

Computer Vision

CNN Architectures

Dr. Mrinmoy Ghorai

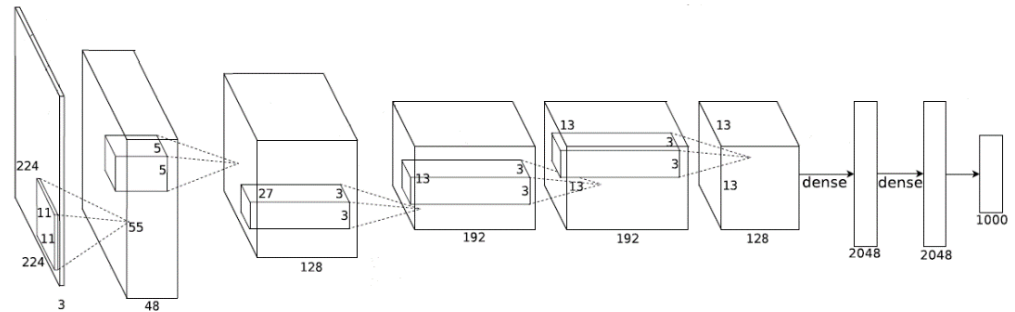
**Indian Institute of Information Technology
Sri City, Chittoor**



This Class

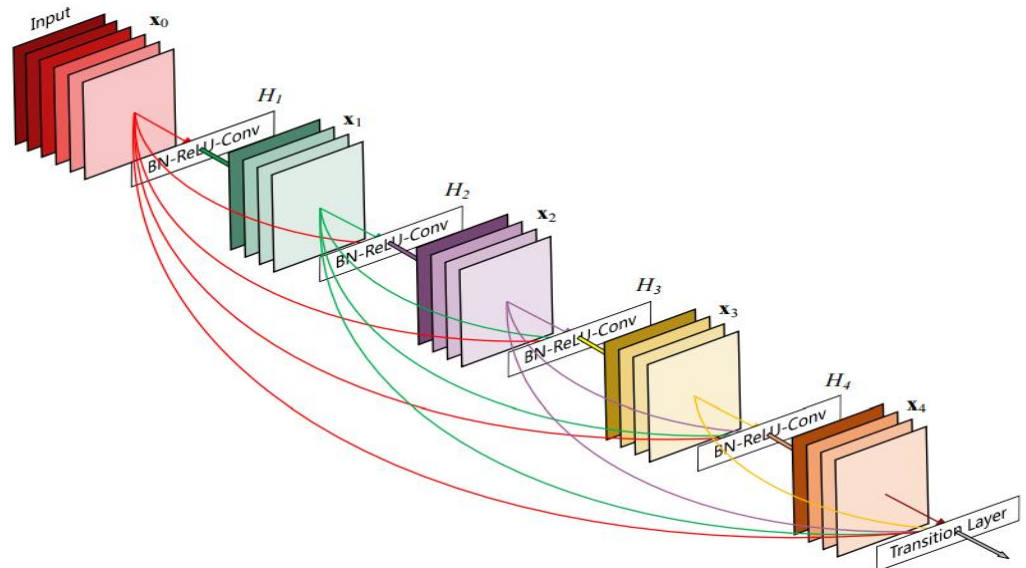
CNN Architectures: Plain Models

- LeNet
- AlexNet
- ZFNet
- VggNet
- Network in Network



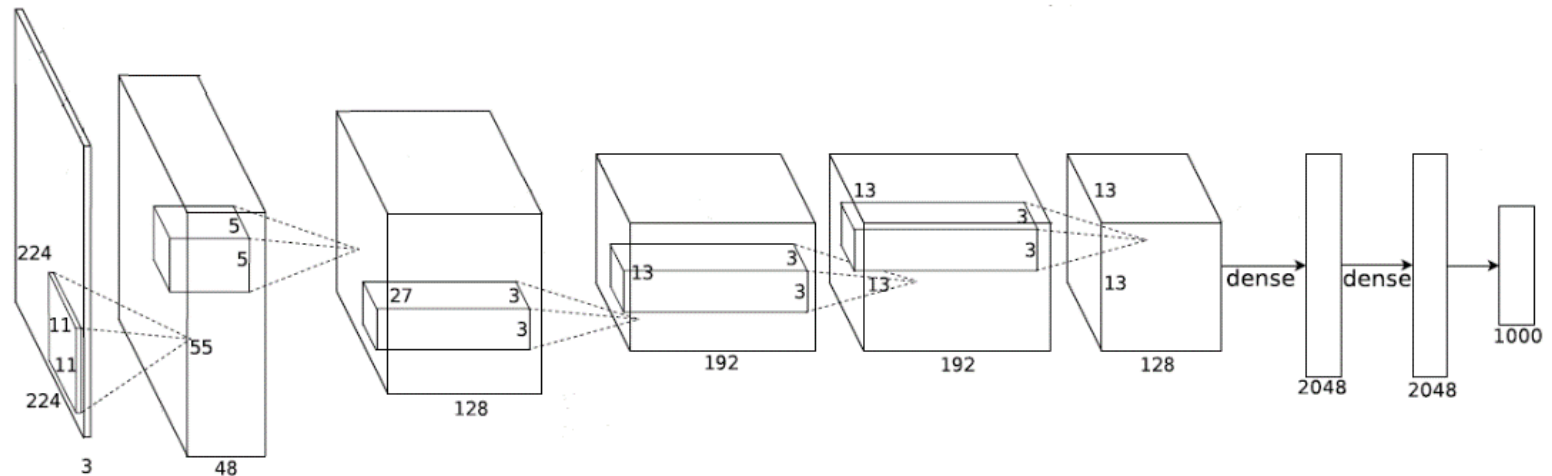
CNN Architectures: DAG Models

- GoogLeNet
- ResNet
- Pre-act ResNet
- SENet
- DenseNet
- ResNetXt
- Etc.

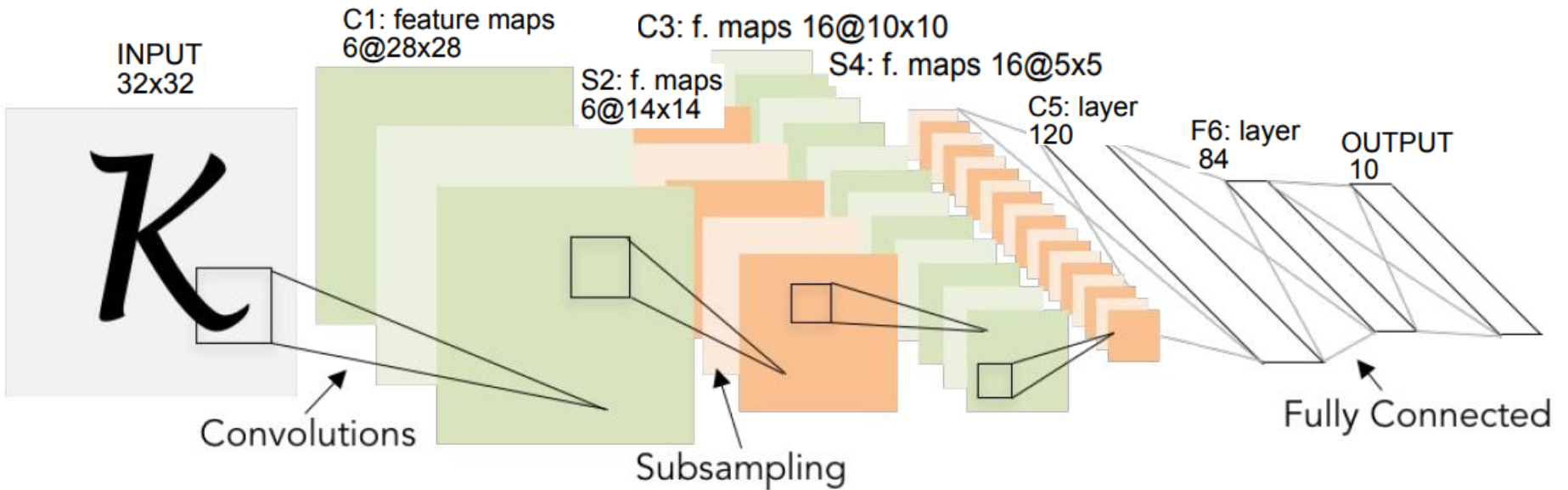


CNN Architectures: Plain Models

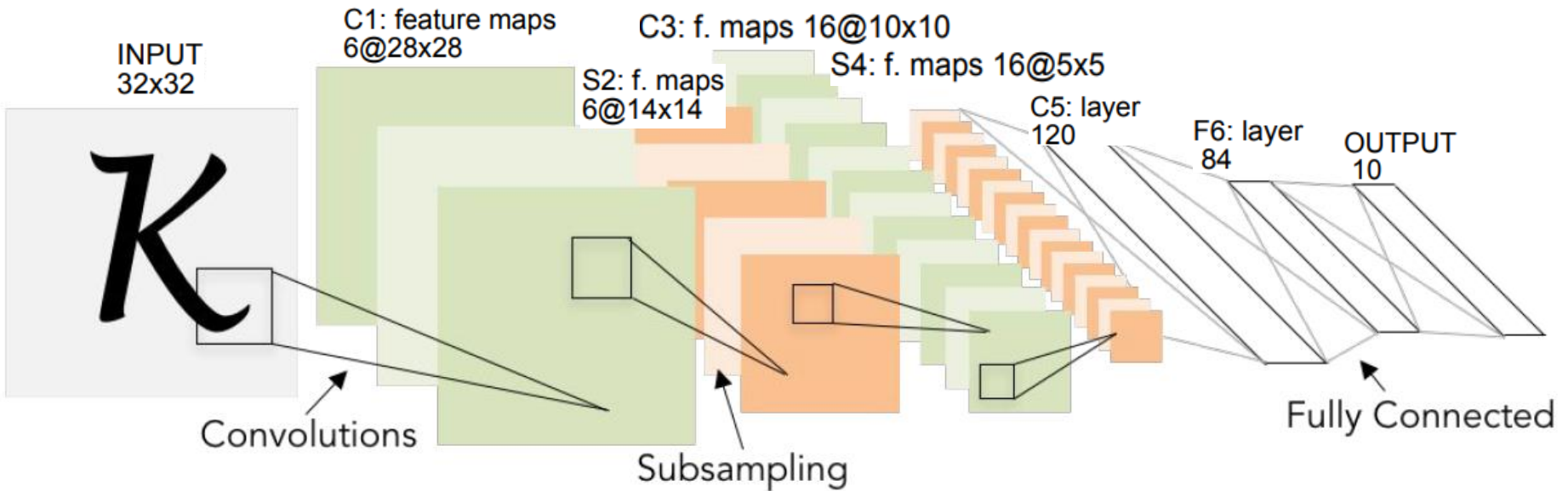
- LeNet
- AlexNet
- ZFNet
- VggNet
- Network in Network



Review: LeNet-5



Review: LeNet-5

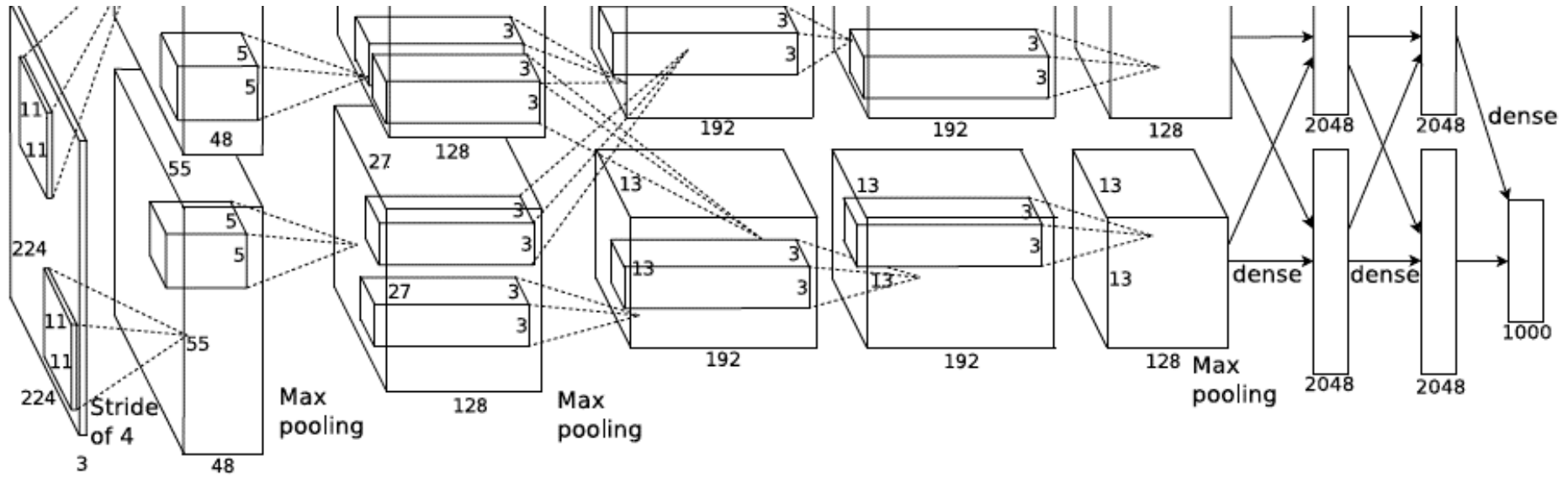


Conv filters are 5x5, applied at stride 1

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

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

AlexNet



Architecture:

CONV1

MAX POOL1

NORM1(Local Response Normalization)

CONV2

MAX POOL2

NORM2(Local Response Normalization)

CONV3

CONV4

CONV5

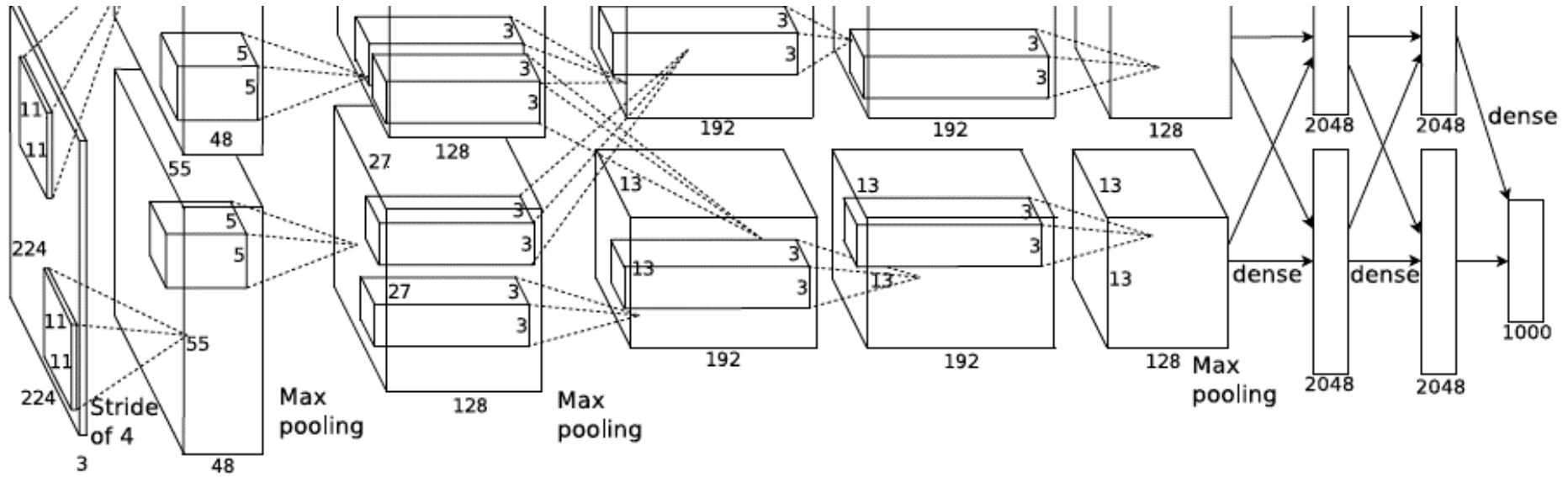
Max POOL3

FC6

FC7

FC8

AlexNet



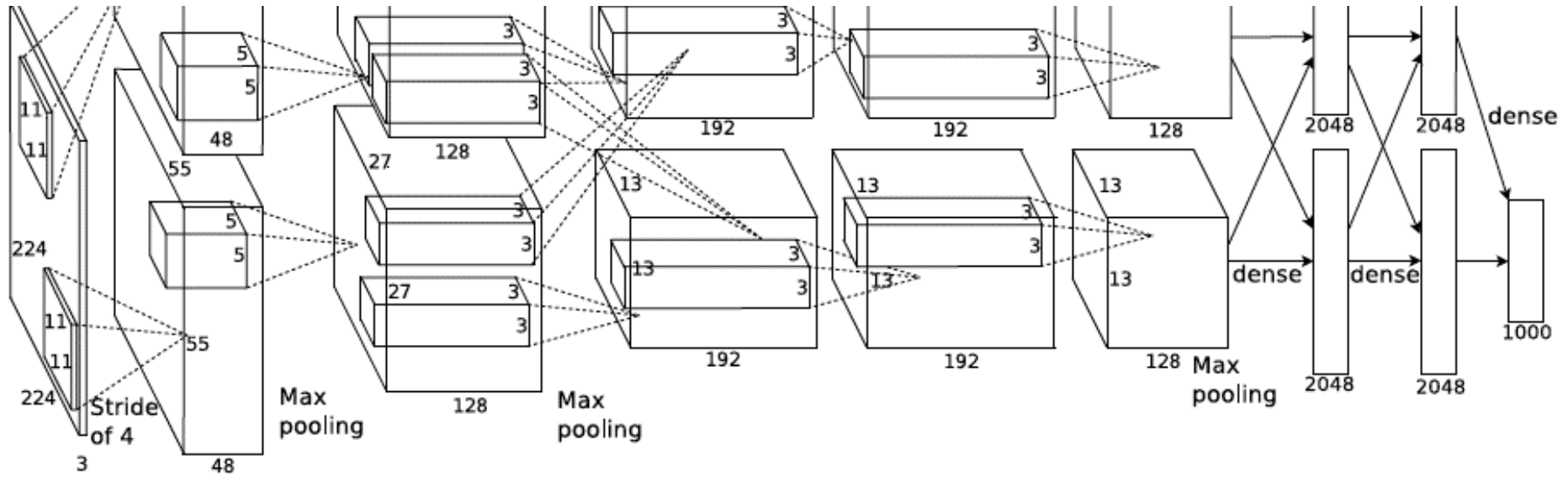
Input: 227x227x3 images

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

=>

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

AlexNet



Input: 227x227x3 images

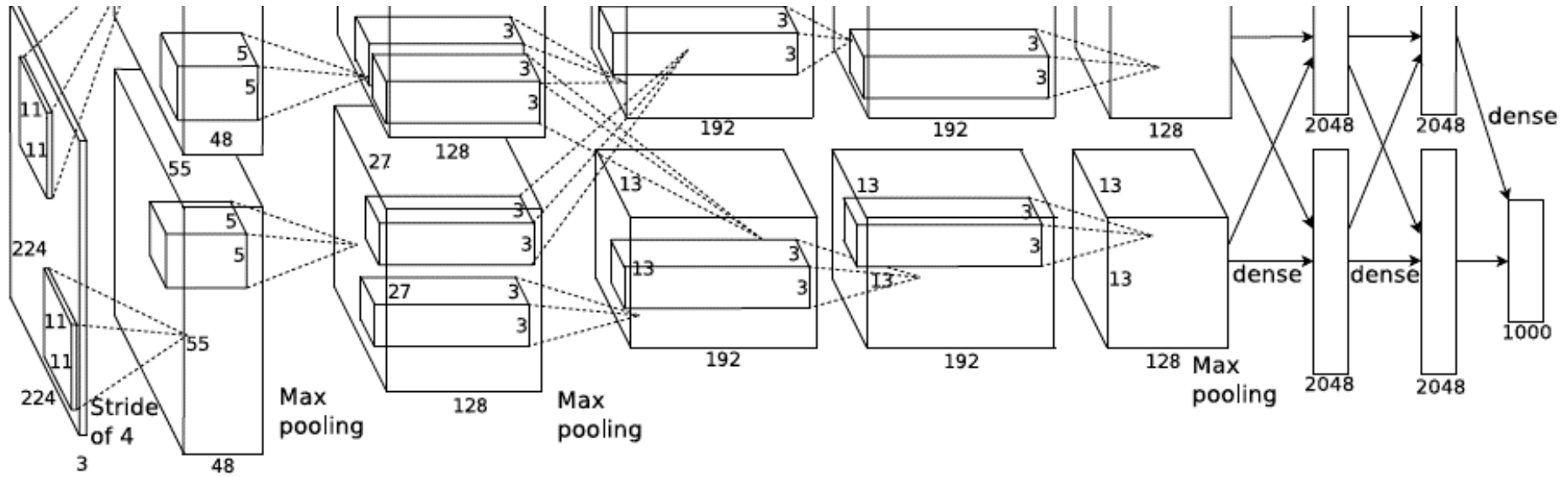
First layer (CONV1): 96 11x11 filters applied at stride 4

=>

Output volume [**55x55x96**]

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

AlexNet



Input: 227x227x3 images

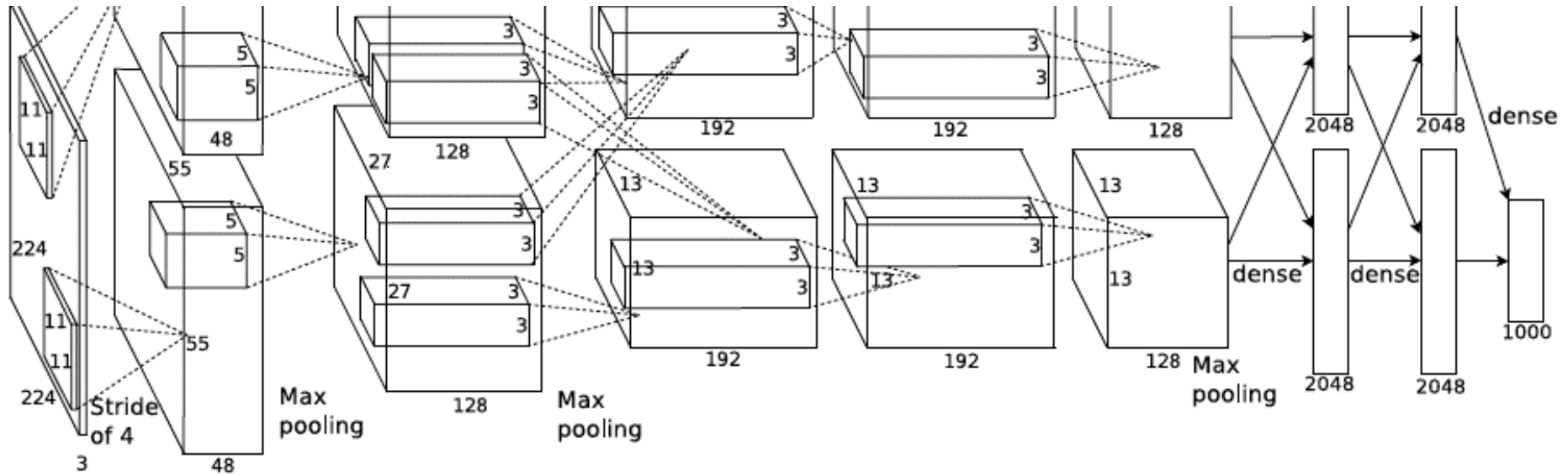
First layer (CONV1): 96 11x11 filters applied at stride 4

=>

Output volume [**55x55x96**]

Parameters: $(11*11*3)*96 = \mathbf{35K}$

AlexNet



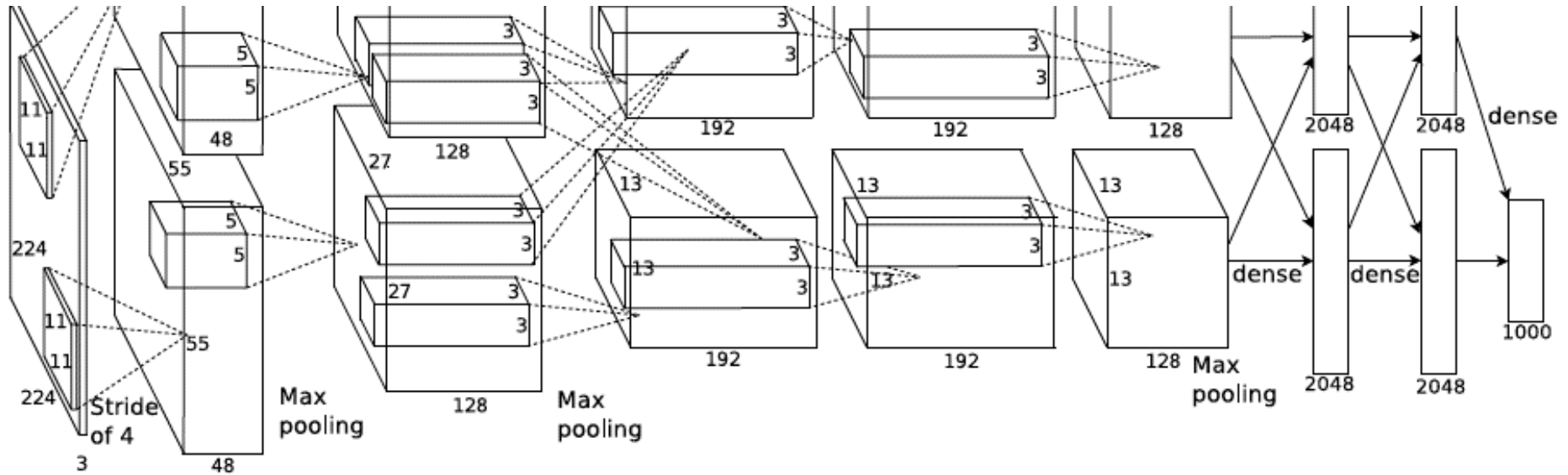
Input: 227x227x3 images

After CONV1: 55x55x96

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

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

AlexNet



Input: 227x227x3 images

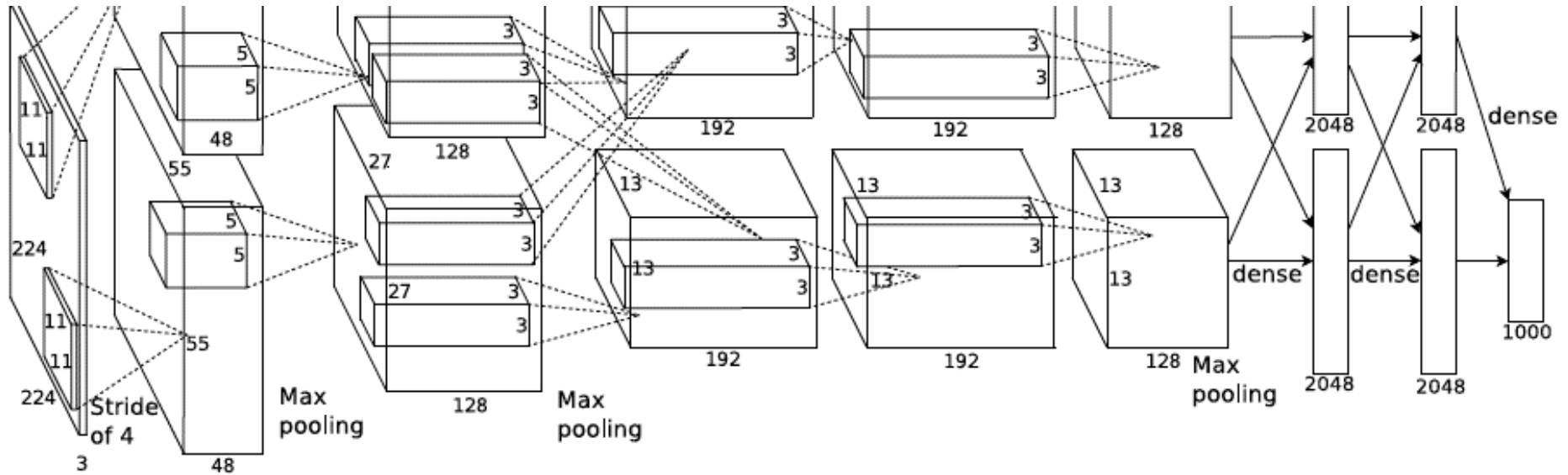
After CONV1: 55x55x96

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

Output volume [**27x27x96**]

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

AlexNet



Input: $227 \times 227 \times 3$ images

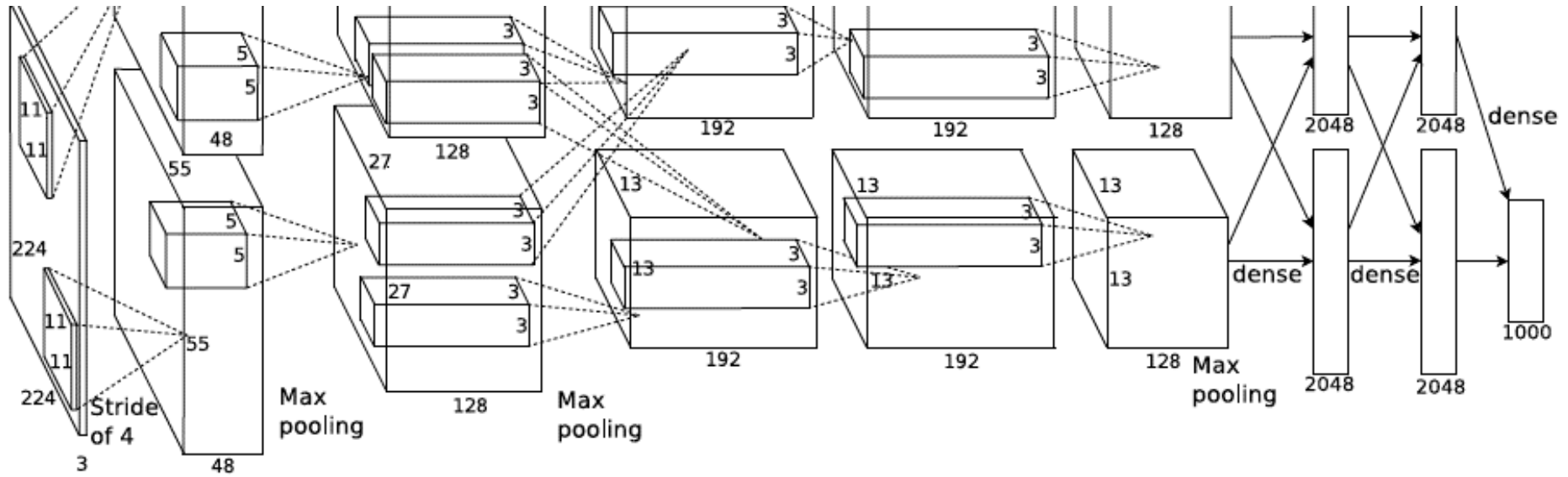
After CONV1: $55 \times 55 \times 96$

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

Output volume [**$27 \times 27 \times 96$**]

Parameters: 0!

