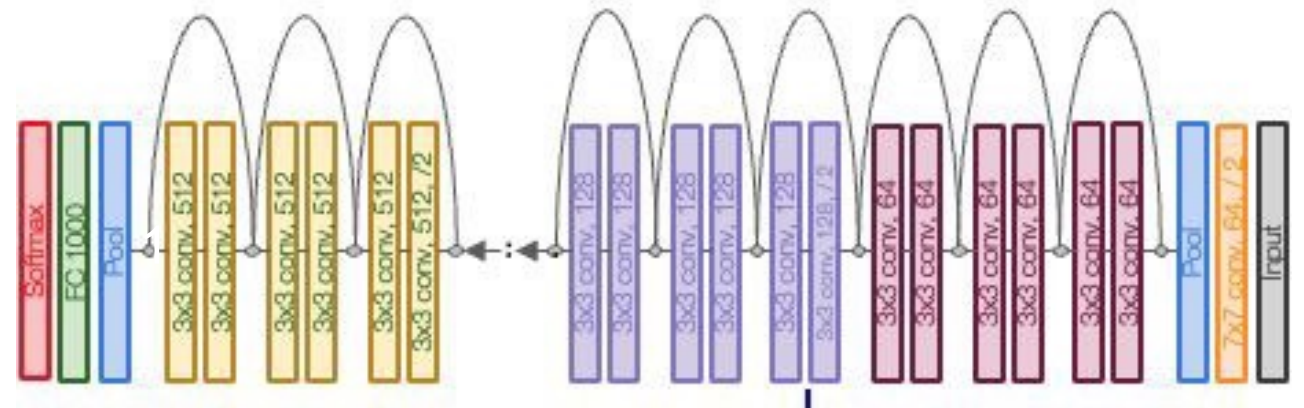
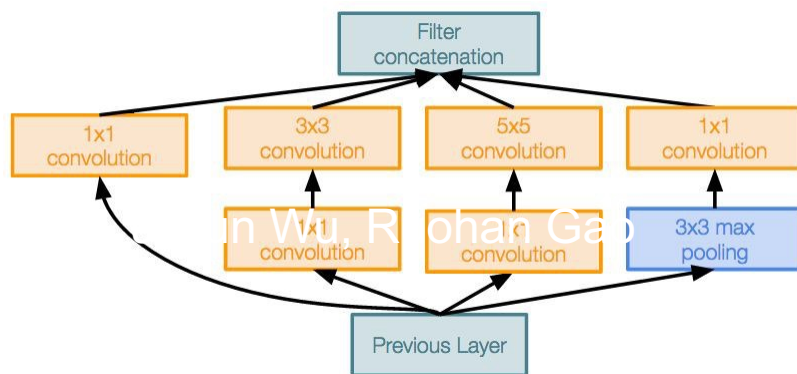
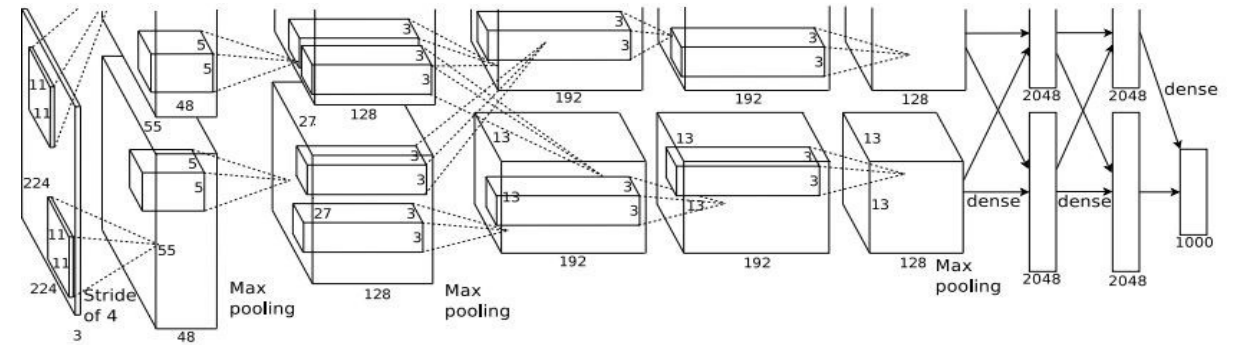
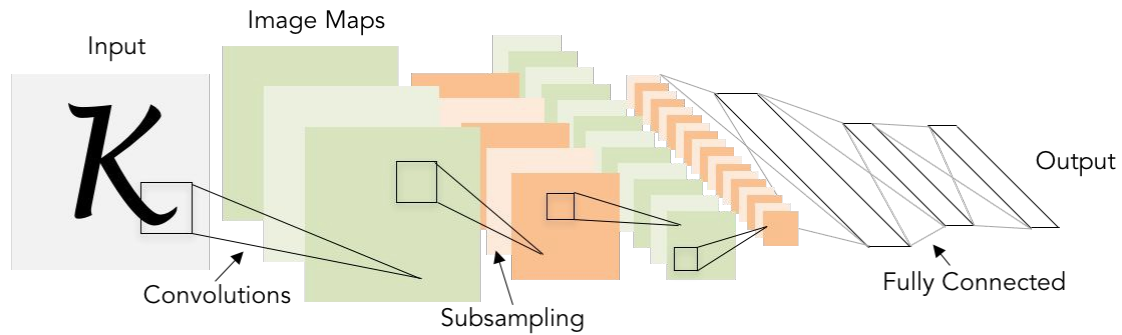
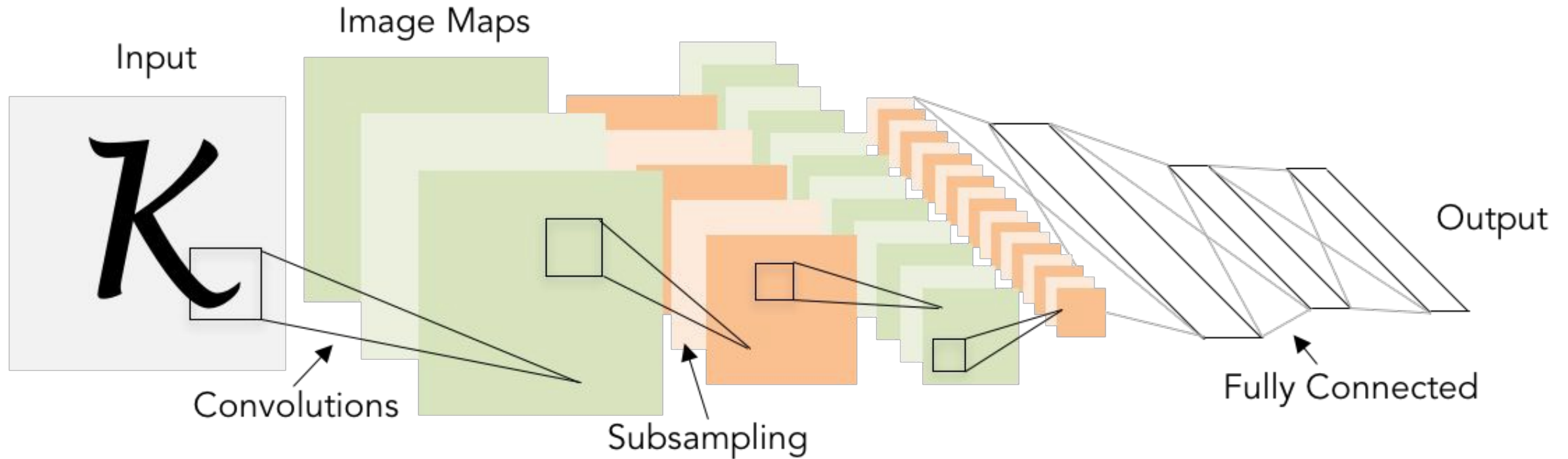


Today: CNN Architectures



Review: LeNet-5

[LeCun et al., 1998]

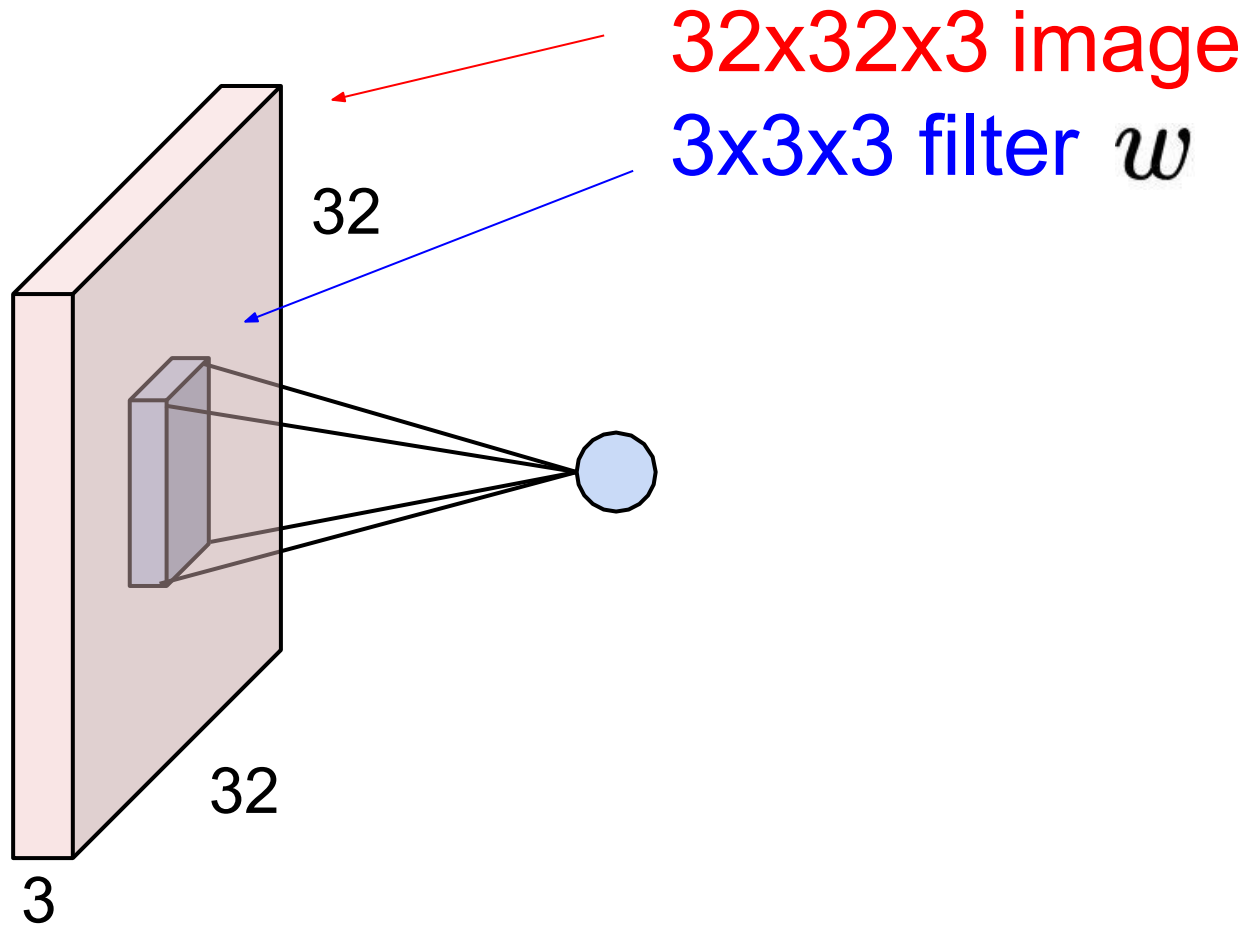


Conv filters were 5x5, applied at stride 1

Subsampling (Pooling) layers were 2x2 applied at stride 2

i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

Review: Convolution



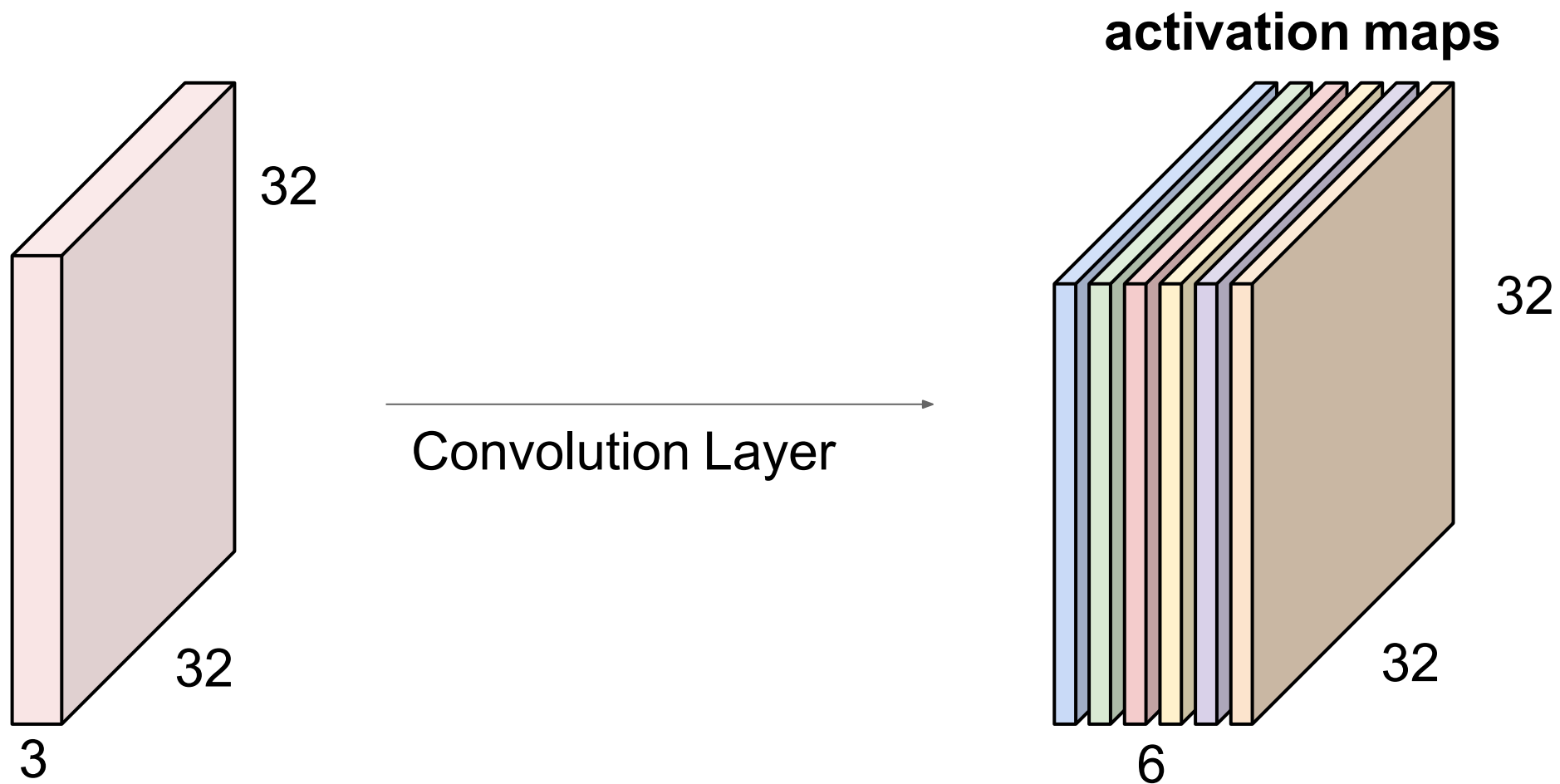
0	0	0	0	0	0				
0									
0									
0									
0									
0									

Stride:
Downsample
output activations

0	0	0	0	0	0				
0									
0									
0									
0									

Padding:
Preserve
input spatial
dimensions in
output activations

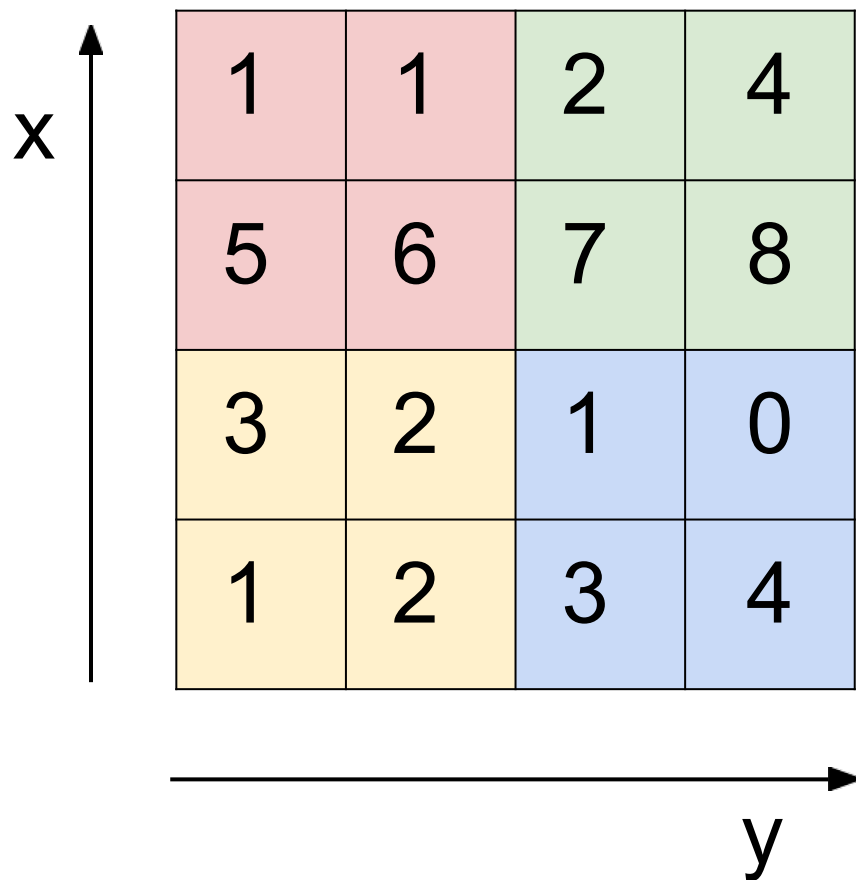
Review: Convolution



Each conv filter outputs a “slice” in the activation

Review: Pooling

Single depth slice



1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters
and stride 2



6	8
3	4

Today: CNN Architectures

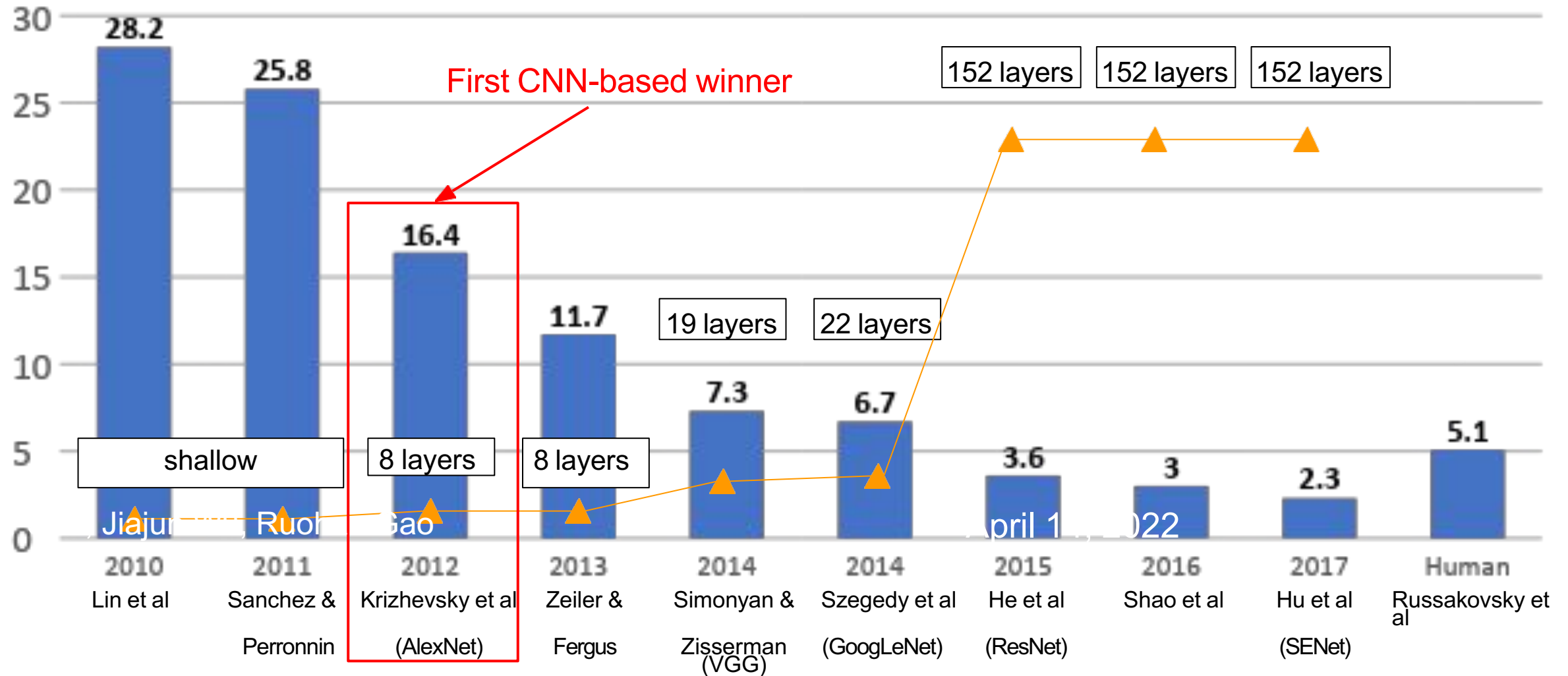
Case Studies

- AlexNet
- VGG
- GoogLeNet
- ResNet

Also....

- SENet
- Wide ResNet
- ResNeXT
- DenseNet
- MobileNets
- NASNet
- EfficientNet

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

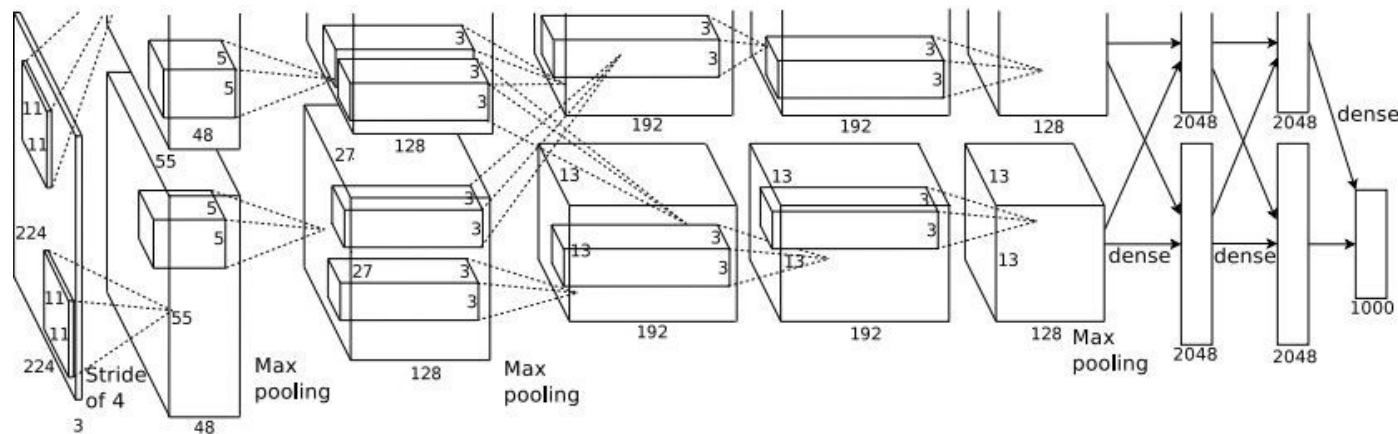
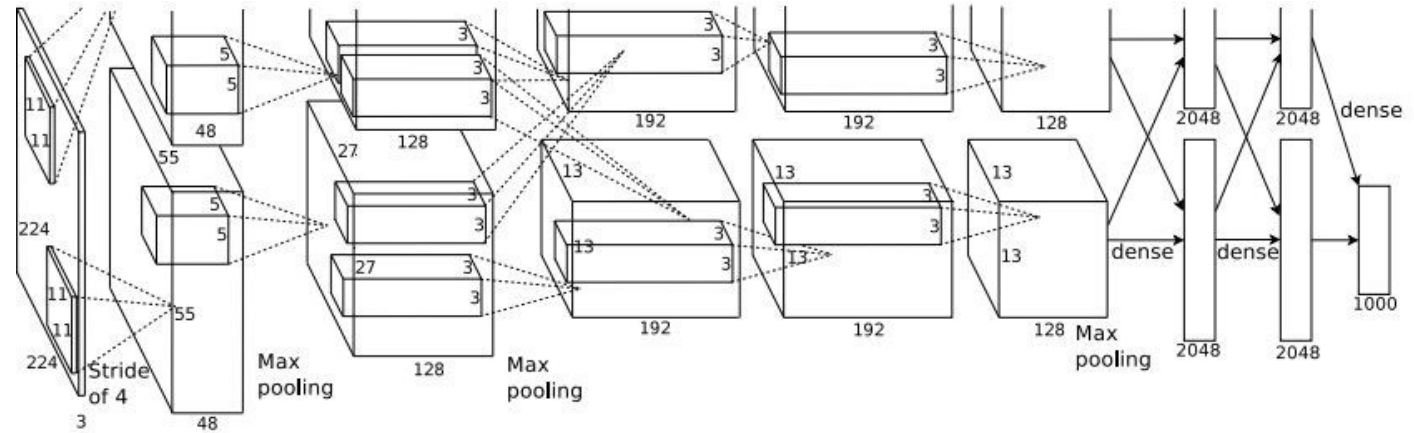


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

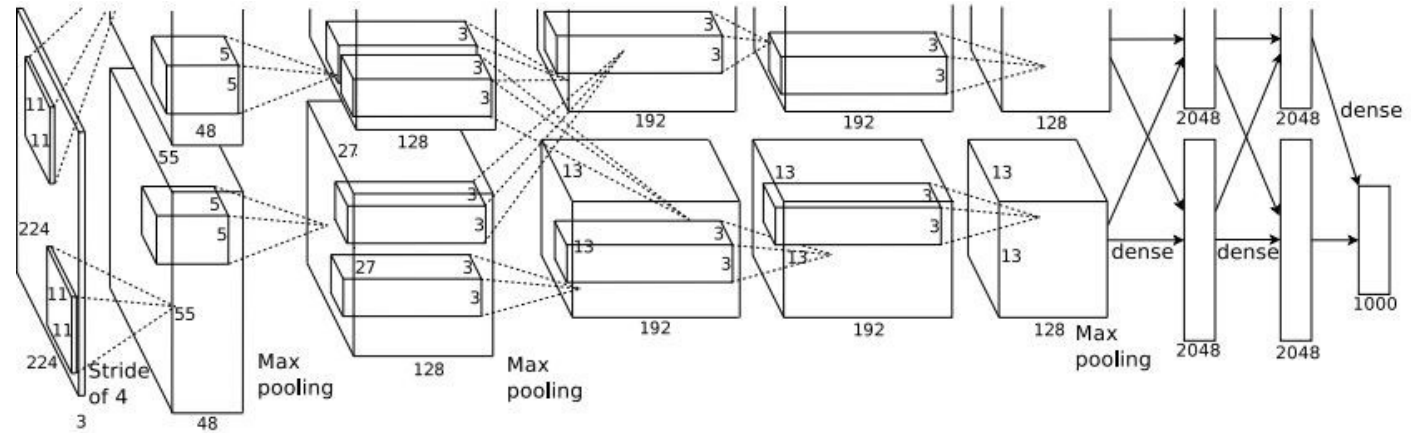
First layer (CONV1): 96 11x11 filters applied at stride 4

=>

Q: what is the output volume size?

