Lecture 12:
Modern Architectures

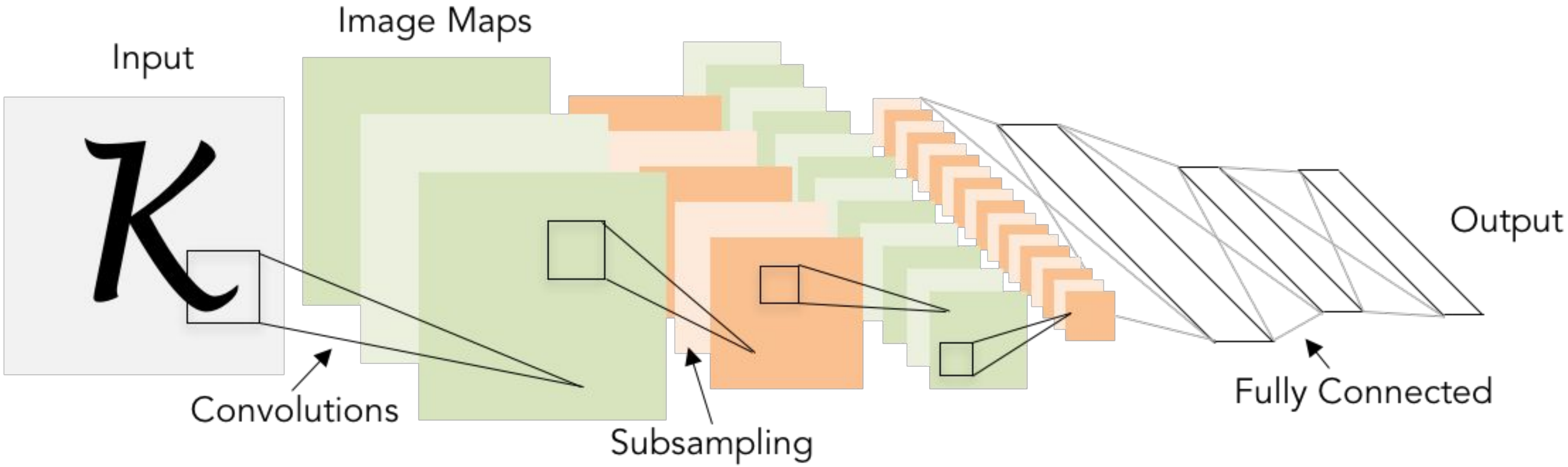
Administrative

- A3 is due tonight
- Quiz 3 on Friday (covers up to this lecture)

Today: Modern Architectures

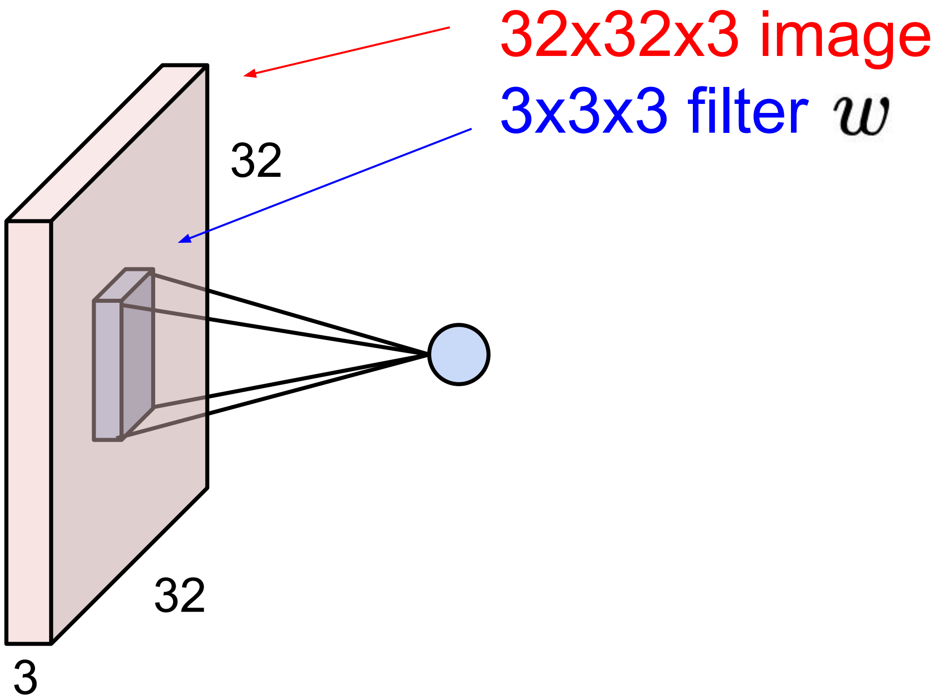
Review: LeNet-5

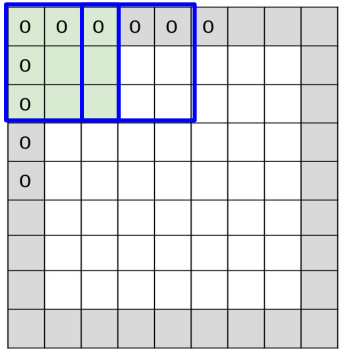
[LeCun et al., 1998]



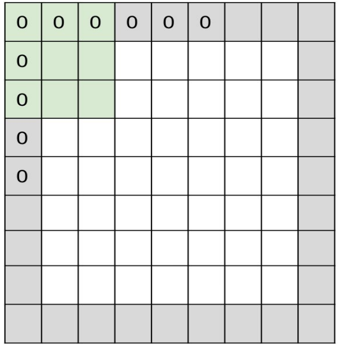
Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

Review: Convolution



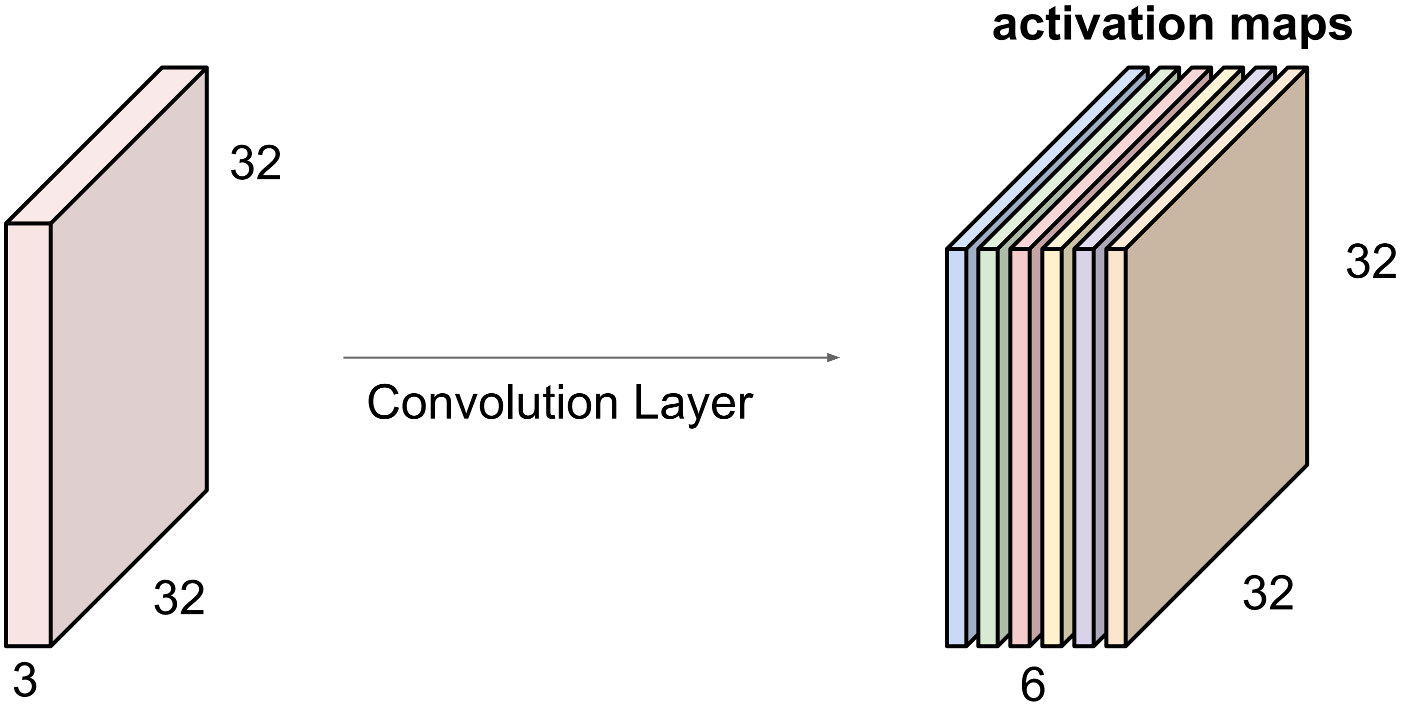


Stride:
Downsample output activations



Padding:
Preserve input spatial dimensions in output activations

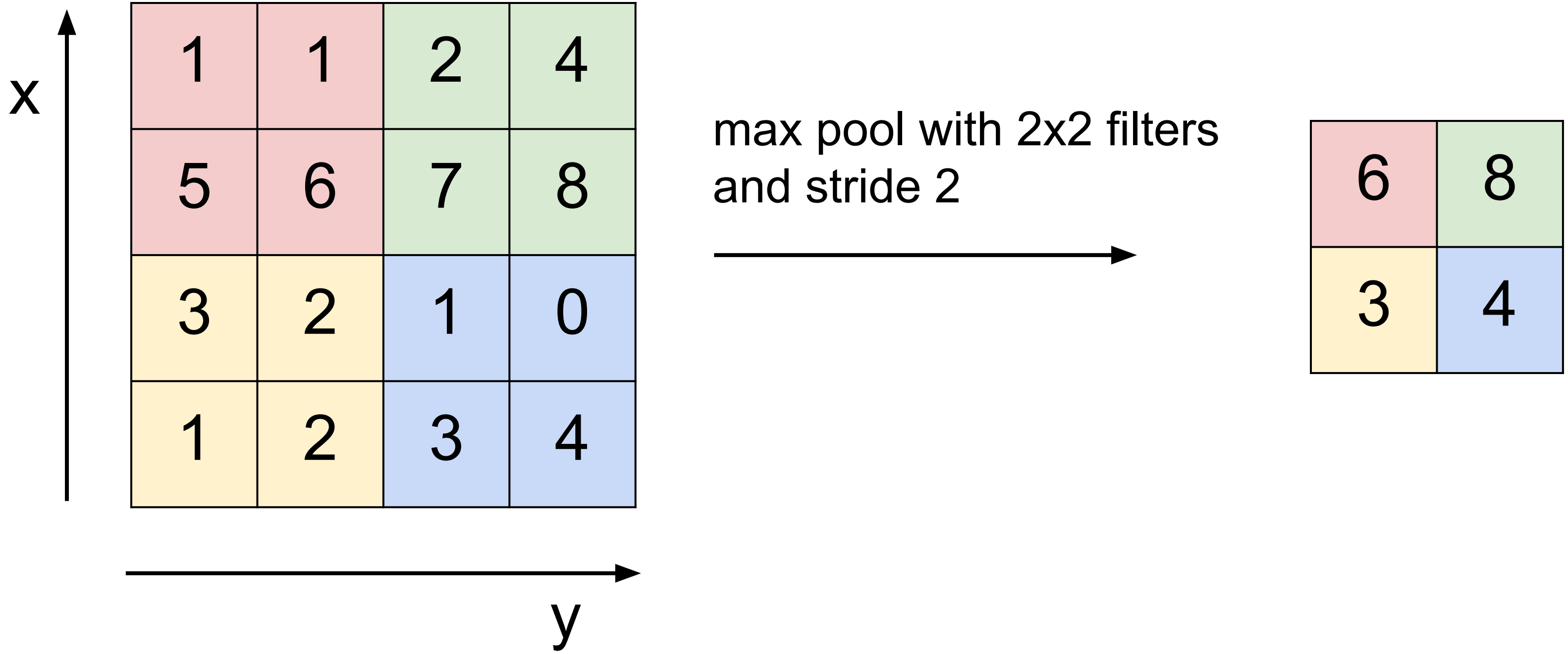
Review: Convolution



Each conv filter outputs a “slice” in the activation

Review: Pooling

Single depth slice



Today: Modern Architectures

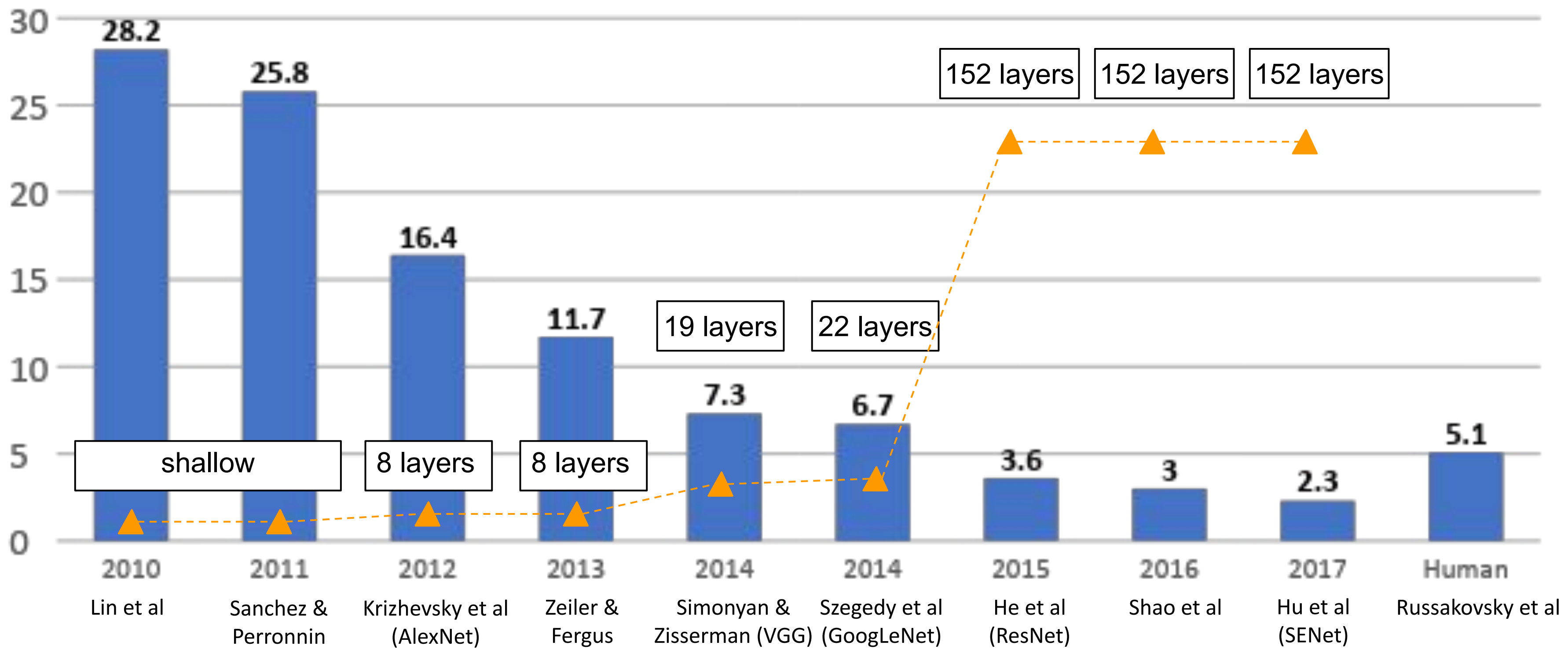
Case Studies

- AlexNet
- VGG
- GoogLeNet
- ResNet
- ViT
- MLP Mixers

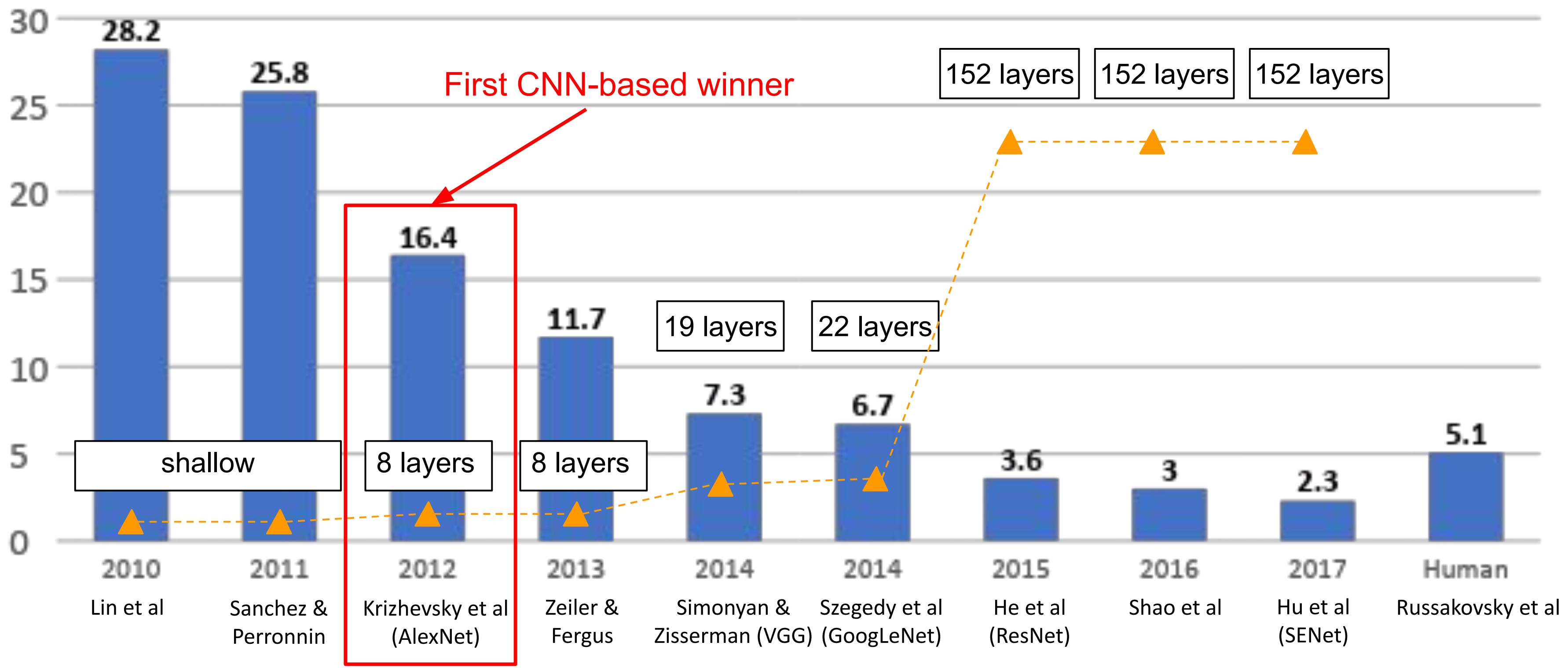
Also....

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

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

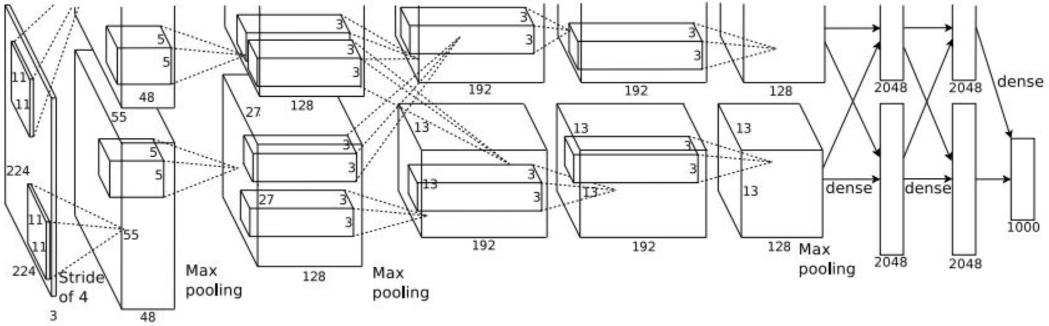


ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Case Study: AlexNet

[Krizhevsky et al. 2012]

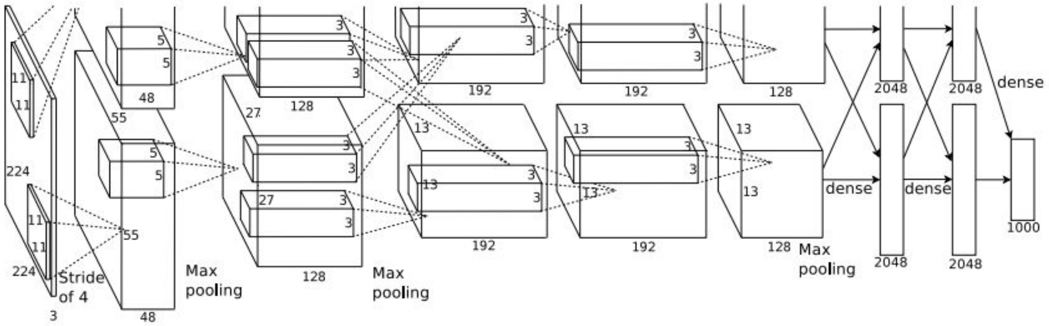


Architecture:

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

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

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

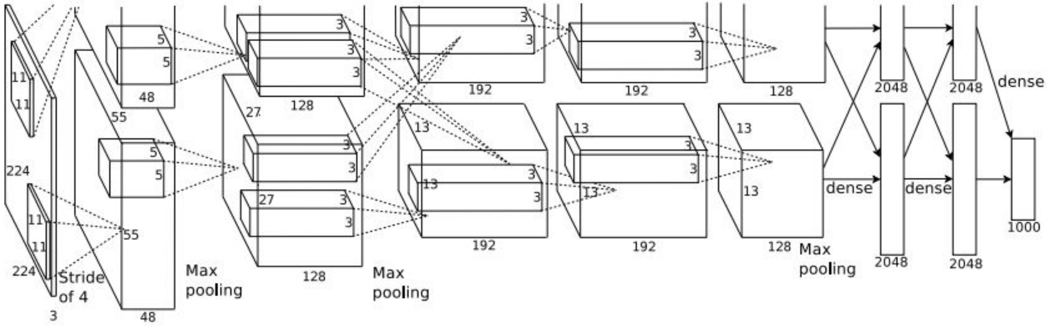
=>

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]



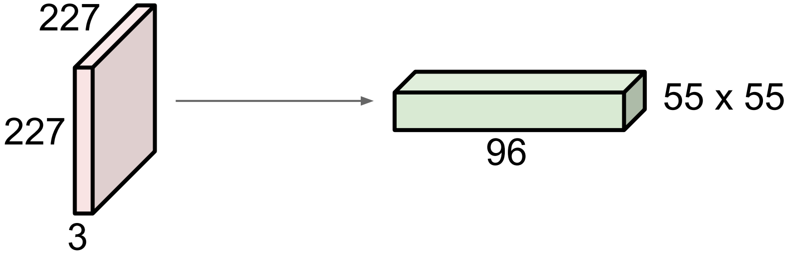
Input: 227x227x3 images

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

=>

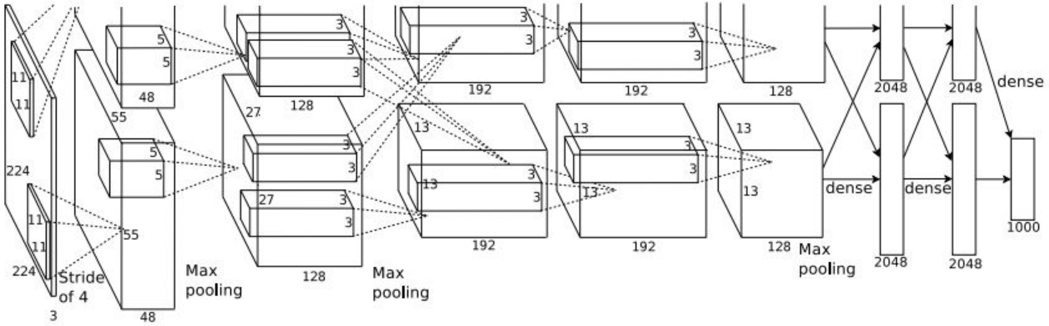
Output volume **[55x55x96]**

W’ = (W - F + 2P) / S + 1



Case Study: AlexNet

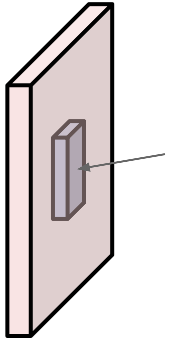
[Krizhevsky et al. 2012]



Input: 227x227x3 images

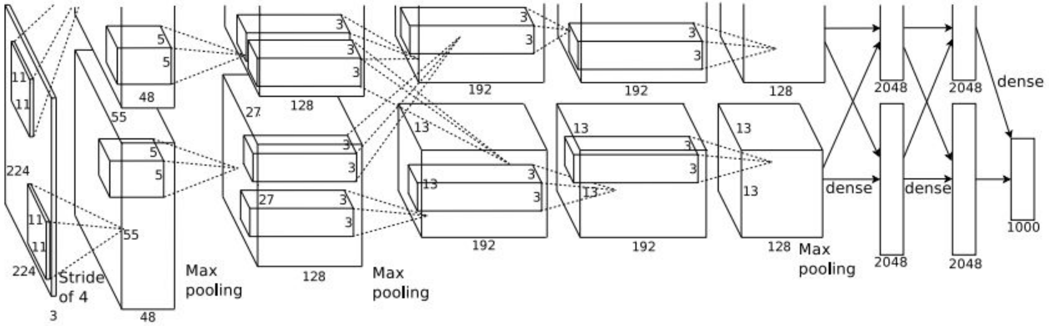
First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

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



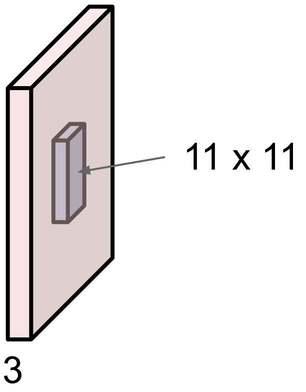
Case Study: AlexNet

[Krizhevsky et al. 2012]



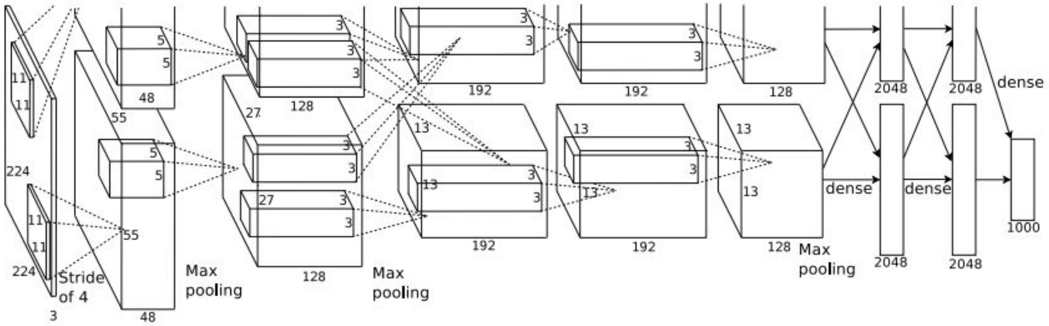
Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3 + 1)*96 = **35K**



Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96

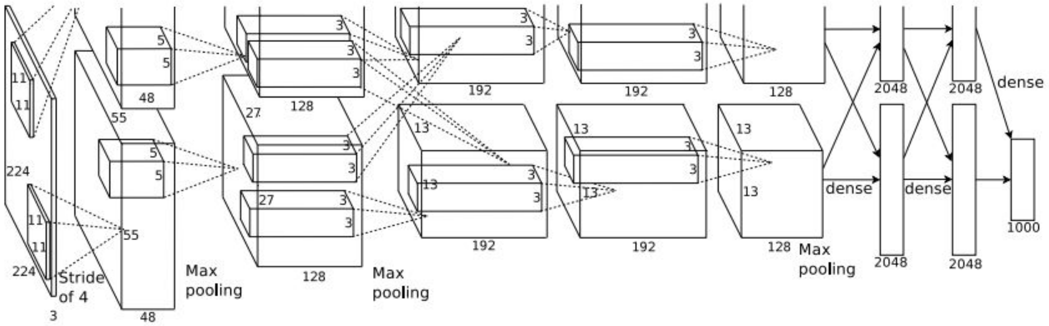
W' = (W - F + 2P) / S + 1

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

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

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96

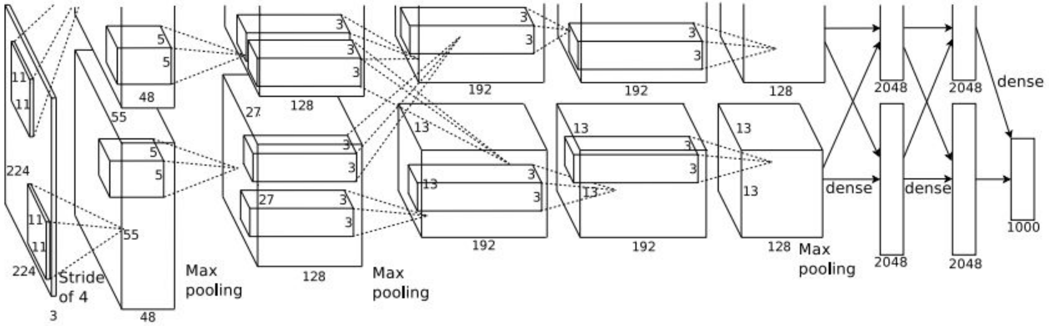
W' = (W - F + 2P) / S + 1

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

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

Case Study: AlexNet

[Krizhevsky et al. 2012]

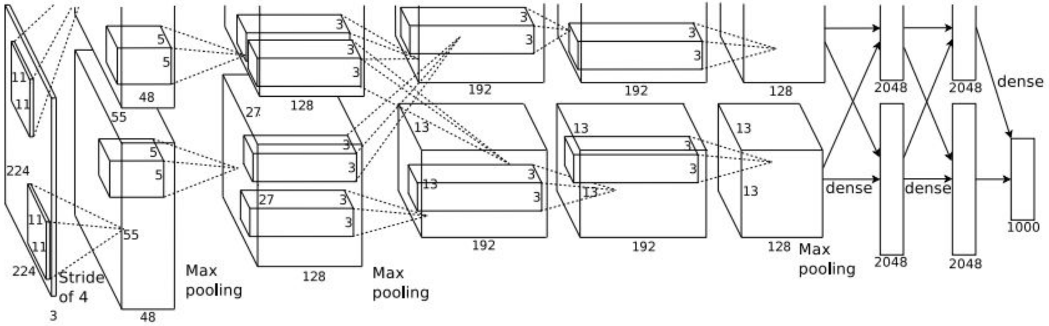


Input: 227x227x3 images
After CONV1: 55x55x96

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

Case Study: AlexNet

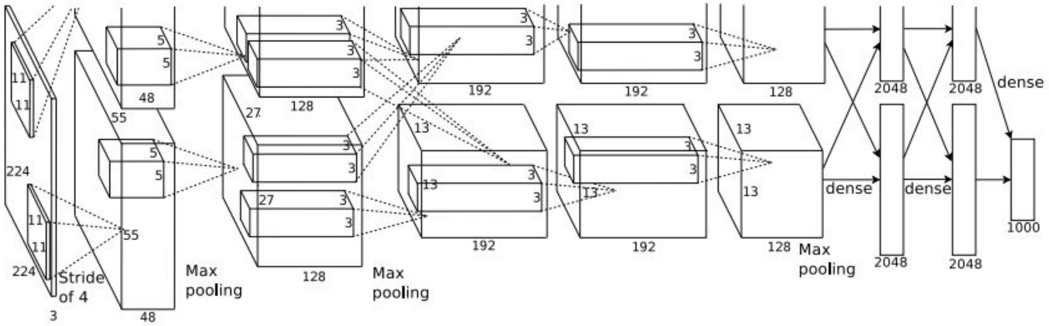
[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...