AlexNet



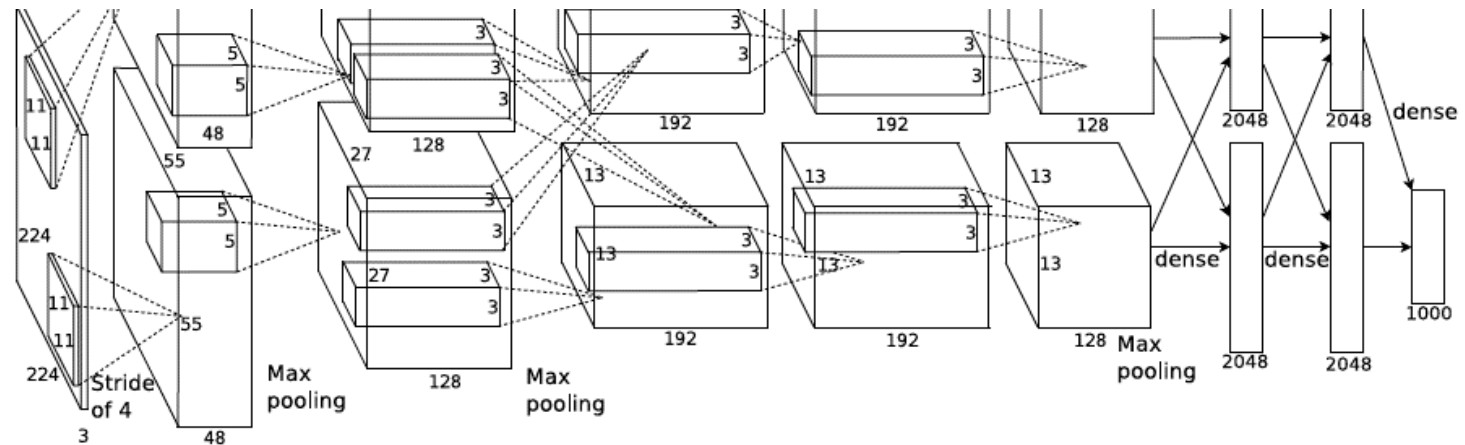
Input: $227 \times 227 \times 3$ images

After CONV1: $55 \times 55 \times 96$

After POOL1: $27 \times 27 \times 96$

...

AlexNet



Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0

[27x27x96] **MAX POOL1**: 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2

[13x13x256] **MAX POOL2**: 3x3 filters at stride 2

[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1

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

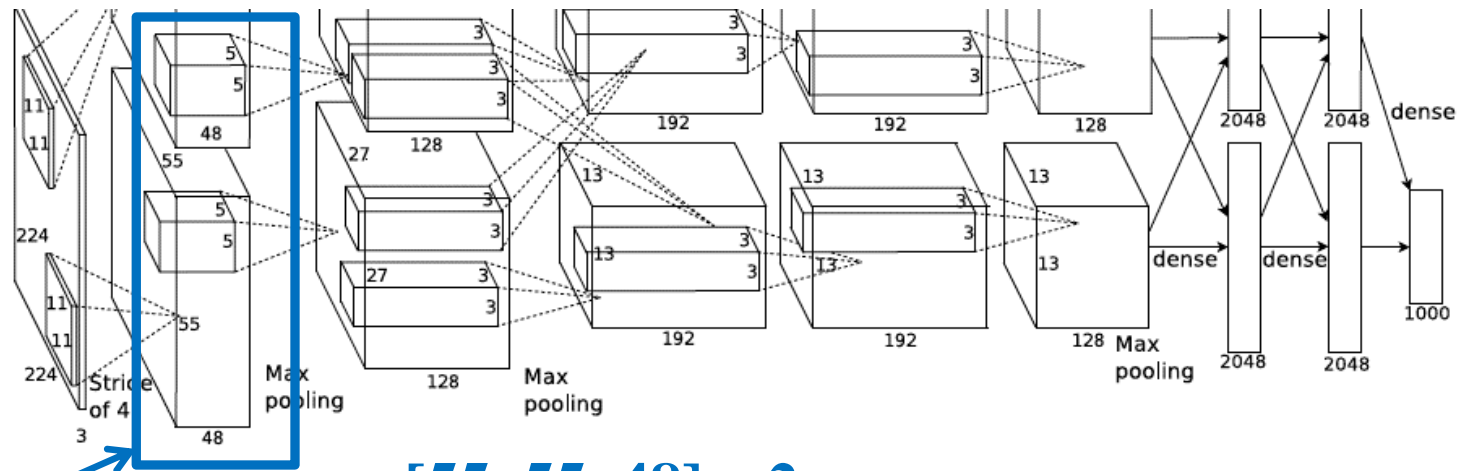
[6x6x256] **MAX POOL3**: 3x3 filters at stride 2

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

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

AlexNet



Full (simplified) AlexNet architecture: **[55x55x48] x 2**

[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

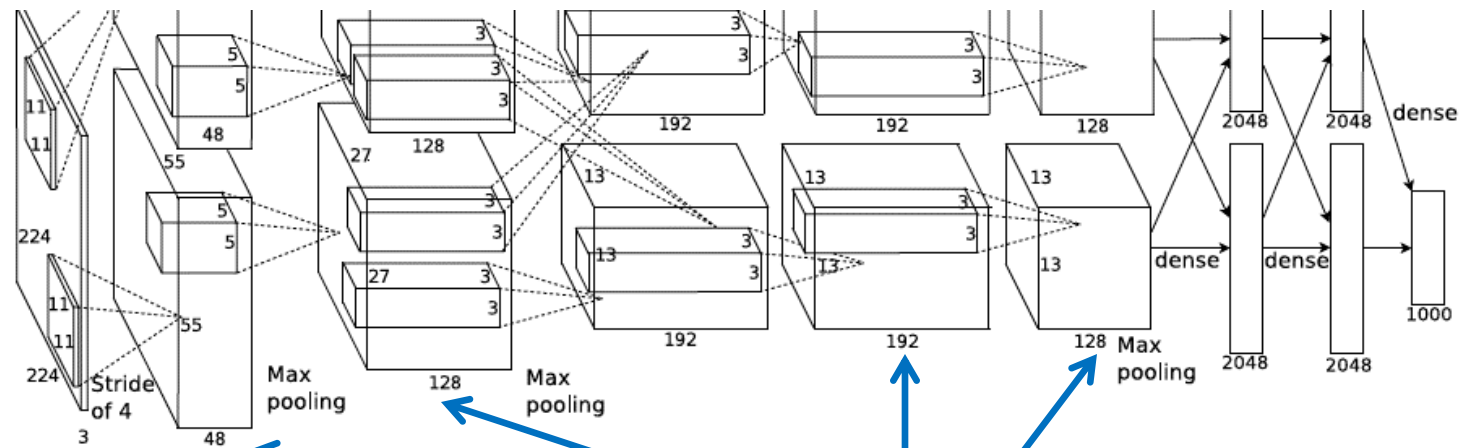
[4096] **FC6**: 4096 neurons

[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.

AlexNet



Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0

[27x27x96] **MAX POOL1**: 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2

[13x13x256] **MAX POOL2**: 3x3 filters at stride 2

[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1

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

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

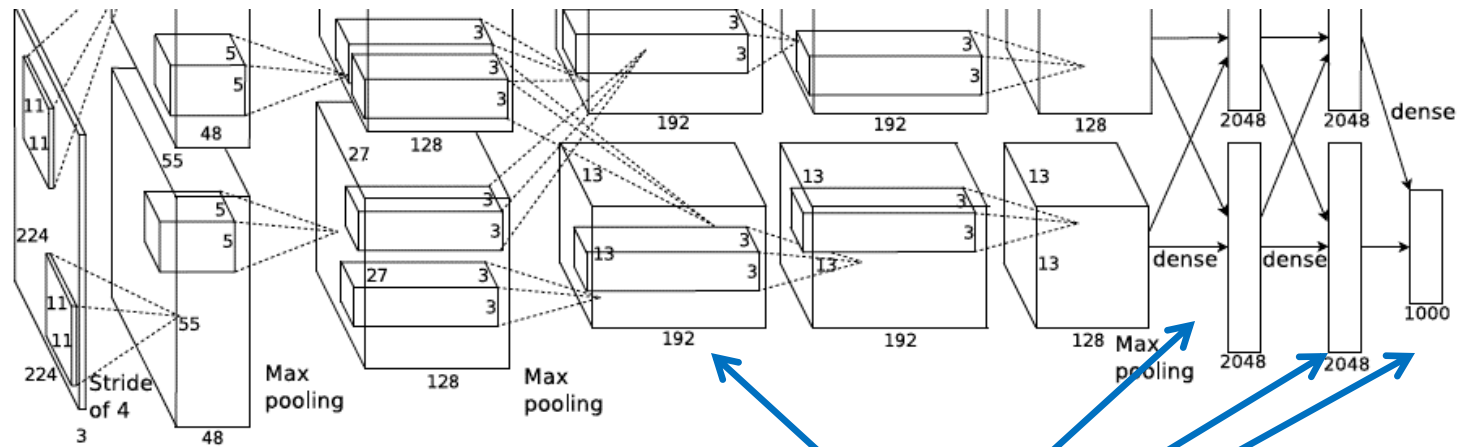
[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

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

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

AlexNet



Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0

[27x27x96] **MAX POOL1**: 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2

[13x13x256] **MAX POOL2**: 3x3 filters at stride 2

[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1

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

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

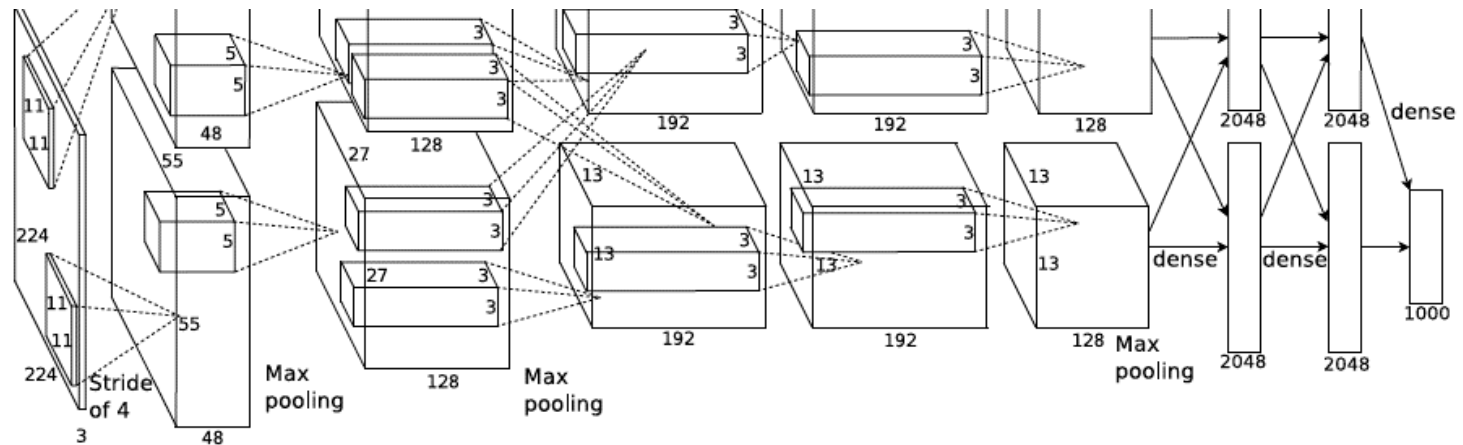
[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

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

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

AlexNet



Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0

[27x27x96] **MAX POOL1**: 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2

[13x13x256] **MAX POOL2**: 3x3 filters at stride 2

[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1

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

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

[4096] **FC6**: 4096 neurons

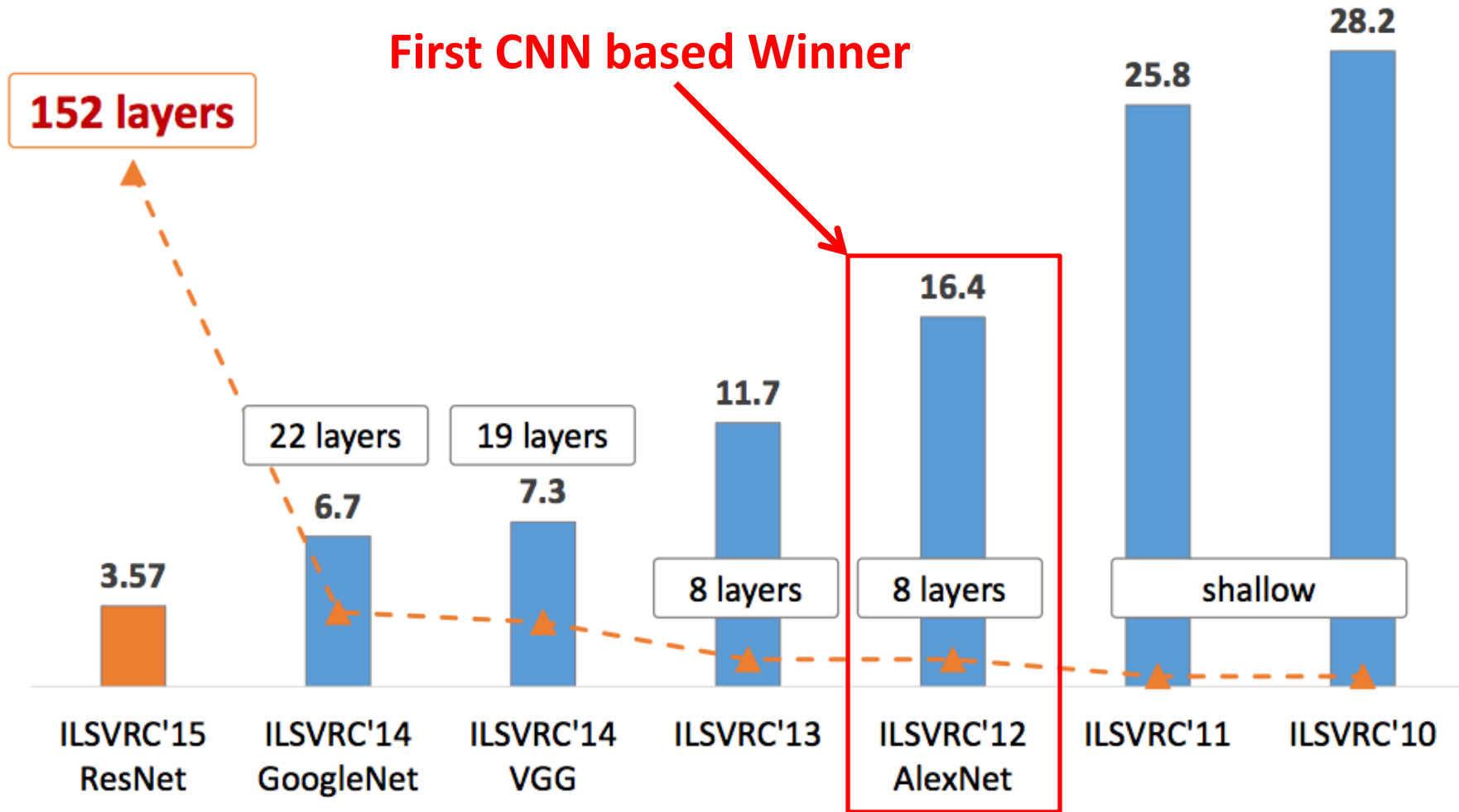
[4096] **FC7**: 4096 neurons

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

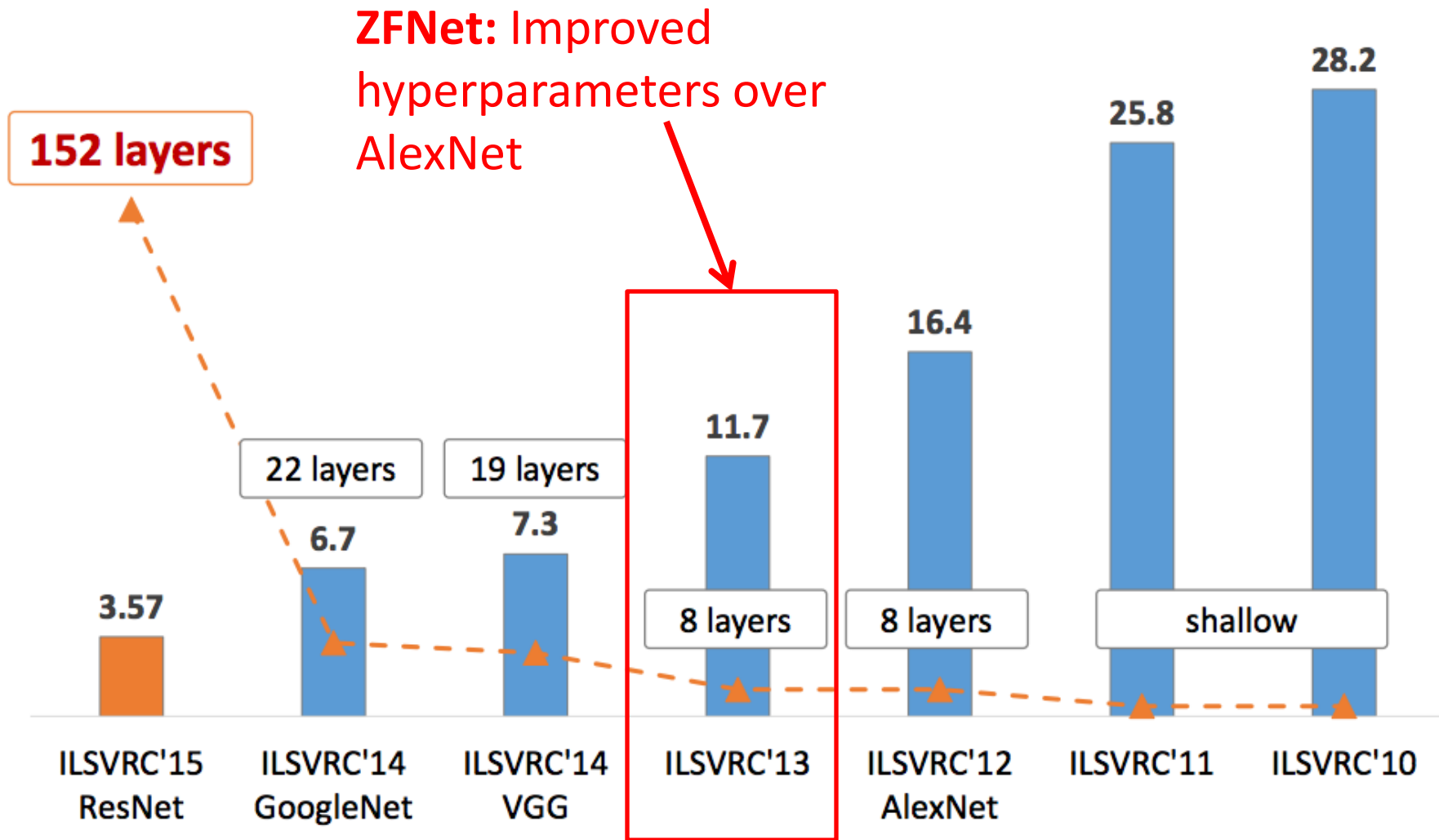
Details/Retrospectives:

- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- batch size 128
- SGD Momentum 0.9
- Learning rate 0.01, reduced manually when val accuracy saturates

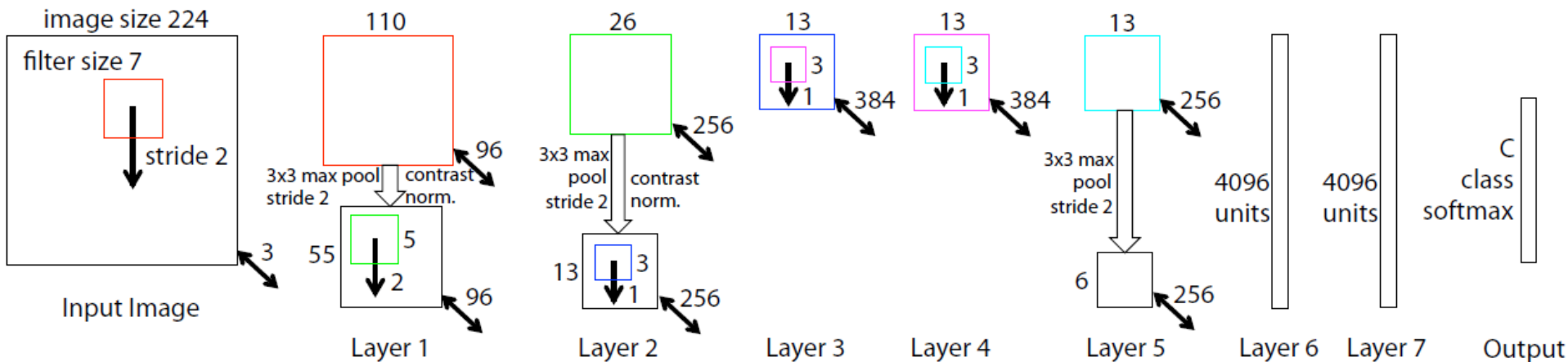
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ZFNet



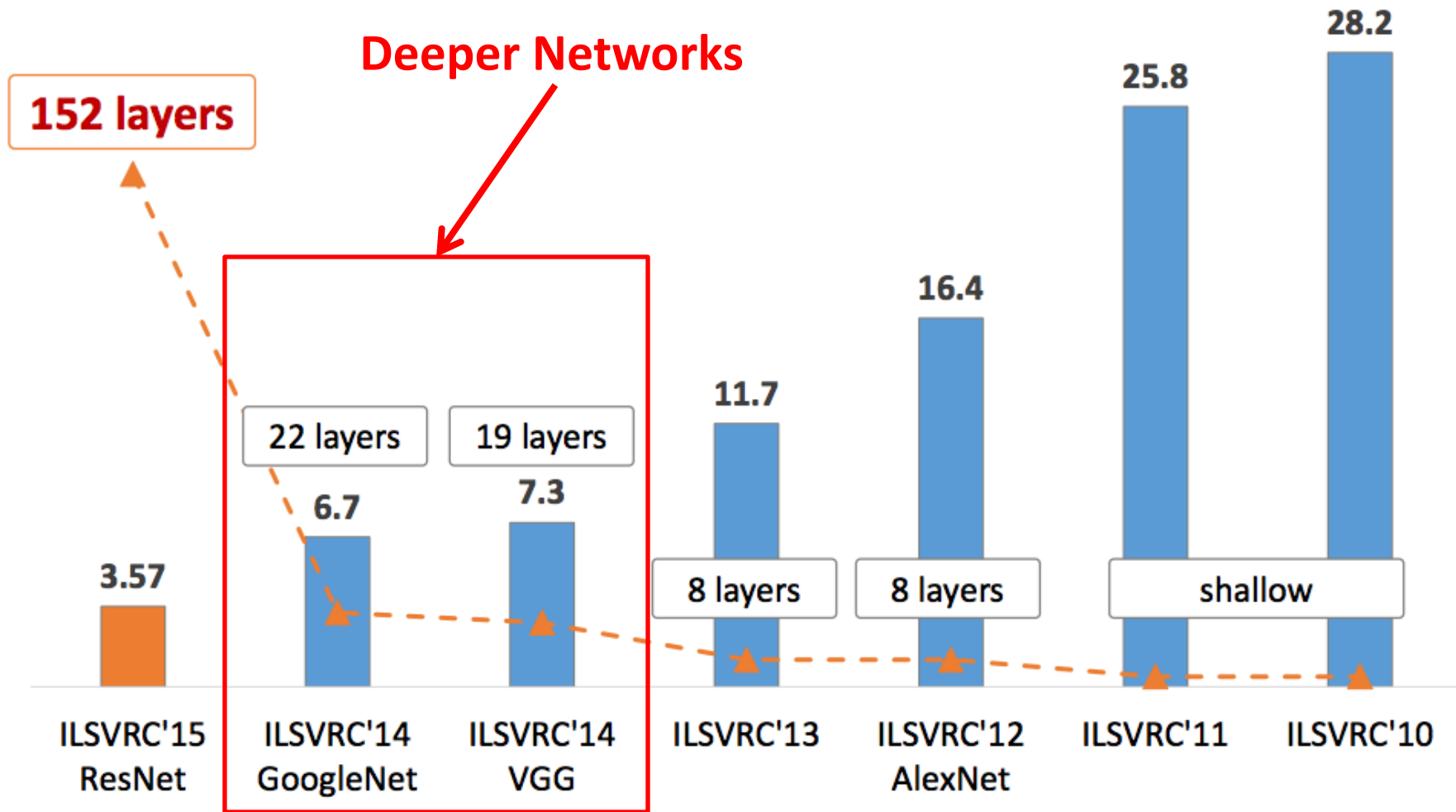
AlexNet but:

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

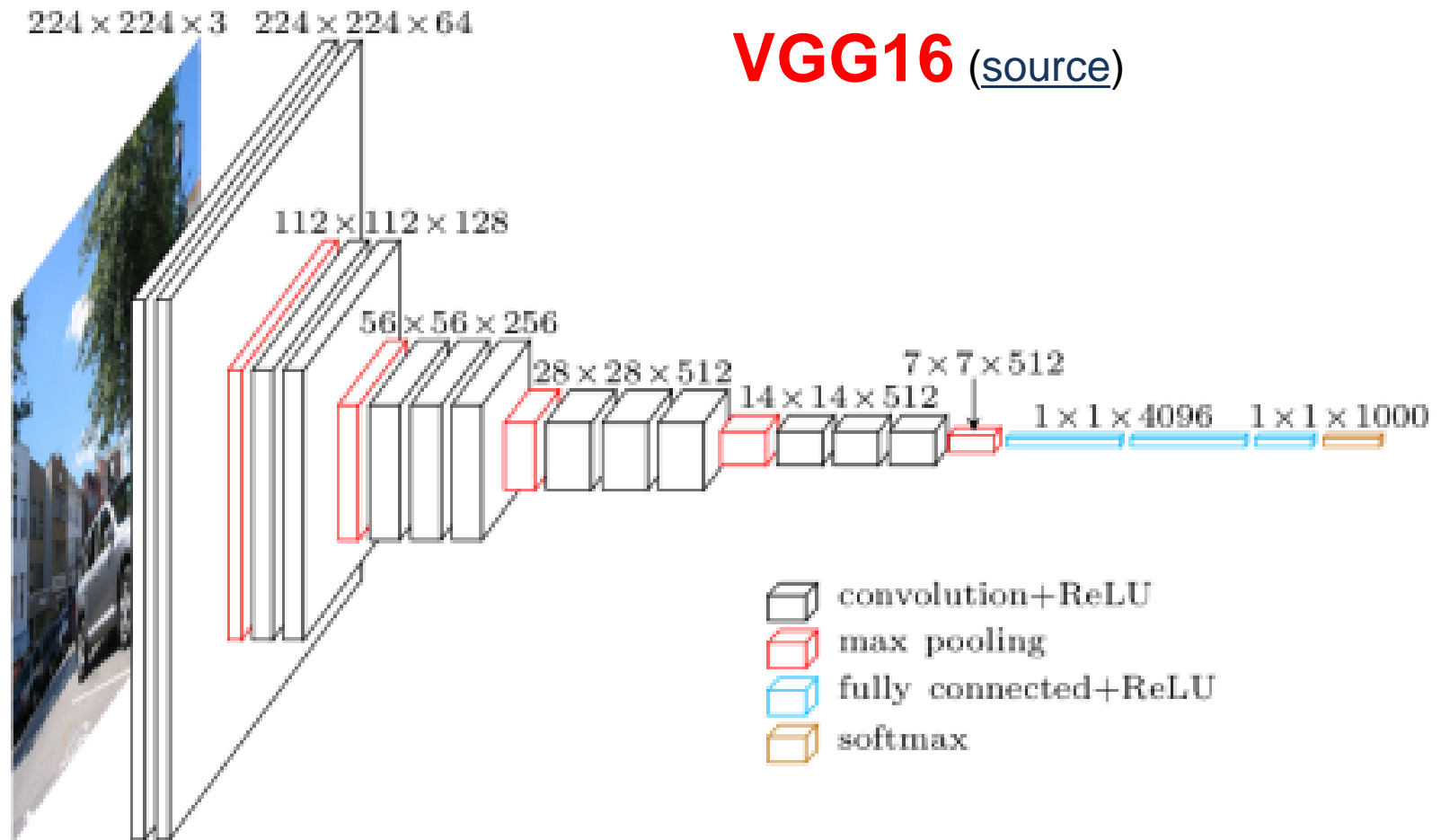
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 16.4% -> 11.7%