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

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

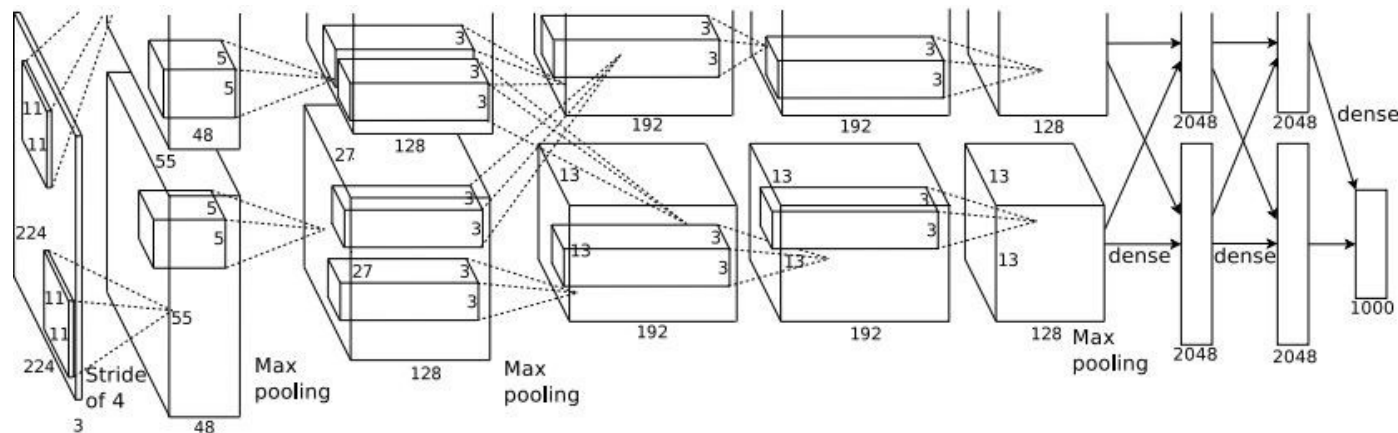
=>

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

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

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

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

=>

Output volume **[55x55x96]**

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

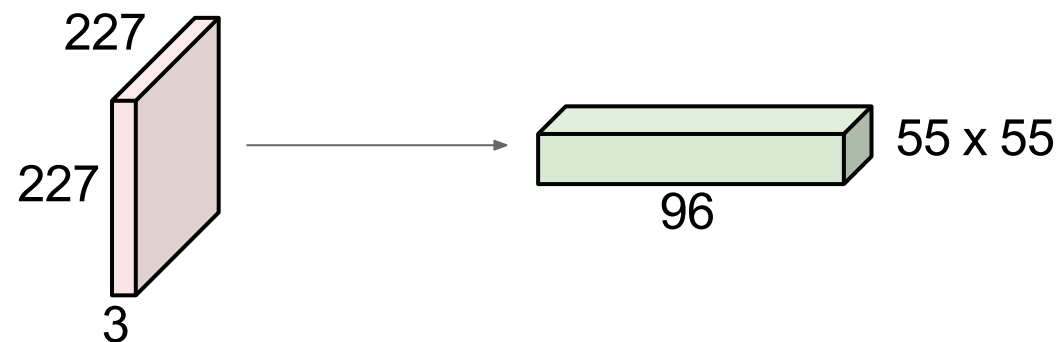
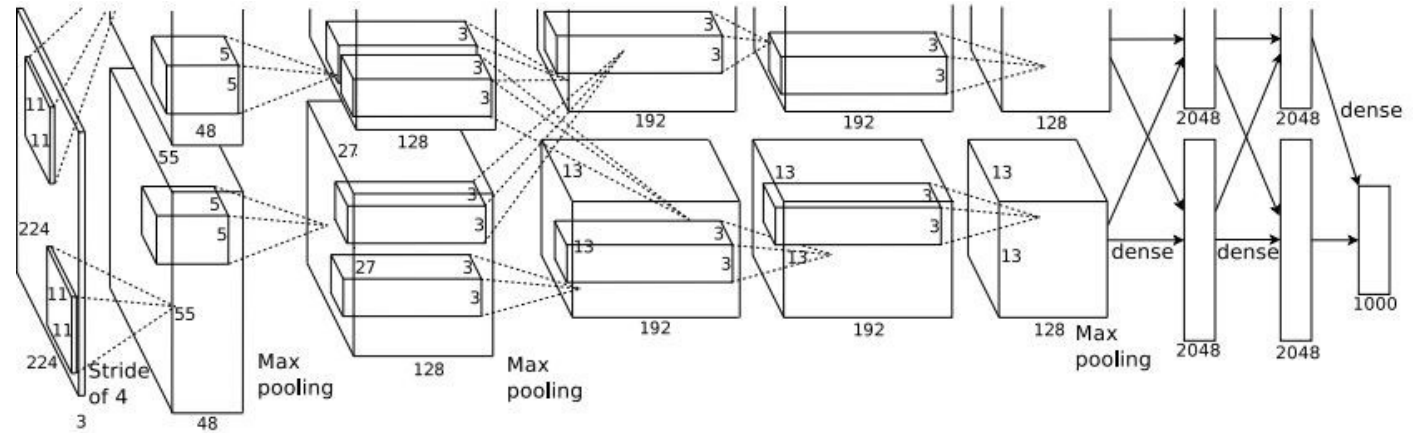


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Case Study: AlexNet

[Krizhevsky et al. 2012]



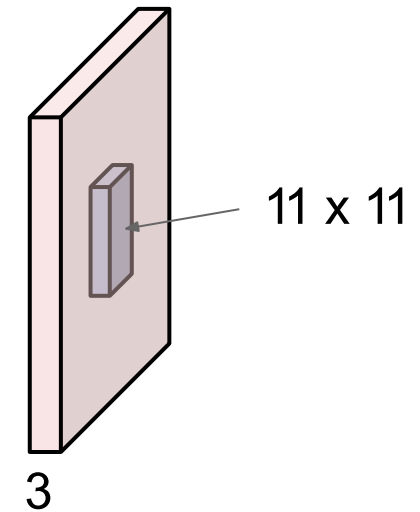
Input: 227x227x3 images

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

=>

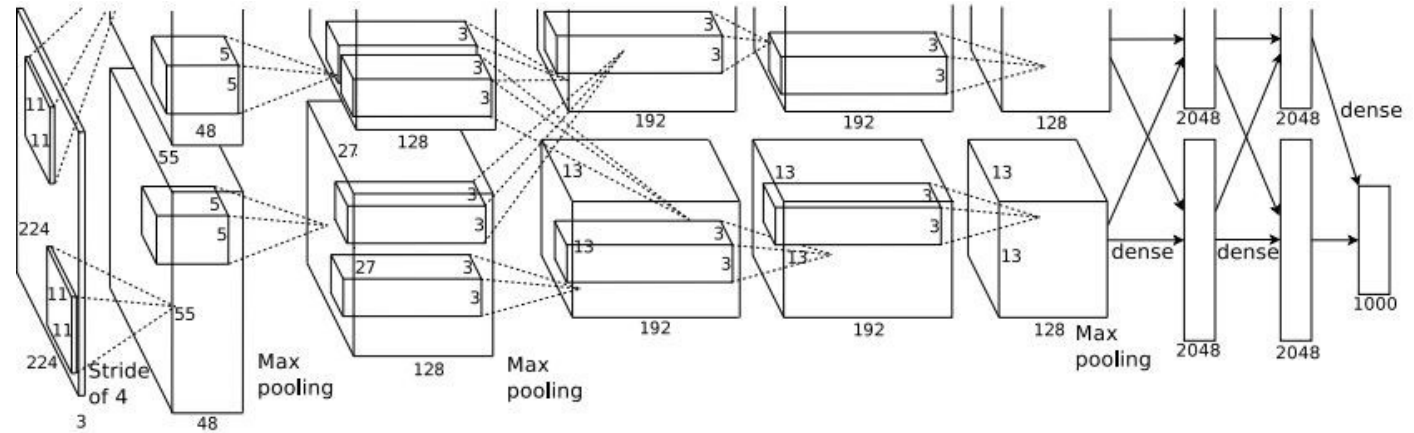
Output volume **[55x55x96]**

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



Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

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

=>

Output volume **[55x55x96]**

Parameters: $(11 \cdot 11 \cdot 3 + 1) \cdot 96 = \mathbf{35K}$

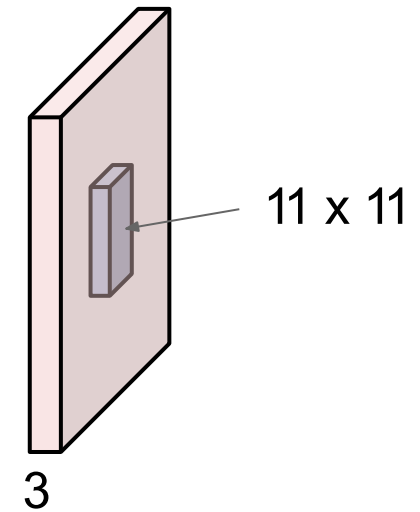
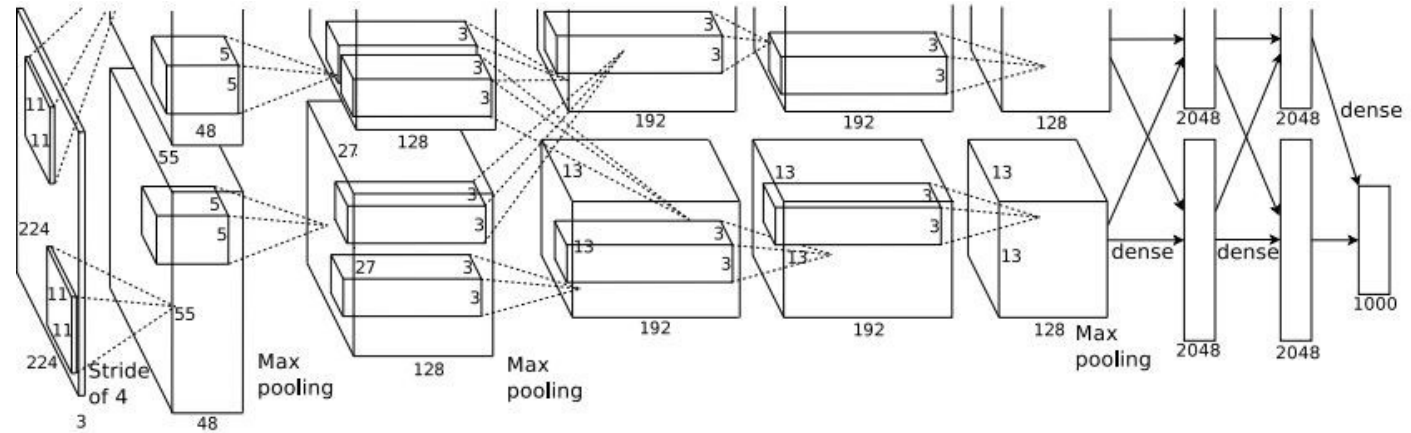


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

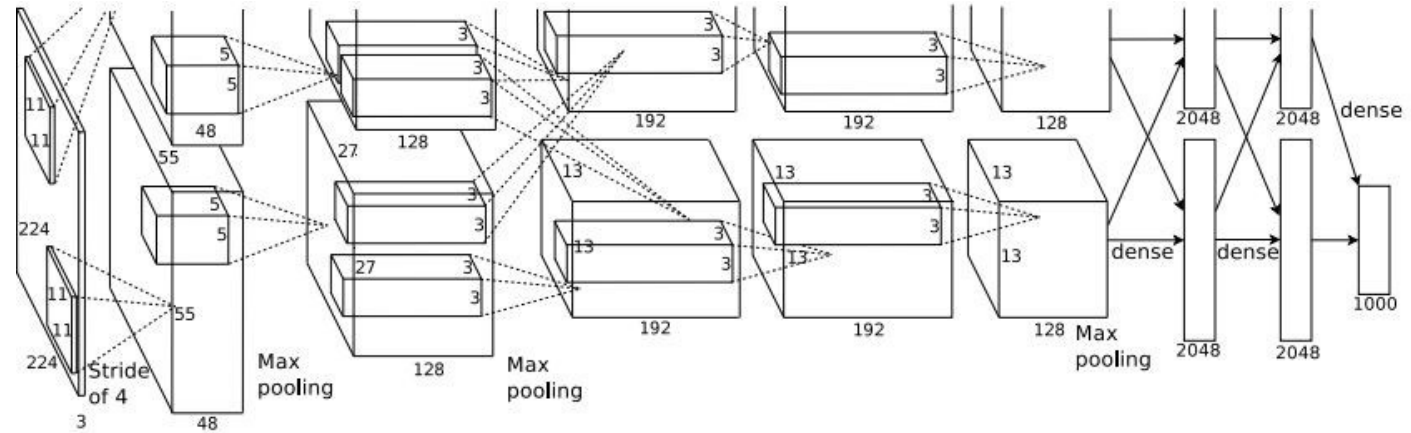
After CONV1: 55x55x96

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

Q: what is the output volume size?

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

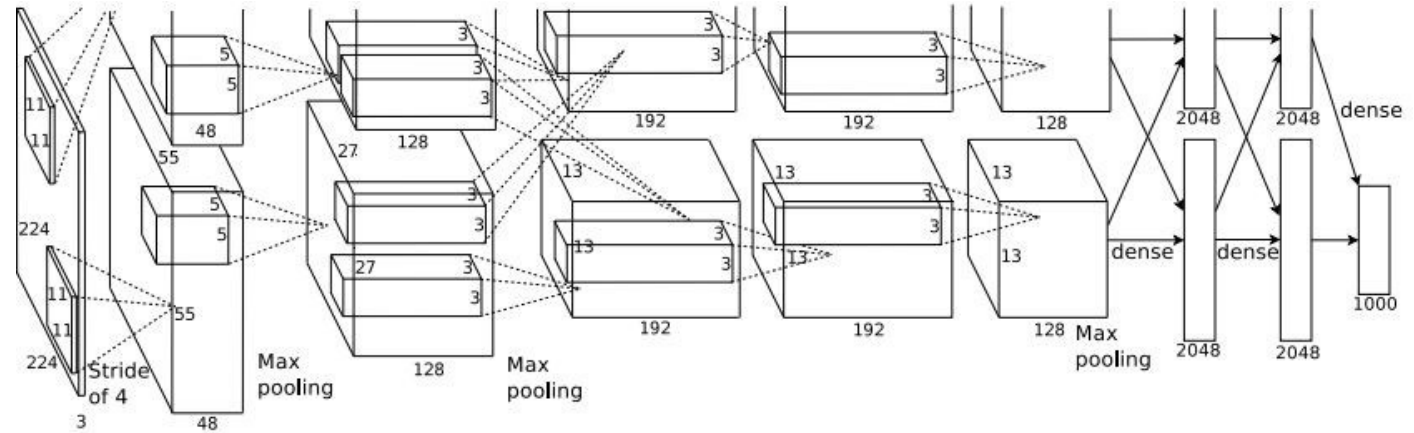
After CONV1: 55x55x96

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

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

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

[Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

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

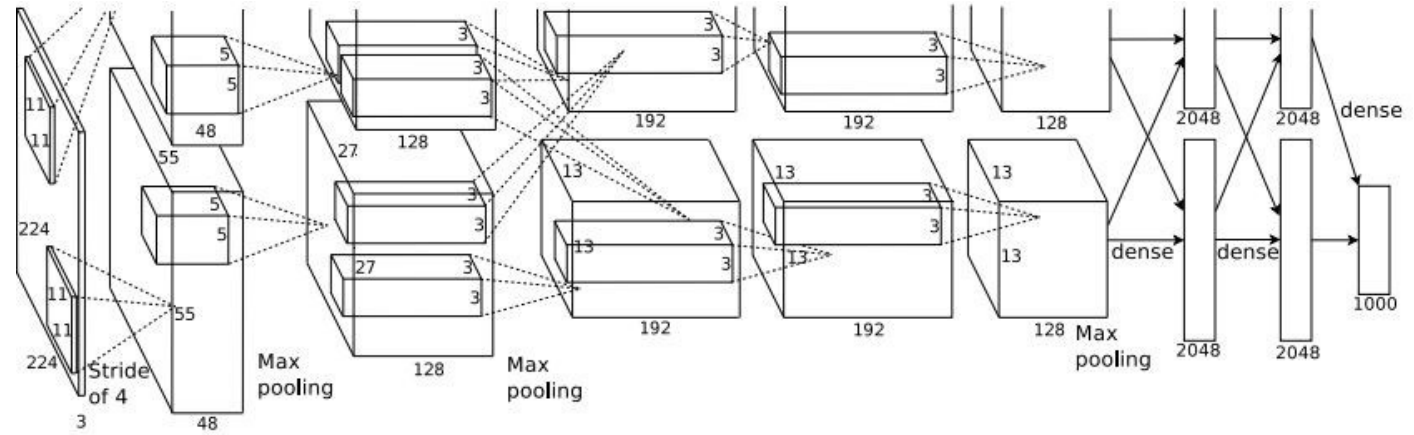
Second layer (POOL1): 3x3 filters applied at stride 2

Output volume: 27x27x96

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

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!

Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images

After CONV1: 55x55x96

After POOL1: 27x27x96

...

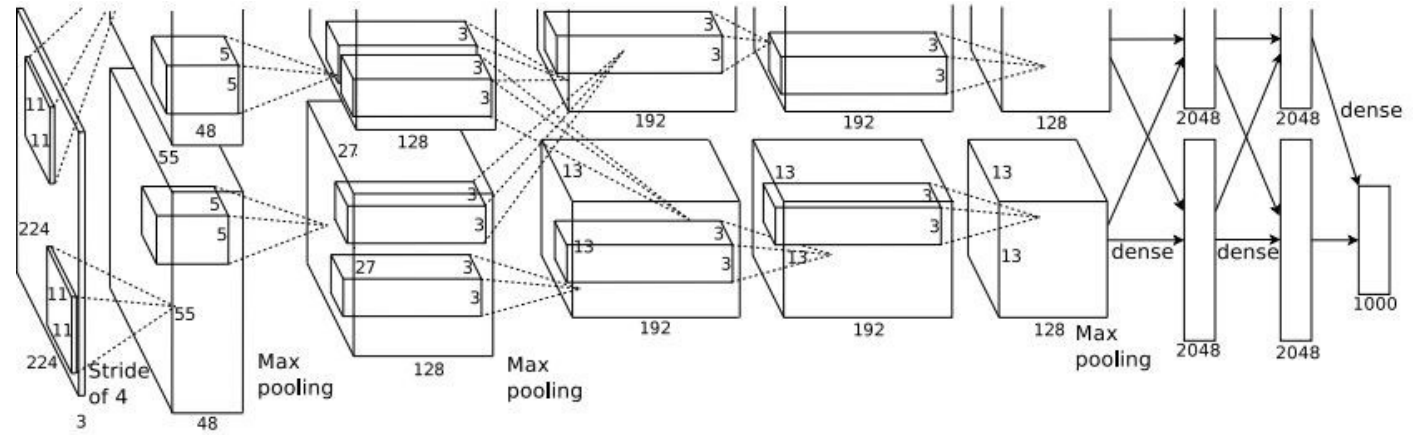


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

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)

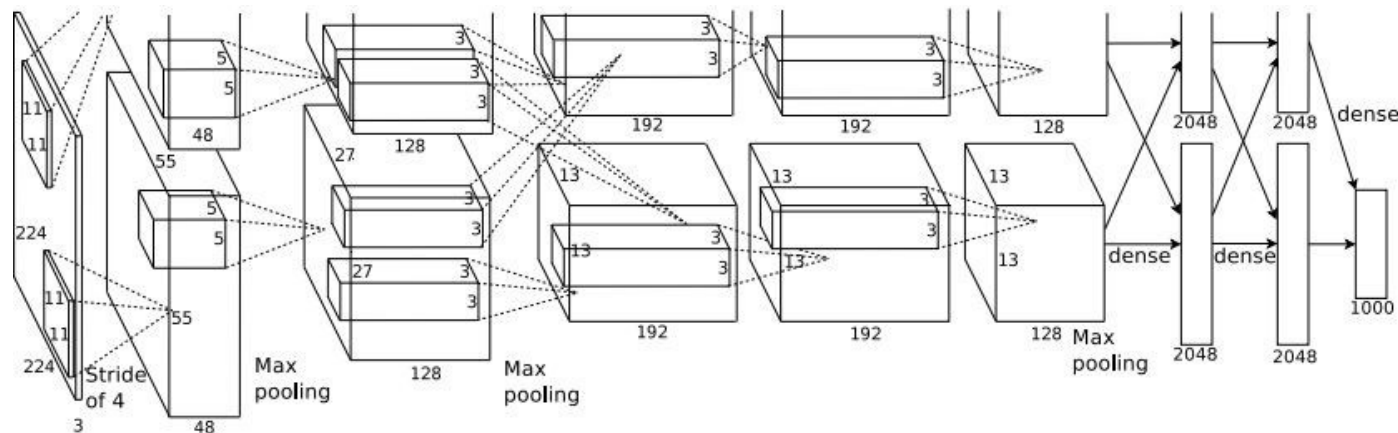


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

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

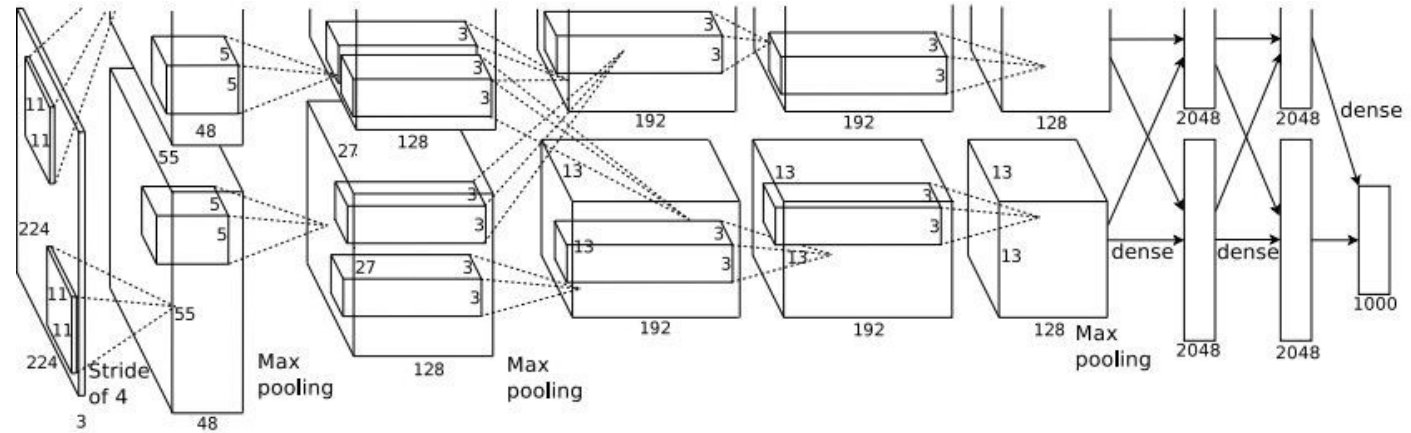
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

[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 LRN 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

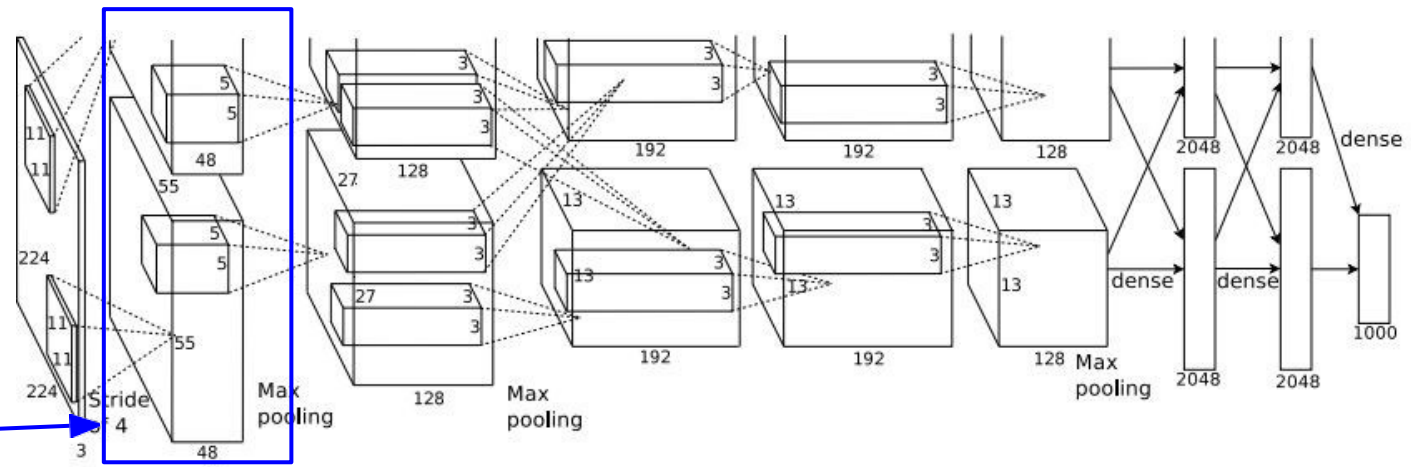
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

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

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)



[55x55x48] x 2

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

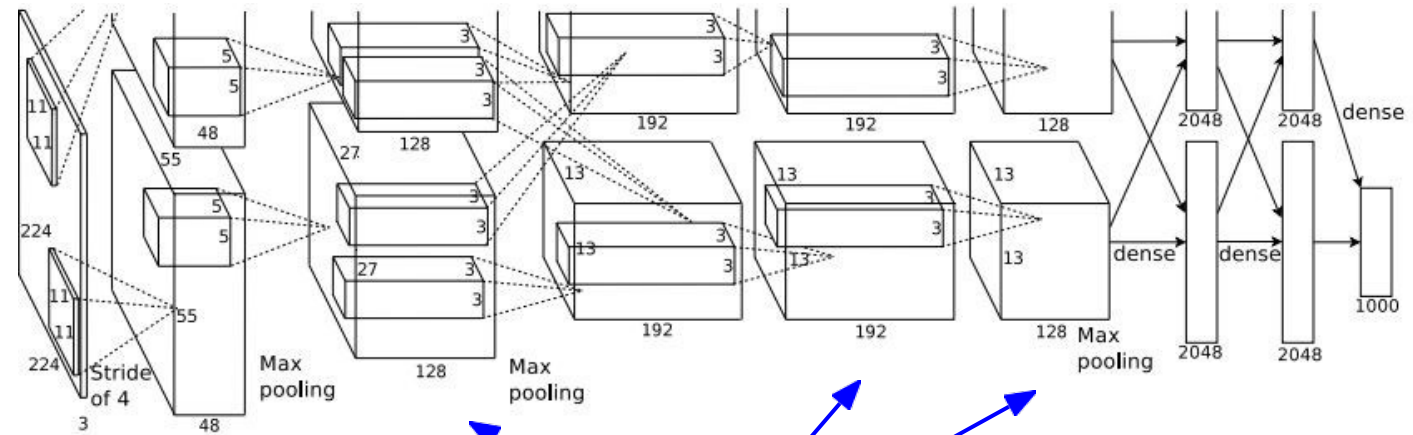
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