Case Study: AlexNet

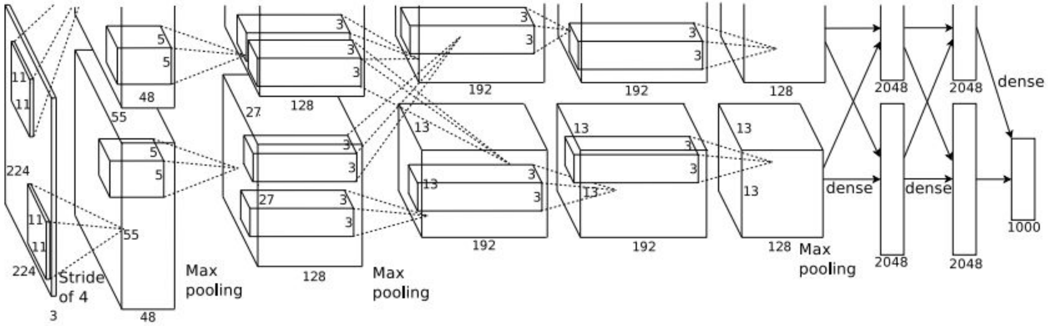
[Krizhevsky et al. 2012]



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)

Case Study: AlexNet

[Krizhevsky et al. 2012]

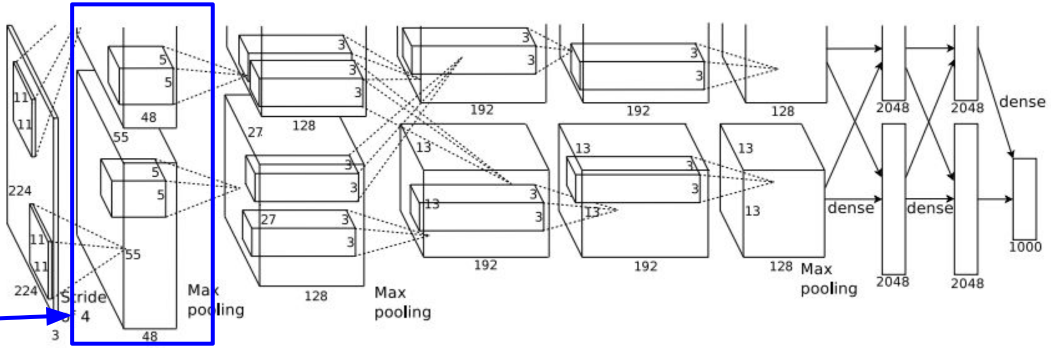


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] 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%

Case Study: AlexNet

[Krizhevsky et al. 2012]



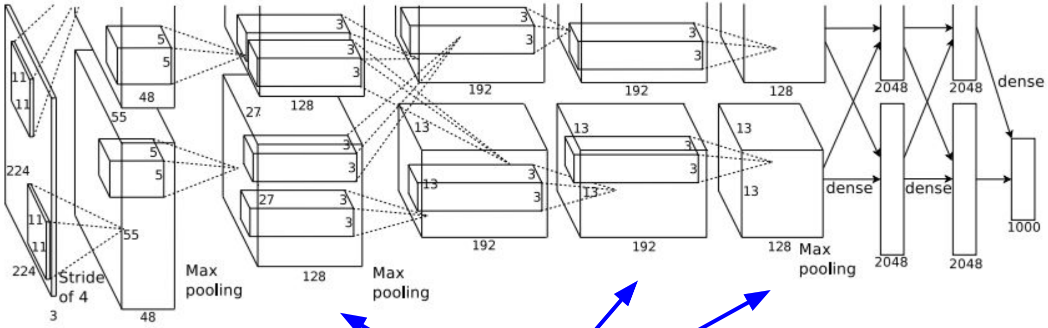
[55x55x48] x 2

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.

Case Study: AlexNet

[Krizhevsky et al. 2012]

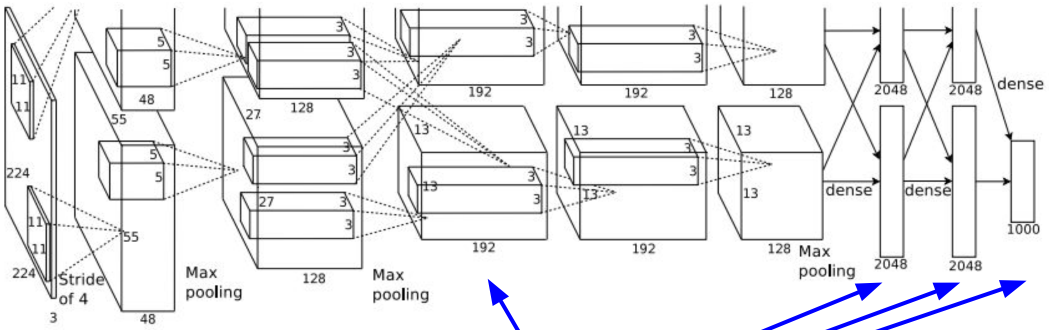


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] 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

Case Study: AlexNet

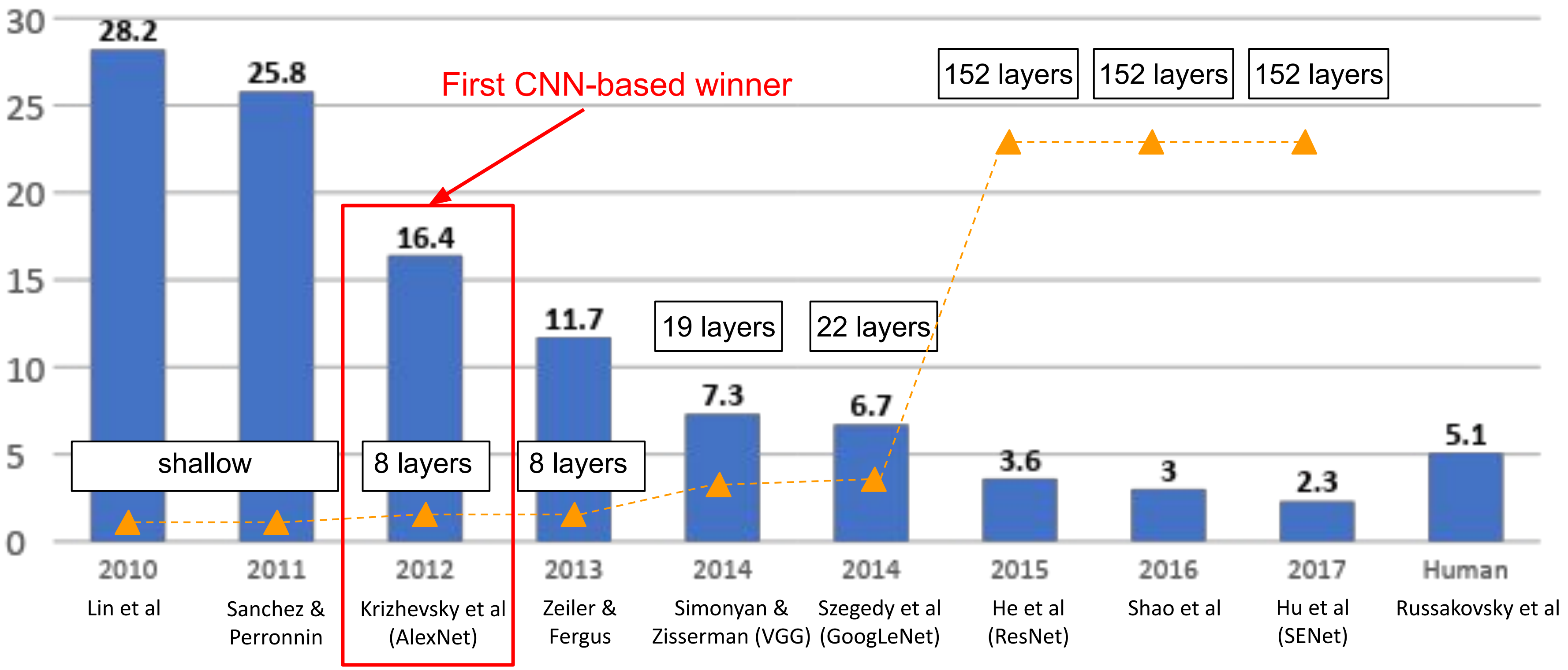
[Krizhevsky et al. 2012]



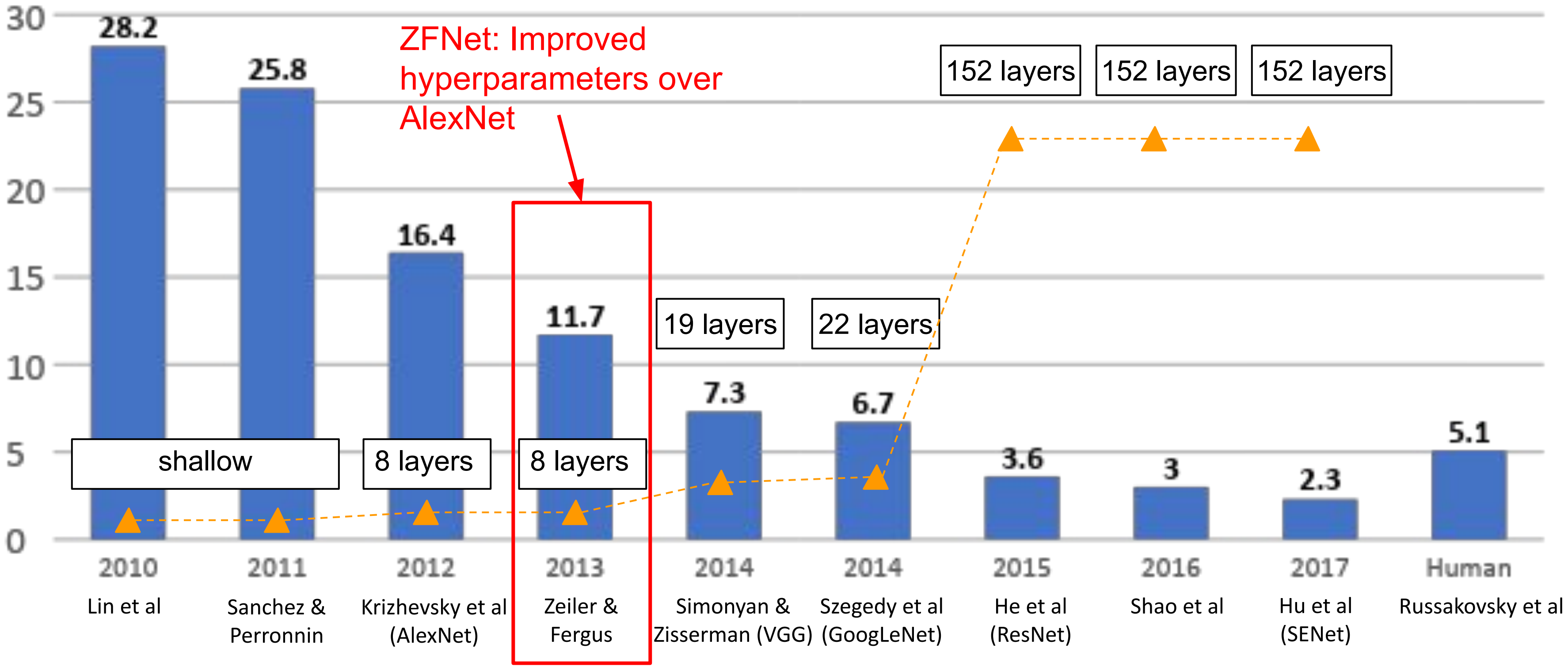
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] 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

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

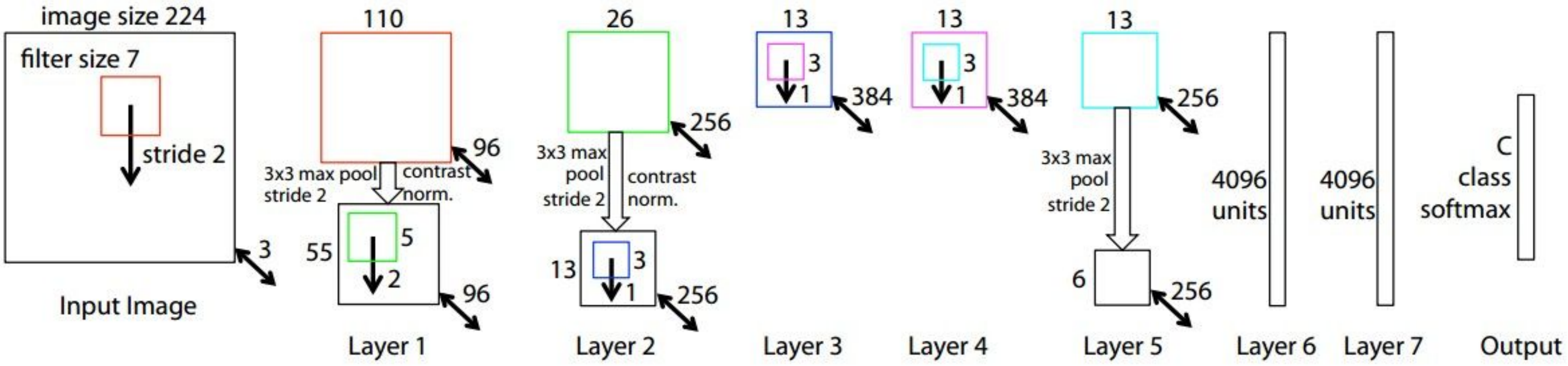


ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ZFNet

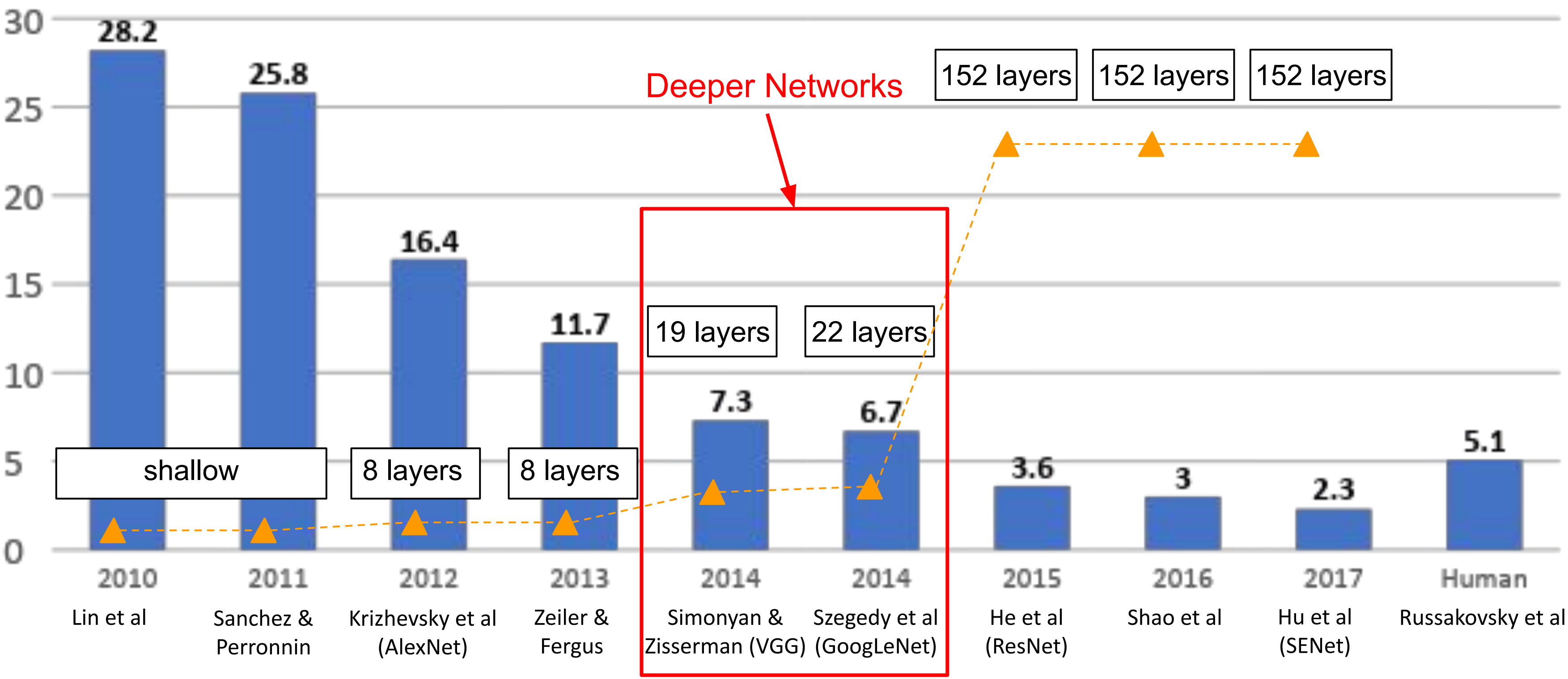
[Zeiler and Fergus, 2013]



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% -> 11.7%

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Case Study: VGGNet

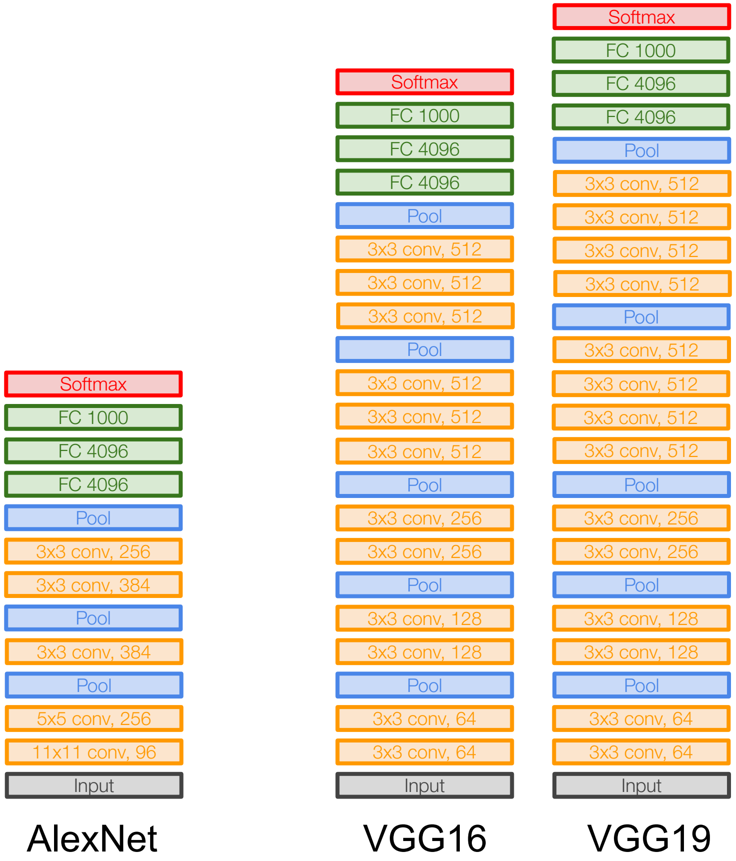
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

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

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

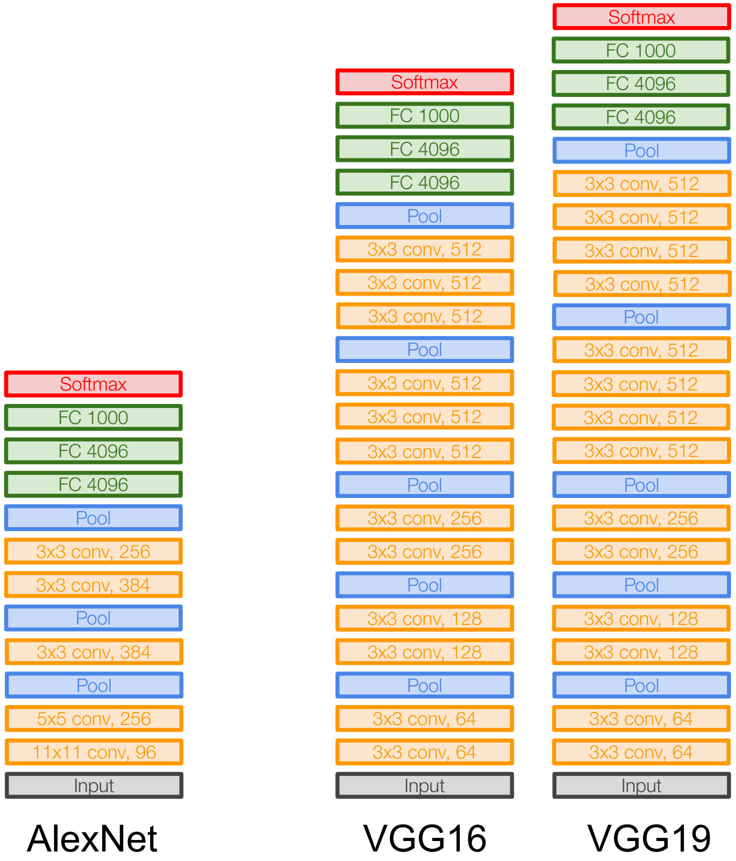
11.7% top 5 error in ILSVRC'13 (ZFNet)
-> 7.3% top 5 error in ILSVRC'14



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

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



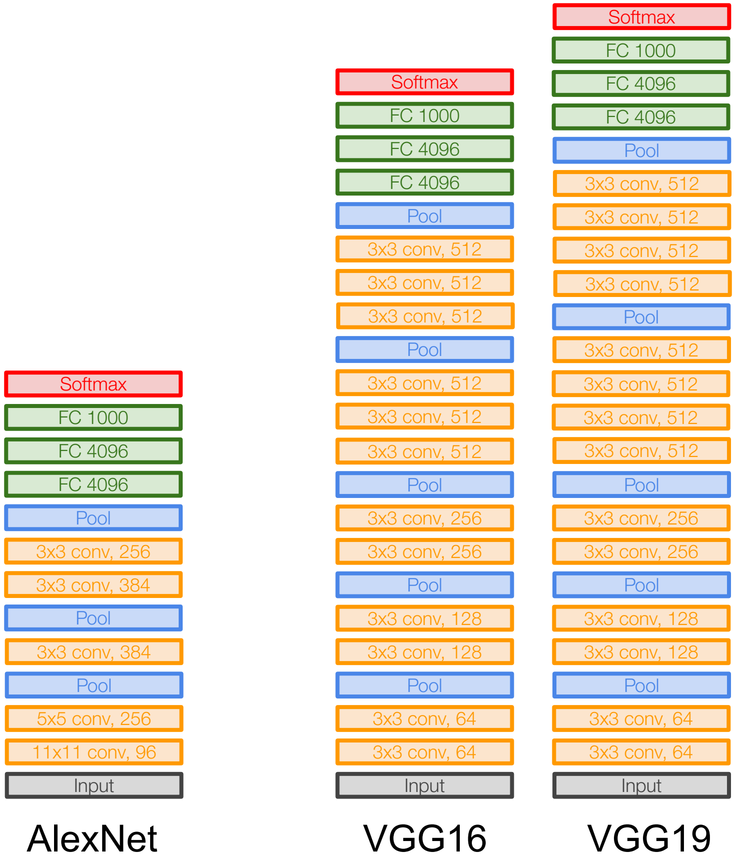
Case Study: VGGNet

[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

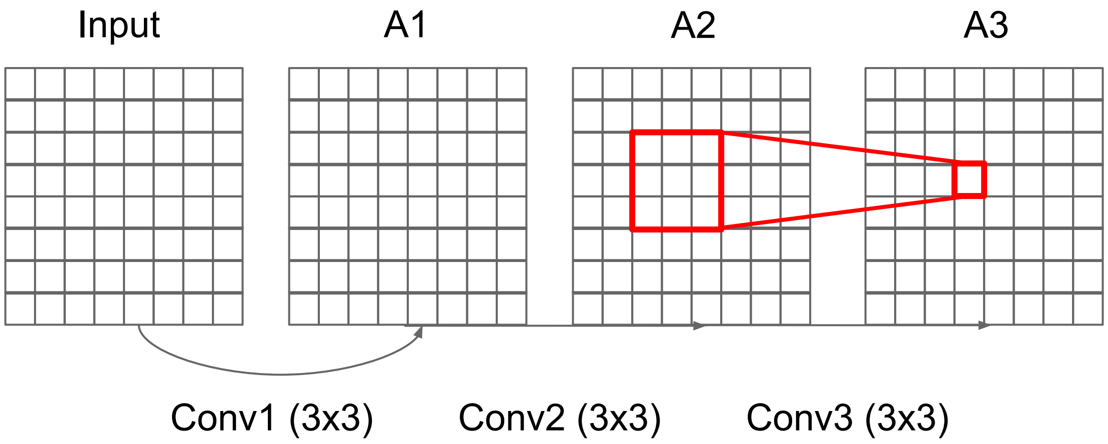
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

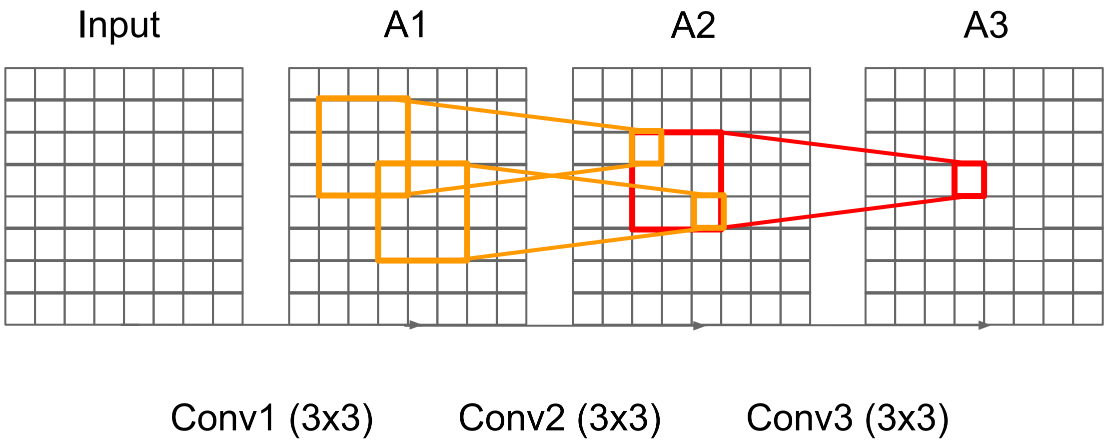
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

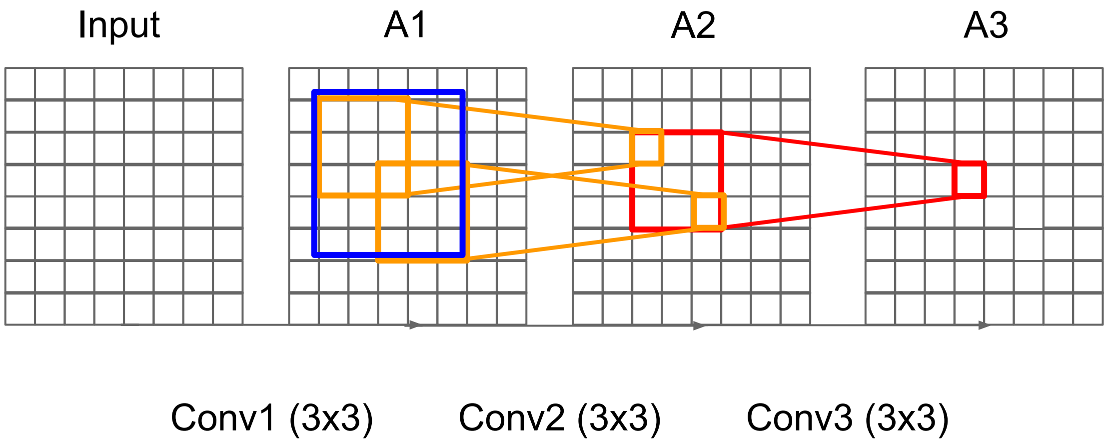
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

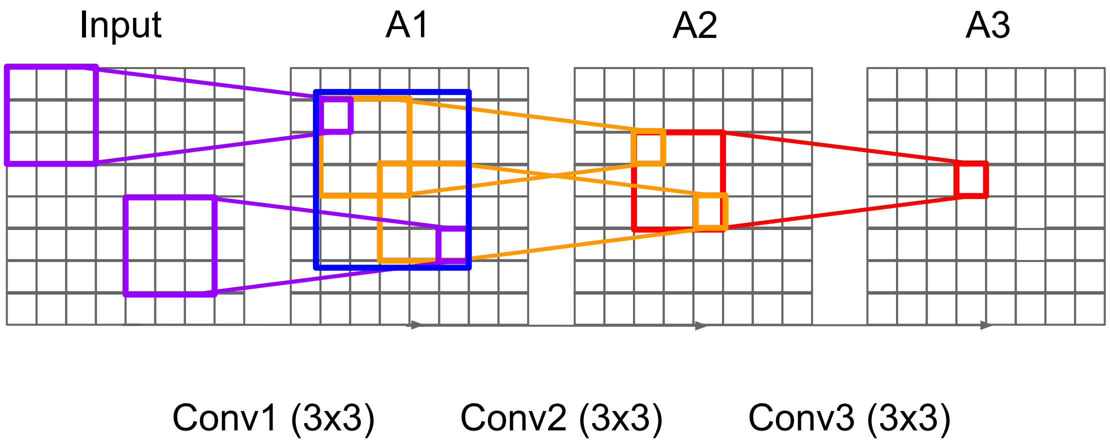
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

