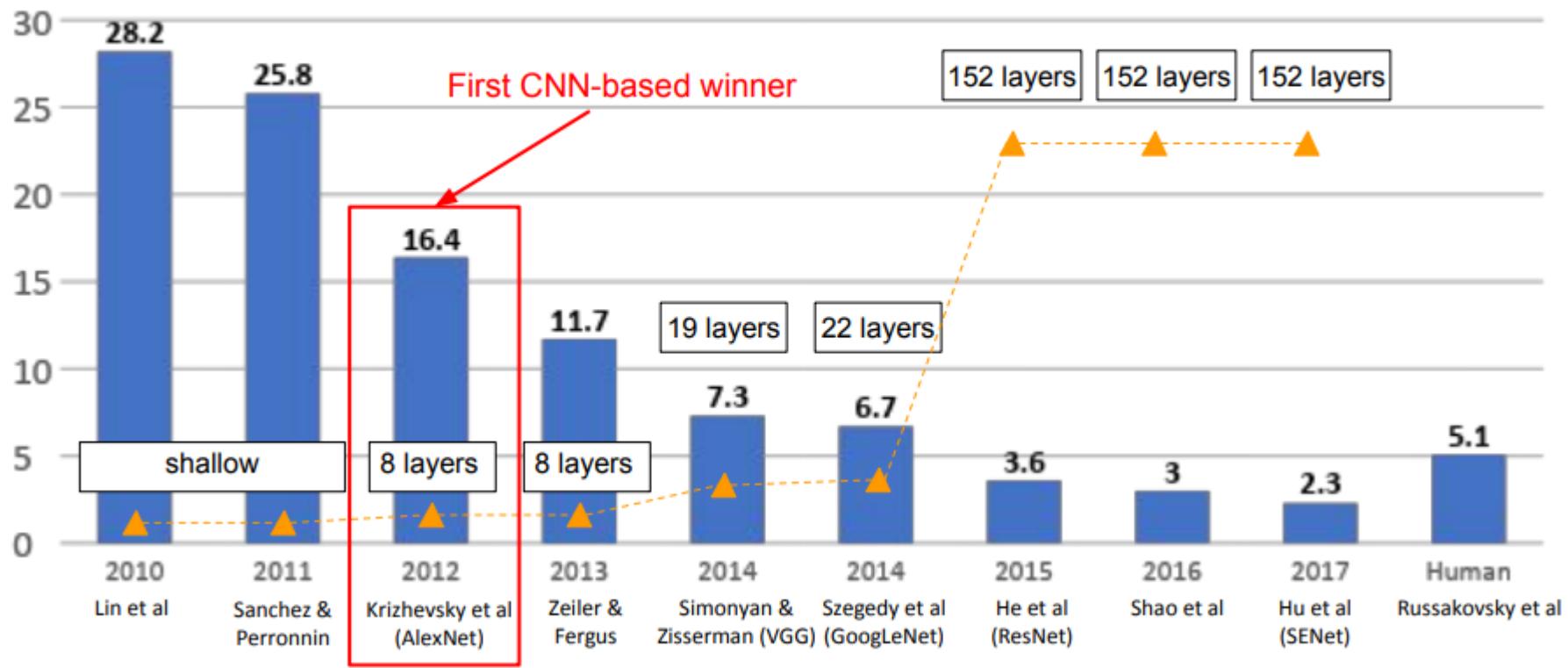
[1000] FC8: 1000 neurons (class scores)



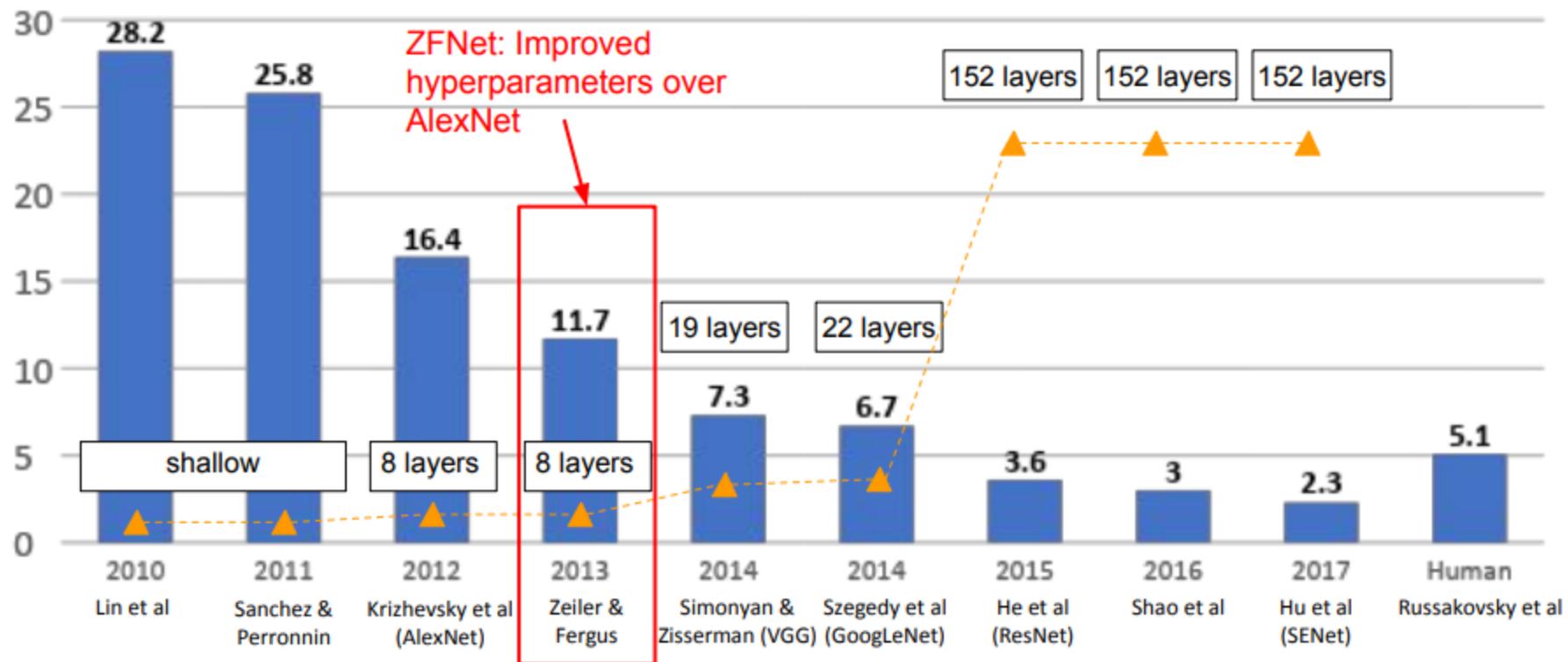
CONV3, FC6, FC7, FC8:
Connections with all feature maps in
preceding layer, communication
across GPUs

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

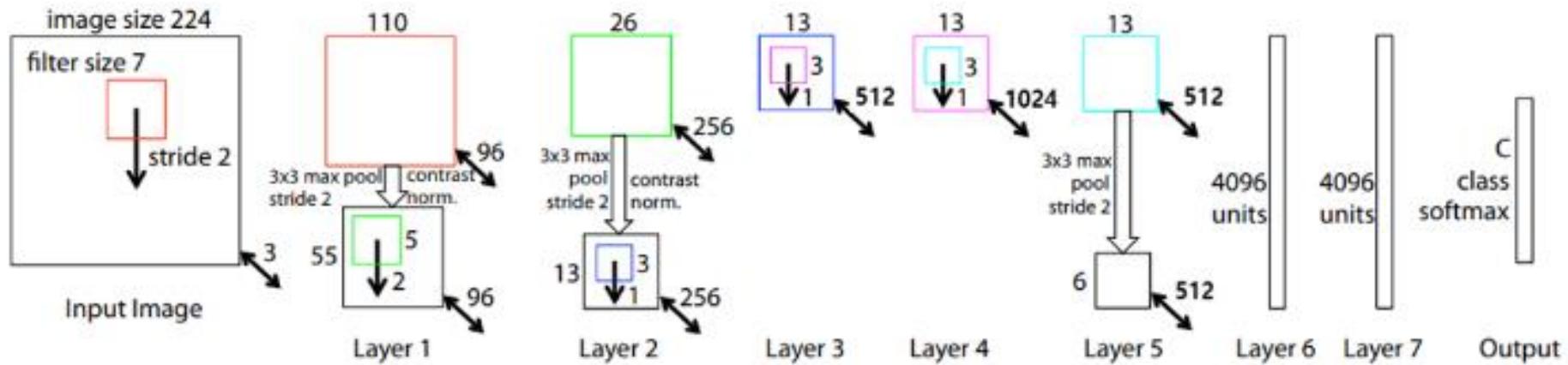


ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ZFNet

[Zeiler and Fergus, 2013]



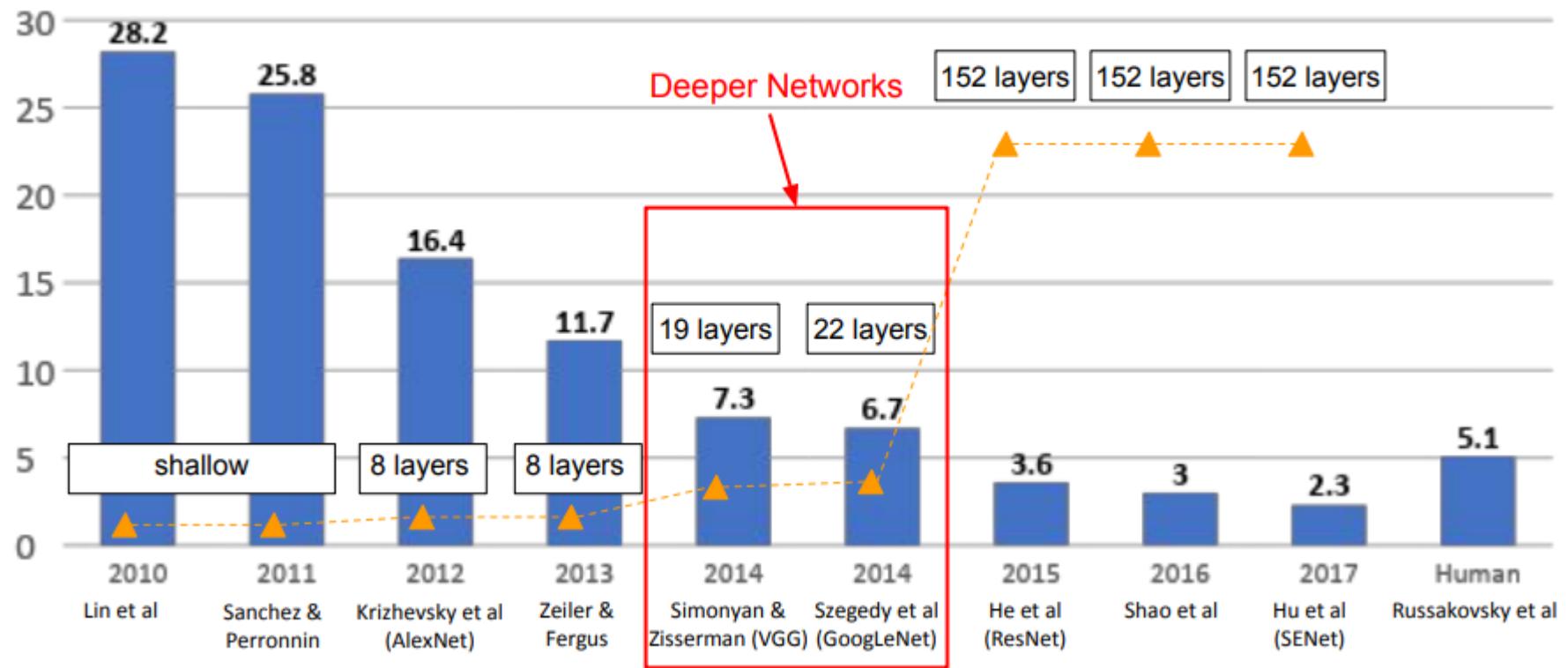
AlexNet but:

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

CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 16.4% \rightarrow 11.7%

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

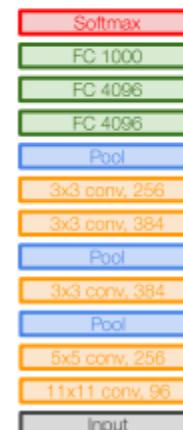
8 layers (AlexNet)

-> 16 - 19 layers (VGG16Net)

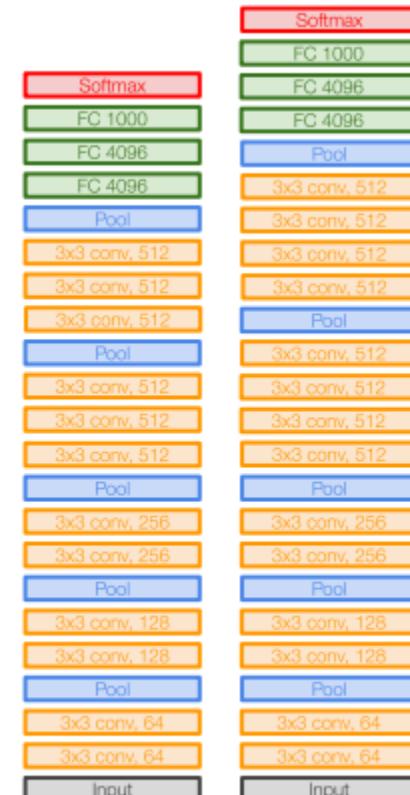
Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13 (ZFNet)