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



VGGNet



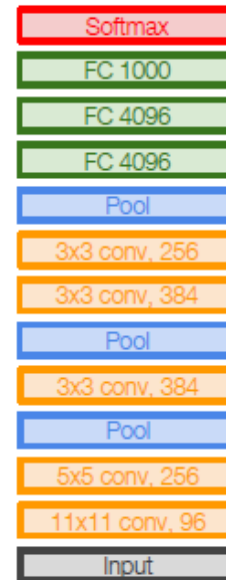
VGGNet

Small filters, Deeper networks

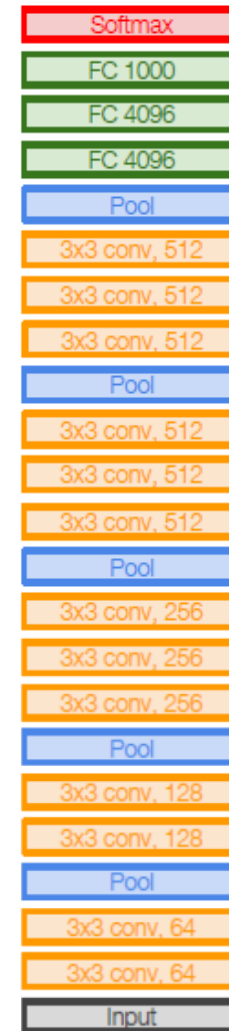
8 layers (AlexNet)

-> 16 - 19 layers (VGGNet)

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



AlexNet



VGG16



VGG19

VGGNet

Small filters, Deeper networks

8 layers (AlexNet)

-> 16 - 19 layers (VGGNet)

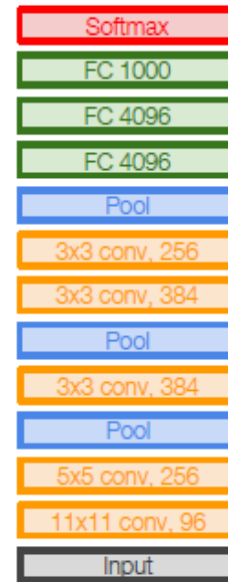
Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

ImageNet top 5 error:

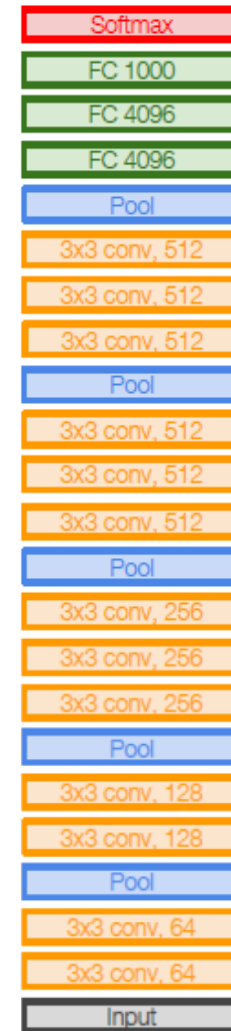
11.4% (ZFNet, 2013)

->

7.3% (VGGNet, 2014)



AlexNet



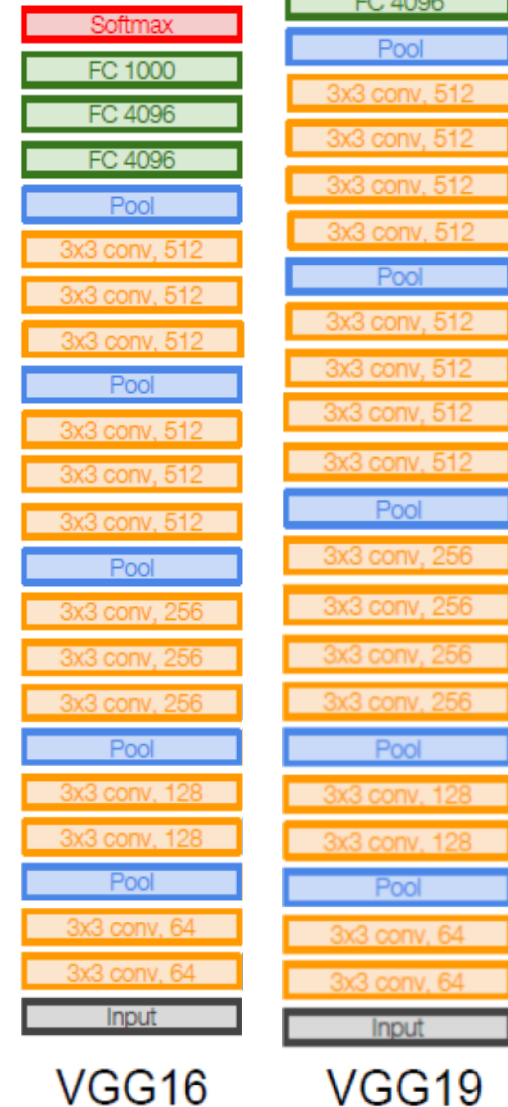
VGG16



VGG19

VGGNet

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



VGG16

VGG19

VGGNet

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

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



VGGNet

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

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

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



VGG16

VGG19

VGGNet

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

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

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

[7x7]

But deeper, more non-linearities

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



VGG16

VGG19

VGGNet

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: $4096*1000 = 4,096,000$

VGGNet

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

VGGNet

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: $4096*1000 = 4,096,000$

Note:

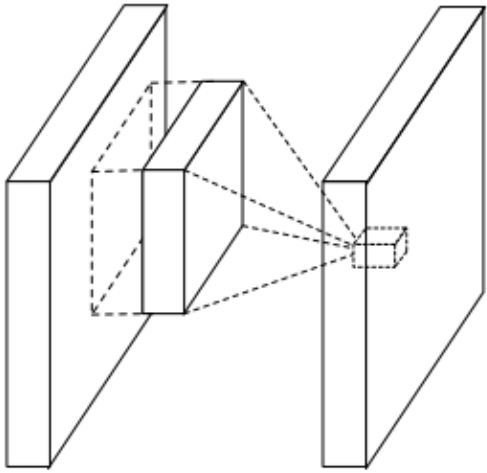
**Most
memory is in
early CONV**

**Most params
are in late
FC**

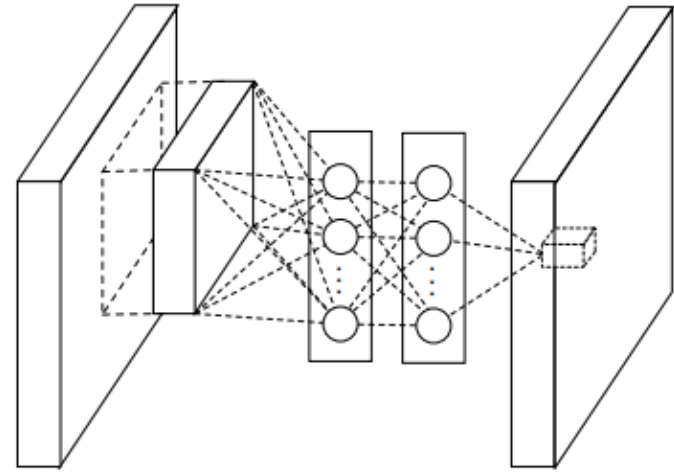
TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)

TOTAL params: 138M parameters

Network in Network (NiN)

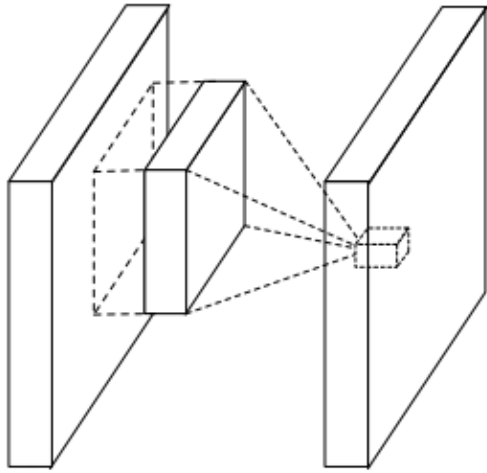


(a) Linear convolution layer

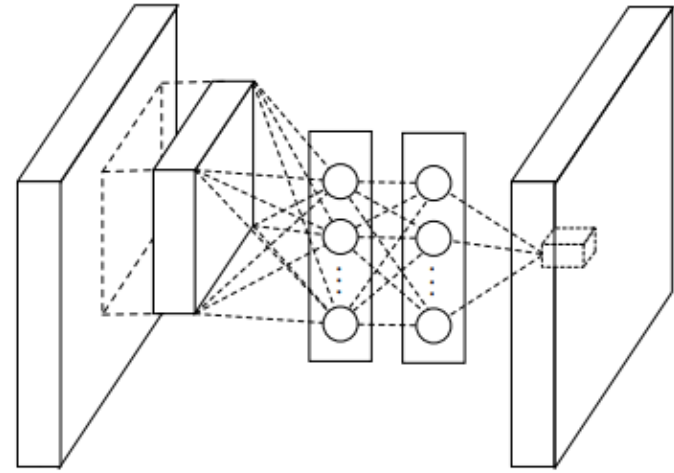


(b) Mlpconv layer

Network in Network (NiN)



(a) Linear convolution layer

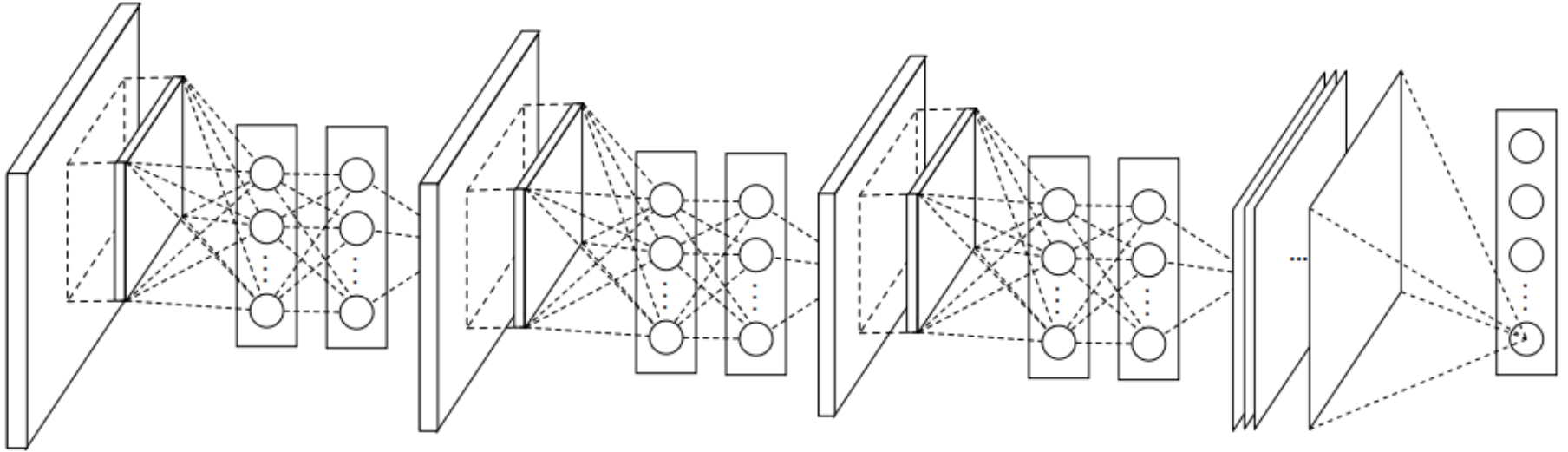


(b) Mlpconv layer

- Mlpconv layer with “micronetwork” within each conv layer to compute more abstract features for local patches
- Micronetwork uses multilayer perceptron (FC, i.e. 1x1 conv layers)

Network in Network (NiN)

The overall structure of NiN: stacking of three mlpconv layers and one global average pooling layer



Network in Network (NiN)

The overall structure of NiN: stacking of three mlpconv layers and one global average pooling layer

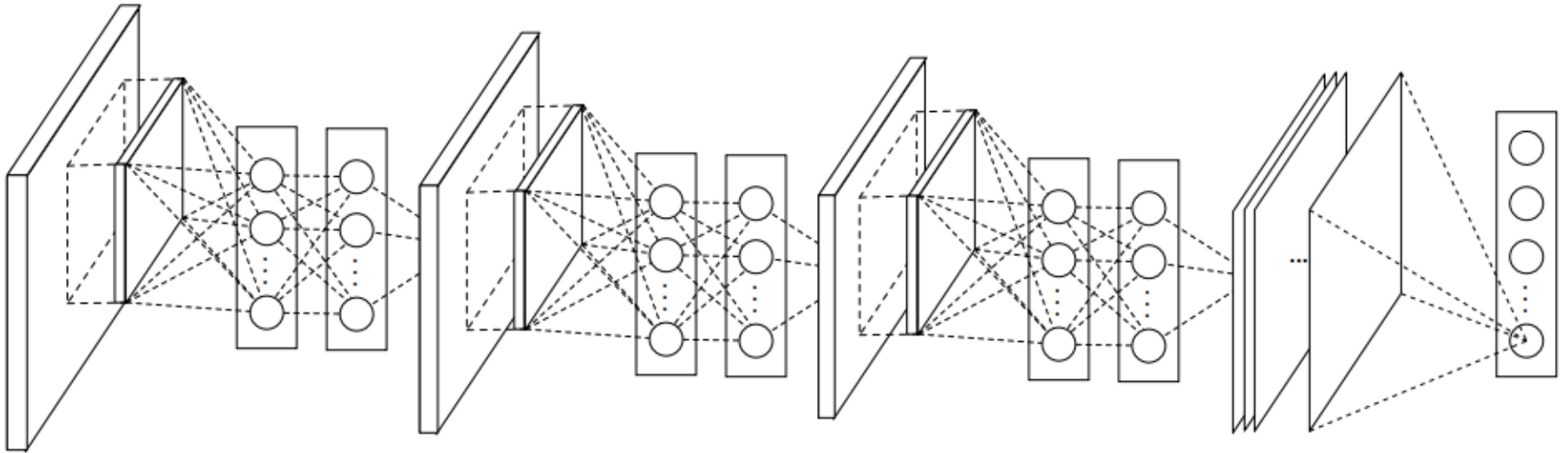
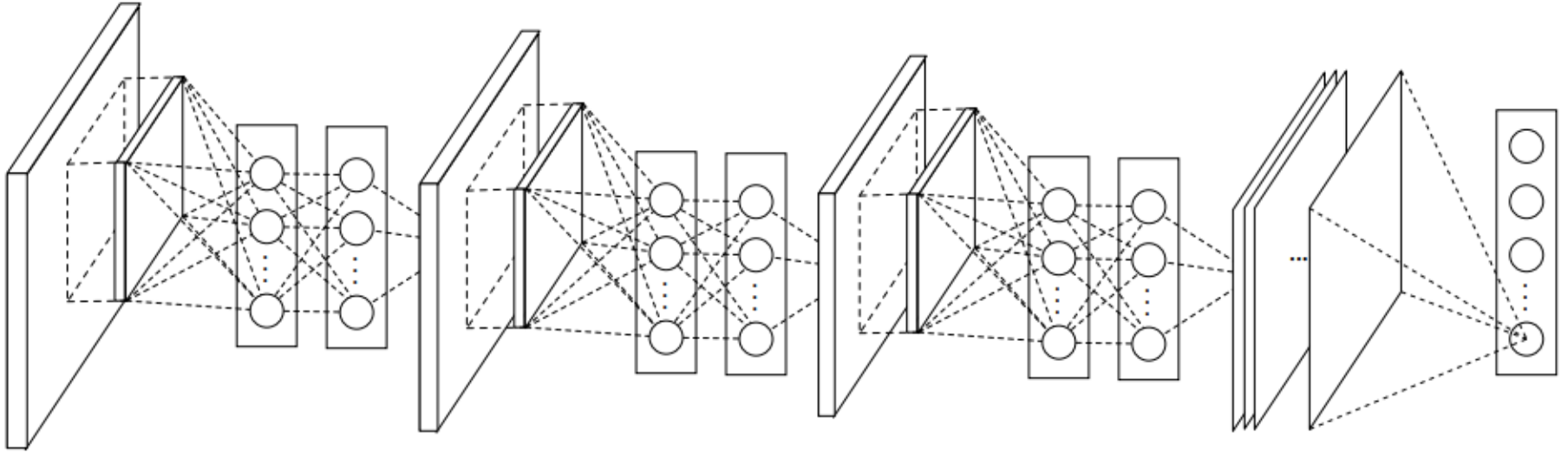


Table 1: Test set error rates for CIFAR-10 of various methods.

Method	Test Error
Stochastic Pooling [11]	15.13%
CNN + Spearmint [14]	14.98%
Conv. maxout + Dropout [8]	11.68%
NIN + Dropout	10.41%
CNN + Spearmint + Data Augmentation [14]	9.50%
Conv. maxout + Dropout + Data Augmentation [8]	9.38%
DropConnect + 12 networks + Data Augmentation [15]	9.32%
NIN + Dropout + Data Augmentation	8.81%

Network in Network (NiN)

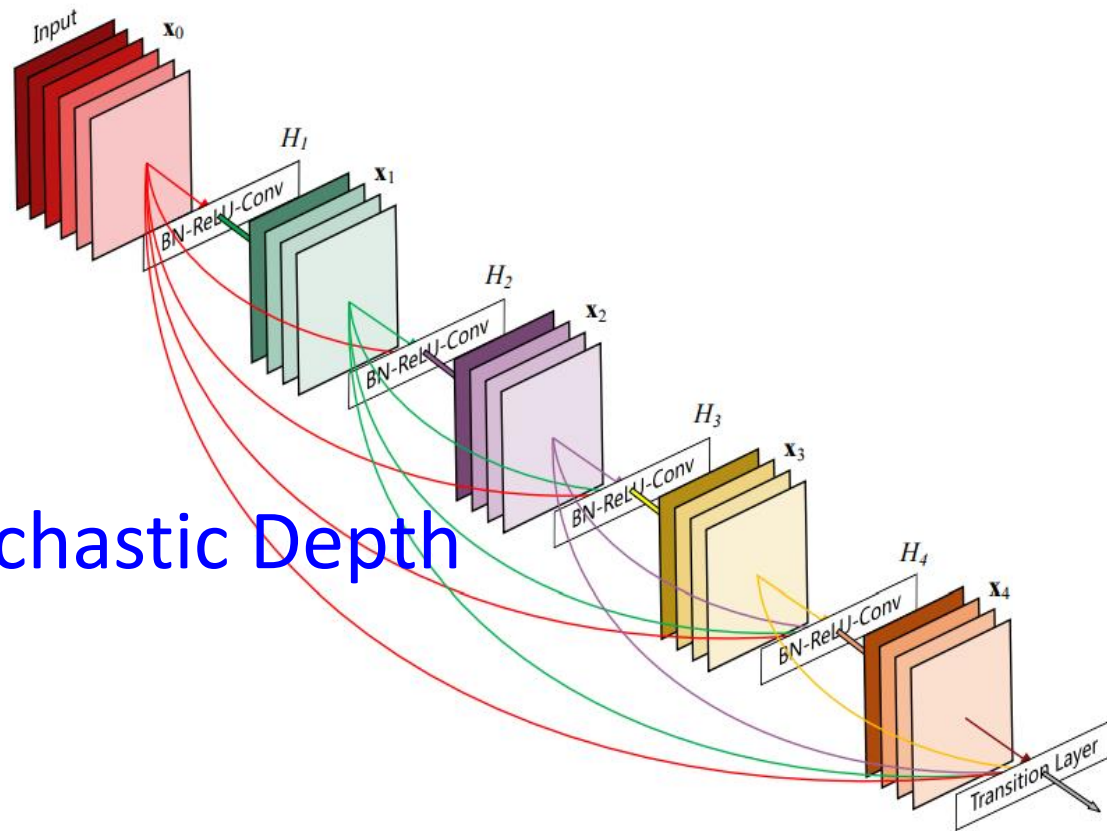
The overall structure of NiN: stacking of three mlpconv layers and one global average pooling layer



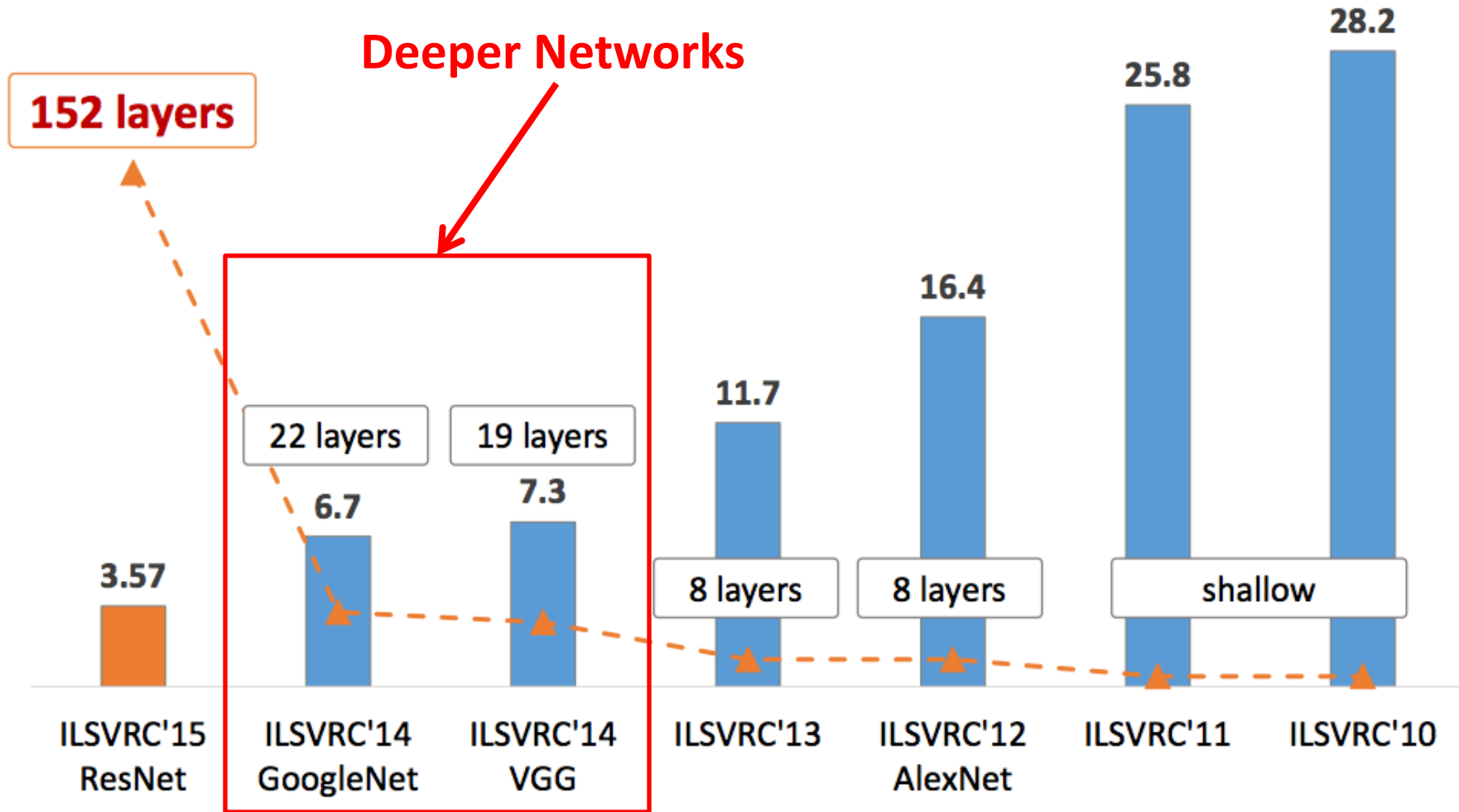
- Precursor to GoogLeNet and ResNet “bottleneck” layers
- Philosophical inspiration for GoogLeNet

CNN Architectures: DAG Models

- GoogLeNet
- ResNet
- Pre-act ResNet
- SENet
- Network with Stochastic Depth
- DenseNet
- ResNetXt



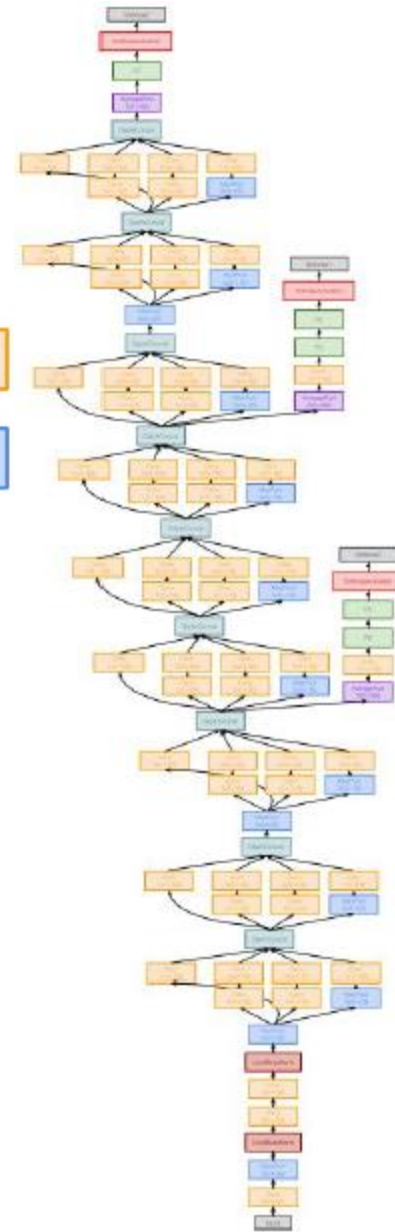
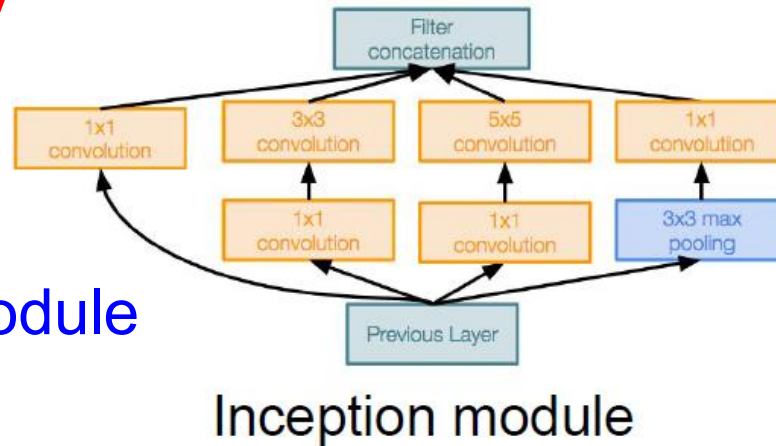
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



GoogLeNet

Deeper networks, with
computational efficiency

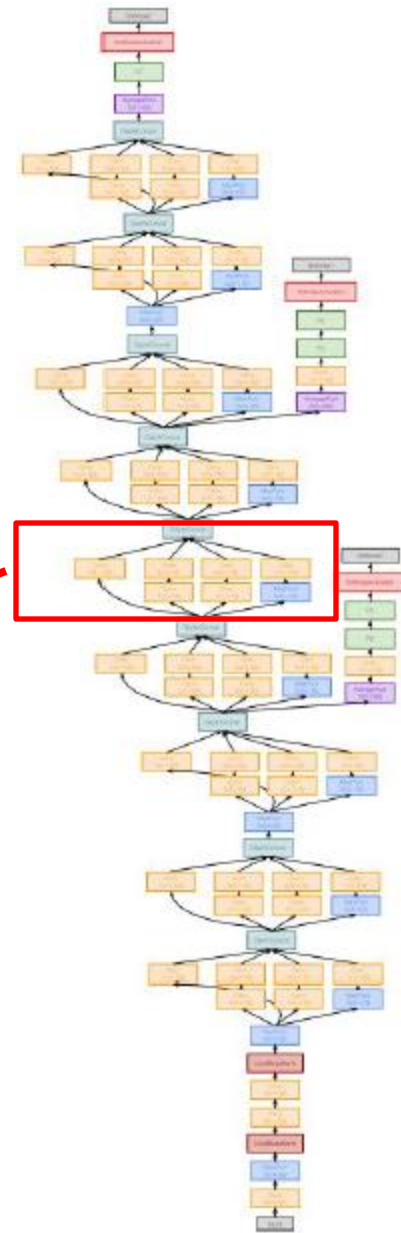
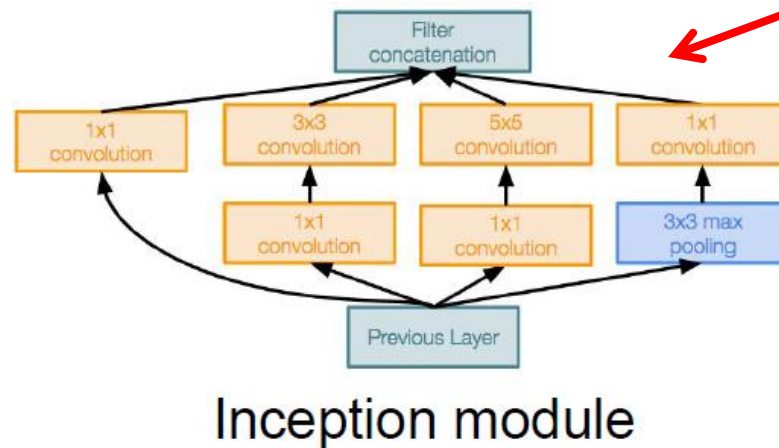
- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters!
12x less than AlexNet
- Imagenet classification winner
(6.7% top 5 error)



