Lecture 8:
CNN Architectures

Reminder: A2 due today!

Due at 11:59pm

Remember to **run the validation script!**

Soon: Assignment 3!

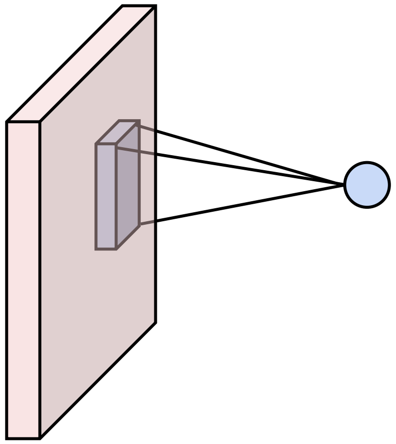
Modular API for backpropagation

Fully-connected networks
Dropout
Update rules: SGD+Momentum, RMSprop, Adam
Convolutional networks
Batch normalization

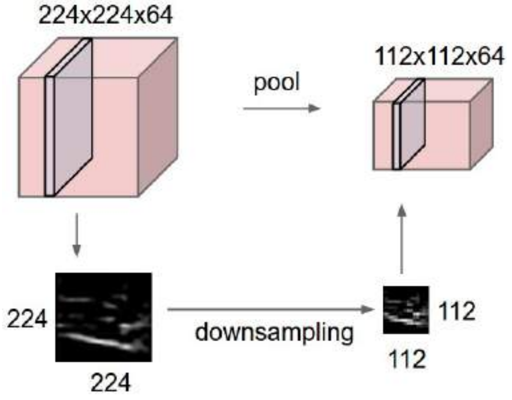
Will be released today or tomorrow
Will be due two weeks from the day it is released

Last Time: Components of Convolutional Networks

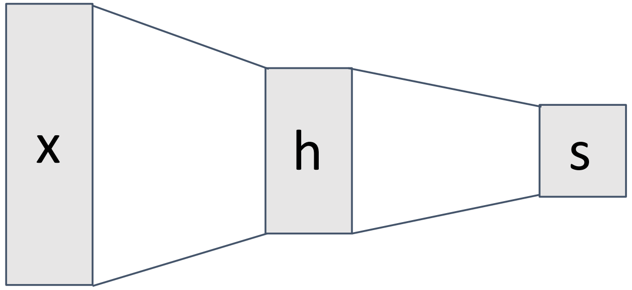
Convolution Layers



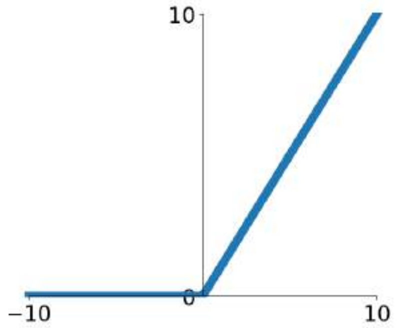
Pooling Layers



Fully-Connected Layers



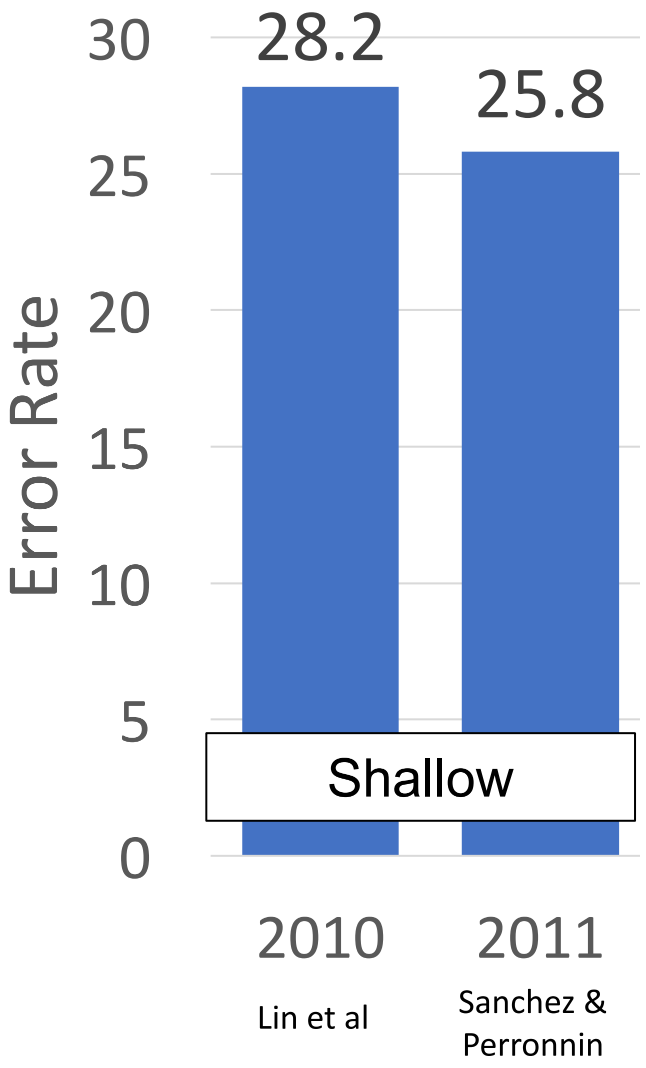
Activation Function



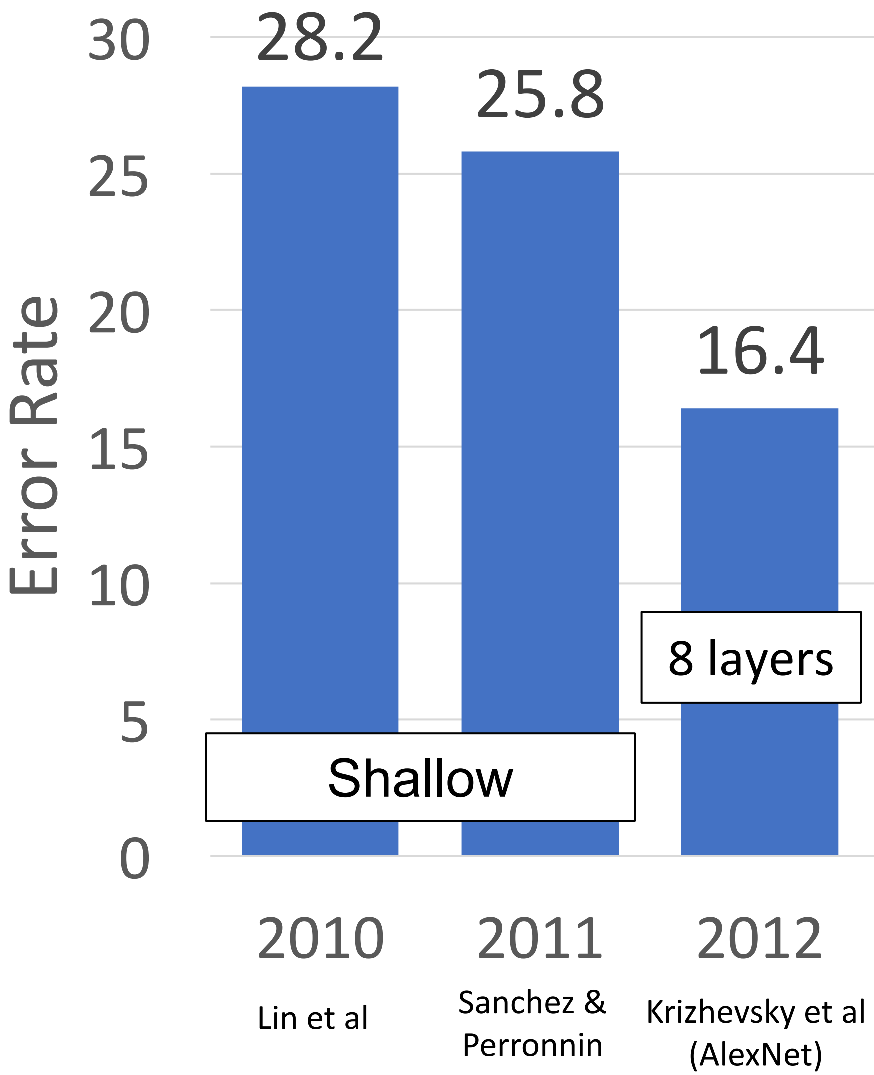
Normalization

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

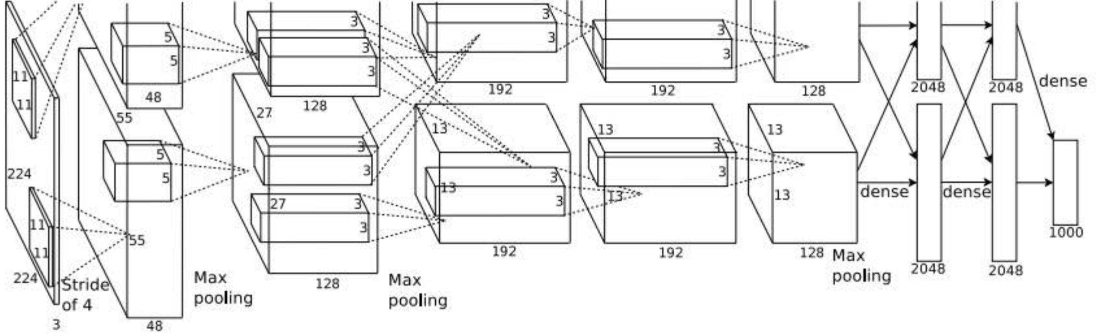
ImageNet Classification Challenge



ImageNet Classification Challenge

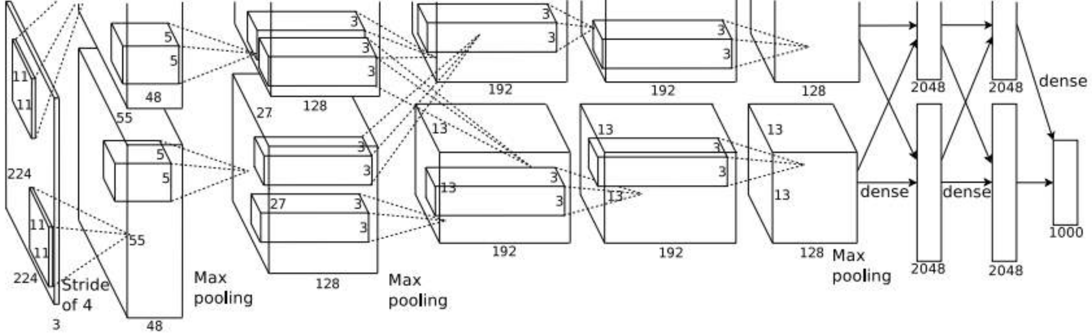


AlexNet



227 x 227 inputs
5 Convolutional layers
Max pooling
3 fully-connected layers
ReLU nonlinearities

AlexNet

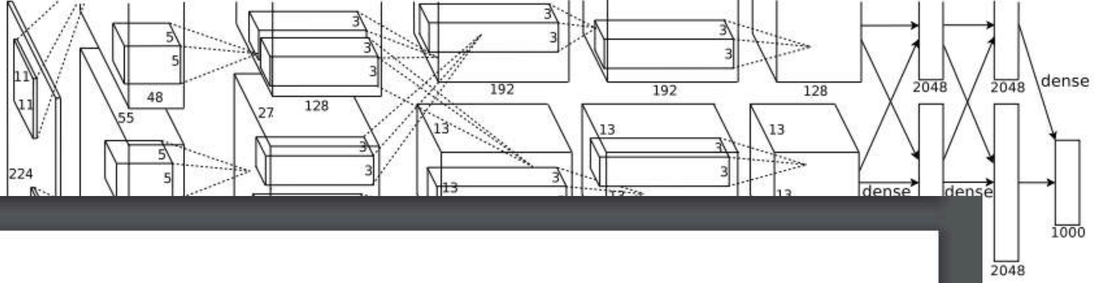


227 x 227 inputs
5 Convolutional layers
Max pooling
3 fully-connected layers
ReLU nonlinearities

Used "Local response normalization";
Not used anymore

Trained on two GTX 580 GPUs – only
3GB of memory each! Model split
over two GPUs

AlexNet



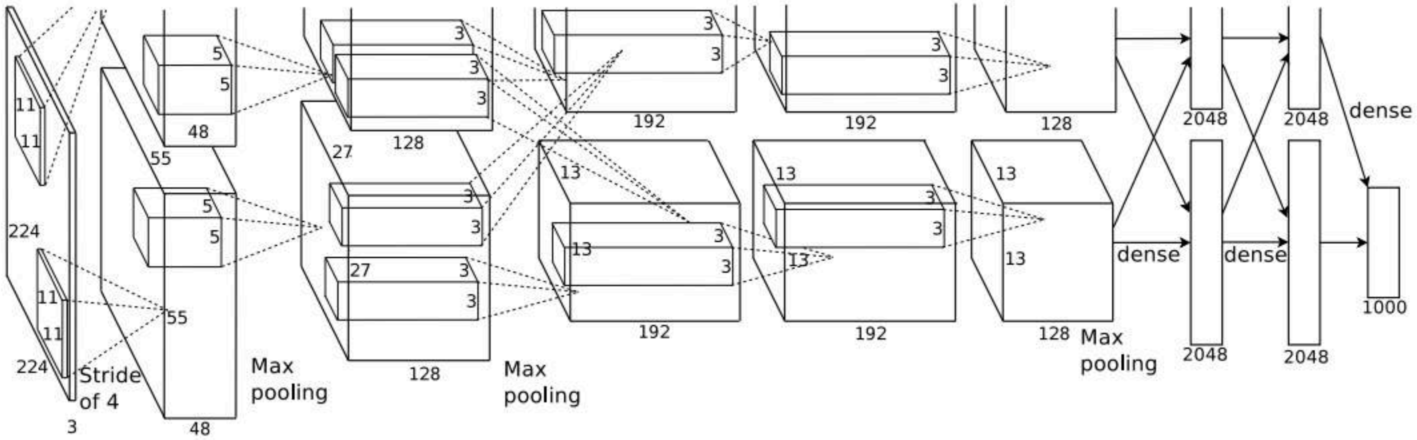
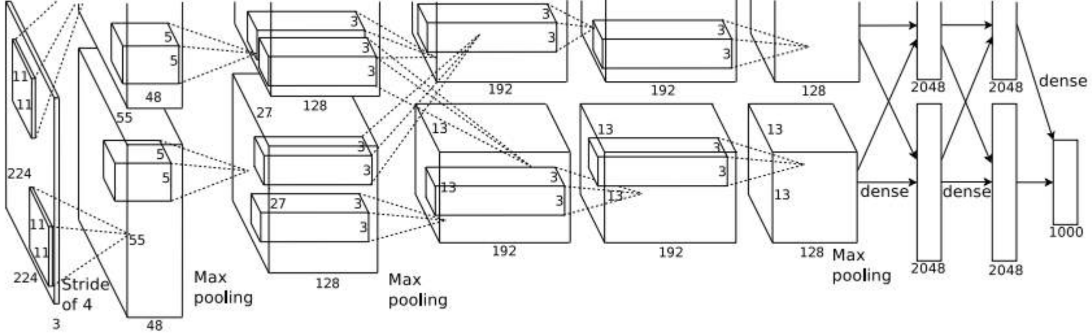