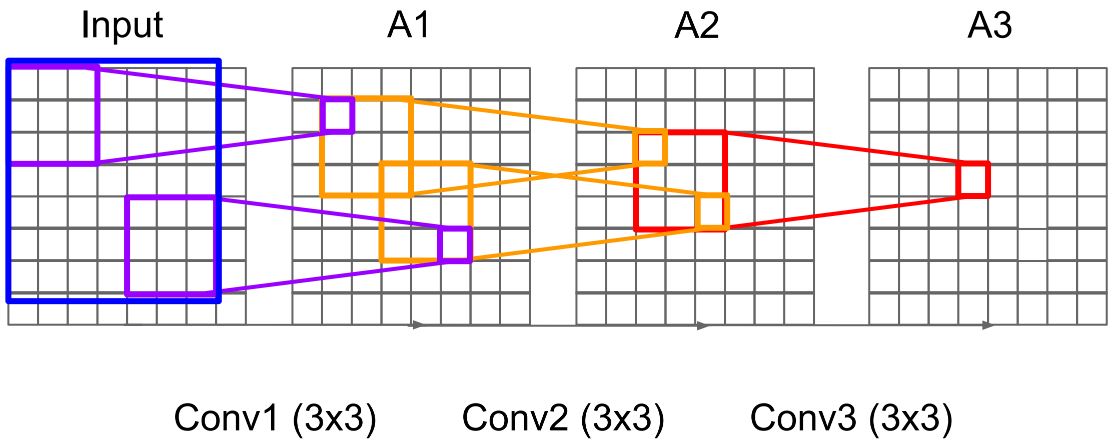
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

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



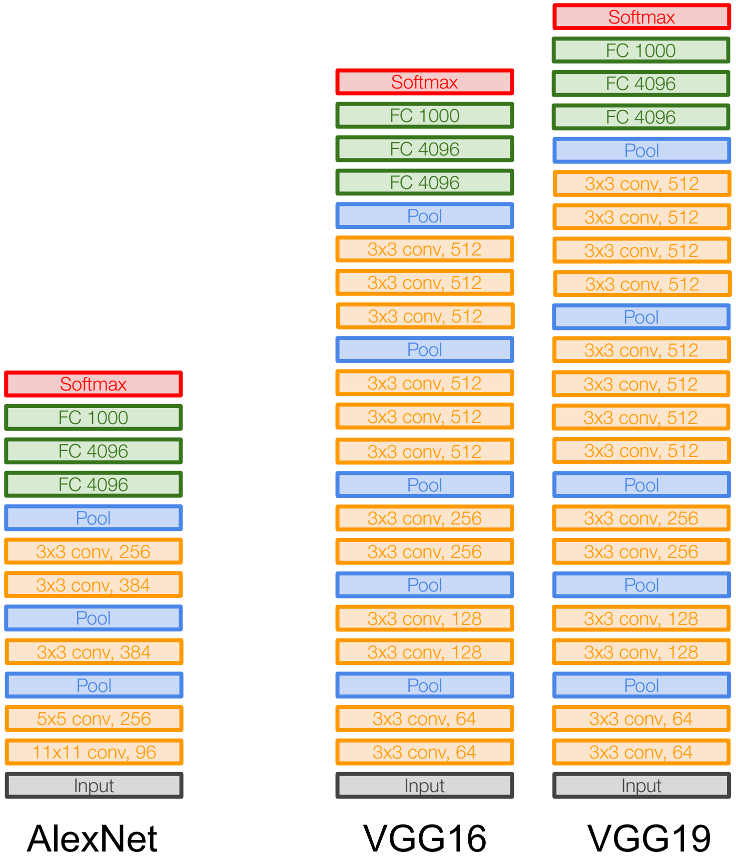
Case Study: VGGNet

[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

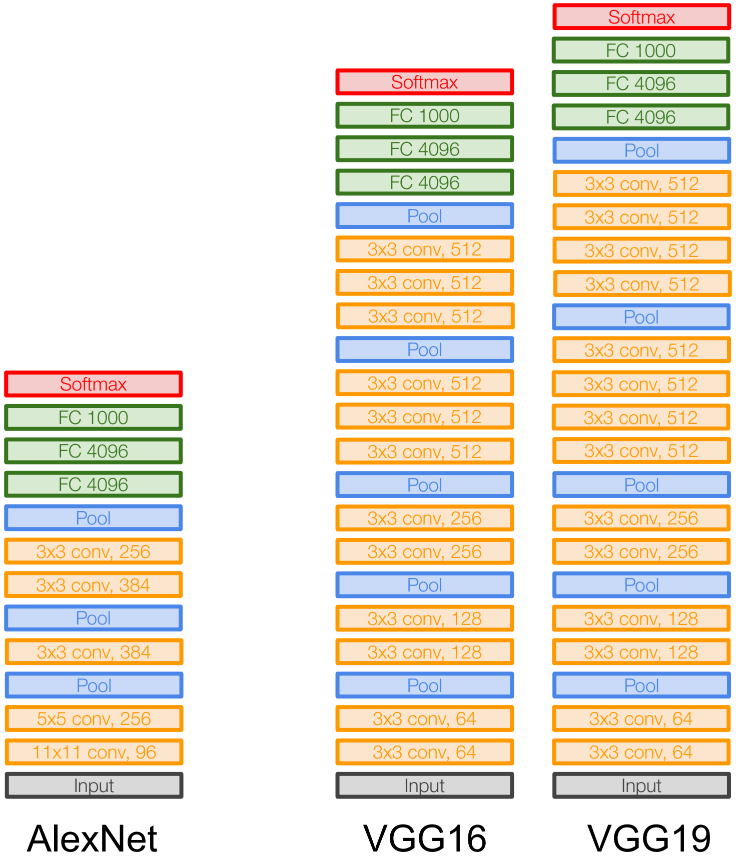
[Simonyan and Zisserman, 2014]

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



(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

VGG16

(not counting biases)

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

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

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

POOL2: [112x112x64] memory: 112*112*64=800K params: 0

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

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

POOL2: [56x56x128] memory: 56*56*128=400K params: 0

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

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

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

POOL2: [28x28x256] memory: 28*28*256=200K params: 0

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

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

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

POOL2: [14x14x512] memory: 14*14*512=100K params: 0

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

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

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

POOL2: [7x7x512] memory: 7*7*512=25K params: 0

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

FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216

FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 96MB / image (for a forward pass)

TOTAL params: 138M parameters

VGG16

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

Note:

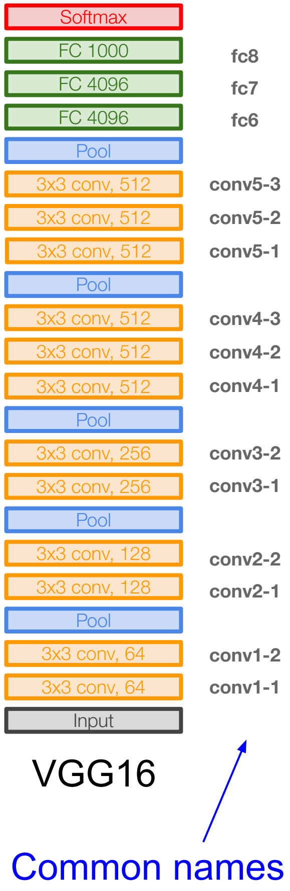
Most memory is in early CONV

Most params are in late FC

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

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

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

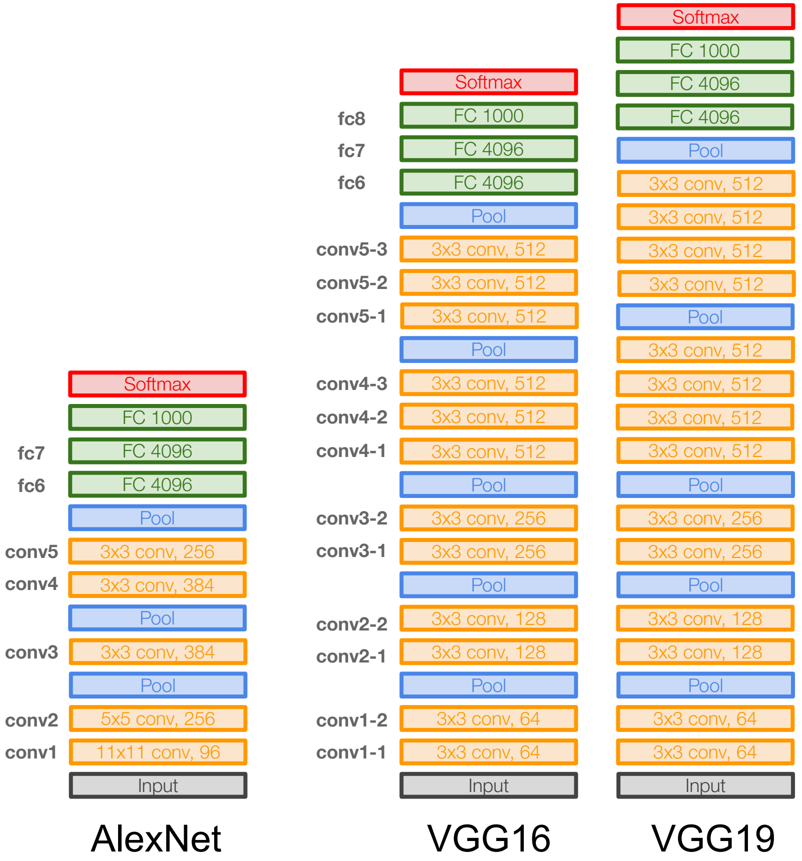


Case Study: VGGNet

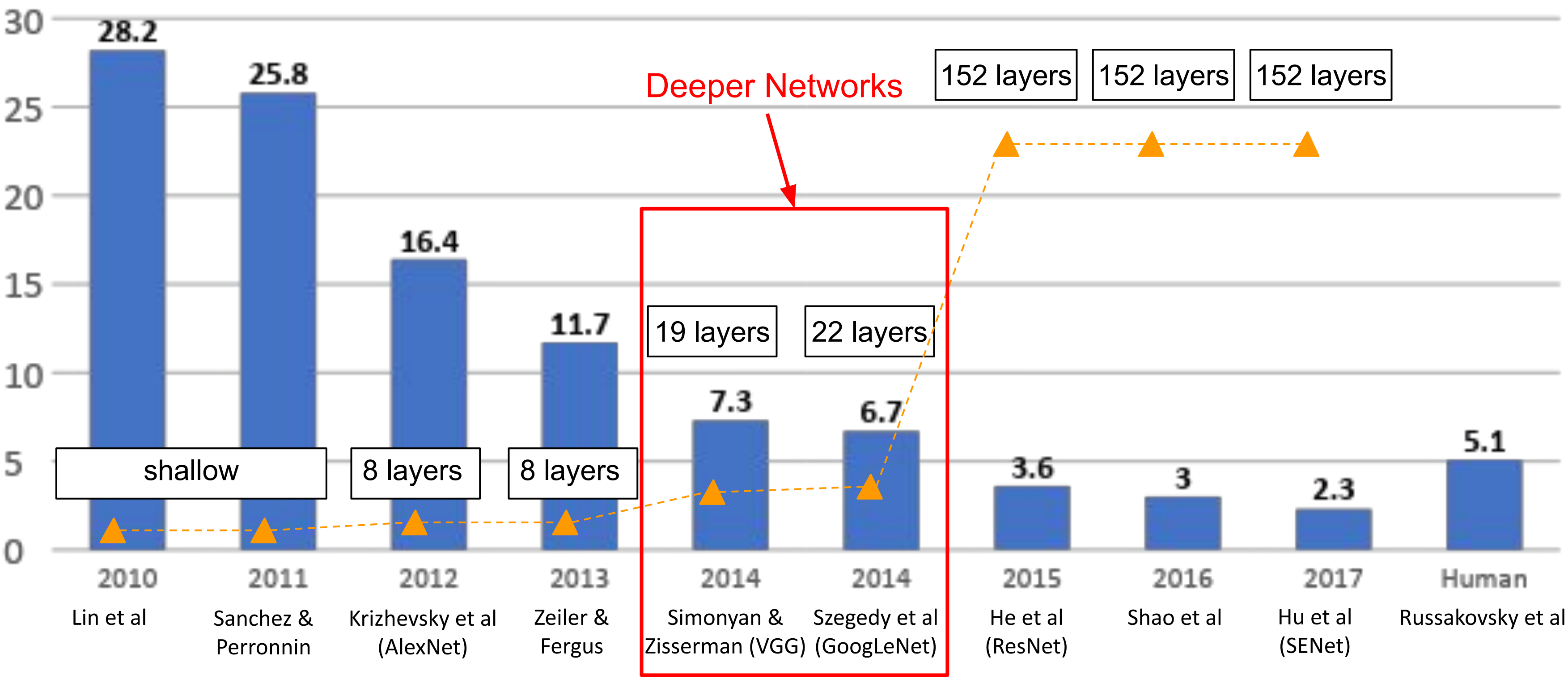
[Simonyan and Zisserman, 2014]

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



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

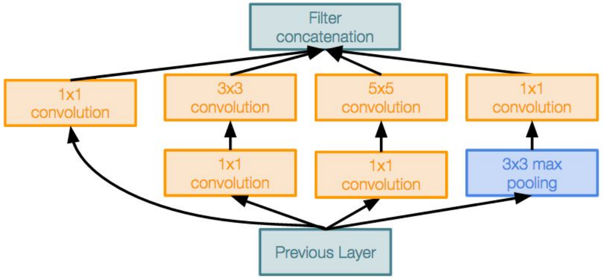


Case Study: GoogLeNet

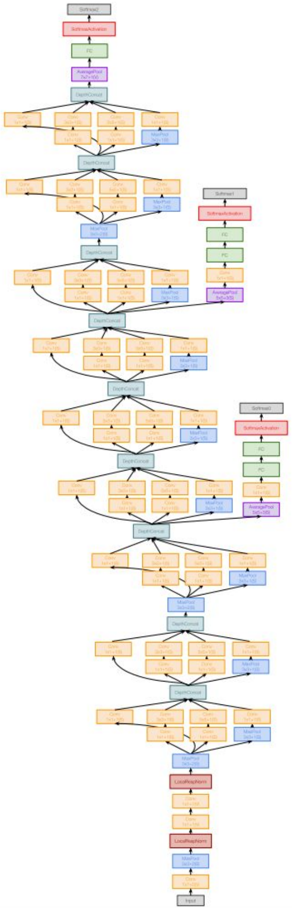
[Szegedy et al., 2014]

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



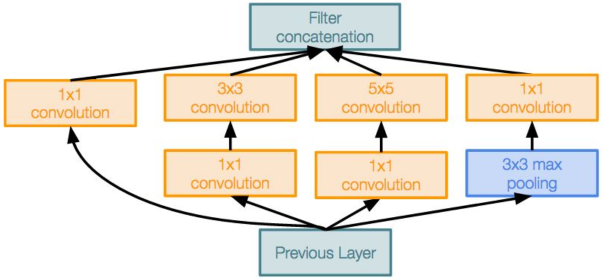
Inception module



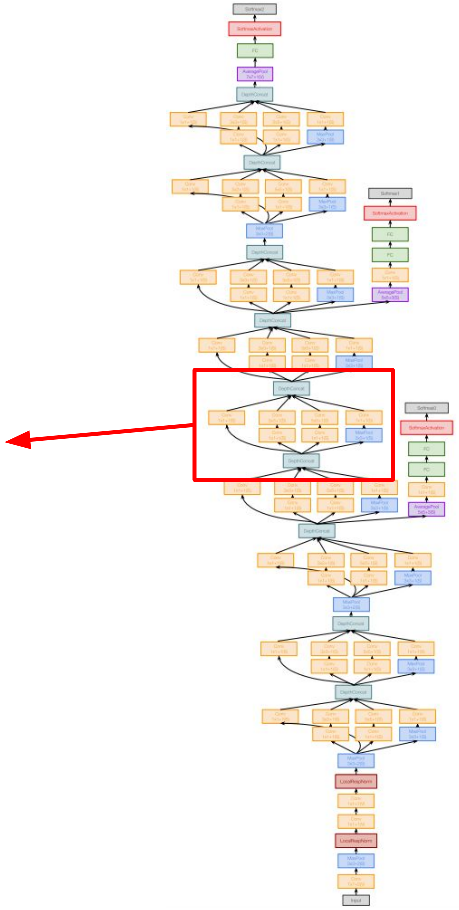
Case Study: GoogLeNet

[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

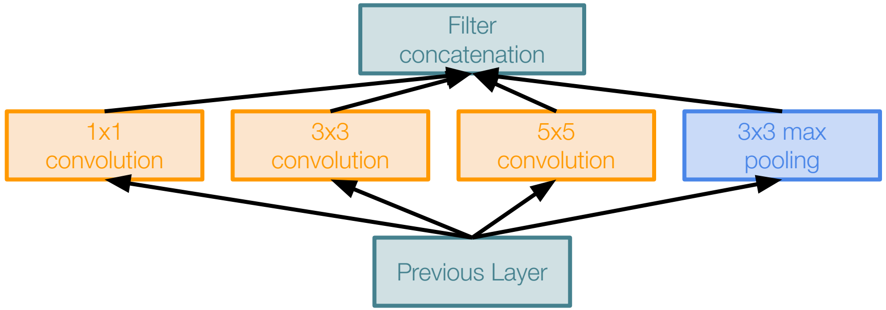


Inception module



Case Study: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

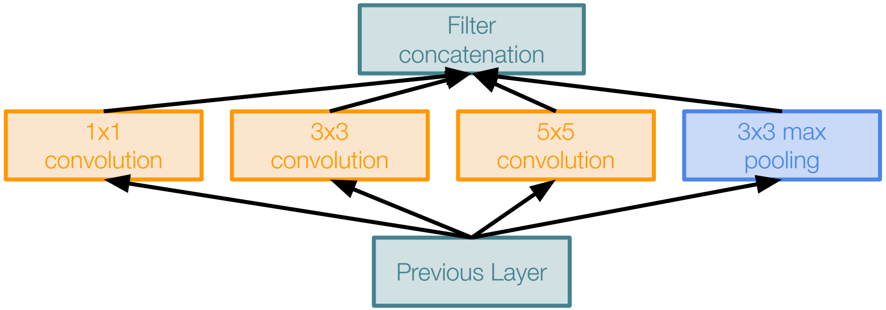
Apply parallel filter operations on the input from previous layer:

- 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

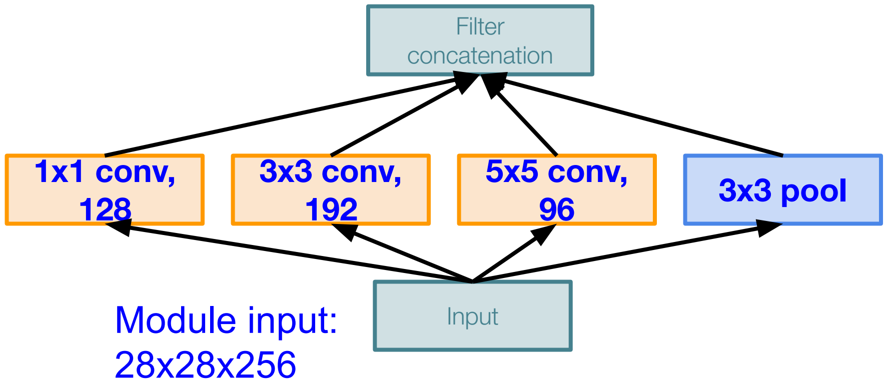
Q: What is the problem with this?
[Hint: Computational complexity]

Case Study: GoogLeNet

[Szegedy et al., 2014]

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

Example:



Naive Inception module

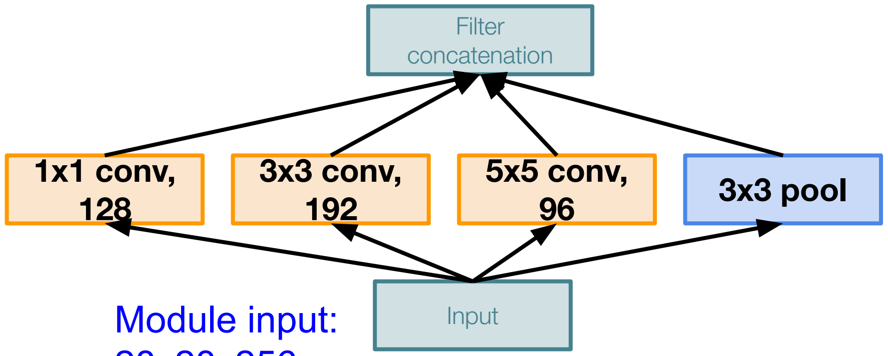
Case Study: GoogLeNet

[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?



Module input: 28x28x256

Naive Inception module

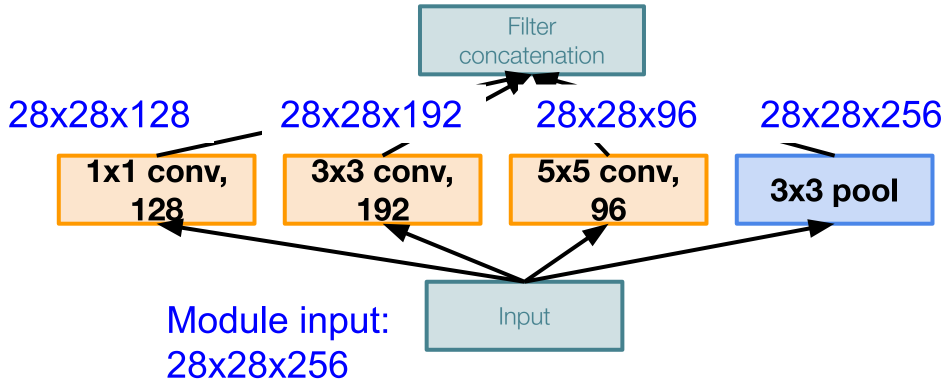
Case Study: GoogLeNet

[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

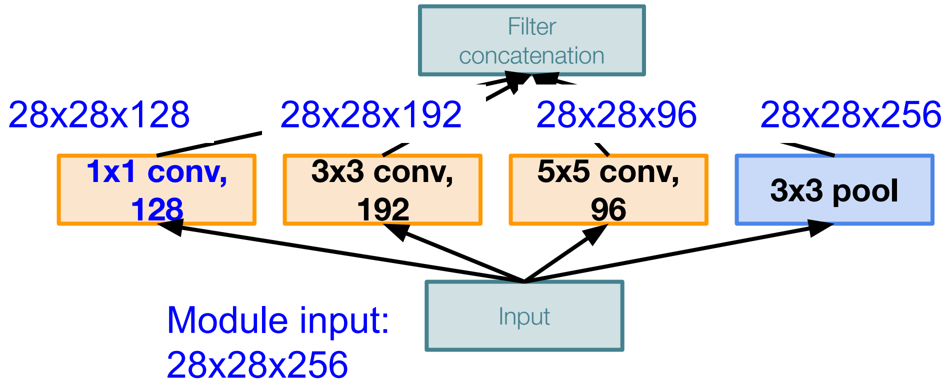
Case Study: GoogLeNet

[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

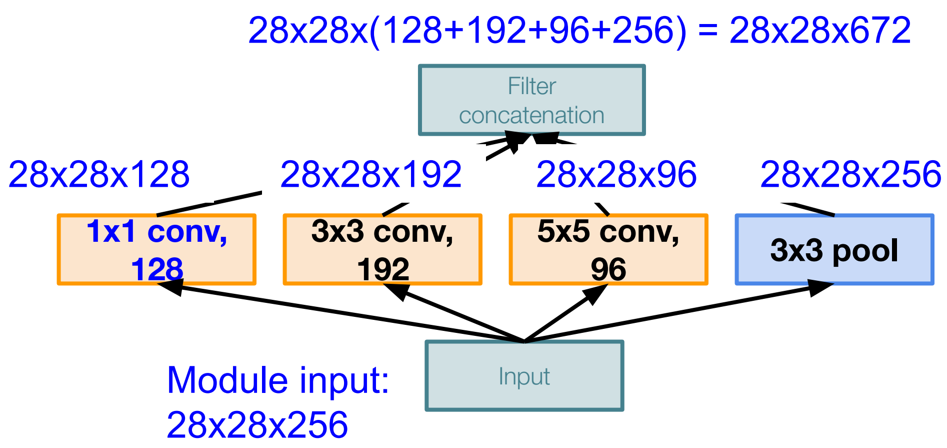
Case Study: GoogLeNet

[Szegedy et al., 2014]

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

Example:

Q2:What is output size after filter concatenation?



Naive Inception module

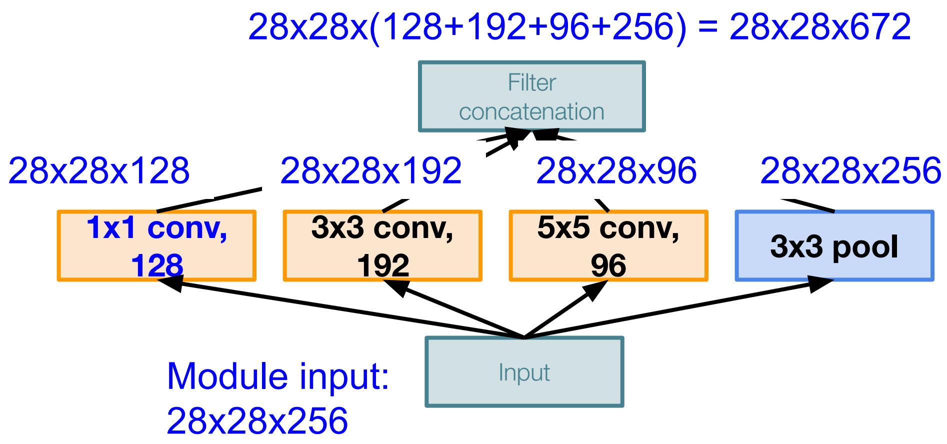
Case Study: GoogLeNet

[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

Conv Ops:

[1x1 conv, 128] 28x28x128x1x1x256

[3x3 conv, 192] 28x28x**192x3x3x256**

[5x5 conv, 96] 28x28x**96x5x5x256**

Total: 854M ops

Case Study: GoogLeNet

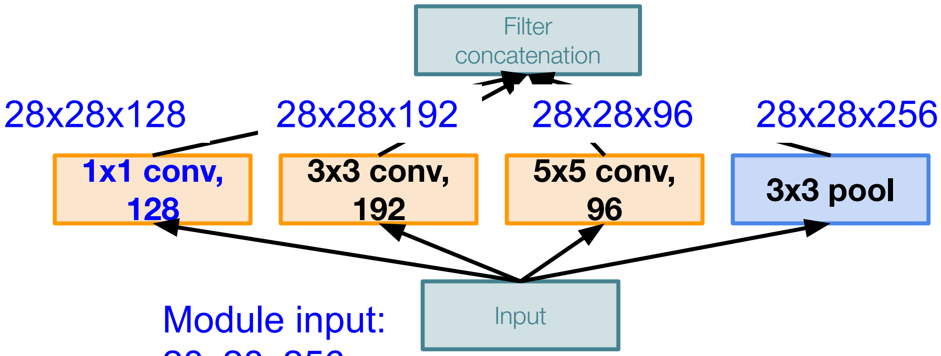
[Szegedy et al., 2014]

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

Example:

Q2:What is output size after filter concatenation?

28x28x(128+192+96+256) = 28x28x672



Module input: 28x28x256

Naive Inception module

Conv Ops:

[1x1 conv, 128] 28x28x128x1x1x256

[3x3 conv, 192] 28x28x**192x3x3x256**

[5x5 conv, 96] 28x28x**96x5x5x256**

Total: 854M ops

Very expensive compute

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

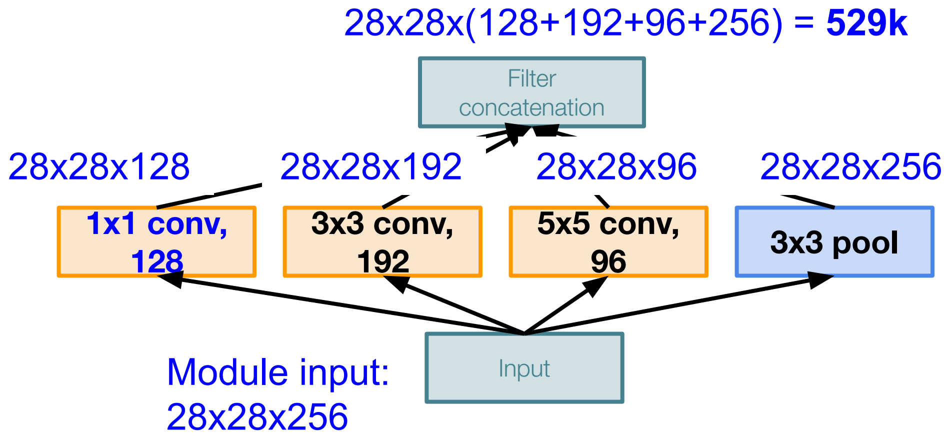
Case Study: GoogLeNet

[Szegedy et al., 2014]

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

Example:

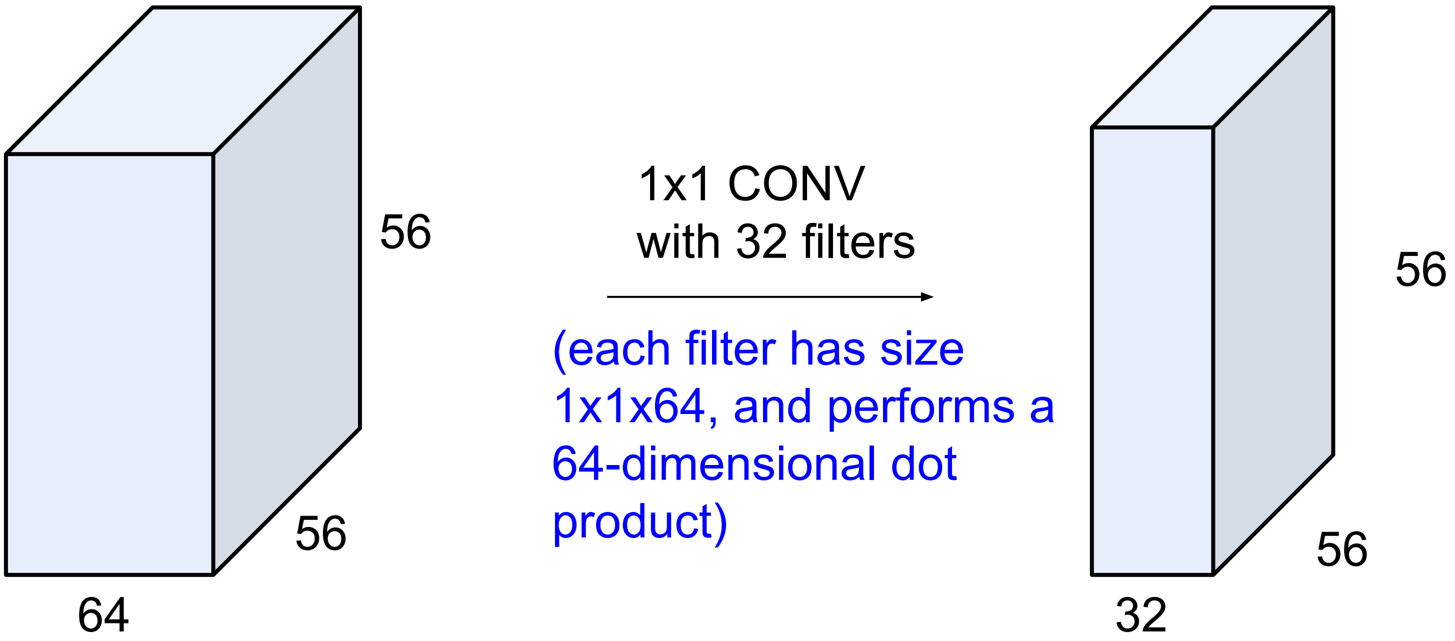
Q2:What is output size after filter concatenation?