-> 7.3% top 5 error in ILSVRC'14



AlexNet



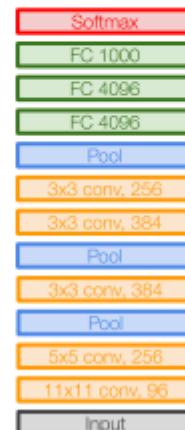
VGG16

VGG19

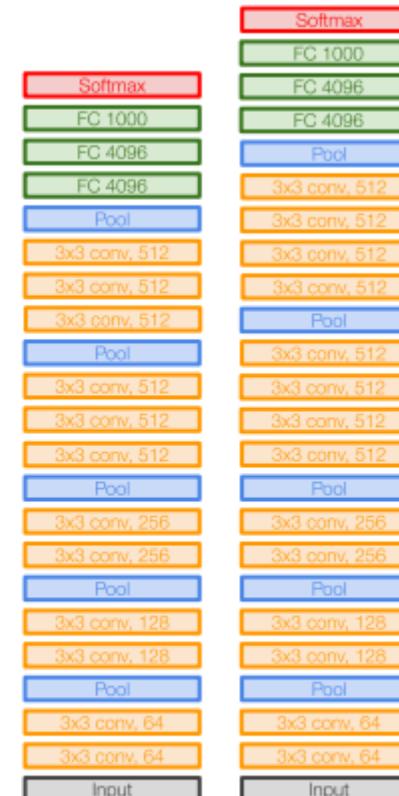
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

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



AlexNet



VGG16

VGG19

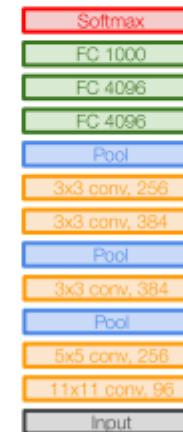
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

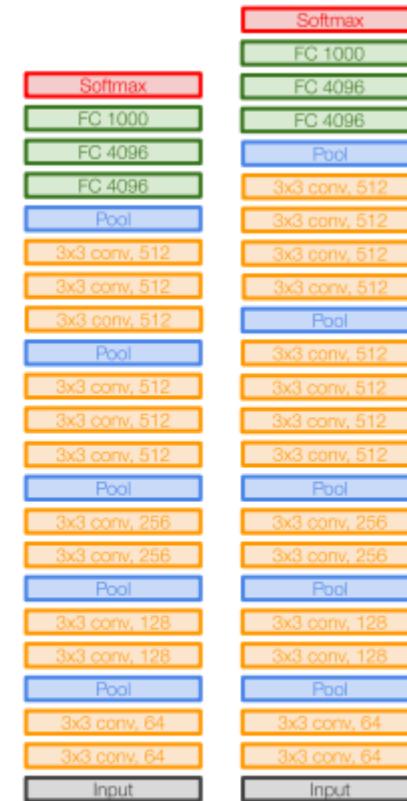
Q: Why use smaller filters? (3x3 conv)

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

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



AlexNet



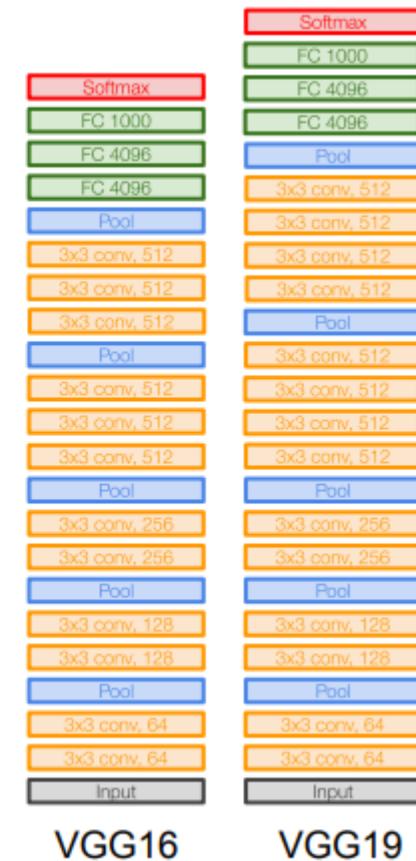
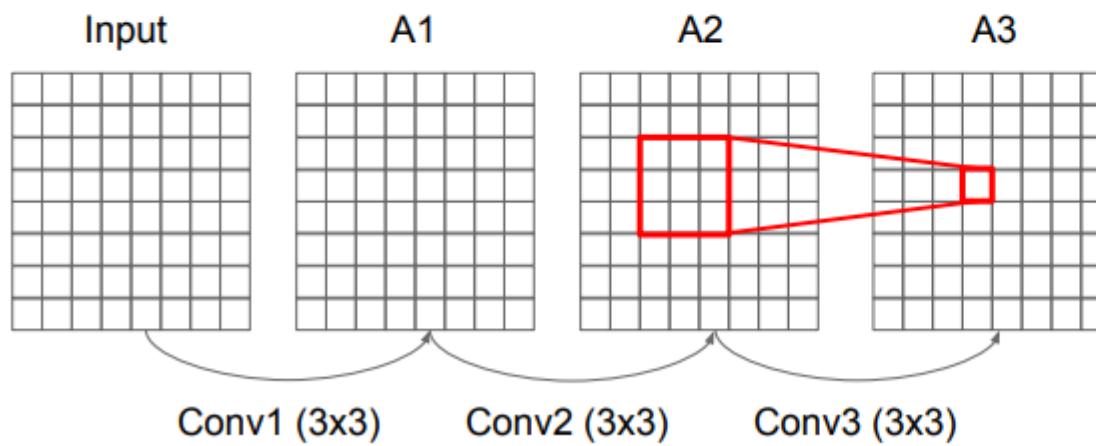
VGG16

VGG19

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

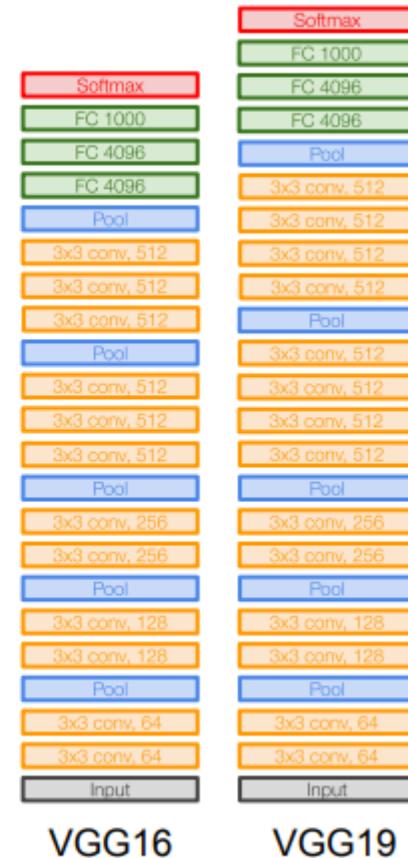
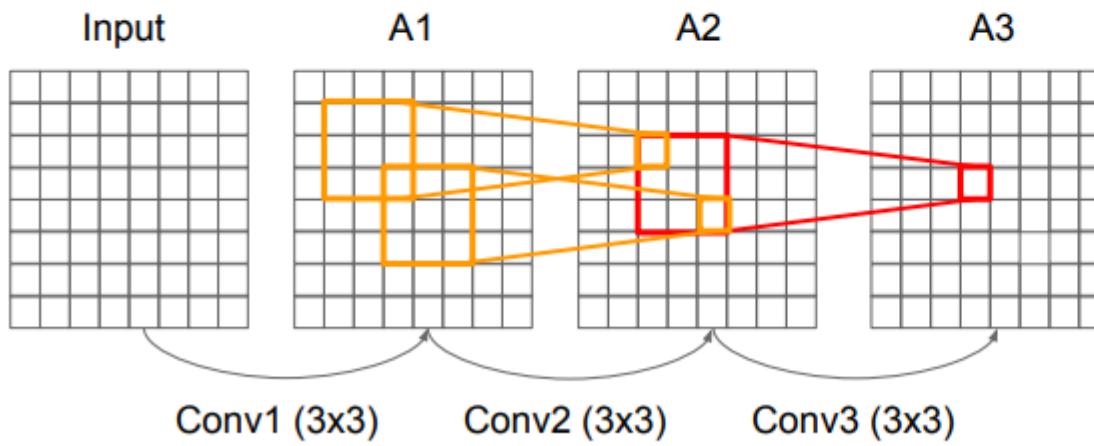
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

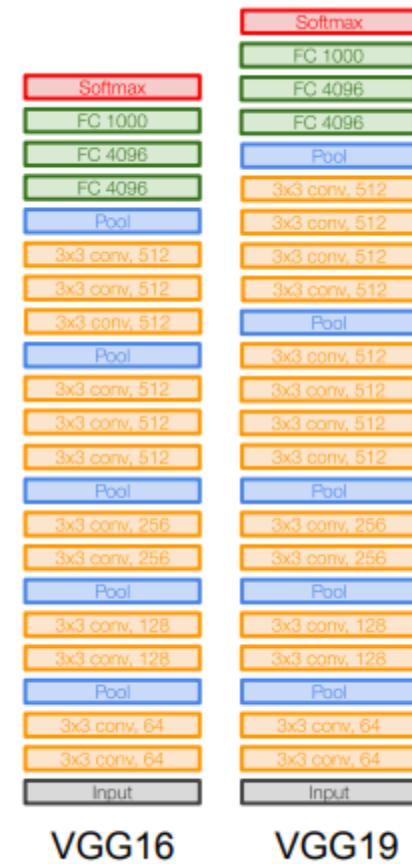
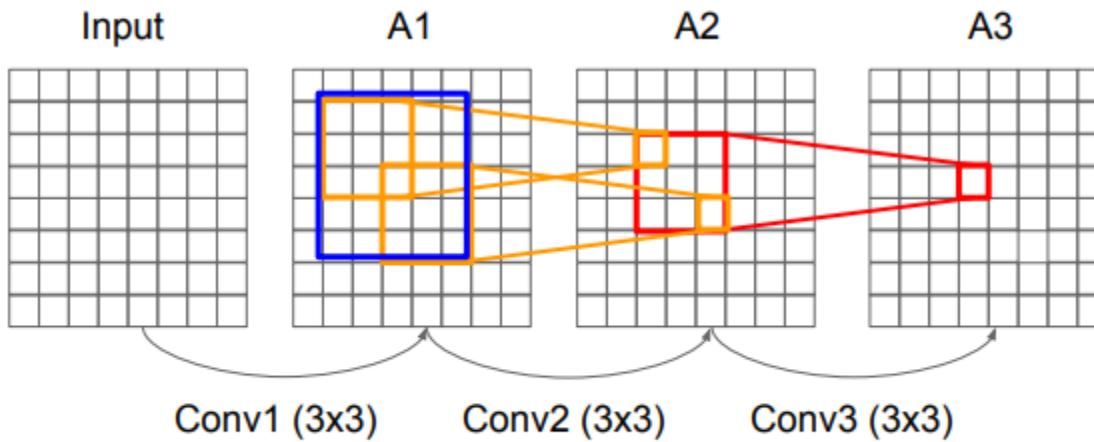
Q: What is the effective receptive field
of three 3x3 conv (stride 1) layers?



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

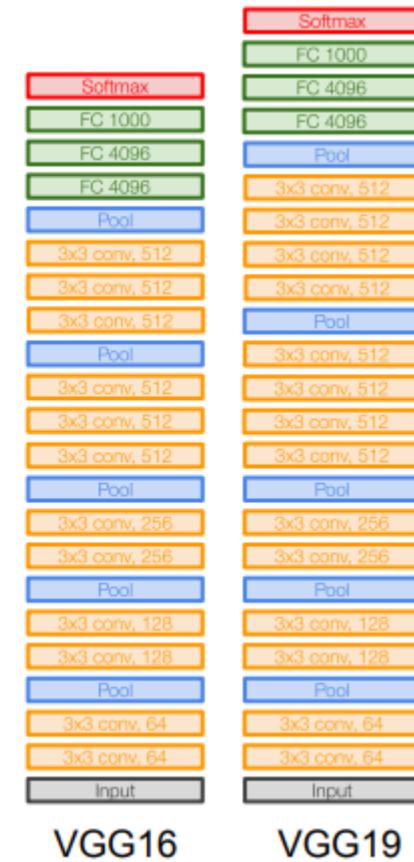
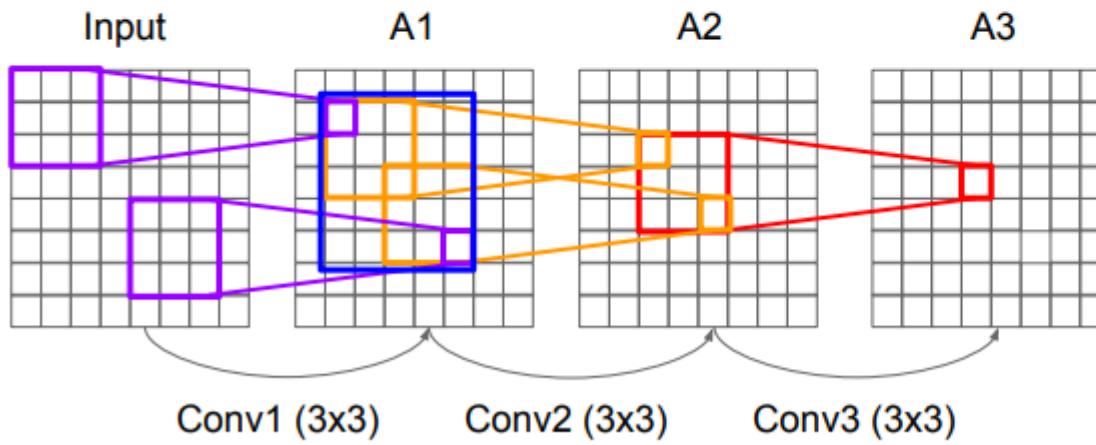
Q: What is the effective receptive field
of three 3x3 conv (stride 1) layers?