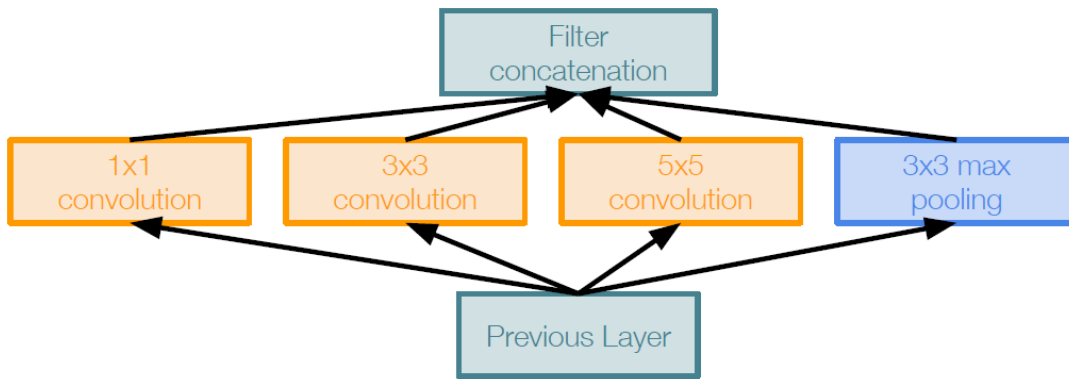
GoogLeNet

“Inception module”:

design a good local network topology and then stack these modules on top of each other



GoogLeNet



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 depth-wise

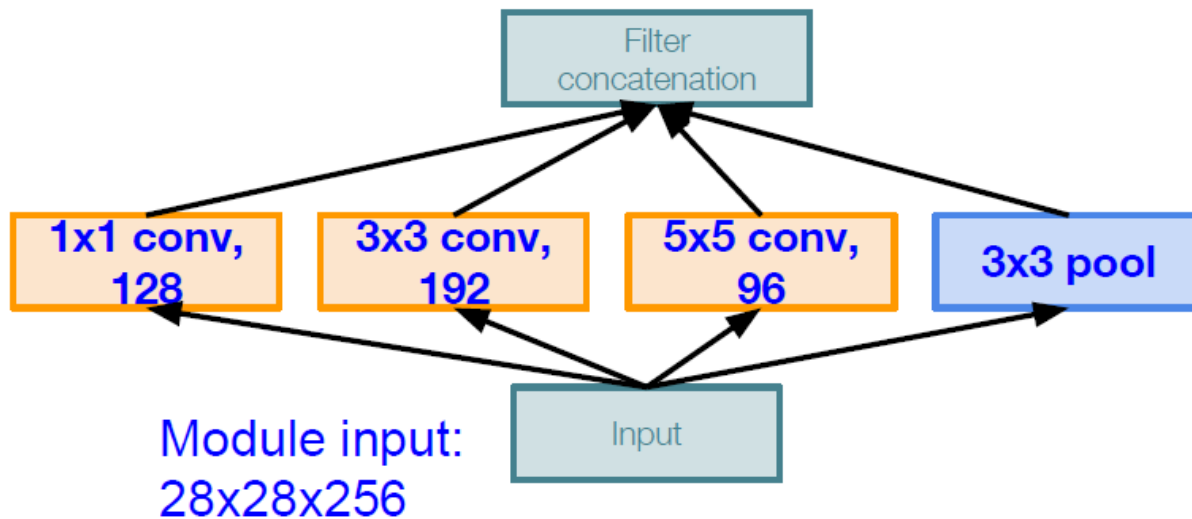
Problem:
Computational Complexity

GoogLeNet

Problem:
Computational Complexity

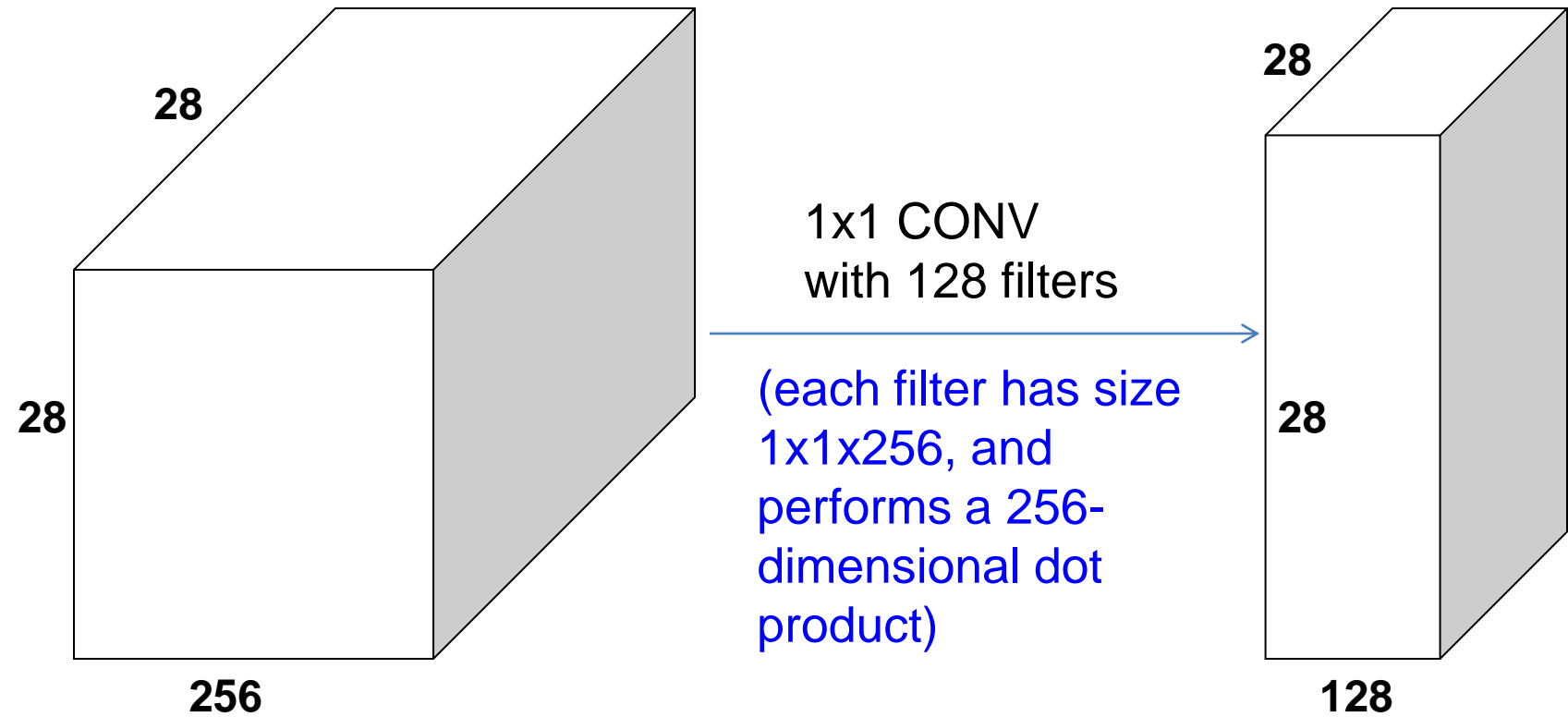
Q1: What is the output size of the
1x1 conv, with 128 filters?

Example:



Naive Inception module

1 × 1 Convolutions

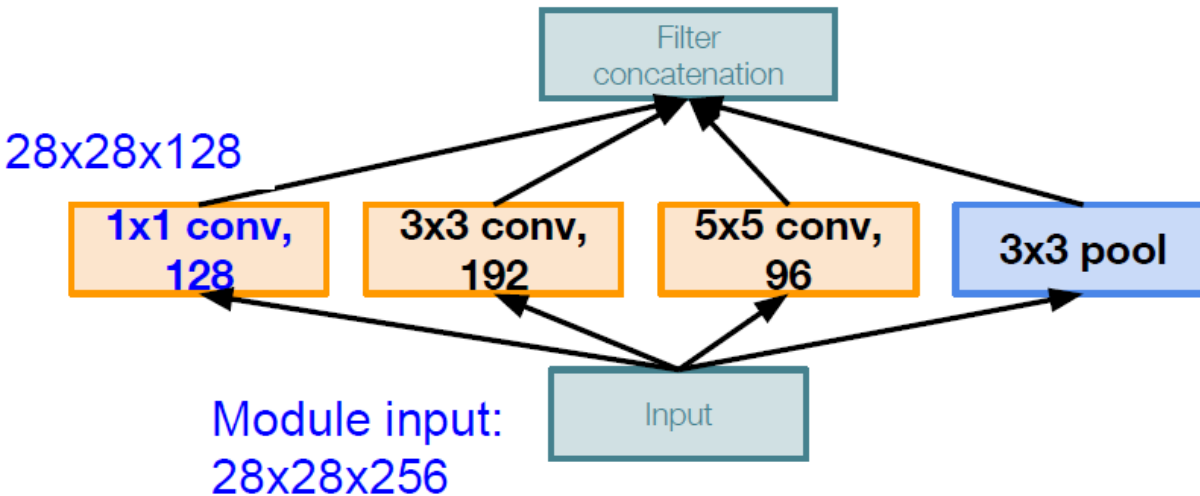


GoogLeNet

Problem:
Computational Complexity

Q1: What is the output size of the
1x1 conv, with 128 filters?

Example:



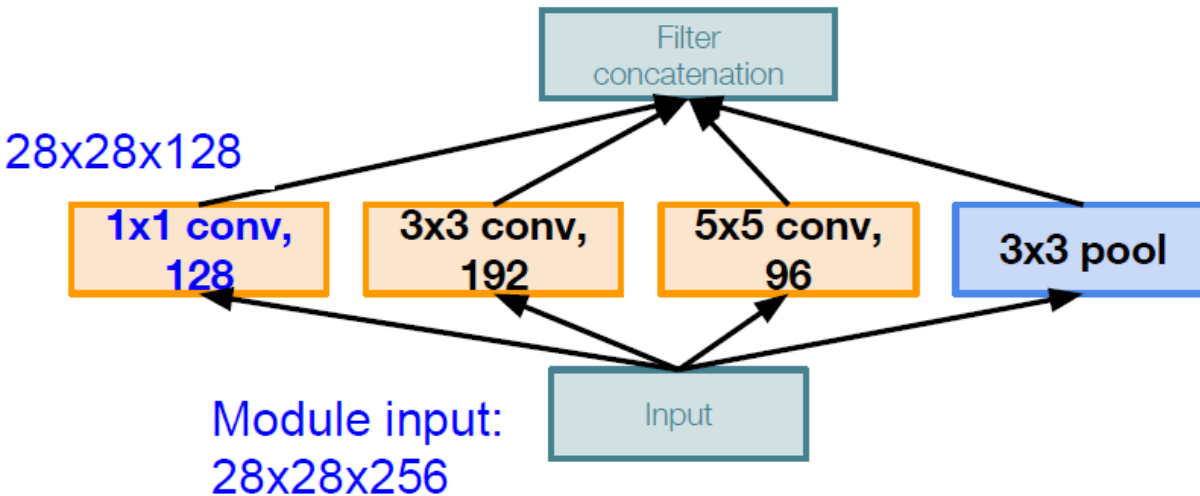
Naive Inception module

GoogLeNet

Problem:
Computational Complexity

Q2: What are the output sizes of
all different filter operations?

Example:



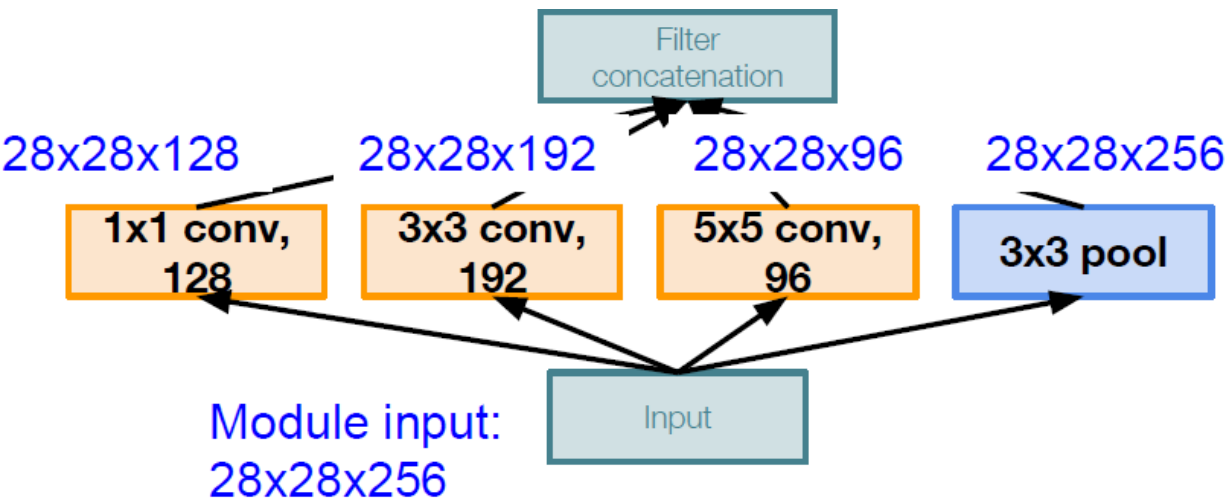
Naive Inception module

GoogLeNet

Problem:
Computational Complexity

Q2: What are the output sizes of
all different filter operations?

Example:



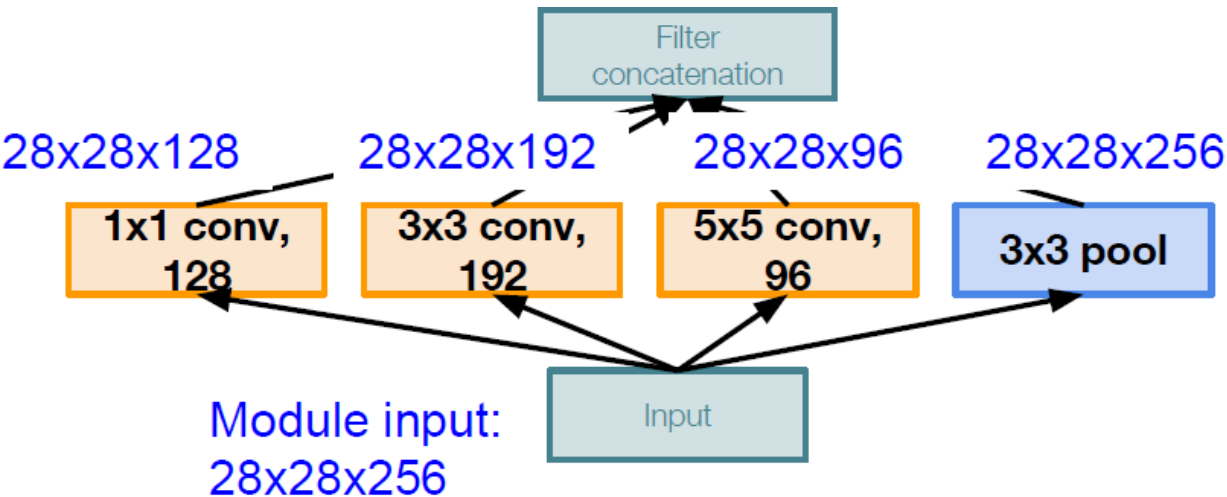
Naive Inception module

GoogLeNet

Problem:
Computational Complexity

Q3: What is output size after
filter concatenation?

Example:



Naive Inception module

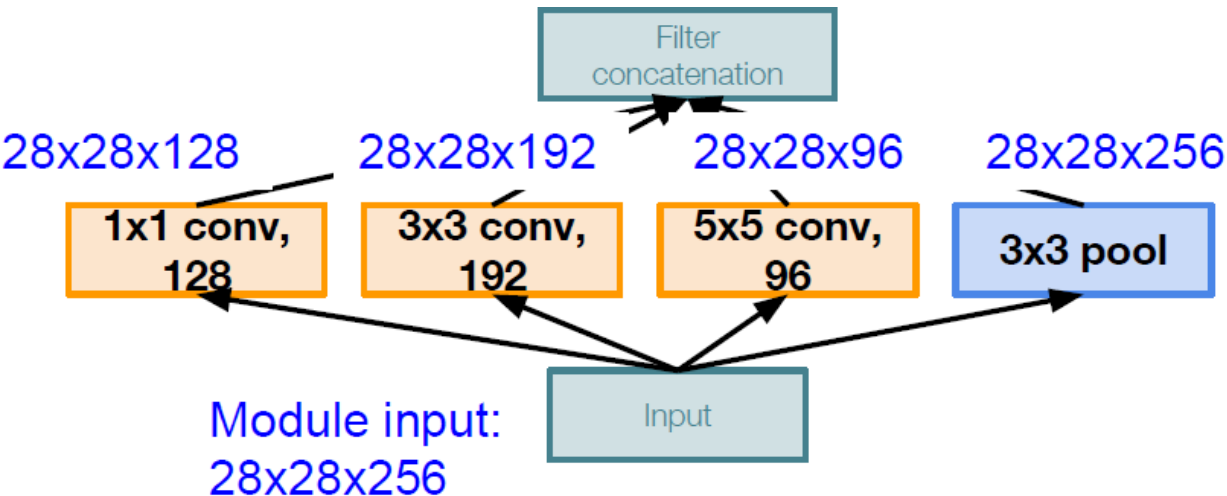
GoogLeNet

Problem:
Computational Complexity

Q3: What is output size after
filter concatenation?

Example:

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



Naive Inception module

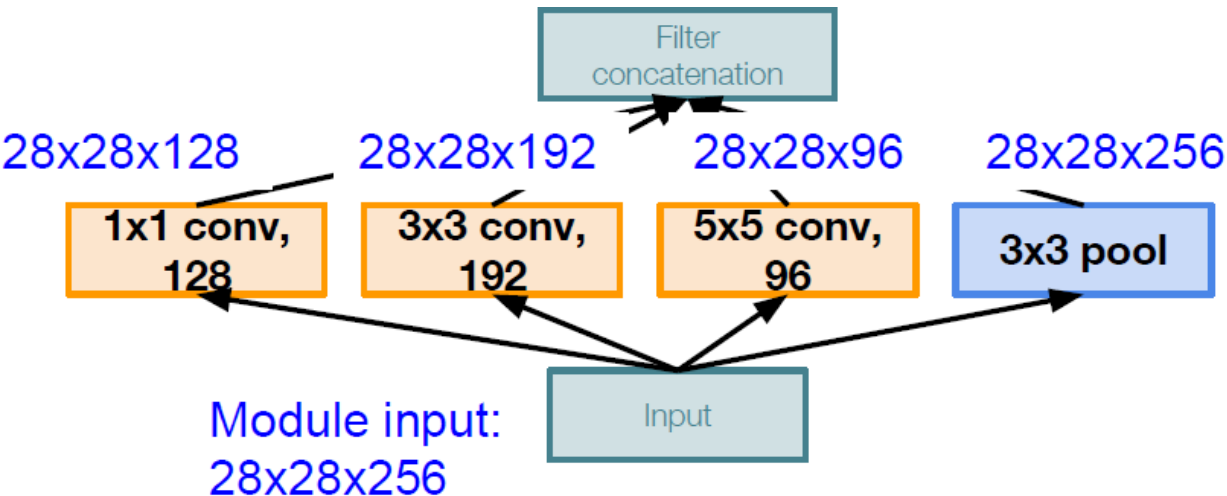
GoogLeNet

Problem:
Computational Complexity

Q3: What is output size after
filter concatenation?

Example:

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



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

**Very expensive
compute**

GoogLeNet

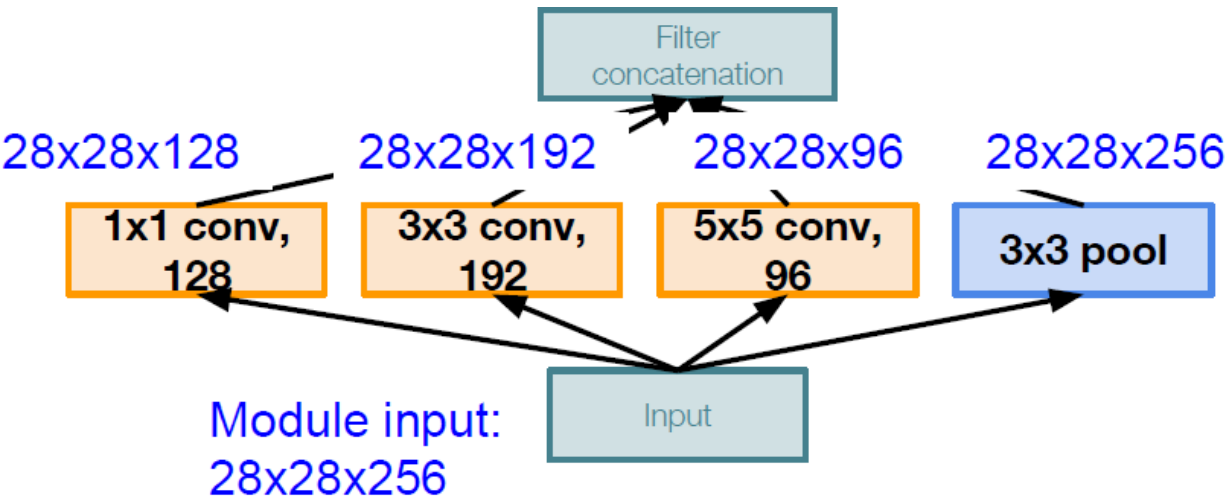
Problem:
Computational Complexity

Q3: What is output size after
filter concatenation?

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

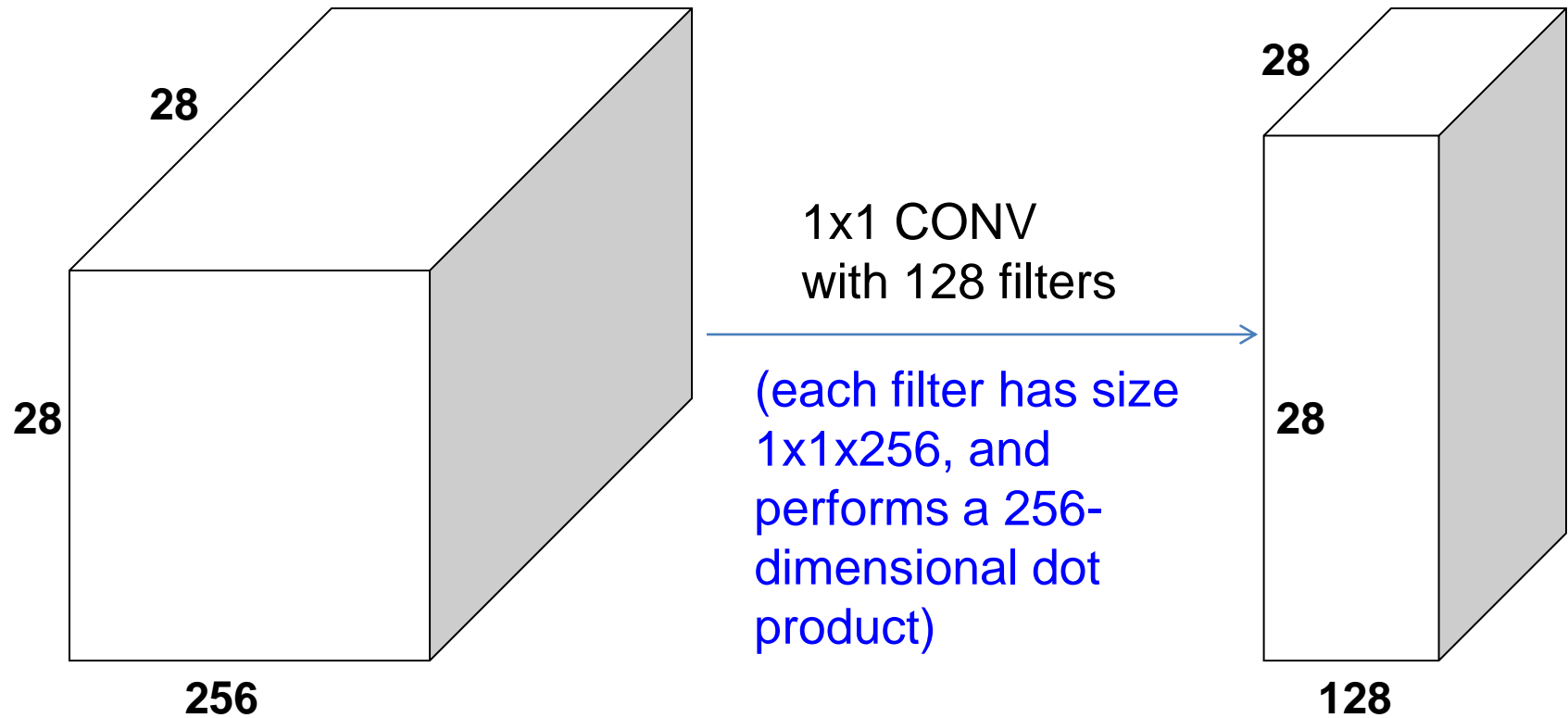
Example:

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



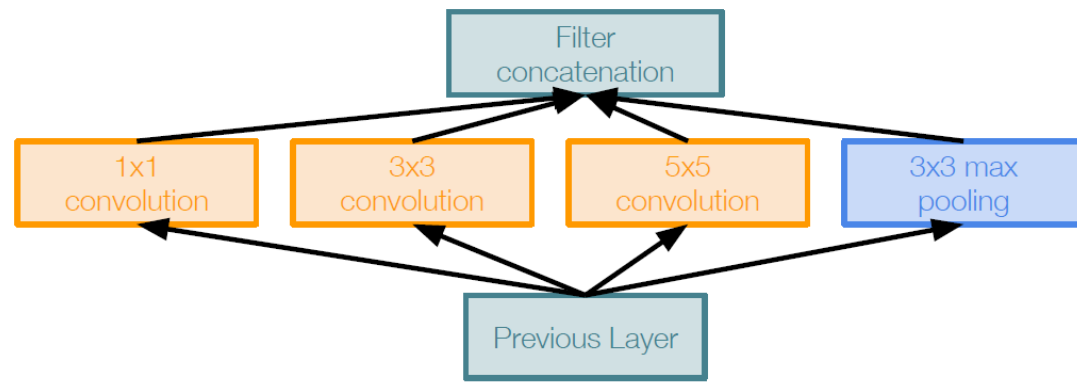
Naive Inception module

1 × 1 Convolutions



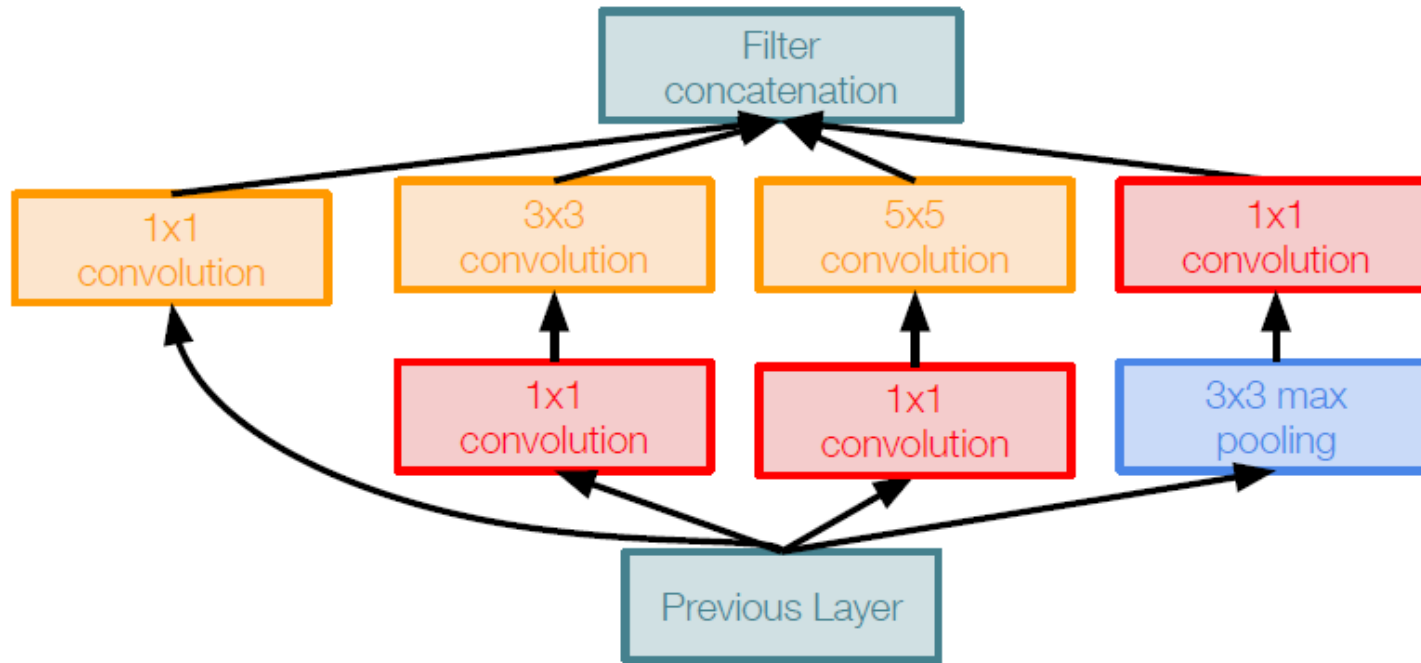
preserves spatial dimensions, reduces depth!
Projects depth to lower dimension (combination of feature maps)