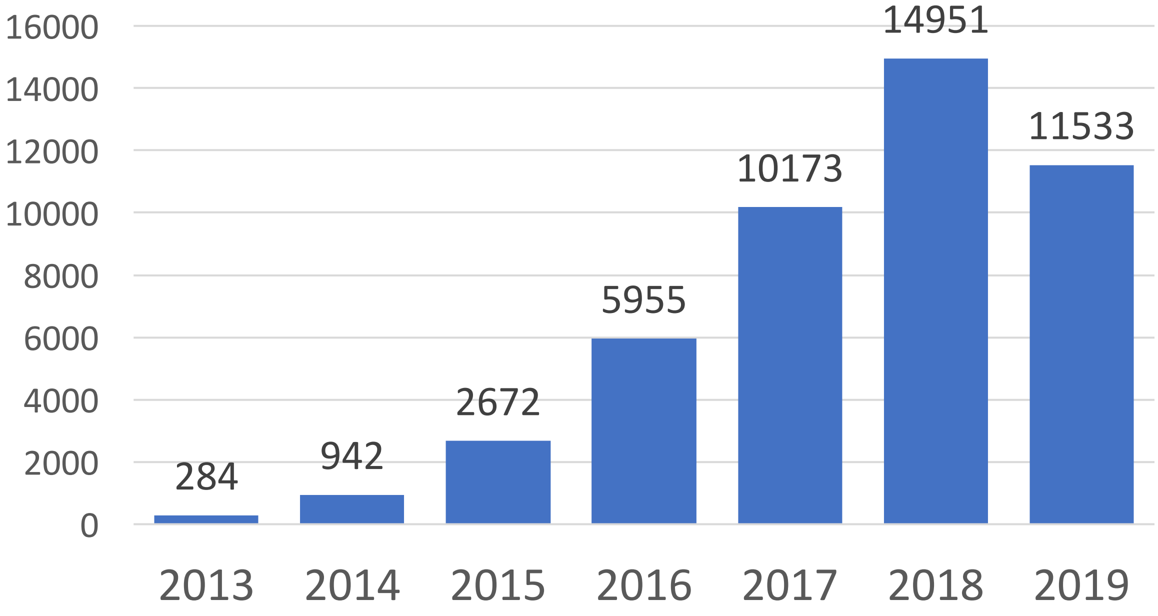
Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

AlexNet



AlexNet Citations per year

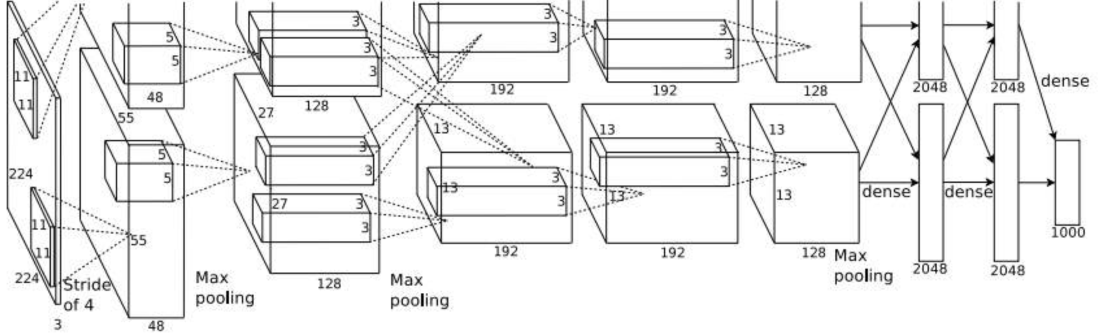
(As of 9/30/2019)



Total Citations: **46,510**

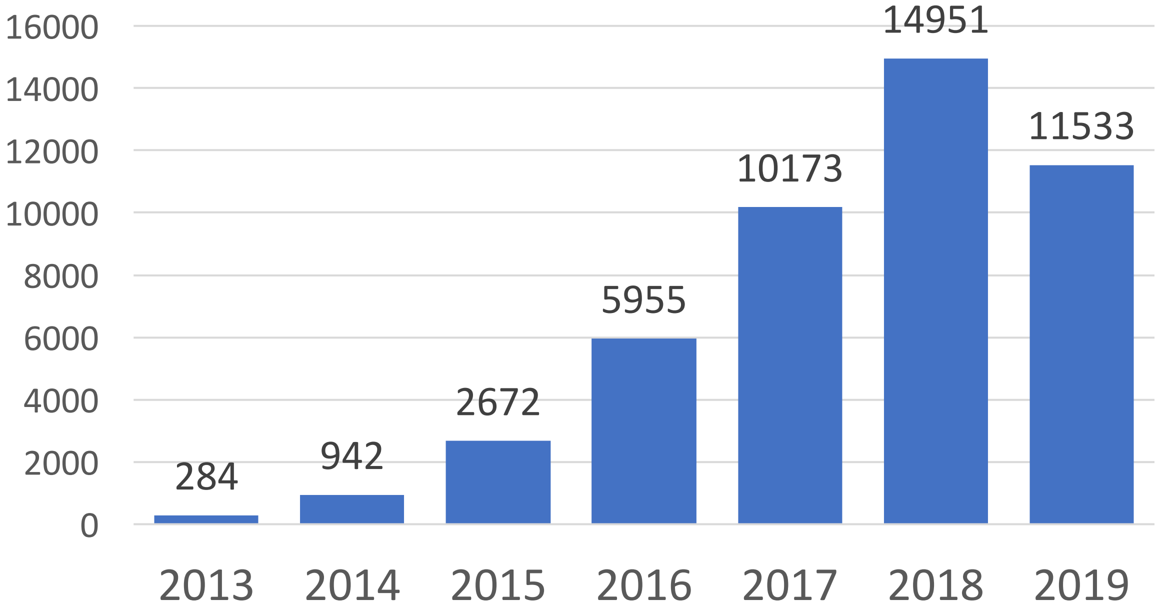
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

AlexNet



AlexNet Citations per year

(As of 9/30/2019)



Total Citations: **46,510**

Citation Counts

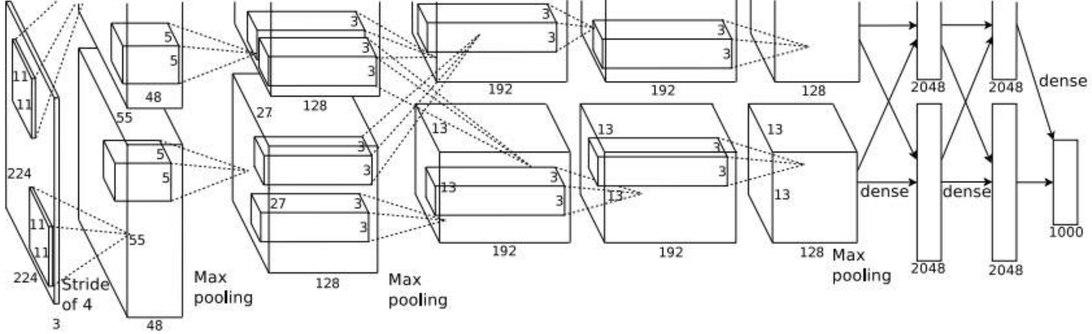
Darwin, “On the origin of species”, 1859: **50,007**

Shannon, “A mathematical theory of communication”, 1948: **69,351**

Watson and Crick, “Molecular Structure of Nucleic Acids”, 1953: **13,111**

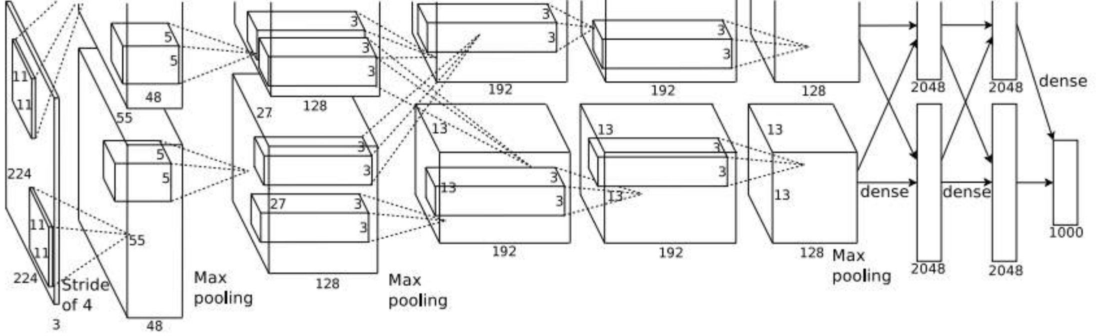
ATLAS Collaboration, “Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC”, 2012: **14,424**

AlexNet



	Input size		Layer				Output size	
Layer	C	H / W	filters	kernel	stride	pad	C	H / W
conv1	3	227	64	11	4	2	?	

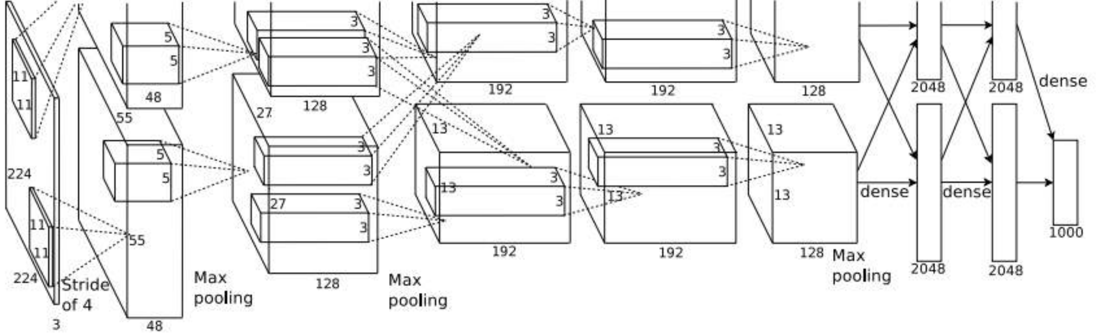
AlexNet



	Input size		Layer				Output size	
Layer	C	H / W	filters	kernel	stride	pad	C	H / W
conv1	3	227	64	11	4	2	64	?

Recall: Output channels = number of filters

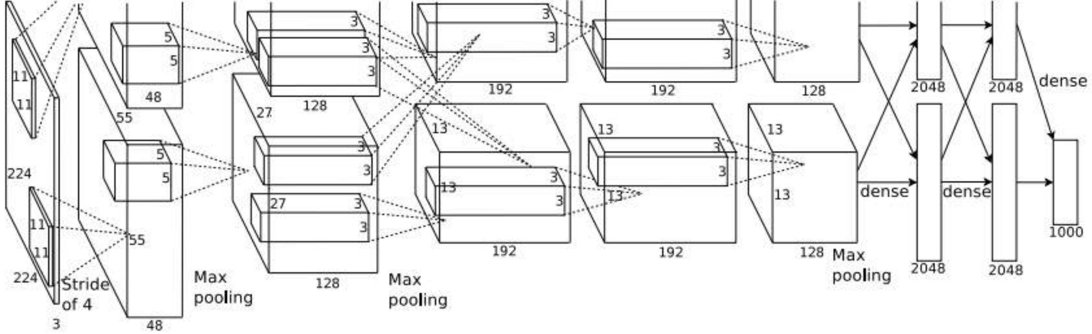
AlexNet



	Input size		Layer				Output size	
Layer	C	H / W	filters	kernel	stride	pad	C	H / W
conv1	3	227	64	11	4	2	64	56

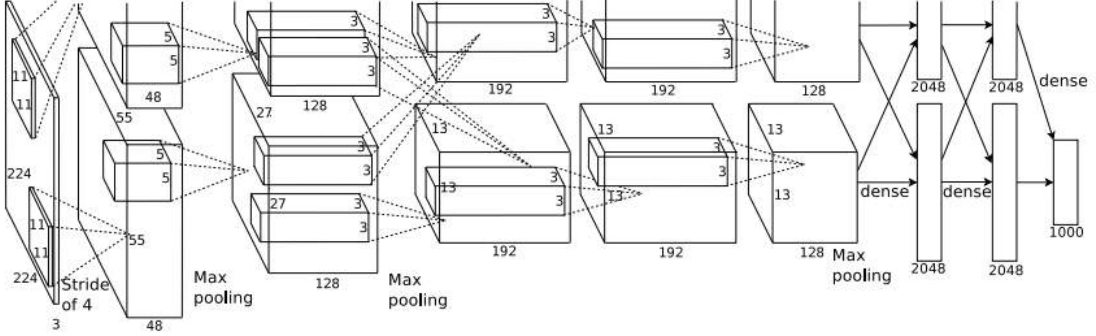
Recall: W' = (W – K + 2P) / S + 1
= 227 – 11 + 2*2) / 4 + 1
= 220/4 + 1 = 56

AlexNet



	Input size		Layer				Output size		
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)
conv1	3	227	64	11	4	2	64	56	?