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

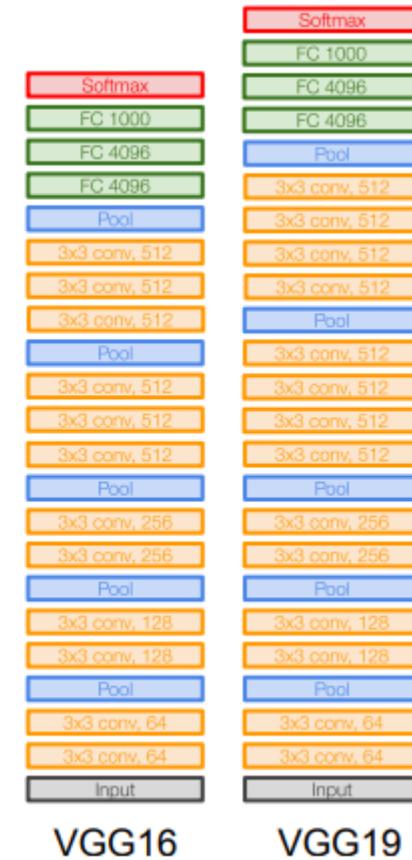
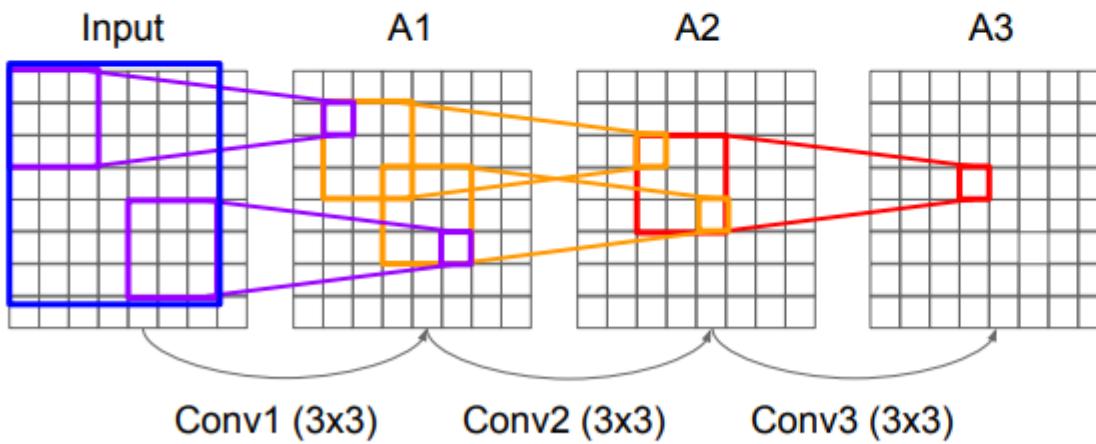
Q: What is the effective receptive field
of three 3x3 conv (stride 1) layers?



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

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



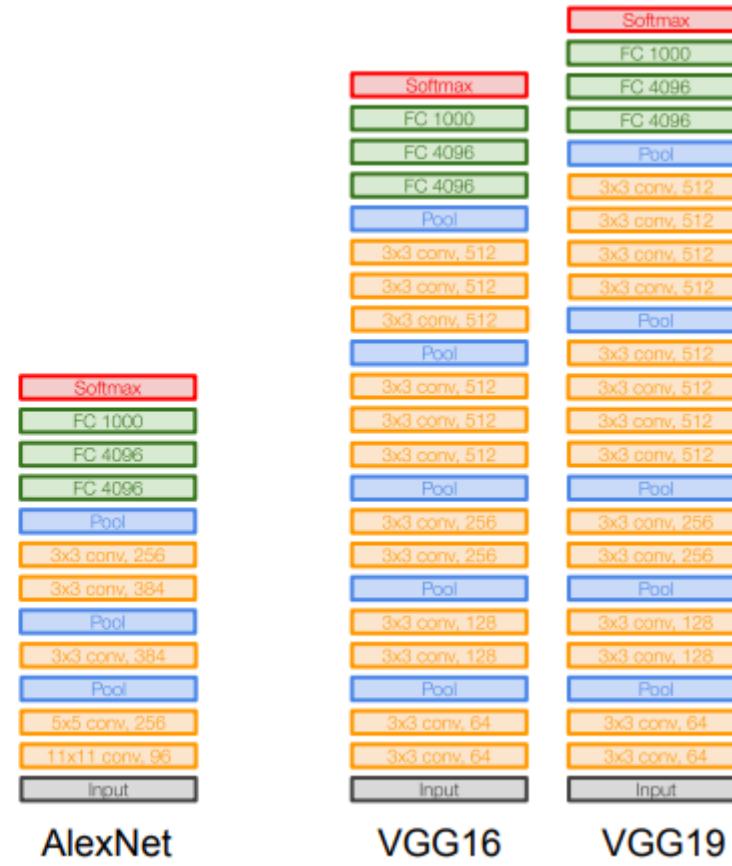
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

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

[7x7]



Case Study: VGGNet

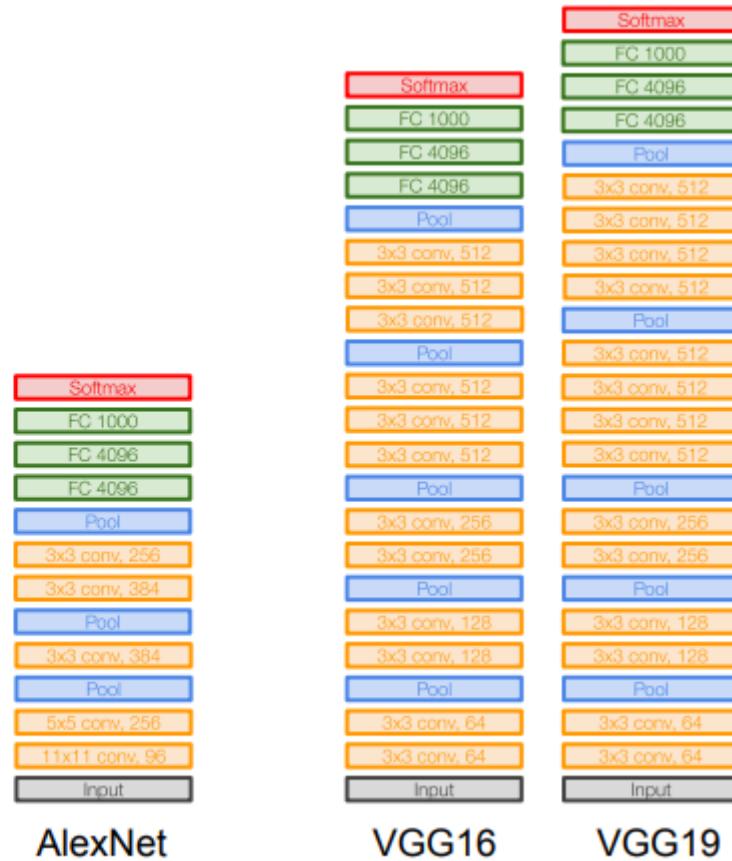
[Simonyan and Zisserman, 2014]

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

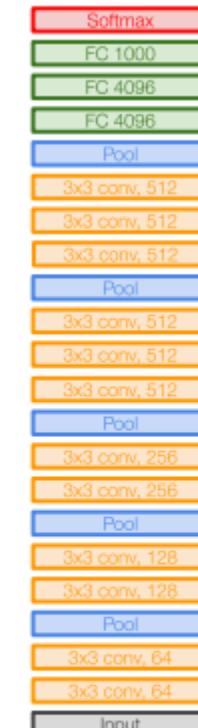
But deeper, more non-linearities

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



VGGNet

INPUT: [224x224x3] memory: $224 \times 224 \times 3 = 150K$ params: 0 (not counting biases)
 CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2M$ params: $(3 \times 3 \times 3) \times 64 = 1,728$
 CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2M$ params: $(3 \times 3 \times 64) \times 64 = 36,864$
 POOL2: [112x112x64] memory: $112 \times 112 \times 64 = 800K$ params: 0
 CONV3-128: [112x112x128] memory: $112 \times 112 \times 128 = 1.6M$ params: $(3 \times 3 \times 64) \times 128 = 73,728$
 CONV3-128: [112x112x128] memory: $112 \times 112 \times 128 = 1.6M$ params: $(3 \times 3 \times 128) \times 128 = 147,456$
 POOL2: [56x56x128] memory: $56 \times 56 \times 128 = 400K$ params: 0
 CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 128) \times 256 = 294,912$
 CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 256) \times 256 = 589,824$
 CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 256) \times 256 = 589,824$
 POOL2: [28x28x256] memory: $28 \times 28 \times 256 = 200K$ params: 0
 CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 256) \times 512 = 1,179,648$
 CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
 CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
 POOL2: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: 0
 CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
 CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
 CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
 POOL2: [7x7x512] memory: $7 \times 7 \times 512 = 25K$ params: 0
 FC: [1x1x4096] memory: 4096 params: $7 \times 7 \times 512 \times 4096 = 102,760,448$
 FC: [1x1x4096] memory: 4096 params: $4096 \times 4096 = 16,777,216$
 FC: [1x1x1000] memory: 1000 params: $4096 \times 1000 = 4,096,000$



VGG16

TOTAL memory: $24M * 4 \text{ bytes} \approx 96\text{MB} / \text{image}$ (for a forward pass)

TOTAL params: 138M parameters

VGGNet

INPUT: [224x224x3] memory: $224 \times 224 \times 3 = 150K$ params: 0 (not counting biases)

CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2M$ params: $(3 \times 3 \times 3) \times 64 = 1,728$

CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2M$ params: $(3 \times 3 \times 64) \times 64 = 36,864$

POOL2: [112x112x64] memory: $112 \times 112 \times 64 = 800K$ params: 0

CONV3-128: [112x112x128] memory: $112 \times 112 \times 128 = 1.6M$ params: $(3 \times 3 \times 64) \times 128 = 73,728$

CONV3-128: [112x112x128] memory: $112 \times 112 \times 128 = 1.6M$ params: $(3 \times 3 \times 128) \times 128 = 147,456$

POOL2: [56x56x128] memory: $56 \times 56 \times 128 = 400K$ params: 0

CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 128) \times 256 = 294,912$

CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 256) \times 256 = 589,824$

CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 256) \times 256 = 589,824$

POOL2: [28x28x256] memory: $28 \times 28 \times 256 = 200K$ params: 0

CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 256) \times 512 = 1,179,648$

CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

POOL2: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: 0

CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

POOL2: [7x7x512] memory: $7 \times 7 \times 512 = 25K$ params: 0

FC: [1x1x4096] memory: 4096 params: $7 \times 7 \times 512 \times 4096 = 102,760,448$

FC: [1x1x4096] memory: 4096 params: $4096 \times 4096 = 16,777,216$

FC: [1x1x1000] memory: 1000 params: $4096 \times 1000 = 4,096,000$

TOTAL memory: $24M * 4$ bytes $\approx 96MB / \text{image}$ (only forward! ~ 2 for bwd)

TOTAL params: 138M parameters

Note:

Most memory is in early CONV

Most params are in late FC

VGGNet

INPUT: [224x224x3] memory: $224 \times 224 \times 3 = 150\text{K}$ params: 0 (not counting biases)
 CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2\text{M}$ params: $(3 \times 3 \times 3) \times 64 = 1,728$
 CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2\text{M}$ params: $(3 \times 3 \times 64) \times 64 = 36,864$
 POOL2: [112x112x64] memory: $112 \times 112 \times 64 = 800\text{K}$ params: 0
 CONV3-128: [112x112x128] memory: $112 \times 112 \times 128 = 1.6\text{M}$ params: $(3 \times 3 \times 64) \times 128 = 73,728$
 CONV3-128: [112x112x128] memory: $112 \times 112 \times 128 = 1.6\text{M}$ params: $(3 \times 3 \times 128) \times 128 = 147,456$
 POOL2: [56x56x128] memory: $56 \times 56 \times 128 = 400\text{K}$ params: 0
 CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800\text{K}$ params: $(3 \times 3 \times 128) \times 256 = 294,912$
 CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800\text{K}$ params: $(3 \times 3 \times 256) \times 256 = 589,824$
 CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800\text{K}$ params: $(3 \times 3 \times 256) \times 256 = 589,824$
 POOL2: [28x28x256] memory: $28 \times 28 \times 256 = 200\text{K}$ params: 0
 CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400\text{K}$ params: $(3 \times 3 \times 256) \times 512 = 1,179,648$
 CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400\text{K}$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
 CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400\text{K}$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
 POOL2: [14x14x512] memory: $14 \times 14 \times 512 = 100\text{K}$ params: 0
 CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100\text{K}$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
 CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100\text{K}$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
 CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100\text{K}$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
 POOL2: [7x7x512] memory: $7 \times 7 \times 512 = 25\text{K}$ params: 0
 FC: [1x1x4096] memory: 4096 params: $7 \times 7 \times 512 \times 4096 = 102,760,448$
 FC: [1x1x4096] memory: 4096 params: $4096 \times 4096 = 16,777,216$
 FC: [1x1x1000] memory: 1000 params: $4096 \times 1000 = 4,096,000$

TOTAL memory: $24\text{M} * 4 \text{ bytes} \approx 96\text{MB} / \text{image}$ (only forward! ~ 2 for bwd)