Naive Inception module

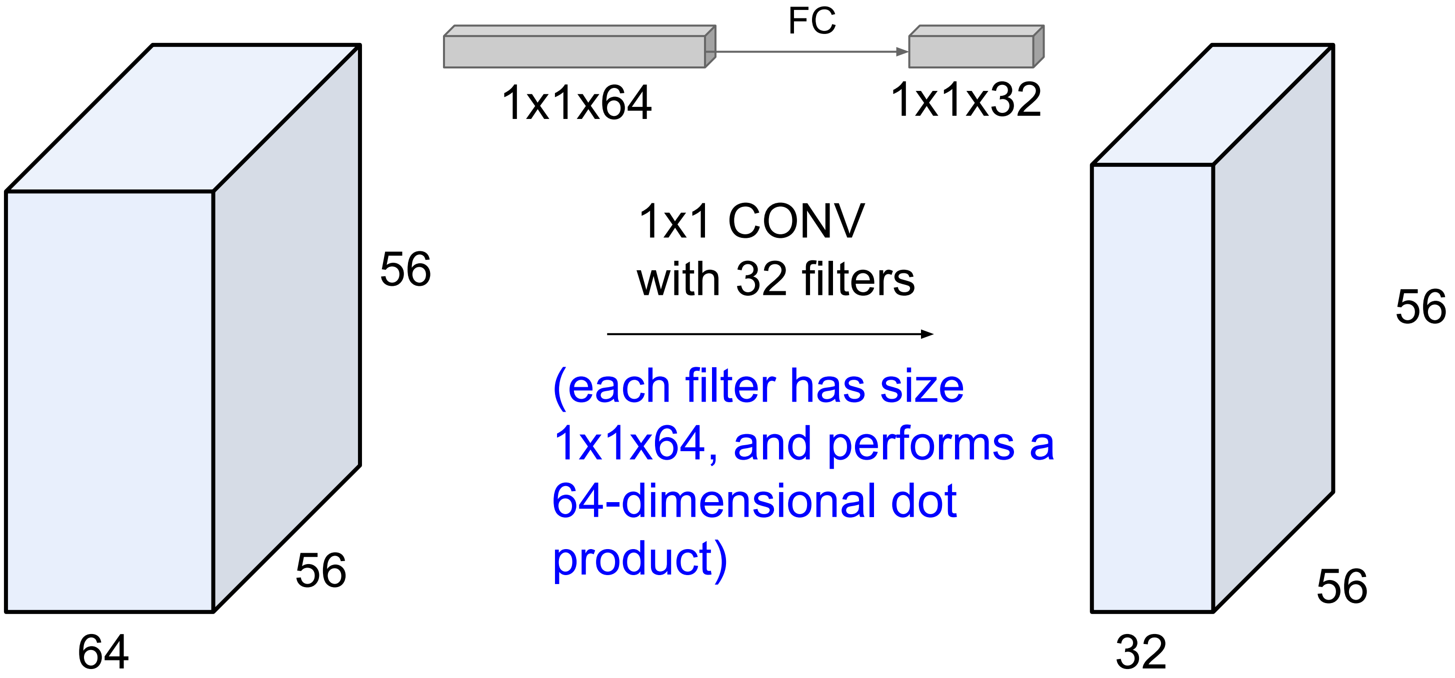
Solution: "bottleneck" layers that use 1x1 convolutions to reduce feature channel size

Review: 1x1 convolutions



Review: 1x1 convolutions

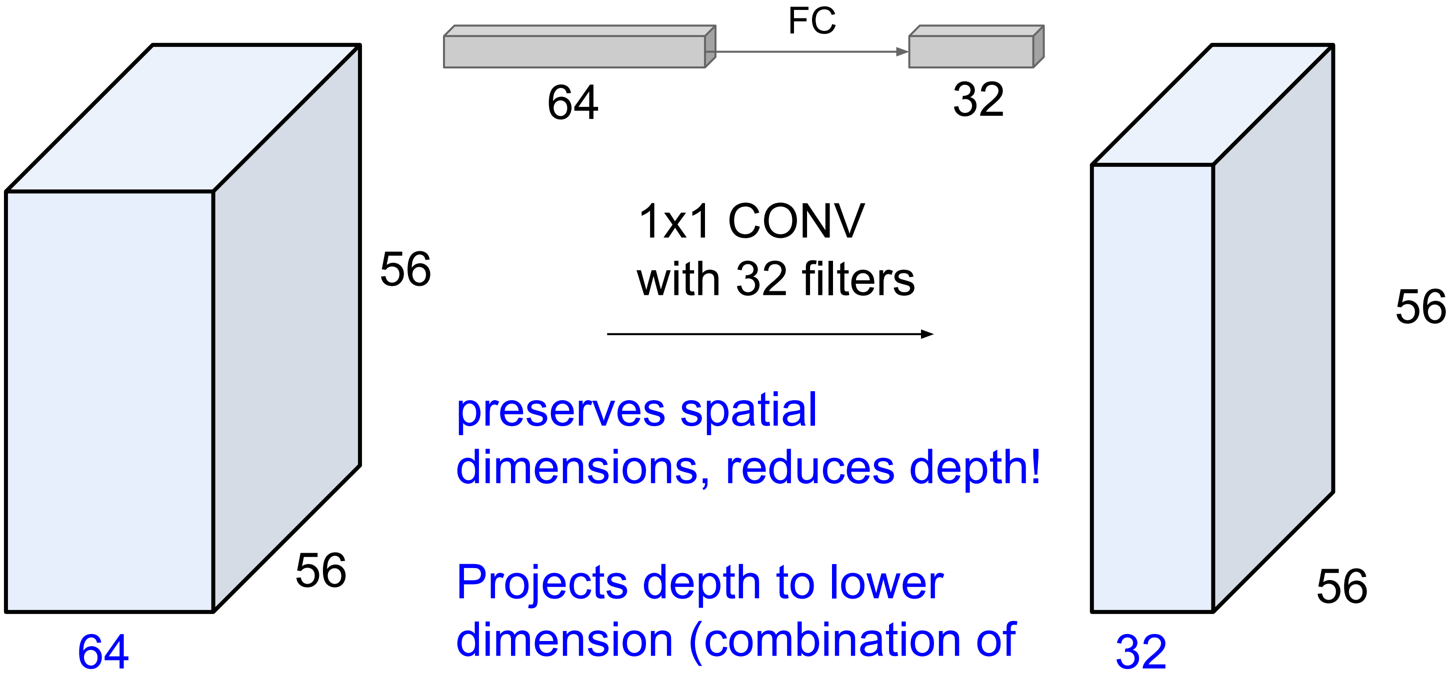
Alternatively, interpret it as applying the same FC layer on each input pixel



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

Review: 1x1 convolutions

Alternatively, interpret it as applying the same FC layer on each input pixel

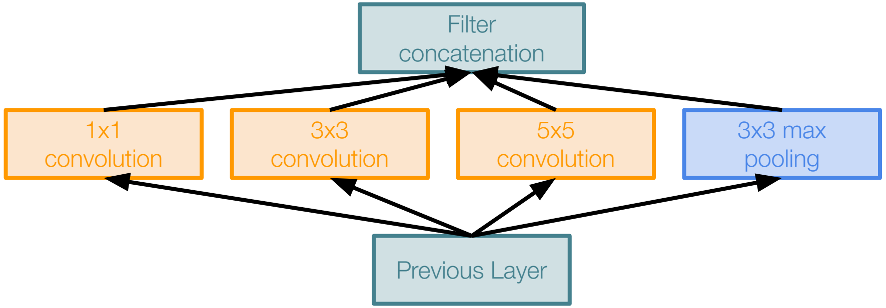


preserves spatial dimensions, reduces depth!

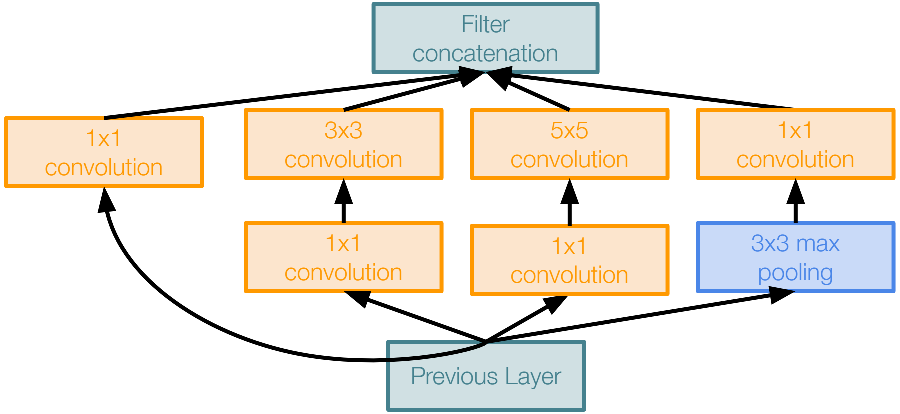
Projects depth to lower dimension (combination of feature maps)

Case Study: GoogLeNet

[Szegedy et al., 2014]



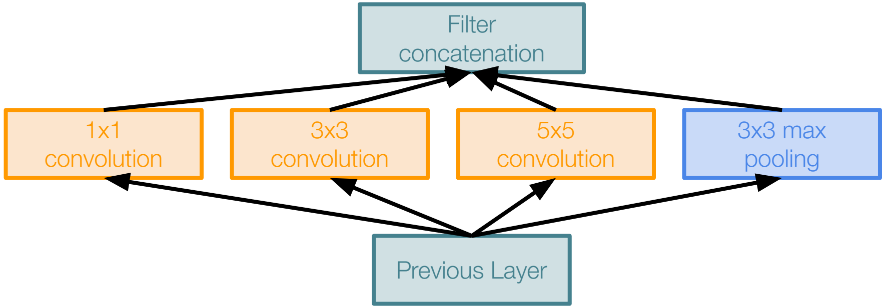
Naive Inception module



Inception module with dimension reduction

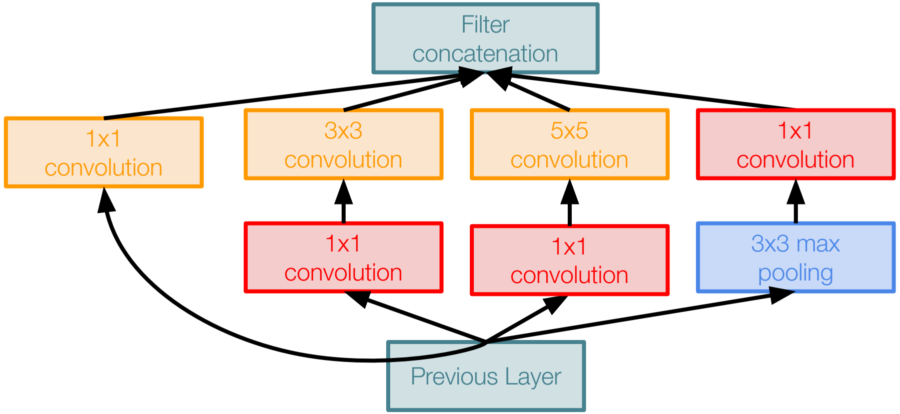
Case Study: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

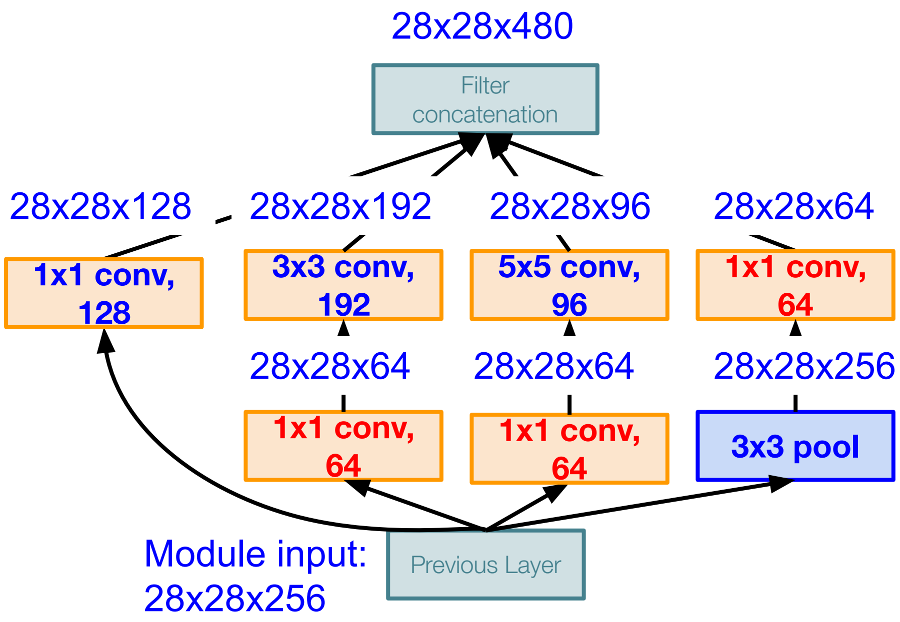
1x1 conv "bottleneck" layers



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 “1x1 conv, 64 filter” bottlenecks:

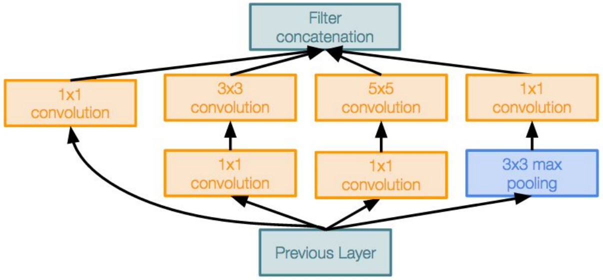
Conv Ops:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops

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

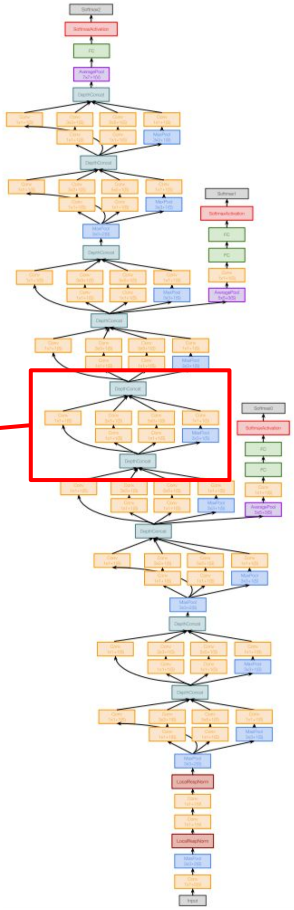
Case Study: GoogLeNet

[Szegedy et al., 2014]

Stack Inception modules
with dimension reduction
on top of each other



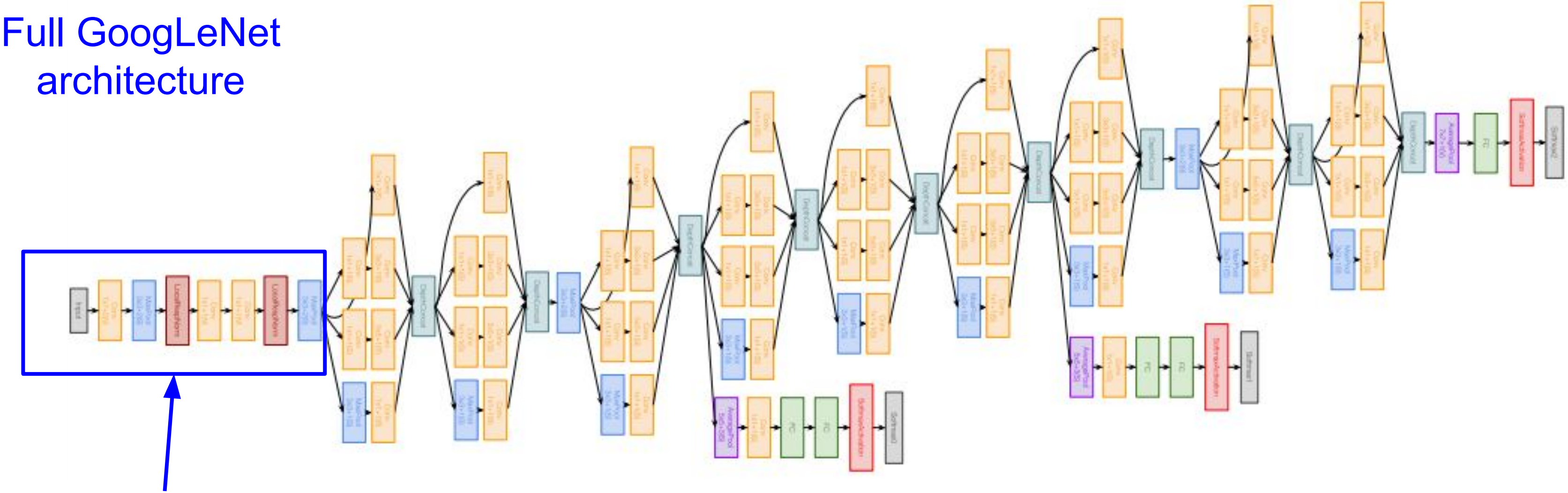
Inception module



Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture

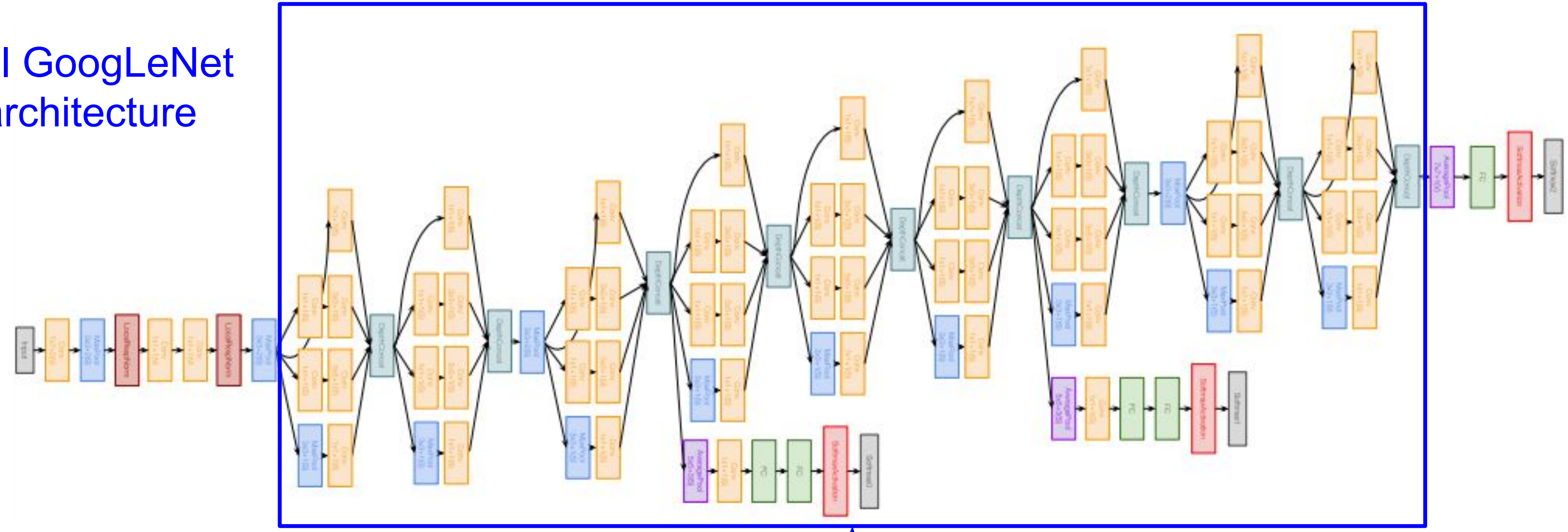


Stem Network:
Conv-Pool-
2x Conv-Pool

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture

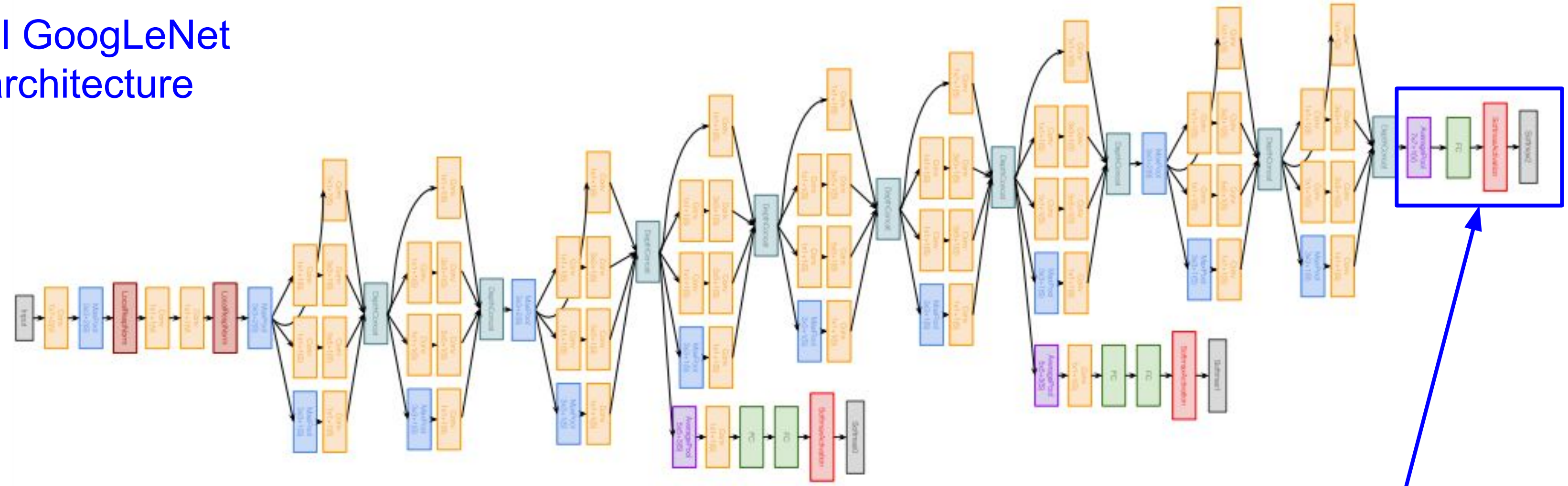


Stacked Inception Modules

Case Study: GoogLeNet

[Szegedy et al., 2014]

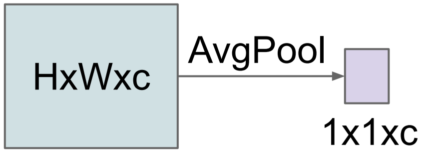
Full GoogLeNet architecture



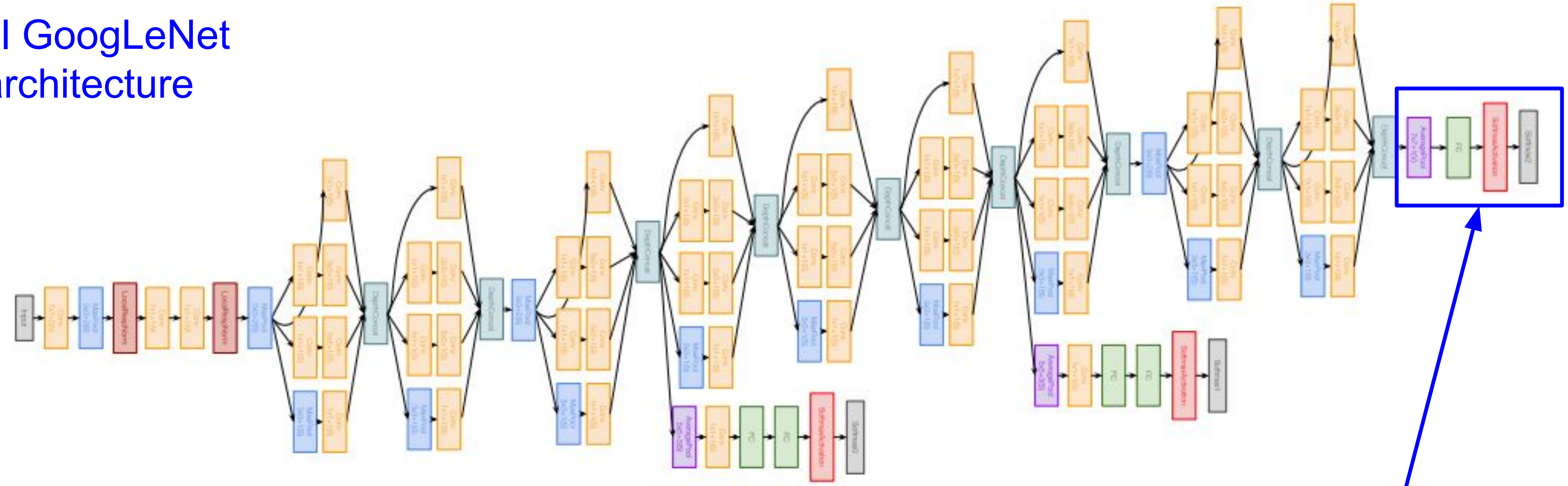
Classifier output

Case Study: GoogLeNet

[Szegedy et al., 2014]



Full GoogLeNet architecture



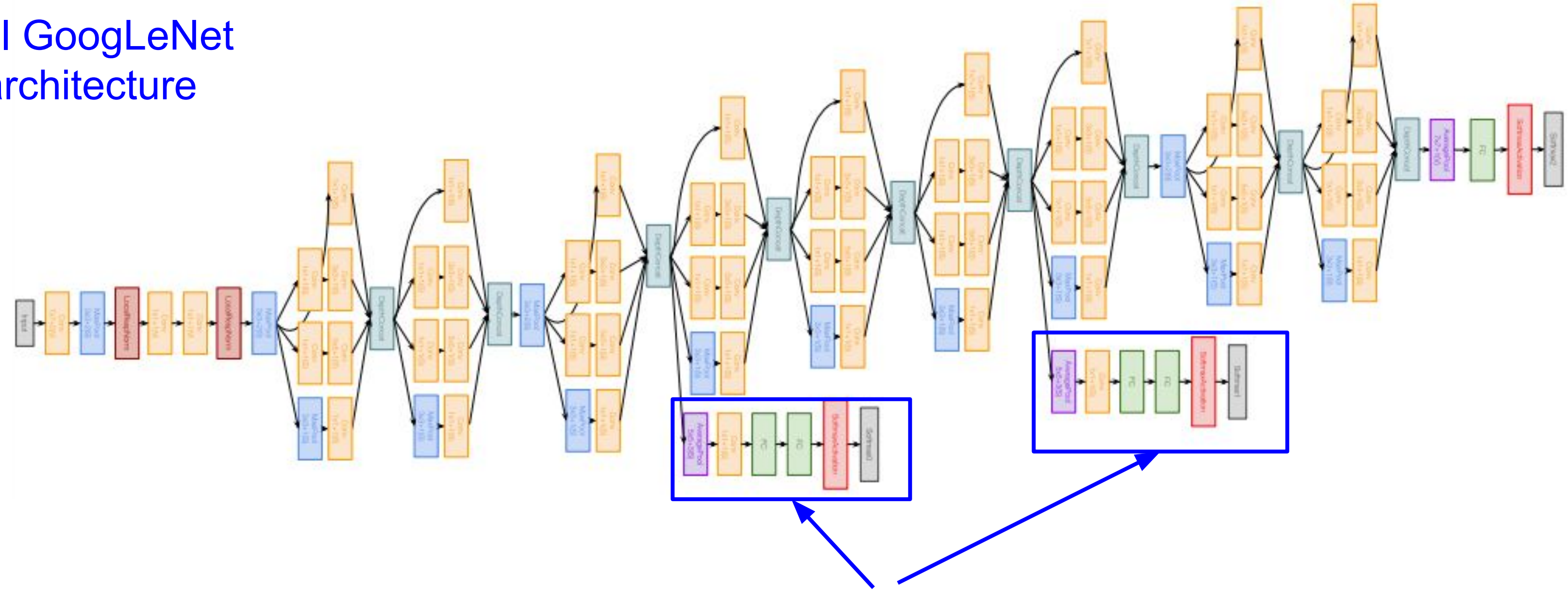
Note: after the last convolutional layer, a global average pooling layer is used that spatially averages across each feature map, before final FC layer. No longer multiple expensive FC layers!