AlexNet



	Input size		Layer				Output size		
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)
conv1	3	227	64	11	4	2	64	56	784

Number of output elements = C * H’ * W’

= 64*56*56 = 200,704

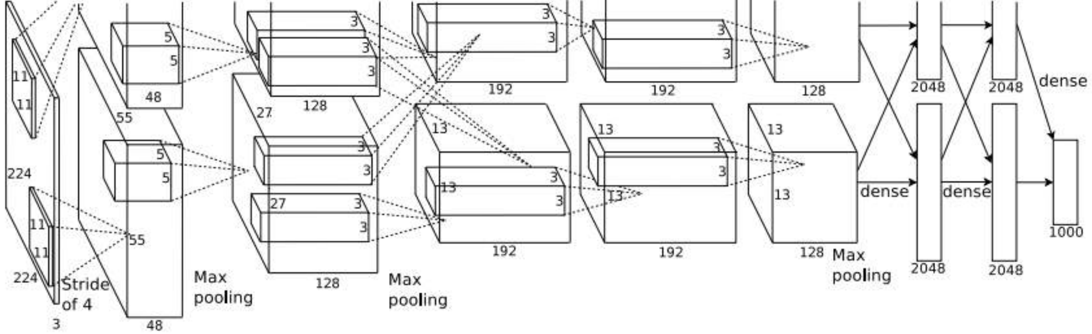
Bytes per element = 4 (for 32-bit floating point)

KB = (number of elements) * (bytes per elem) / 1024

= 200704 * 4 / 1024

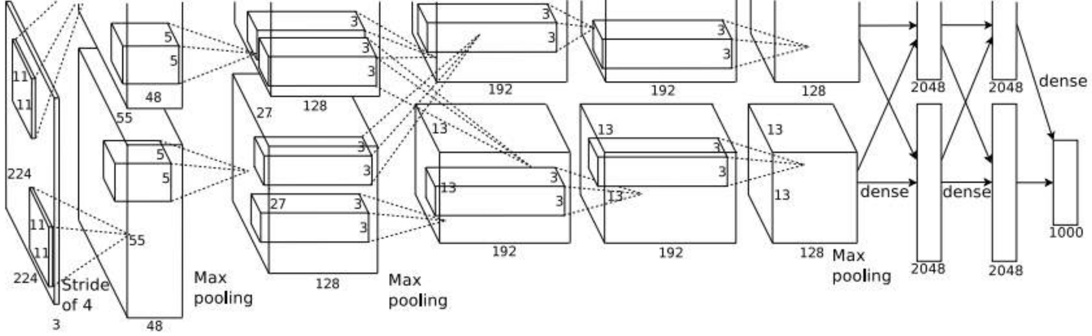
= **784**

AlexNet



	Input size		Layer				Output size			
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)
conv1	3	227	64	11	4	2	64	56	784	?

AlexNet



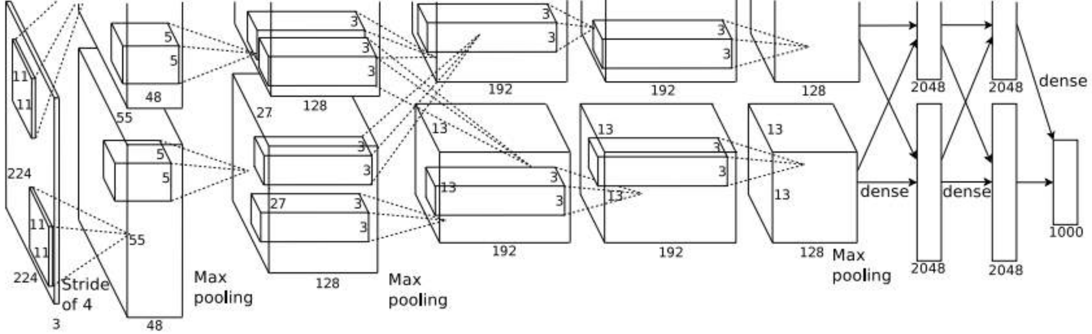
	Input size		Layer				Output size			
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)
conv1	3	227	64	11	4	2	64	56	784	23

Weight shape = C_{out} x C_{in} x K x K
= 64 x 3 x 11 x 11

Bias shape = C_{out} = 64

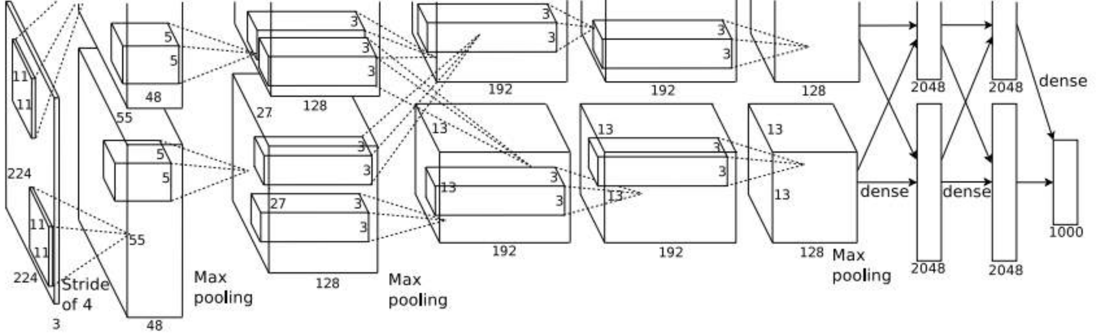
Number of weights = 64*3*11*11 + 64
= **23,296**

AlexNet



	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	?

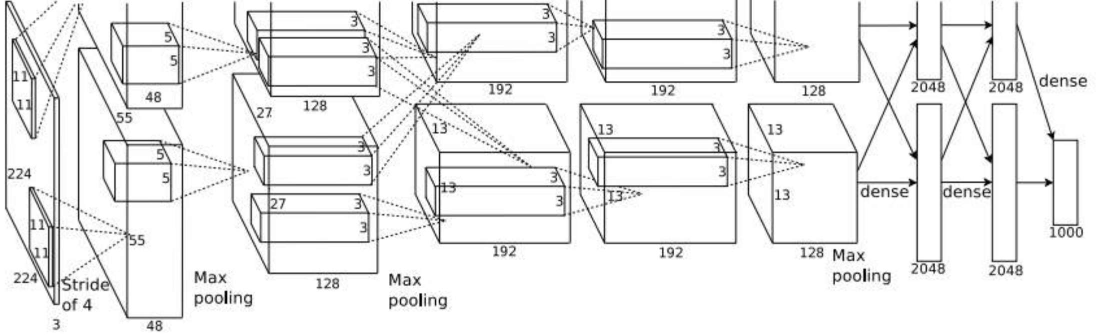
AlexNet



	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73

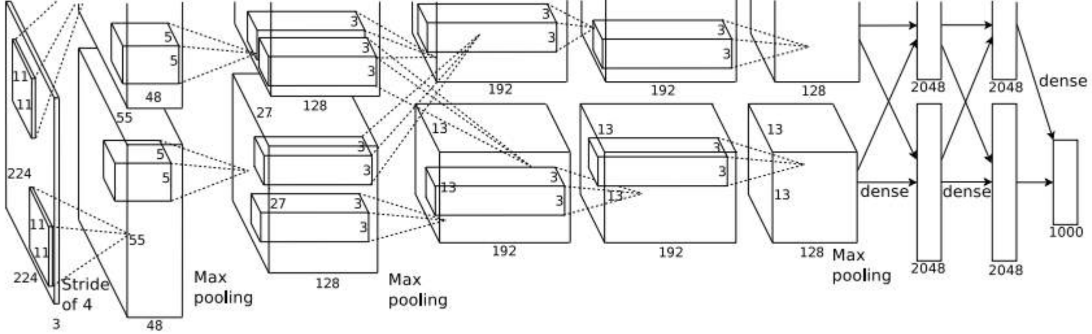
Number of floating point operations (multiply+add)
= (number of output elements) * (ops per output elem)
= $(C_{out} \times H' \times W') * (C_{in} \times K \times K)$
= (64 * 56 * 56) * (3 * 11 * 11)
= 200,704 * 363
= **72,855,552**

AlexNet



	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0		?			

AlexNet



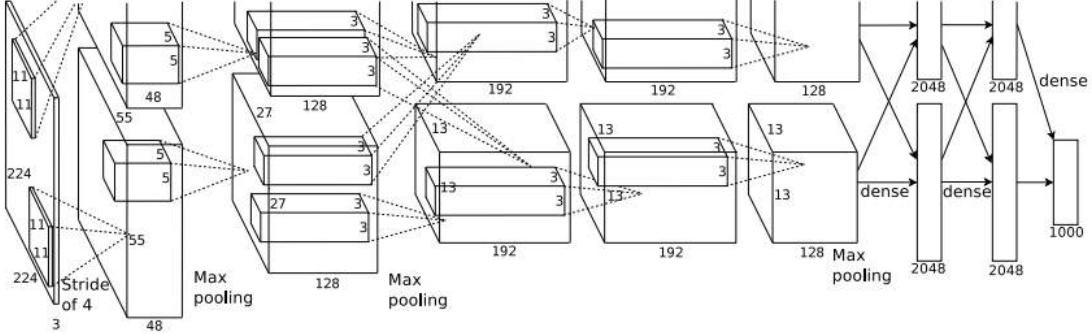
	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27			

For pooling layer:

#output channels = #input channels = 64

$$W' = \text{floor}((W - K) / S + 1)$$
$$= \text{floor}(53 / 2 + 1) = \text{floor}(27.5) = \mathbf{27}$$

AlexNet



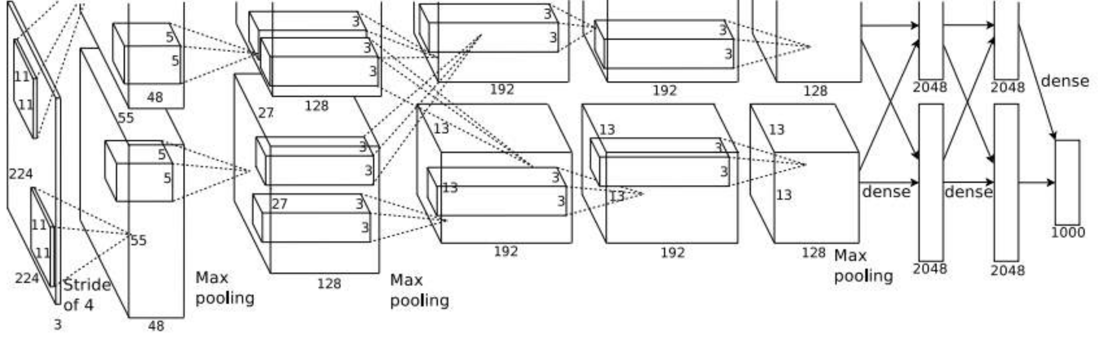
	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	?	

#output elems = C_{out} x H' x W'

Bytes per elem = 4

$$KB = C_{out} * H' * W' * 4 / 1024$$
$$= 64 * 27 * 27 * 4 / 1024$$
$$= \mathbf{182.25}$$

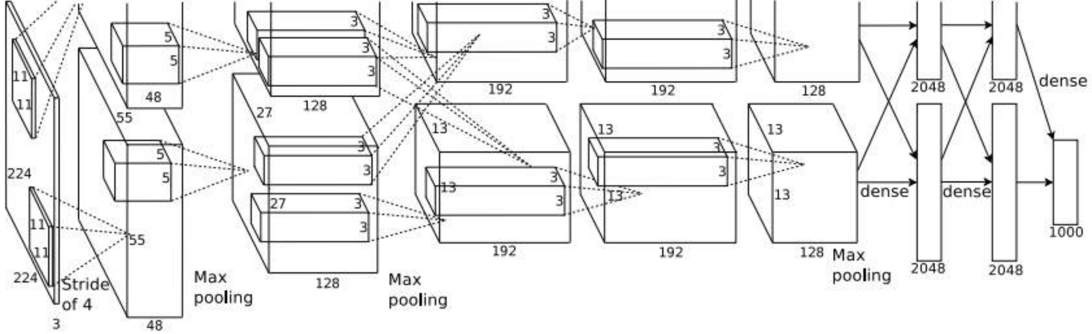
AlexNet



	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	?

Pooling layers have no learnable parameters!