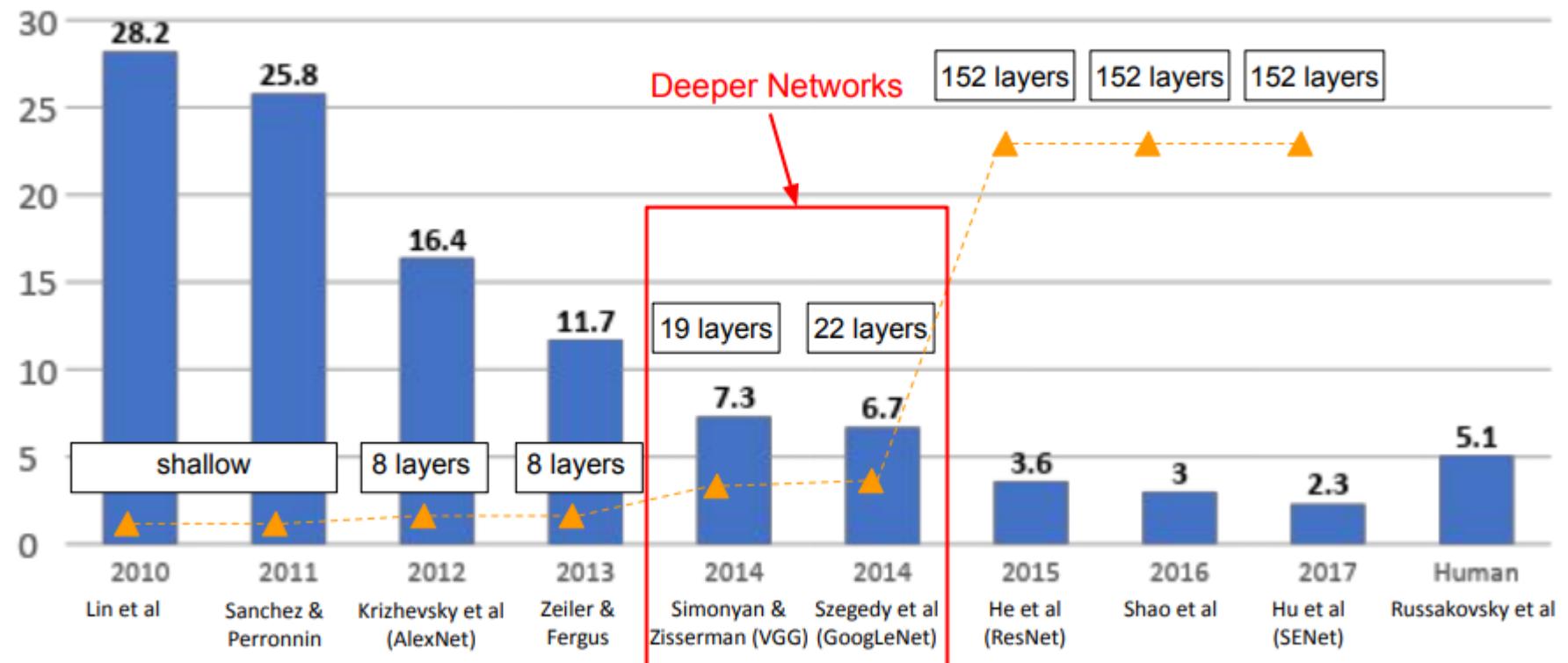
TOTAL params: 138M parameters



VGG16

Common names

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

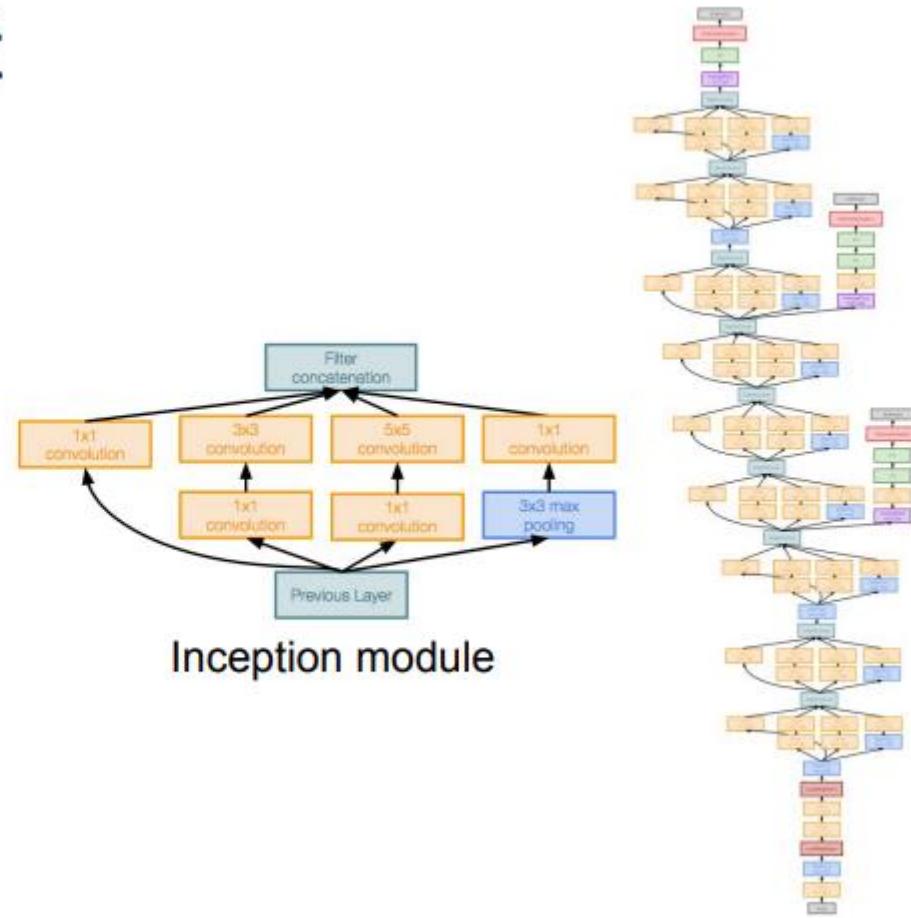


Case Study: GoogLeNet

[Szegedy et al., 2014]

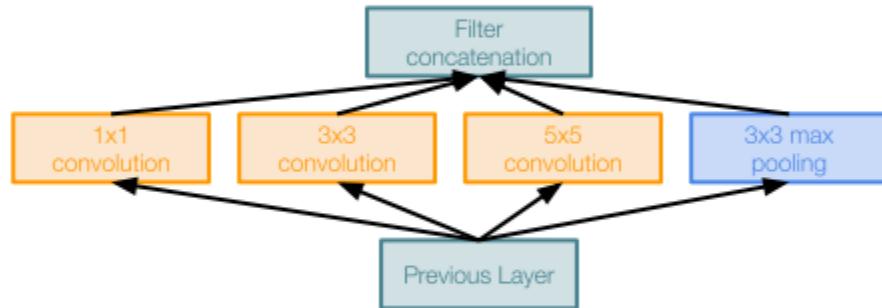
Deeper networks, with computational efficiency

- ILSVRC'14 classification winner (6.7% top 5 error)
- 22 layers
- Only 5 million parameters!
12x less than AlexNet
27x less than VGG-16
- Efficient “Inception” module
- No FC layers



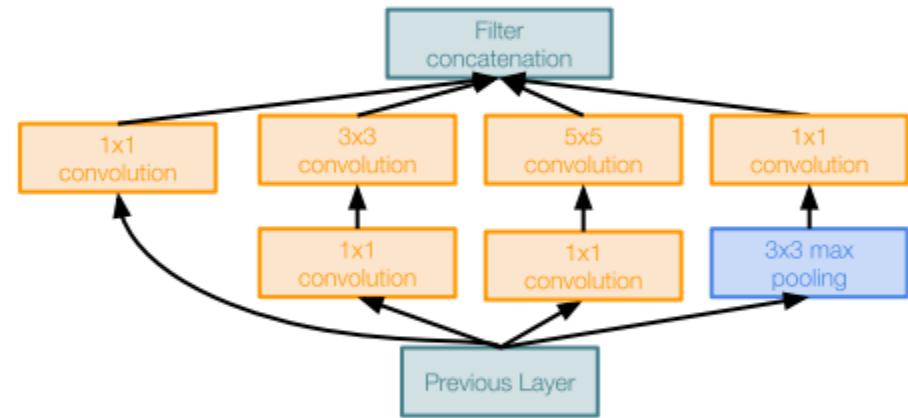
Case Study: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

다양한 크기의 특징을 추출하는 효과



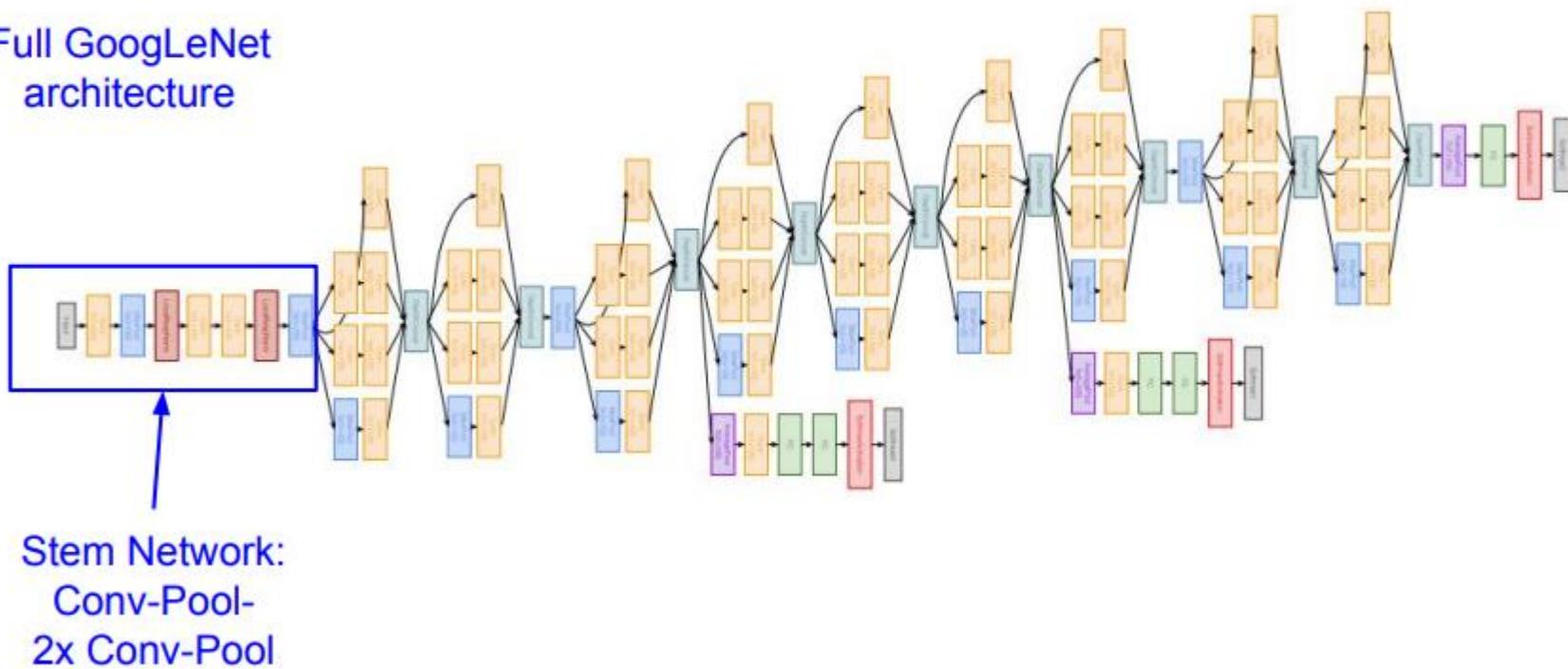
Inception module with dimension reduction

차원을 줄여서 다양한 크기의 특징을 추출하는 효과

Case Study: GoogLeNet

[Szegedy et al., 2014]

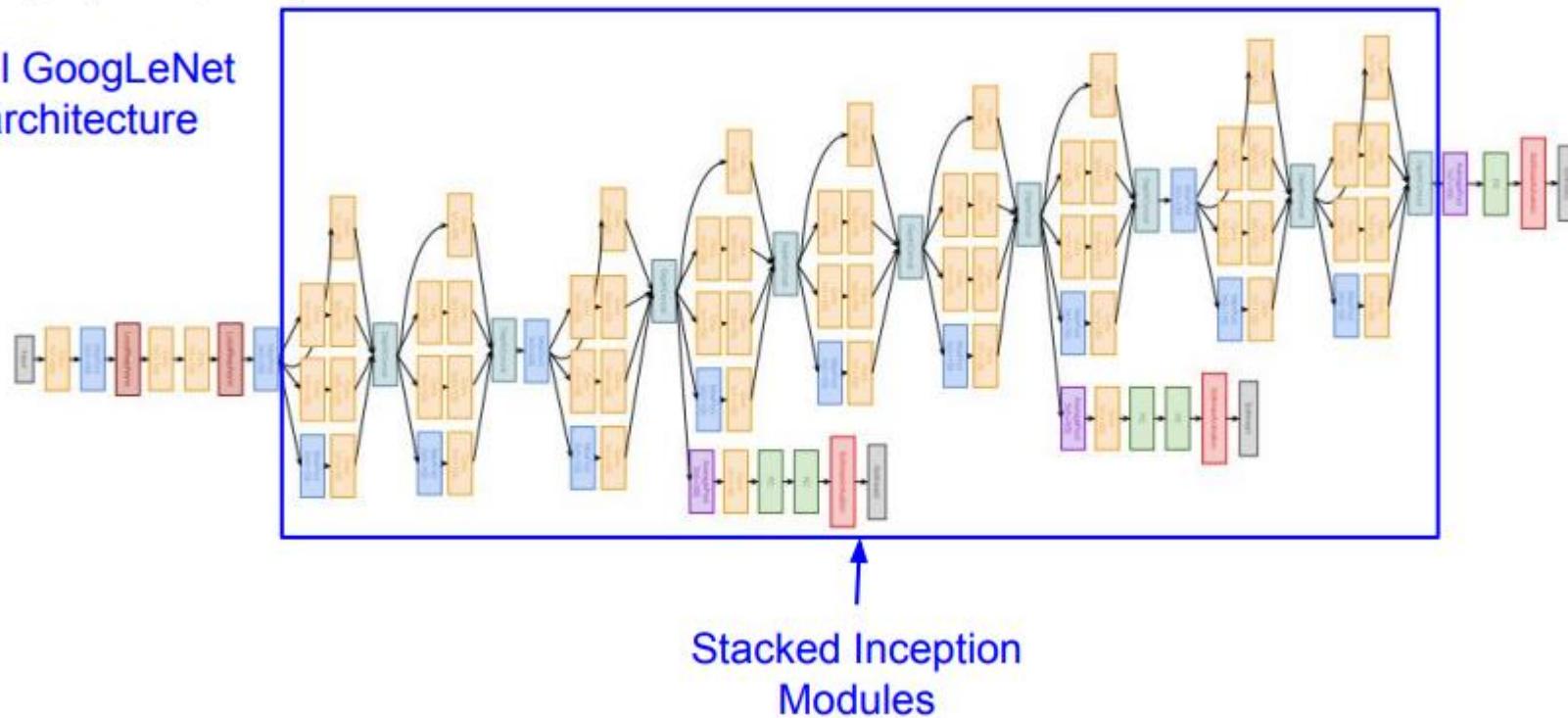
Full GoogLeNet
architecture



Case Study: GoogLeNet

[Szegedy et al., 2014]