[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

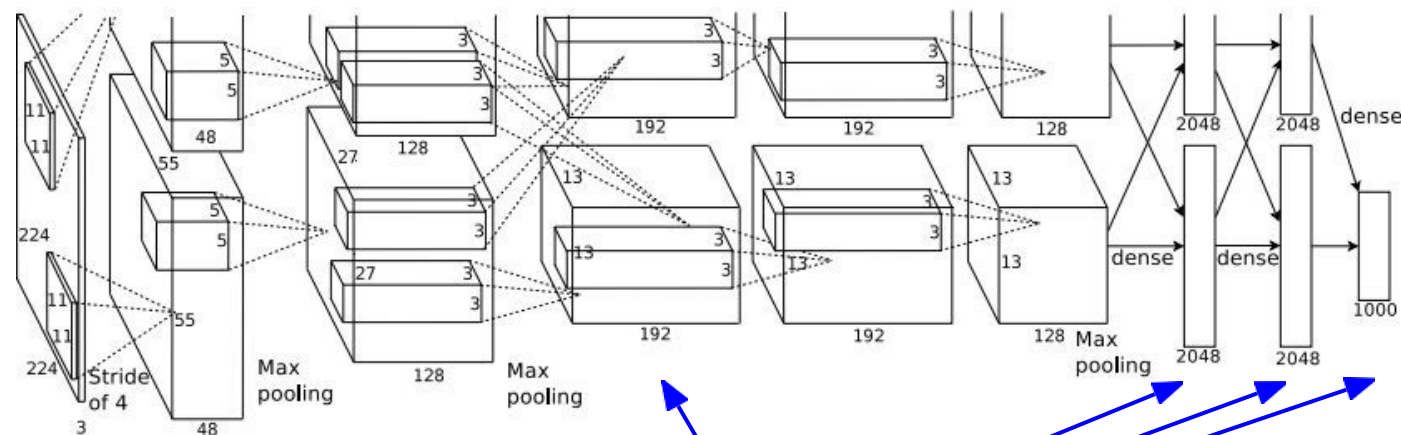
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

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

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)

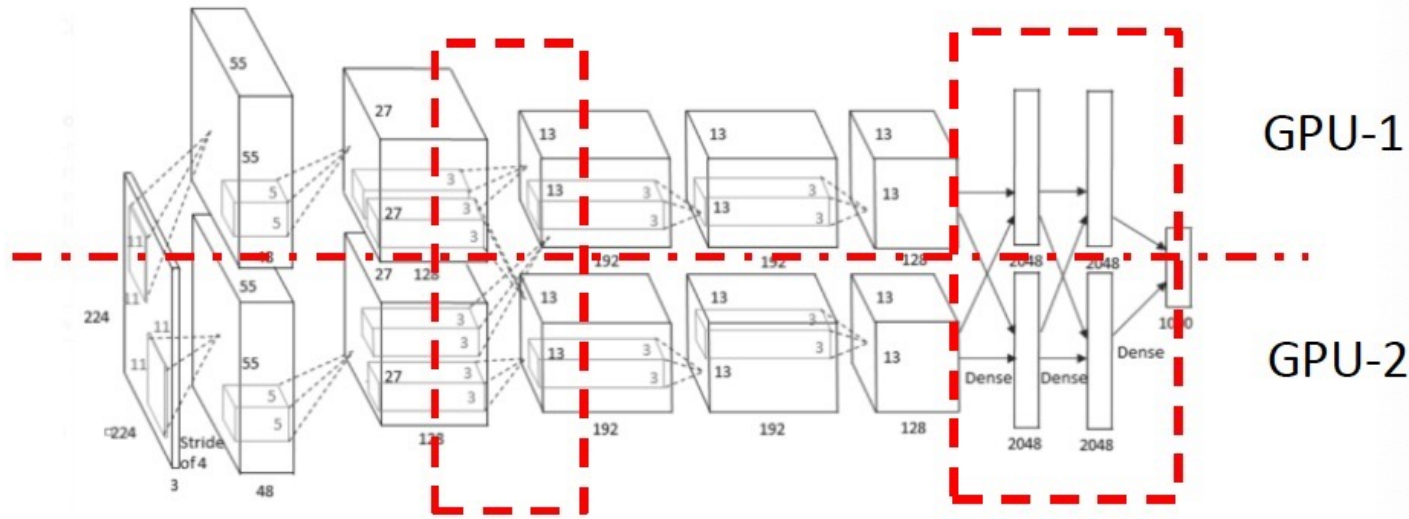


CONV3, FC6, FC7, FC8:
Connections with all feature maps in preceding layer, communication across GPUs

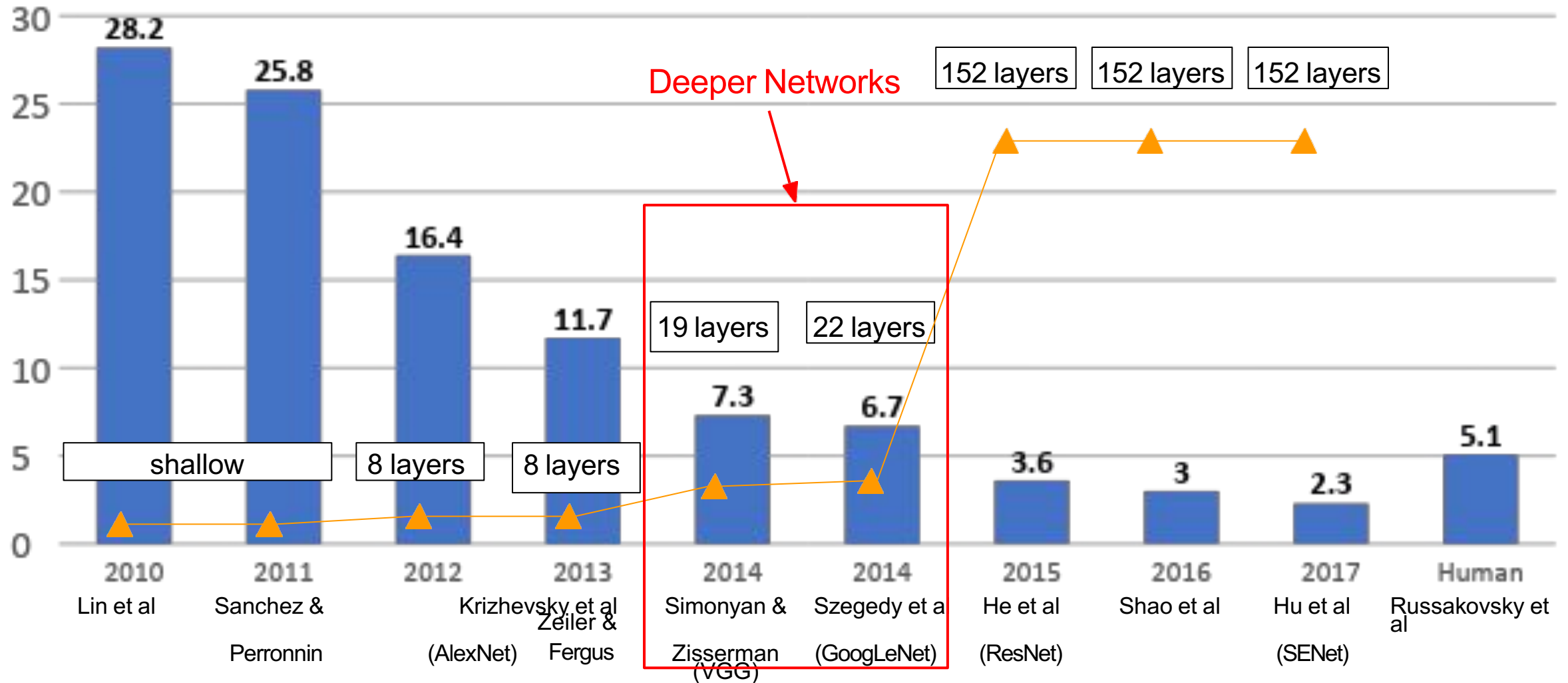
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Case Study: AlexNet

[Krizhevsky et al. 2012]



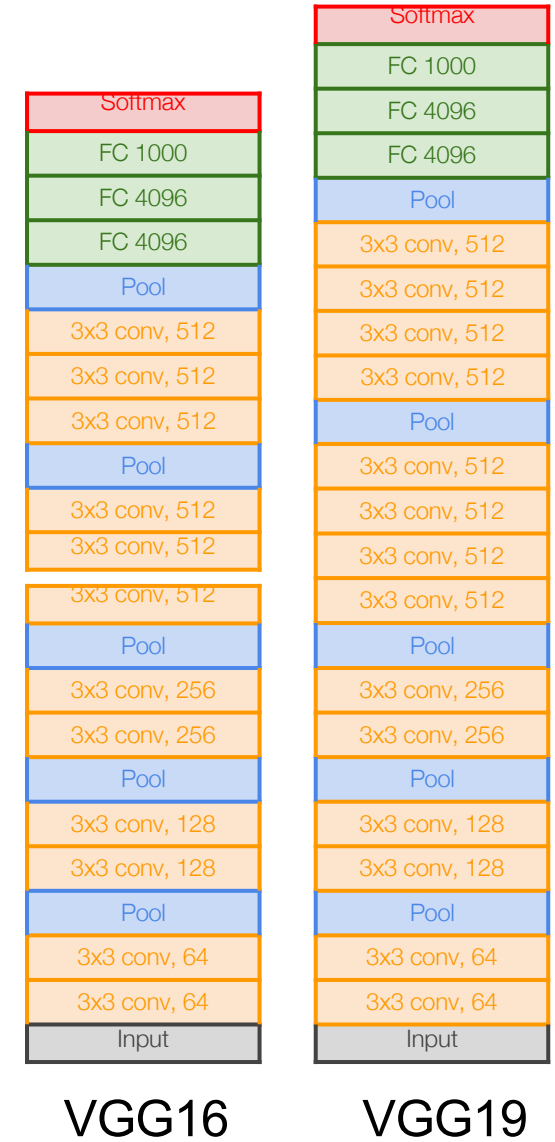
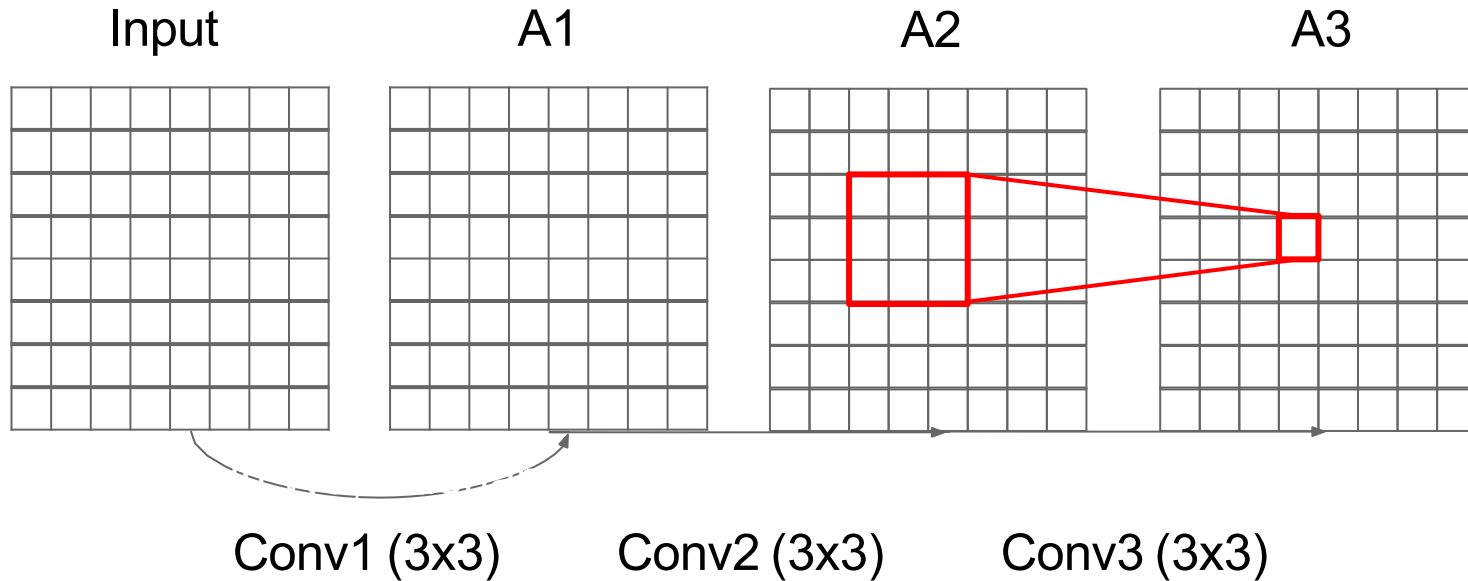
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

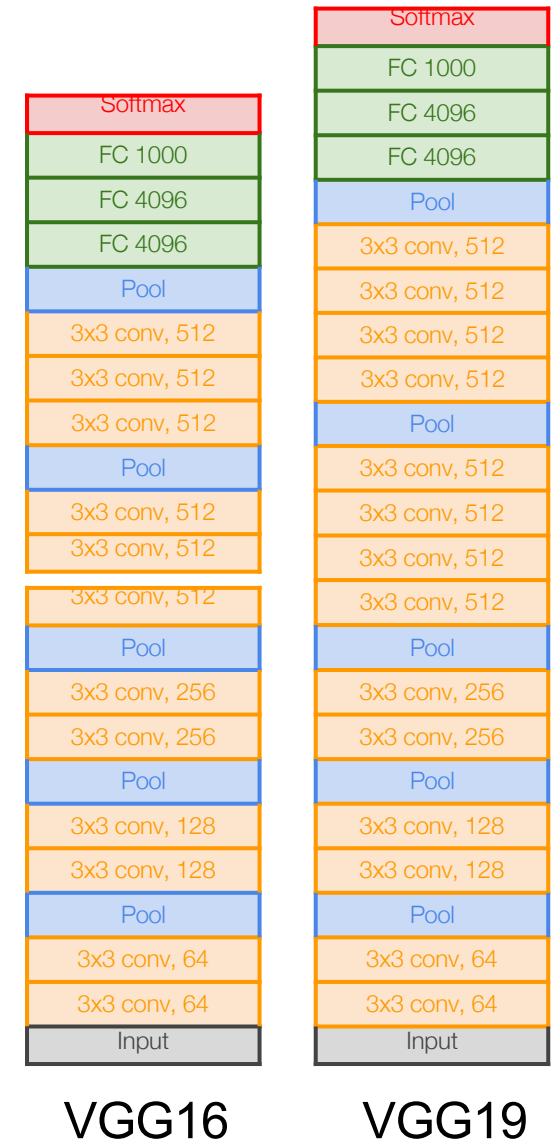
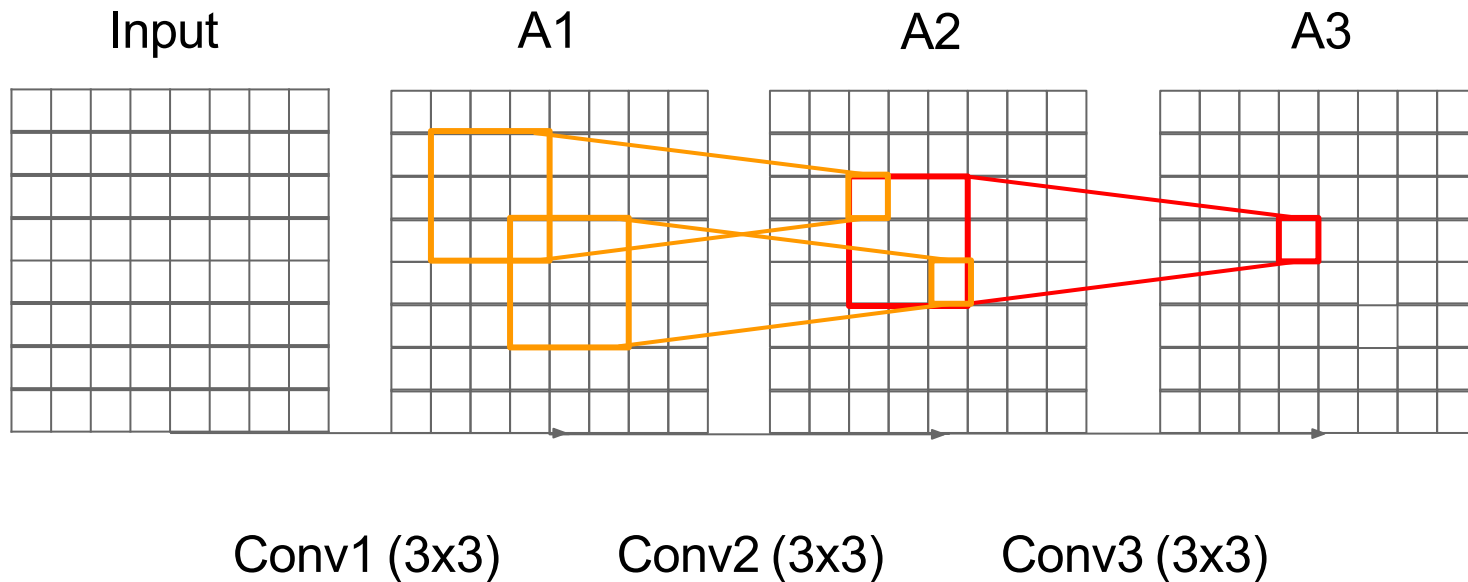
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?

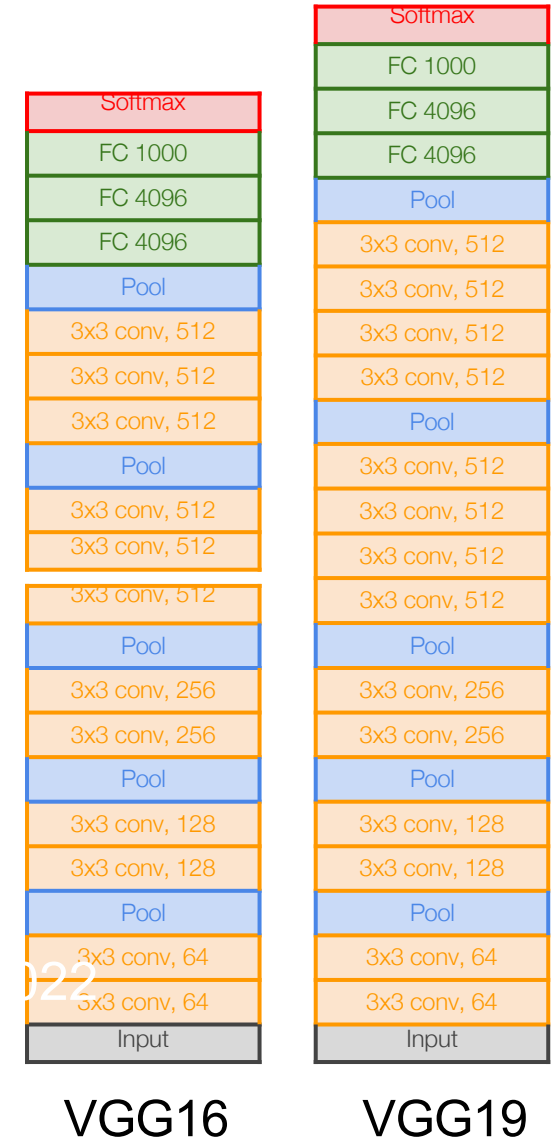


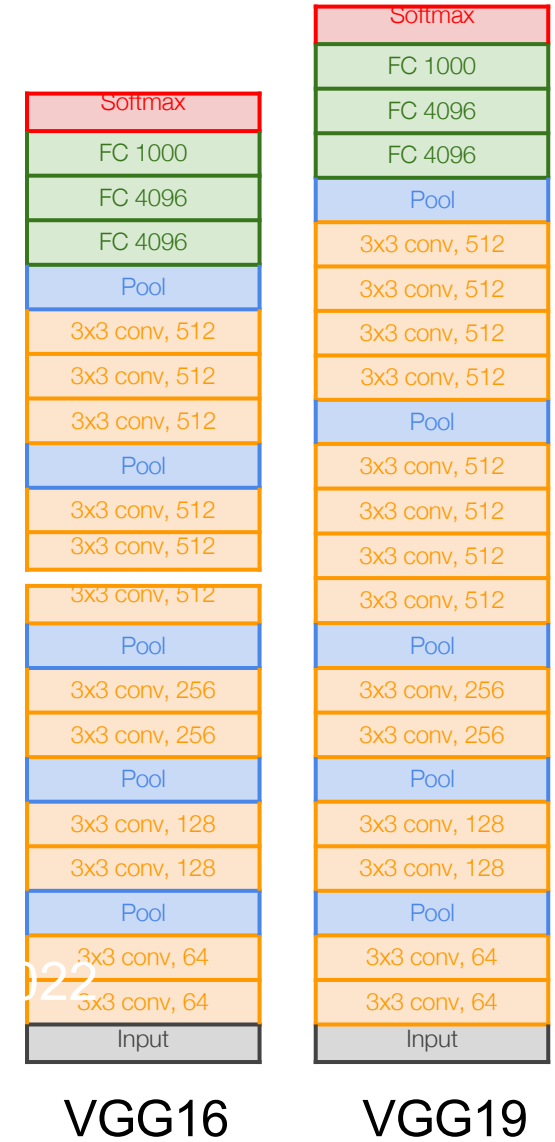
Case Study: VGGNet

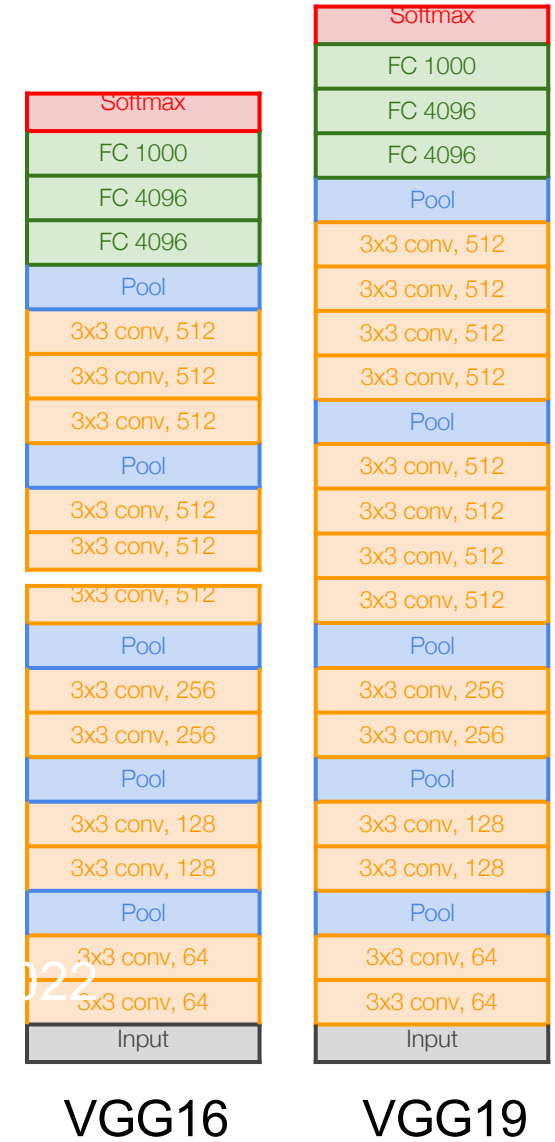
[Simonyan and Zisserman, 2014]

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









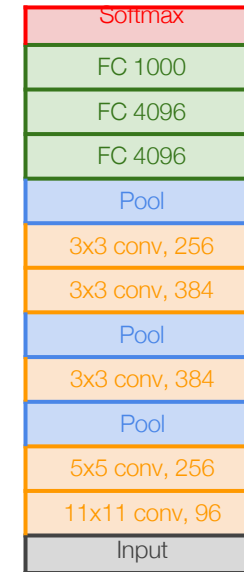
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

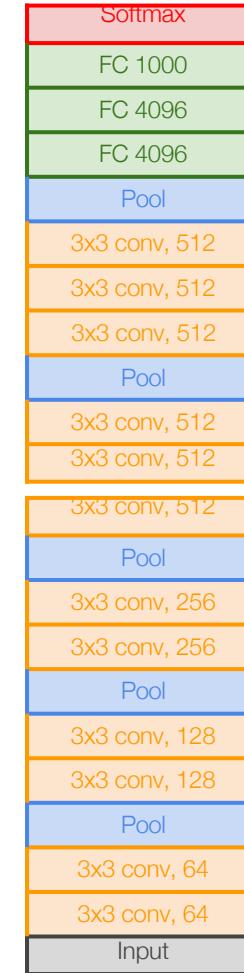
Q: Why use smaller filters? (3x3 conv)

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

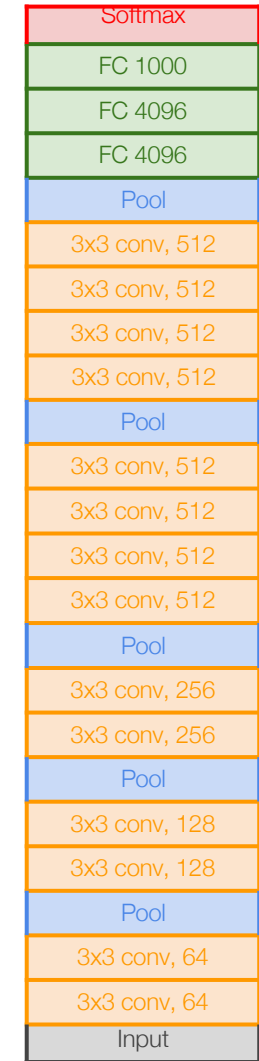
[7x7]



AlexNet



VGG16



VGG19

Case Study: VGGNet

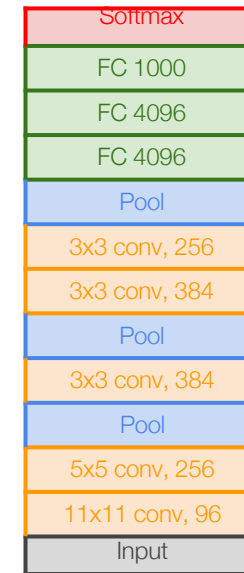
[Simonyan and Zisserman, 2014]

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

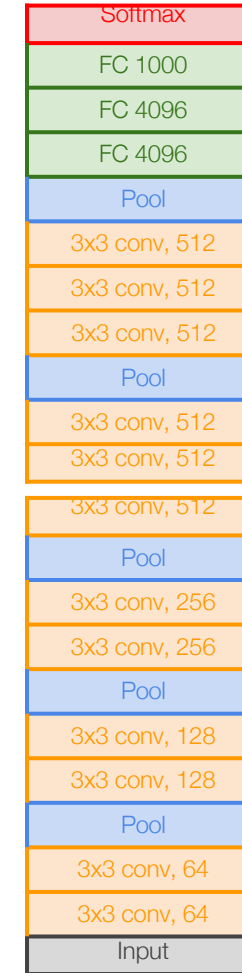
Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