GoogLeNet



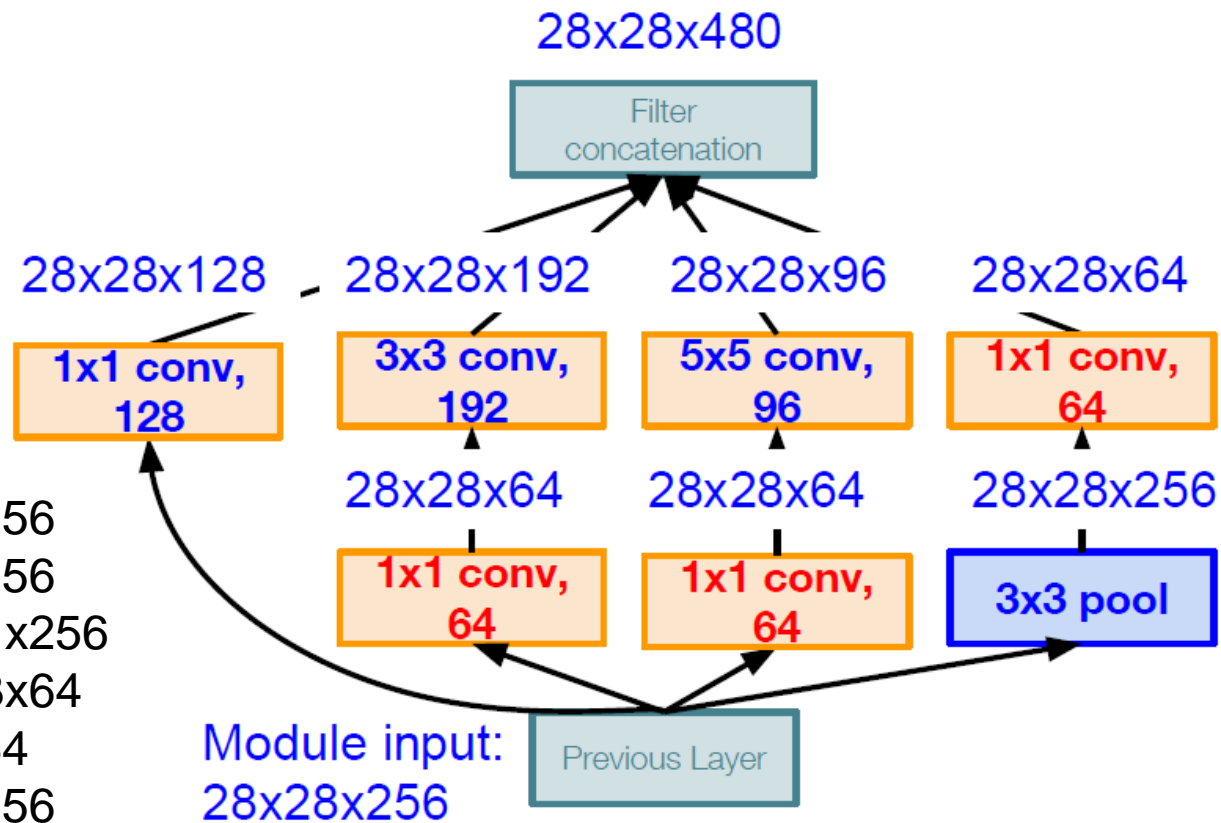
1x1 conv “bottleneck” layers

Naive Inception module



Inception module with dimension reduction

GoogLeNet



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] $28 \times 28 \times 64 \times 1 \times 1 \times 256$

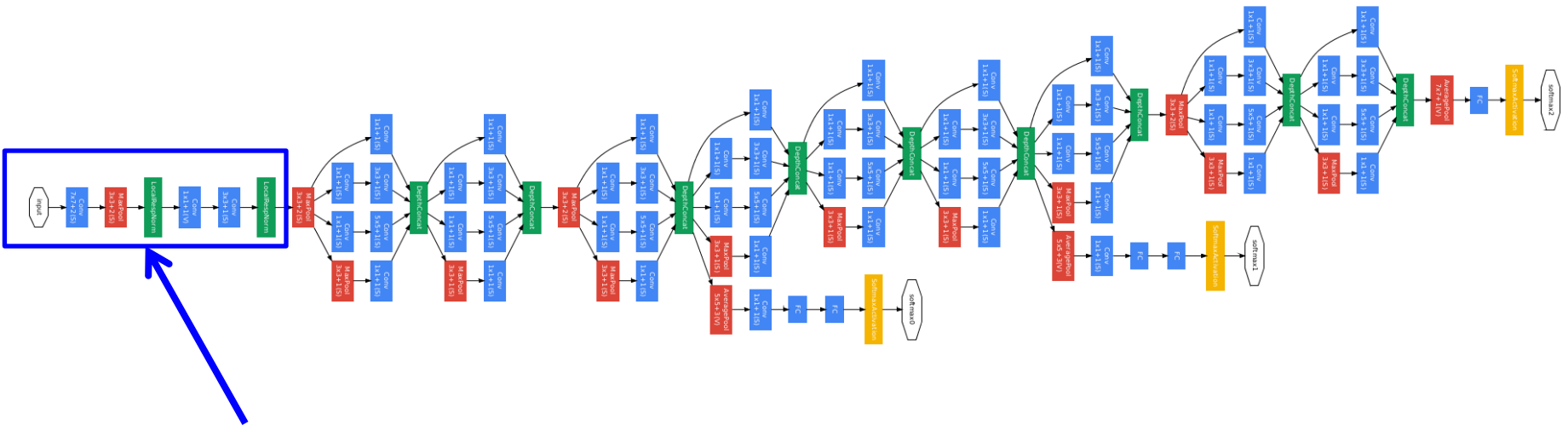
Total: 358M ops

Inception module with dimension reduction

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

GoogLeNet

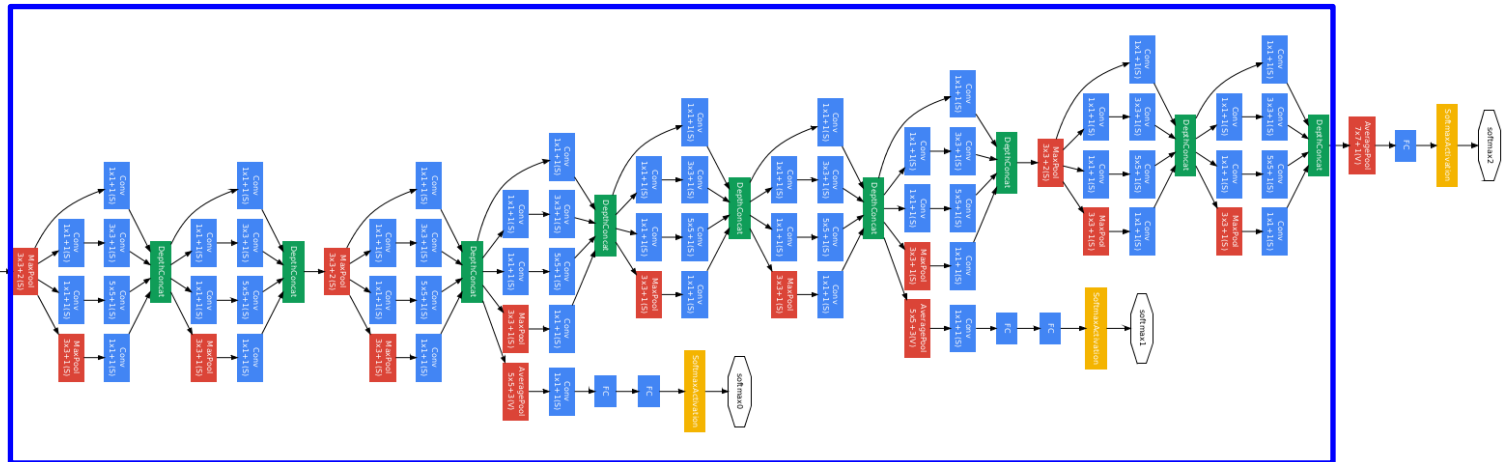
Full GoogLeNet Architecture



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

GoogLeNet

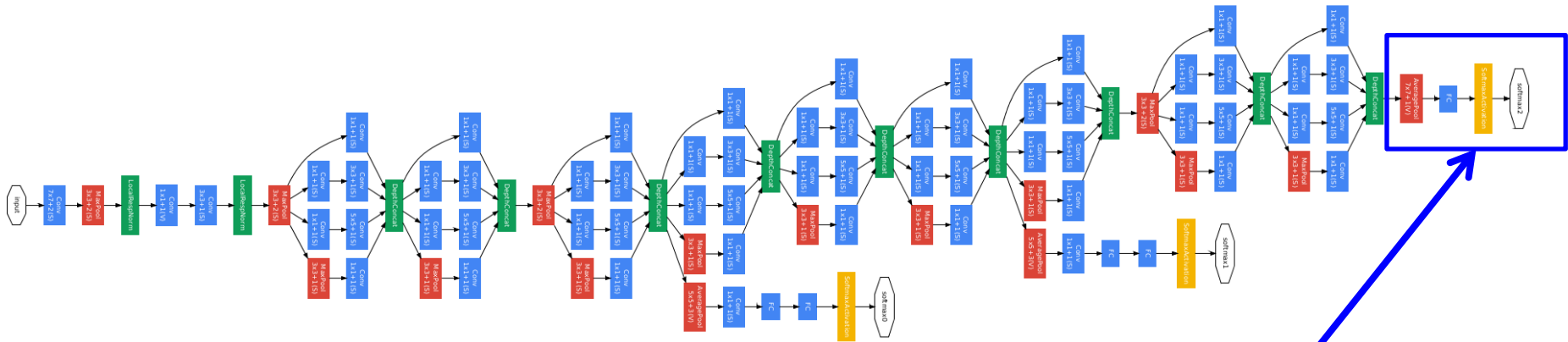
Full GoogLeNet Architecture



Stacked Inception Modules

GoogLeNet

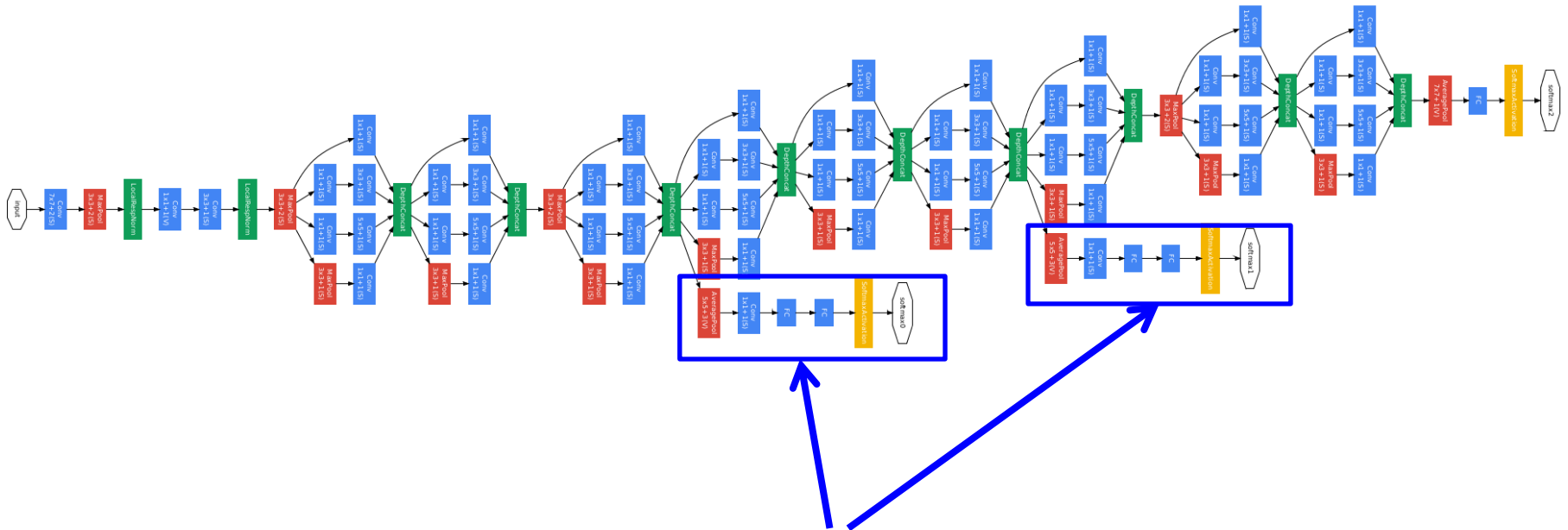
Full GoogLeNet Architecture



Classifier output
(removed expensive FC
layers!)

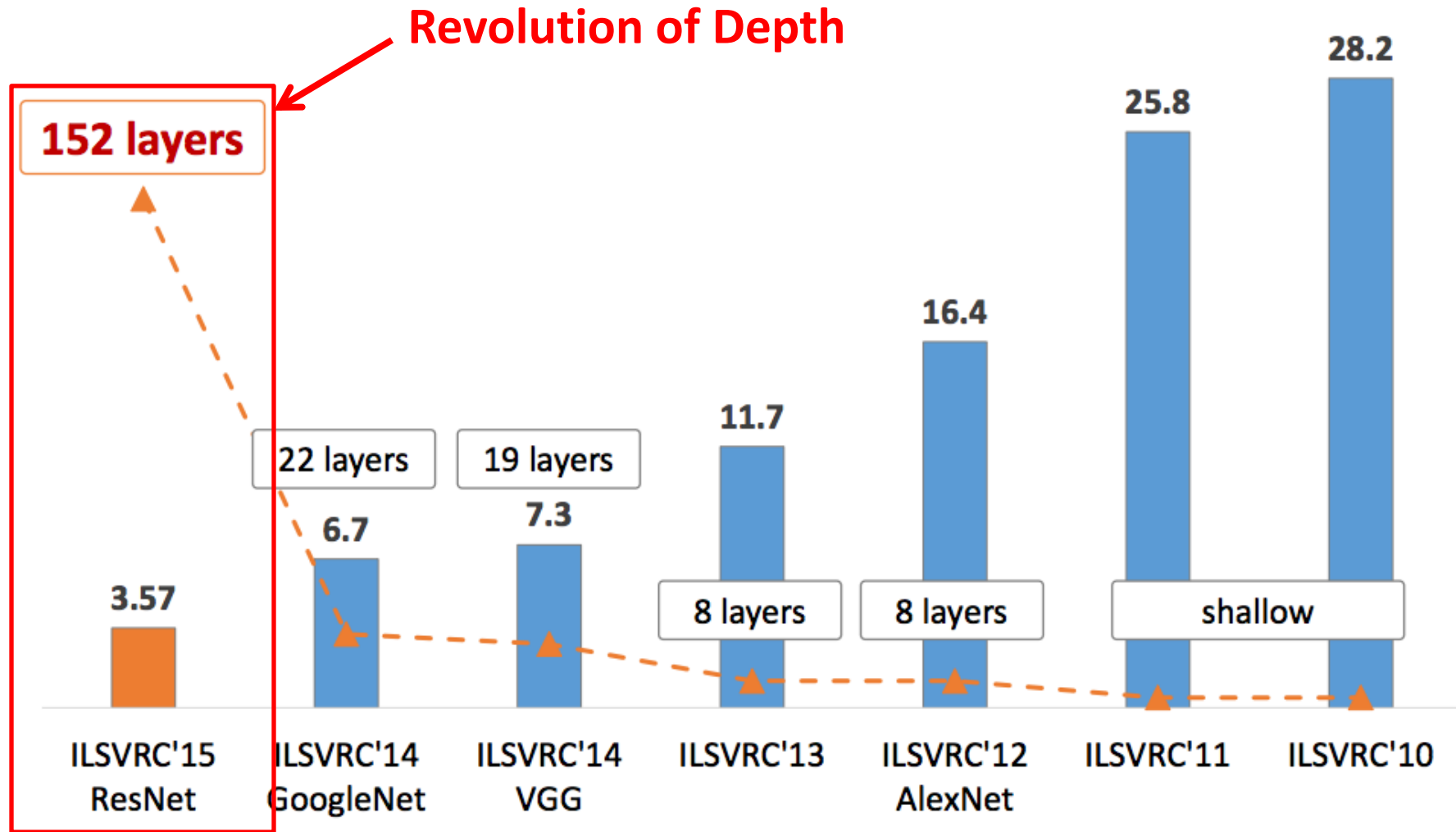
GoogLeNet

Full GoogLeNet Architecture



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

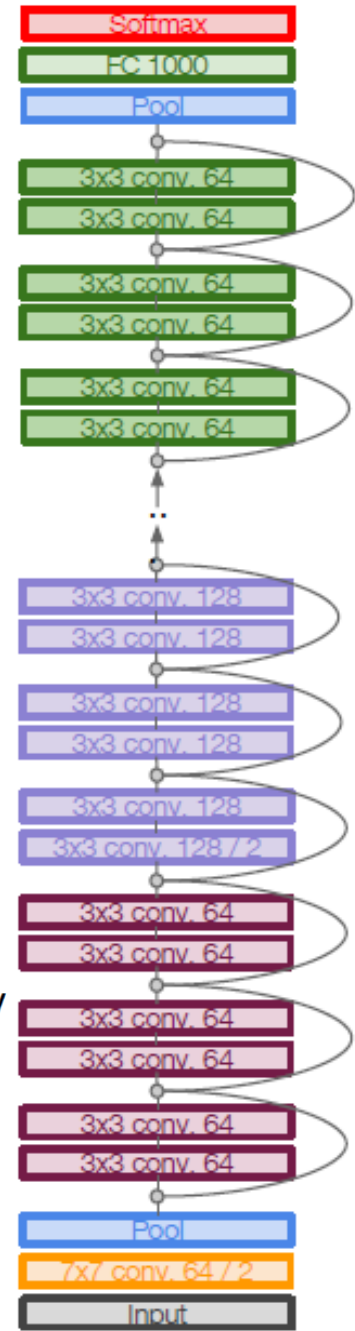
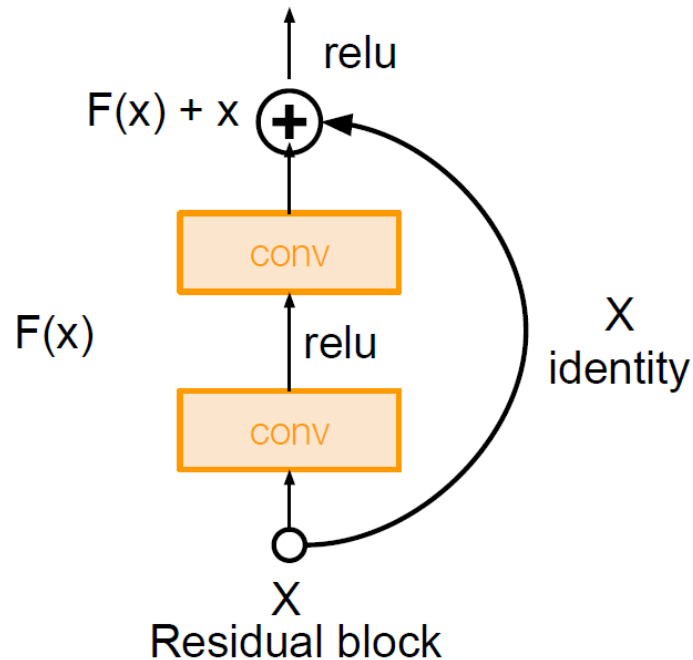
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ResNet

Very deep networks using residual connections

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



ResNet

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

ResNet

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



56-layer model performs worse on both training and test error
-> The deeper model performs worse, but it's not caused by overfitting!

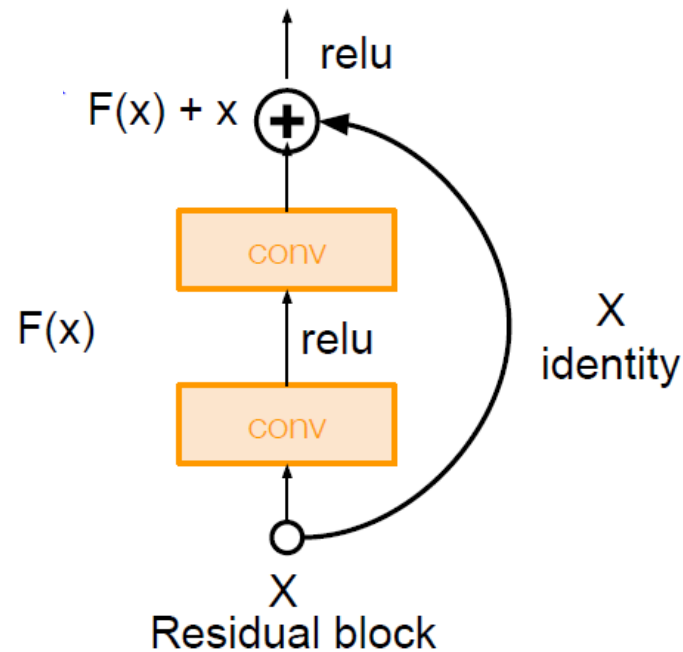
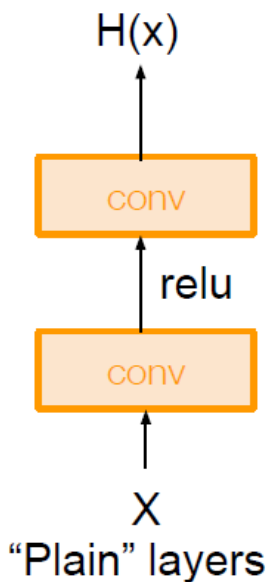
ResNet

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

The deeper model should be able to perform at least as well as the shallower model.

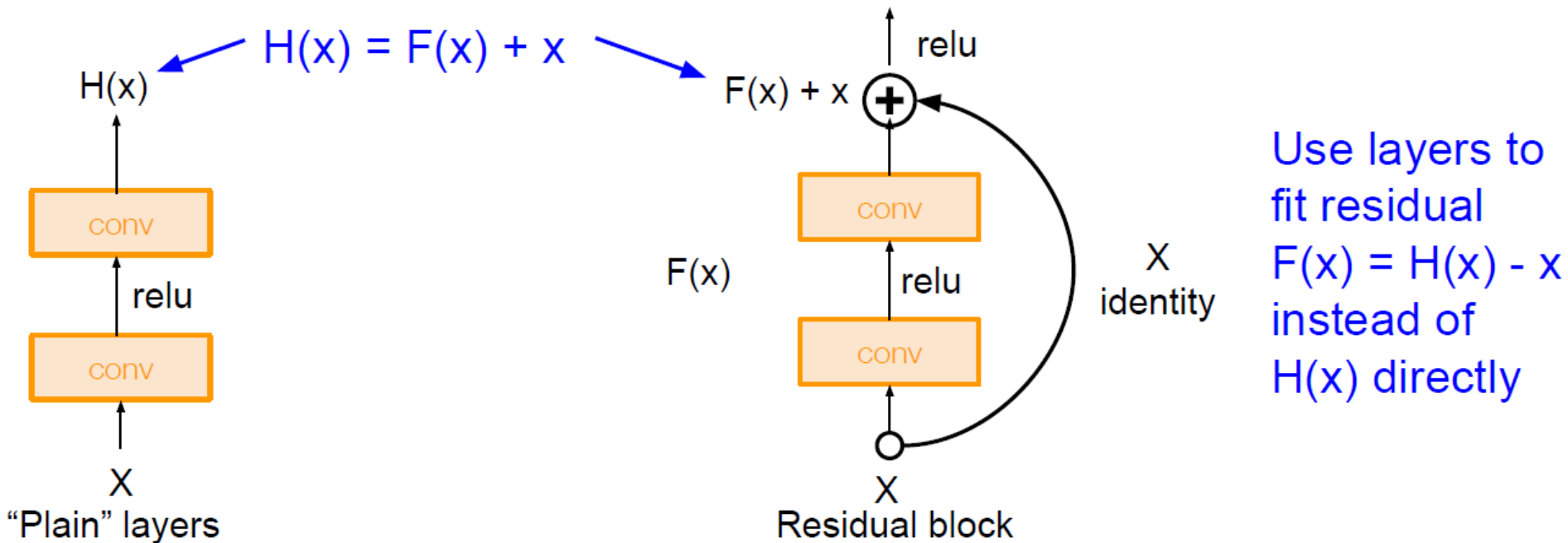
ResNet

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



ResNet

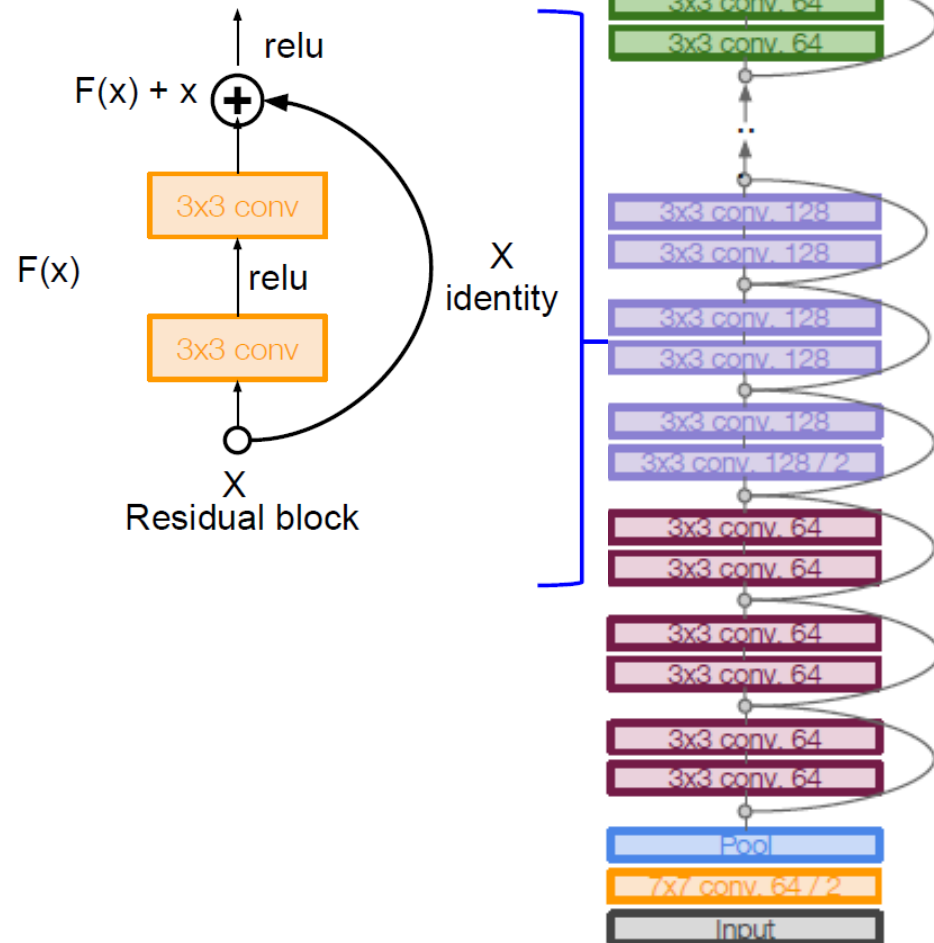
Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



ResNet

Full ResNet architecture:

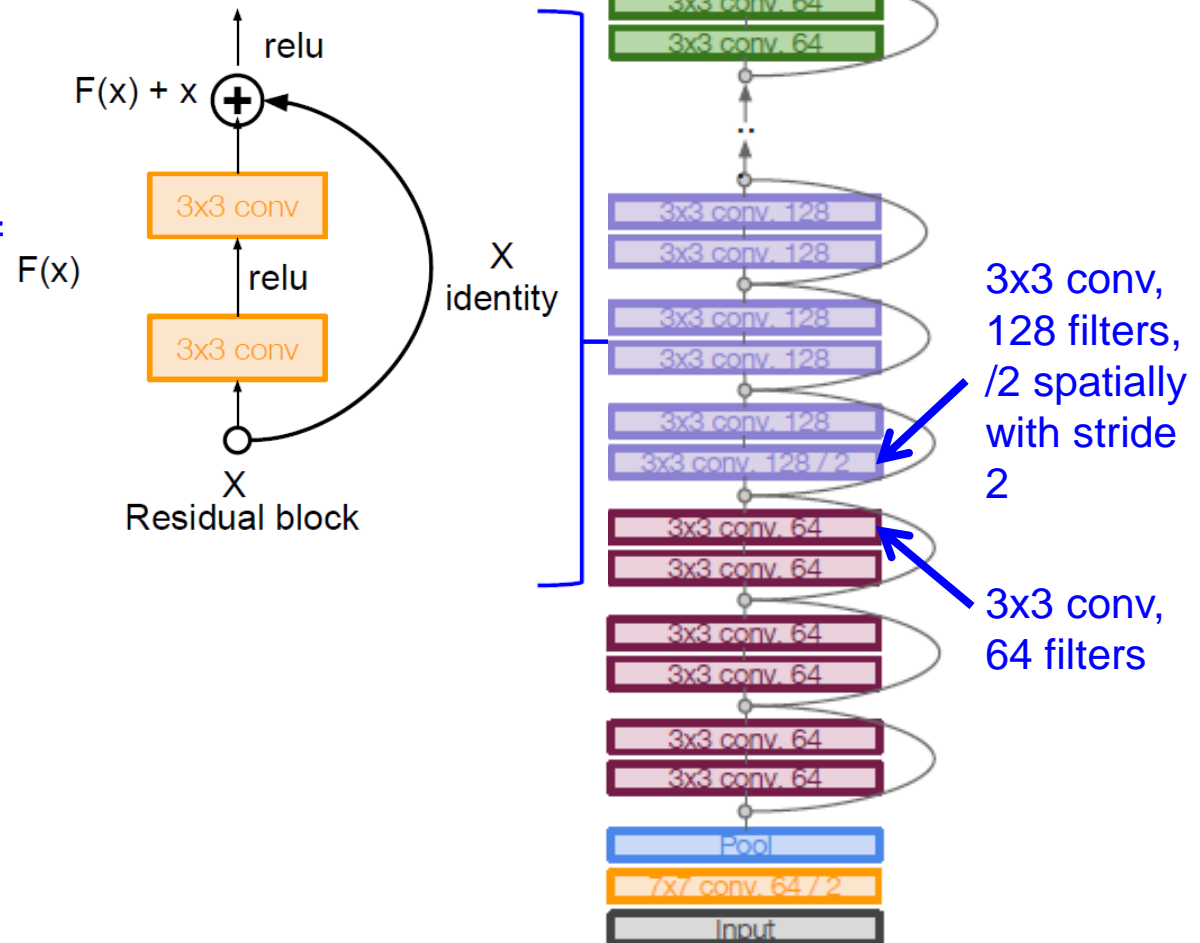
- Stack residual blocks
- Residual block has two 3x3 conv layers



ResNet

Full ResNet architecture:

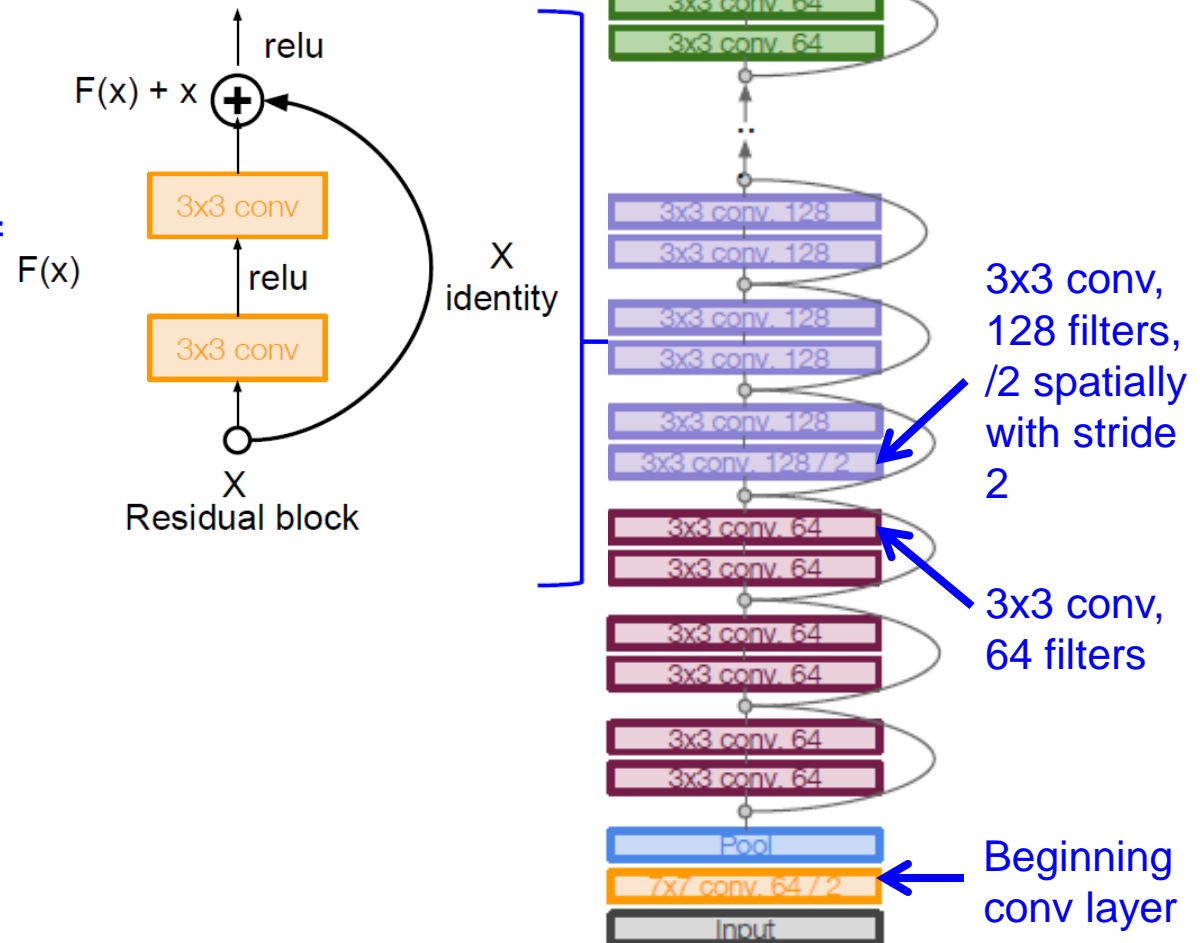
- Stack residual blocks
- Residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)



ResNet

Full ResNet architecture:

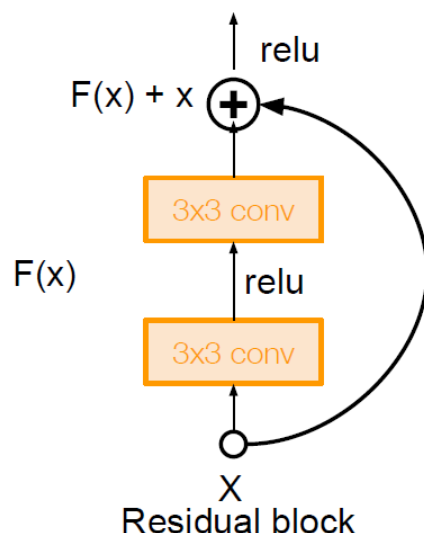
- Stack residual blocks
- Residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning



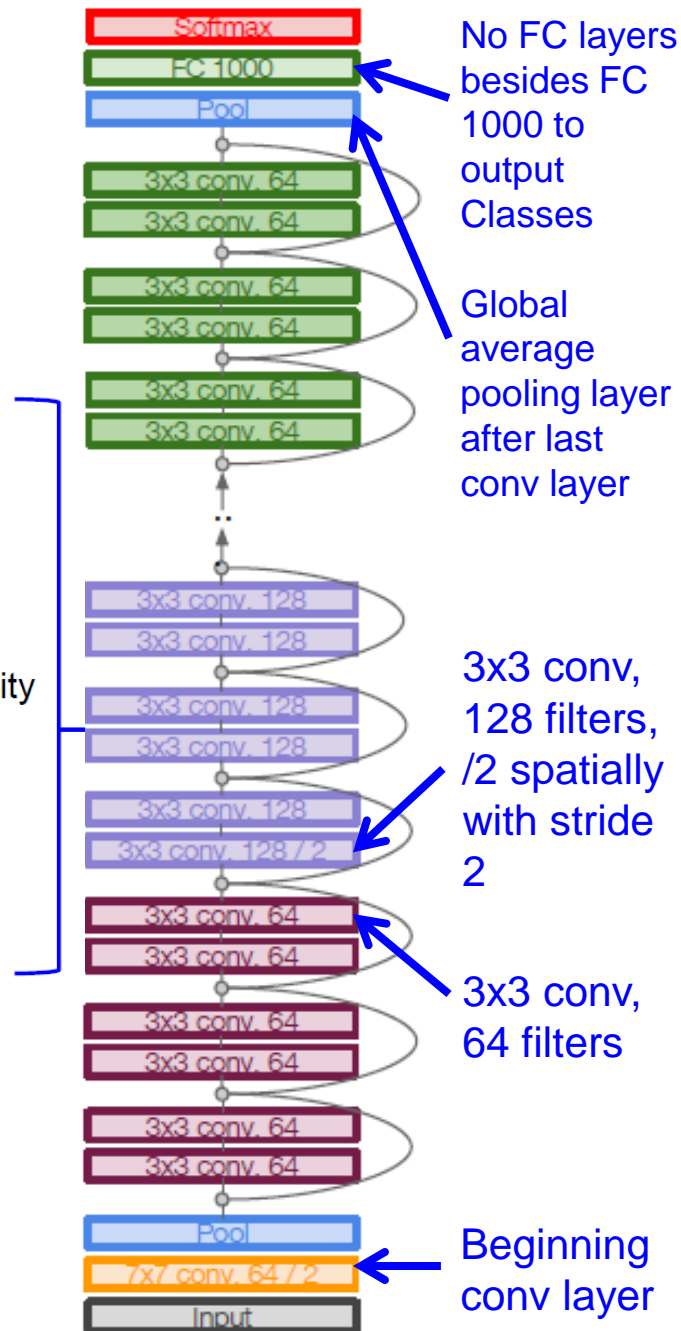
ResNet

Full ResNet architecture:

- Stack residual blocks
- 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
- No FC layers at the end (only FC 1000 to output classes)

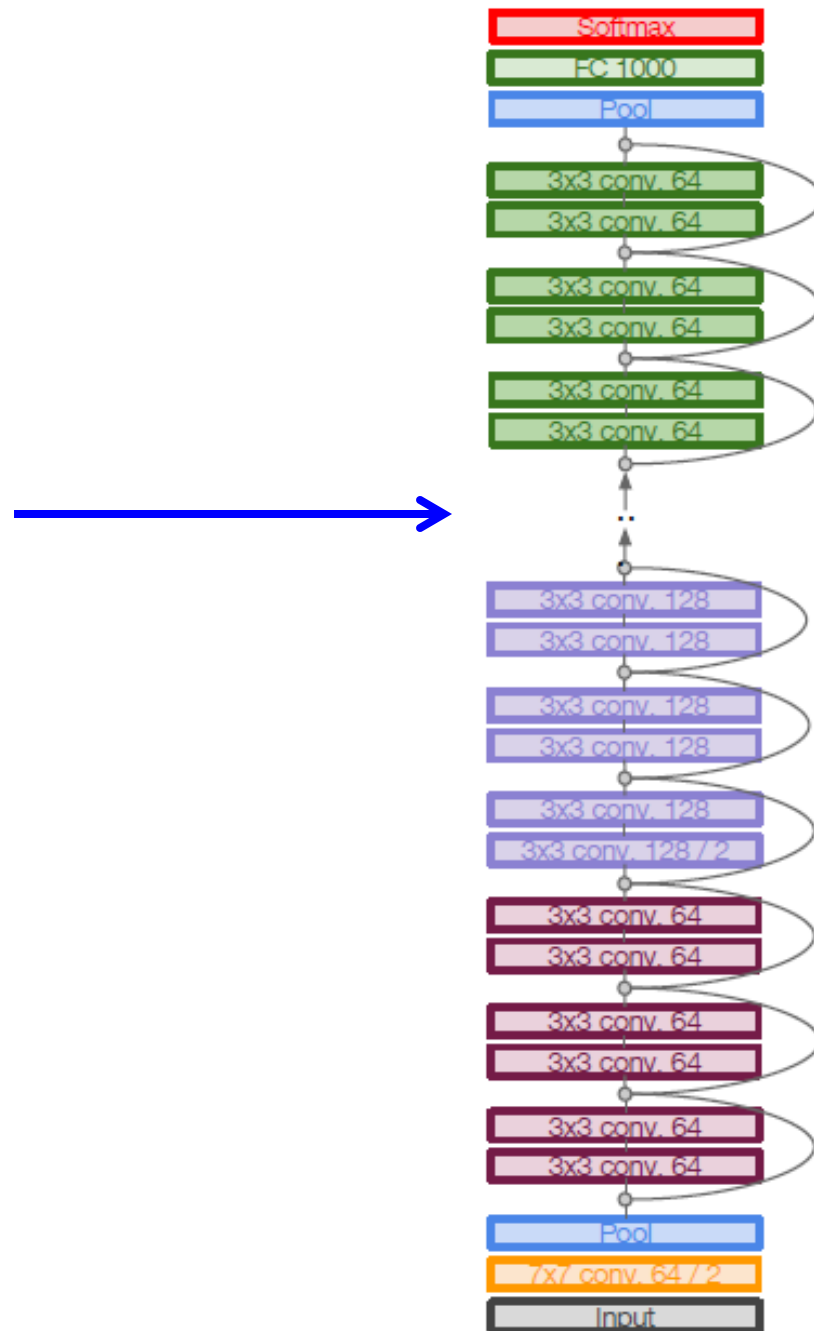


X
identity



ResNet

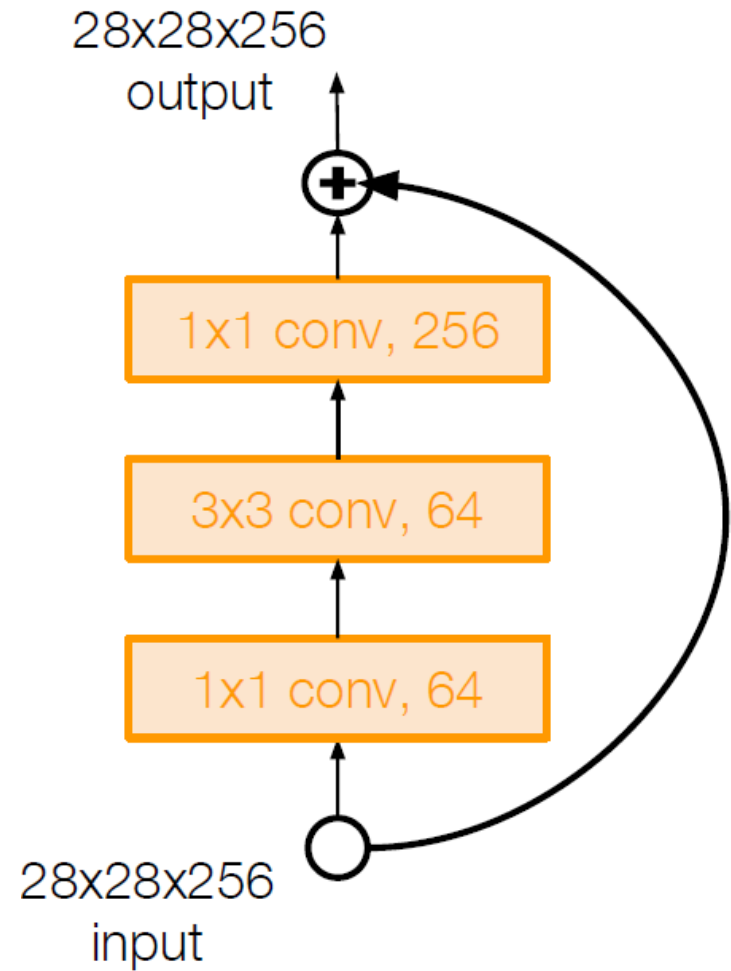
Total depths of 34, 50, 101, or 152 layers for ImageNet



ResNet

For deeper networks (ResNet-50+):

use “bottleneck” layer to improve
efficiency
(similar to GoogLeNet)



ResNet

Training ResNet in practice:

- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error saturates
- Mini-batch size 256
- Weight decay of $1e-5$ for penalizing regularization term
- No dropout used

ResNet

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

ResNet

Experimental Results:

- **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

- 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

ResNet

Experimental Results:

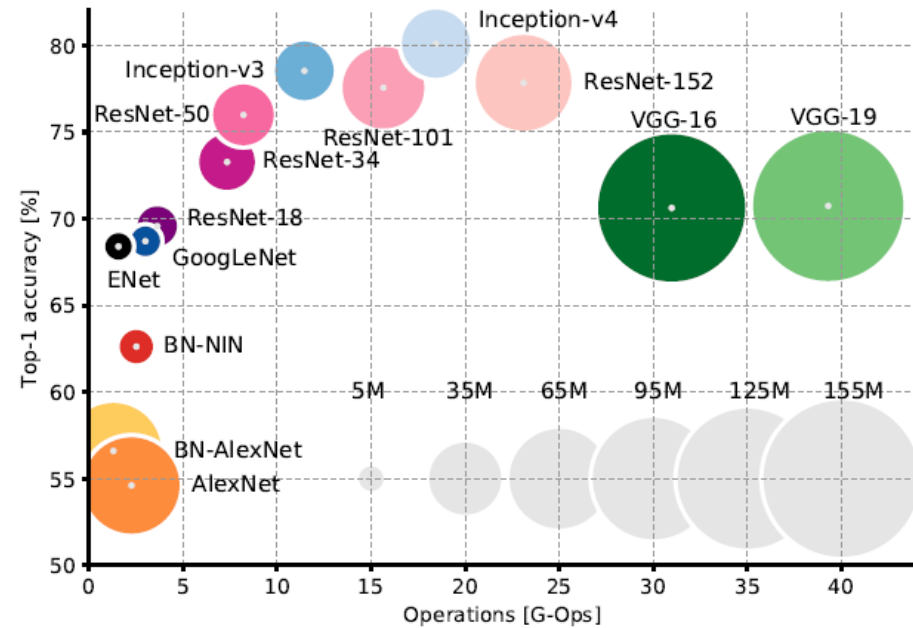
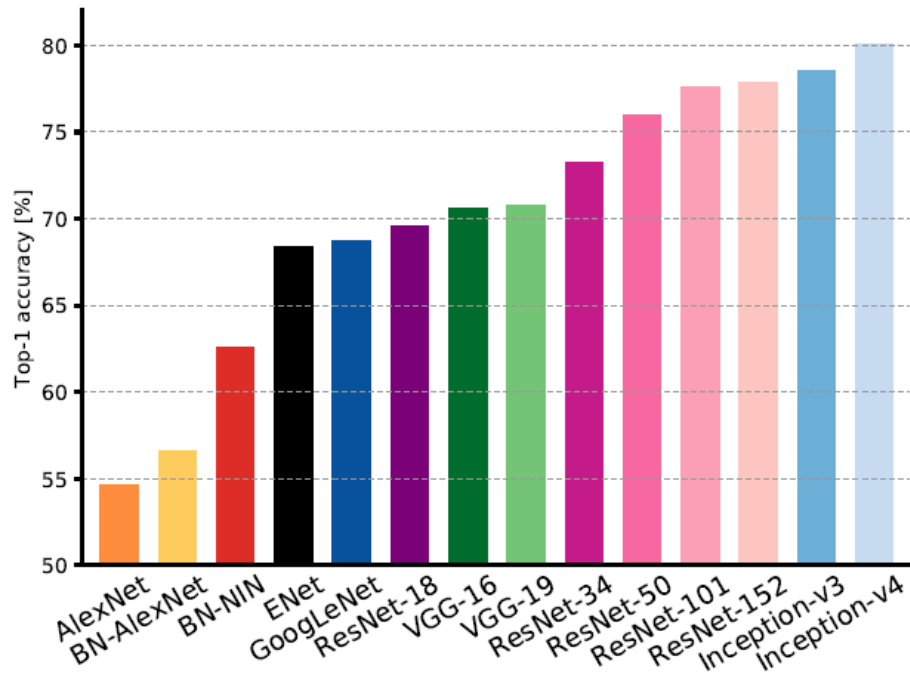
- **1st places in all five main tracks**

- 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

- 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

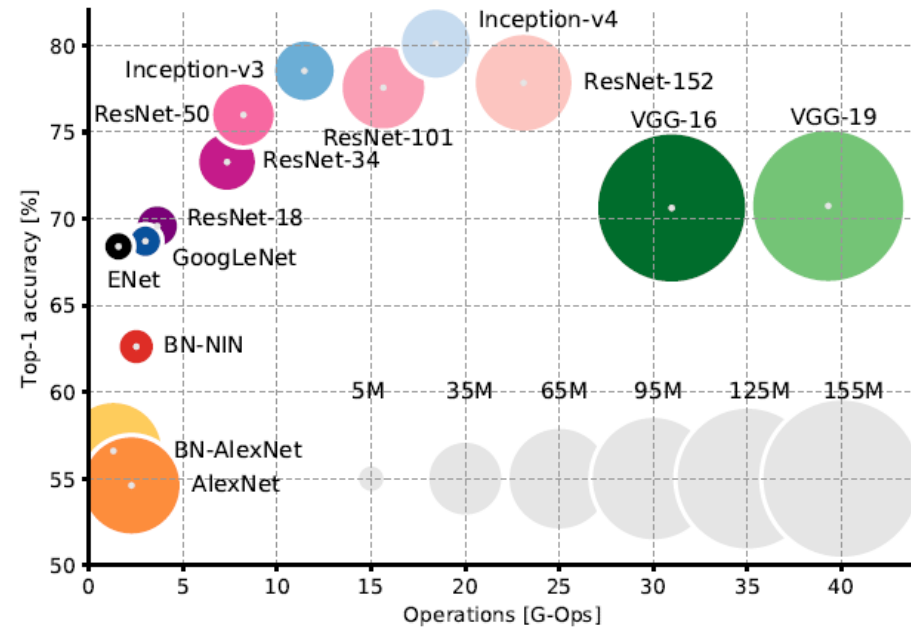
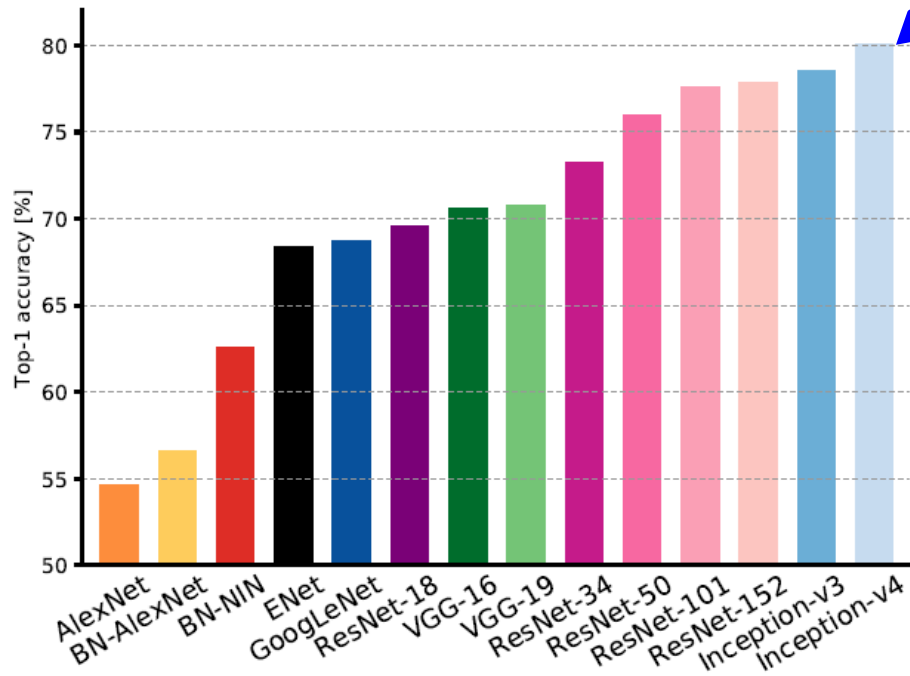
ILSVRC 2015 classification winner
(3.6% top 5 error) -- better than
“human performance”!
(Russakovsky 2014)

Comparing Complexity ...



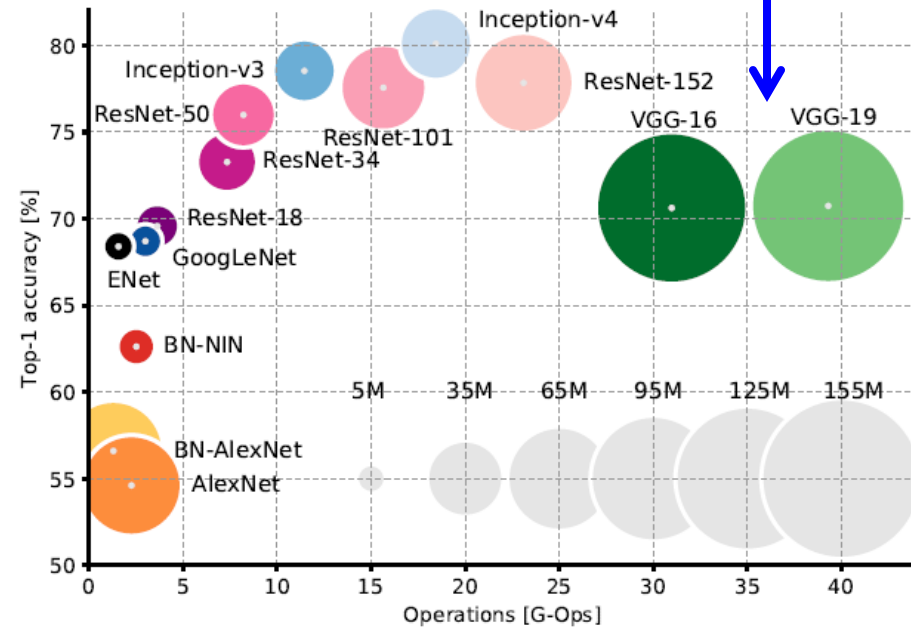
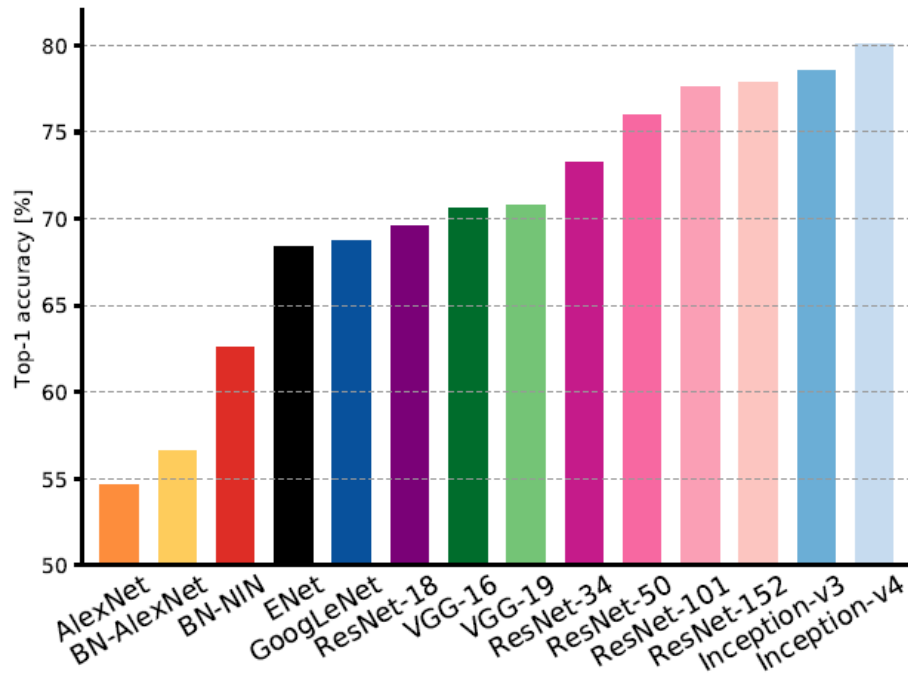
Comparing Complexity ...

Inception-v4: Resnet + Inception!