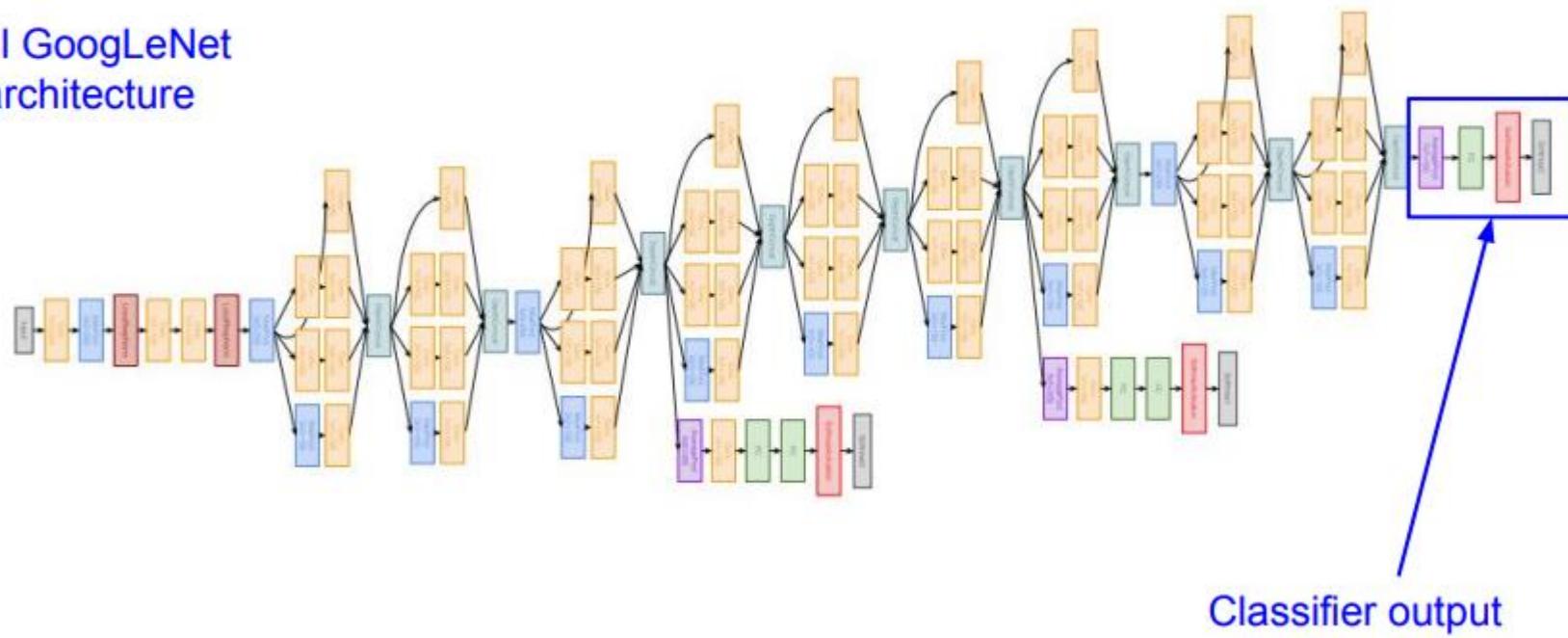
Full GoogLeNet architecture



Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet
architecture



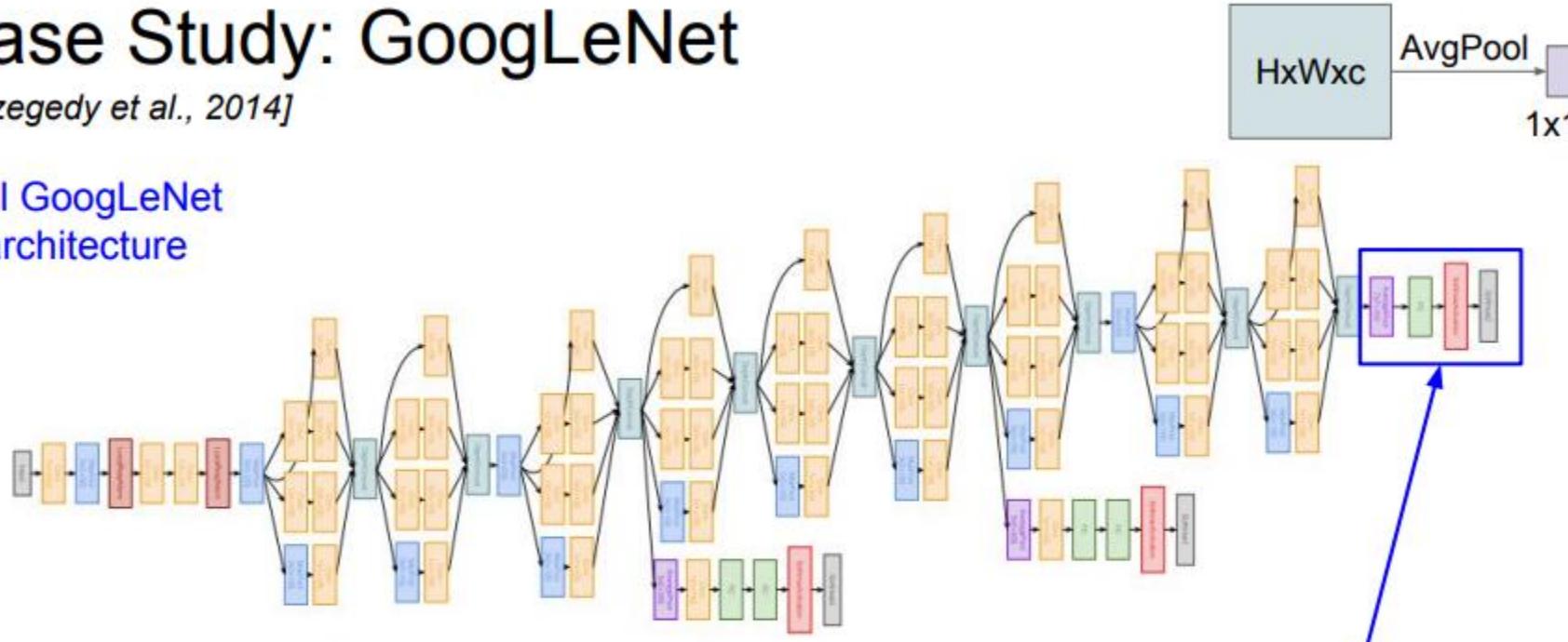
Classifier output

FC layer를 두지 않고 average pooling 사용

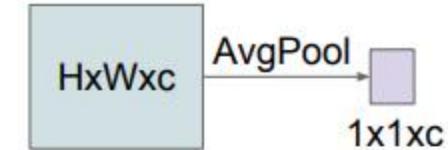
Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet
architecture



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!

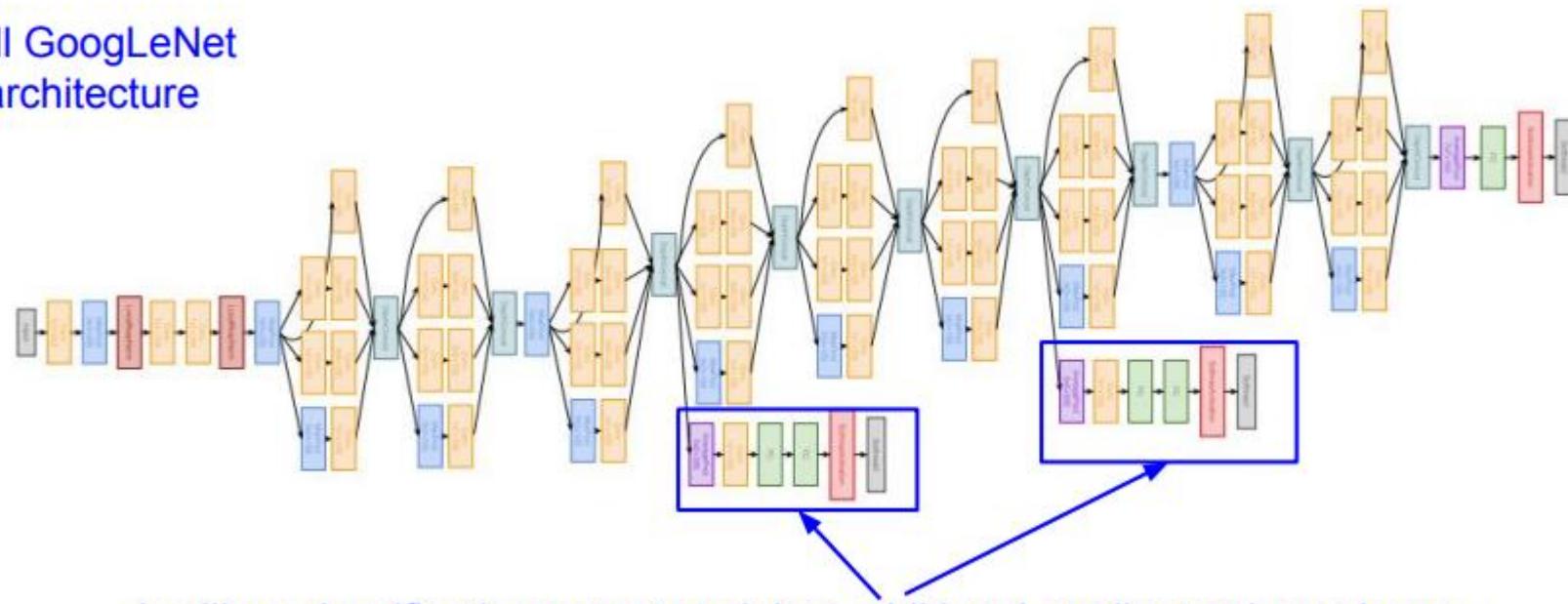


Classifier output

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet
architecture



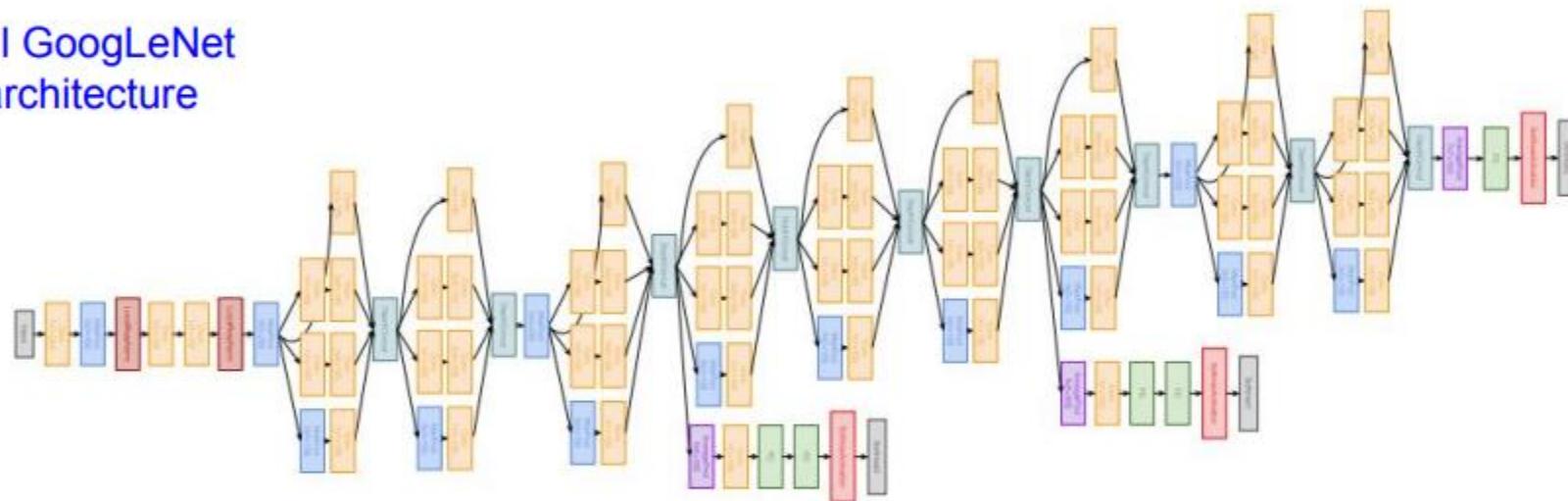
Auxiliary classification outputs to inject additional gradient at lower layers
(AvgPool-1x1Conv-FC-FC-Softmax)