But deeper, more non-linearities

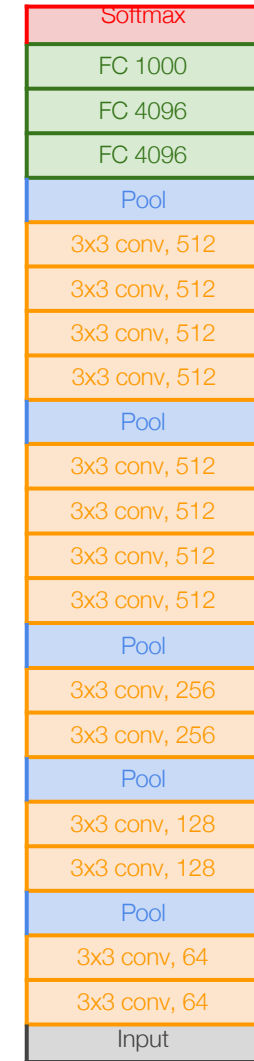
And fewer parameters: $3 * (3^2 C^2)$ vs. $7^2 C^2$ for C channels per layer



AlexNet

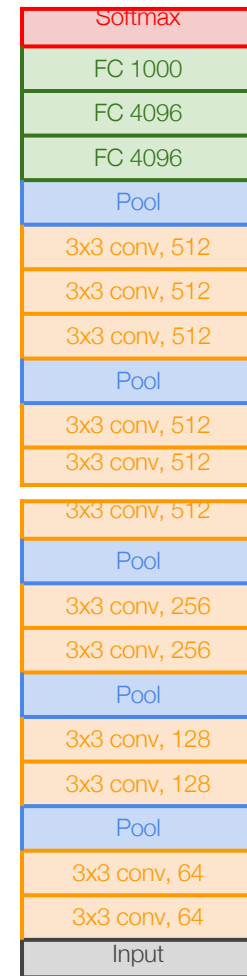


VGG16



VGG19

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

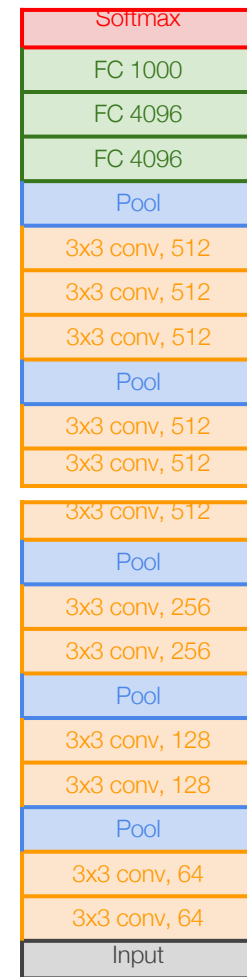


VGG16

INPUT: [224x224x3] memory: $224*224*3=150\text{K}$ params: 0 (not counting biases)
 CONV3-64: [224x224x64] memory: $224*224*64=3.2\text{M}$ params: $(3*3*3)*64 = 1,728$
 CONV3-64: [224x224x64] memory: $224*224*64=3.2\text{M}$ params: $(3*3*64)*64 = 36,864$
 POOL2: [112x112x64] memory: $112*112*64=800\text{K}$ params: 0
 CONV3-128: [112x112x128] memory: $112*112*128=1.6\text{M}$ params: $(3*3*64)*128 = 73,728$
 CONV3-128: [112x112x128] memory: $112*112*128=1.6\text{M}$ params: $(3*3*128)*128 = 147,456$
 POOL2: [56x56x128] memory: $56*56*128=400\text{K}$ params: 0
 CONV3-256: [56x56x256] memory: $56*56*256=800\text{K}$ params: $(3*3*128)*256 = 294,912$
 CONV3-256: [56x56x256] memory: $56*56*256=800\text{K}$ params: $(3*3*256)*256 = 589,824$
 CONV3-256: [56x56x256] memory: $56*56*256=800\text{K}$ params: $(3*3*256)*256 = 589,824$
 POOL2: [28x28x256] memory: $28*28*256=200\text{K}$ params: 0
 CONV3-512: [28x28x512] memory: $28*28*512=400\text{K}$ params: $(3*3*256)*512 = 1,179,648$
 CONV3-512: [28x28x512] memory: $28*28*512=400\text{K}$ params: $(3*3*512)*512 = 2,359,296$
 CONV3-512: [28x28x512] memory: $28*28*512=400\text{K}$ params: $(3*3*512)*512 = 2,359,296$
 POOL2: [14x14x512] memory: $14*14*512=100\text{K}$ params: 0
 CONV3-512: [14x14x512] memory: $14*14*512=100\text{K}$ params: $(3*3*512)*512 = 2,359,296$
 CONV3-512: [14x14x512] memory: $14*14*512=100\text{K}$ params: $(3*3*512)*512 = 2,359,296$
 CONV3-512: [14x14x512] memory: $14*14*512=100\text{K}$ params: $(3*3*512)*512 = 2,359,296$
 POOL2: [7x7x512] memory: $7*7*512=25\text{K}$ 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: $24\text{M} * 4 \text{ bytes} \approx 96\text{MB}$ / image (for a forward pass)

TOTAL params: 138M parameters



VGG16

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

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

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

POOL2: [112x112x64] memory: $112*112*64=800\text{K}$ params: 0

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

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

POOL2: [56x56x128] memory: $56*56*128=400\text{K}$ params: 0

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

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

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

POOL2: [28x28x256] memory: $28*28*256=200\text{K}$ params: 0

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

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

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

POOL2: [14x14x512] memory: $14*14*512=100\text{K}$ params: 0

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

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

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

POOL2: [7x7x512] memory: $7*7*512=25\text{K}$ 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: $24\text{M} * 4 \text{ bytes} \approx 96\text{MB}$ / image (only forward! $\sim*2$ for bwd)

TOTAL params: 138M parameters

Note:

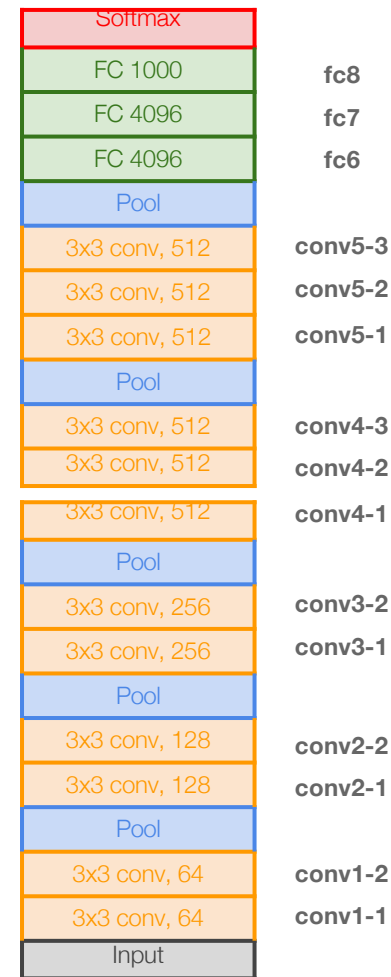
Most memory is in
early CONV

Most params are
in late FC

INPUT: [224x224x3] memory: $224*224*3=150\text{K}$ params: 0 (not counting biases)
 CONV3-64: [224x224x64] memory: $224*224*64=3.2\text{M}$ params: $(3*3*3)*64 = 1,728$
 CONV3-64: [224x224x64] memory: $224*224*64=3.2\text{M}$ params: $(3*3*64)*64 = 36,864$
 POOL2: [112x112x64] memory: $112*112*64=800\text{K}$ params: 0
 CONV3-128: [112x112x128] memory: $112*112*128=1.6\text{M}$ params: $(3*3*64)*128 = 73,728$
 CONV3-128: [112x112x128] memory: $112*112*128=1.6\text{M}$ params: $(3*3*128)*128 = 147,456$
 POOL2: [56x56x128] memory: $56*56*128=400\text{K}$ params: 0
 CONV3-256: [56x56x256] memory: $56*56*256=800\text{K}$ params: $(3*3*128)*256 = 294,912$
 CONV3-256: [56x56x256] memory: $56*56*256=800\text{K}$ params: $(3*3*256)*256 = 589,824$
 CONV3-256: [56x56x256] memory: $56*56*256=800\text{K}$ params: $(3*3*256)*256 = 589,824$
 POOL2: [28x28x256] memory: $28*28*256=200\text{K}$ params: 0
 CONV3-512: [28x28x512] memory: $28*28*512=400\text{K}$ params: $(3*3*256)*512 = 1,179,648$
 CONV3-512: [28x28x512] memory: $28*28*512=400\text{K}$ params: $(3*3*512)*512 = 2,359,296$
 CONV3-512: [28x28x512] memory: $28*28*512=400\text{K}$ params: $(3*3*512)*512 = 2,359,296$
 POOL2: [14x14x512] memory: $14*14*512=100\text{K}$ params: 0
 CONV3-512: [14x14x512] memory: $14*14*512=100\text{K}$ params: $(3*3*512)*512 = 2,359,296$
 CONV3-512: [14x14x512] memory: $14*14*512=100\text{K}$ params: $(3*3*512)*512 = 2,359,296$
 CONV3-512: [14x14x512] memory: $14*14*512=100\text{K}$ params: $(3*3*512)*512 = 2,359,296$
 POOL2: [7x7x512] memory: $7*7*512=25\text{K}$ 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: $24\text{M} * 4 \text{ bytes} \approx 96\text{MB}$ / image (only forward! $\sim*2$ for bwd)

TOTAL params: 138M parameters



VGG16

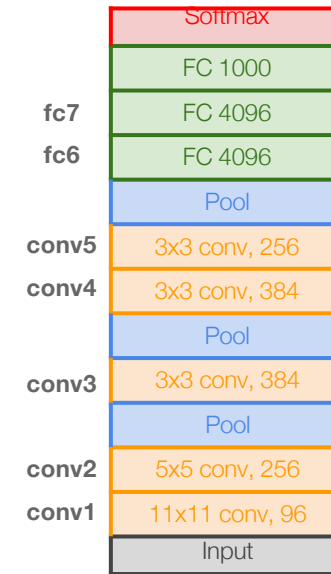
Common names

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Details:

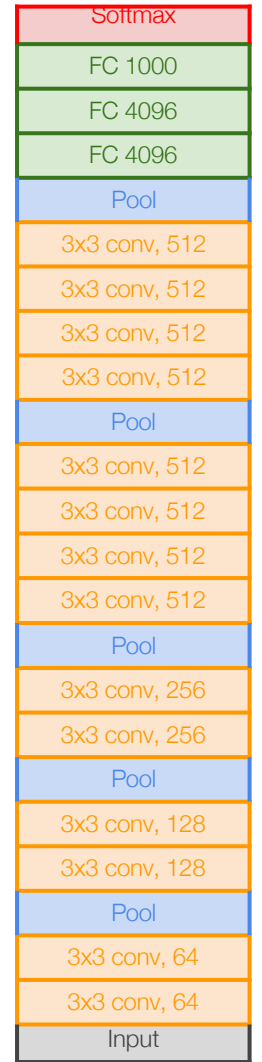
- ILSVRC'14 2nd in classification, 1st in localization
- Similar training procedure as Krizhevsky 2012
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks



AlexNet

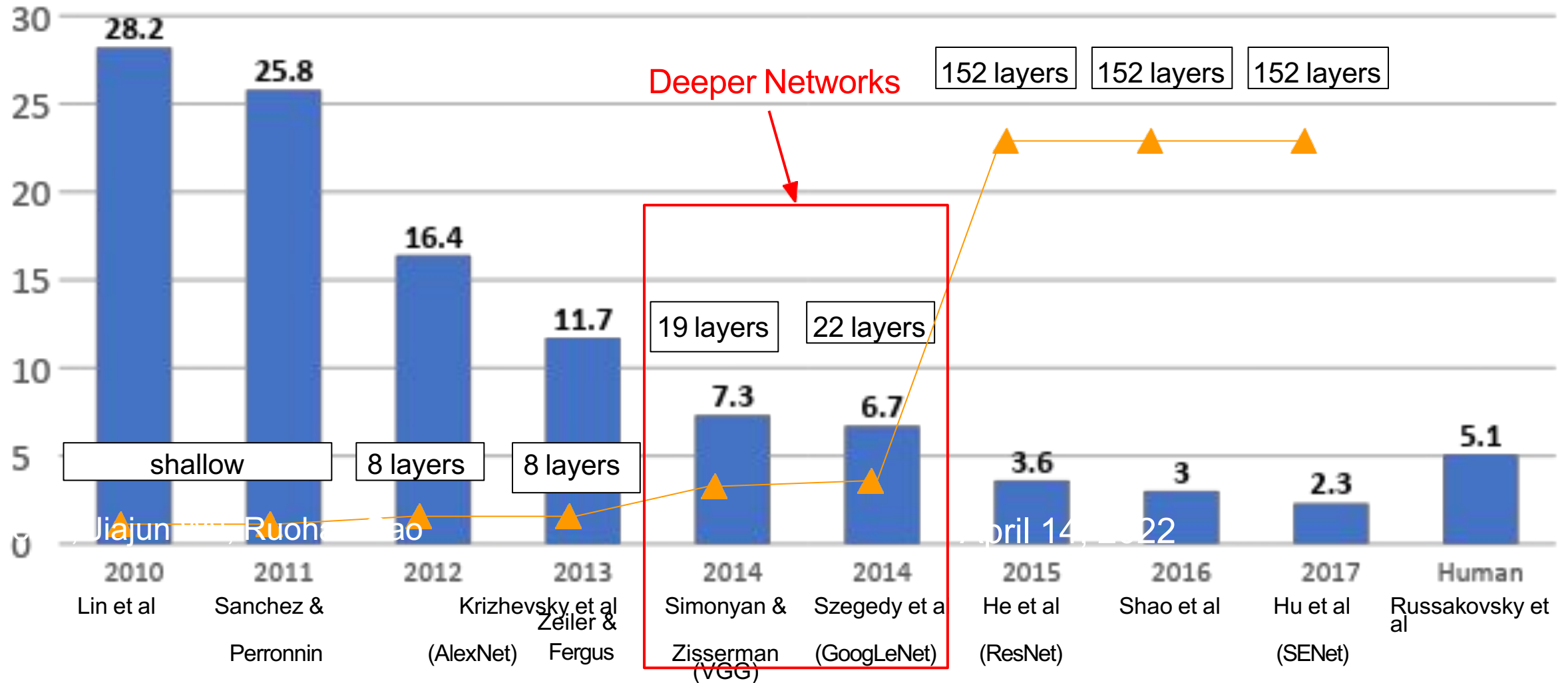


VGG16



VGG19

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

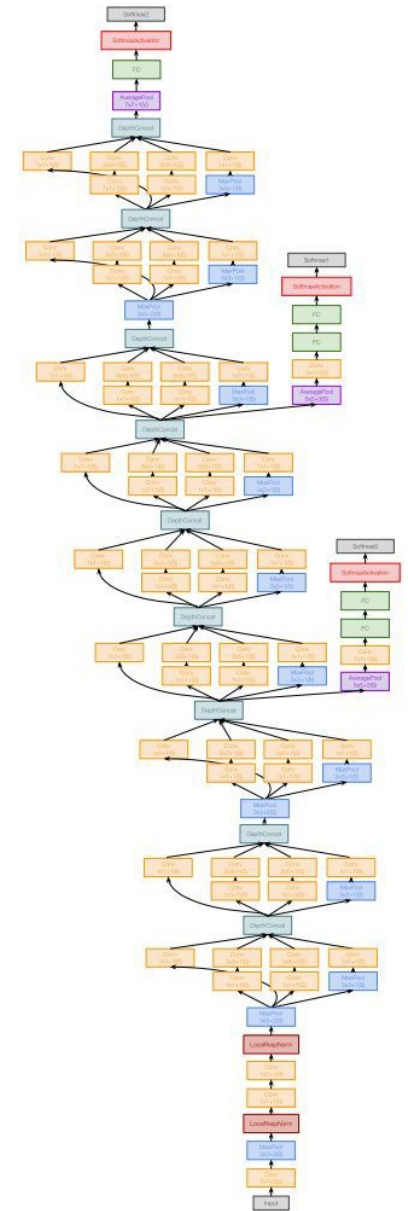
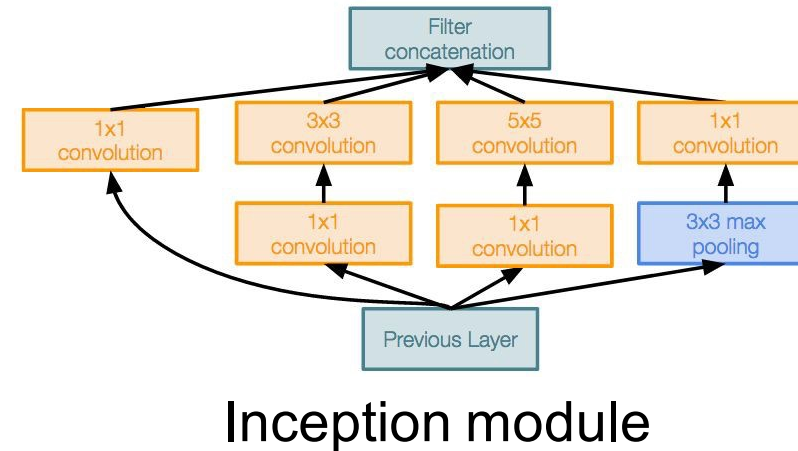


Case Study: GoogLeNet

[Szegedy et al., 2014]

Deeper networks, with computational efficiency

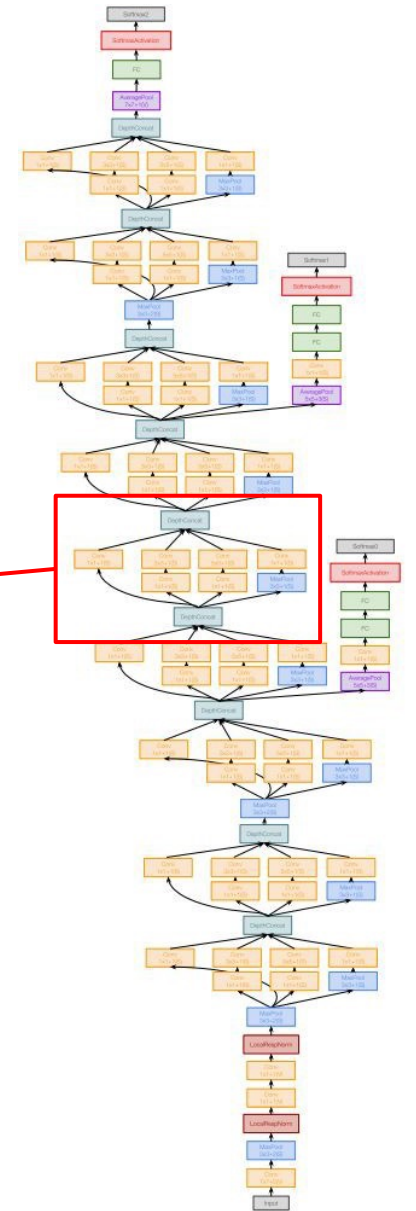
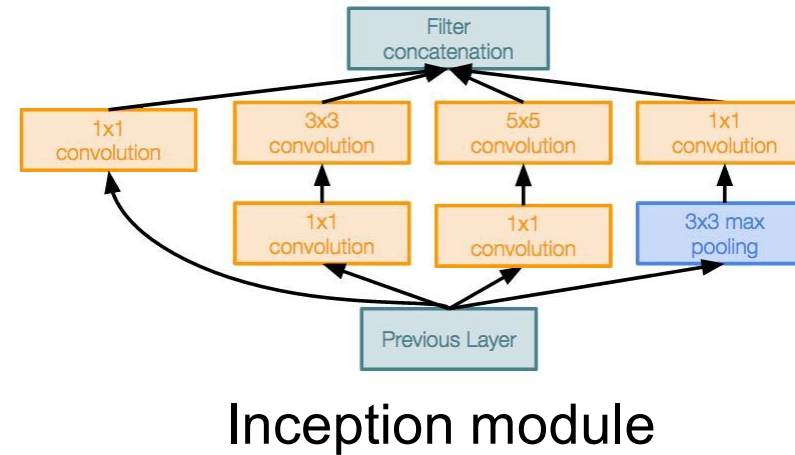
- ILSVRC'14 classification winner (6.7% top 5 error)
- 22 layers
- Only 5 million parameters!
12x less than AlexNet
27x less than VGG-16
- Efficient “Inception” module
- No FC layers



Case Study: GoogLeNet

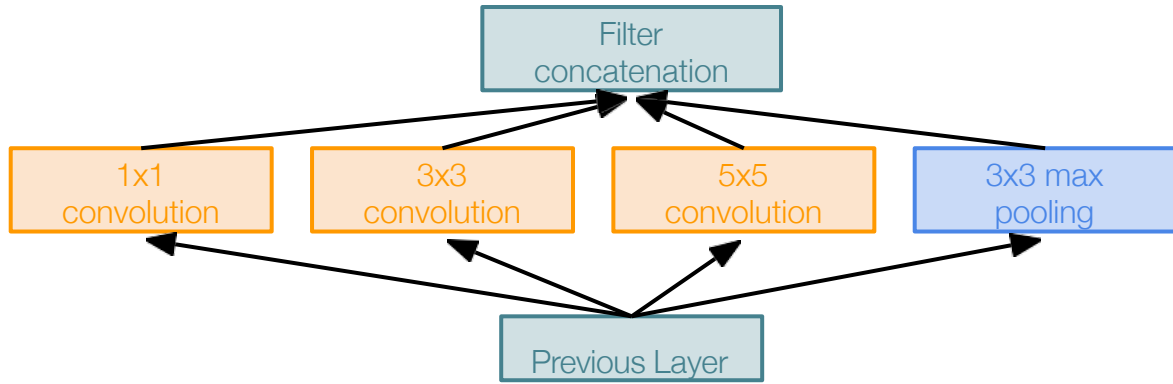
[Szegedy et al., 2014]

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



Case Study: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

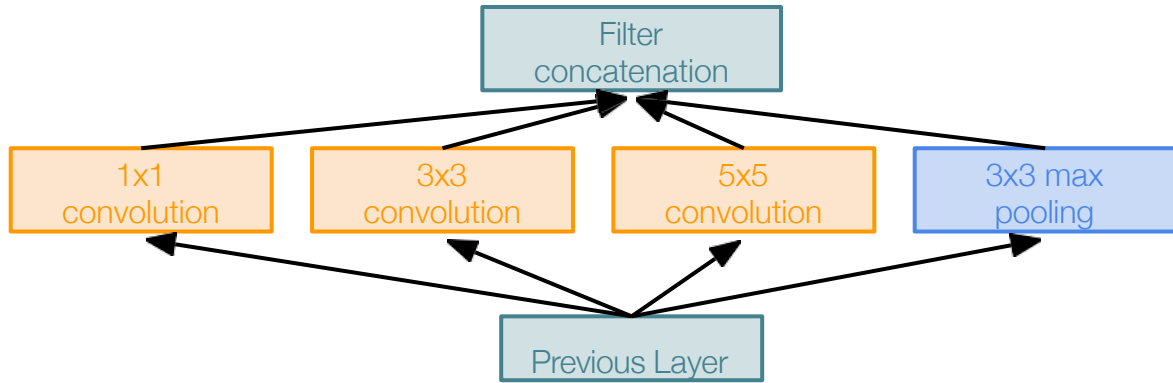
Apply parallel filter operations on the input from previous layer:

- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together channel-wise

Case Study: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

Apply parallel filter operations on the input from previous layer:

- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together channel-wise

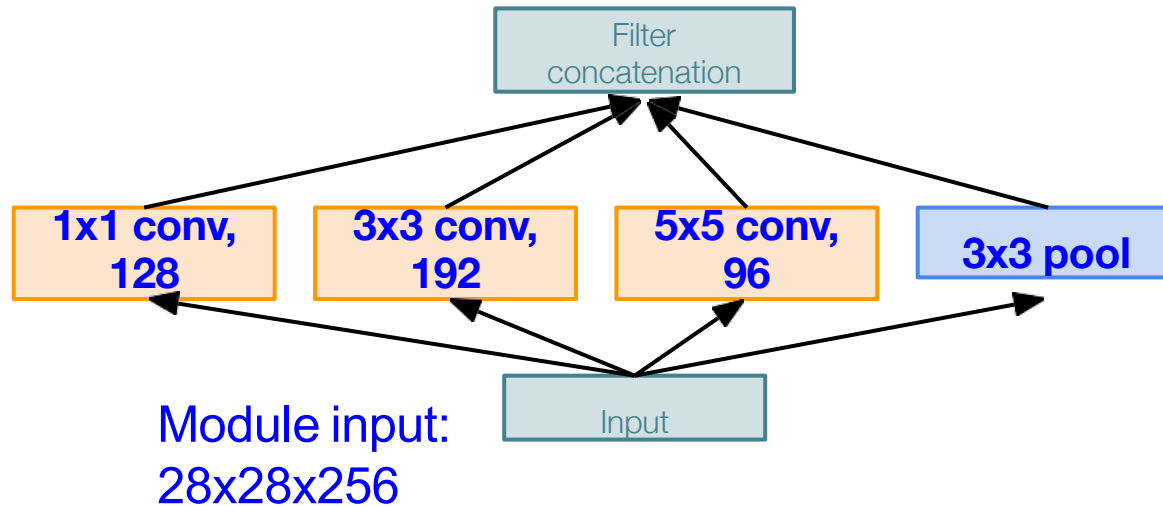
Q: What is the problem with this?
[Hint: Computational complexity]

Case Study: GoogLeNet

[Szegedy et al., 2014]

Q: What is the problem with this?
[Hint: Computational complexity]

Example:



Naive Inception module

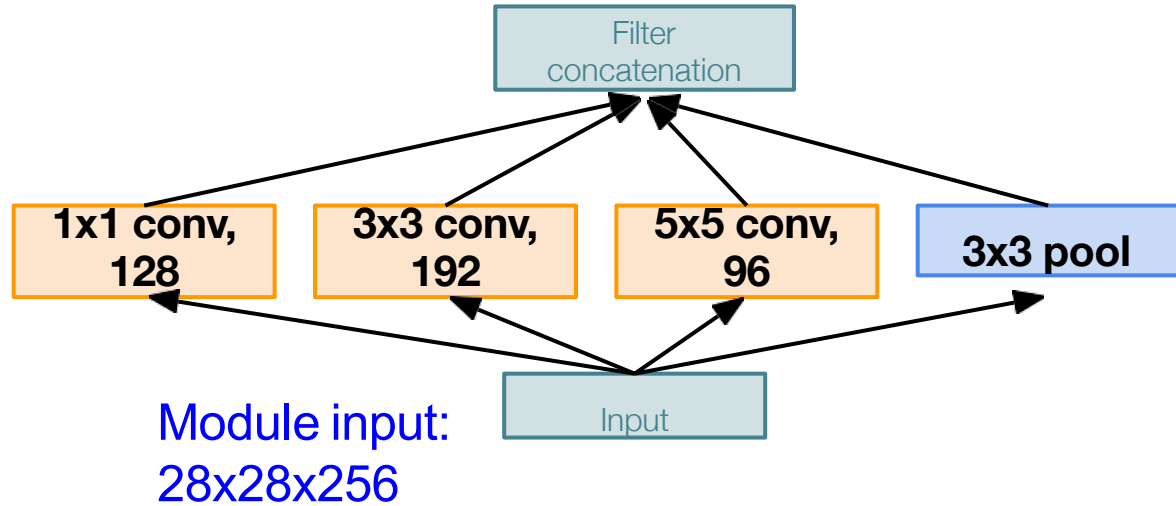
Case Study: GoogLeNet

[Szegedy et al., 2014]

Q: What is the problem with this?
[Hint: Computational complexity]

Example:

Q1: What are the output sizes of
all different filter operations?