AlexNet



	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0

Floating-point ops for pooling layer

= (number of output positions) * (flops per output position)

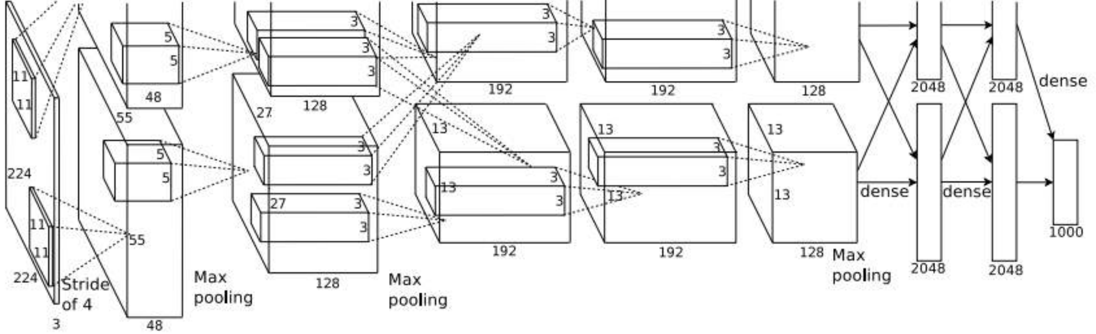
= (C_{out} * H’ * W’) * (K * K)

= (64 * 27 * 27) * (3 * 3)

= 419,904

= **0.4 MFLOP**

AlexNet



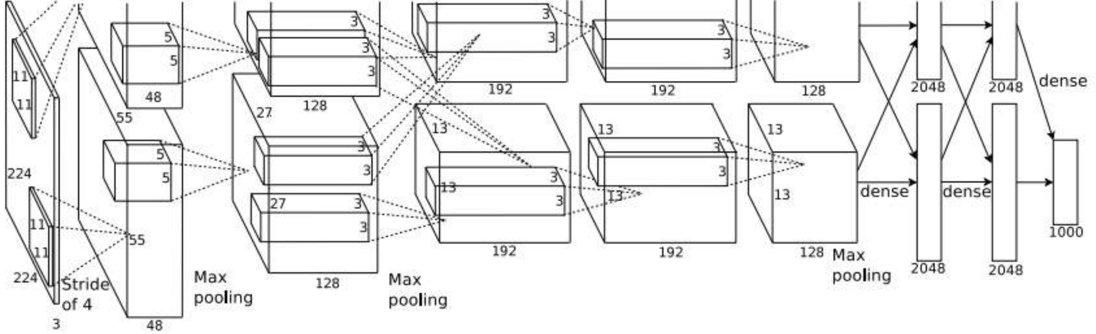
	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664	112
conv4	384	13	256	3	1	1	256	13	169	885	145
conv5	256	13	256	3	1	1	256	13	169	590	100
pool5	256	13		3	2	0	256	6	36	0	0
flatten	256	6					9216		36	0	0

Flatten output size = C_{in} x H x W

= 256 * 6 * 6

= **9216**

AlexNet

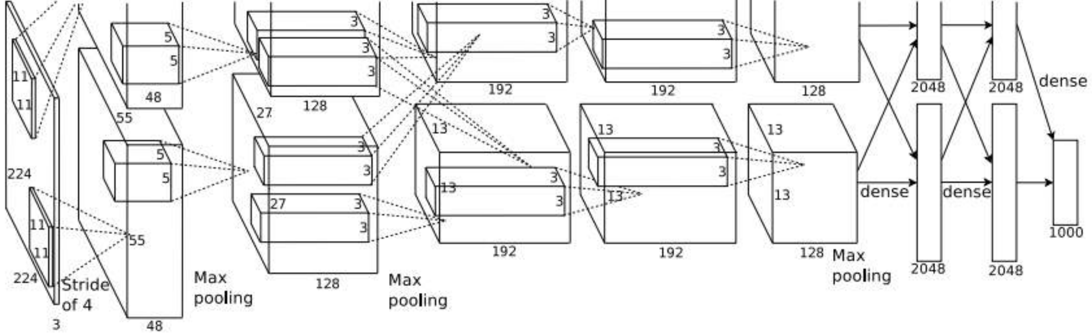


	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664	112
conv4	384	13	256	3	1	1	256	13	169	885	145
conv5	256	13	256	3	1	1	256	13	169	590	100
pool5	256	13		3	2	0	256	6	36	0	0
flatten	256	6					9216		36	0	0
fc6	9216		4096				4096		16	37,749	38

FC params = $C_{in} * C_{out} + C_{out}$
= 9216 * 4096 + 4096
= 37,725,832

FC flops = $C_{in} * C_{out}$
= 9216 * 4096
= 37,748,736

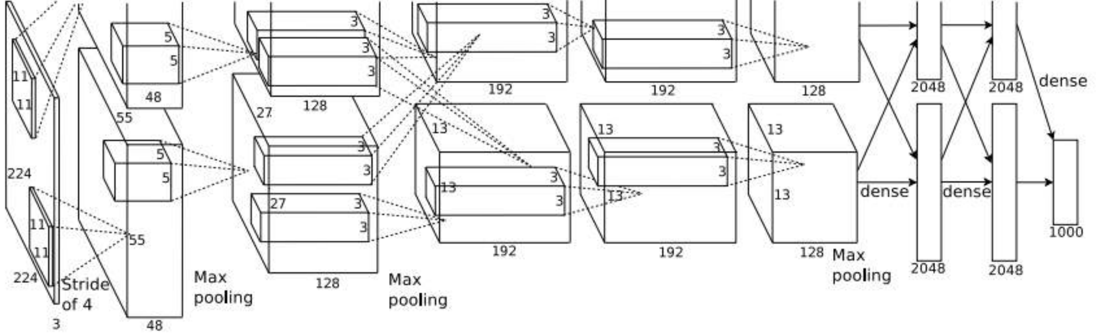
AlexNet



	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664	112
conv4	384	13	256	3	1	1	256	13	169	885	145
conv5	256	13	256	3	1	1	256	13	169	590	100
pool5	256	13		3	2	0	256	6	36	0	0
flatten	256	6					9216		36	0	0
fc6	9216		4096				4096		16	37,749	38
fc7	4096		4096				4096		16	16,777	17
fc8	4096		1000				1000		4	4,096	4

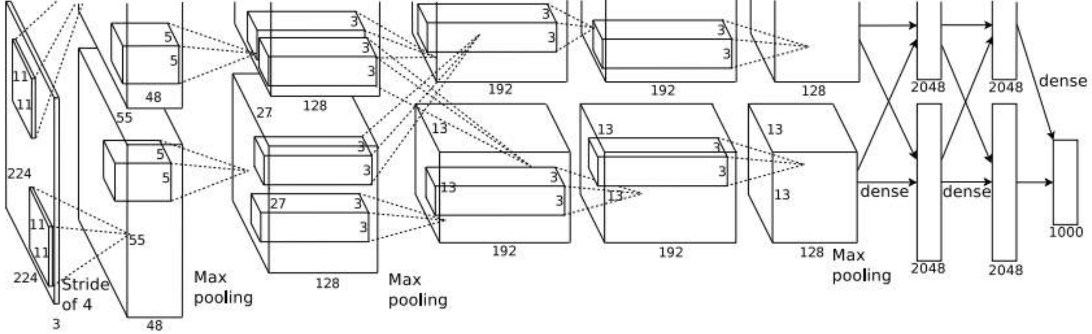
AlexNet

How to choose this?
Trial and error =(



	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664	112
conv4	384	13	256	3	1	1	256	13	169	885	145
conv5	256	13	256	3	1	1	256	13	169	590	100
pool5	256	13		3	2	0	256	6	36	0	0
flatten	256	6					9216		36	0	0
fc6	9216		4096				4096		16	37,749	38
fc7	4096		4096				4096		16	16,777	17
fc8	4096		1000				1000		4	4,096	4

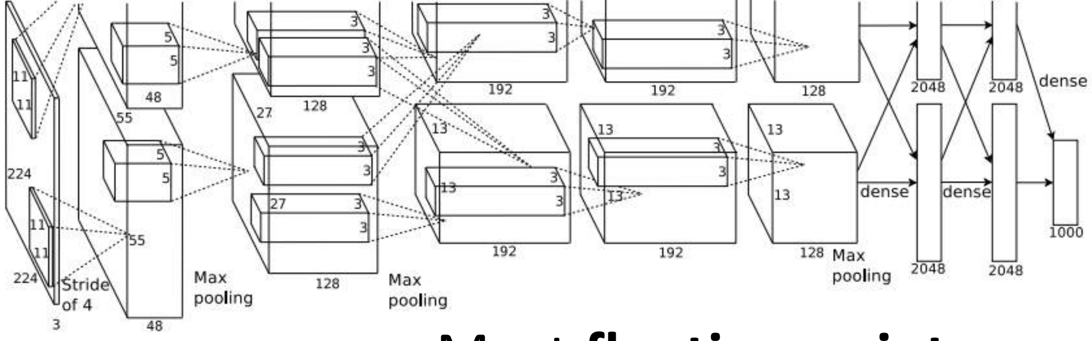
AlexNet



Interesting trends here!

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664	112
conv4	384	13	256	3	1	1	256	13	169	885	145
conv5	256	13	256	3	1	1	256	13	169	590	100
pool5	256	13		3	2	0	256	6	36	0	0
flatten	256	6					9216		36	0	0
fc6	9216		4096				4096		16	37,749	38
fc7	4096		4096				4096		16	16,777	17
fc8	4096		1000				1000		4	4,096	4

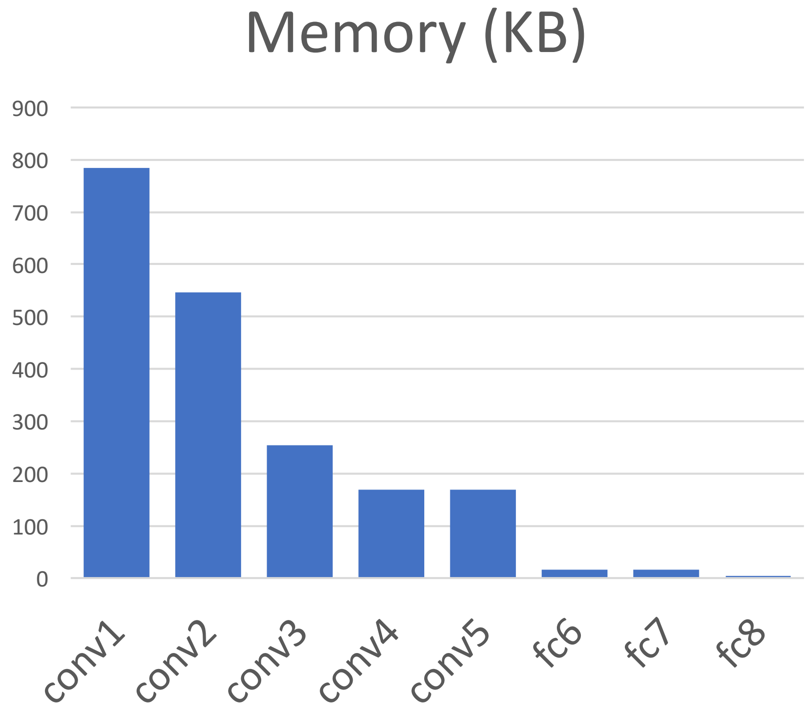
AlexNet

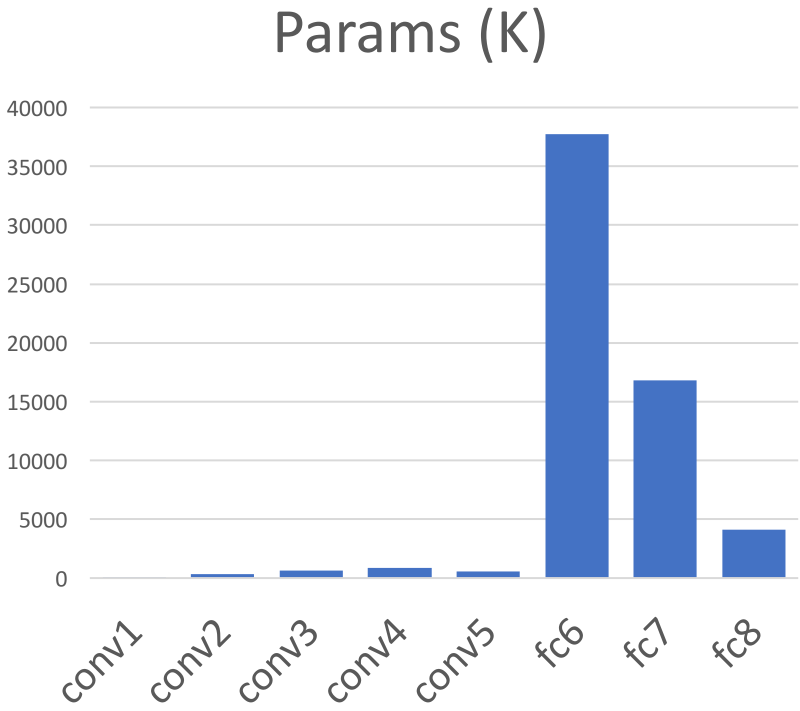


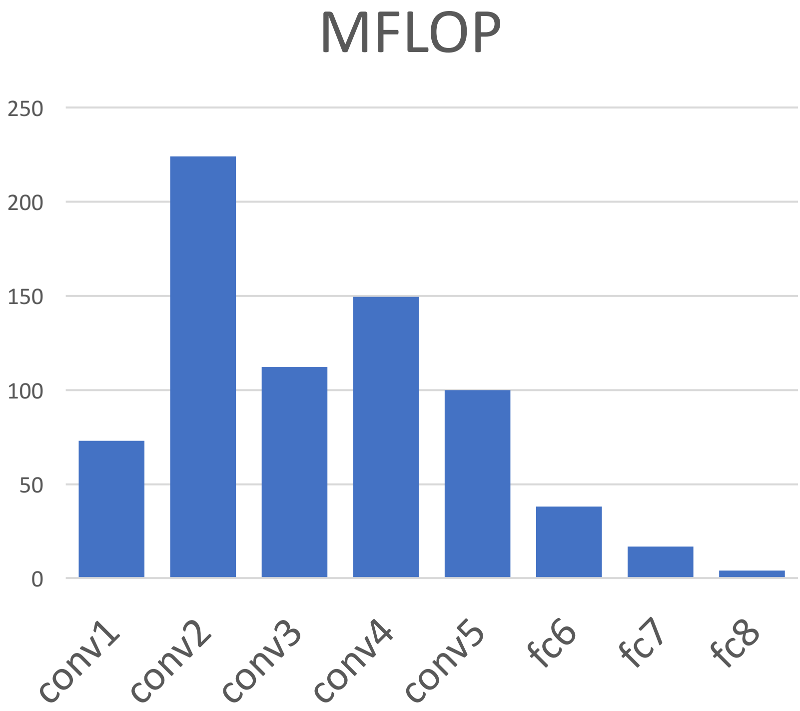
Most of the **memory usage** is in the early convolution layers