Classifier output

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture

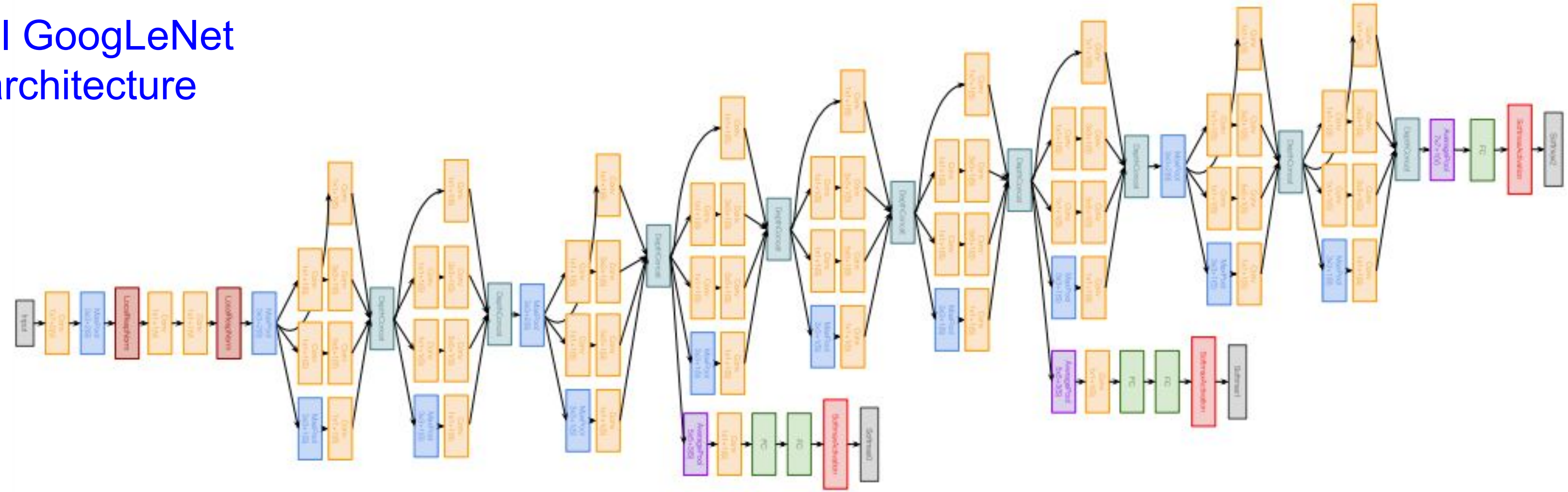


Auxiliary classification outputs to inject additional gradient at lower layers (AvgPool-1x1Conv-FC-FC-Softmax)

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture



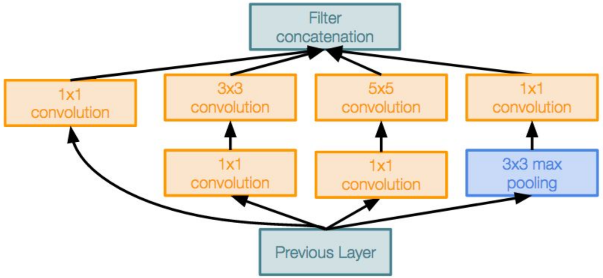
22 total layers with weights
(parallel layers count as 1 layer => 2 layers per Inception module. Don’t count auxiliary output layers)

Case Study: GoogLeNet

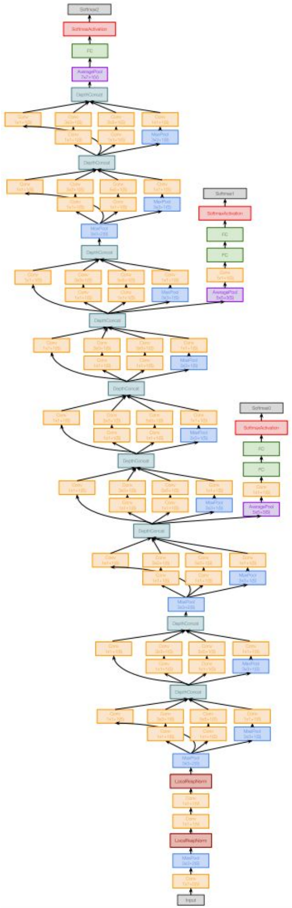
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

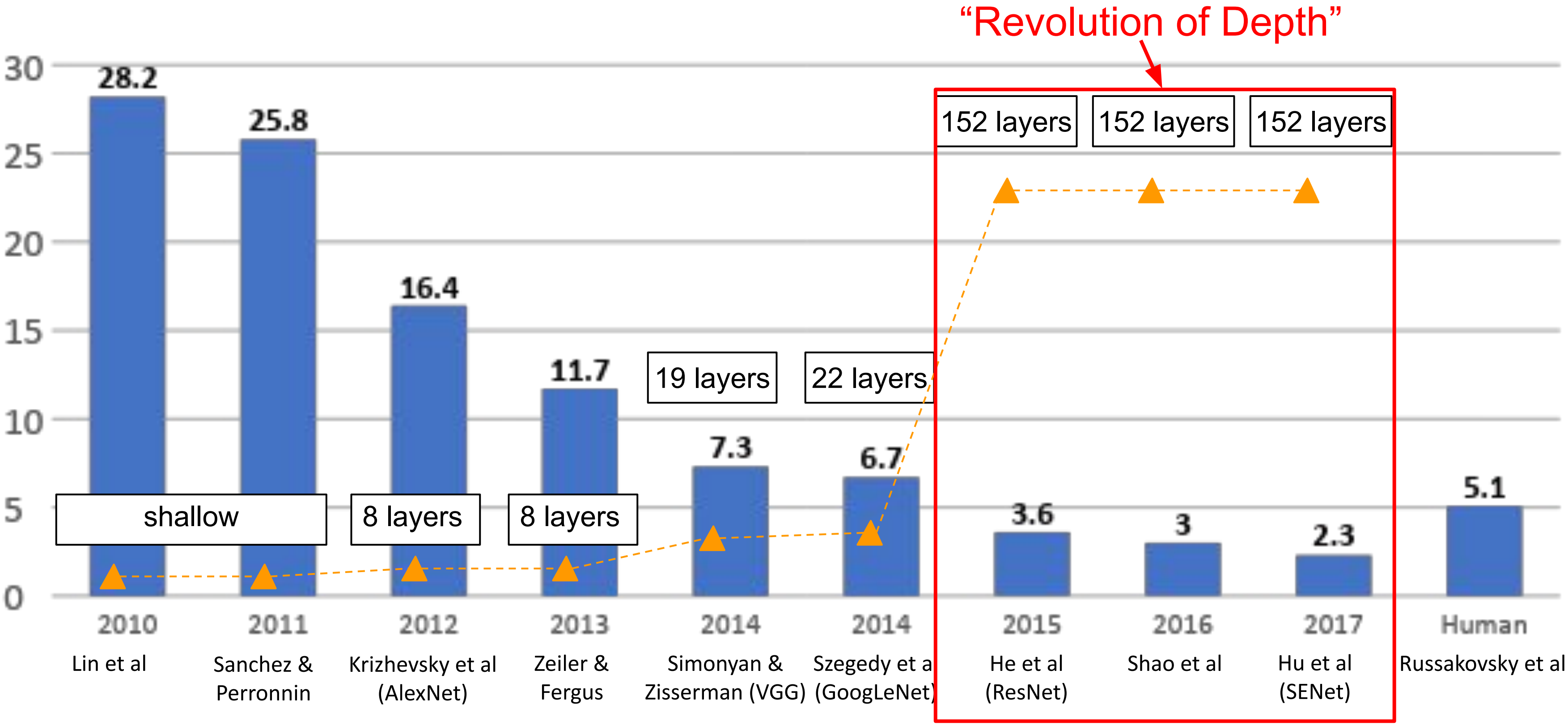
- 22 layers
- Efficient "Inception" module
- Avoids expensive FC layers
- 12x less params than AlexNet
- 27x less params than VGG-16
- ILSVRC'14 classification winner (6.7% top 5 error)



Inception module



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

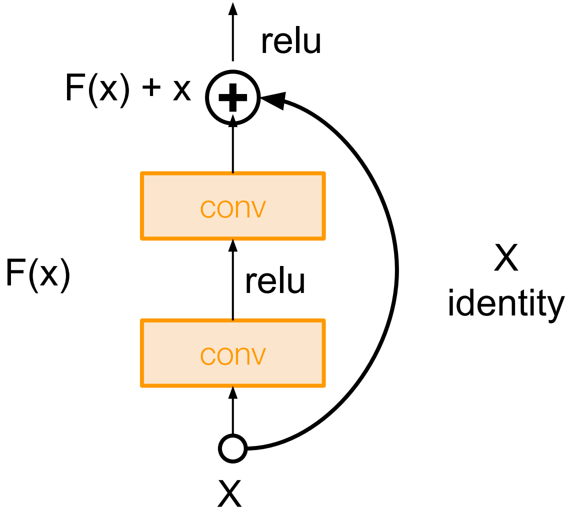


Case Study: ResNet

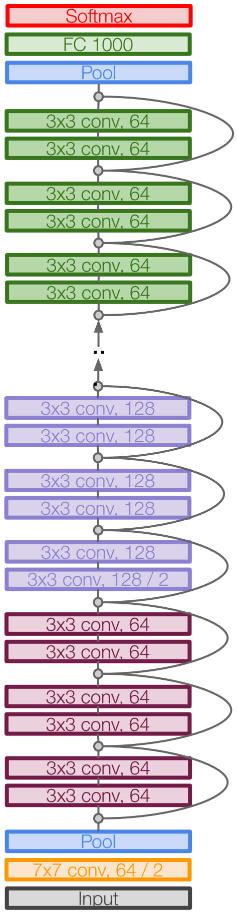
[He et al., 2015]

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!



Residual block



Case Study: ResNet

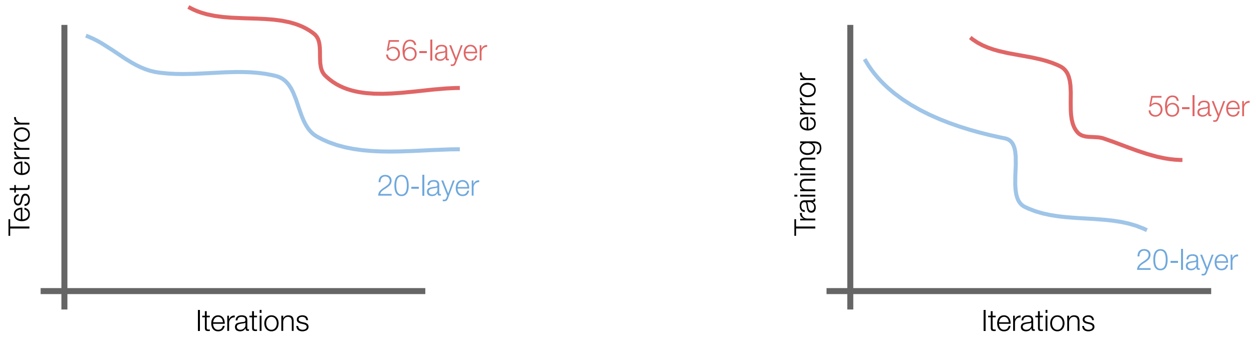
[He et al., 2015]

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

Case Study: ResNet

[He et al., 2015]

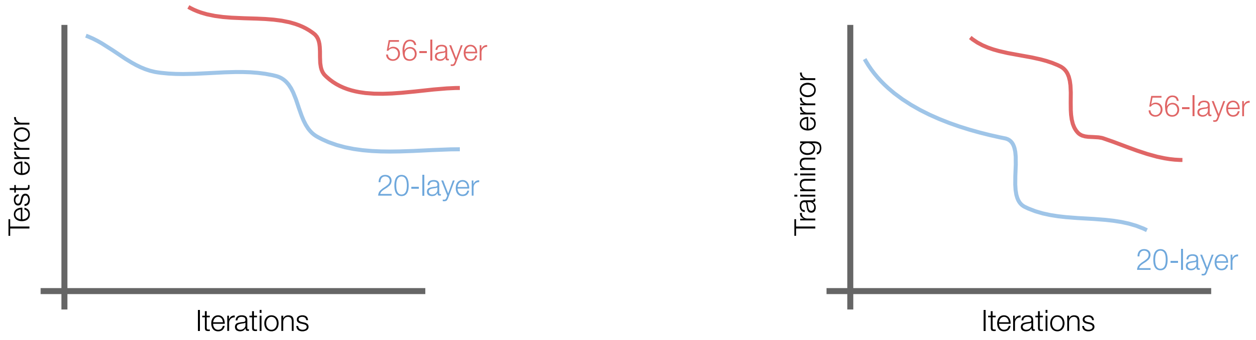
What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



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

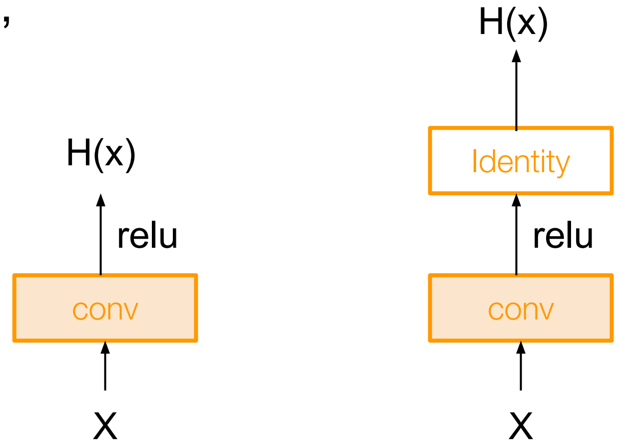
[He et al., 2015]

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?

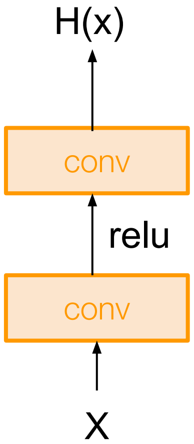
A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.



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

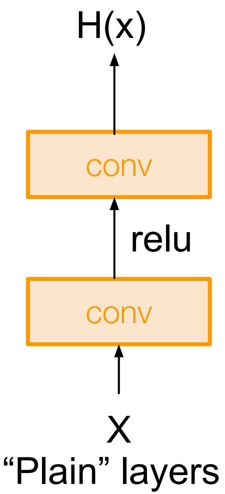


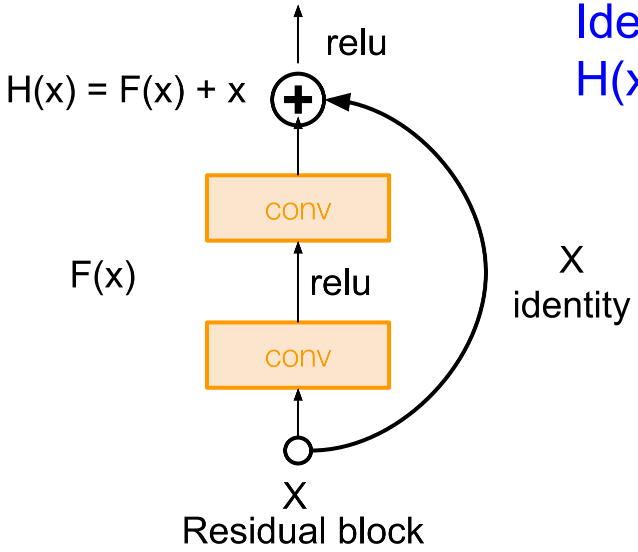
"Plain" layers

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



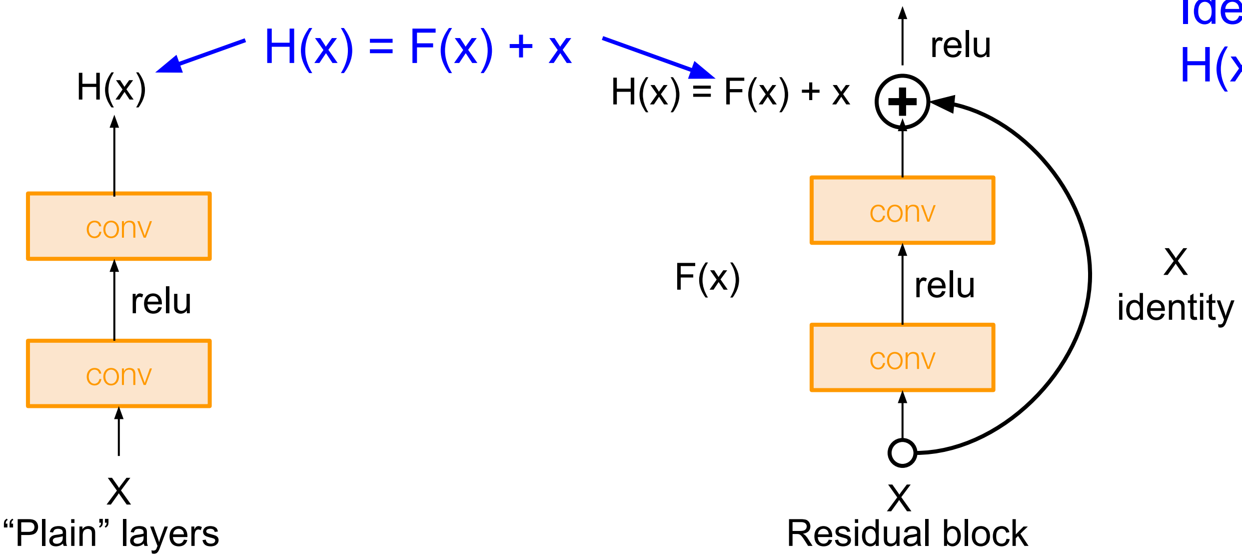


Identity mapping:
$H(x) = x$ if $F(x) = 0$

Case Study: ResNet

[He et al., 2015]

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



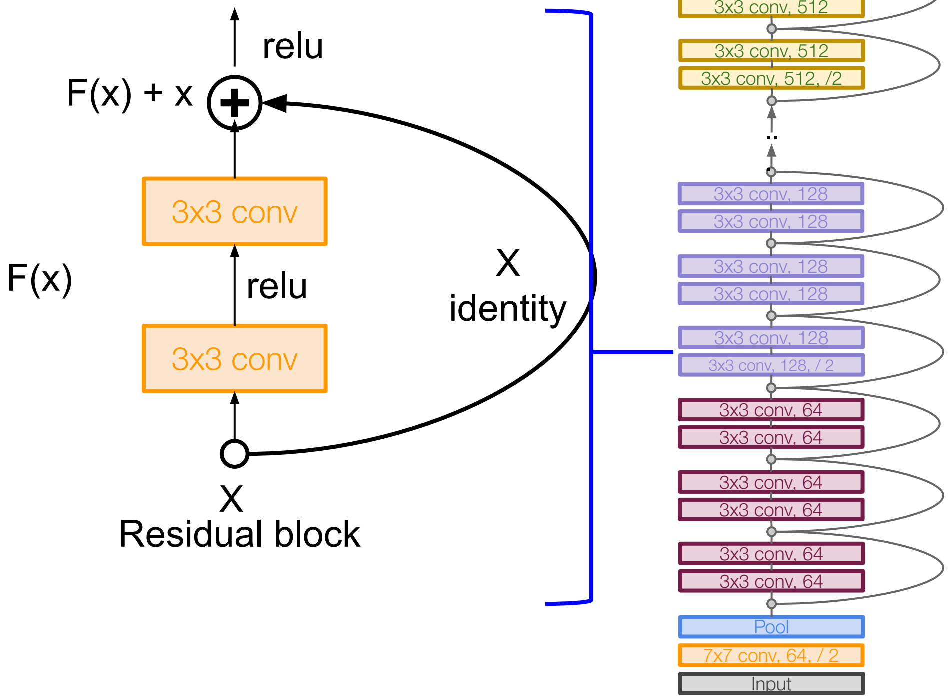
Identity mapping:
H(x) = x if F(x) = 0

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

Case Study: ResNet

[He et al., 2015]

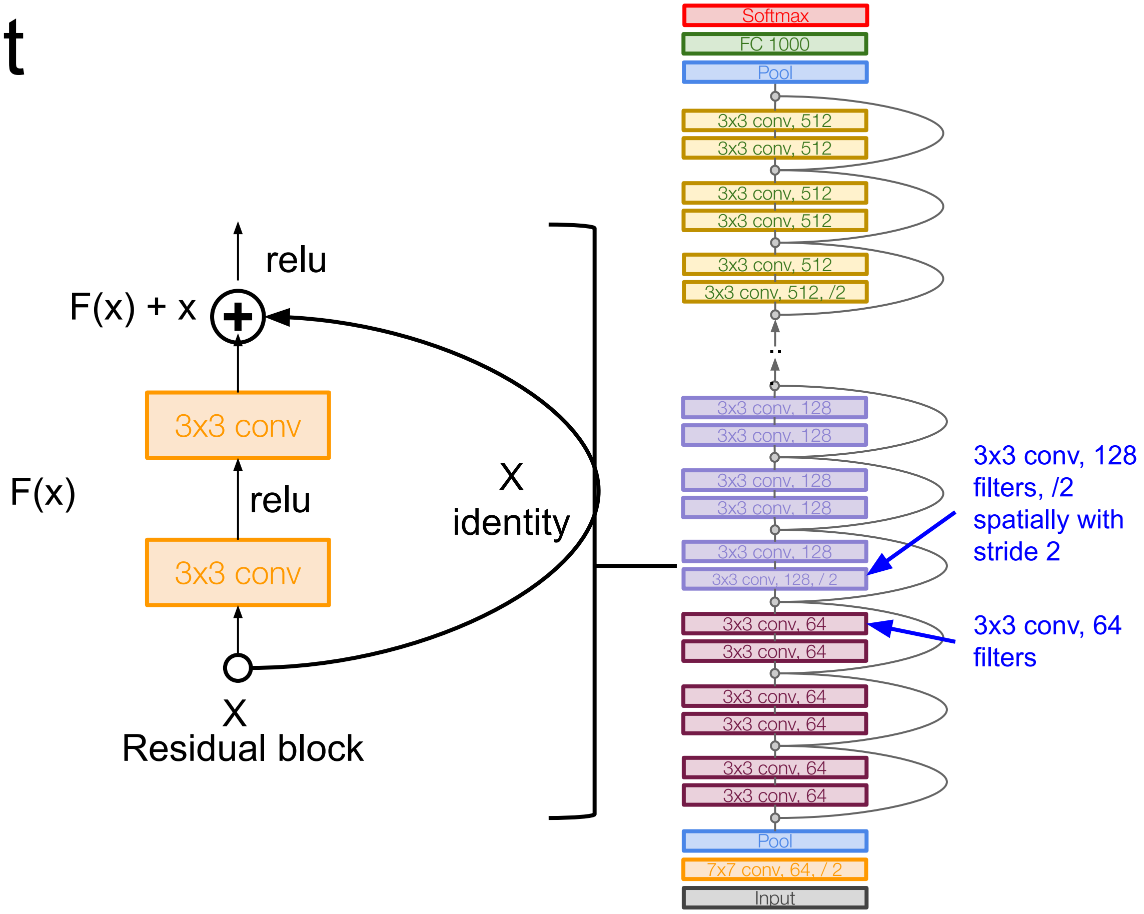
Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers



Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension) Reduce the activation volume by half.

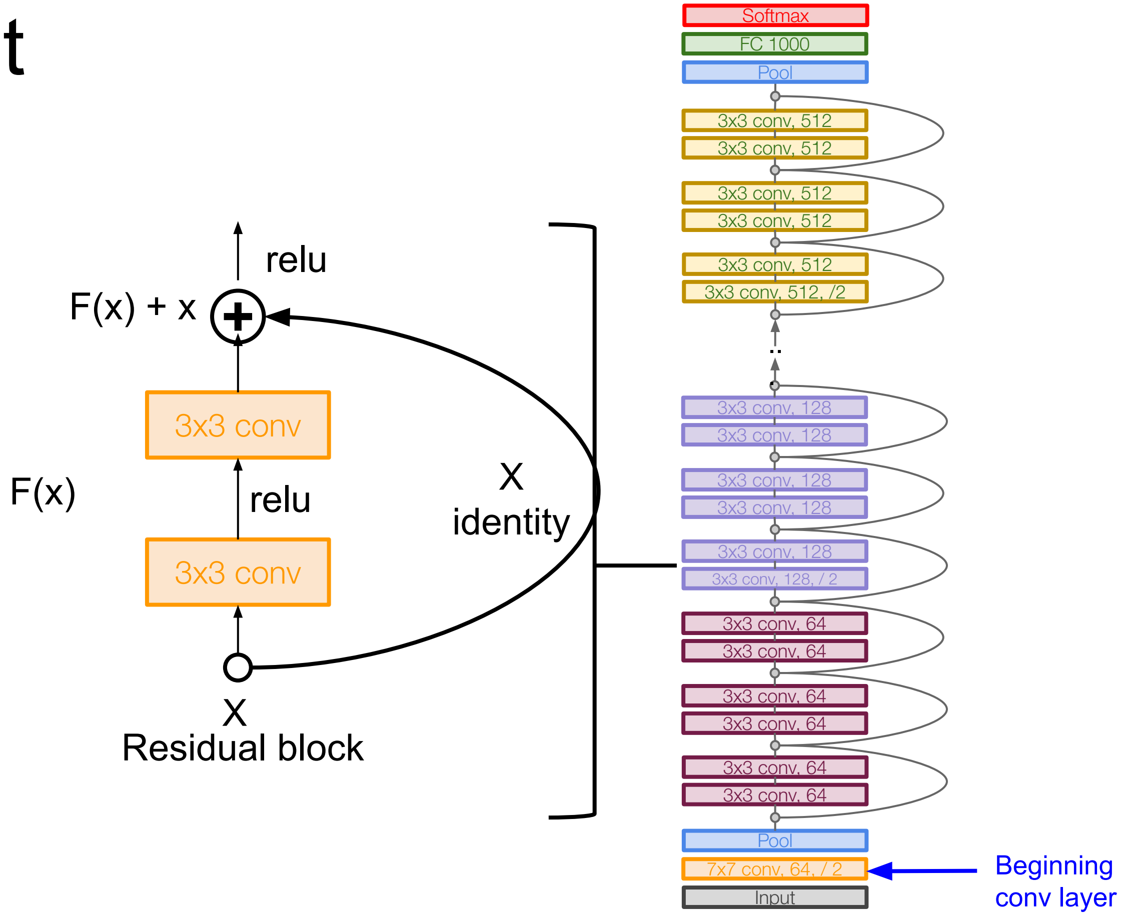


Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

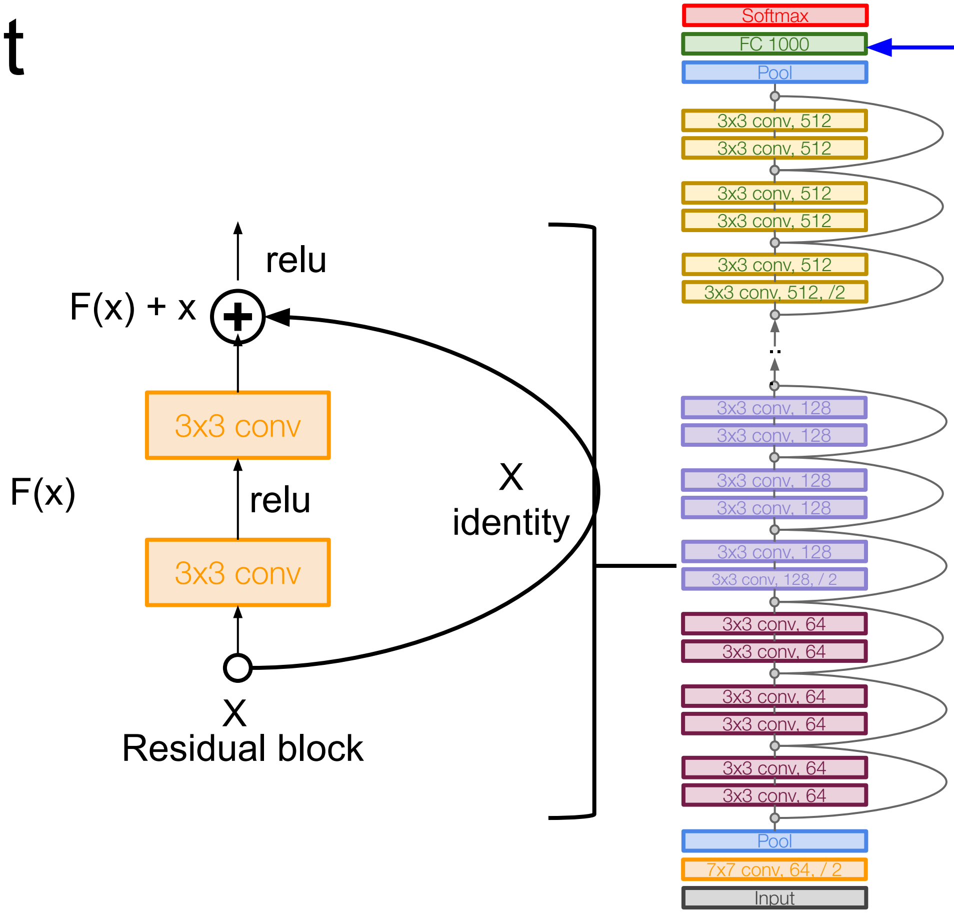
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning (stem)



Case Study: ResNet

[He et al., 2015]

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



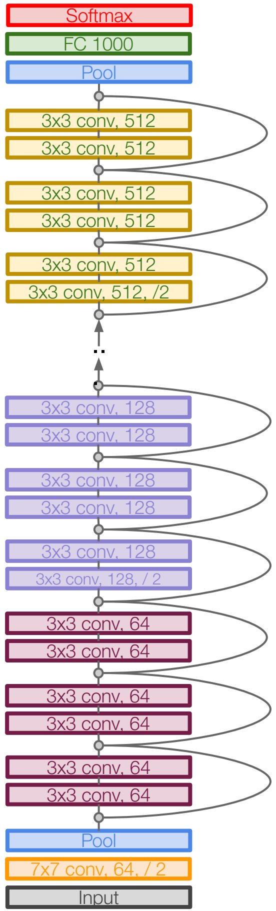
No FC layers besides FC 1000 to output classes

Global average pooling layer after last conv layer

Case Study: ResNet

[He et al., 2015]

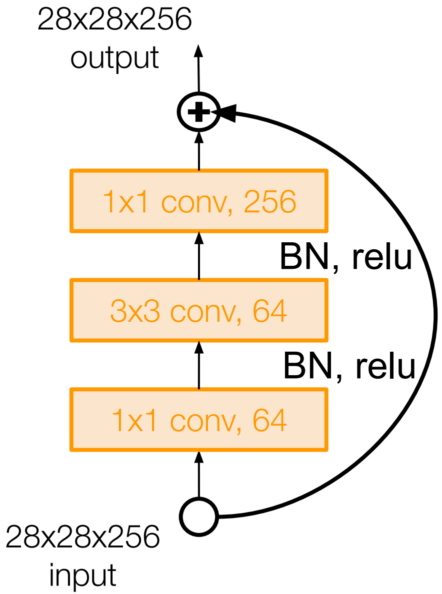
Total depths of 18, 34, 50, 101, or 152 layers for ImageNet