기울기 소멸을 해결하기 위한 보조 분류기

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet
architecture



22 total layers with weights

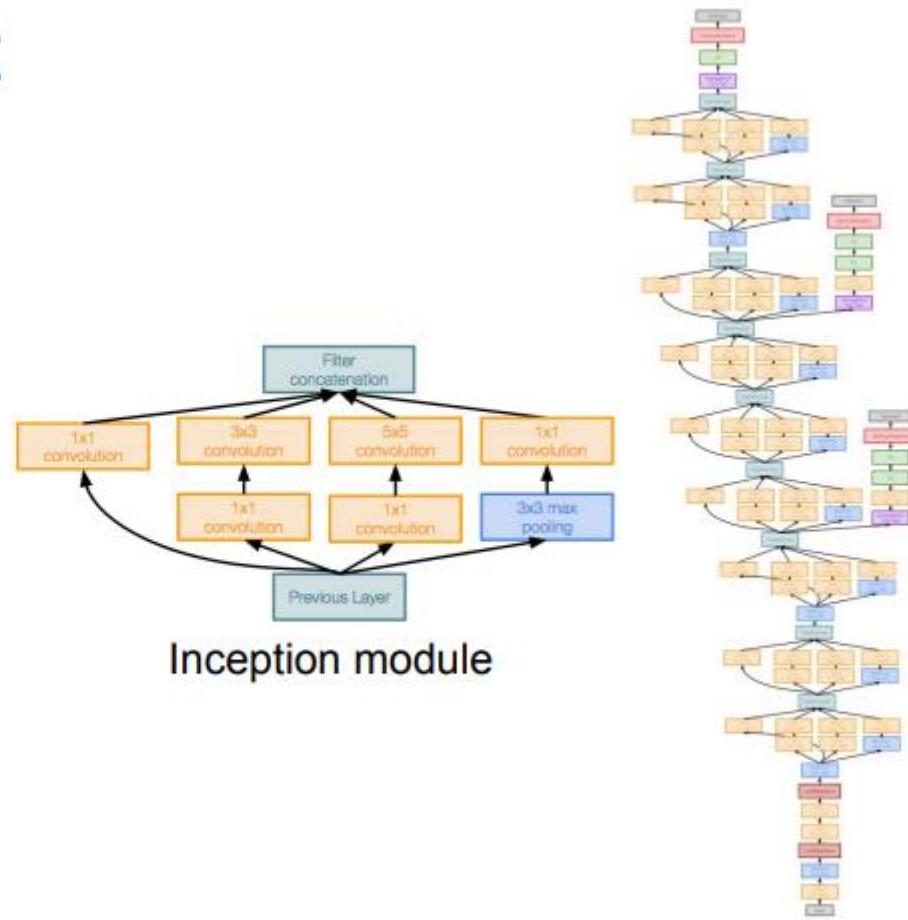
(parallel layers count as 1 layer => 2 layers per Inception module. Don't count auxiliary output layers)

Case Study: GoogLeNet

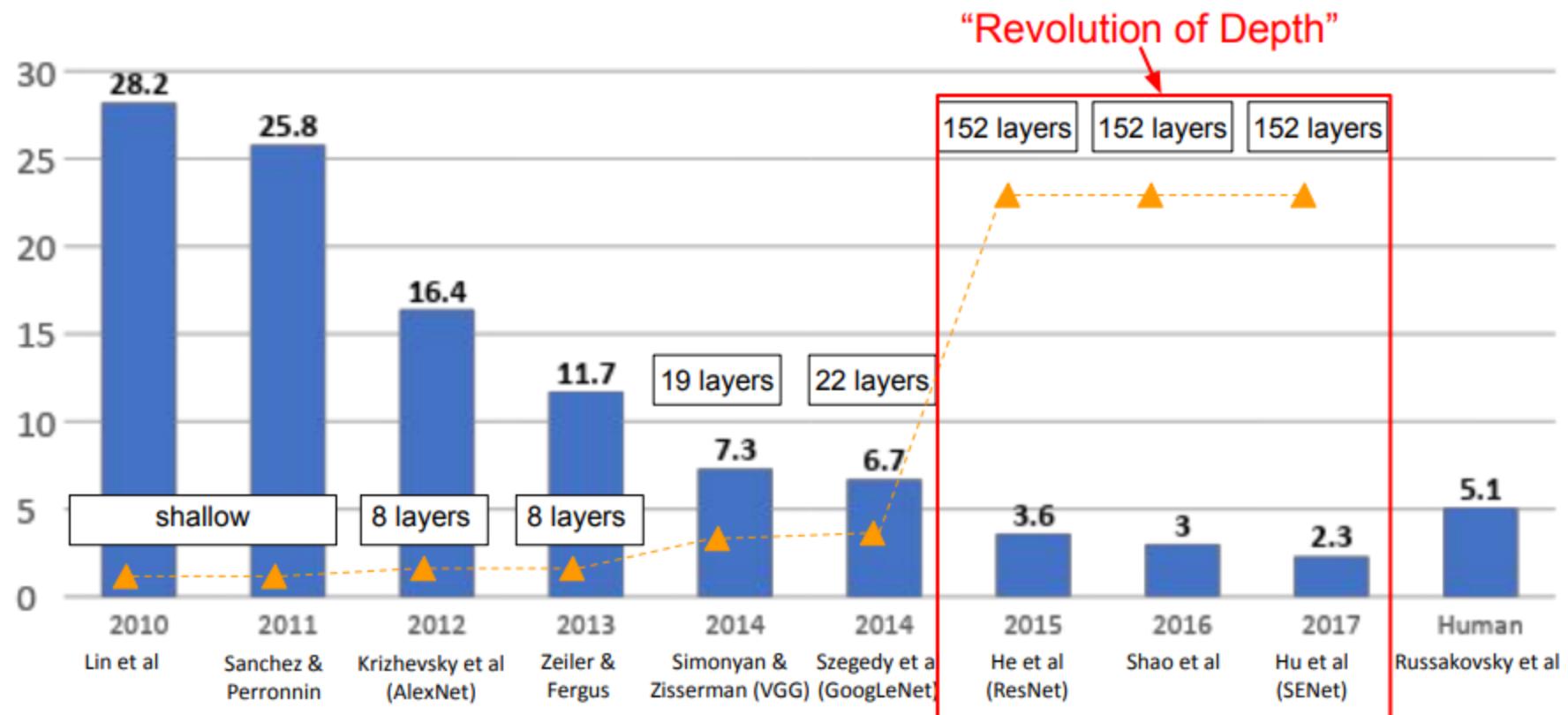
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- Avoids expensive FC layers
- 12x less params than AlexNet
- 27x less params than VGG-16
- ILSVRC’14 classification winner (6.7% top 5 error)



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

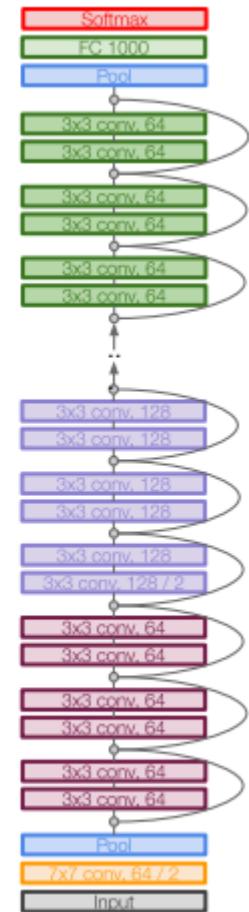
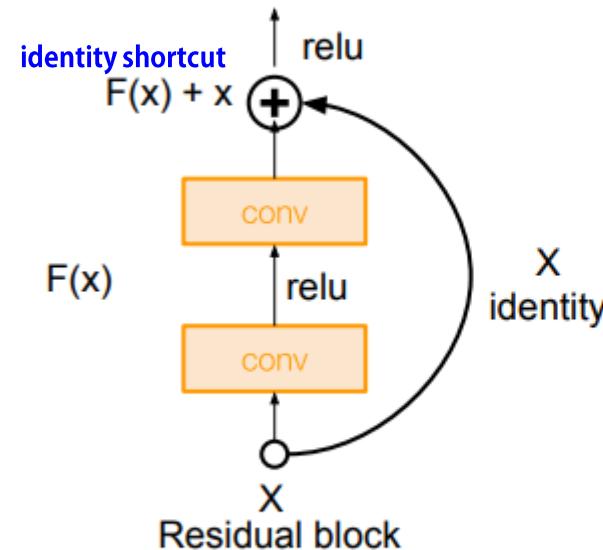


Case Study: ResNet

[He et al., 2015]

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!



ResNet

- ResNet : Degradation 문제를 해결한 구조
- Degradation : 모델의 깊이가 깊어지다 보면 어느 순간 더 얕은 모델의 성능보다 더 나빠지는 현상이 발생

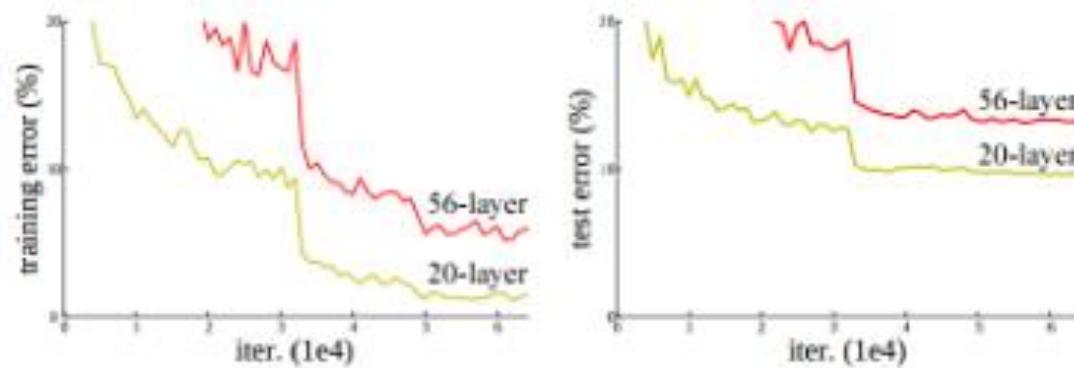
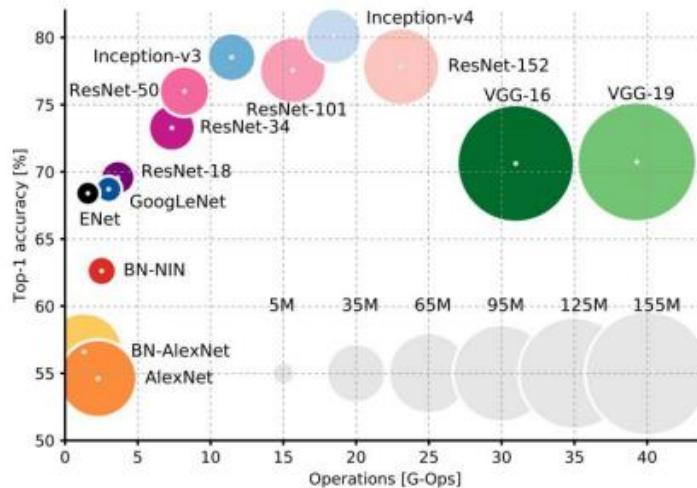
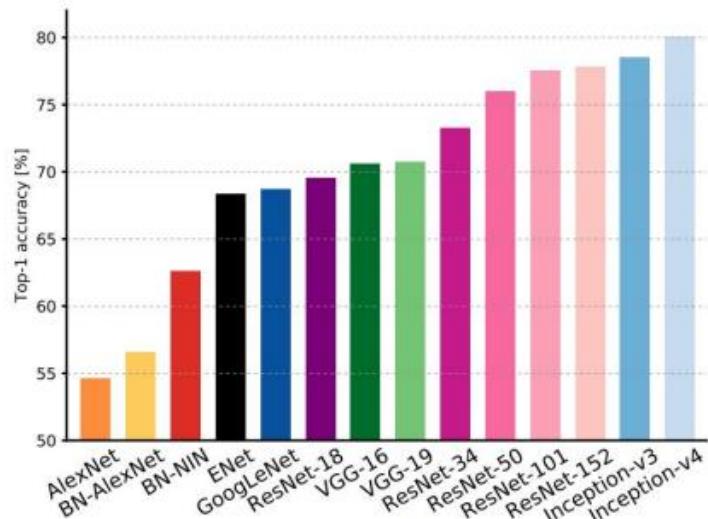


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

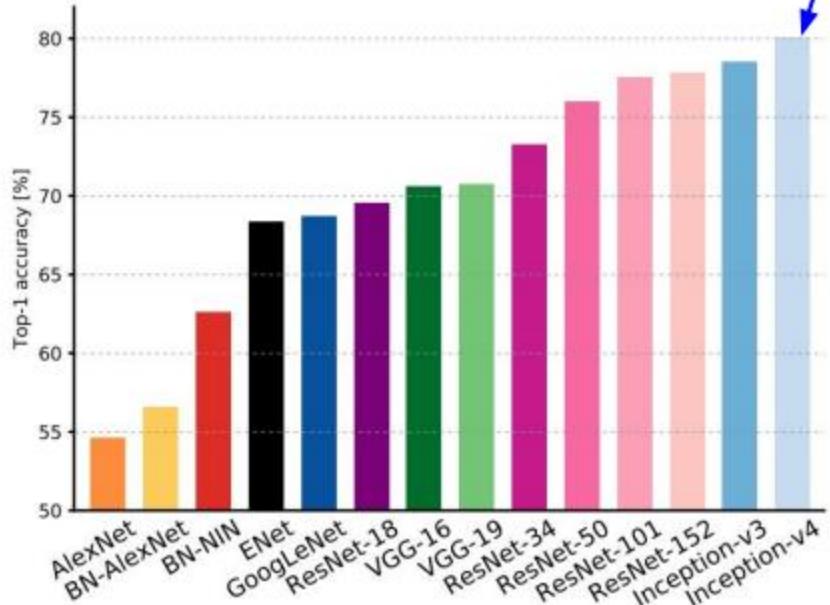
Comparing complexity...