Nearly all **parameters** are in the fully-connected layers

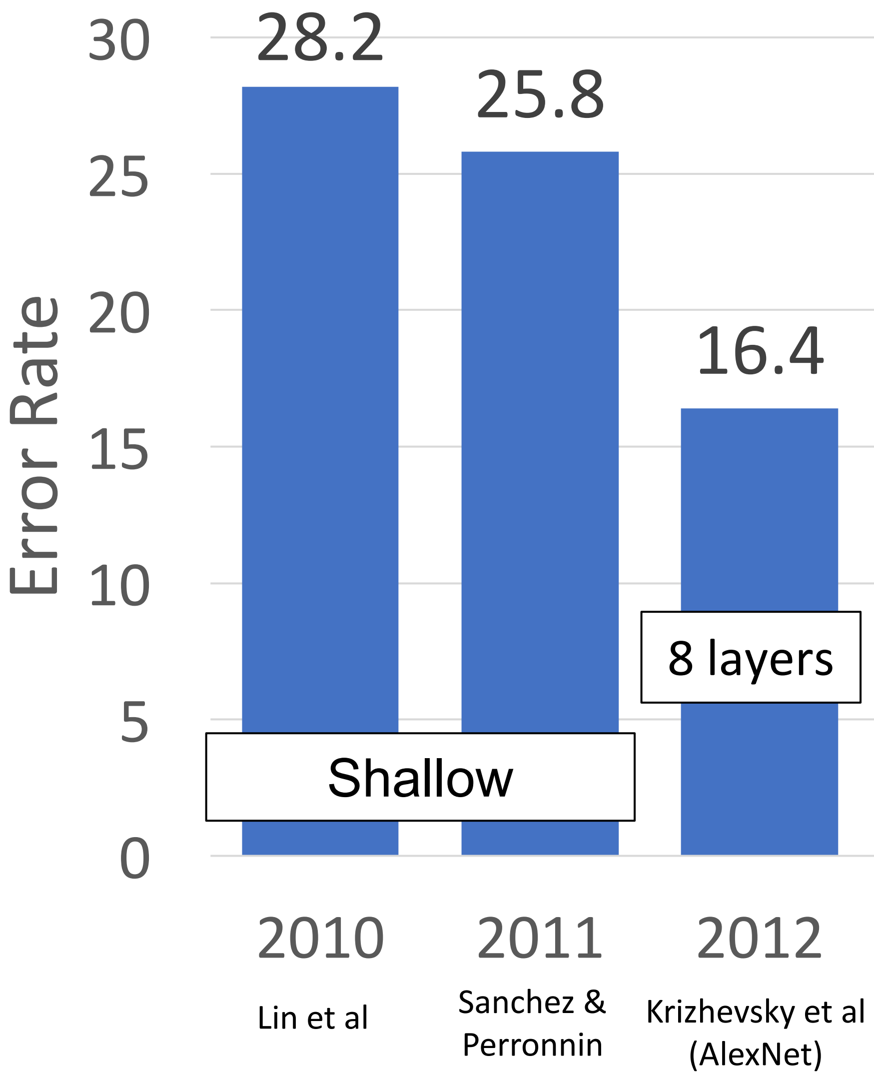
Most **floating-point ops** occur in the convolution layers



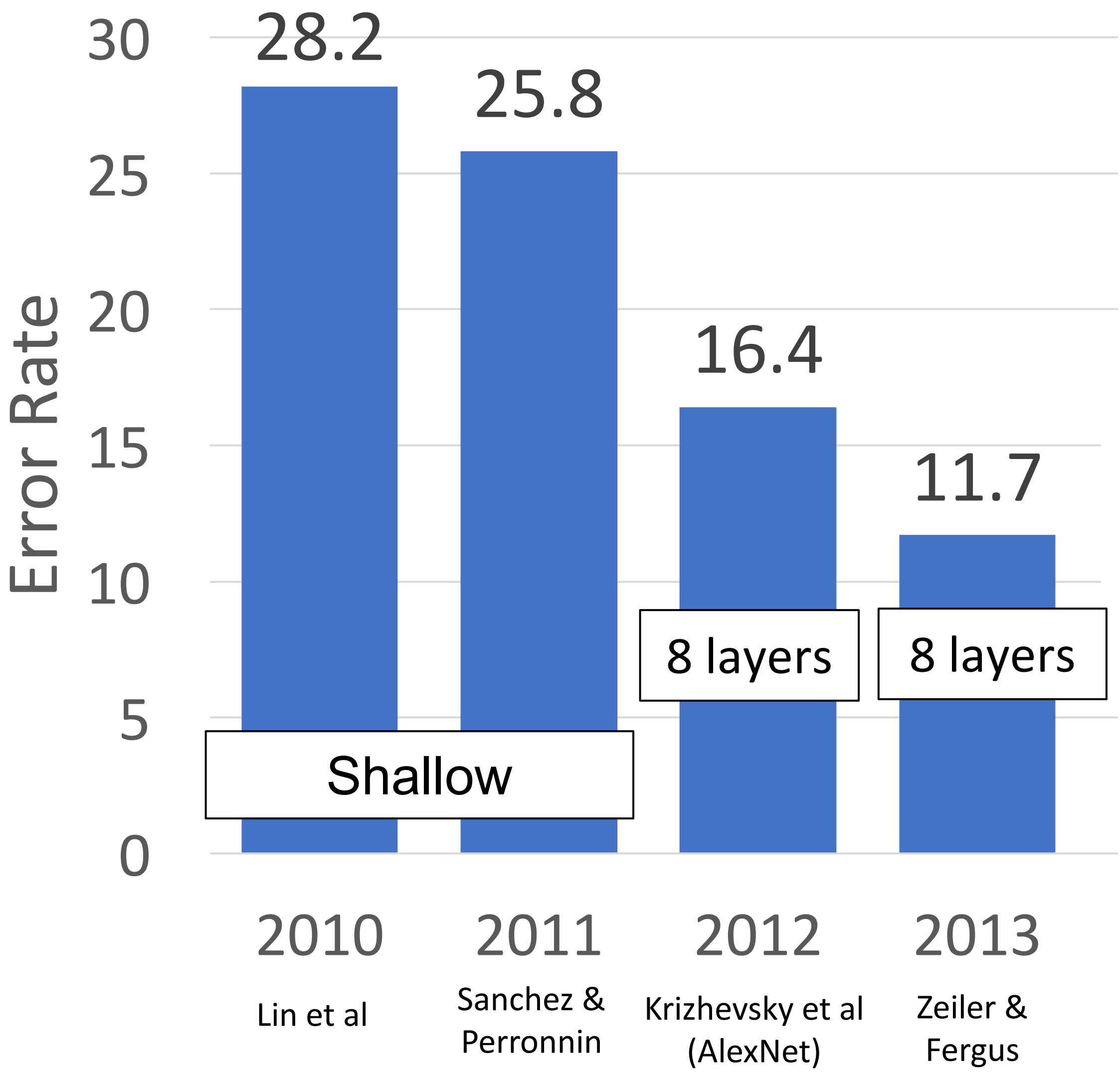




ImageNet Classification Challenge

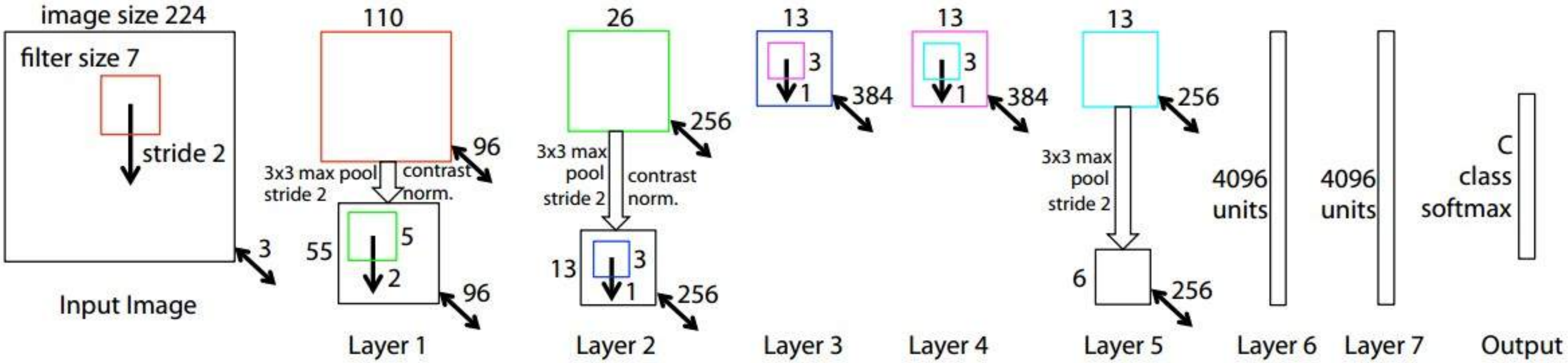


ImageNet Classification Challenge



ZFNet: A Bigger AlexNet

ImageNet top 5 error: 16.4% -> 11.7%



AlexNet but:

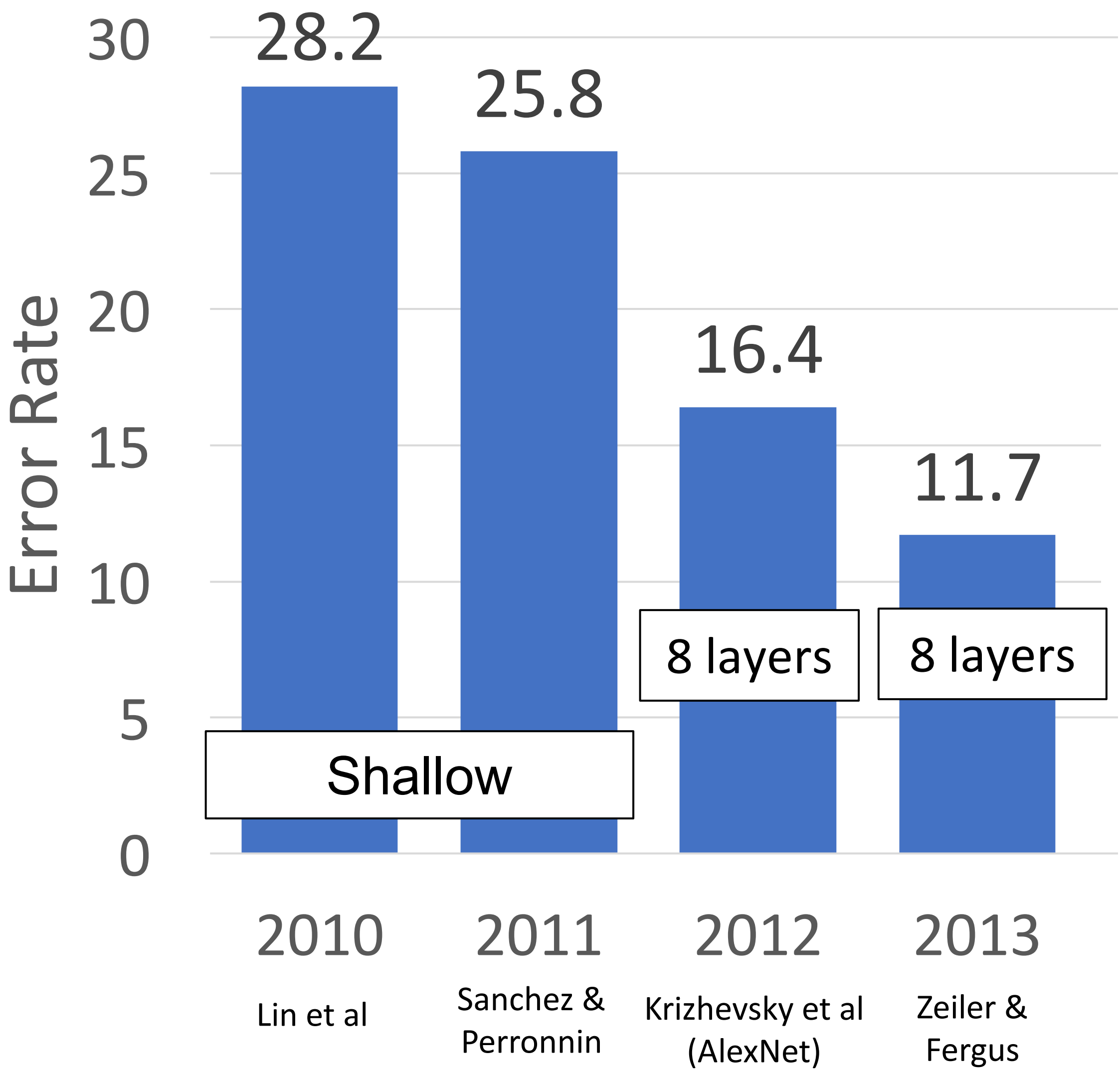
CONV1: change from (11x11 stride 4) to (7x7 stride 2)