Case Study: ResNet

[He et al., 2015]

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



Case Study: ResNet

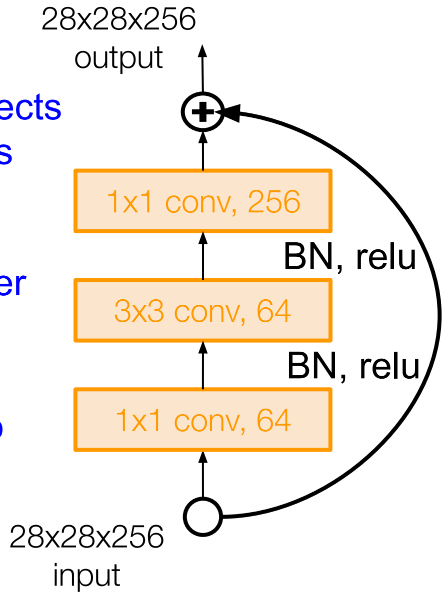
[He et al., 2015]

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

1x1 conv, 256 filters projects back to 256 feature maps (28x28x256)

3x3 conv operates over only 64 feature maps

1x1 conv, 64 filters to project to 28x28x64



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

Case Study: ResNet

[He et al., 2015]

Experimental Results

- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lower training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: *"Ultra-deep"* (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

Case Study: ResNet

[He et al., 2015]

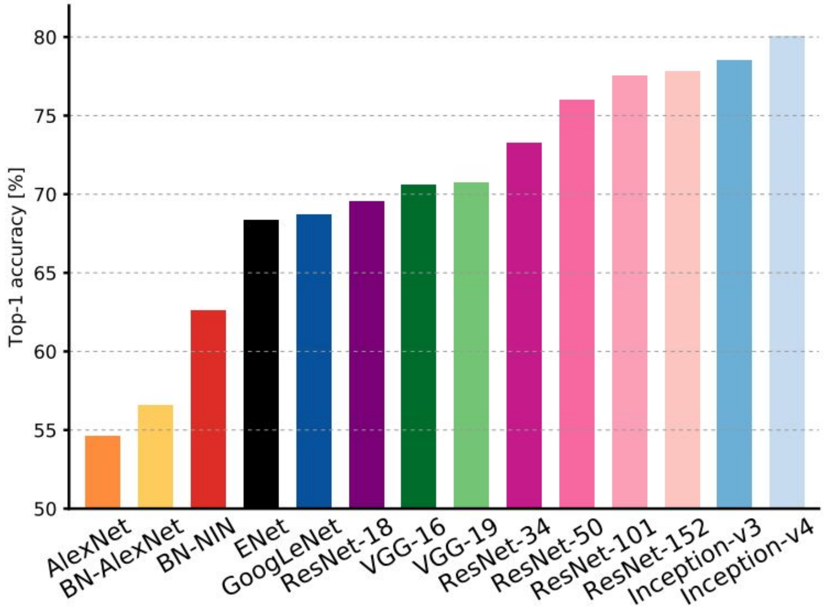
Experimental Results
- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lower training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

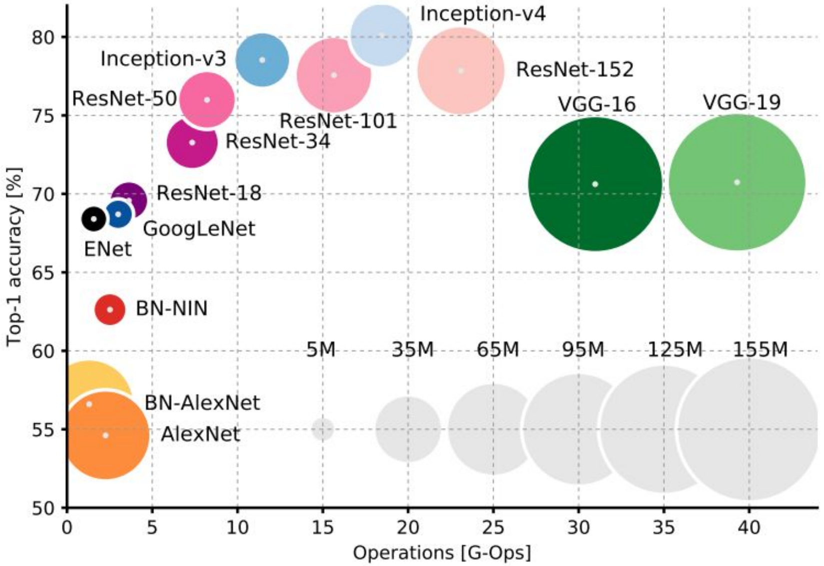
MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: *"Ultra-deep"* (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

ILSVRC 2015 classification winner (3.6% top 5 error) -- better than "human performance"! (Russakovsky 2014)

Comparing complexity...

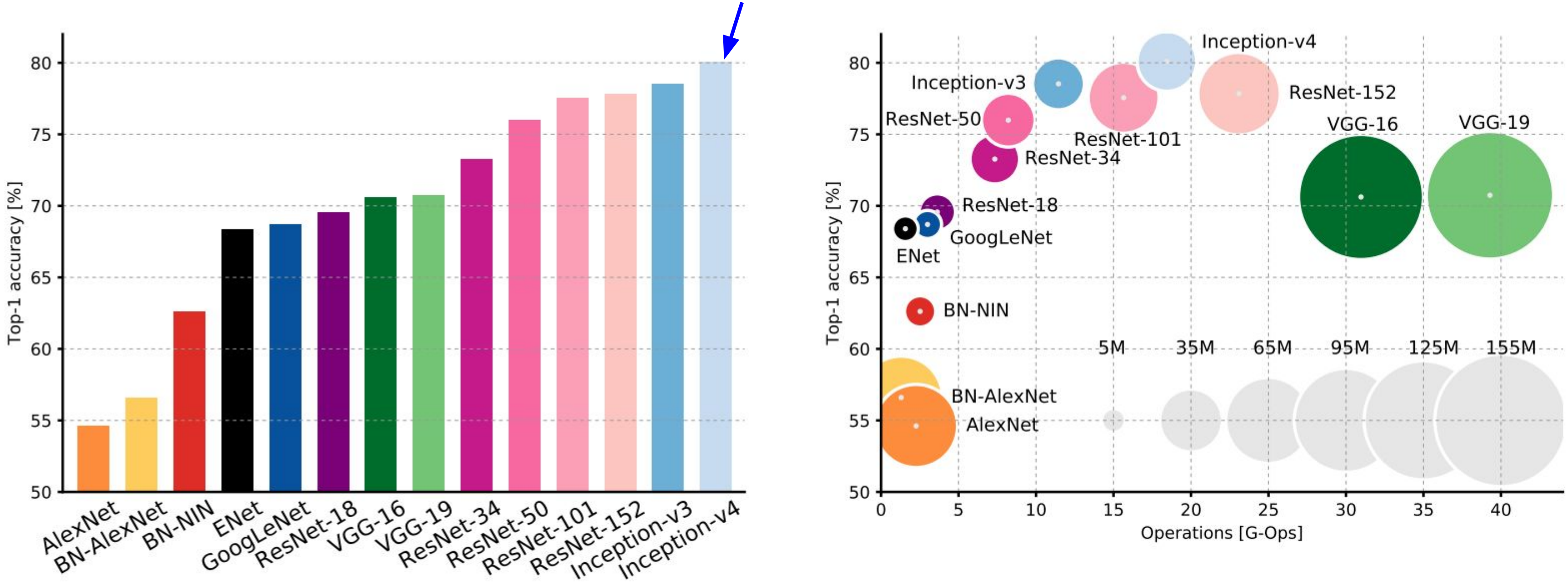




An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Comparing complexity...

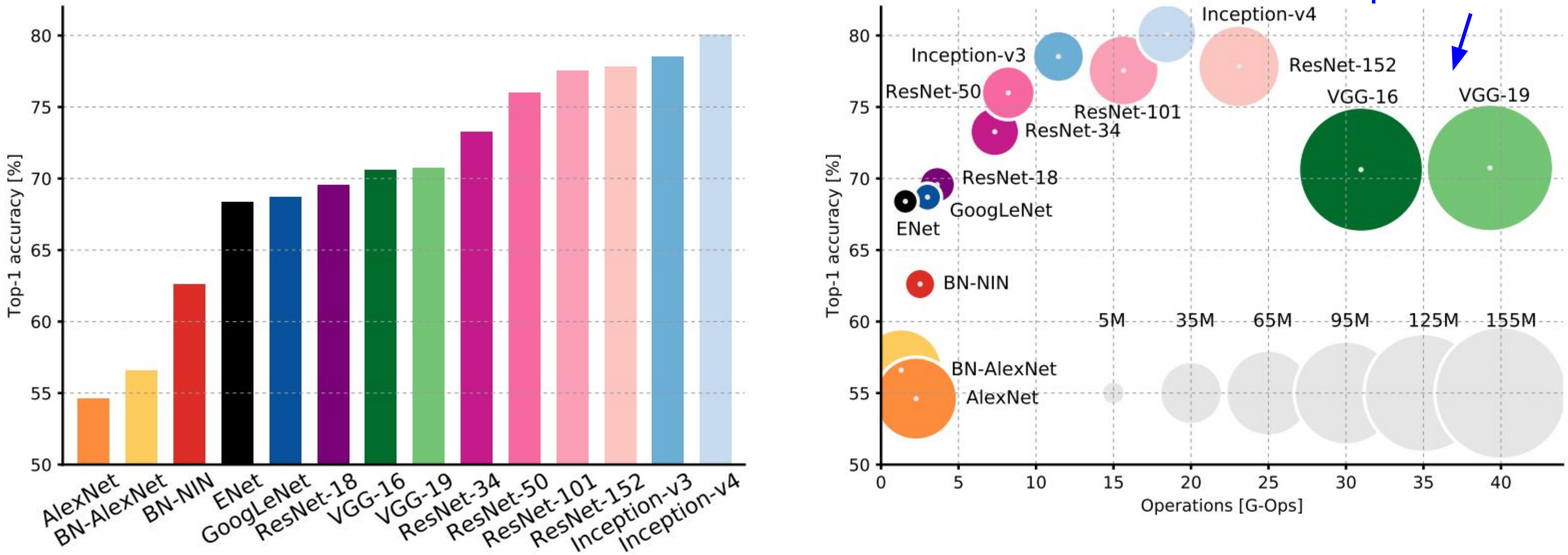
Inception-v4: Resnet + Inception!



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

Comparing complexity...

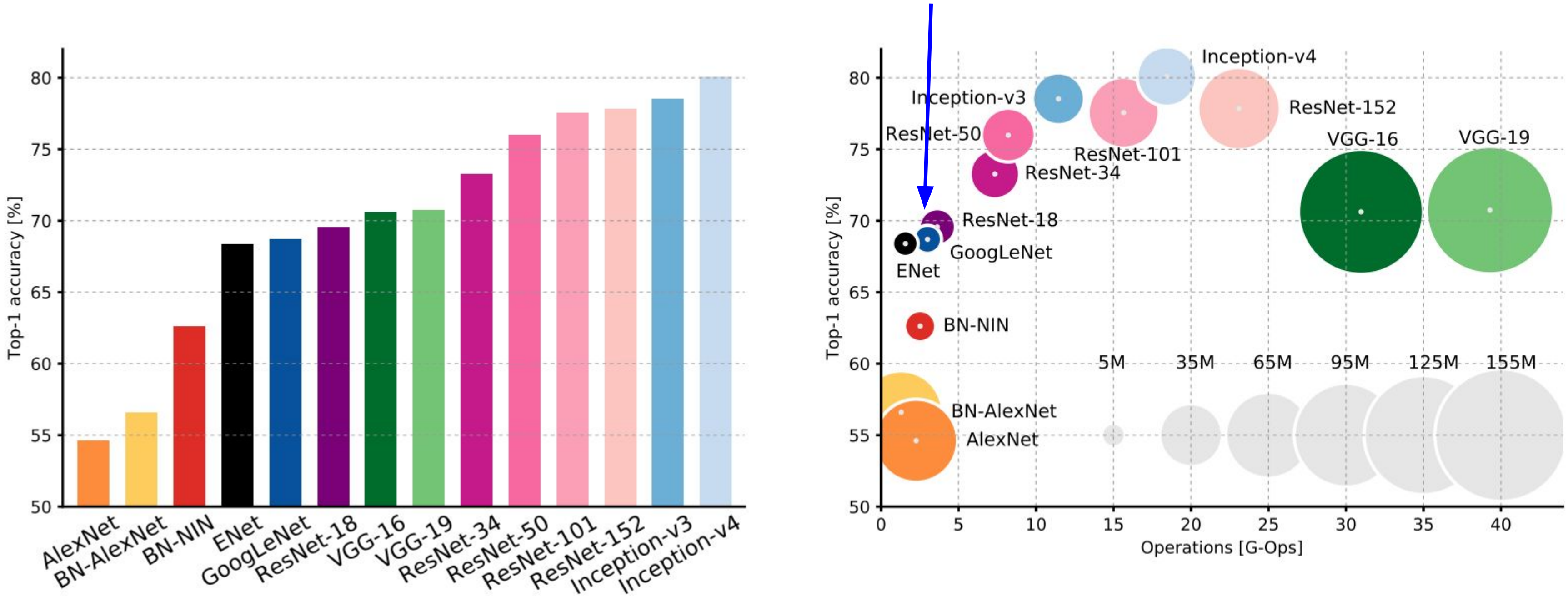
VGG: most parameters, most operations



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

Comparing complexity...

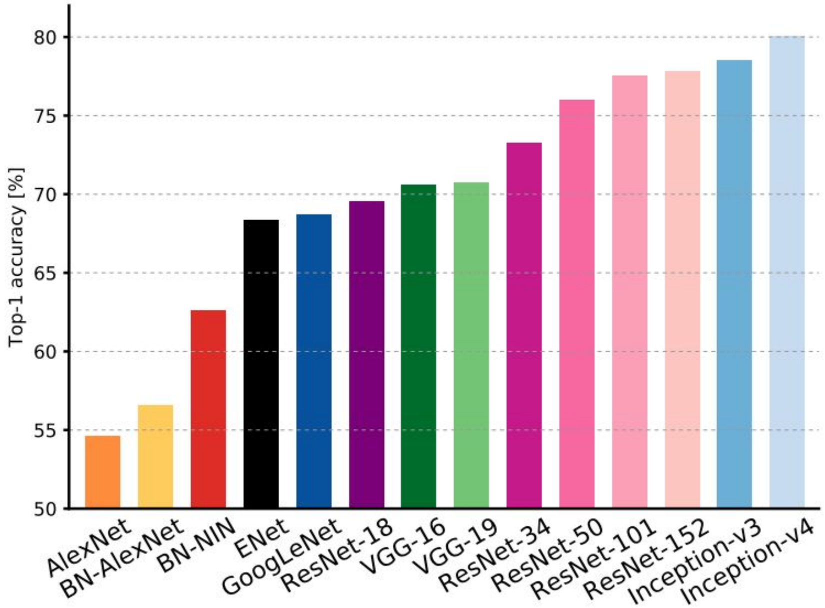
GoogLeNet:
most efficient

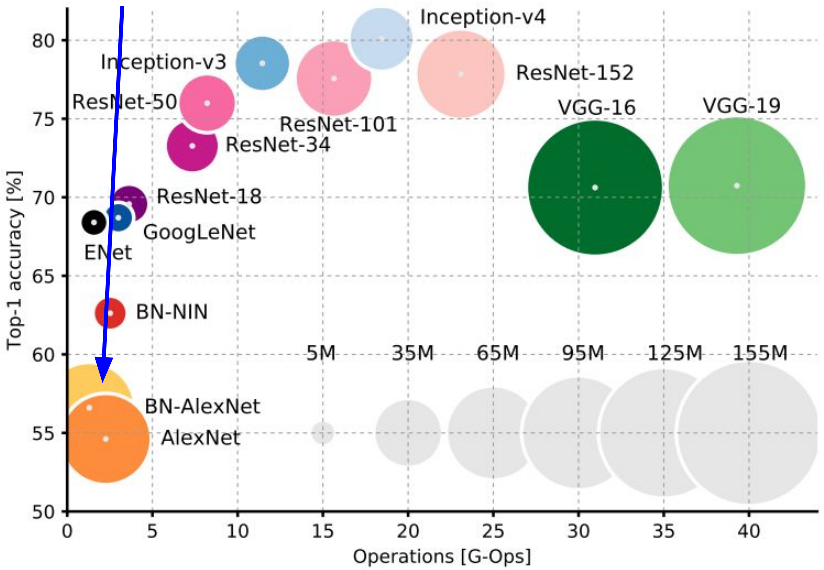


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Comparing complexity...

AlexNet:
Smaller compute, still memory heavy, lower accuracy

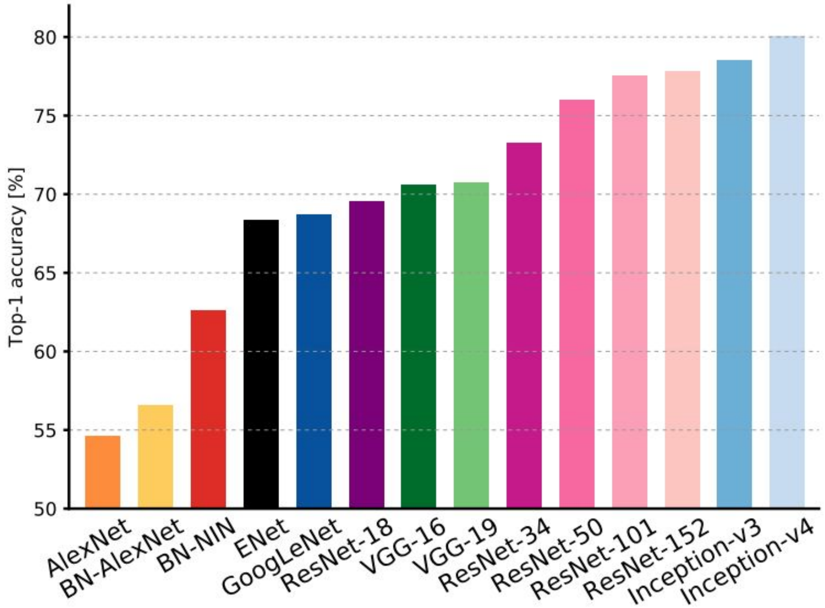


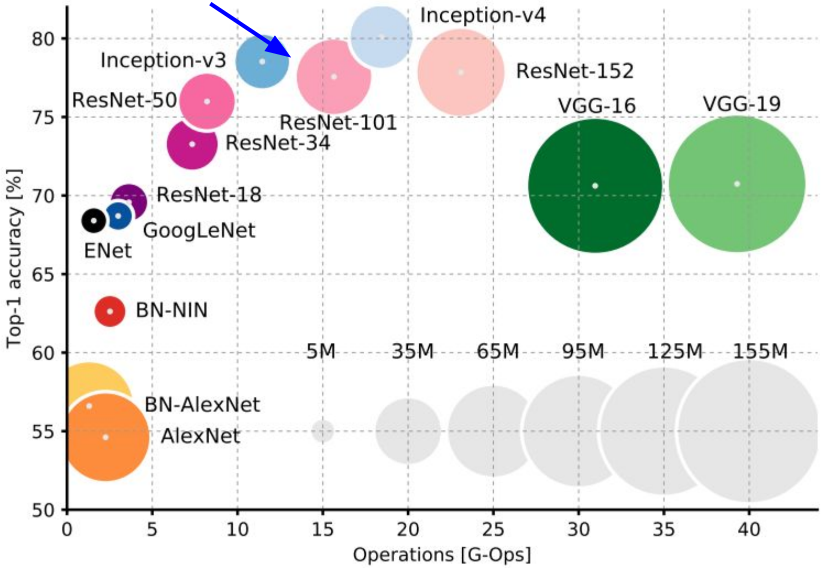


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Comparing complexity...

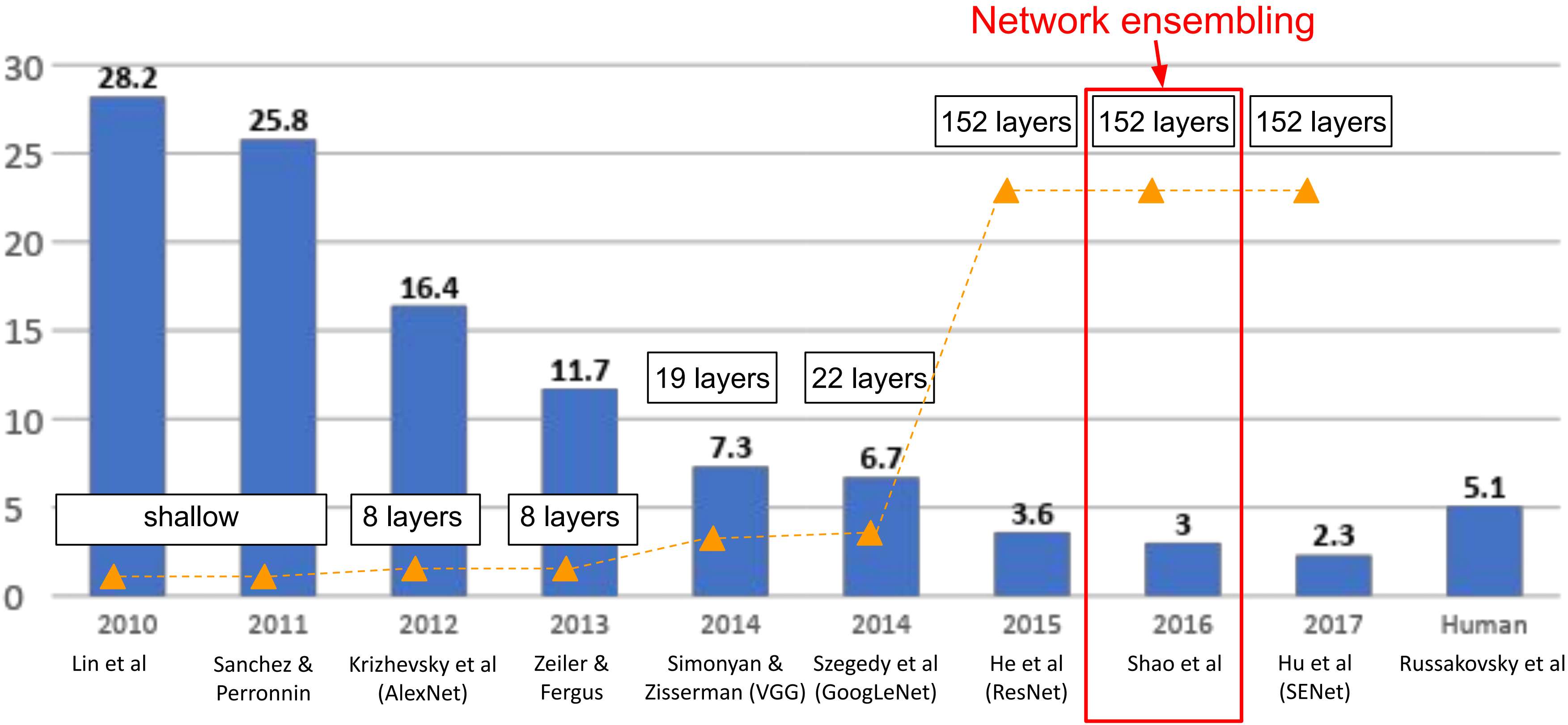
ResNet:
Moderate efficiency depending on model, highest accuracy





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

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



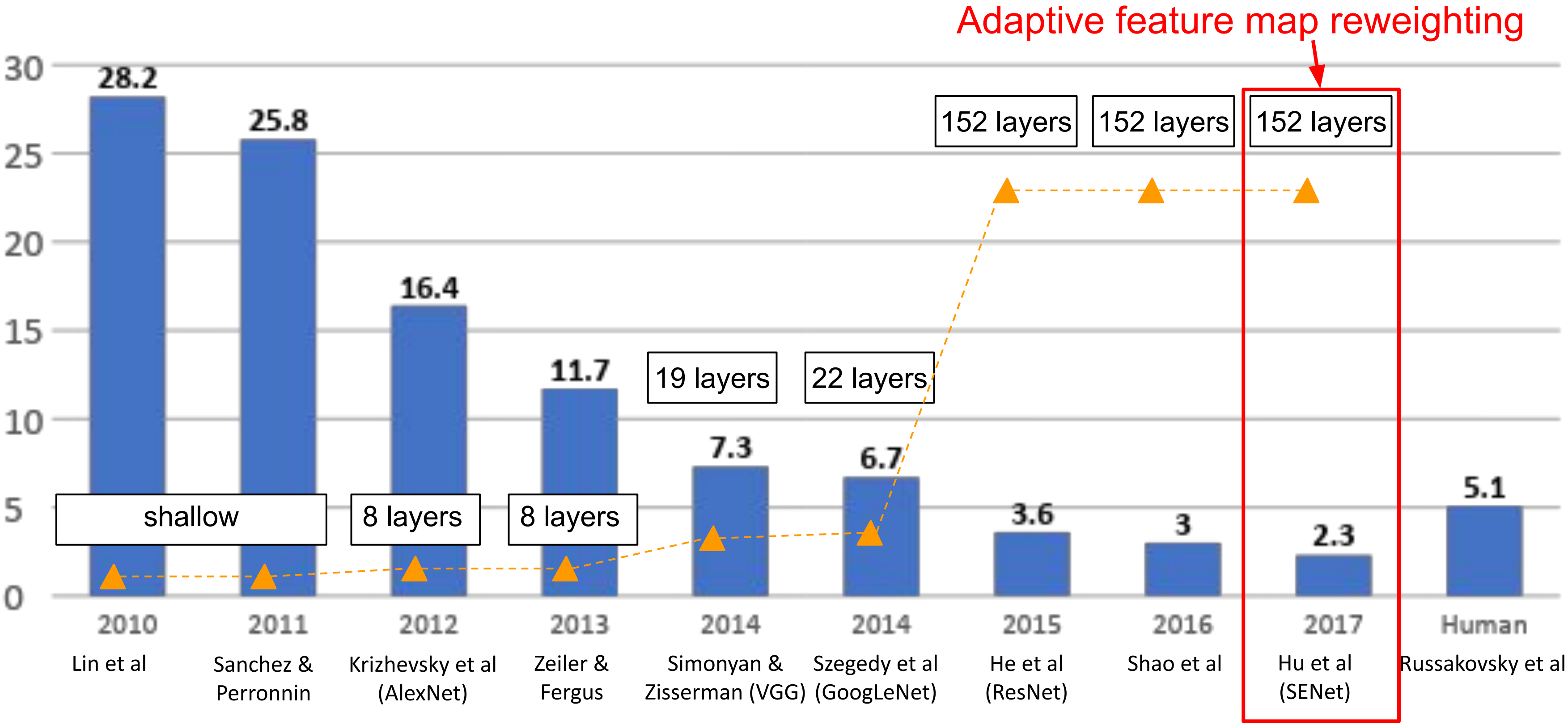
“Good Practices for Deep Feature Fusion”

[Shao et al. 2016]

- Multi-scale ensembling of Inception, Inception-Resnet, Resnet, Wide Resnet models
- ILSVRC’16 classification winner

	Inception-v3	Inception-v4	Inception-Resnet-v2	Resnet-200	Wrn-68-3	Fusion（Val.）	Fusion（Test）
Err. (%)	4.20	4.01	3.52	4.26	4.65	2.92 (-0.6)	2.99

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

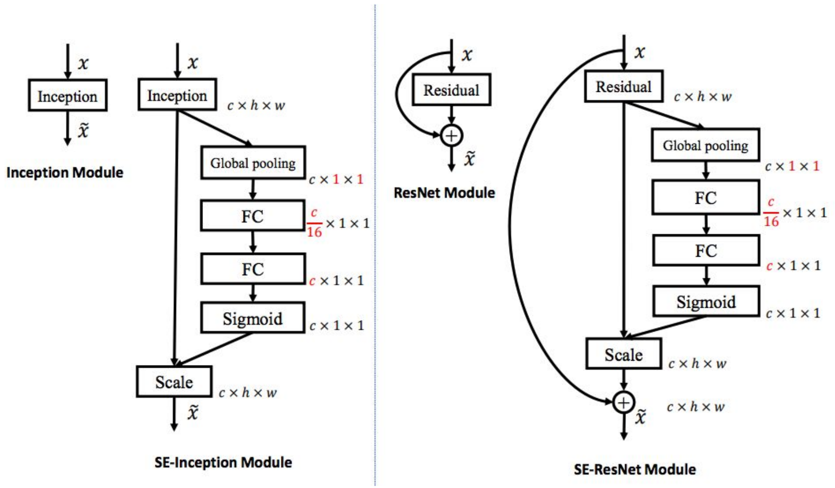


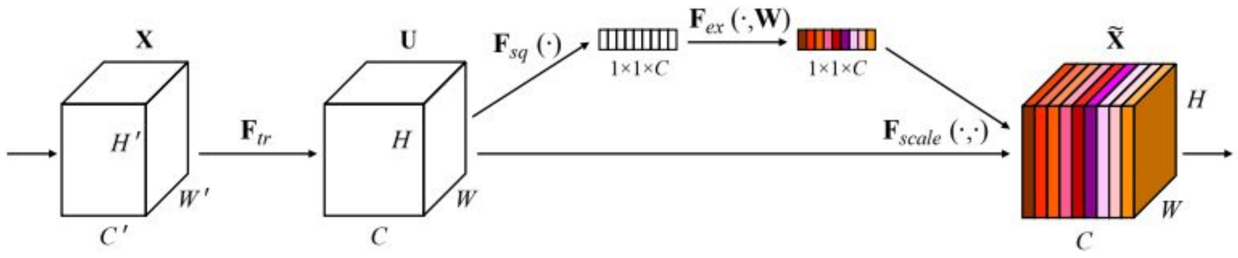
Improving ResNets...