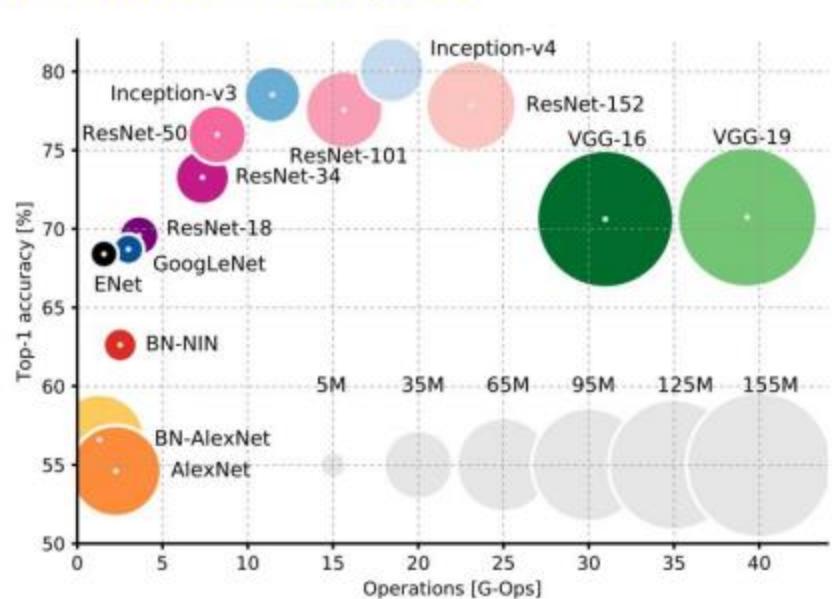
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Comparing complexity...



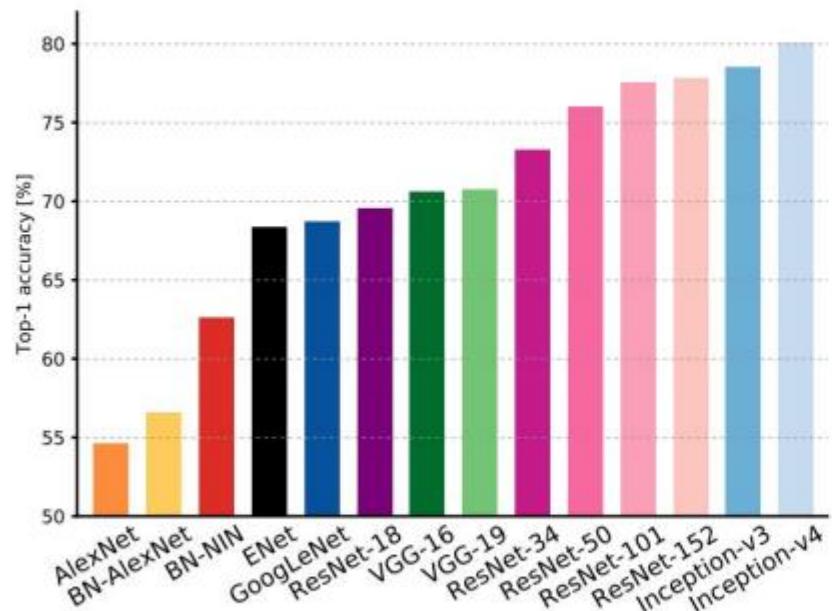
Inception-v4: Resnet + Inception!



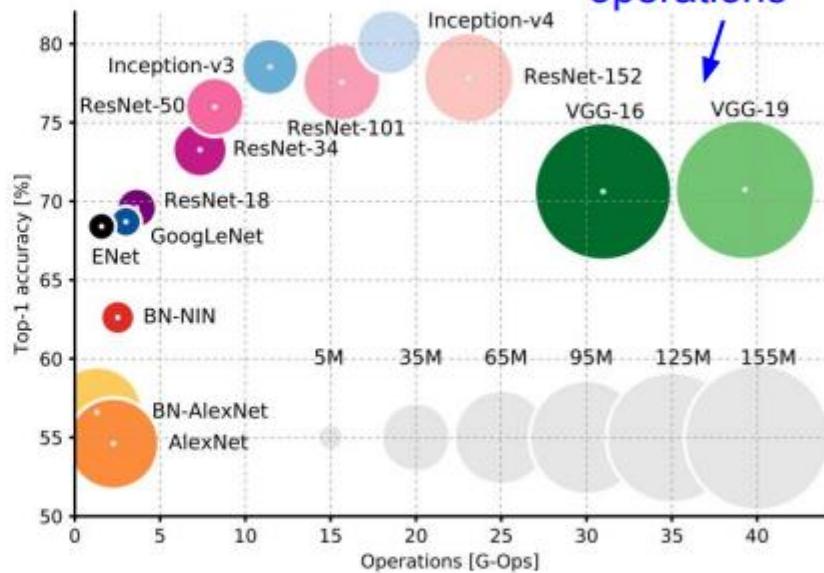
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Comparing complexity...



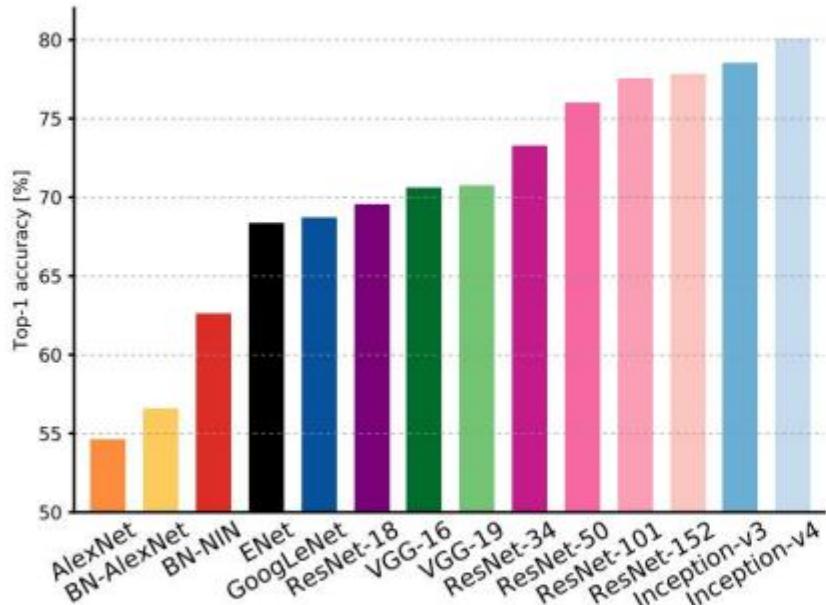
VGG: most parameters, most operations



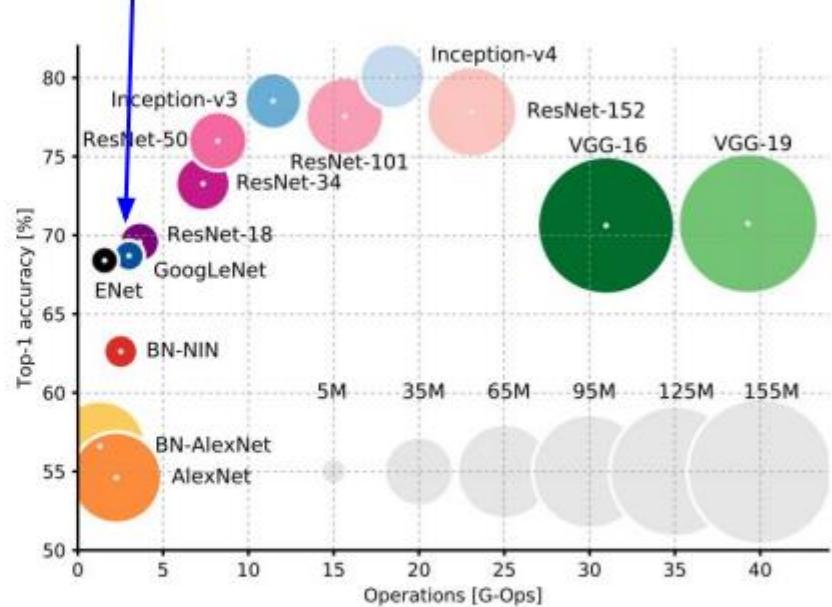
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Comparing complexity...



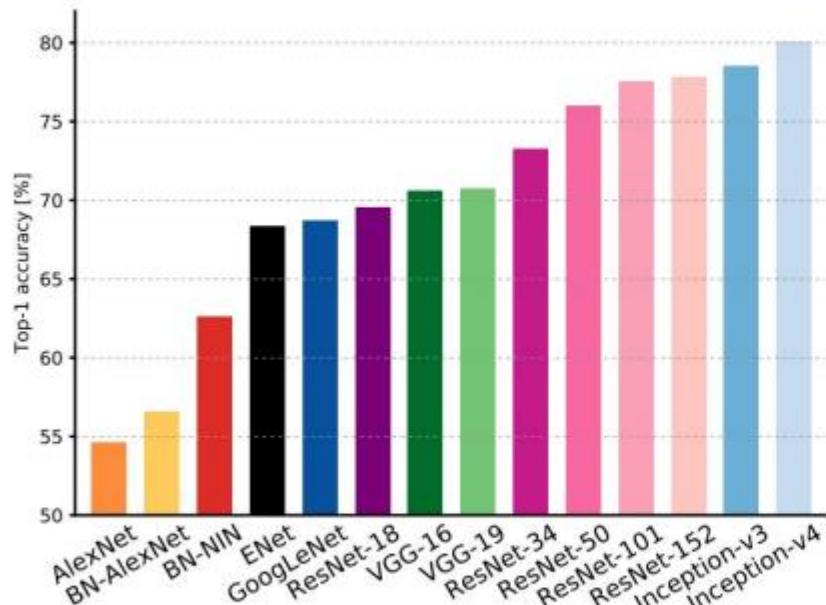
GoogLeNet:
most efficient



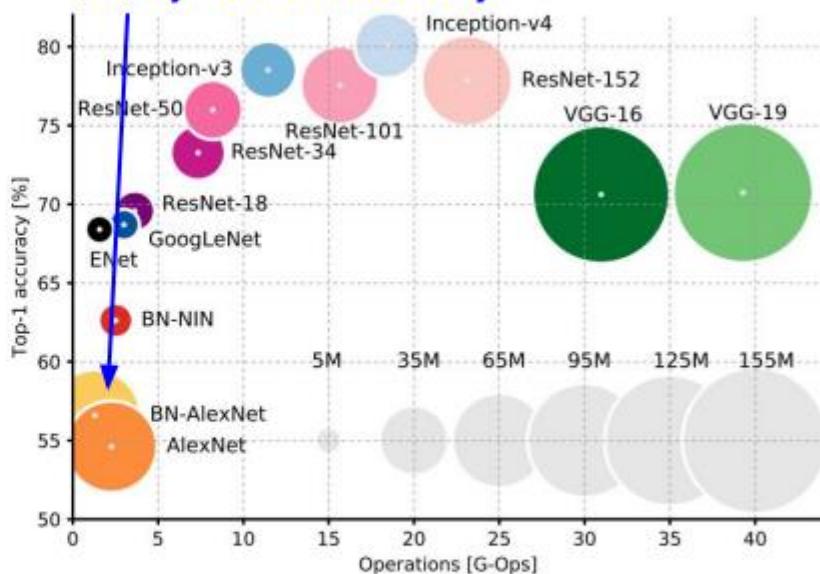
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Comparing complexity...



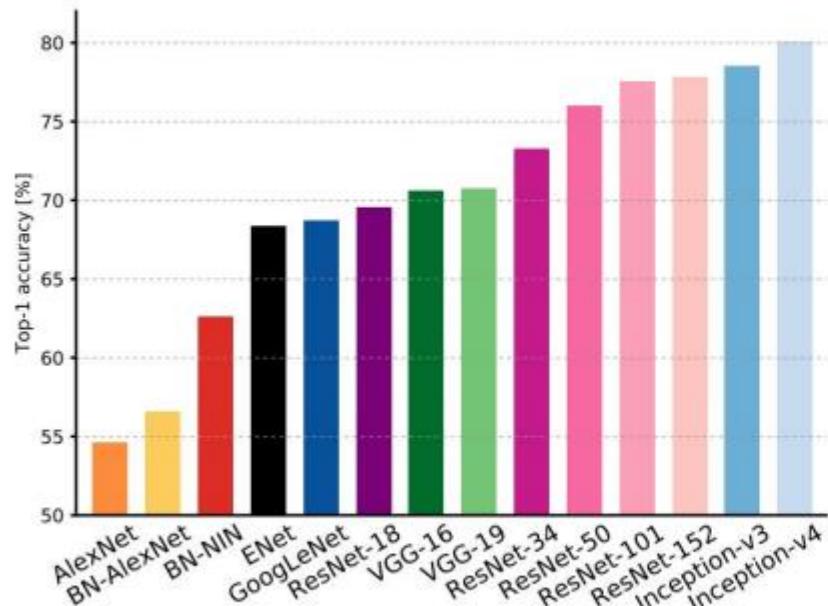
AlexNet:
Smaller compute, still memory heavy, lower accuracy



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

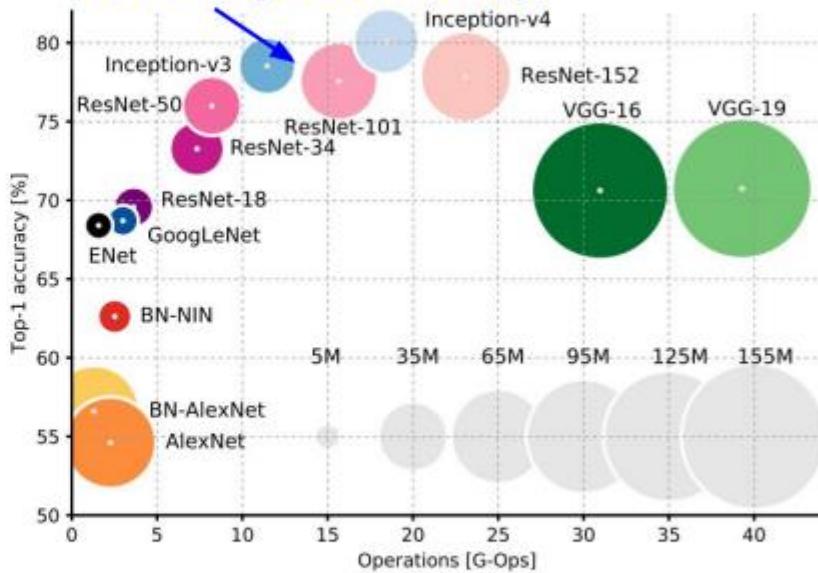
Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Comparing complexity...