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

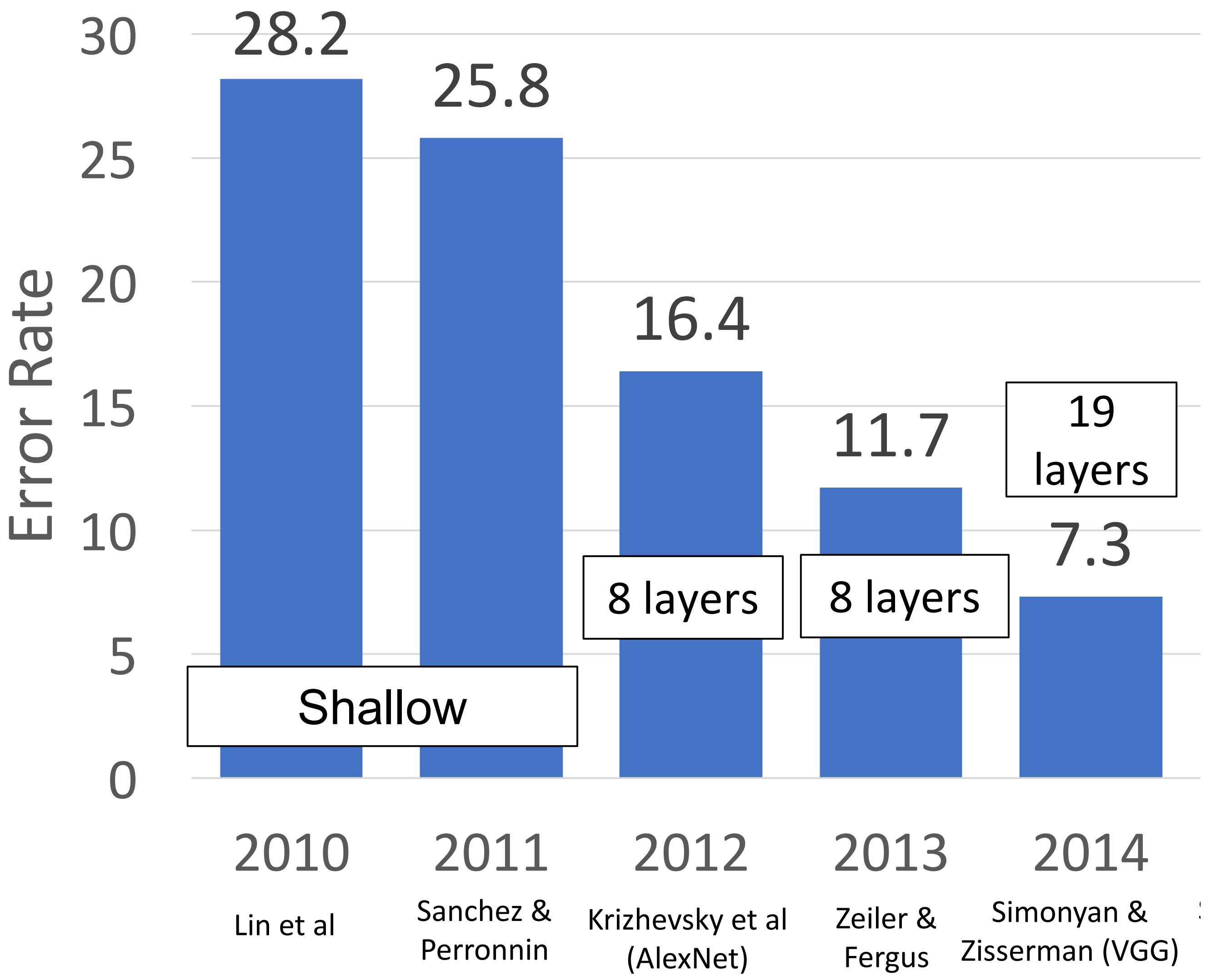
More trial and error =(

Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014

ImageNet Classification Challenge



ImageNet Classification Challenge



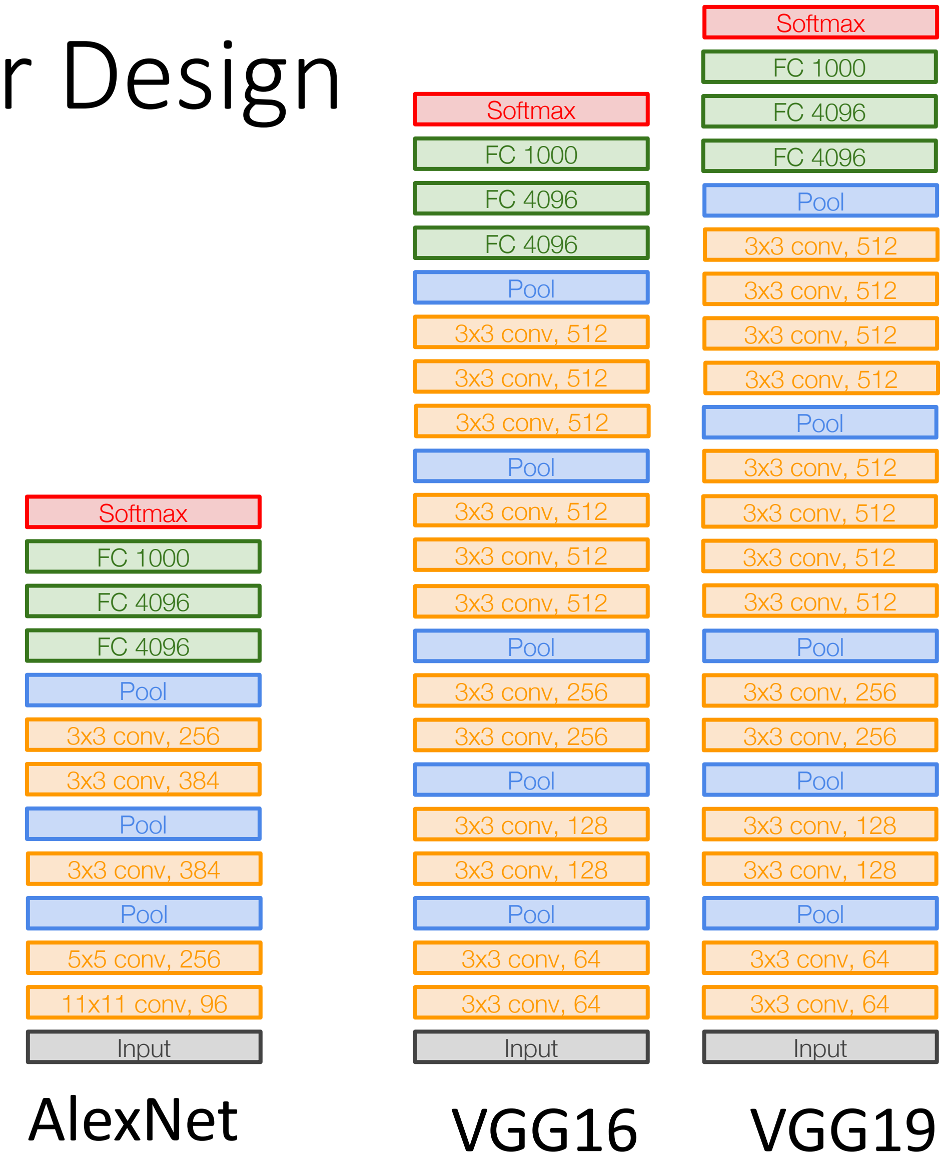
VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels



Simonyan and Zissermann, "Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR 2015

VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

Network has 5 convolutional **stages**:

Stage 1: conv-conv-pool

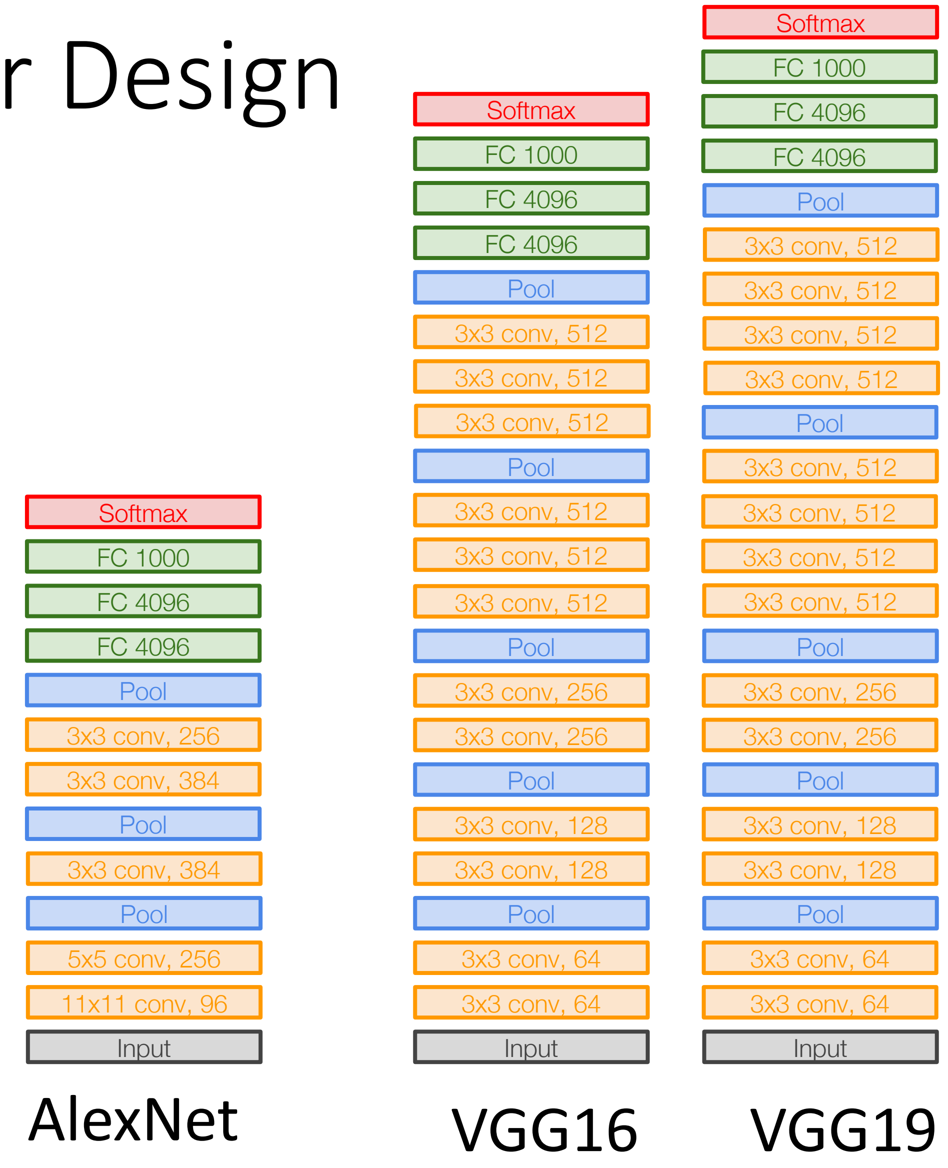
Stage 2: conv-conv-pool

Stage 3: conv-conv-pool

Stage 4: conv-conv-conv-[conv]-pool

Stage 5: conv-conv-conv-[conv]-pool

(VGG-19 has 4 conv in stages 4 and 5)



Simonyan and Zissermann, “Very Deep Convolutional Networks for Large-Scale Image Recognition”, ICLR 2015

VGG: Deeper Networks, Regular Design

VGG Design rules:

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

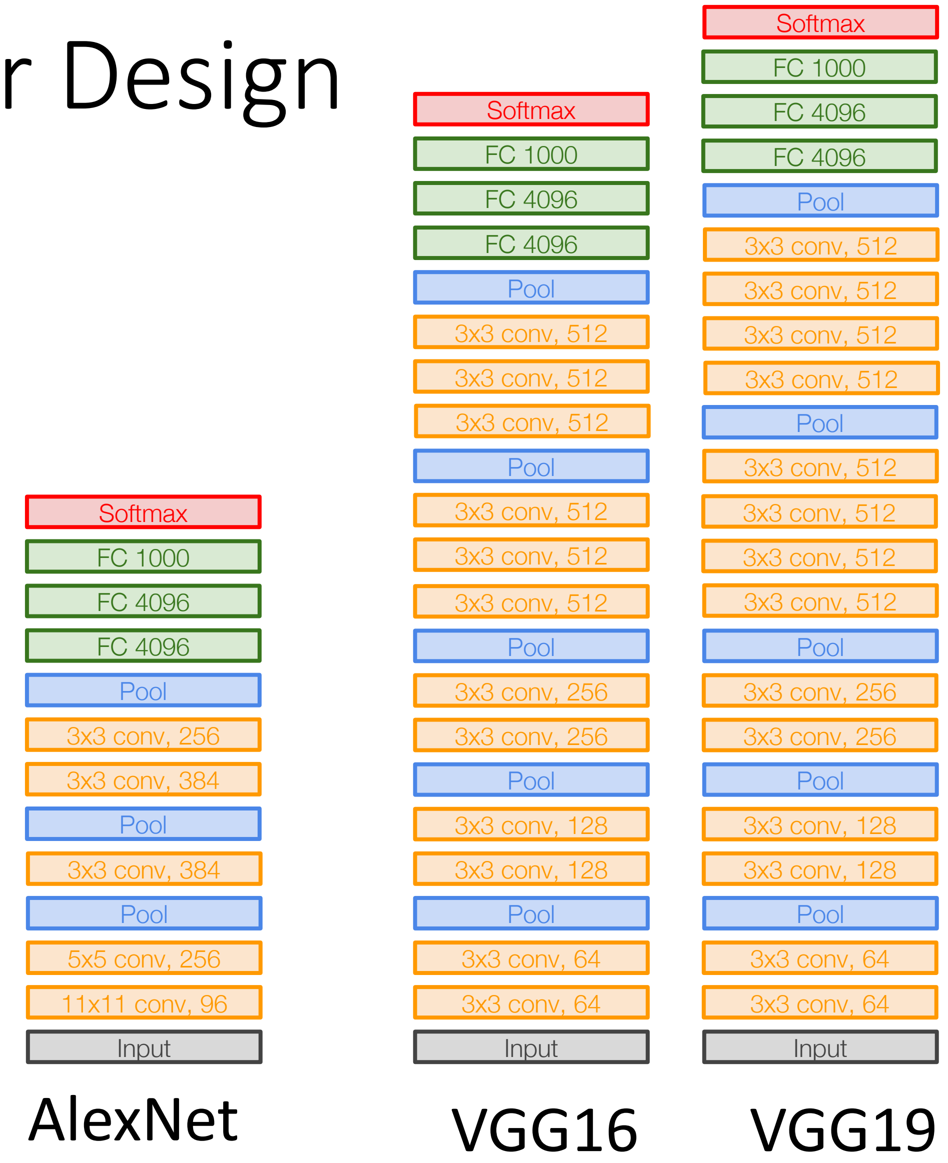
After pool, double #channels

Option 1:

Conv(5x5, C -> C)

Params: $25C^2$

FLOPs: $25C^2HW$



Simonyan and Zissermann, “Very Deep Convolutional Networks for Large-Scale Image Recognition”, ICLR 2015

VGG: Deeper Networks, Regular Design

VGG Design rules:

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

Option 1:

Conv(5x5, C -> C)