Squeeze-and-Excitation Networks (SENet)

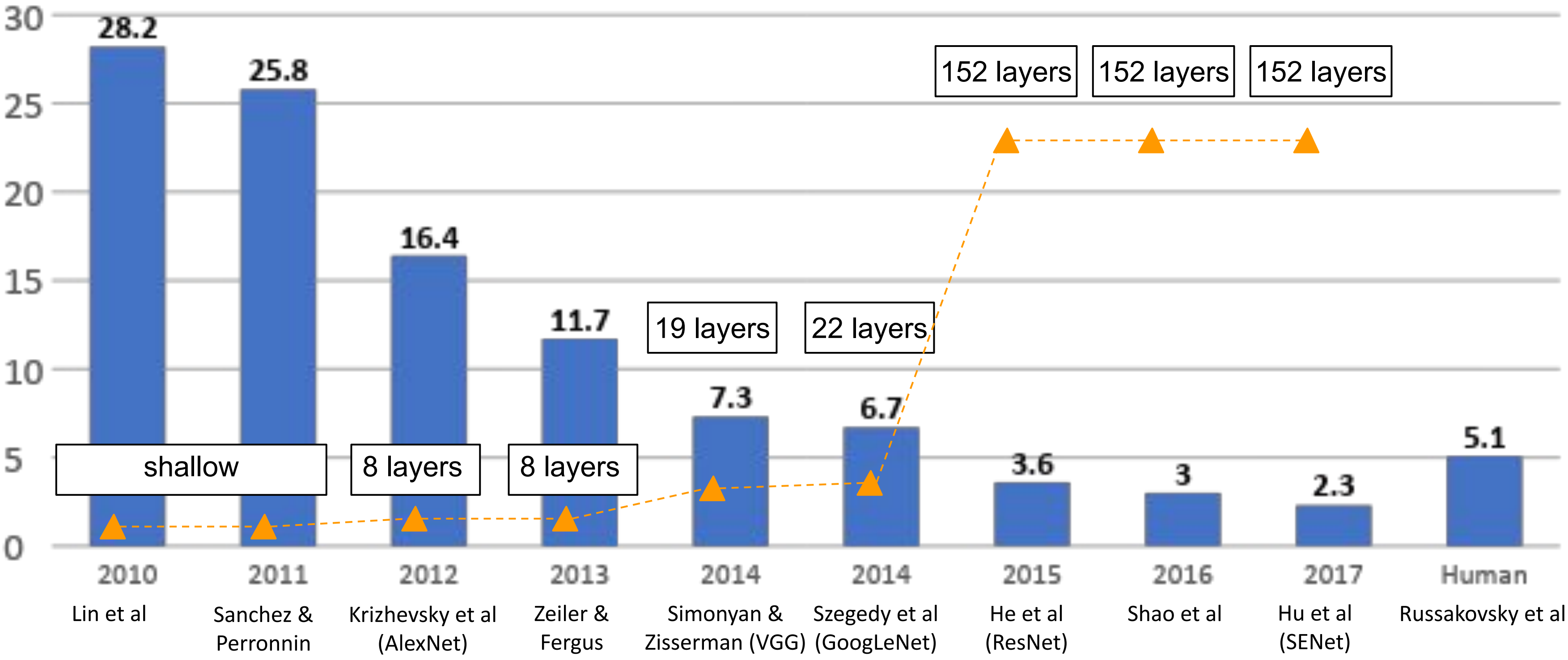
[Hu et al. 2017]

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

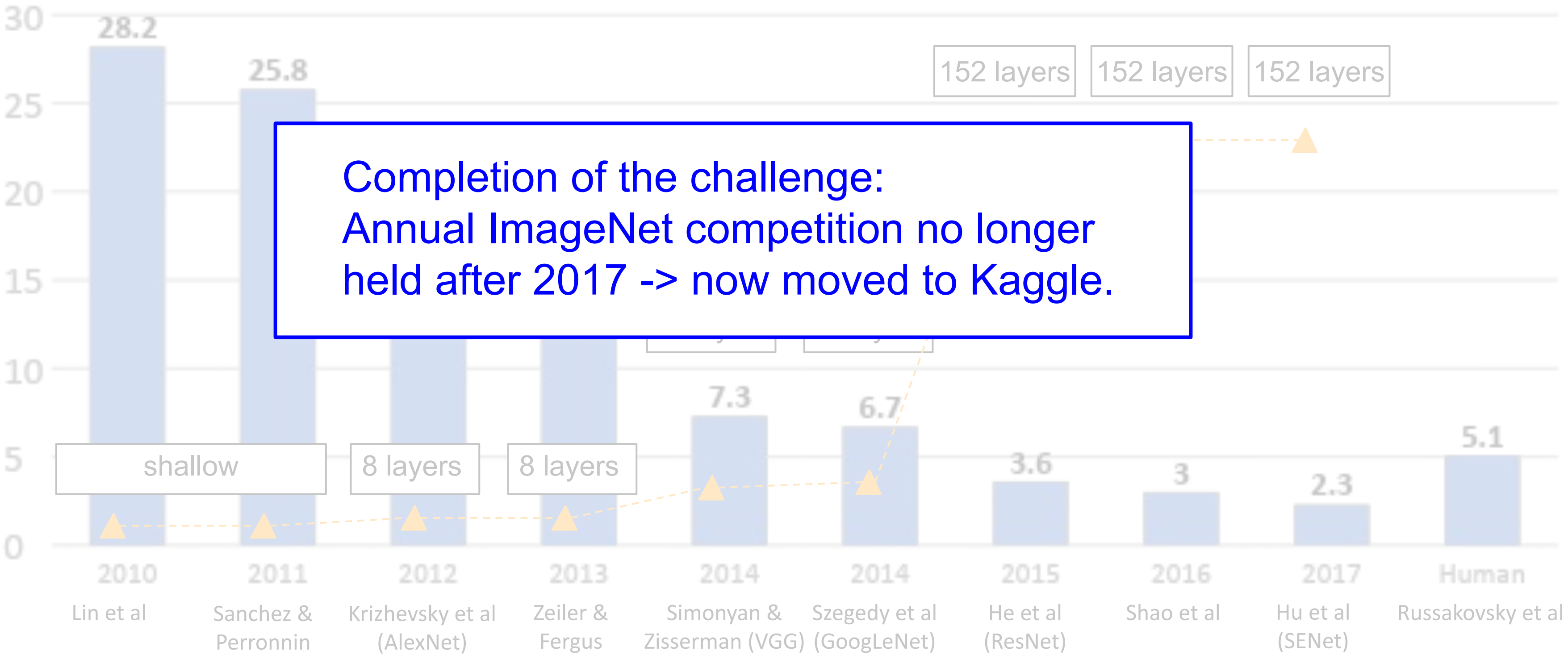




ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

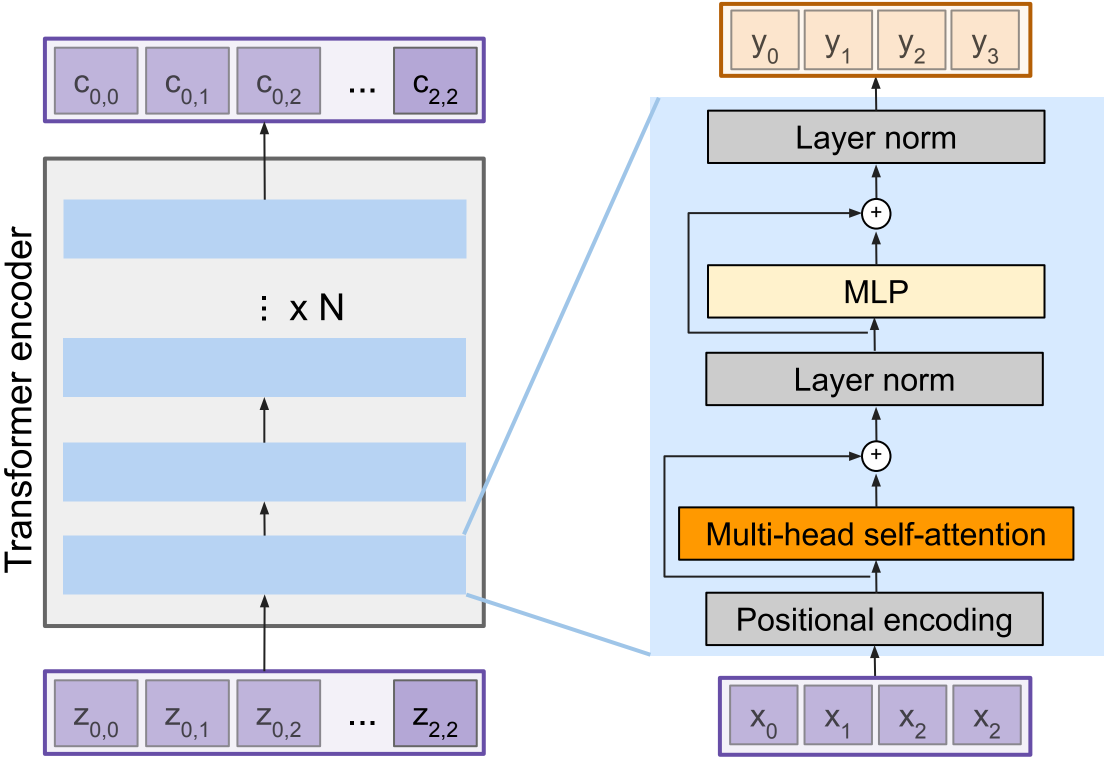


ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Completion of the challenge:
Annual ImageNet competition no longer held after 2017 -> now moved to Kaggle.

How have transformers affected architectures?



Transformer Encoder Block:

Inputs: Set of vectors **x**
Outputs: Set of vectors **y**

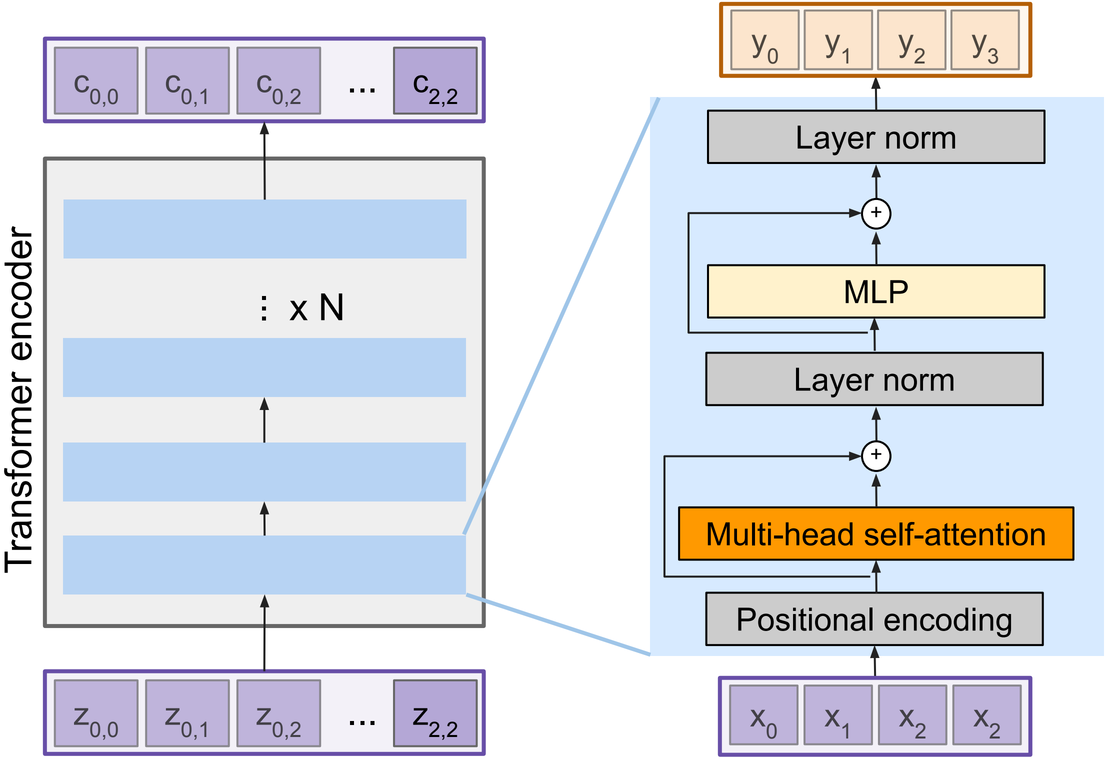
Self-attention is the only interaction between vectors.

Layer norm and MLP operate independently per vector.

Highly scalable, highly parallelizable, but high memory usage.

Vaswani et al, "Attention is all you need", NeurIPS 2017

Notice the residual connections!!

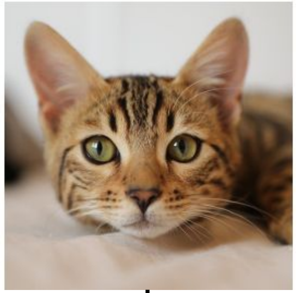


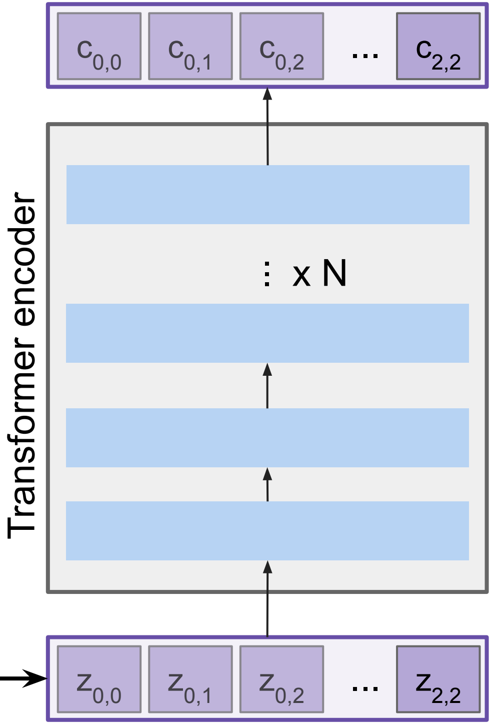
Residual connections inherited from ResNet's design.

Allows for better gradients to flow through all the transformers blocks.

Vaswani et al, "Attention is all you need", NeurIPS 2017

How to incorporate transformers to vision?





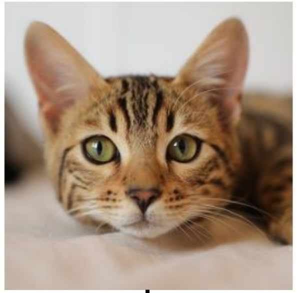
Cat image is free for commercial use

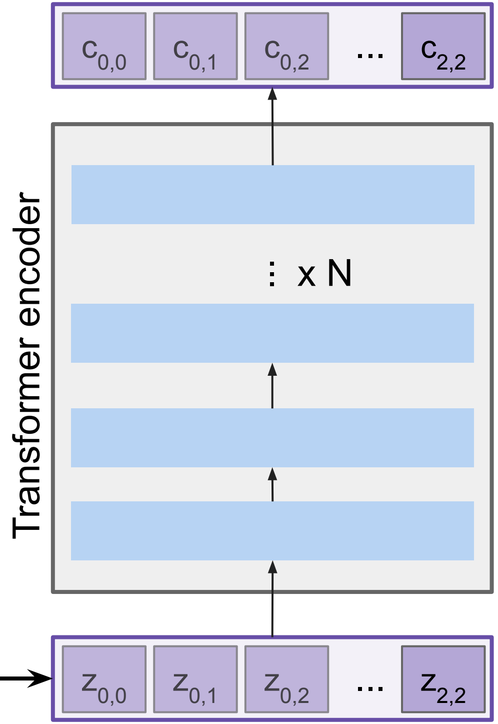
Idea #1: pass the image pixels into the transformer encoder.

So, each $z_{0,0}$ is a pixel.

What is the problem with this idea?

How to incorporate transformers to vision?





Cat image is free for commercial use

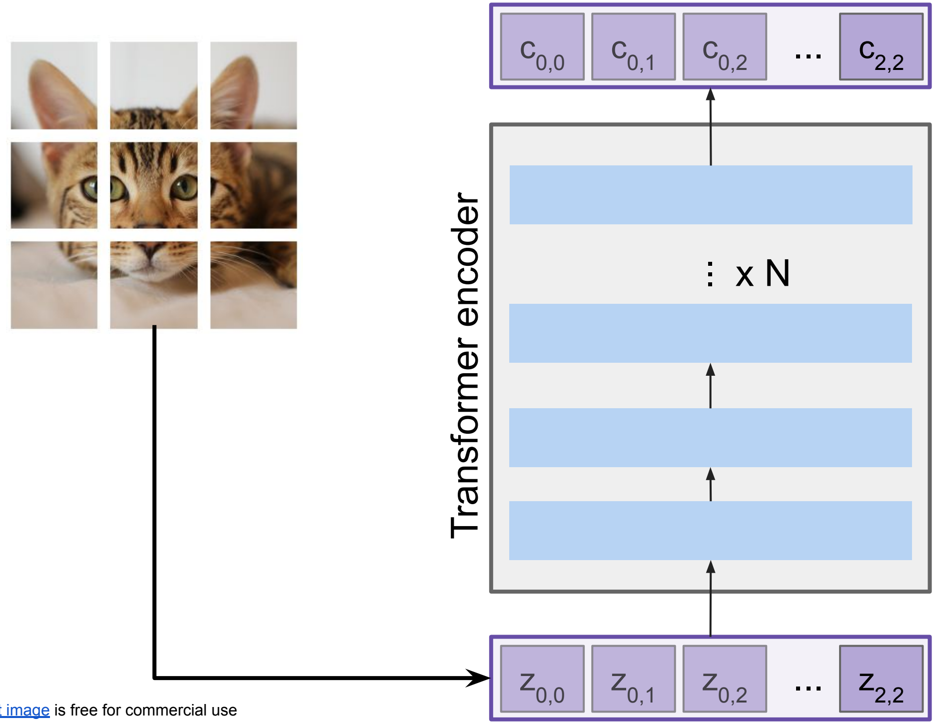
Idea #1: pass the image pixels into the transformer encoder.

So, each $z_{0,0}$ is a pixel.

Q. What is the problem with this idea?

A. Memory issue: Assume images are 224x224 pixels. This means that self attention will produce $224^4 = 10^9$ values!

How to incorporate transformers to vision?



Idea #2: Divide image into patches and pass those patches into the transformer

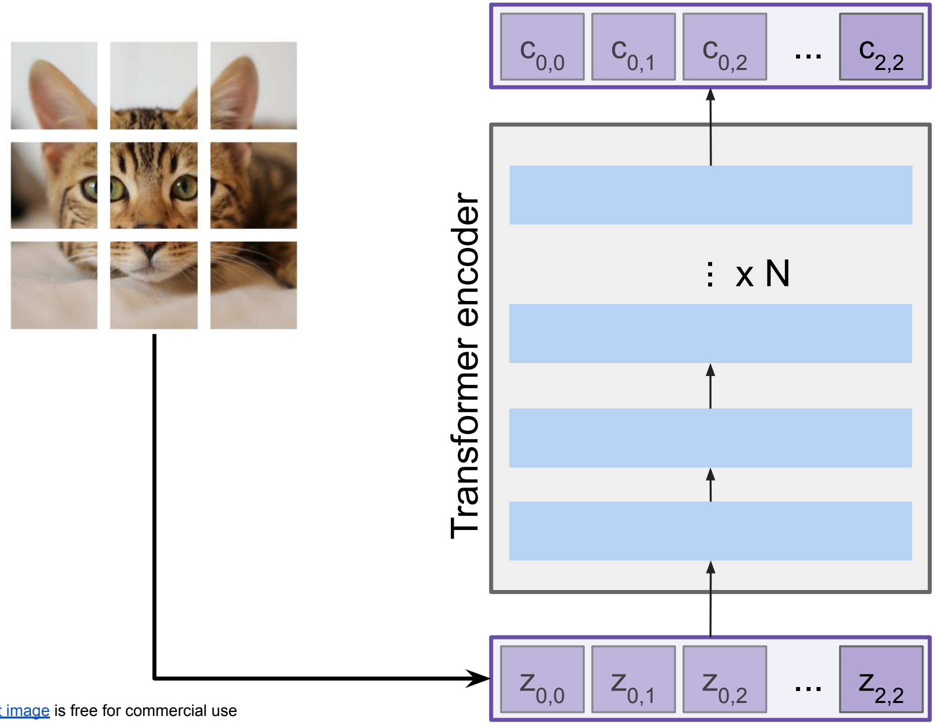
So, each $z_{0,0}$ is a 16x16x3 patch.

Q. What operation do you know already that operates over patches?

Cat image is free for commercial use

Dosovitskiyet al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

How to incorporate transformers to vision?



Idea #2: Divide image into patches and pass those patches into the transformer

So, each $z_{0,0}$ is a 16x16x3 patch.

Q. What operation do you know already that operates over patches?

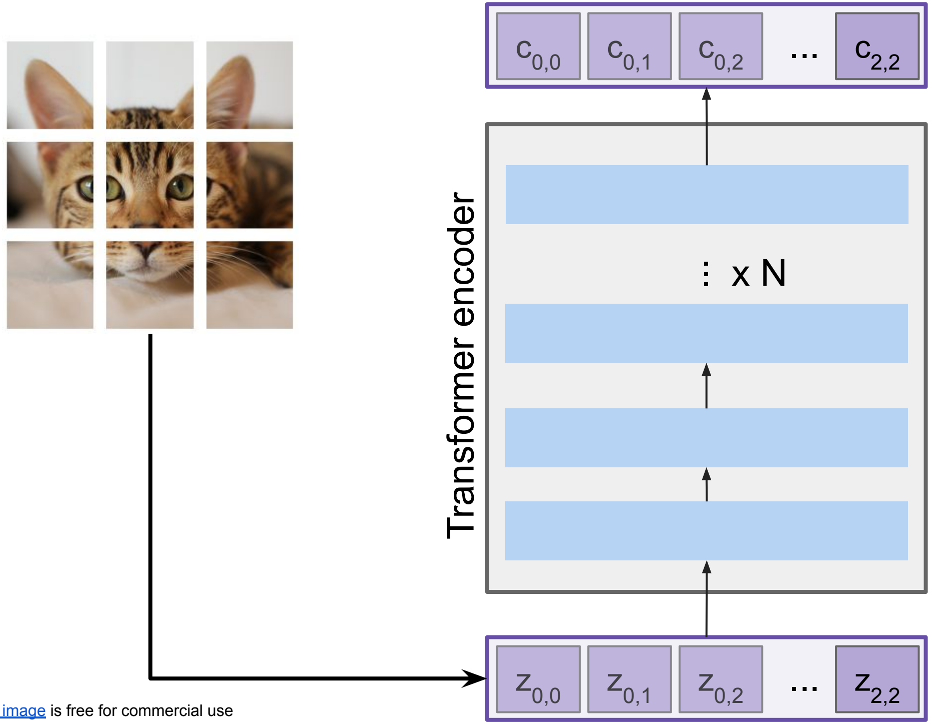
Yes it's a convolution.

Q. What is the kernel size and stride and padding?

Cat image is free for commercial use

Dosovitskiyet al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

How to incorporate transformers to vision?



Idea #2: Divide image into patches and pass those patches into the transformer

So, each $z_{0,0}$ is a 16x16x3 patch.

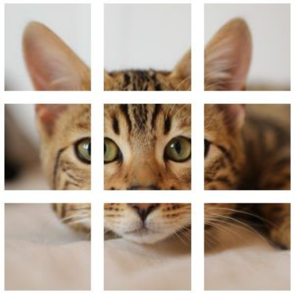
Q. Does this solve the memory problem?

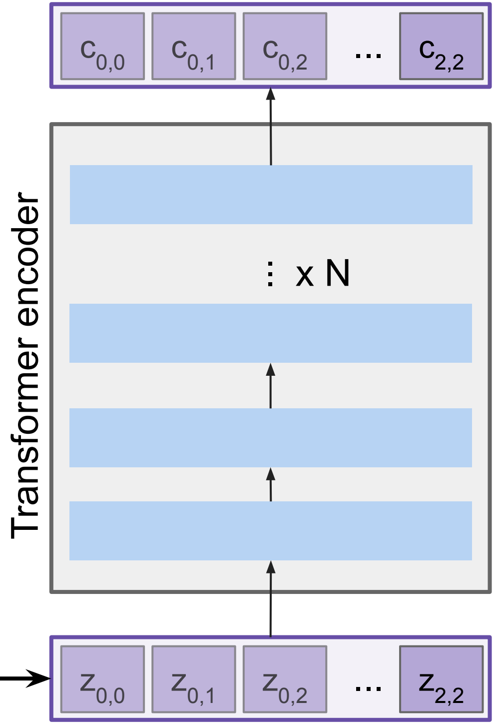
A. $14^2 \times 14^2 = 38416$, much less than 10^9

Cat image is free for commercial use

Dosovitskiyet al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Position encoding





Since transformers are permutation invariant, we want to add position encoding to each patch.

- Patches are 768D.
- Position encoding is some learned 768D.

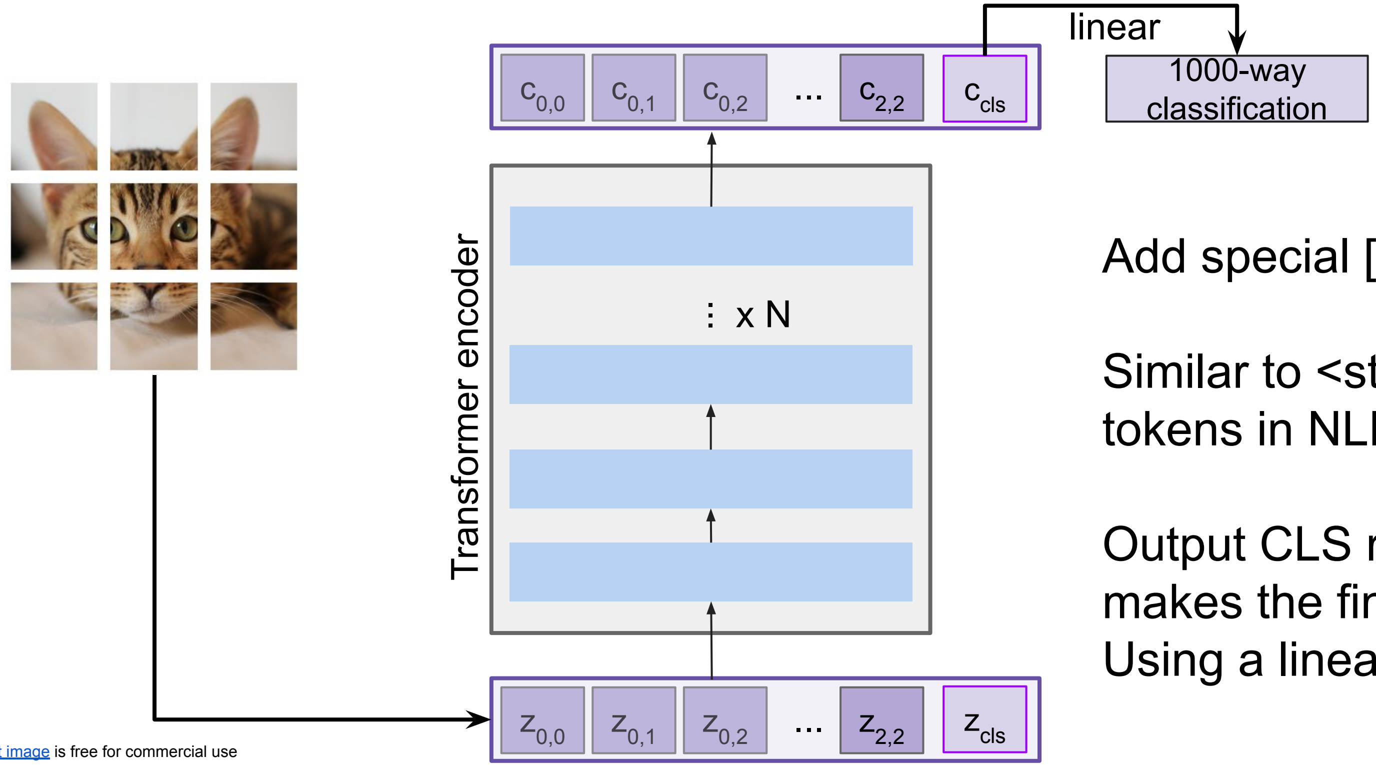
Pick any consistent ordering of patches (e.g. top left patch is always first).

Simply Add position encoding and patch representation.

Cat image is free for commercial use

Dosovitskiyet al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ICLR 2021

How to turn the output to a class prediction?



Add special [CLS] token.

Similar to <start> and <end> tokens in NLP.

Output CLS representation makes the final prediction Using a linear layer

Cat image is free for commercial use

Dosovitskiyet al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Common ViT architectures

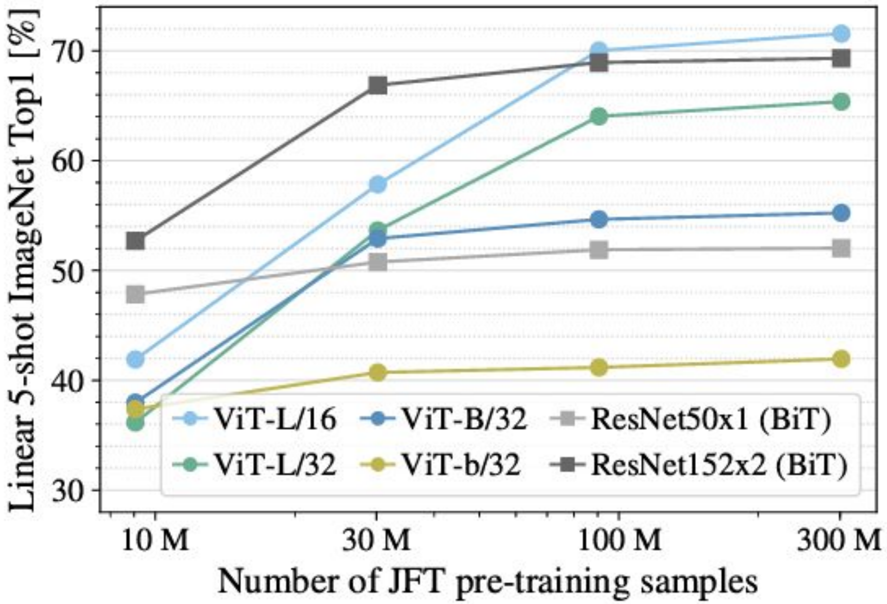
Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Common patch sizes: 32, 16, 14…
Smaller patches results in larger more powerful models.

Nomenclature: ViT-B/32 means that its a ViT model that uses Base values for layers, hidden size, mlp vize, and head. /32 means the input image patches are 32x32.