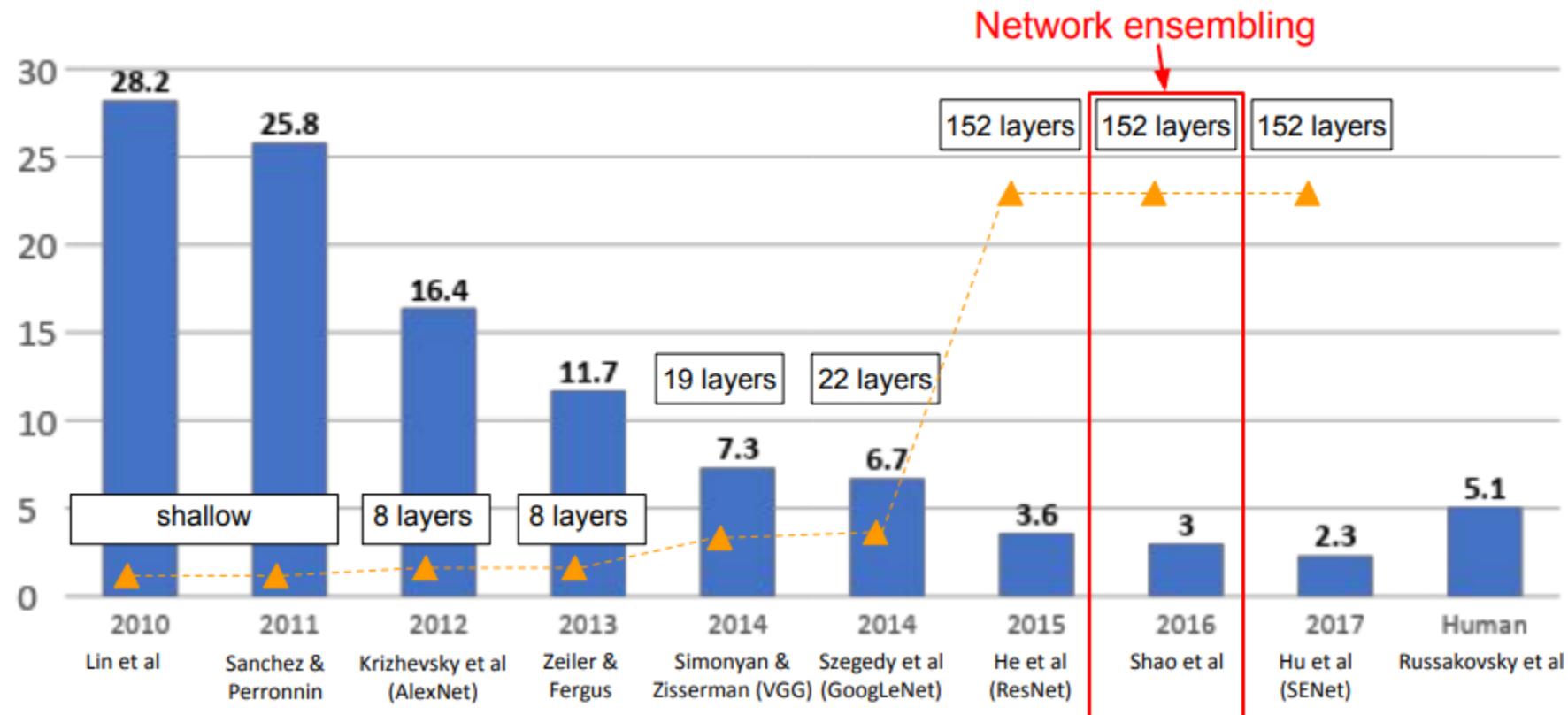
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

ResNet:
Moderate efficiency depending on
model, highest accuracy

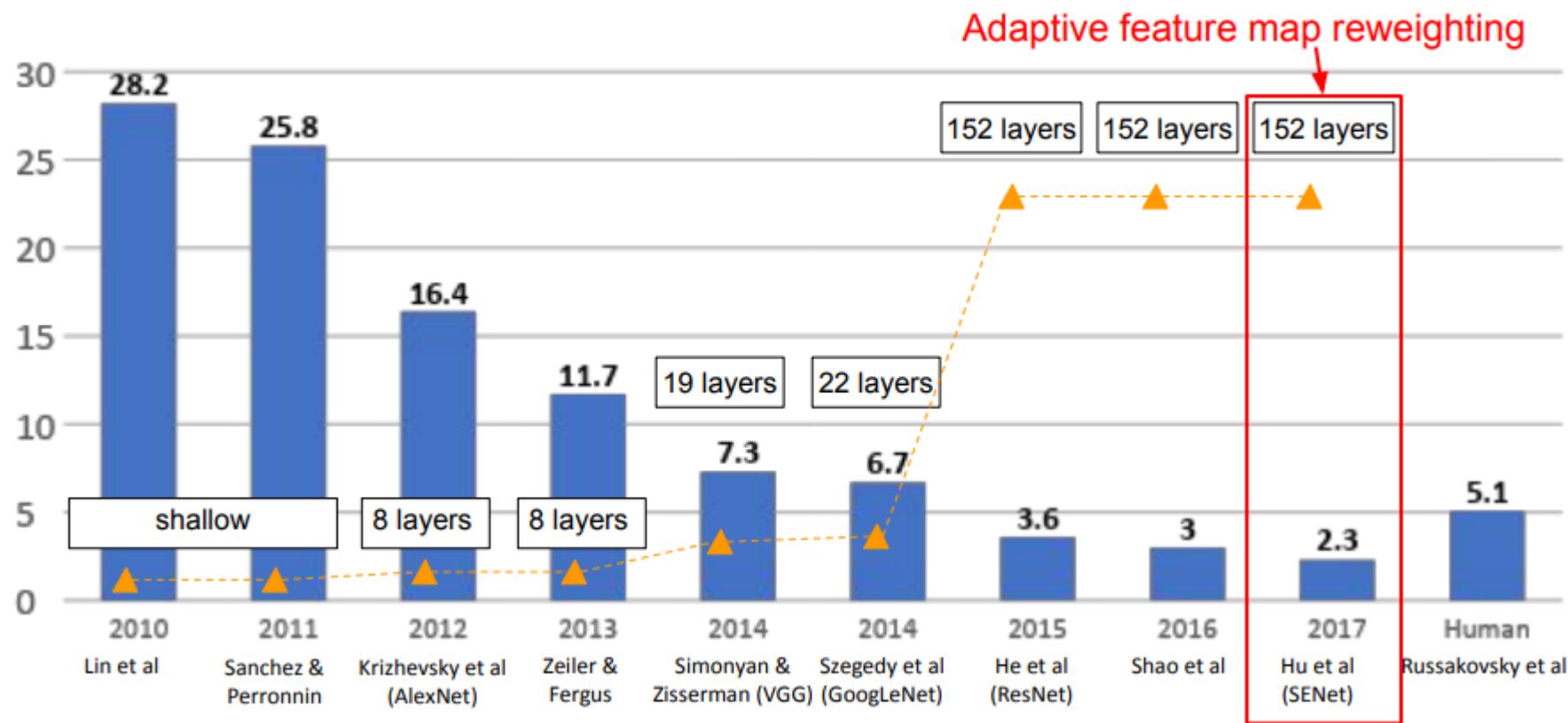


Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Kaggle

<https://www.kaggle.com/>

The screenshot shows the 'Competitions' section of the Kaggle website. At the top, there's a search bar and links for 'Sign In' and 'Register'. Below the header, there's a 'New to Kaggle? Start here!' section featuring the 'Titanic: Machine Learning from Disaster' competition. This section includes a thumbnail, a brief description, and a 'Knowledge' button. The main area displays a list of 'All Competitions' under the 'Active' tab. Two competitions are listed: 'OSIC Pulmonary Fibrosis Progression' (with a \$55,000 prize) and 'SIIM-ISIC Melanoma Classification' (with a \$30,000 prize). Both entries show their respective logos, descriptions, and team counts. A footer at the bottom of the page contains a cookie consent message.

The screenshot shows the 'Datasets' section of the Kaggle website. At the top, there's a search bar and links for 'Sign In' and 'Register'. Below the header, there's a 'Datasets' section with a 'New Dataset' button. The main area displays a section titled 'Engage With Dataset Tasks' with a cartoon illustration of a person working on a laptop. Below this, there's a search bar and a list of datasets under the 'PUBLIC' category. One dataset, 'COVID-19 Open Research Dataset Challenge (CORD-19)', is highlighted with a thumbnail, description, and statistics (7741 tasks). A footer at the bottom of the page contains a cookie consent message.

Kaggle

<https://www.kaggle.com/>

The screenshot shows the Kaggle interface for the COVID-19 Open Research Dataset Challenge (CORD-19). The main header displays the challenge name and a banner for AI2, CZI, MSR, Georgetown, NIH & The White House. Below the banner, there are tabs for Data, Tasks (17), Kernels (1,574), Discussion (354), Activity, Download (13 GB), and New Notebook. The Data tab is selected. On the left sidebar, there are icons for Dataset, Task, Kernel, Discussion, and Activity. The main content area includes sections for Usability (8.8), License (Other specified in description), and Tags (business, natural and physical sciences, computer science, health, biology and 3 more). A detailed description of the dataset is provided, stating it's a resource of over 181,000 scholarly articles related to COVID-19, SARS-CoV-2, and other coronaviruses. At the bottom, a cookie consent message from Kaggle is displayed.

The screenshot shows the Kaggle kernels page for the same challenge. The main header and challenge details are identical. The Kernels tab is selected, showing a list of notebooks. The first four notebooks listed are:

- Covid-19 - Before and After Quarantine (95 upvotes, 5h ago, beginner, data visualization, covid19)
- CORD-19 Analysis with Sentence Embeddings (199 upvotes, 21h ago, Task Submission, eda, nlp, covid19)
- COVID-19 Case Study - Analysis, Viz & Comparisons (1230 upvotes, 20h ago, epidemiology, health, data visualization, starter code, covid19)
- COVID-19 Literature Clustering (700 upvotes, 14h ago, Py, covid19)

Below the notebook list, a cookie consent message from Kaggle is displayed.

이미지 분류 모델(ResNet) 사용 예

Module: tf.keras.applications

Modules

`densenet` module: DenseNet models for Keras.

`imagenet_utils` module: Utilities for ImageNet data preprocessing & prediction decoding.

`inception_resnet_v2` module: Inception-ResNet V2 model for Keras.

`inception_v3` module: Inception V3 model for Keras.

`mobilenet` module: MobileNet v1 models for Keras.

`mobilenet_v2` module: MobileNet v2 models for Keras.

`nasnet` module: NASNet-A models for Keras.

`resnet` module: ResNet models for Keras.

`resnet50` module: Public API for tf.keras.applications.resnet50 namespace.

`resnet_v2` module: ResNet v2 models for Keras.

`vgg16` module: VGG16 model for Keras.

`vgg19` module: VGG19 model for Keras.

`xception` module: Xception V1 model for Keras.

https://www.tensorflow.org/api_docs/python/tf/keras/applications

이미지 분류 모델(ResNet) 사용 예

Module: `tf.keras.applications.imagenet_utils`



TensorFlow 1 version

Utilities for ImageNet data preprocessing & prediction decoding.

Functions

`decode_predictions(...)` : Decodes the prediction of an ImageNet model.

`preprocess_input(...)` : Preprocesses a tensor or Numpy array encoding a batch of images.

이미지 분류 모델(ResNet) 사용 예

ResNet50을 사용한 ImageNet 클래스 분류

```
from keras.applications.resnet50 import ResNet50
from keras.preprocessing import image
from keras.applications.resnet50 import preprocess_input, decode_predictions
import numpy as np
import matplotlib.pyplot as plt

model = ResNet50(weights='imagenet')
#model.summary()

img_path = 'data/cat.jpg'
img = image.load_img(img_path, target_size=(224, 224))
plt.imshow(img)

x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model.predict(x)
# 결과를 튜플의 리스트(클래스, 설명, 확률)로 디코딩
# (배치 내 각 샘플 당 하나의 리스트)
print('Predicted:', decode_predictions(preds, top=3)[0])
```

Predicted: [(['n02123045', 'tabby', 0.7820335), ('n02124075', 'Egyptian_cat', 0.1186032), ('n02123159', 'tiger_cat', 0.08334199)]