Params: $25C^2$

FLOPs: $25C^2HW$

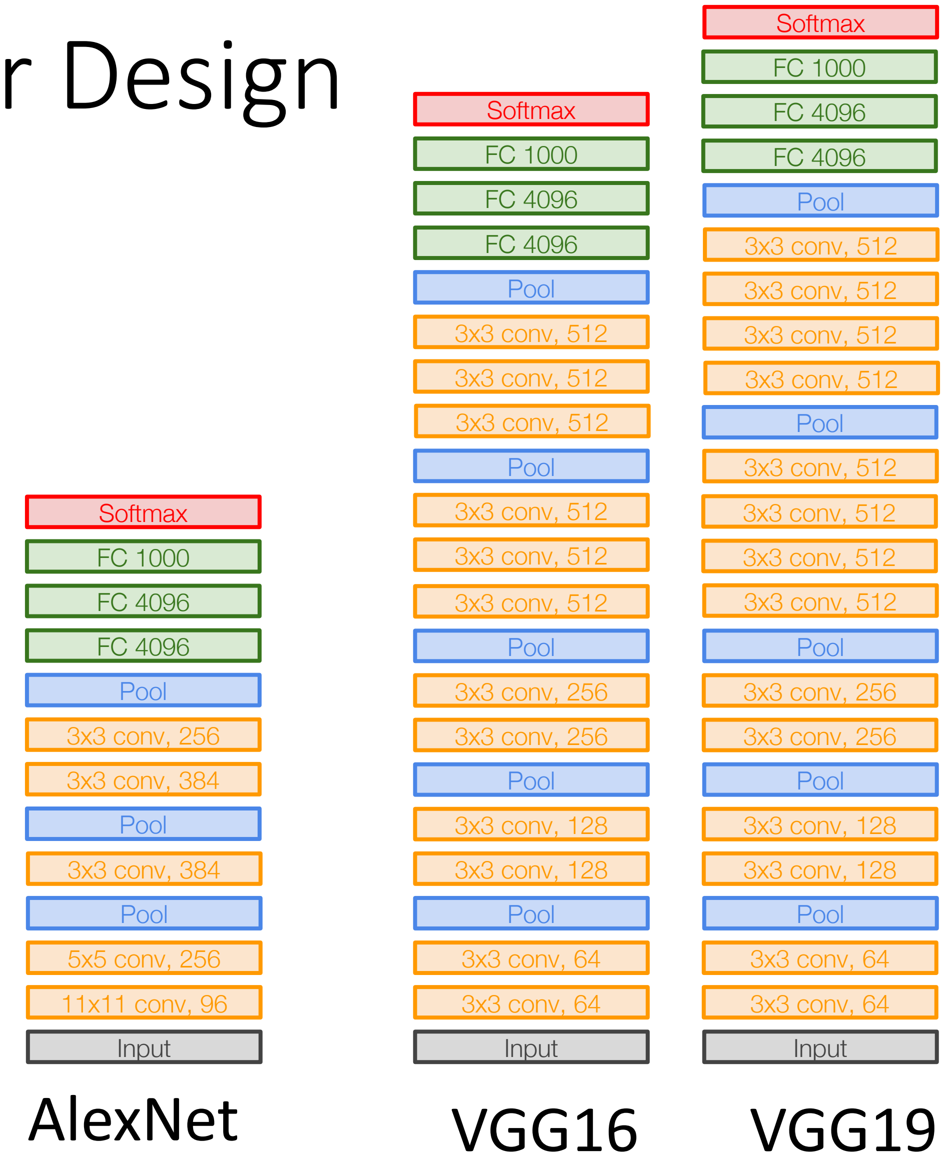
Option 2:

Conv(3x3, C -> C)

Conv(3x3, C -> C)

Params: $18C^2$

FLOPs: $18C^2HW$



Simonyan and Zissermann, “Very Deep Convolutional Networks for Large-Scale Image Recognition”, ICLR 2015

VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

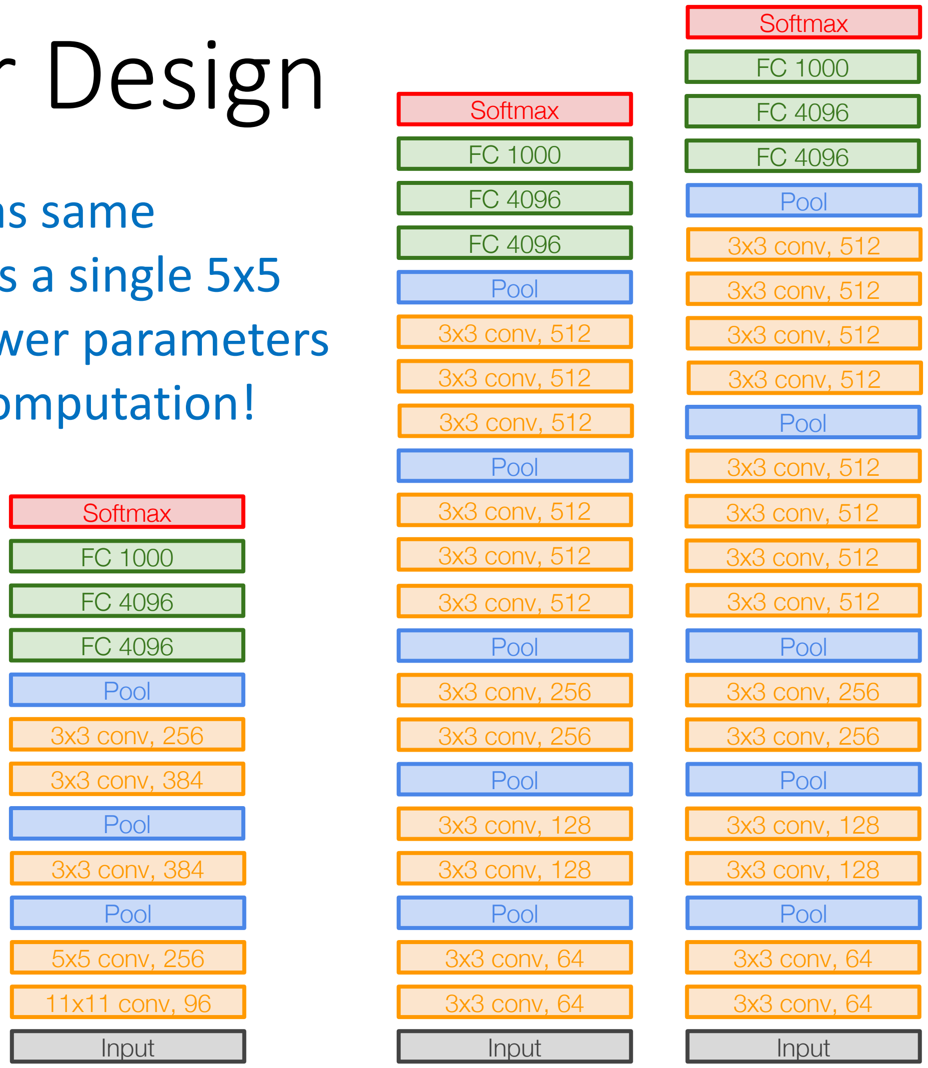
After pool, double #channels

Two 3x3 conv has same receptive field as a single 5x5 conv, but has fewer parameters and takes less computation!

also more non-linearity

Option 1:	Option 2:
Conv(5x5, C -> C)	Conv(3x3, C -> C)
	Conv(3x3, C -> C)
Params: $25C^2$	Params: $18C^2$
FLOPs: $25C^2HW$	FLOPs: $18C^2HW$

ReLU



AlexNet VGG16 VGG19

Simonyan and Zissermann, "Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR 2015

VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

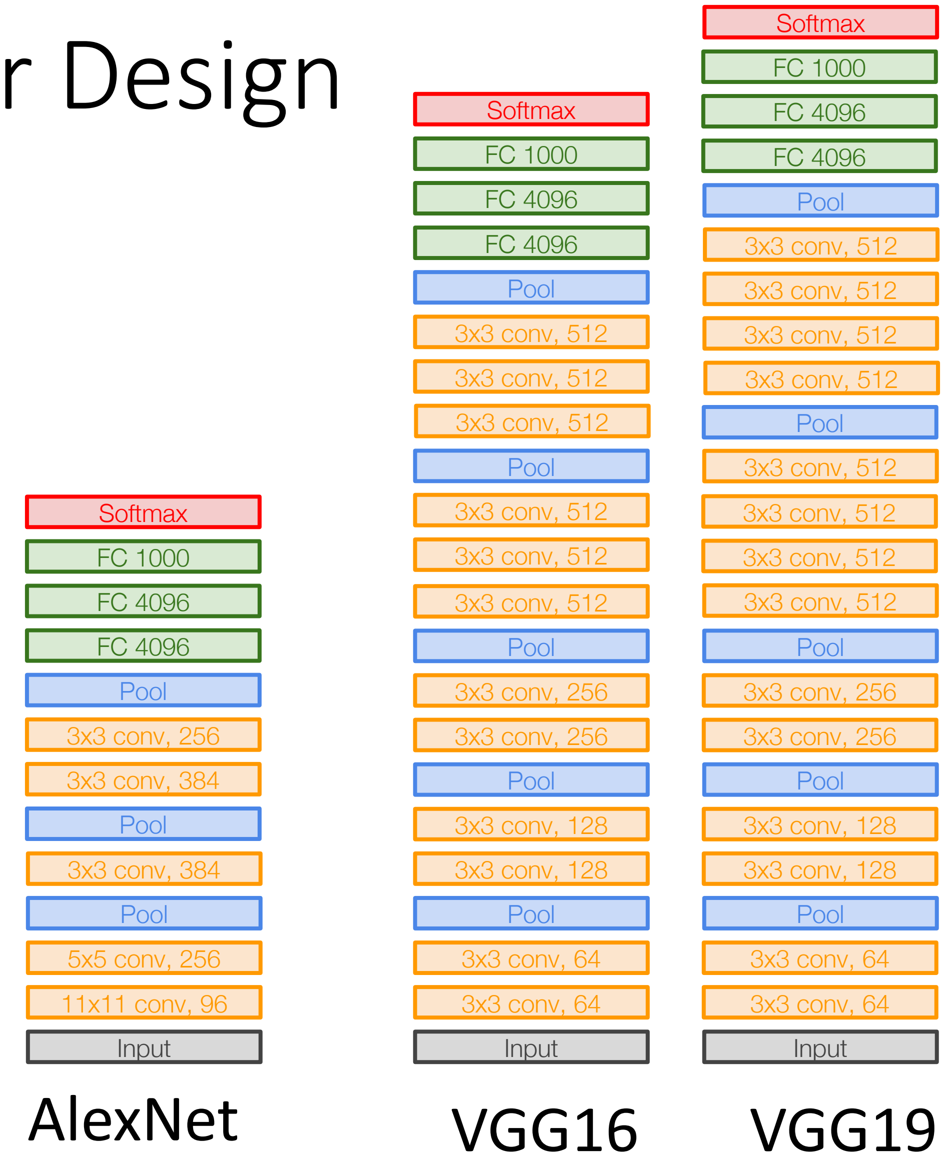
Input: C x 2H x 2W

Layer: Conv(3x3, C->C)

Memory: 4HWC

Params: $9C^2$

FLOPs: $4HWC^2$



Simonyan and Zissermann, “Very Deep Convolutional Networks for Large-Scale Image Recognition”, ICLR 2015

VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

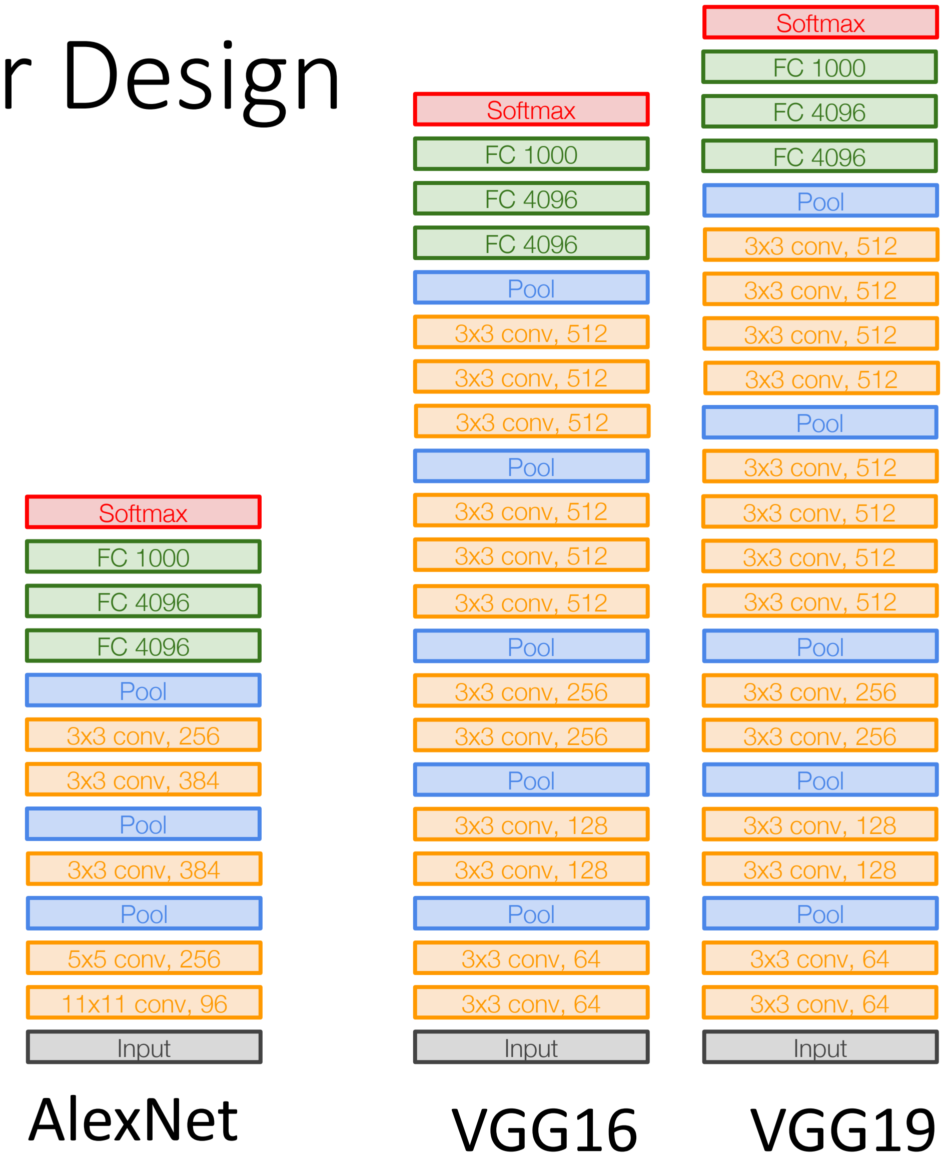
After pool, double #channels

Input: C x 2H x 2W
Layer: Conv(3x3, C->C)

Memory: 4HWC
Params: $9C^2$
FLOPs: $4HWC^2$

Input: 2C x H x W
Conv(3x3, 2C -> 2C)