Naive Inception module

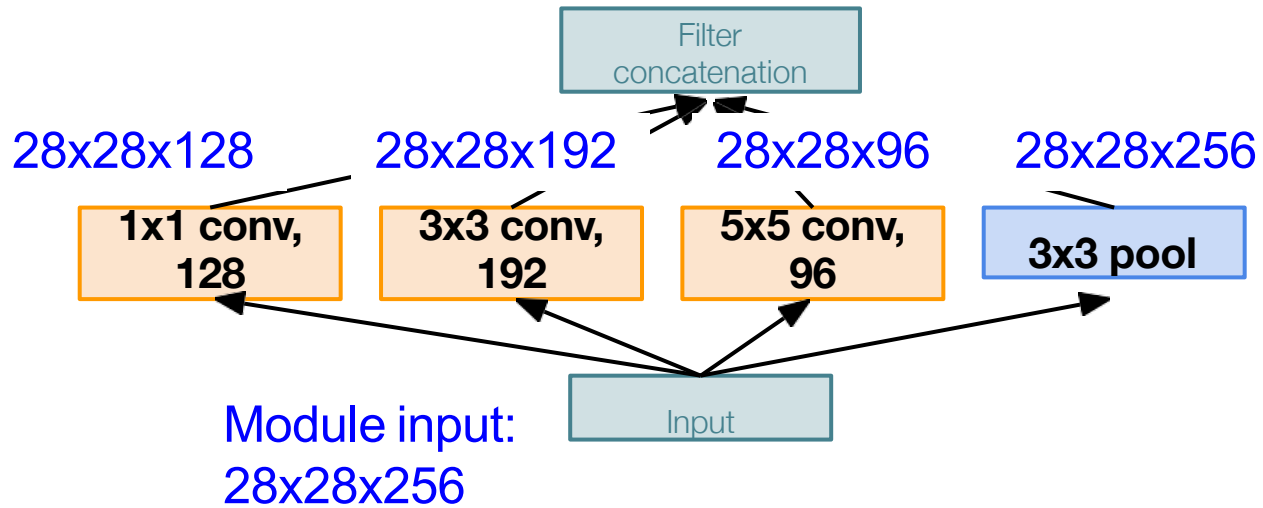
Case Study: GoogLeNet

[Szegedy et al., 2014]

Q: What is the problem with this?
[Hint: Computational complexity]

Example:

Q1: What are the output sizes of
all different filter operations?



Naive Inception module

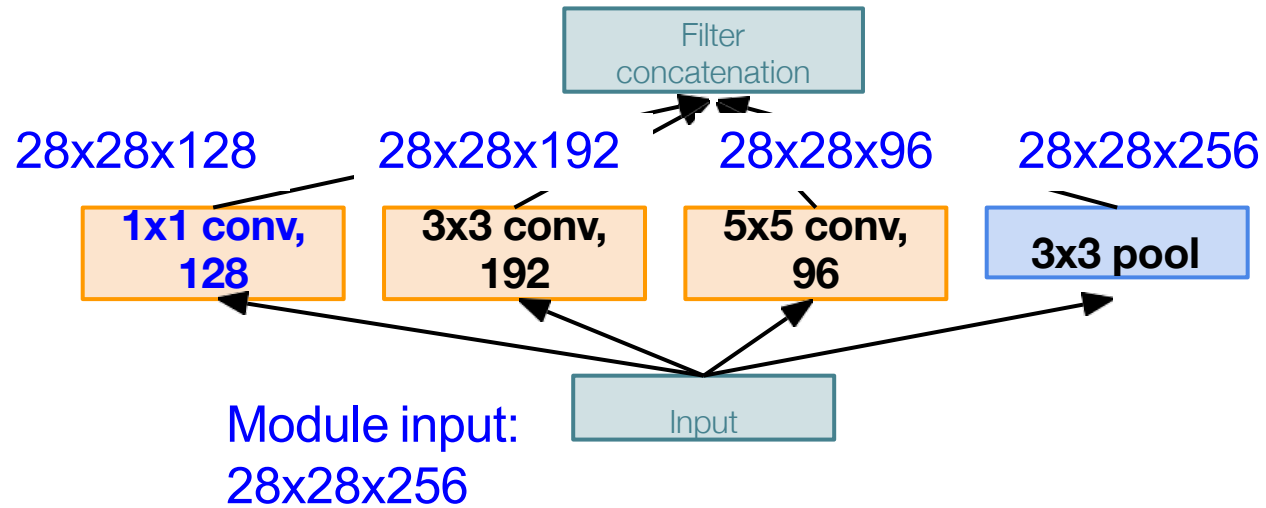
Case Study: GoogLeNet

[Szegedy et al., 2014]

Q: What is the problem with this?
[Hint: Computational complexity]

Example:

Q2: What is output size after
filter concatenation?



Naive Inception module

Case Study: GoogLeNet

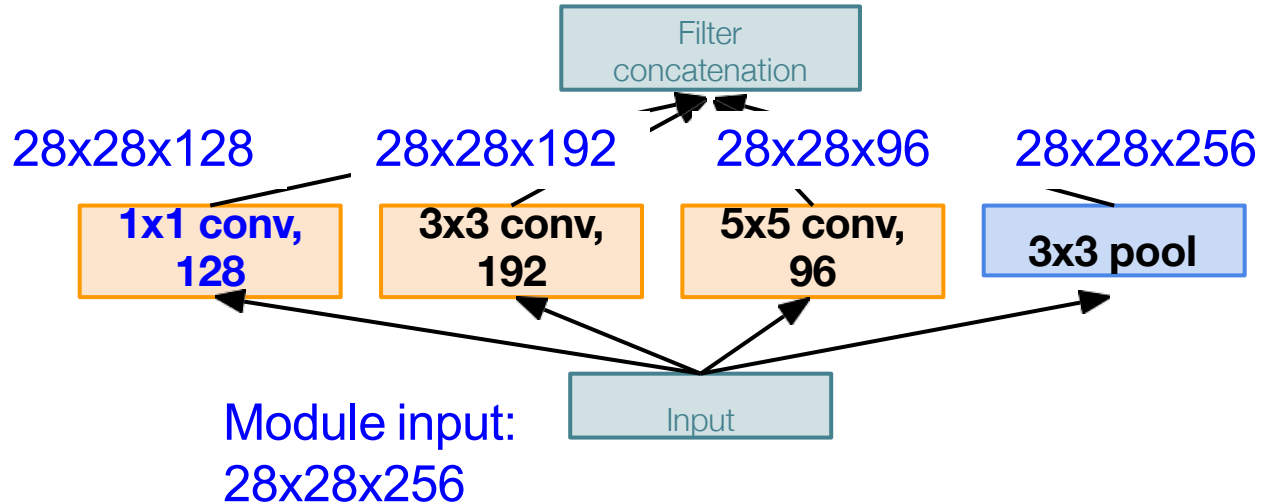
[Szegedy et al., 2014]

Q: What is the problem with this?
[Hint: Computational complexity]

Example:

Q2: What is output size after
filter concatenation?

$$28 \times 28 \times (128 + 192 + 96 + 256) = 28 \times 28 \times 672$$



Naive Inception module

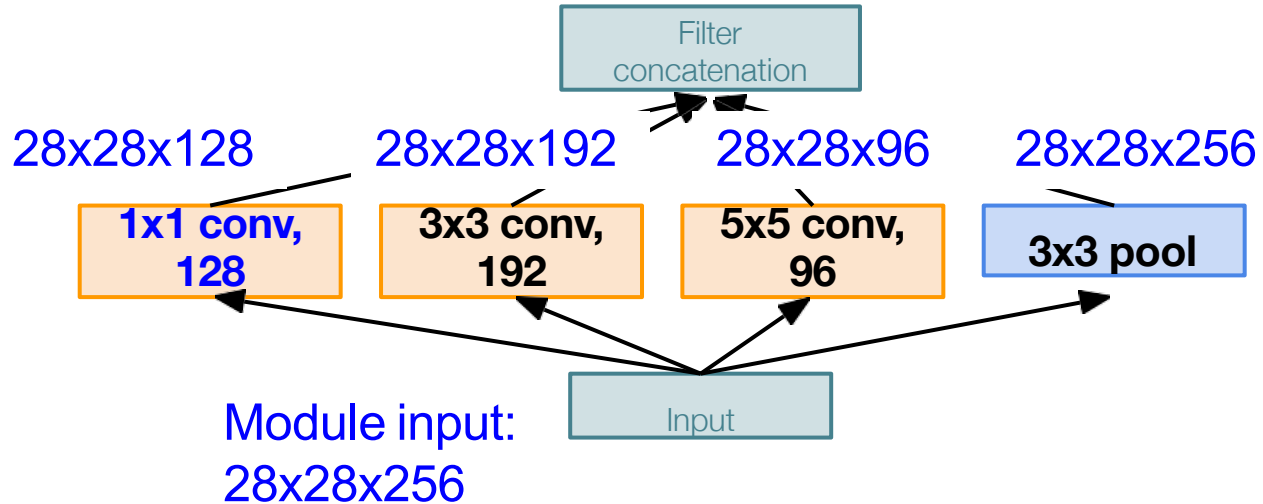
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q2: What is output size after filter concatenation?

$$28 \times 28 \times (128 + 192 + 96 + 256) = 28 \times 28 \times 672$$



Naive Inception module

Q: What is the problem with this?
[Hint: Computational complexity]

Conv Ops:

[1x1 conv, 128] $28 \times 28 \times 128 \times 1 \times 1 \times 256$

[3x3 conv, 192] $28 \times 28 \times 192 \times 3 \times 3 \times 256$

[5x5 conv, 96] $28 \times 28 \times 96 \times 5 \times 5 \times 256$

Total: 854M ops

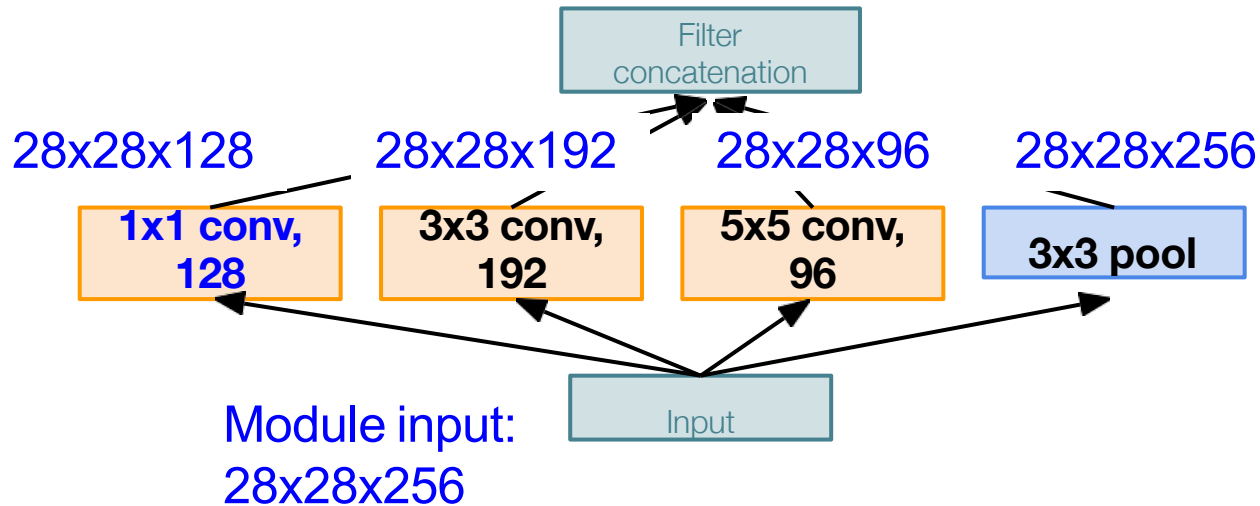
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q2: What is output size after filter concatenation?

$$28 \times 28 \times (128 + 192 + 96 + 256) = 28 \times 28 \times 672$$



Naive Inception module

Q: What is the problem with this?
[Hint: Computational complexity]

Conv Ops:

[1x1 conv, 128] $28 \times 28 \times 128 \times 1 \times 1 \times 256$

[3x3 conv, 192] $28 \times 28 \times 192 \times 3 \times 3 \times 256$

[5x5 conv, 96] $28 \times 28 \times 96 \times 5 \times 5 \times 256$

Total: 854M ops

Very expensive compute

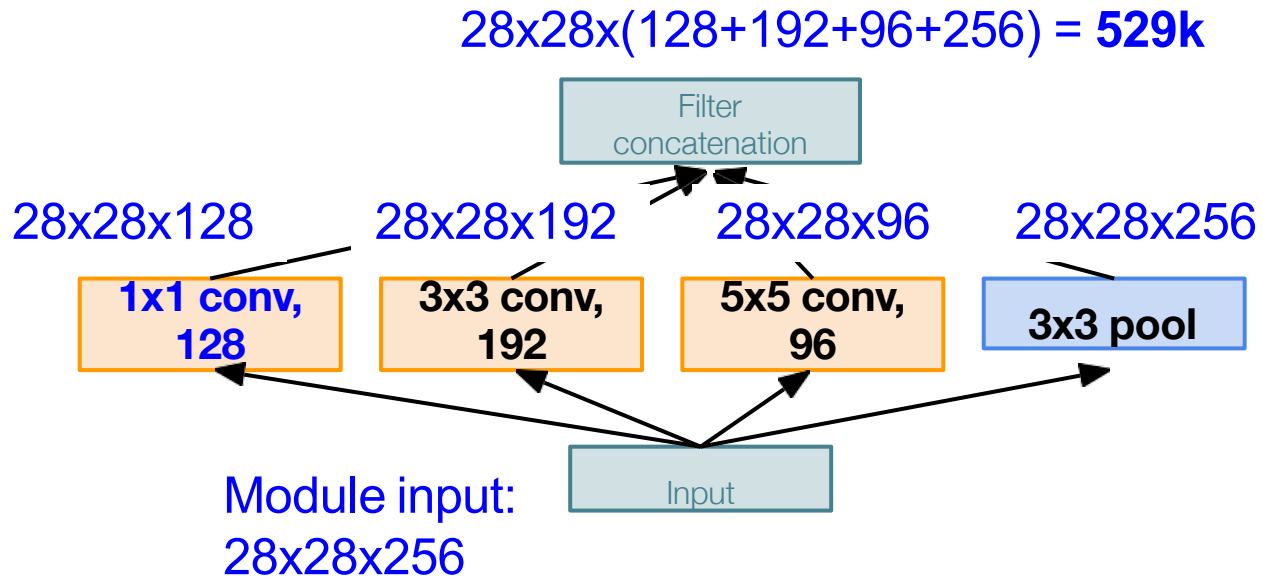
Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer!

Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q2: What is output size after filter concatenation?



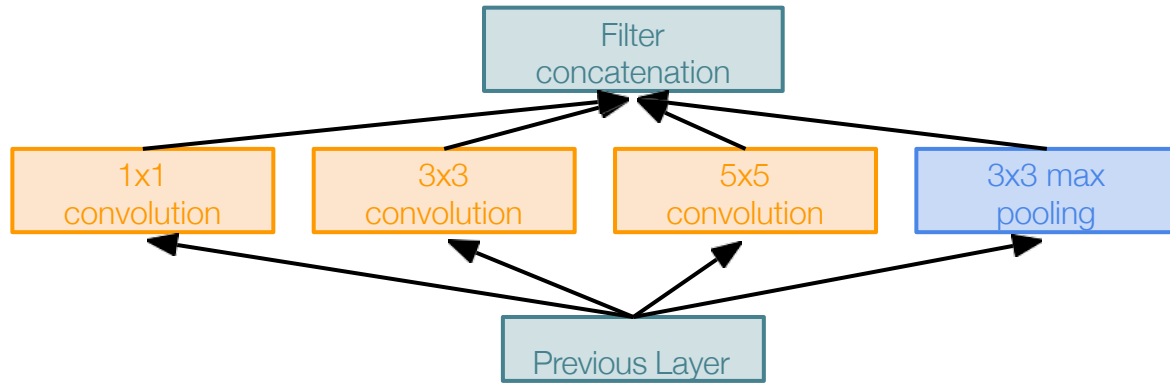
Naive Inception module

Q: What is the problem with this?
[Hint: Computational complexity]

Solution: “bottleneck” layers that use 1x1 convolutions to reduce feature channel size

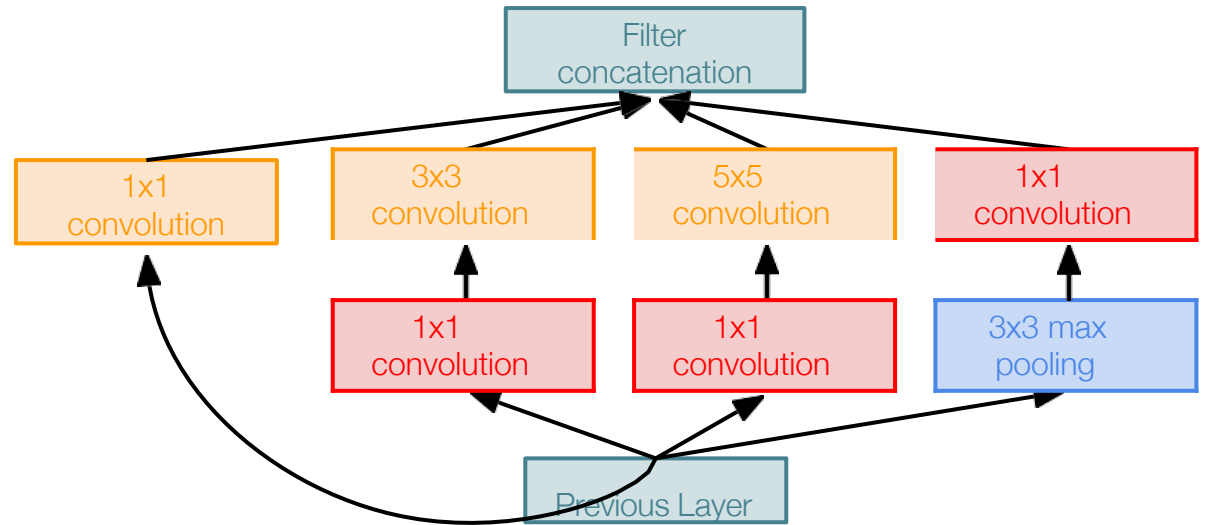
Case Study: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

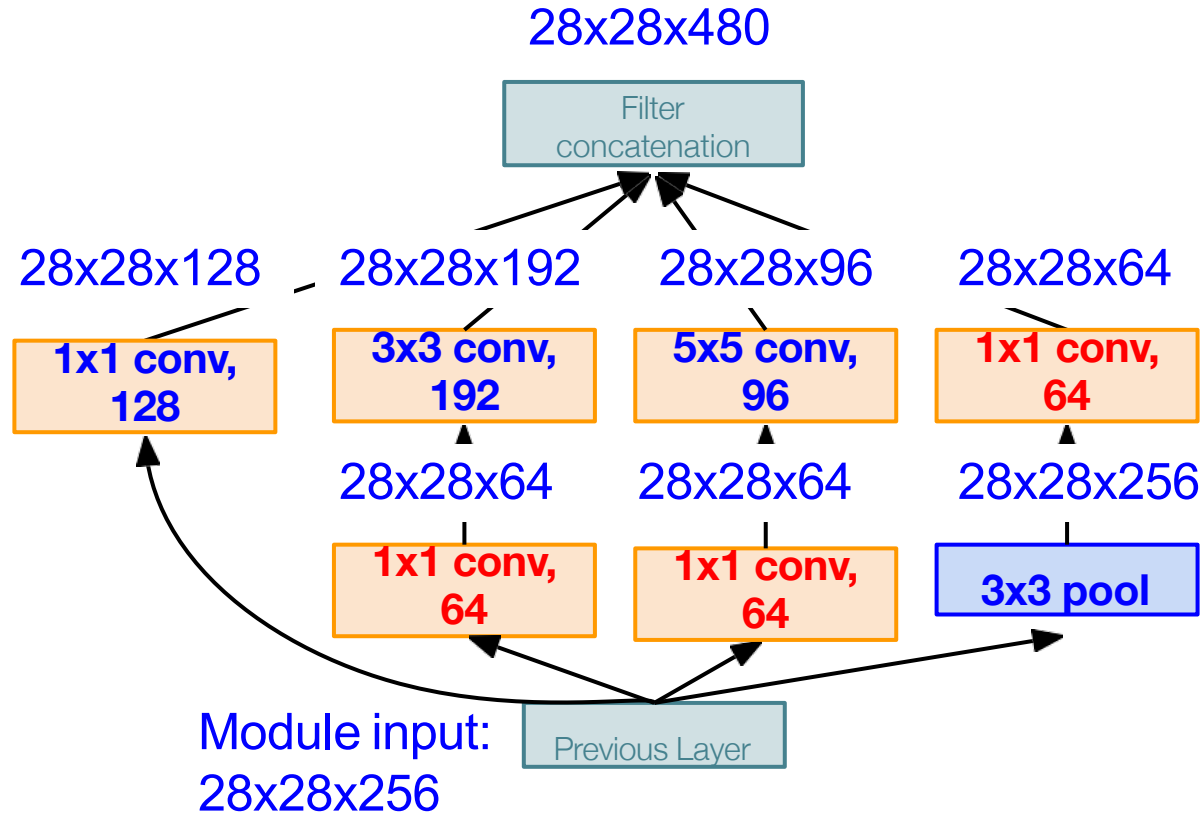
1x1 conv “bottleneck”
layers



Inception module with dimension reduction

Case Study: GoogLeNet

[Szegedy et al., 2014]



Inception module with dimension reduction

Using same parallel layers as naive example, and adding “ 1×1 conv, 64 filter” bottlenecks:

Conv Ops:

[1×1 conv, 64] $28 \times 28 \times 64 \times 1 \times 1 \times 256$
[1×1 conv, 64] $28 \times 28 \times 64 \times 1 \times 1 \times 256$
[1×1 conv, 128] $28 \times 28 \times 128 \times 1 \times 1 \times 256$
[3×3 conv, 192] $28 \times 28 \times 192 \times 3 \times 3 \times 64$
[5×5 conv, 96] $28 \times 28 \times 96 \times 5 \times 5 \times 64$
[1×1 conv, 64] $28 \times 28 \times 64 \times 1 \times 1 \times 256$

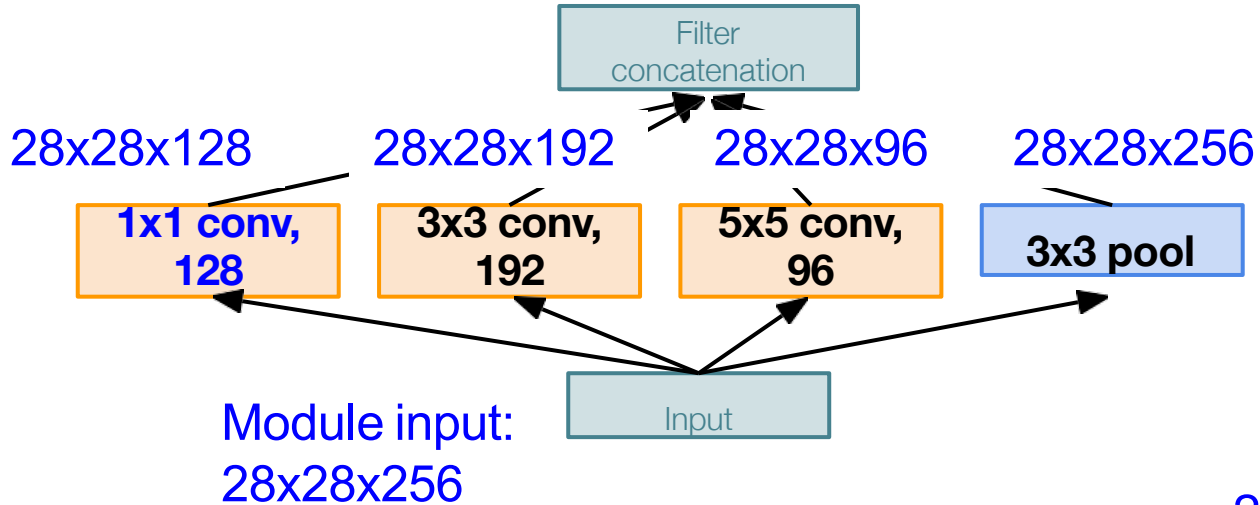
Total: 358M ops

Compared to 854M ops for naive version
Bottleneck can also reduce depth after pooling layer

Case Study: GoogLeNet

[Szegedy et al., 2014]

$$28 \times 28 \times (128 + 192 + 96 + 256) = 529k$$



Naive Inception module

Conv Ops:

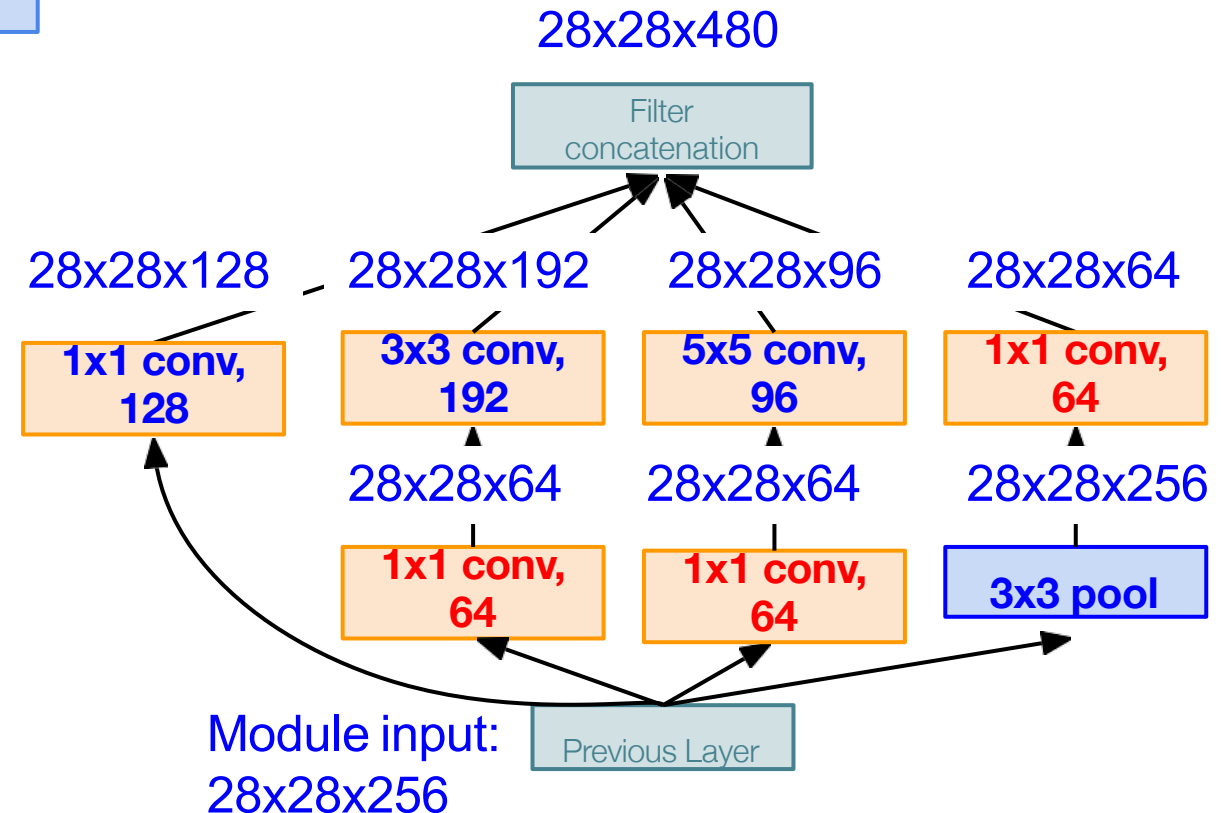
[1x1 conv, 128] $28 \times 28 \times 128 \times 1 \times 1 \times 256$
[3x3 conv, 192] $28 \times 28 \times 192 \times 3 \times 3 \times 256$
[5x5 conv, 96] $28 \times 28 \times 96 \times 5 \times 5 \times 256$