Dosovitskiyet al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ICLR 2021

Comparing ResNets with ViTs



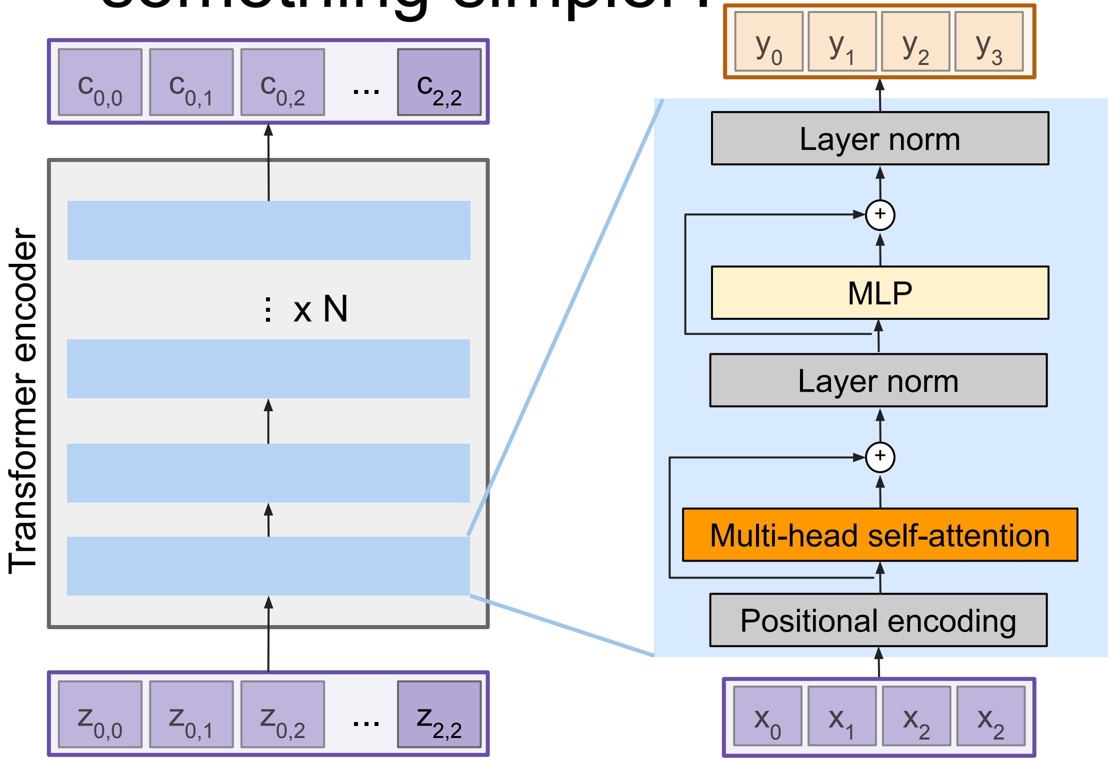
Models are initially trained on a large dataset called JFT-300M

And then the last linear layer is finetuned on ImageNet-1.5M

ViT performs worse when only 10M images are used from JFT. But ViT outperforms ResNets with larger training data (300M images from JFT).

Dosovitskiyet al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ICLR 2021

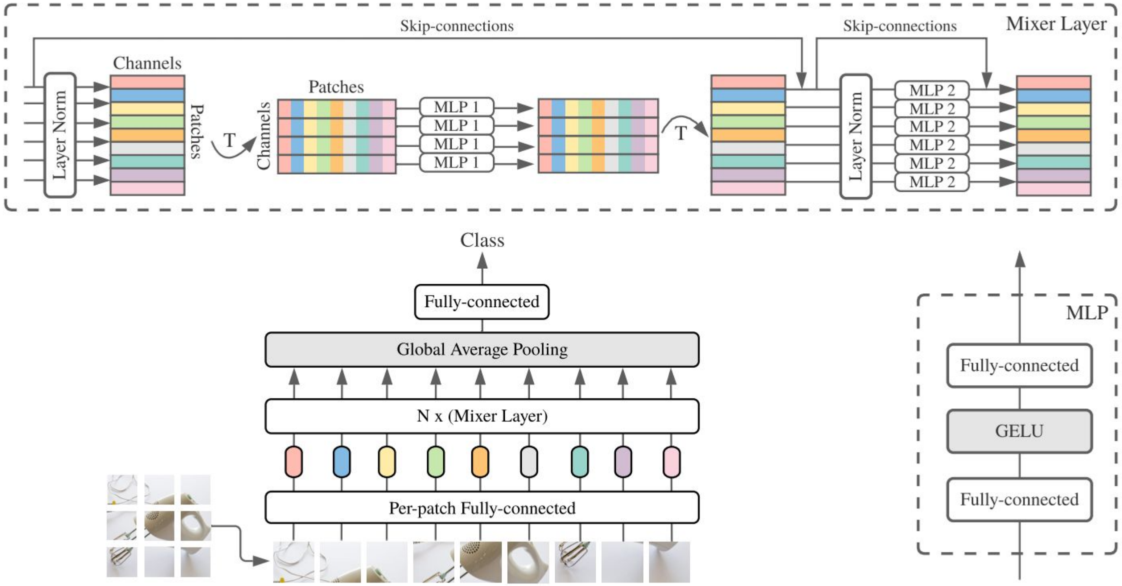
Self-attention is expensive… can we design something simpler?



Every self-attention is expensive. We want each input to "interact" with other tokens but can we simplify the operation a bit?

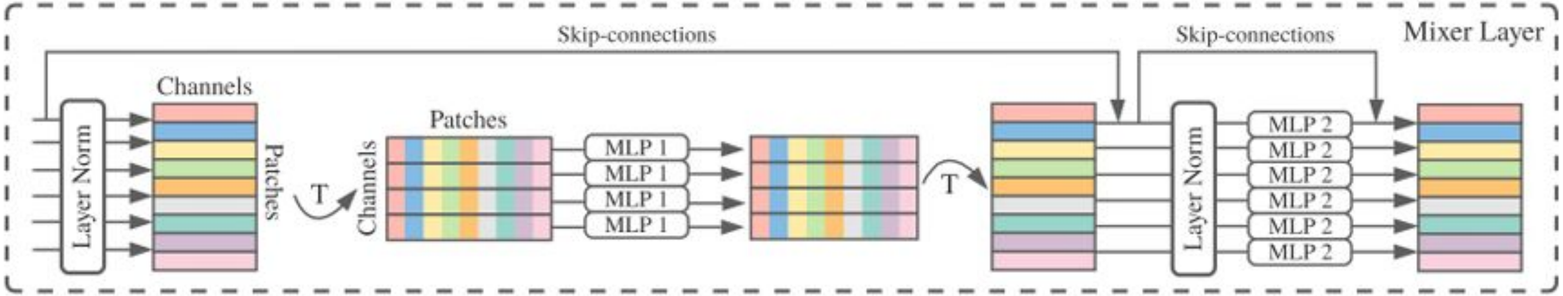
Vaswani et al, "Attention is all you need", NeurIPS 2017

MLP-Mixer: an all-MLP architecture



Tolstikhinet al, “MLP-Mixer: An all-MLP architecture for vision”, NeurIPS2021

MLP-Mixer: an all-MLP architecture



Input: N x C
N patches with
C channels each

MLP 1: C -> C,
apply to each of the
N patches

MLP 2: N -> N,
apply to each of the
C patches

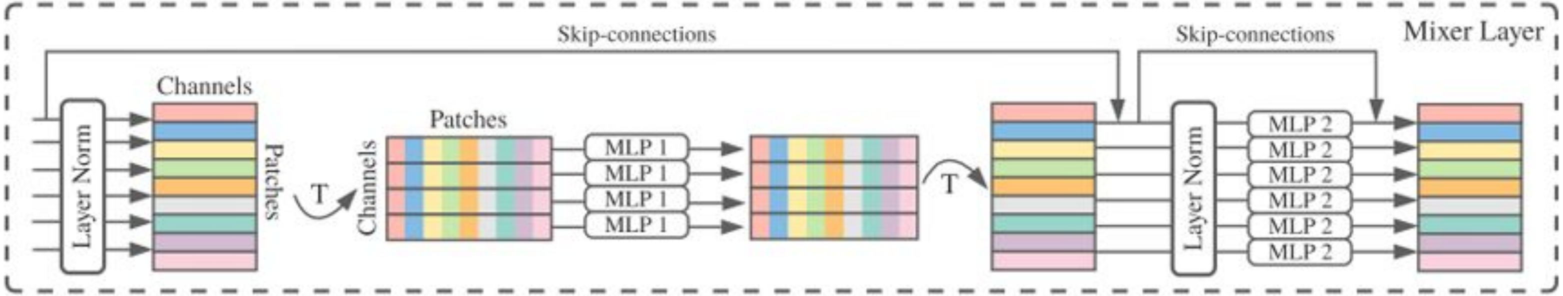
Tolstikhinet al, “MLP-Mixer: An all-MLP architecture for vision”, NeurIPS2021

MLP-Mixer: The MLPs are sort of like **convs**

Cool idea; but hasn't taken off yet.

Equivalent to Conv(1x1, C->C, stride=1)

Equivalent to Conv($N^{1/2}$ x $N^{1/2}$, C->C, groups=C)



Input: N x C
N patches with
C channels each

MLP 1: C -> C,
apply to each of the
N patches

MLP 2: N -> N,
apply to each of the
C patches

Tolstikhinet al, "MLP-Mixer: An all-MLP architecture for vision", NeurIPS2021

117

MLP-Mixer: Many concurrent and followups

Touvronet al, “ResMLP: Feedforward Networks for Image Classification with Data-Efficient Training”, arXiv2021,
https://arxiv.org/abs/2105.03404

Tolstikhinet al, “MLP-Mixer: An all-MLP architecture for vision”, NeurIPS2021,
https://arxiv.org/abs/2105.01601

Liu et al, “Pay Attention to MLPs”, NeurIPS2021, https://arxiv.org/abs/2105.08050

Yu et al, “S2-MLP: Spatial-Shift MLP Architecture for Vision”, WACV 2022,
https://arxiv.org/abs/2106.07477

Chen et al, “CycleMLP: AMLP-like Architecture for Dense Prediction”, ICLR 2022,
https://arxiv.org/abs/2107.10224

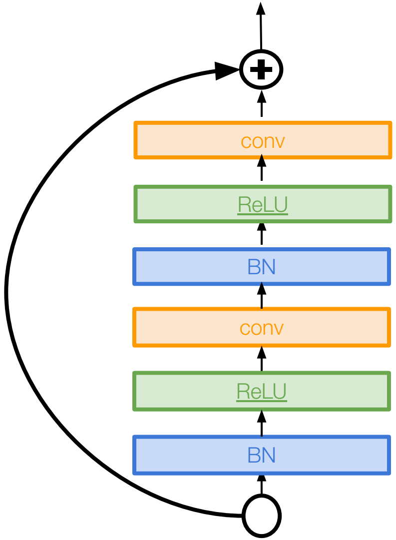
But research has continued since ImageNet

(Will go over following slides if time,
Otherwise skip to the summary slides in the end.)

Identity Mappings in Deep Residual Networks

[He et al. 2016]

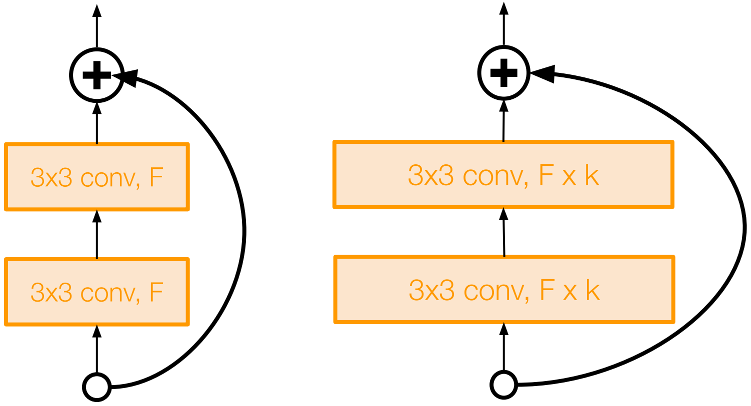
- Improved ResNet block design from creators of ResNet
- Creates a more direct path for propagating information throughout network
- Gives better performance



Wide Residual Networks

[Zagoruyko et al. 2016]

- Argues that residuals are the important factor, not depth
- User wider residual blocks (F x k filters instead of F filters in each layer)
- 50-layer wide ResNet outperforms 152-layer original ResNet
- Increasing width instead of depth more computationally efficient (parallelizable)



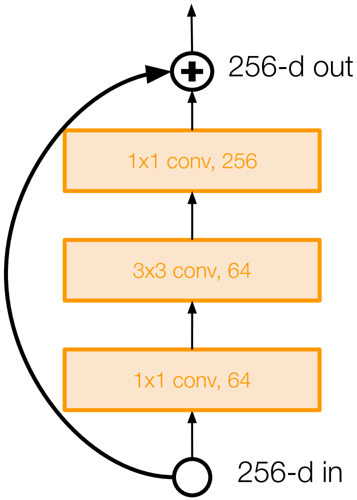
Basic residual block Wide residual block

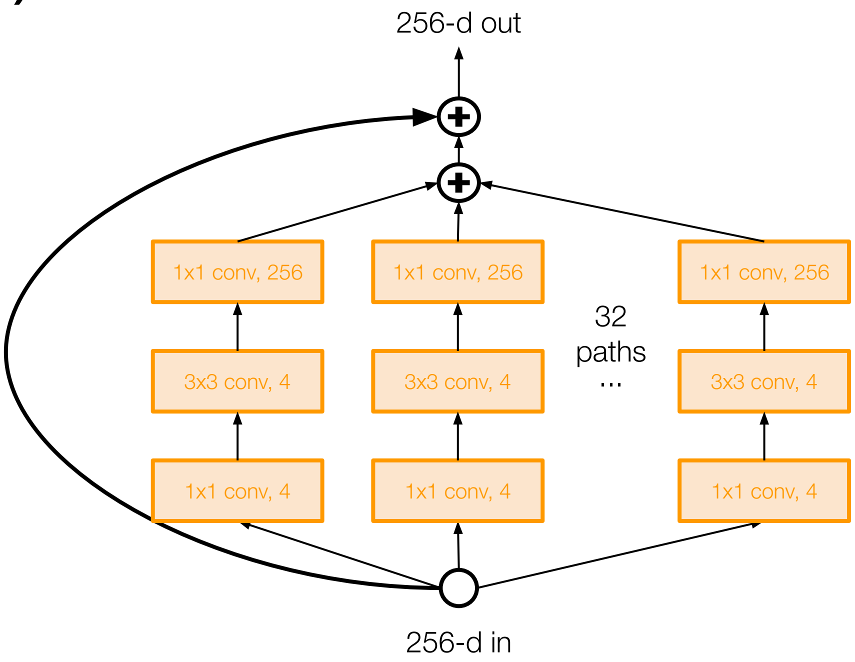
Improving ResNets...

Aggregated Residual Transformations for Deep Neural Networks (ResNeXt)

[Xie et al. 2016]

- Also from creators of ResNet
- Increases width of residual block through multiple parallel pathways ("cardinality")
- Parallel pathways similar in spirit to Inception module



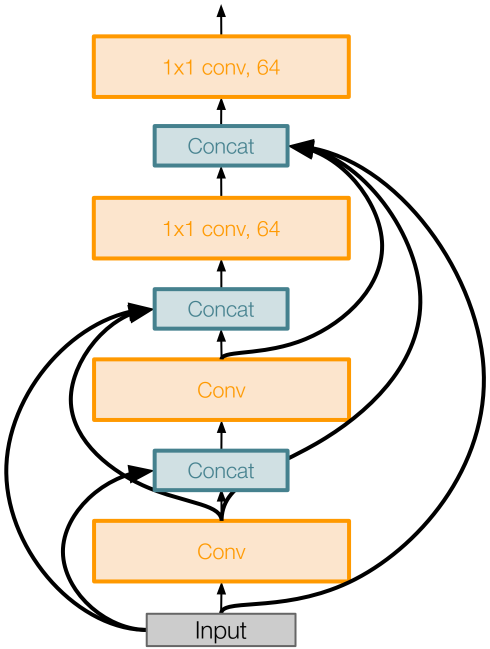


Other ideas...

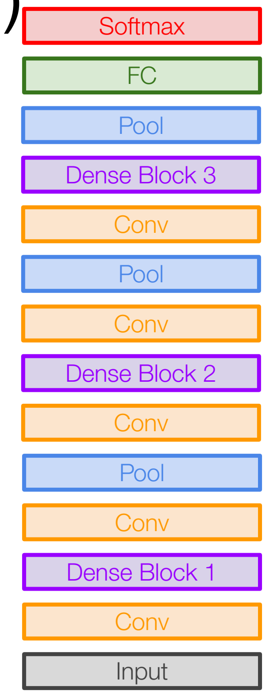
Densely Connected Convolutional Networks (DenseNet)

[Huang et al. 2017]

- Dense blocks where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse
- Showed that shallow 50-layer network can outperform deeper 152 layer ResNet



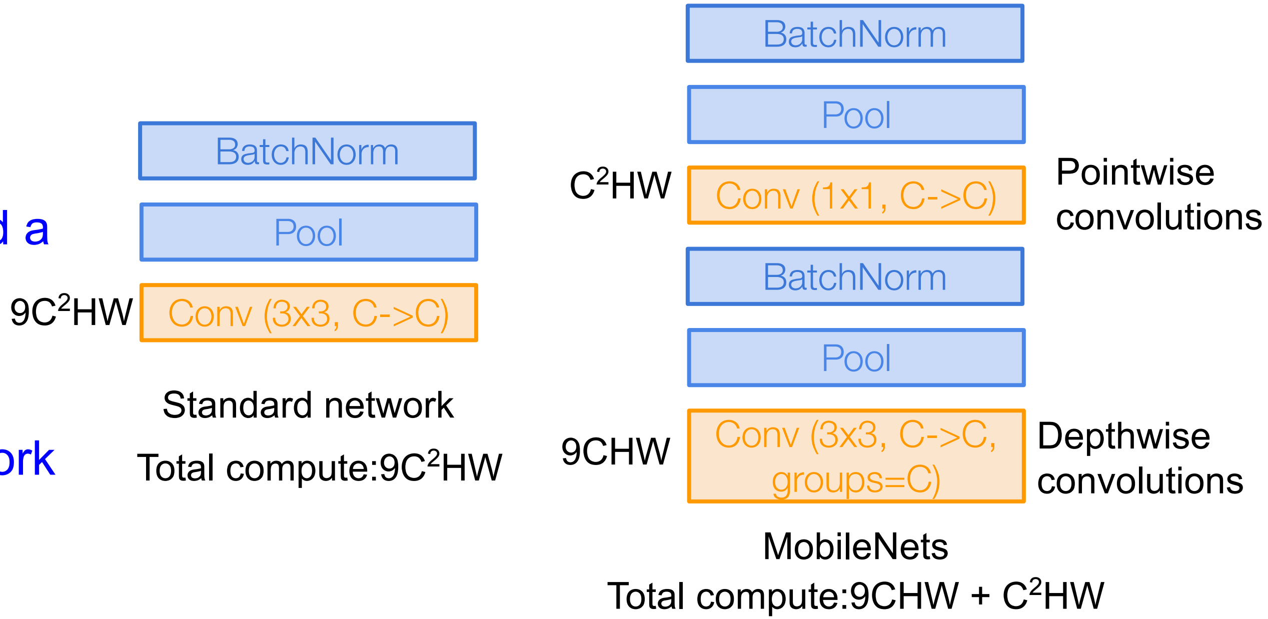
Dense Block



Efficient networks...

MobileNets: Efficient Convolutional Neural Networks for Mobile Applications *[Howard et al. 2017]*

- Depthwise separable convolutions replace standard convolutions by factorizing them into a depthwise convolution and a 1x1 convolution
- Much more efficient, with little loss in accuracy
- Follow-up MobileNetV2 work in 2018 (Sandler et al.)
- ShuffleNet: Zhang et al, CVPR 2018

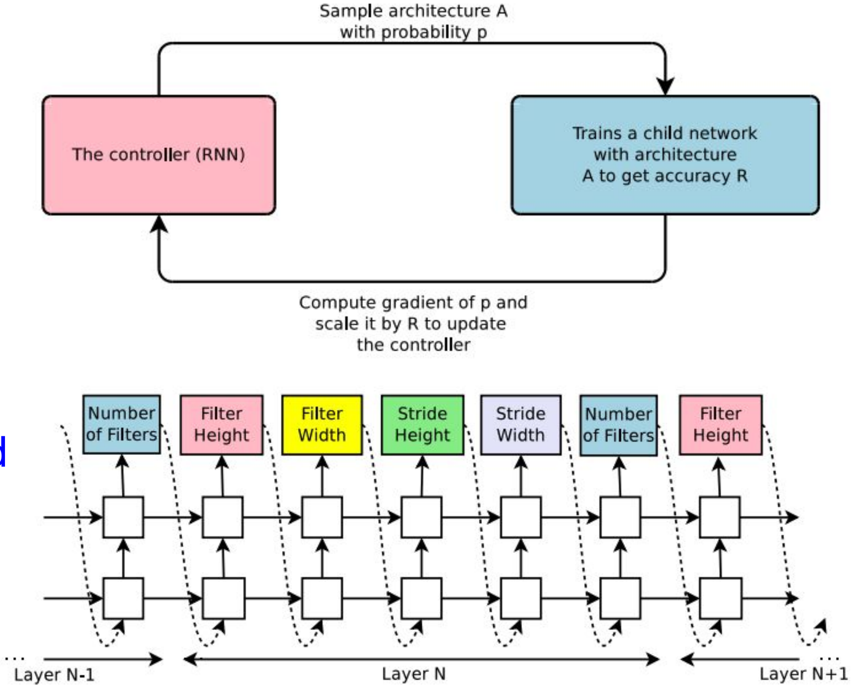


Learning to search for network architectures...

Neural Architecture Search with Reinforcement Learning (NAS)

[Zoph et al. 2016]

- "Controller" network that learns to design a good network architecture (output a string corresponding to network design)
- Iterate:
  1) Sample an architecture from search space
  2) Train the architecture to get a "reward" R corresponding to accuracy
  3) Compute gradient of sample probability, and scale by R to perform controller parameter update (i.e. increase likelihood of good architecture being sampled, decrease likelihood of bad architecture)

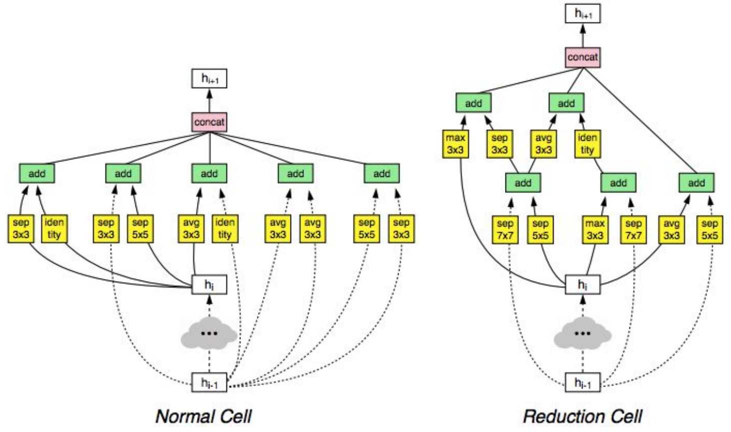


Learning to search for network architectures...

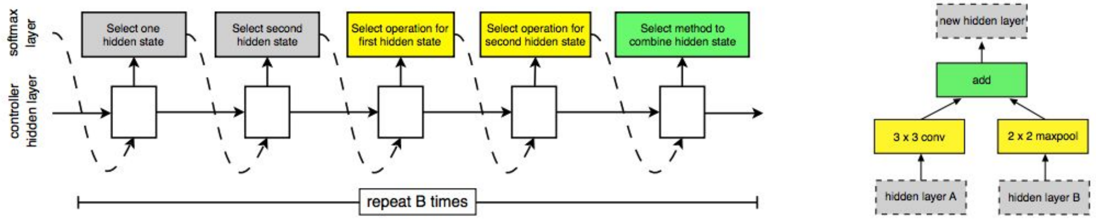
Learning Transferable Architectures for Scalable Image Recognition

[Zoph et al. 2017]

- Applying neural architecture search (NAS) to a large dataset like ImageNet is expensive
- Design a search space of building blocks ("cells") that can be flexibly stacked
- NASNet: Use NAS to find best cell structure on smaller CIFAR-10 dataset, then transfer architecture to ImageNet



- Many follow-up works in this space e.g. AmoebaNet (Real et al. 2019) and ENAS (Pham, Guan et al. 2018)



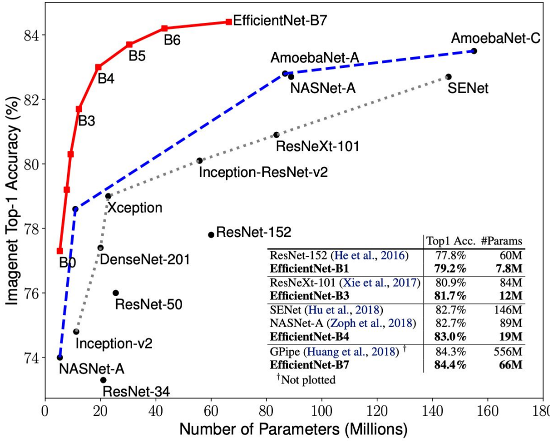
But sometimes smart heuristic is better than NAS ...

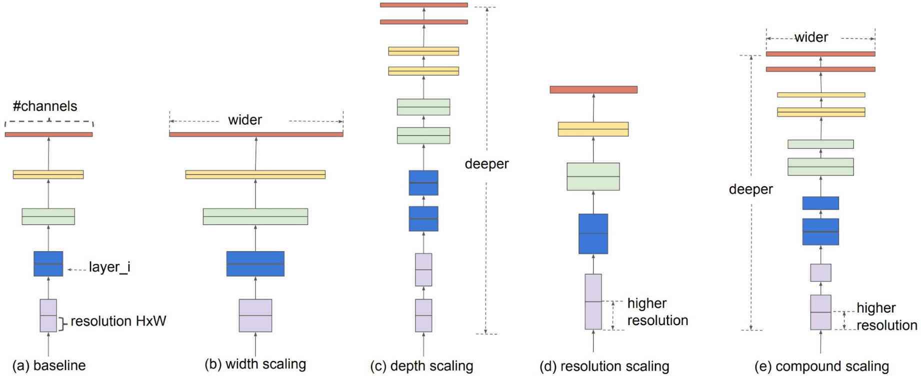
EfficientNet: Smart Compound Scaling

[Tan and Le. 2019]

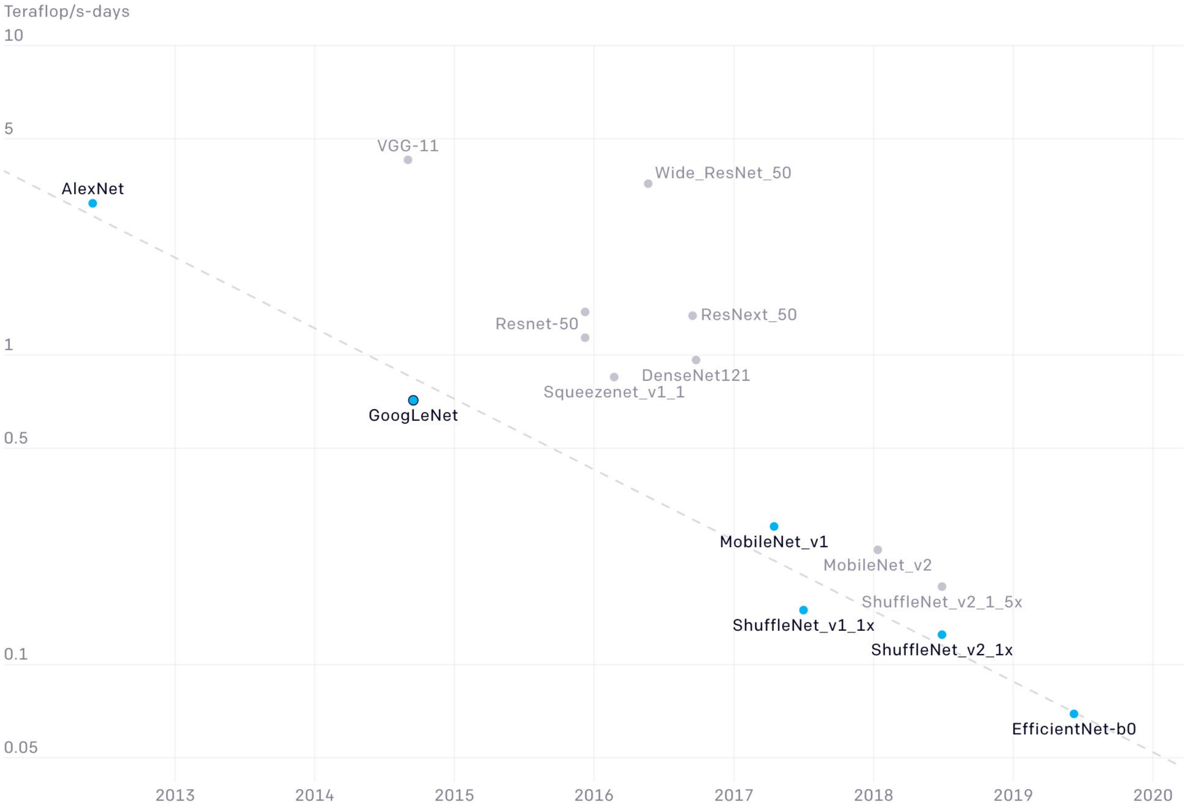
- Increase network capacity by scaling width, depth, and resolution, while balancing accuracy and efficiency.
- Search for optimal set of compound scaling factors given a compute budget (target memory & flops).
- Scale up using smart heuristic rules

$$
\begin{aligned}
\text{depth: } & d = \alpha^{\phi} \\
\text{width: } & w = \beta^{\phi} \\
\text{resolution: } & r = \gamma^{\phi} \\
\text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\
& \alpha \geq 1, \beta \geq 1, \gamma \geq 1
\end{aligned}
$$





Efficient networks...



https://openai.com/blog/ai-and-efficiency/

Summary: Modern Architectures

Case Studies

- AlexNet
- VGG
- GoogLeNet
- ResNet
- Vit
- MLP-Mixer

Also....

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

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
ViT is the current favorite architecture but requires a lot of compute and data
MLP-Mixers have presented an alternative to transformers but they haven't taken off.

Summary: Modern Architectures

- **ResNet-50** and **ViT** currently good defaults to use

- Next time: Structure prediction

Next time: Structured Prediction