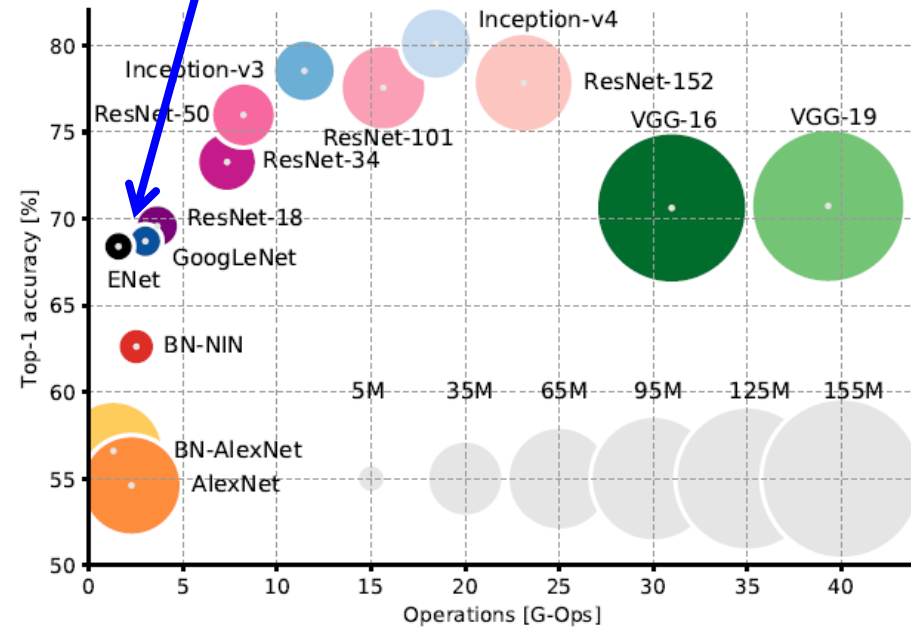
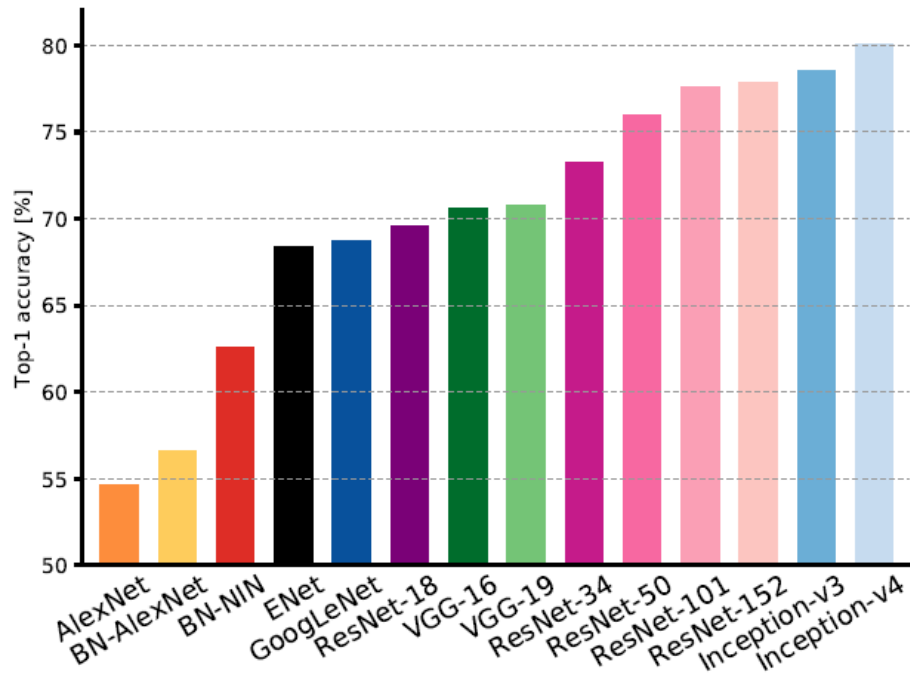
Comparing Complexity ...

VGG: Highest
memory, most
operations

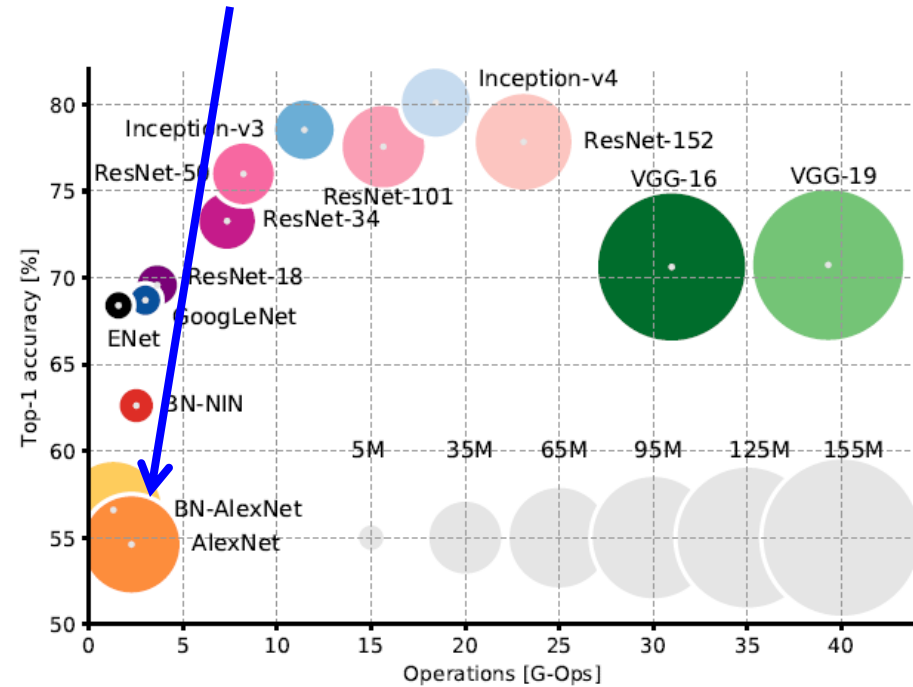
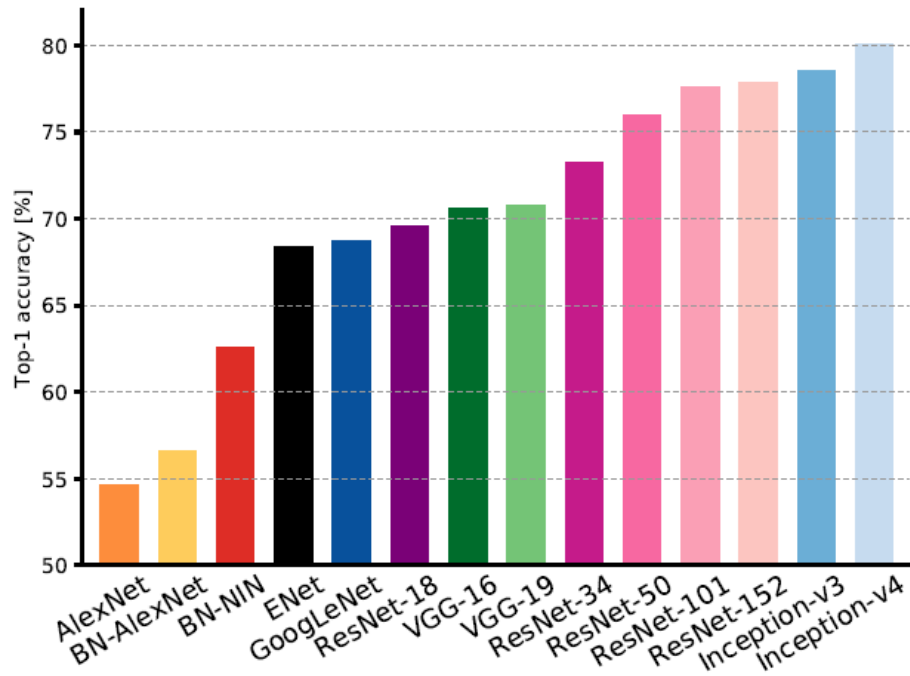


Comparing Complexity ...

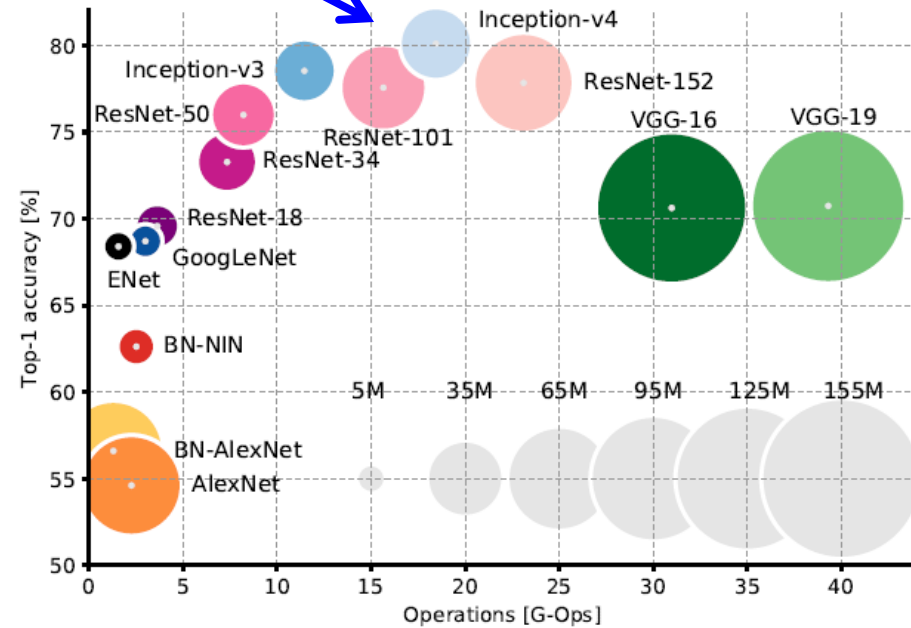
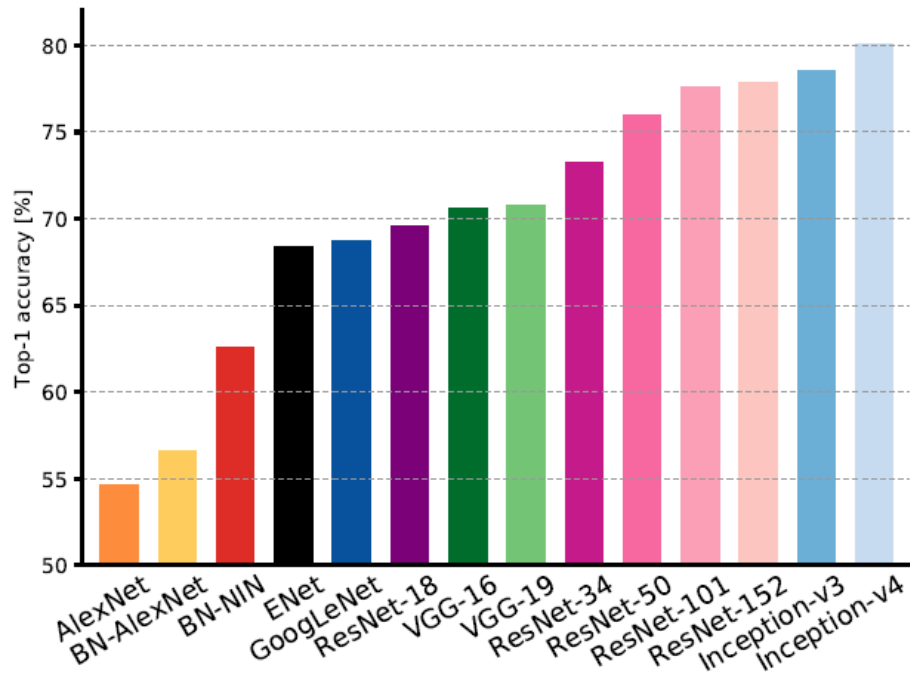
GoogLeNet:
most efficient



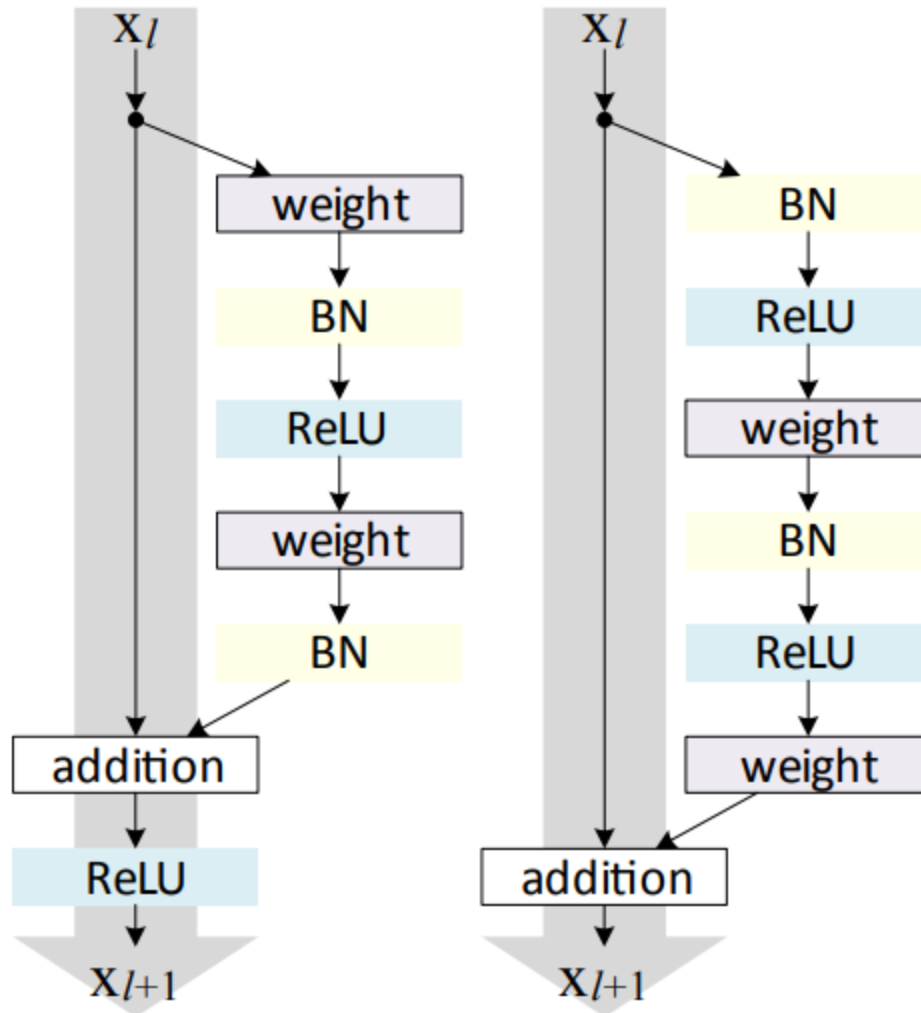
Comparing Complexity ... AlexNet: Smaller compute, still memory heavy, lower accuracy



Comparing Complexity ... ResNet: Moderate efficiency depending on model, highest accuracy



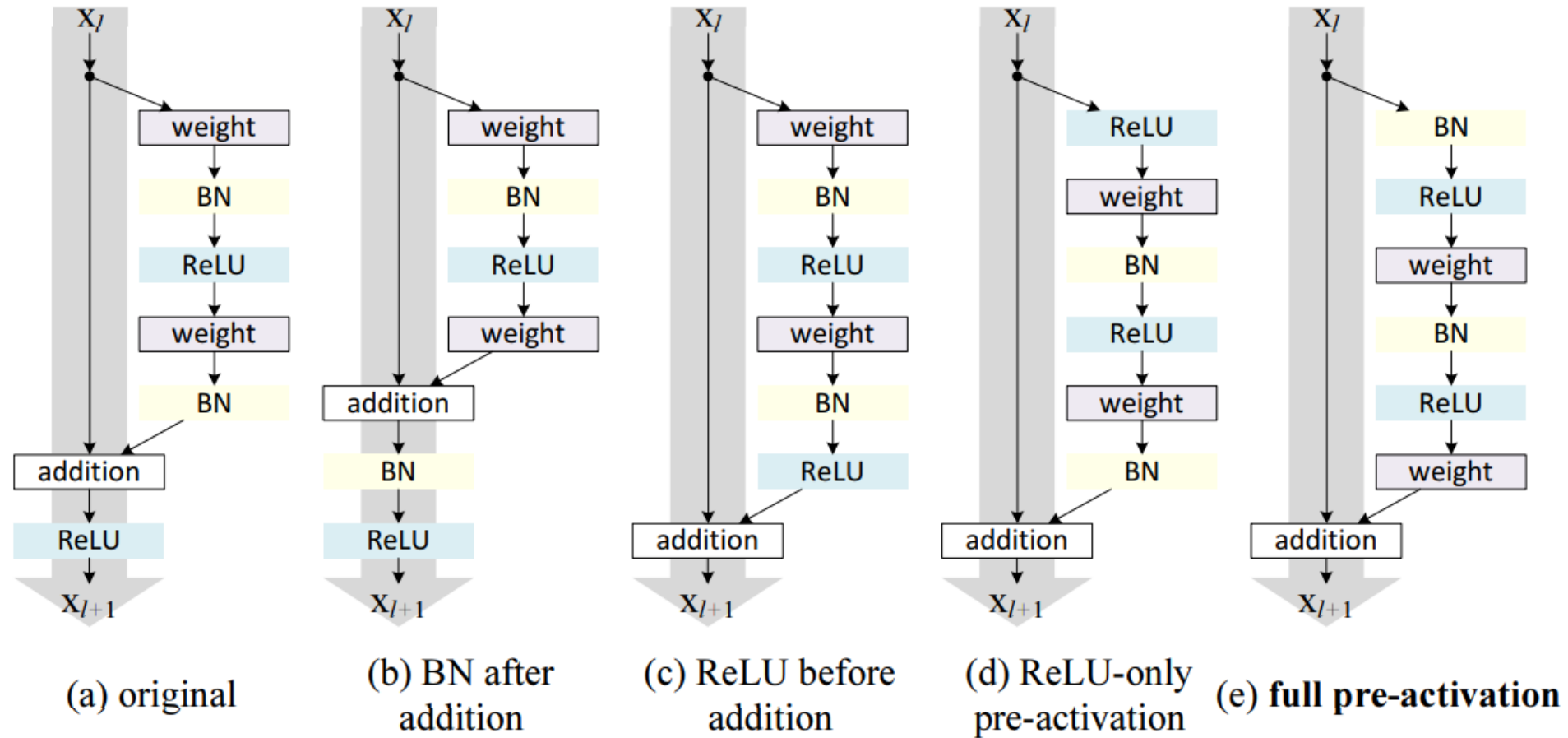
Pre-activated ResNet



(a) original

(b) proposed

Pre-activated ResNet



Pre-activated ResNet

Classification error (%) on the CIFAR-10 test set using different activation functions.

case	ResNet-110	ResNet-164
original Residual Unit [1]	6.61	5.93
BN after addition	8.17	6.50
ReLU before addition	7.84	6.14
ReLU-only pre-activation	6.71	5.91
full pre-activation	6.37	5.46

SENet (Squeeze and Excitation Network)

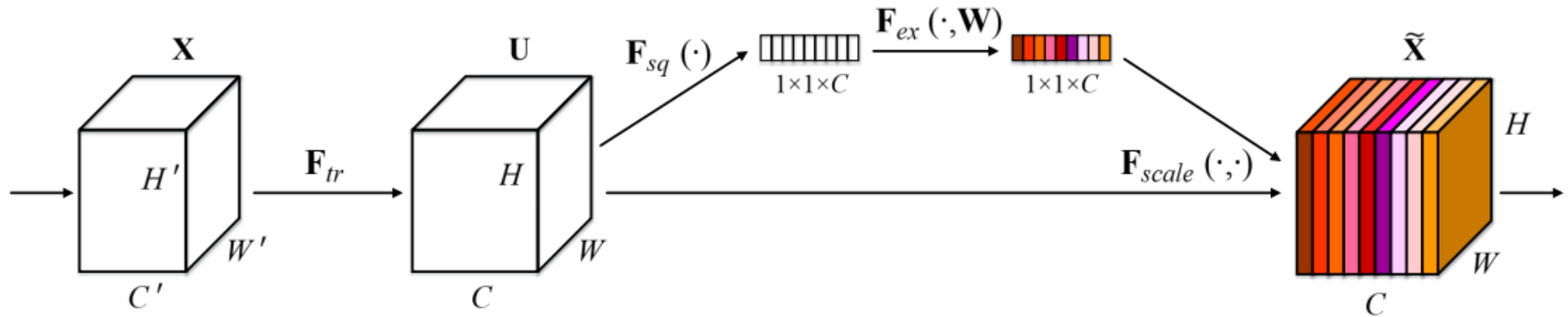
2017 ImageNet Challenge Winner

Top-5 Error: 2.251%

SENet (Squeeze and Excitation Network)

2017 ImageNet Challenge Winner

Top-5 Error: 2.251%

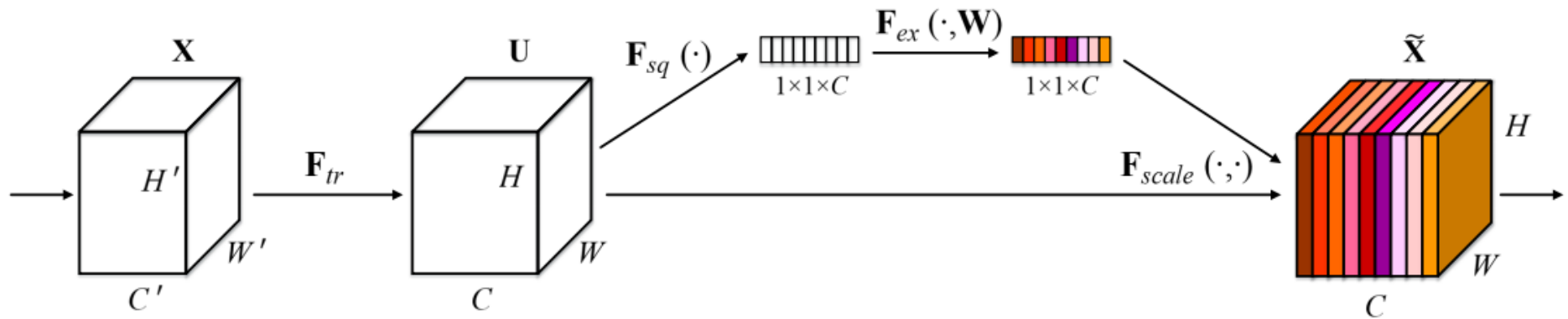


“Squeeze-and-Excitation”(SE) block adaptively recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels.

SENet (Squeeze and Excitation Network)

2017 ImageNet Challenge Winner

Top-5 Error: 2.251%



Squeeze: Average Global Pooling

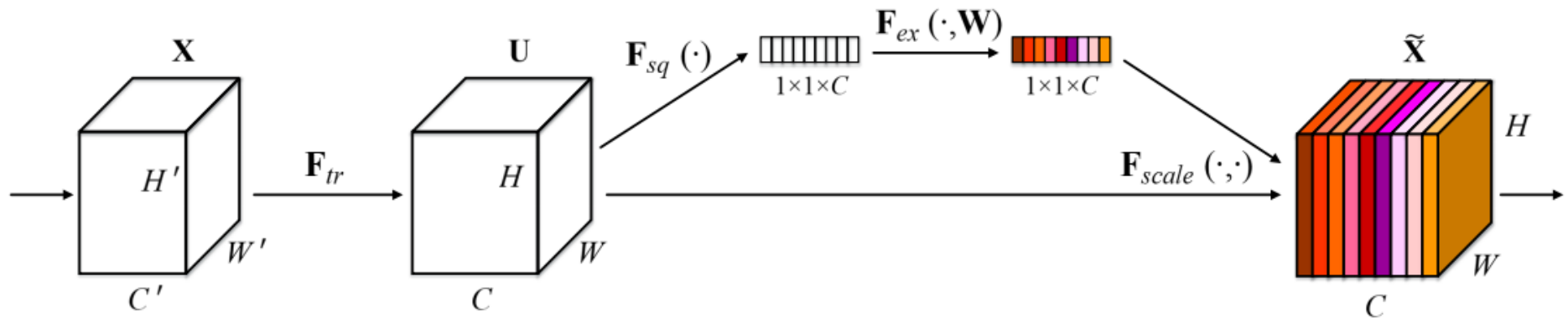
$$z_c = \mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{W \times H} \sum_{i=1}^W \sum_{j=1}^H u_c(i, j)$$

c^{th} channel of C

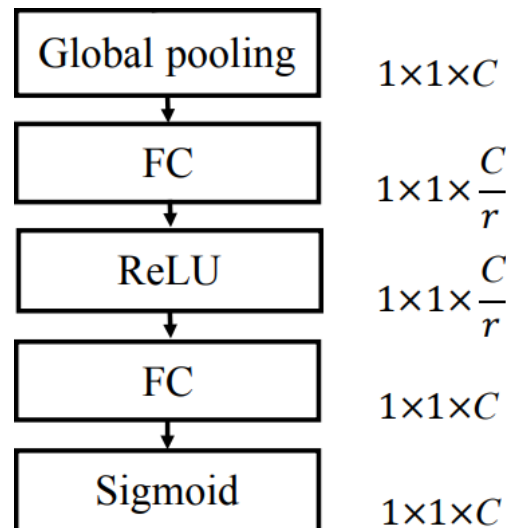
SENet (Squeeze and Excitation Network)

2017 ImageNet Challenge Winner

Top-5 Error: 2.251%



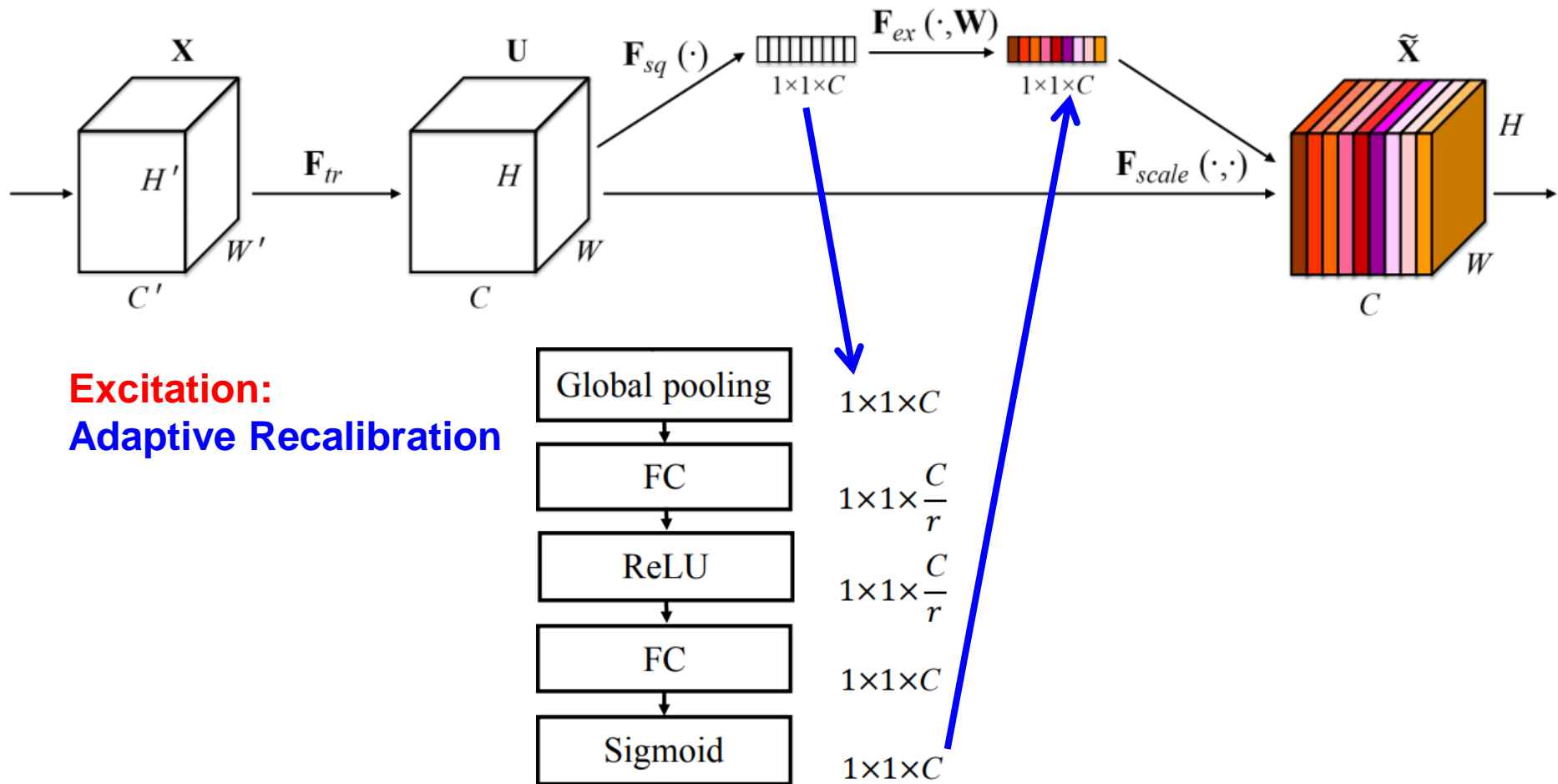
Excitation:
Adaptive Recalibration



SENet (Squeeze and Excitation Network)