Total: 854M ops

Conv Ops:

[1x1 conv, 64] $28 \times 28 \times 64 \times 1 \times 1 \times 256$
[1x1 conv, 64] $28 \times 28 \times 64 \times 1 \times 1 \times 256$
[1x1 conv, 128] $28 \times 28 \times 128 \times 1 \times 1 \times 256$
[3x3 conv, 192] $28 \times 28 \times 192 \times 3 \times 3 \times 64$
[5x5 conv, 96] $28 \times 28 \times 96 \times 5 \times 5 \times 64$
[1x1 conv, 64] $28 \times 28 \times 64 \times 1 \times 1 \times 256$

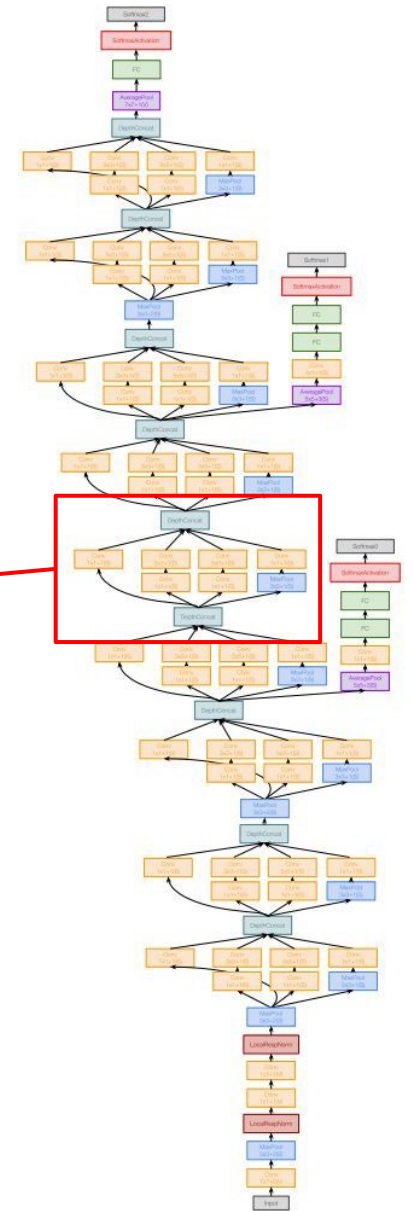
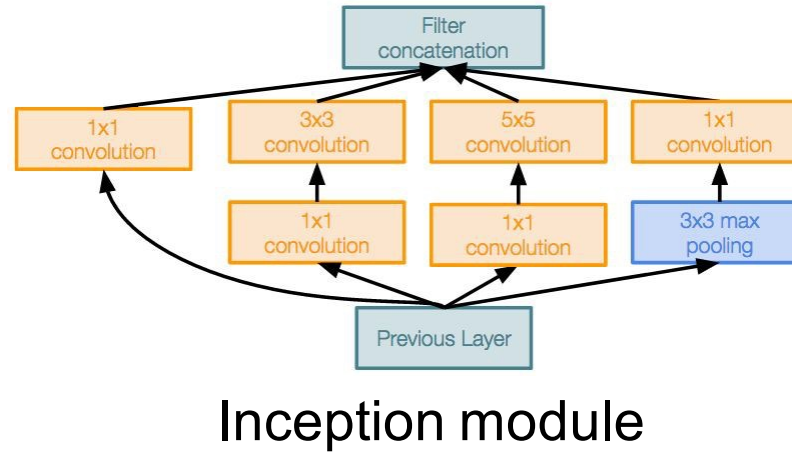
Total: 358M ops



Case Study: GoogLeNet

[Szegedy et al., 2014]

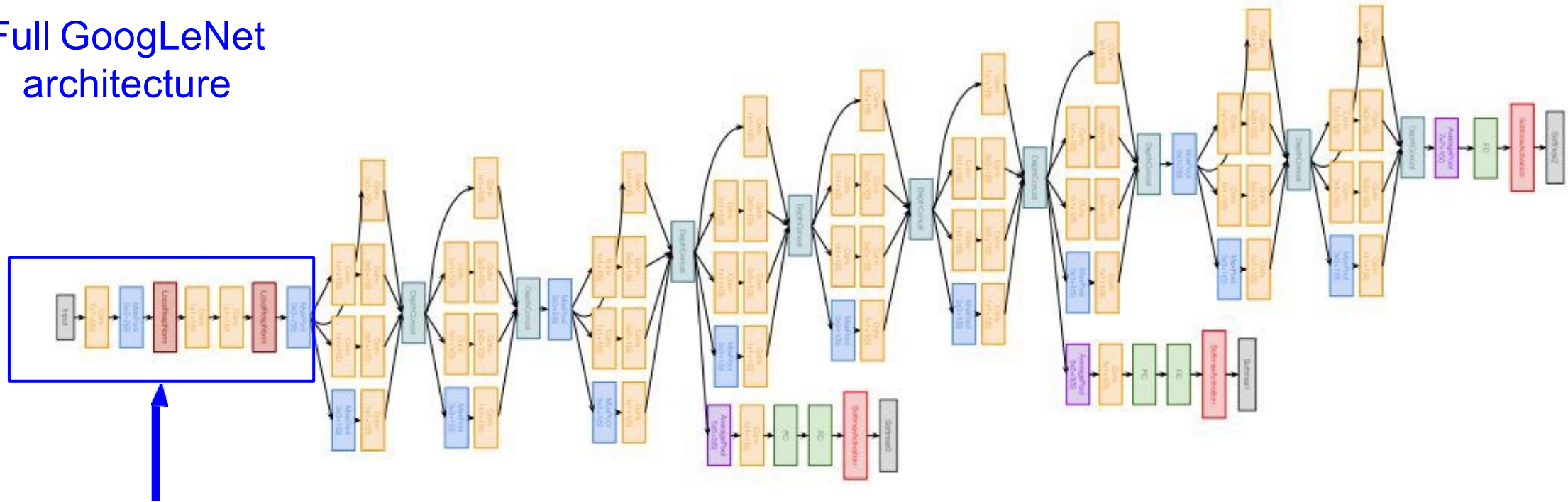
Stack Inception modules
with dimension reduction
on top of each other



Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet
architecture

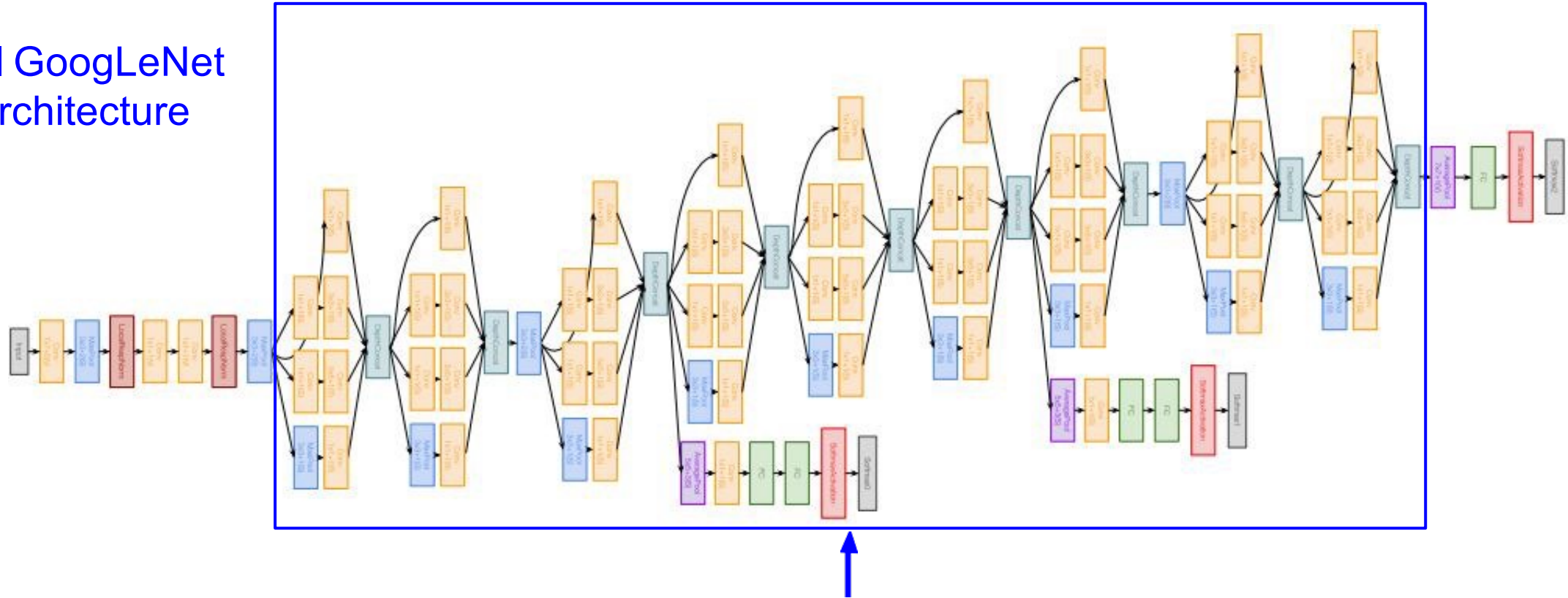


Stem Network:
Conv-Pool-
2x Conv-Pool

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture

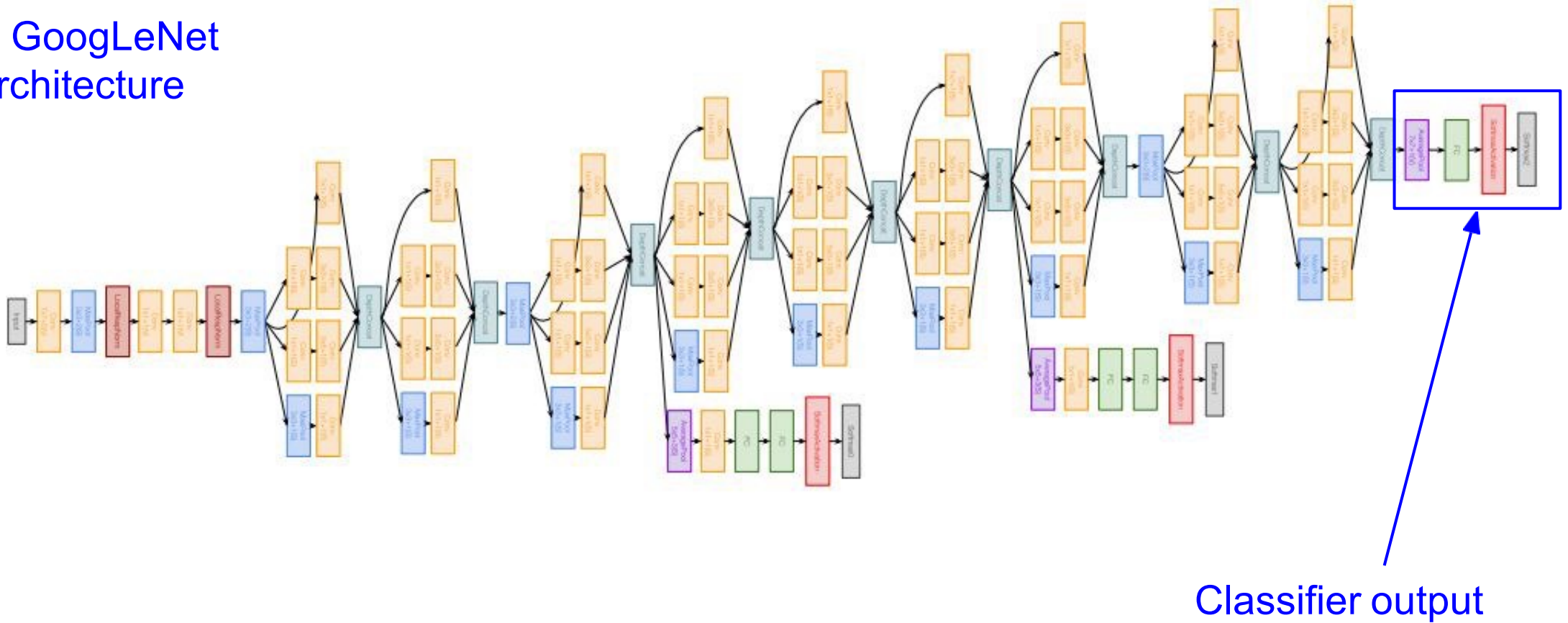


Stacked Inception Modules

Case Study: GoogLeNet

[Szegedy et al., 2014]

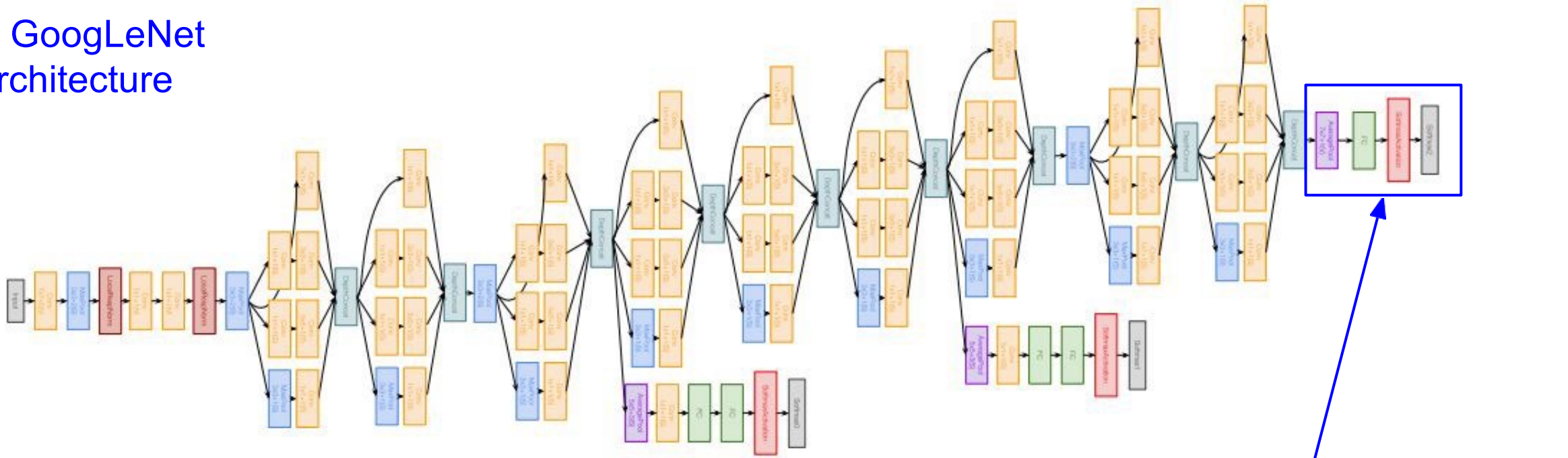
Full GoogLeNet architecture



Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture



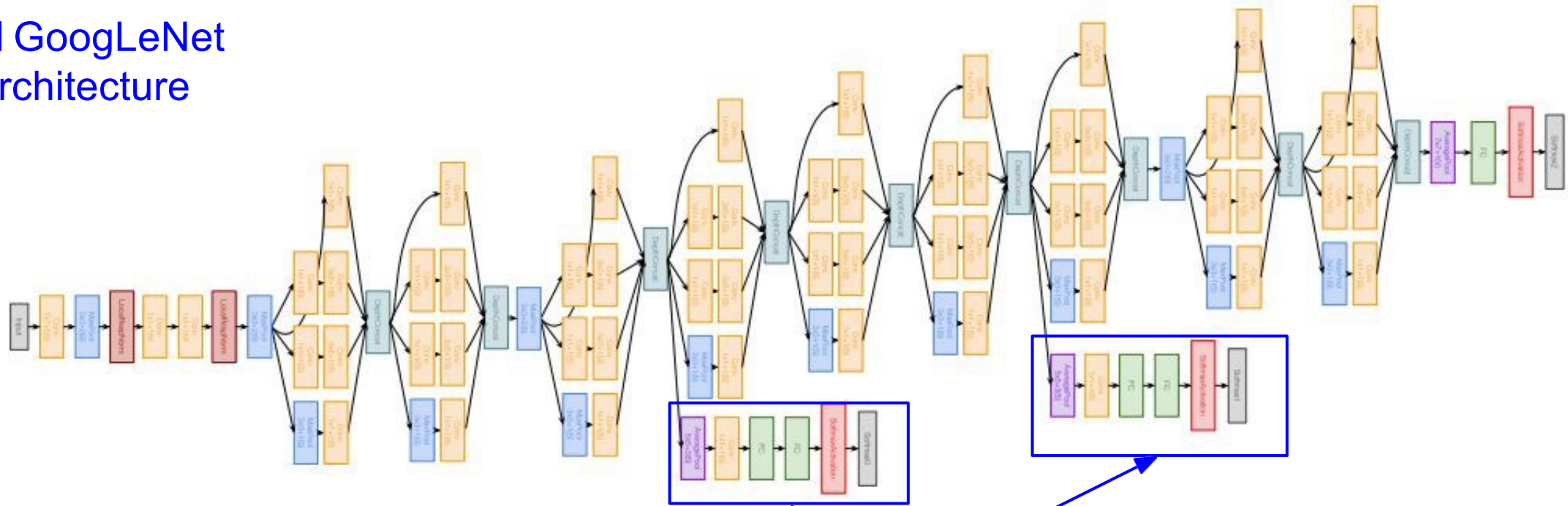
Note: after the last convolutional layer, a global average pooling layer is used that spatially averages across each feature map, before final FC layer. No longer multiple expensive FC layers!

Classifier output

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture

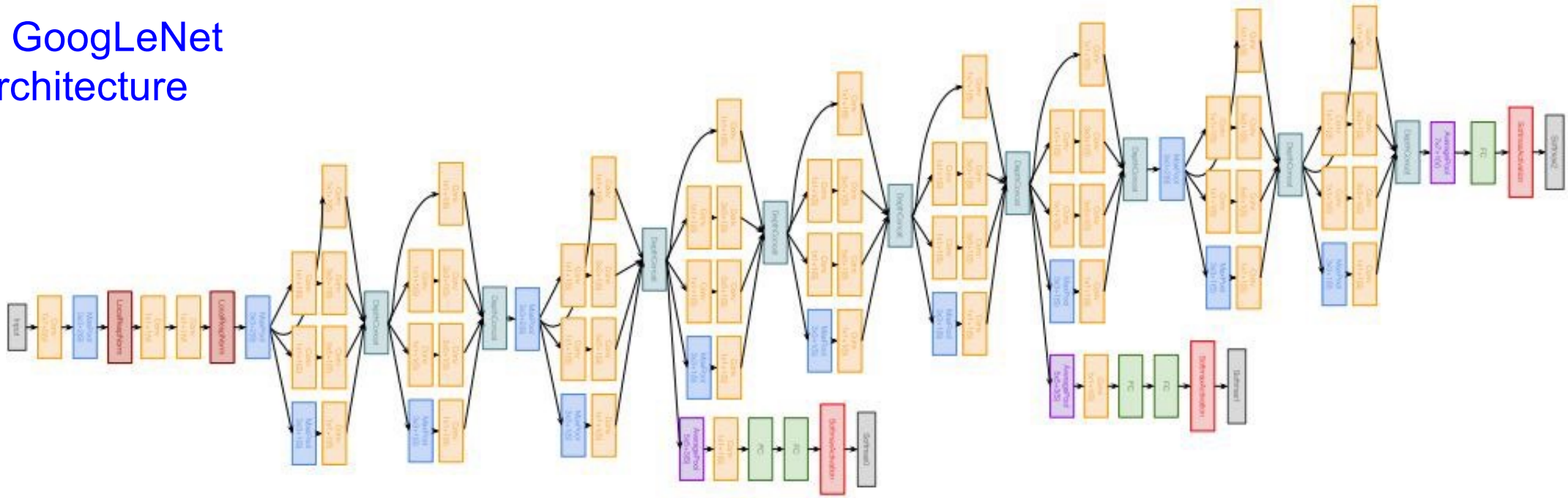


Auxiliary classification outputs to inject additional gradient at lower layers
(AvgPool-1x1Conv-FC-FC-Softmax)

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture



22 total layers with weights

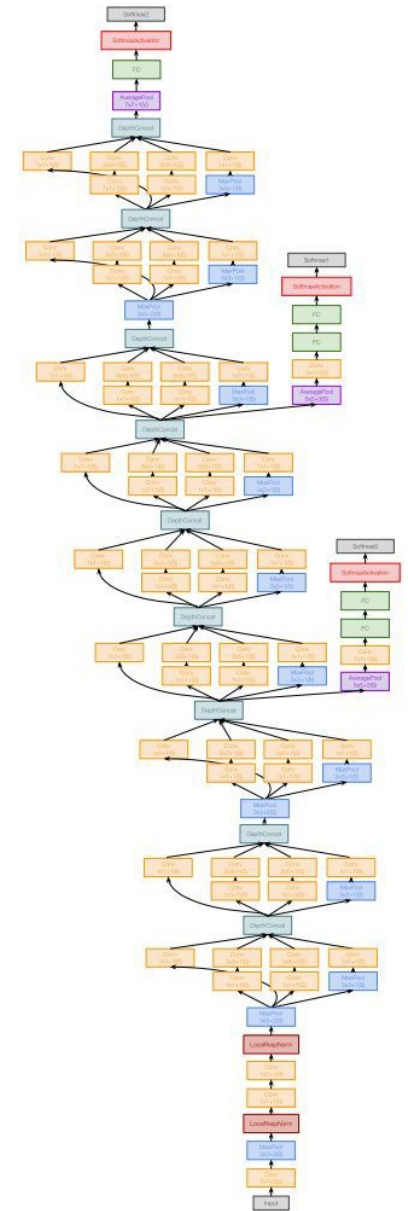
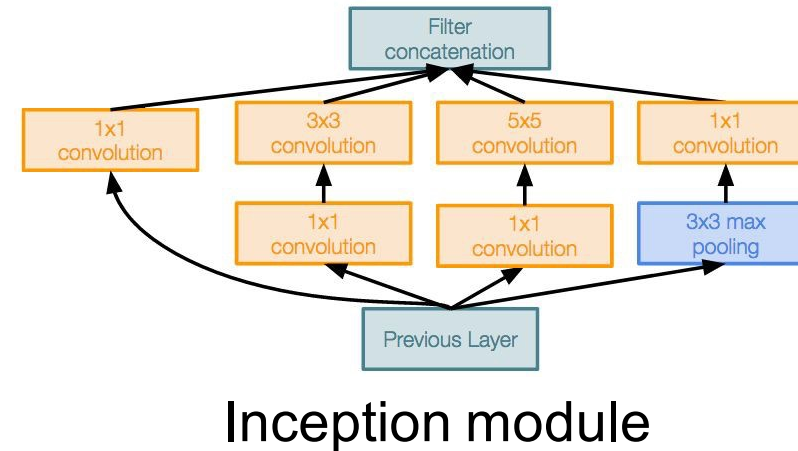
(parallel layers count as 1 layer => 2 layers per Inception module. Don't count auxiliary output layers)

Case Study: GoogLeNet

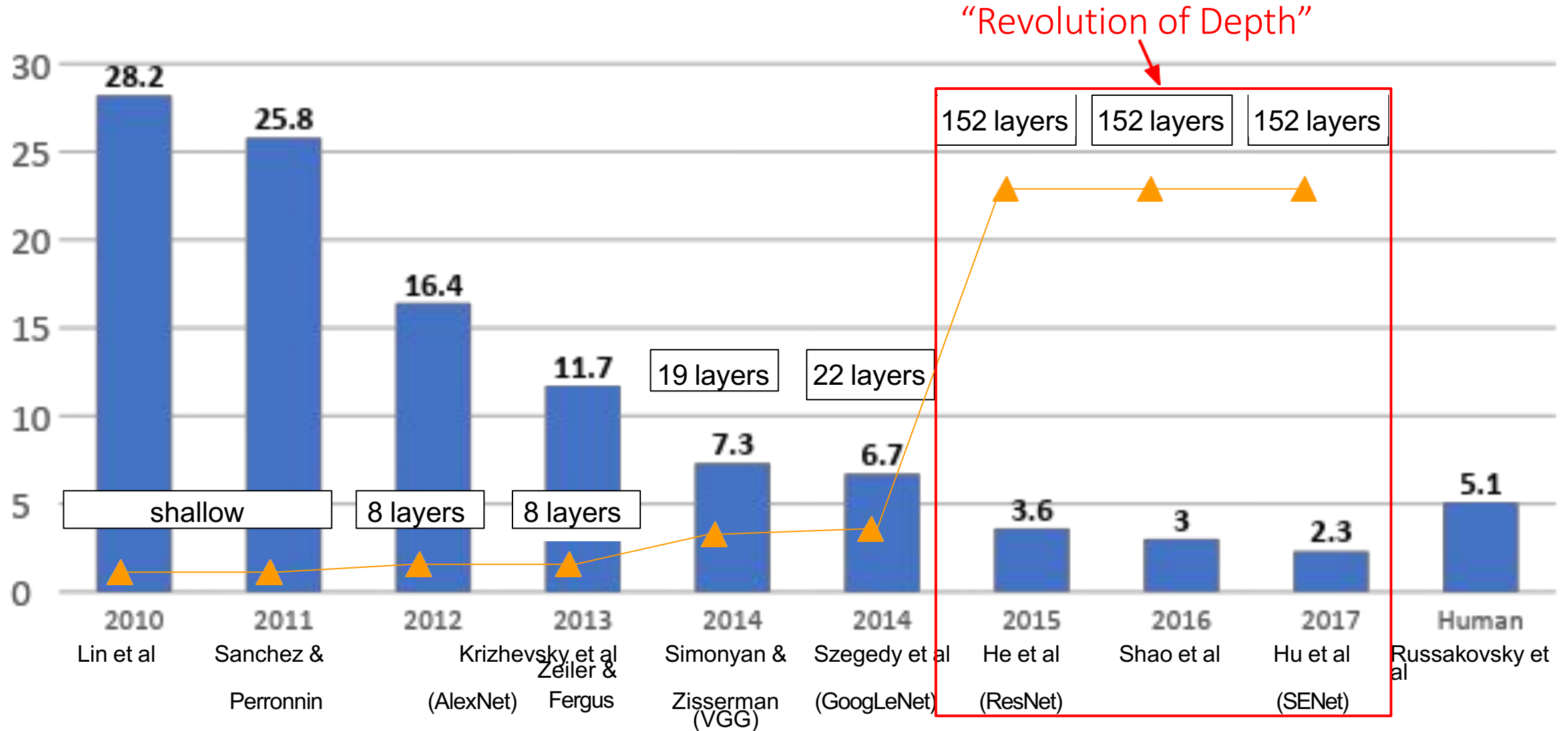
[Szegedy et al., 2014]