2017 ImageNet Challenge Winner

Top-5 Error: 2.251%

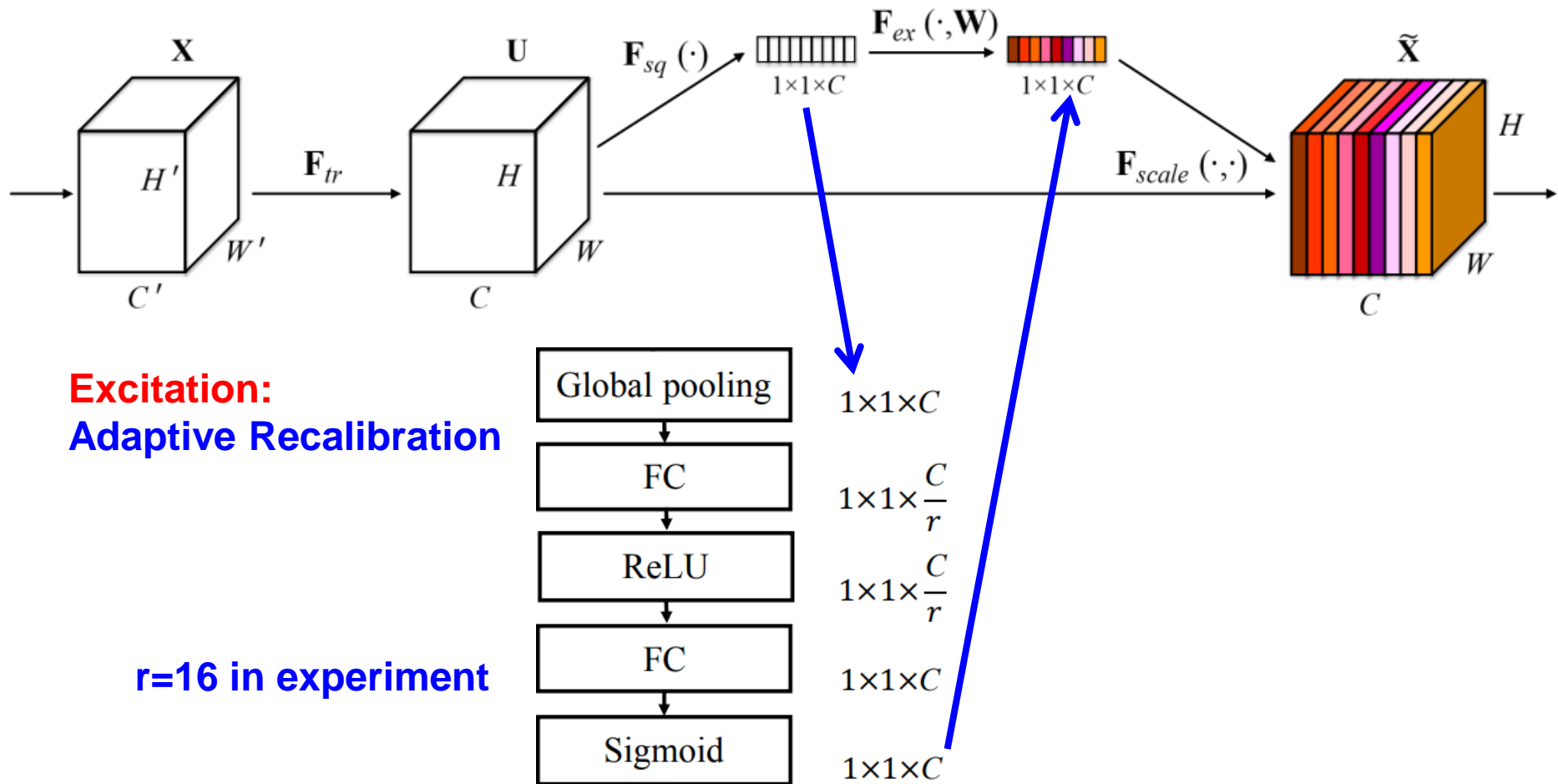


Excitation:
Adaptive Recalibration

SENet (Squeeze and Excitation Network)

2017 ImageNet Challenge Winner

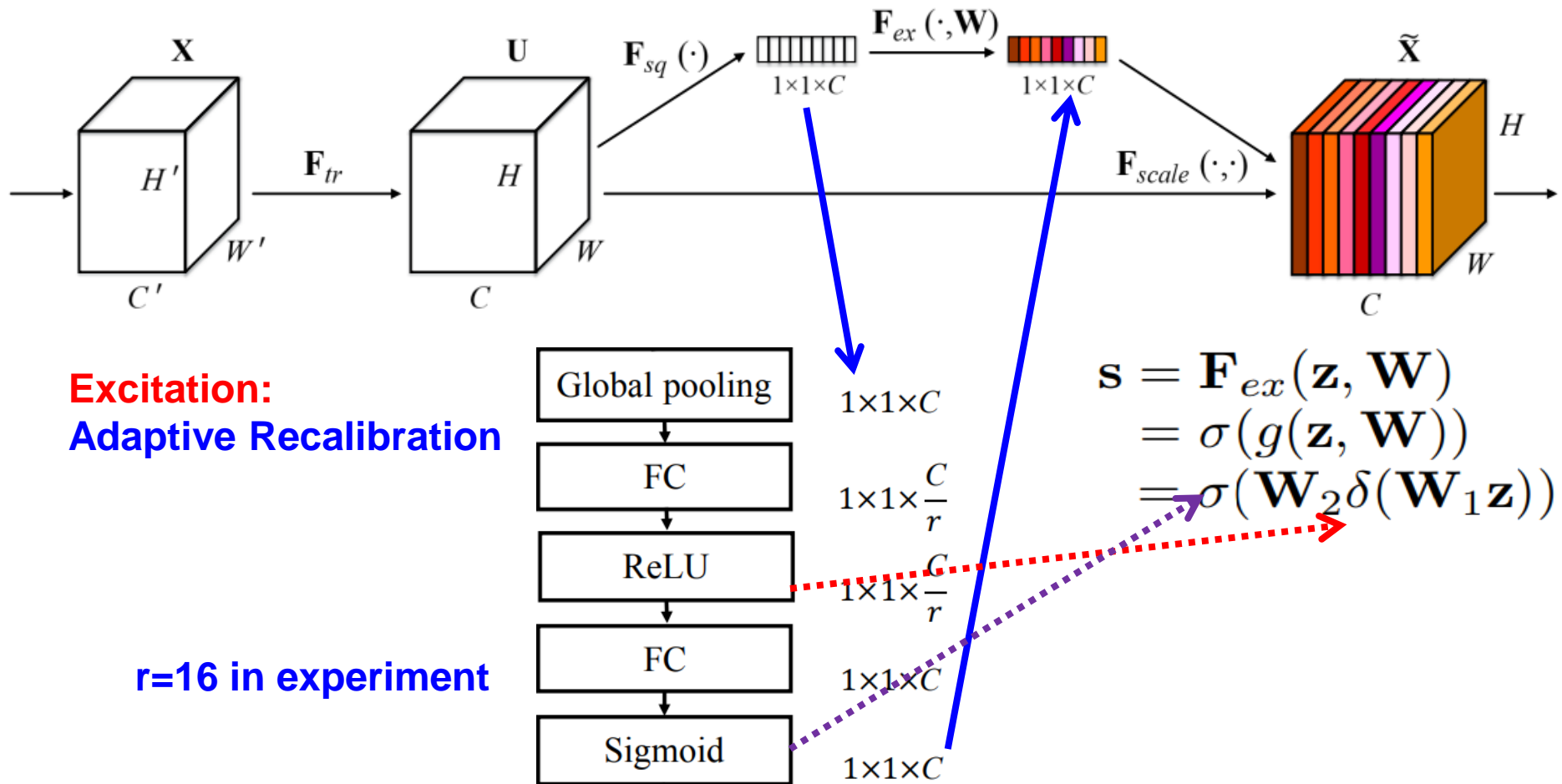
Top-5 Error: 2.251%



SENet (Squeeze and Excitation Network)

2017 ImageNet Challenge Winner

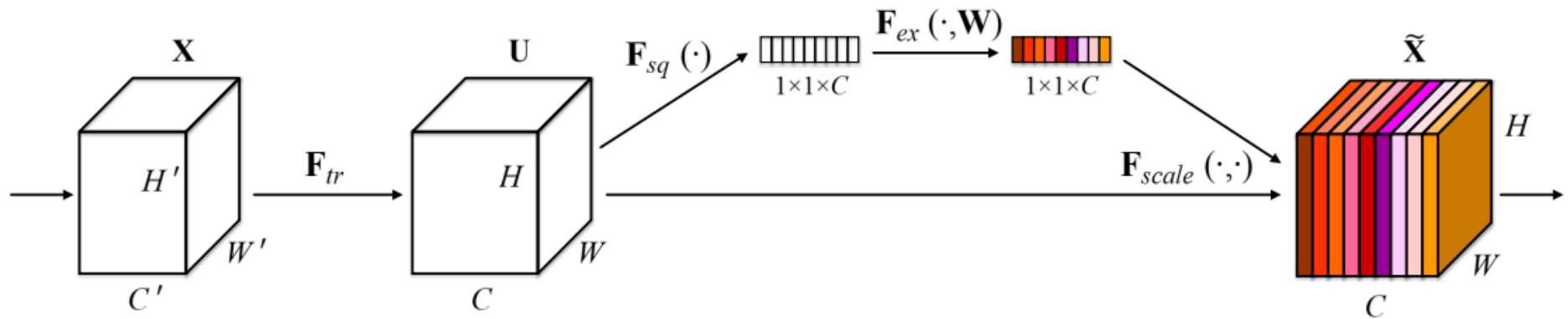
Top-5 Error: 2.251%



SENet (Squeeze and Excitation Network)

2017 ImageNet Challenge Winner

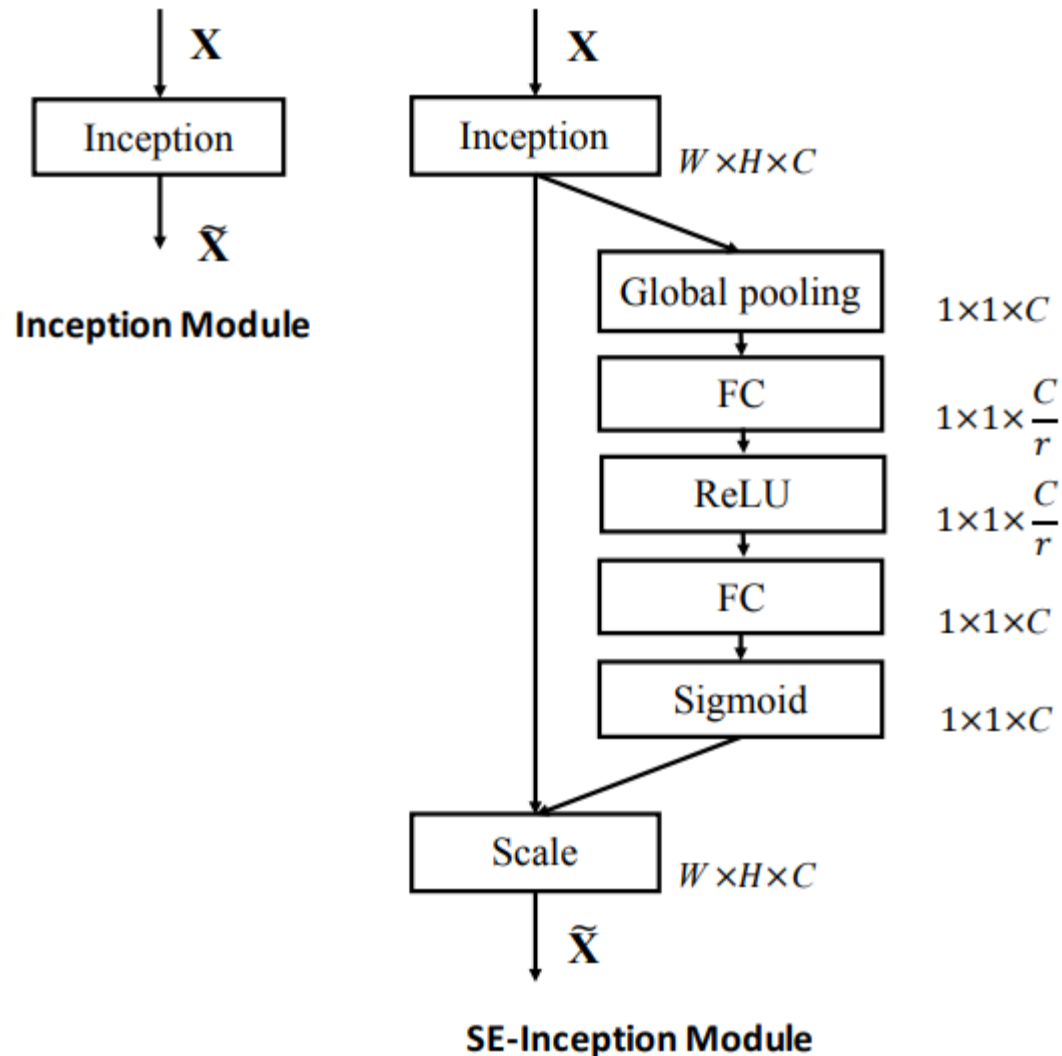
Top-5 Error: 2.251%



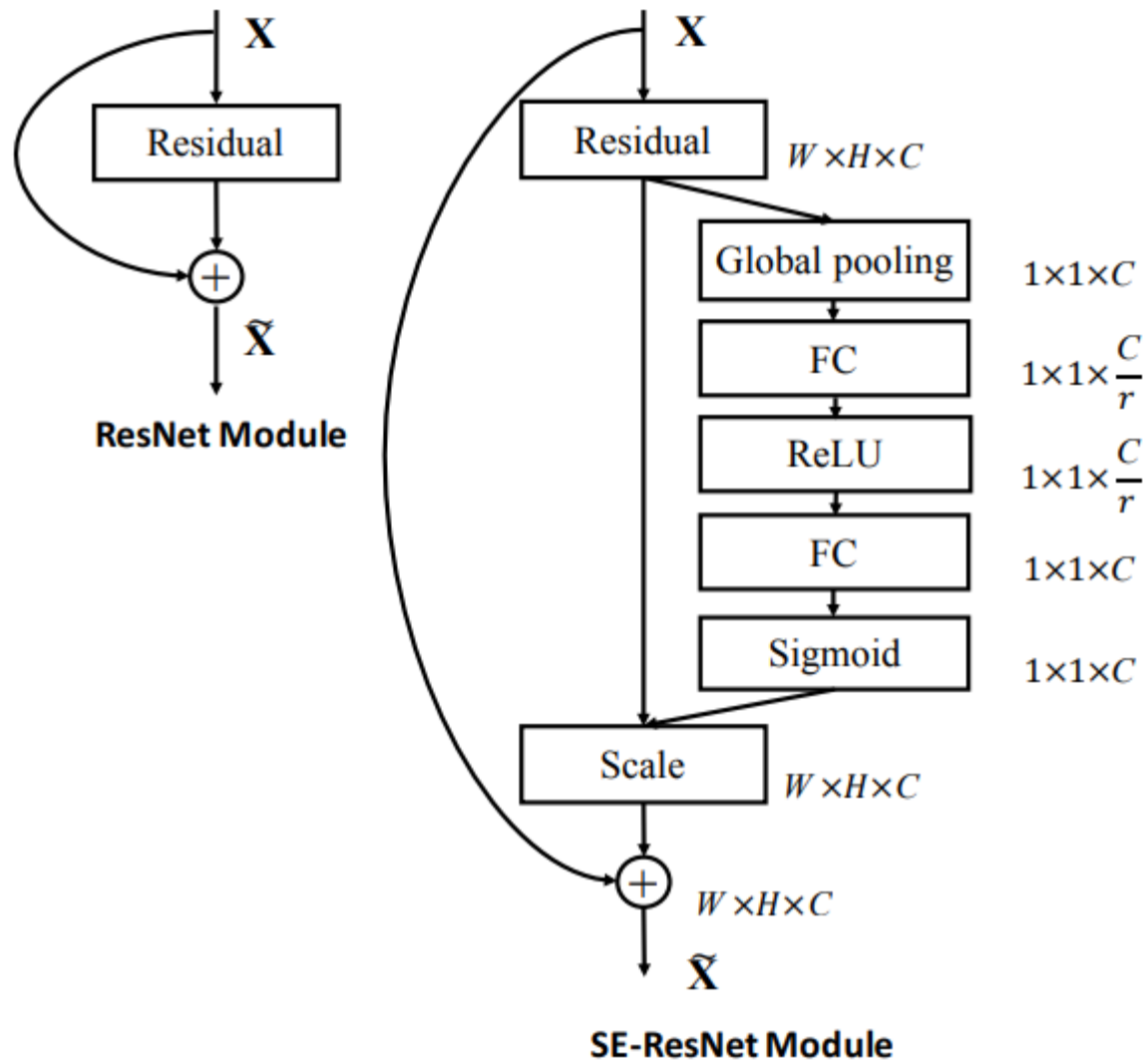
Scaling:

$$\tilde{\mathbf{x}}_c = \mathbf{F}_{scale}(\mathbf{u}_c, s_c) = s_c \cdot \mathbf{u}_c$$

SE-Inception Module



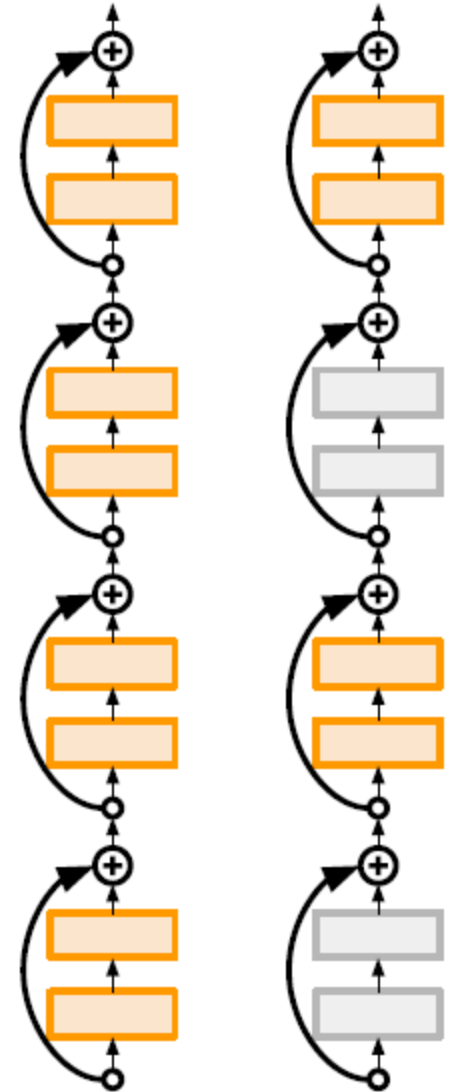
SE-ResNet Module



Other ResNet Improvements to Know ...

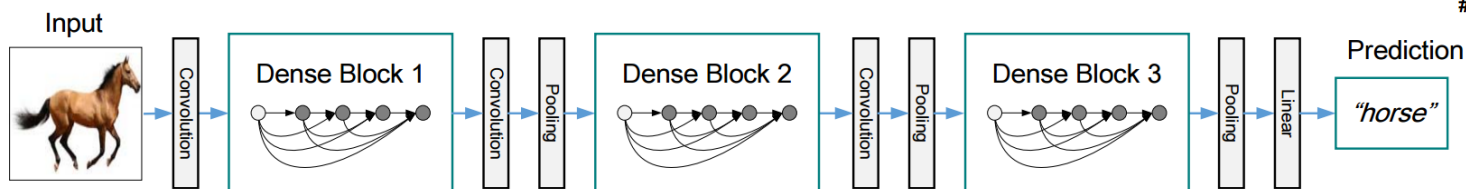
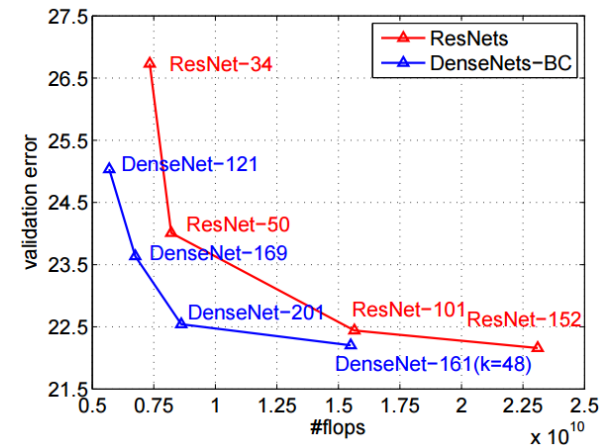
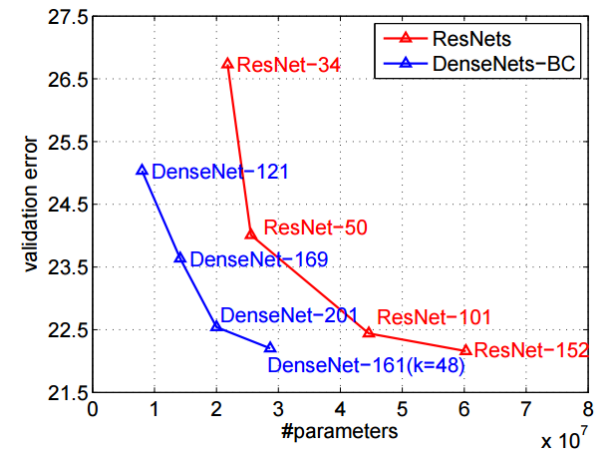
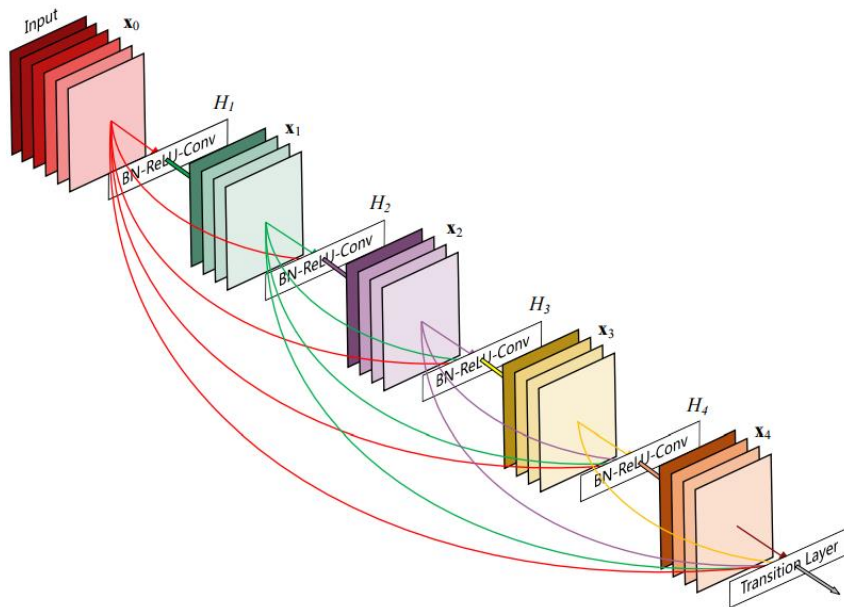
Deep Networks with Stochastic Depth

- Motivation: reduce vanishing gradients and training time through short networks during training
- Randomly drop a subset of layers during each training pass
- Bypass with identity function
- Use full deep network at test time



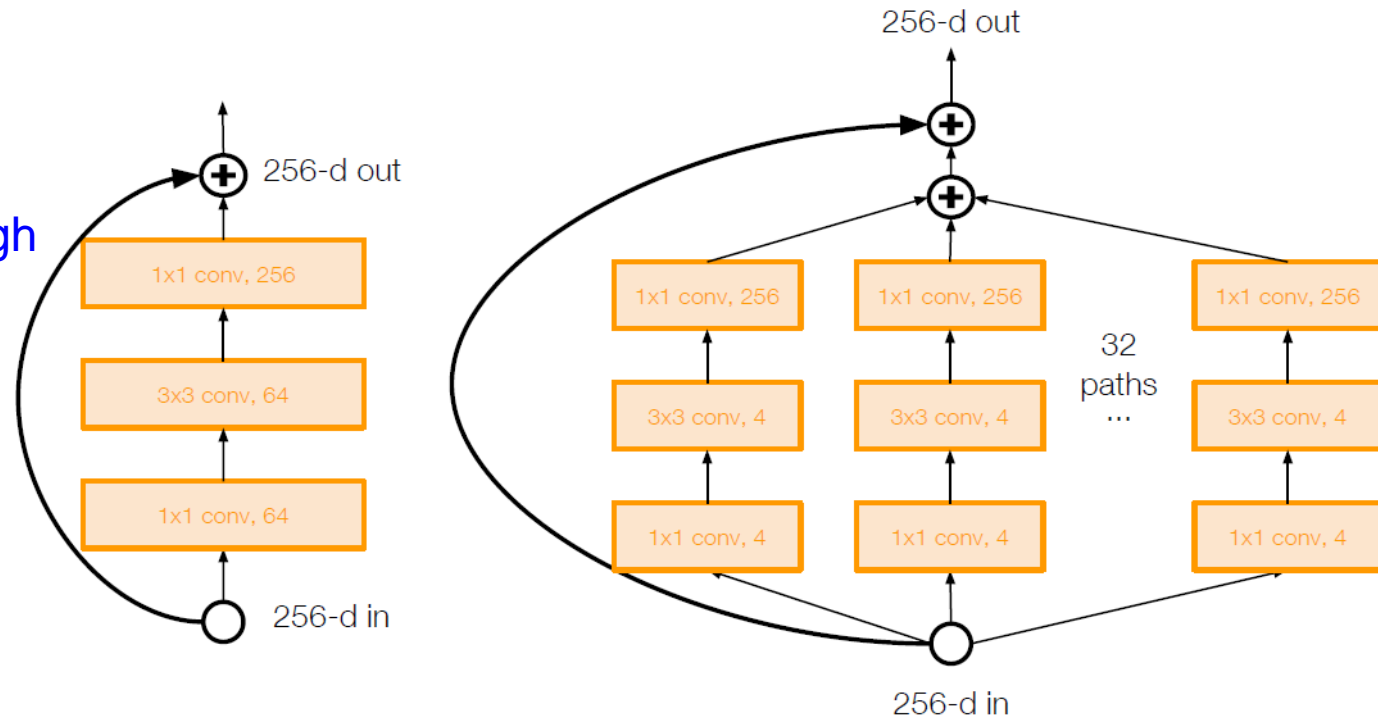
DenseNet

- Shorter connections (like ResNet) help
- Why not just connect them all?



Aggregated Residual Transformations for Deep Neural Networks (ResNeXt)

- Improved ResNet
- Increases width of residual block through multiple parallel pathways (“cardinality”)
- Parallel pathways similar in spirit to Inception module



Things to remember

- Architectures: Plain Models
 - LeNet (1998)
 - 5 Layers
 - No progress till 2012 due to lack of large scale data and computational resources
 - AlexNet (2012)
 - 8 Layers
 - Game changer in Computer Vision Area
 - ZFNet (2013)
 - 8 Layers with improved hypermeter setting
 - VGGNet (2014)
 - Deeper model: 16 or 19 Layers
 - Uniform filters
 - NiN (Network in Network) (2014)
 - Inspiration to DAG Model

Things to remember

- DAG Architectures

- GoogLeNet
- ResNet
- Pre-activated ResNet
- SENet
- Stochastic Depth
- DenseNet
- ResNetXt

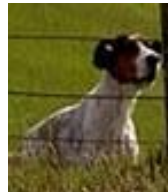
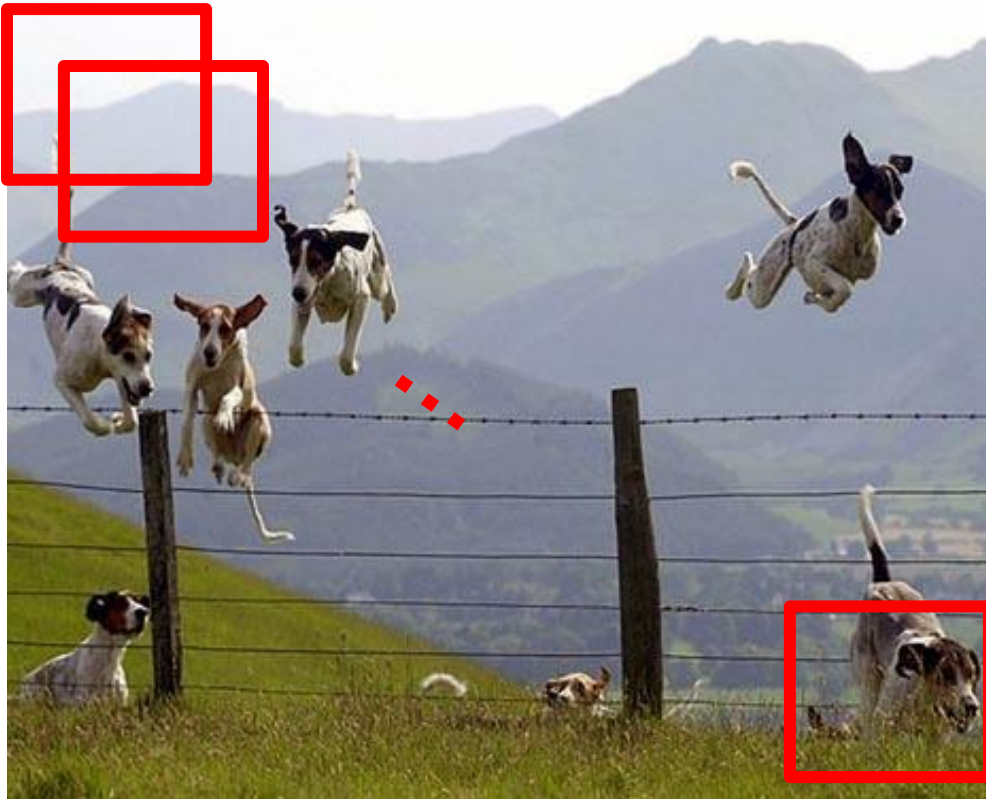
- VGG, GoogLeNet, ResNet all in wide use, available in model zoos
- ResNet current best default
- Trend towards extremely deep networks
- Significant research centers around design of layer / skip connections and improving gradient flow
- Even more recent trend towards examining necessity of depth vs. width and residual connections

Acknowledgements

- Thanks to the following researchers for making their teaching/research material online
 - Forsyth
 - Steve Seitz
 - Noah Snavely
 - J.B. Huang
 - Derek Hoiem
 - D. Lowe
 - A. Bobick
 - S. Lazebnik
 - K. Grauman
 - R. Zaleski
 - Antonio Torralba
 - Rob Fergus
 - Leibe
 - And many more

Next Class

Object Detection



**Object or
Non-Object?**