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

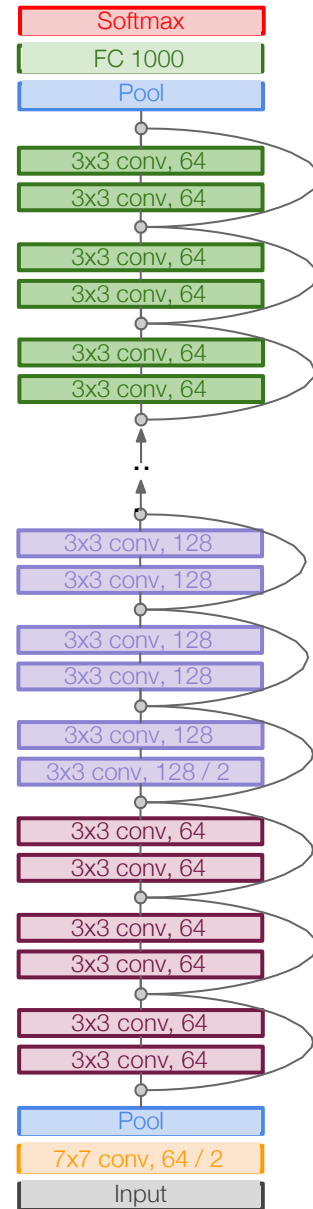
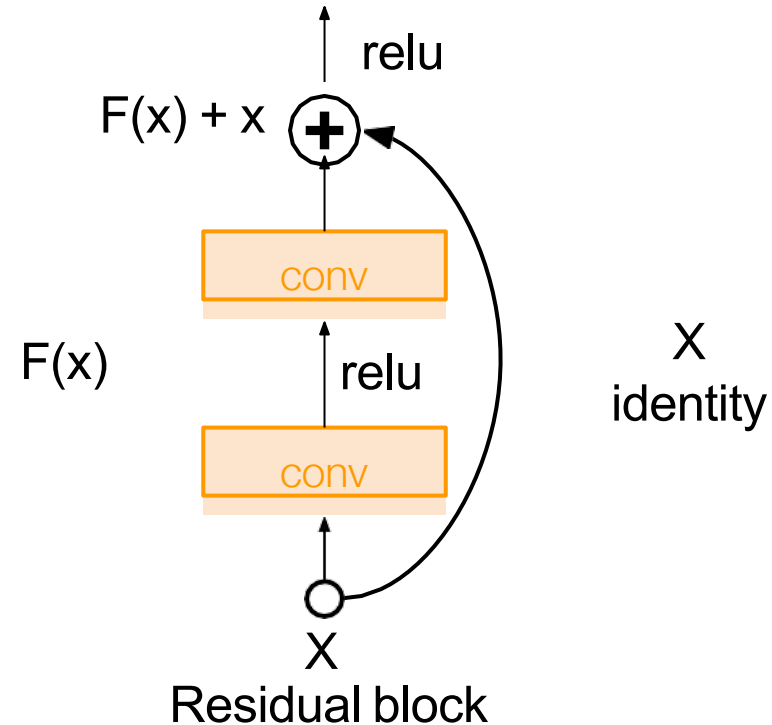


Case Study: ResNet

[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!



Case Study: ResNet

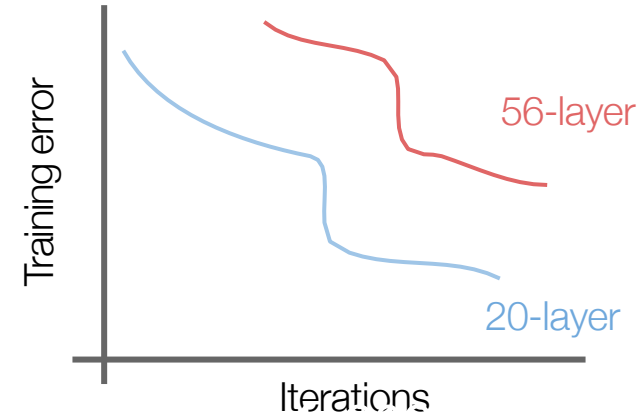
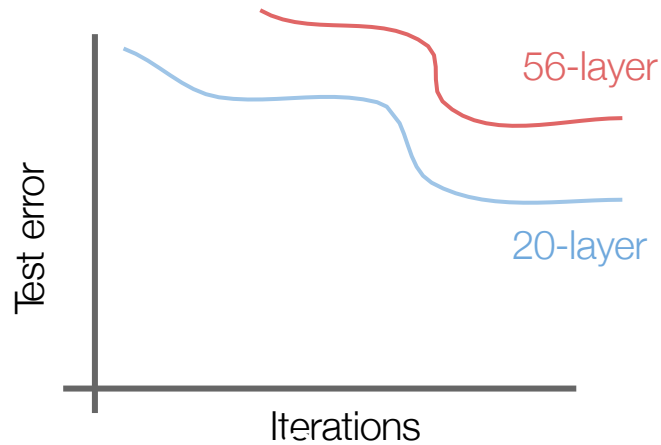
[He et al., 2015]

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

Case Study: ResNet

[He et al., 2015]

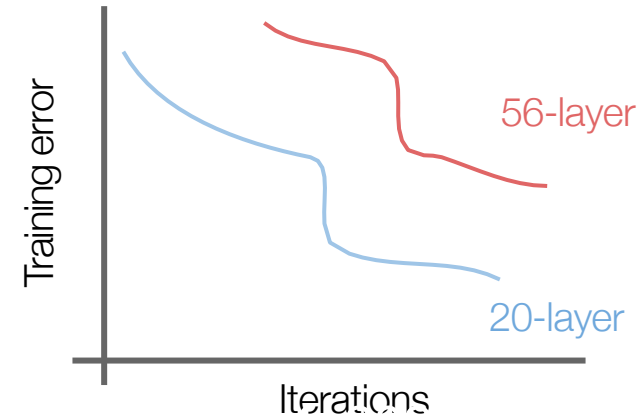
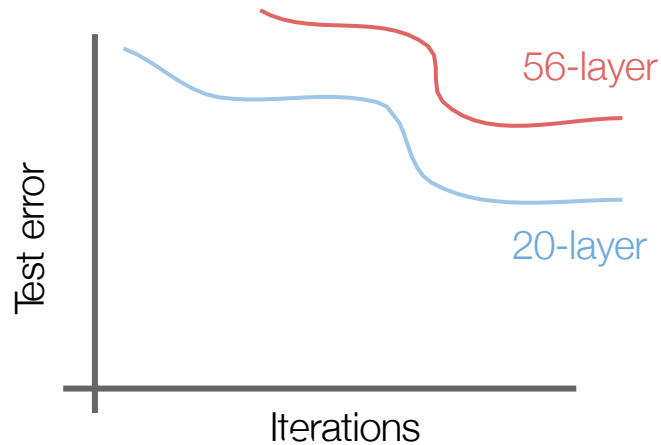
What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



Case Study: ResNet

[He et al., 2015]

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!**

Case Study: ResNet

[He et al., 2015]

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

Case Study: ResNet

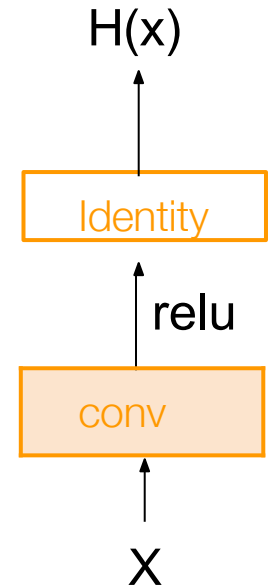
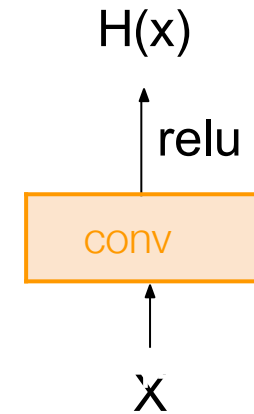
[He et al., 2015]

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?

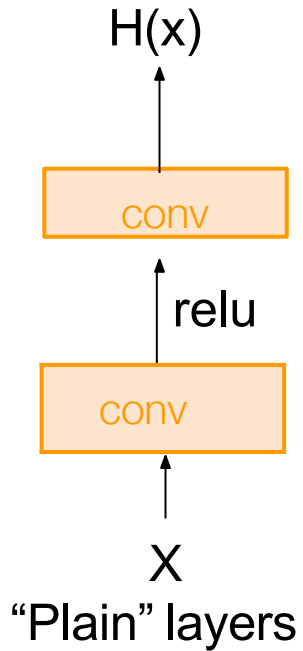
A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.



Case Study: ResNet

[He et al., 2015]

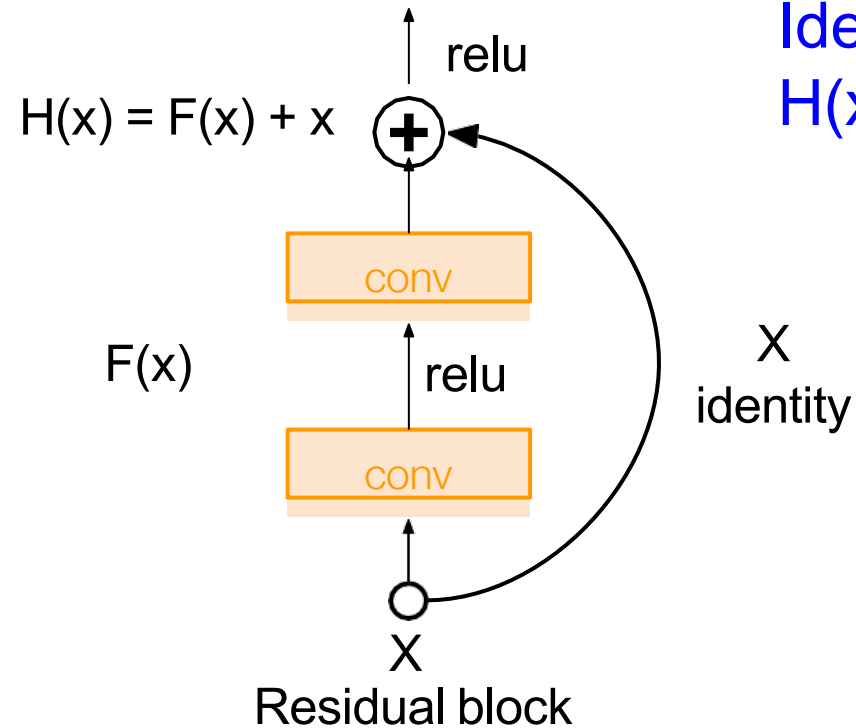
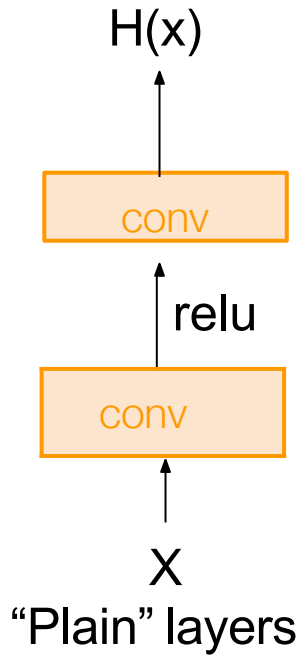
Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Case Study: ResNet

[He et al., 2015]

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

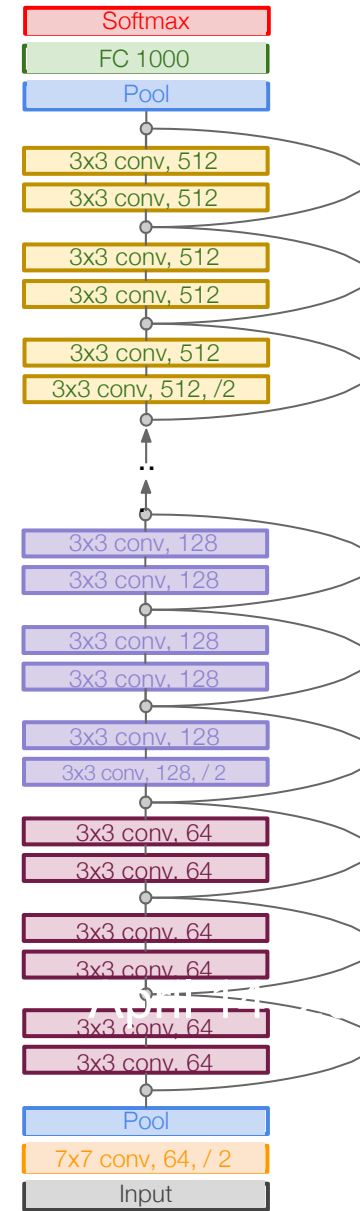


Identity mapping:
 $H(x) = x$ if $F(x) = 0$

Case Study: ResNet

[He et al., 2015]

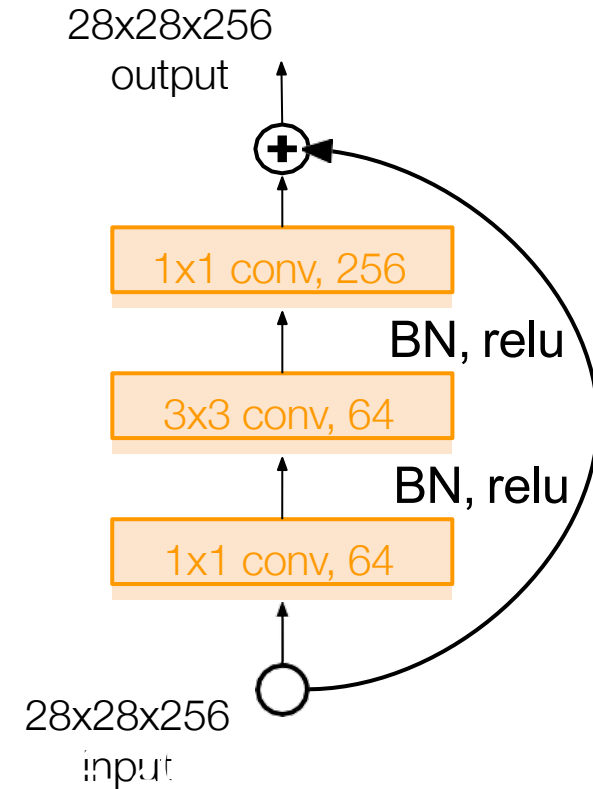
Total depths of 18, 34, 50,
101, or 152 layers for
ImageNet



Case Study: ResNet

[He et al., 2015]

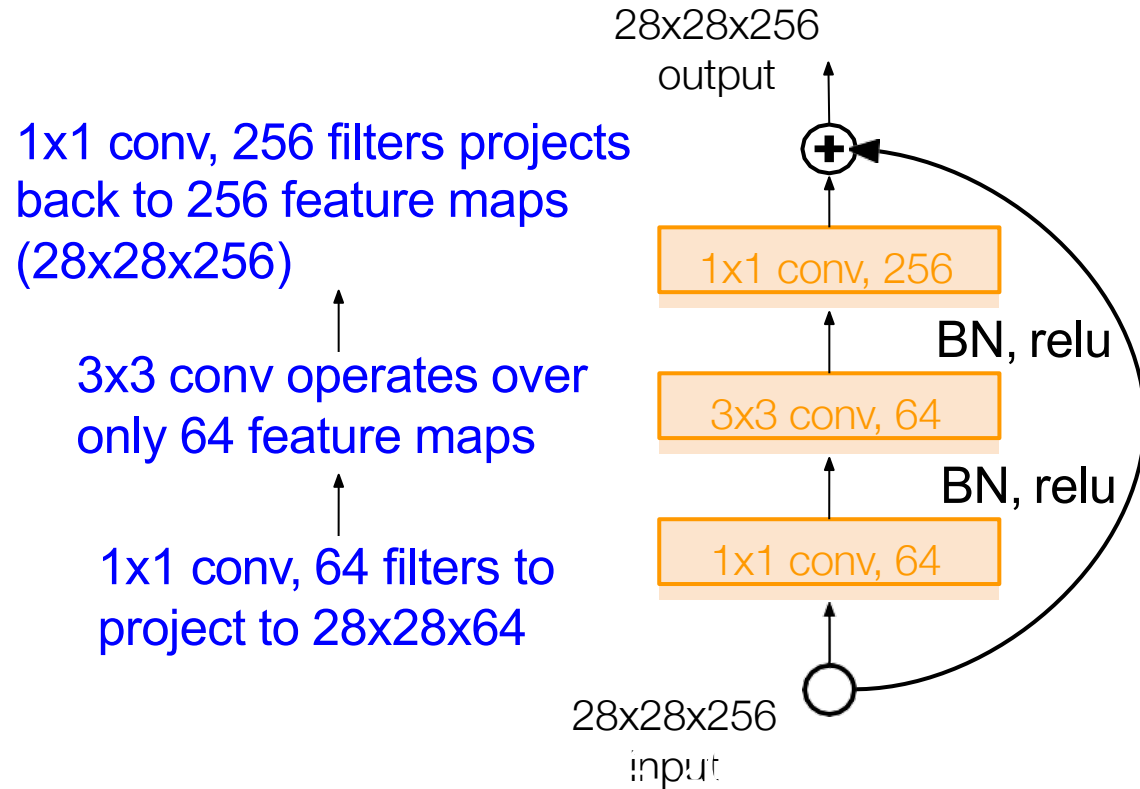
For deeper networks
(ResNet-50+), use “bottleneck”
layer to improve efficiency
(similar to GoogLeNet)



Case Study: ResNet

[He et al., 2015]

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



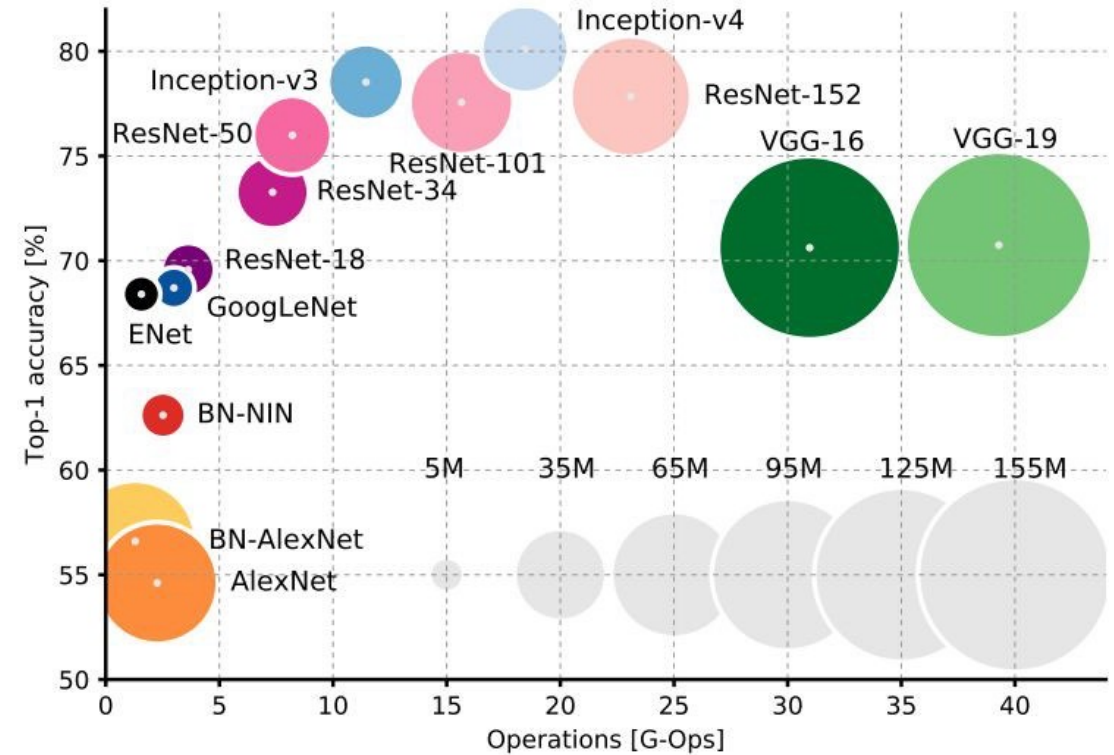
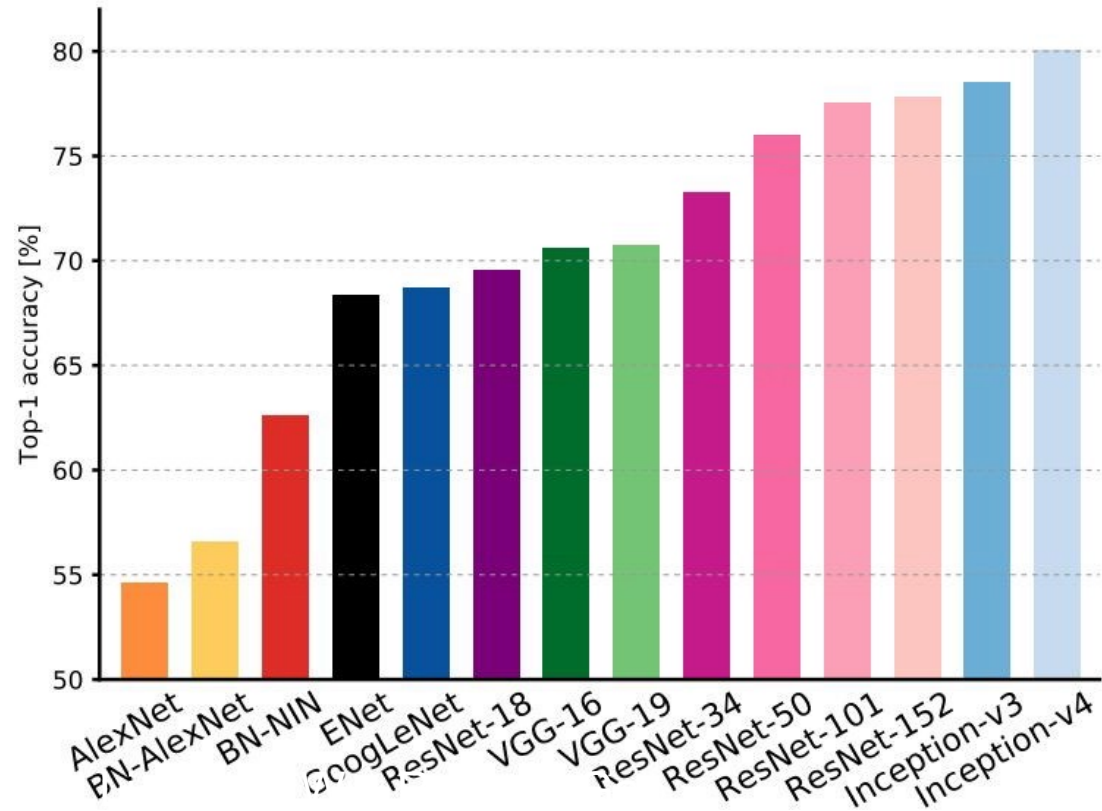
Case Study: ResNet

[He et al., 2015]

Training ResNet in practice:

- Batch Normalization after every CONV layer
- Xavier initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of $1e-5$
- No dropout used

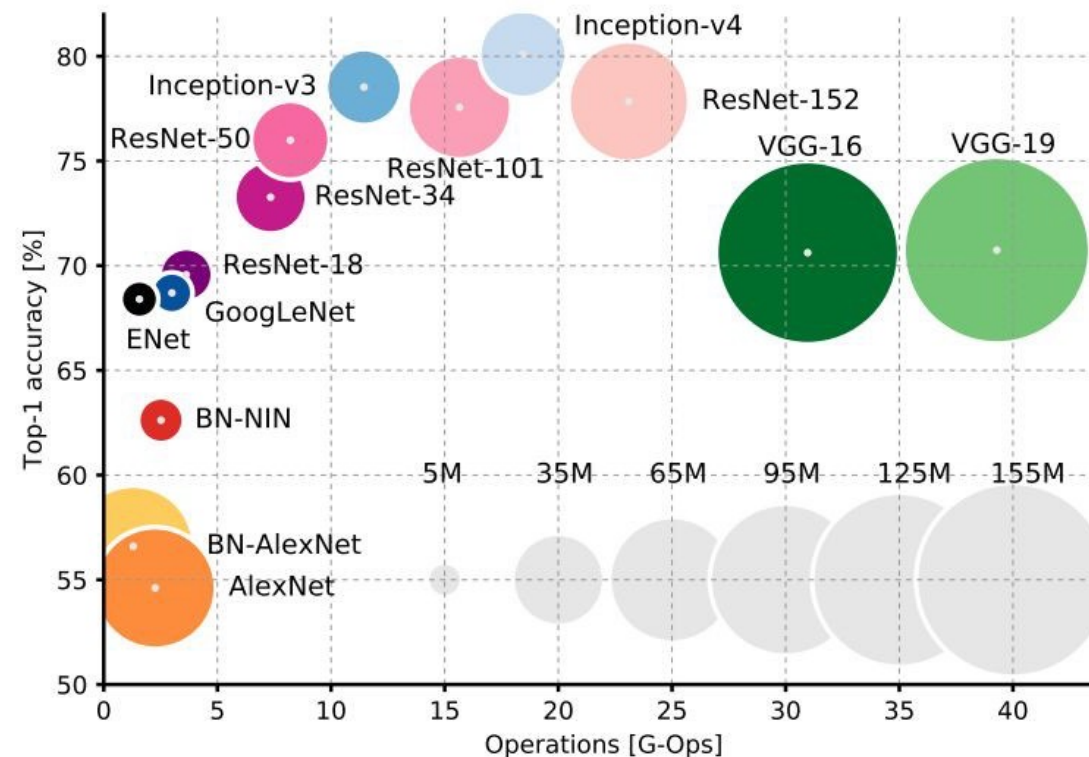
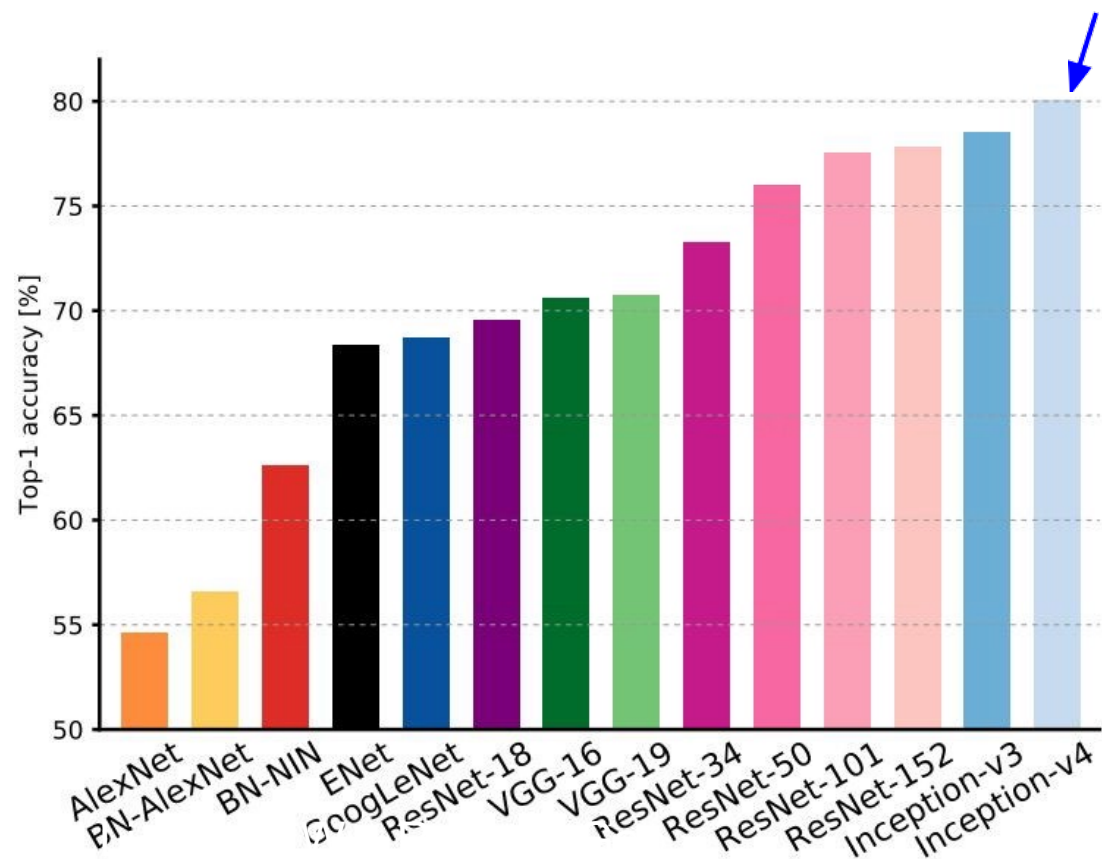
Comparing complexity...



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

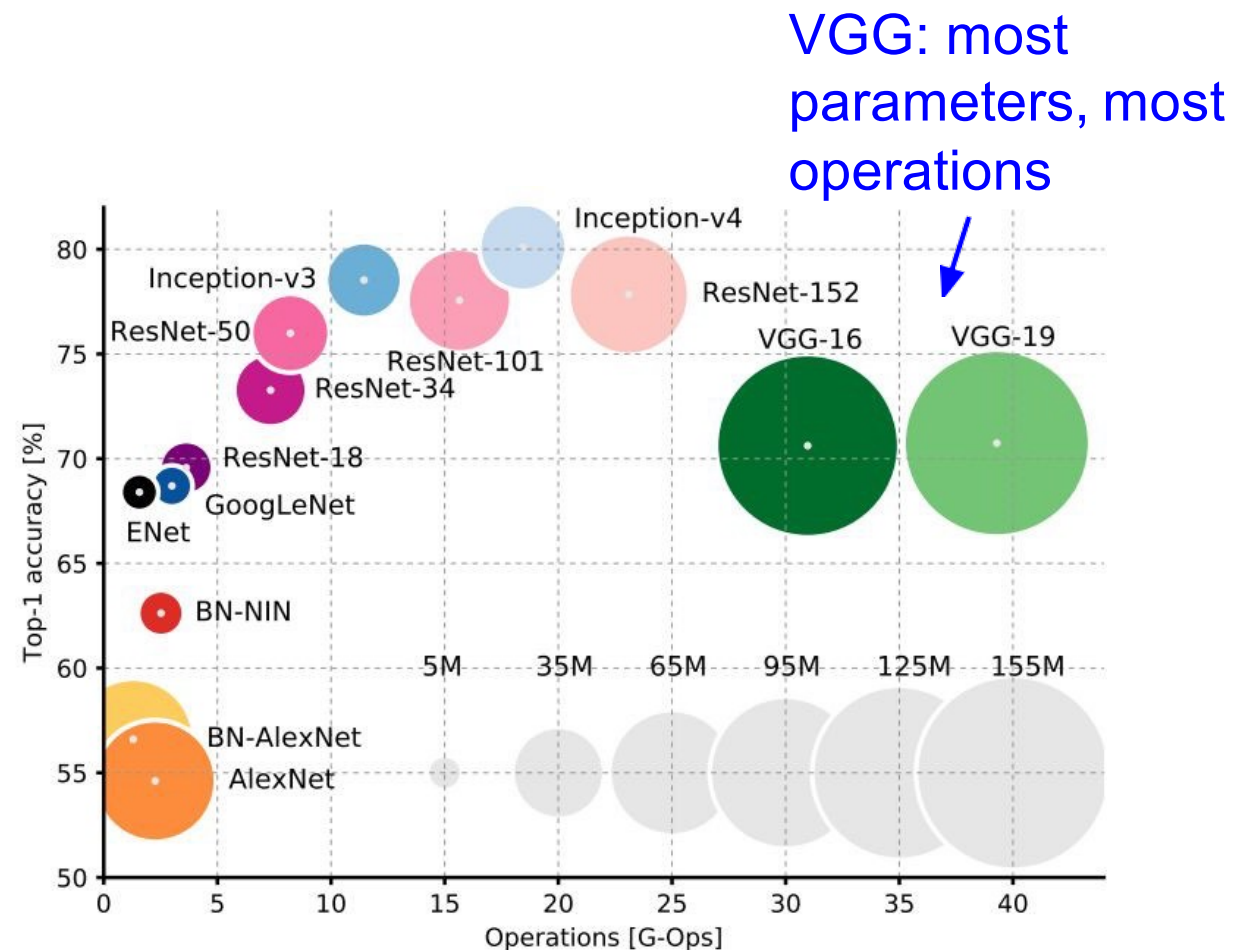
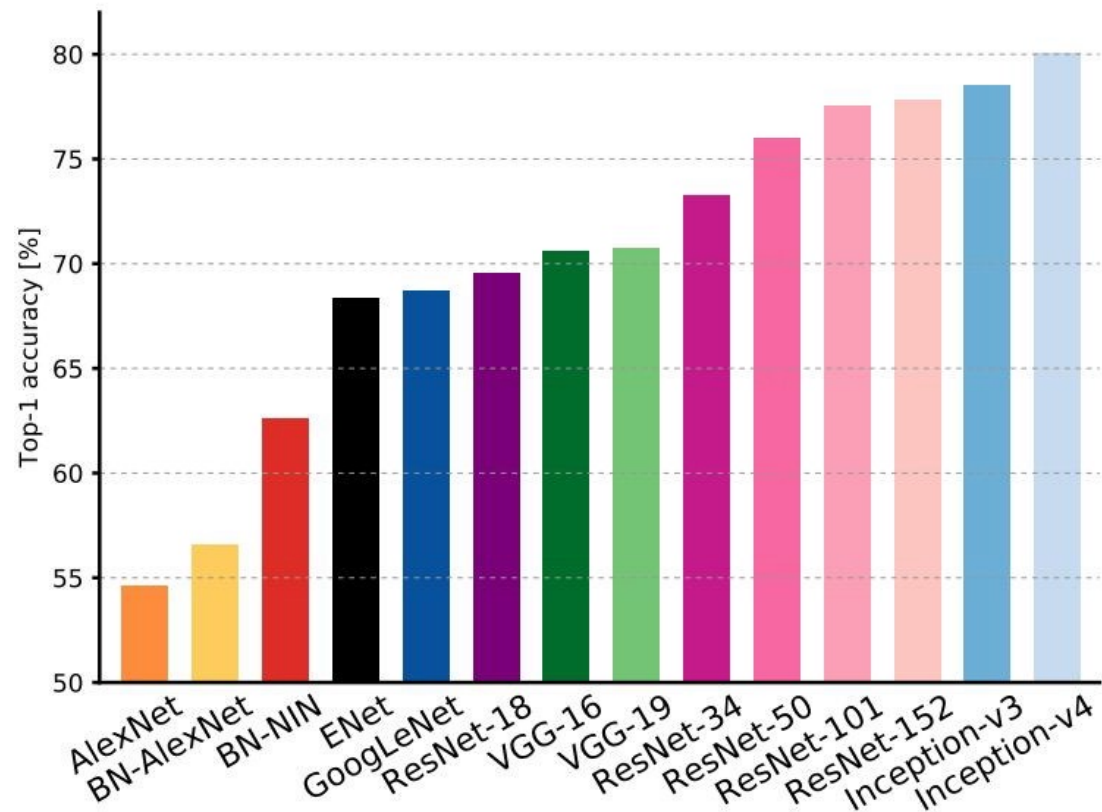
Comparing complexity...

Inception-v4: Resnet + Inception!



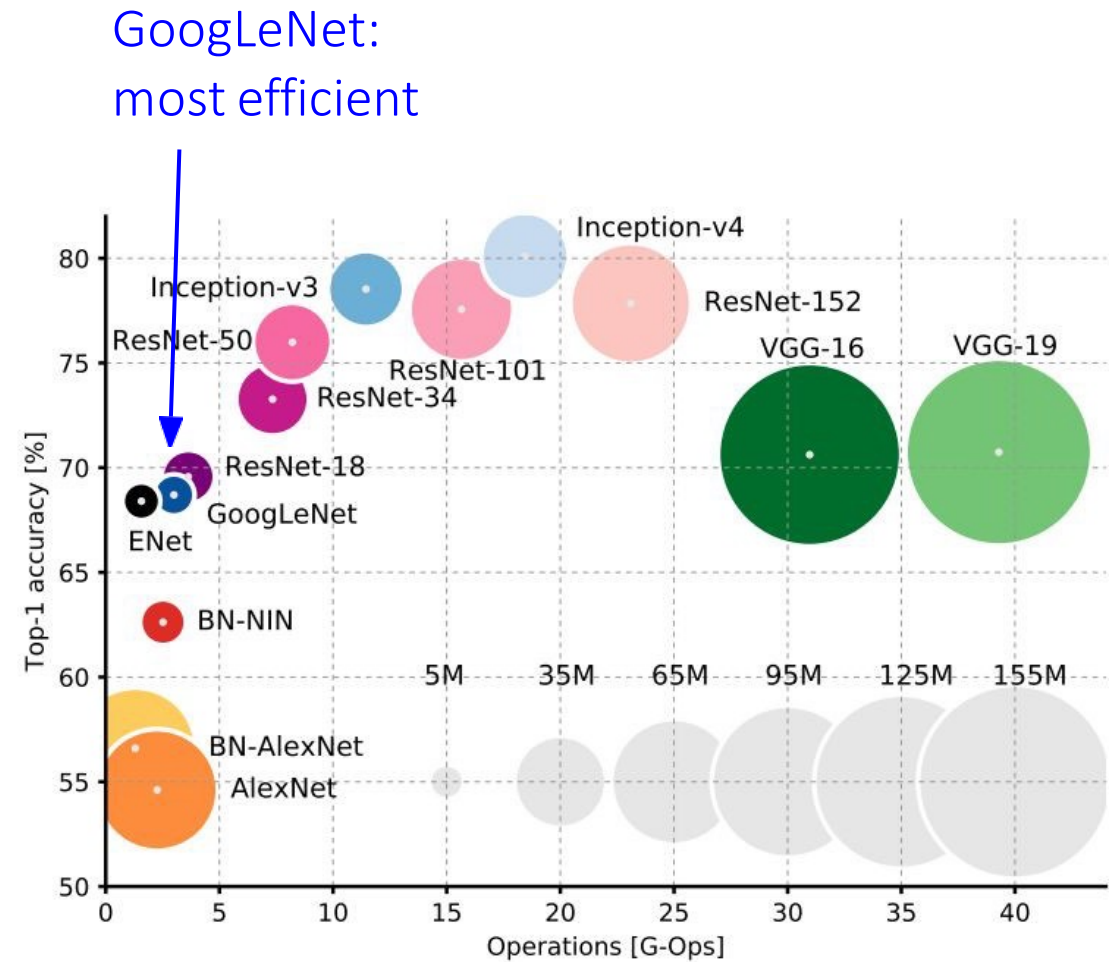
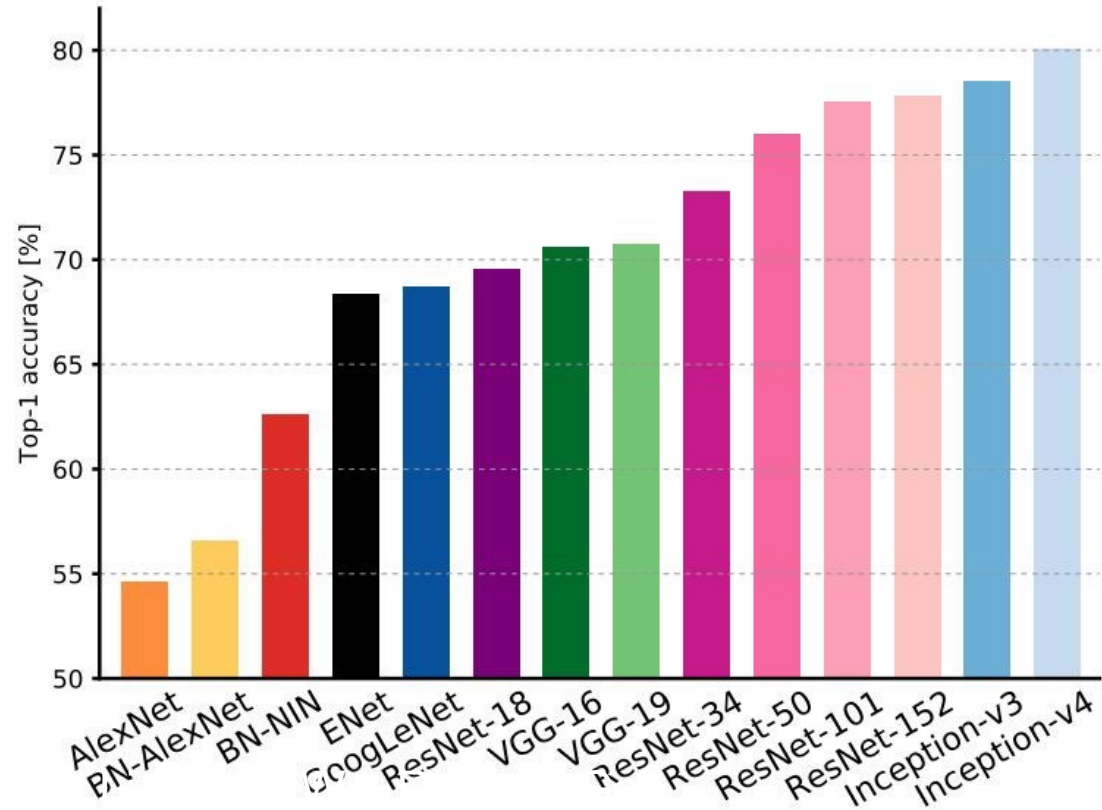
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Comparing complexity...



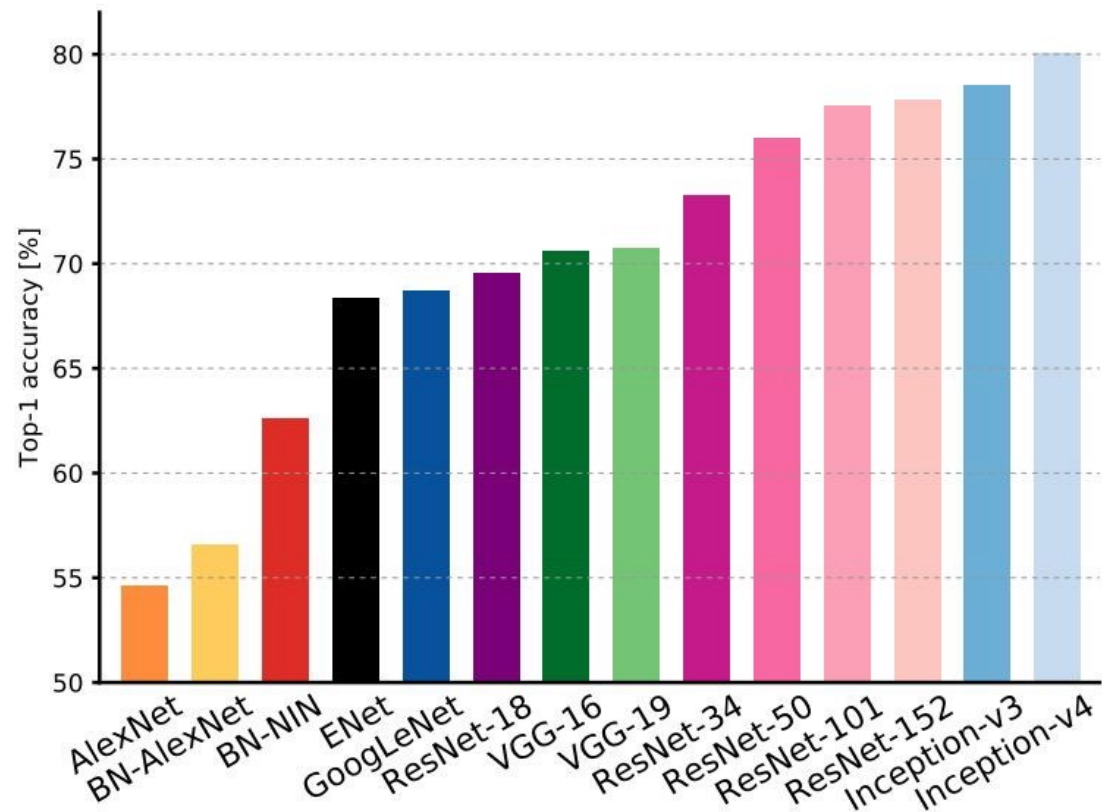
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Comparing complexity...

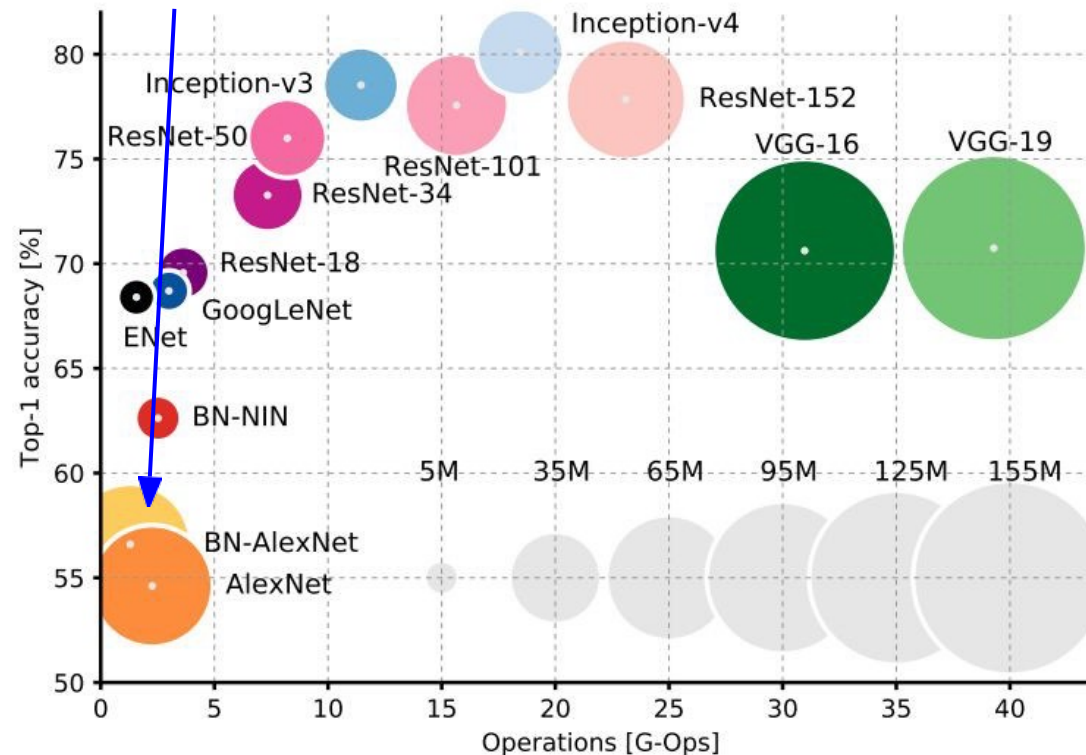


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Comparing complexity...

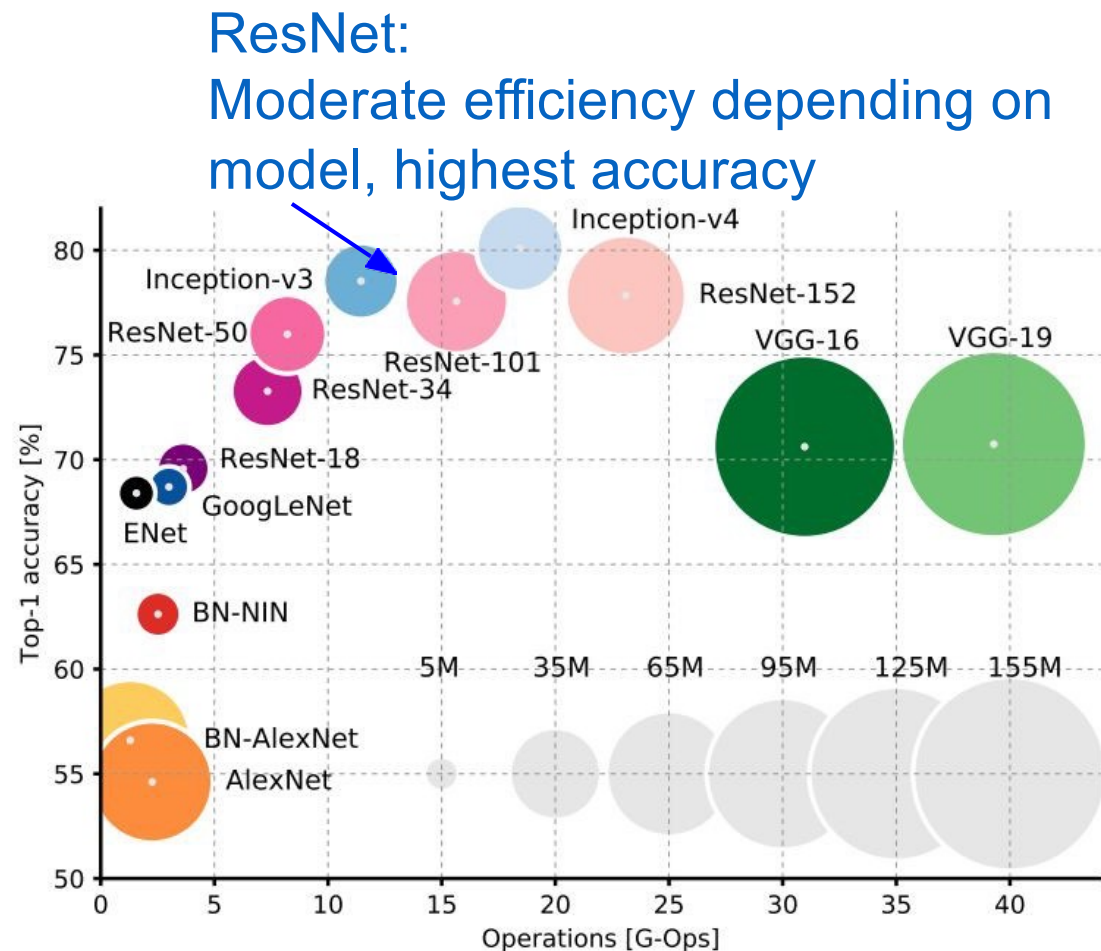
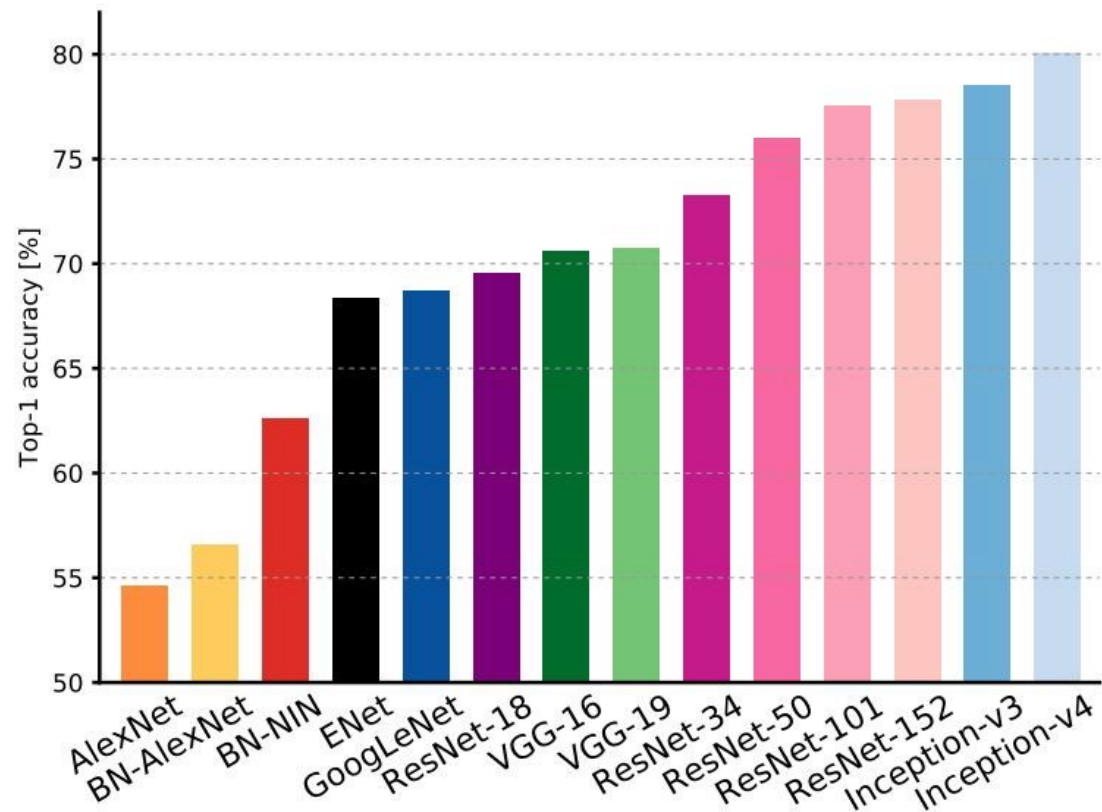


AlexNet:
Smaller compute, still memory
heavy, lower accuracy



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

Comparing complexity...



An Analysis of Deep Neural Network Models for Practical Applications, 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**, **ShuffleNet**

Neural Architecture Search can now automate architecture design

Summary: CNN Architectures

- Many popular architectures are available in model zoos.
- ResNets are currently good defaults to use.
- Networks have gotten increasingly deep over time.
- Many other aspects of network architectures are also continuously being investigated and improved.