Memory: 2HWC
Params: $36C^2$
FLOPs: $4HWC^2$



Simonyan and Zissermann, "Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR 2015

VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

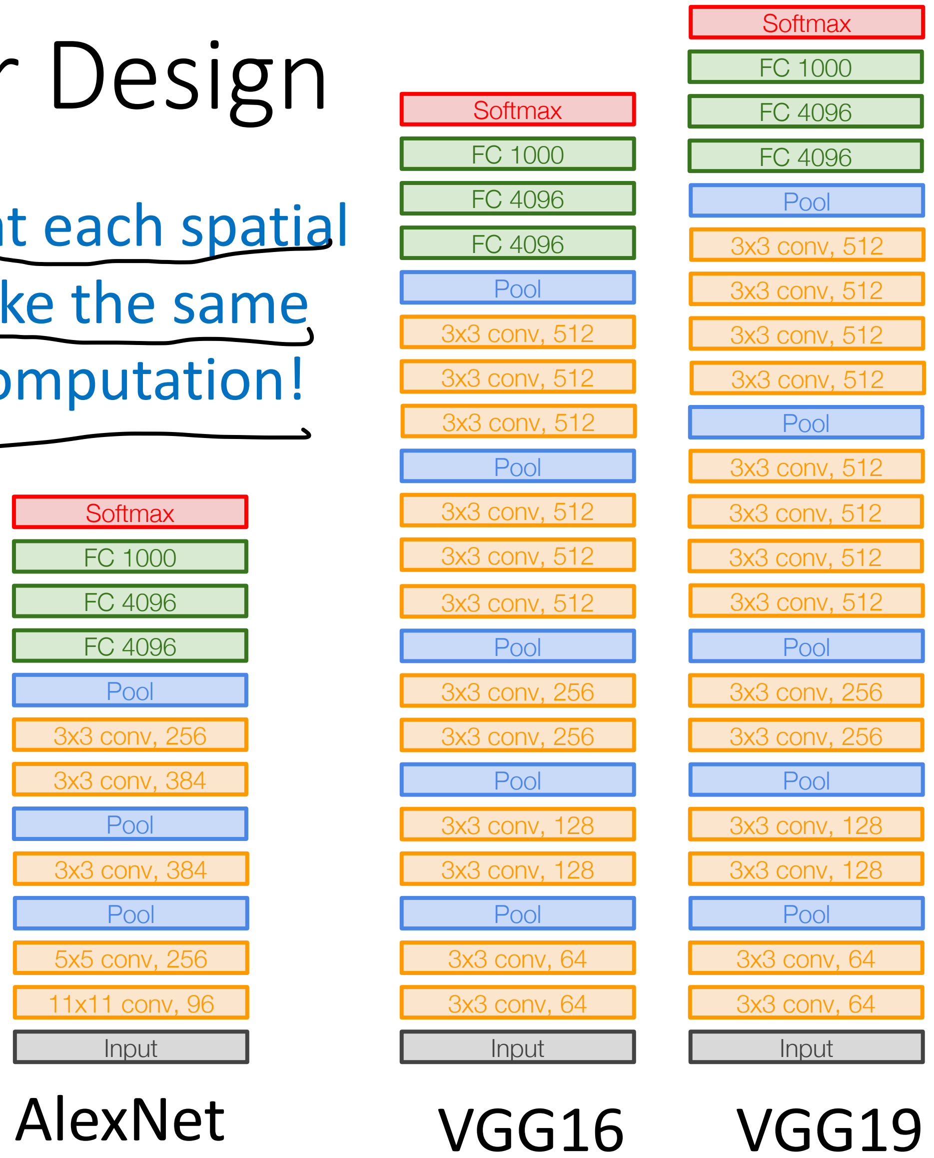
All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

Conv layers at each spatial resolution take the same amount of computation!

Input: C x 2H x 2W	Input: 2C x H x W
Layer: Conv(3x3, C->C)	Conv(3x3, 2C -> 2C)
Memory: 4HWC	Memory: 2HWC
Params: $9C^2$	Params: $36C^2$
FLOPs: $4HWC^2$	FLOPs: $4HWC^2$

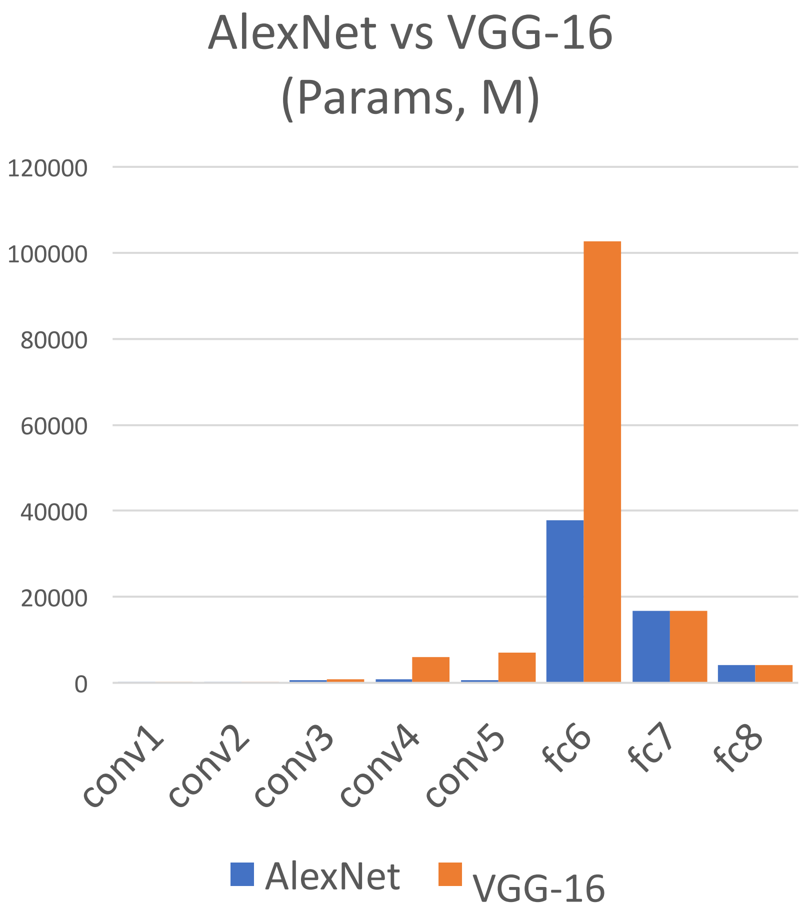


Simonyan and Zissermann, "Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR 2015

AlexNet vs VGG-16: Much bigger network!

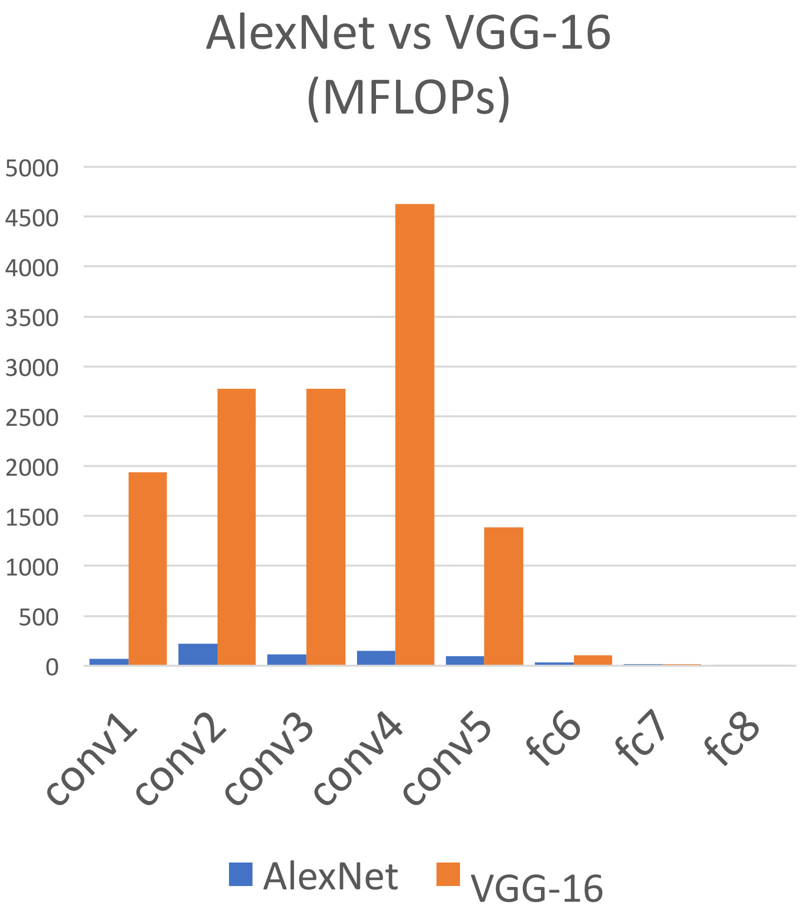
AlexNet total: 1.9 MB

VGG-16 total: 48.6 MB (25x)



AlexNet total: 61M

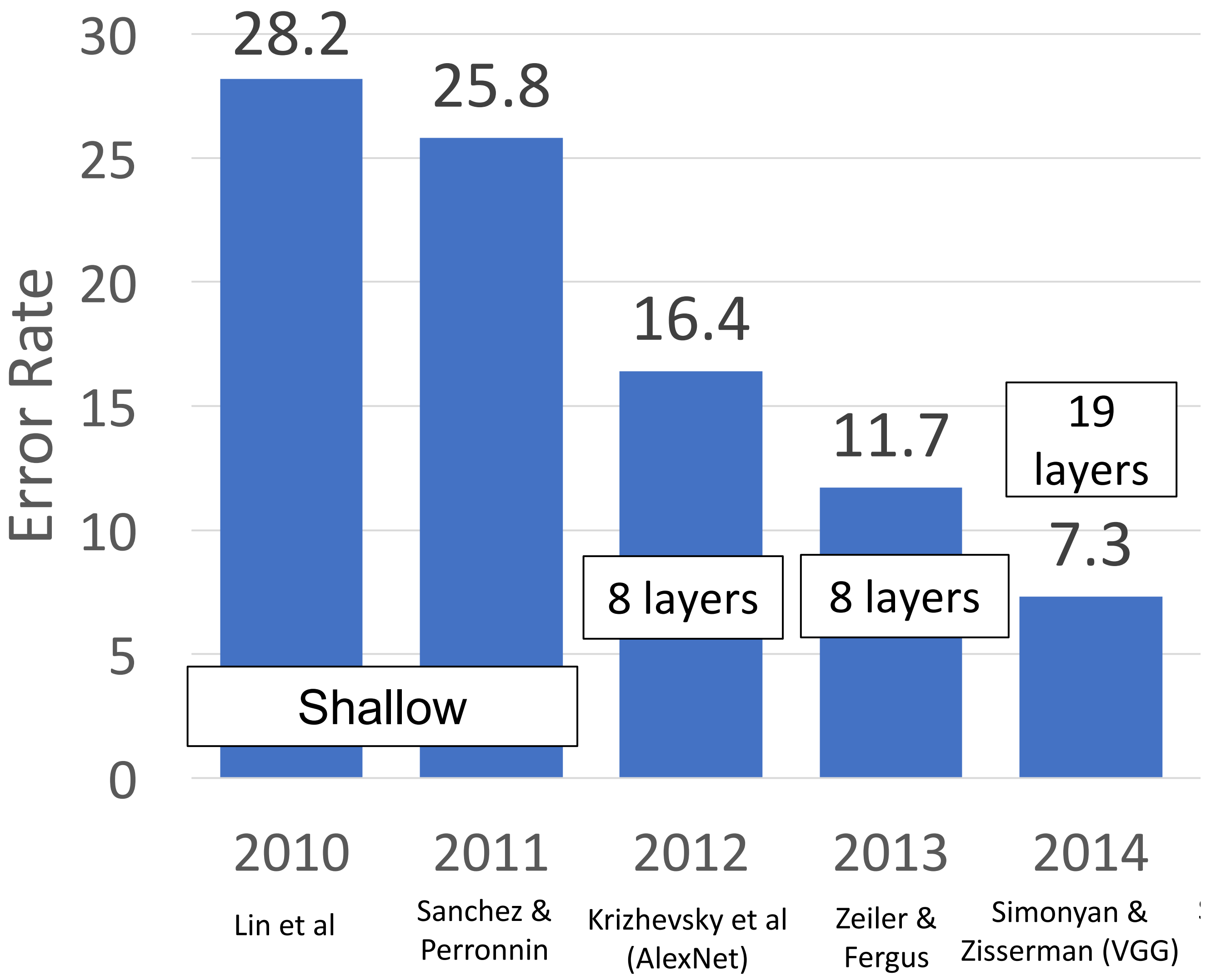
VGG-16 total: 138M (2.3x)

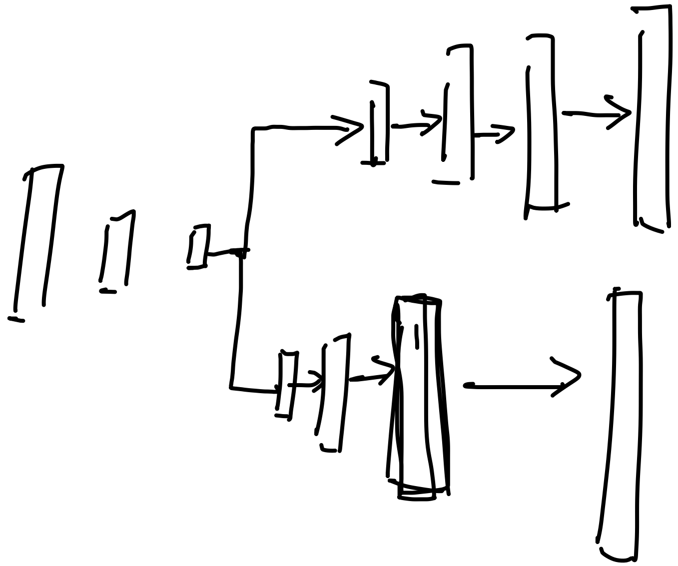


AlexNet total: 0.7 GFLOP

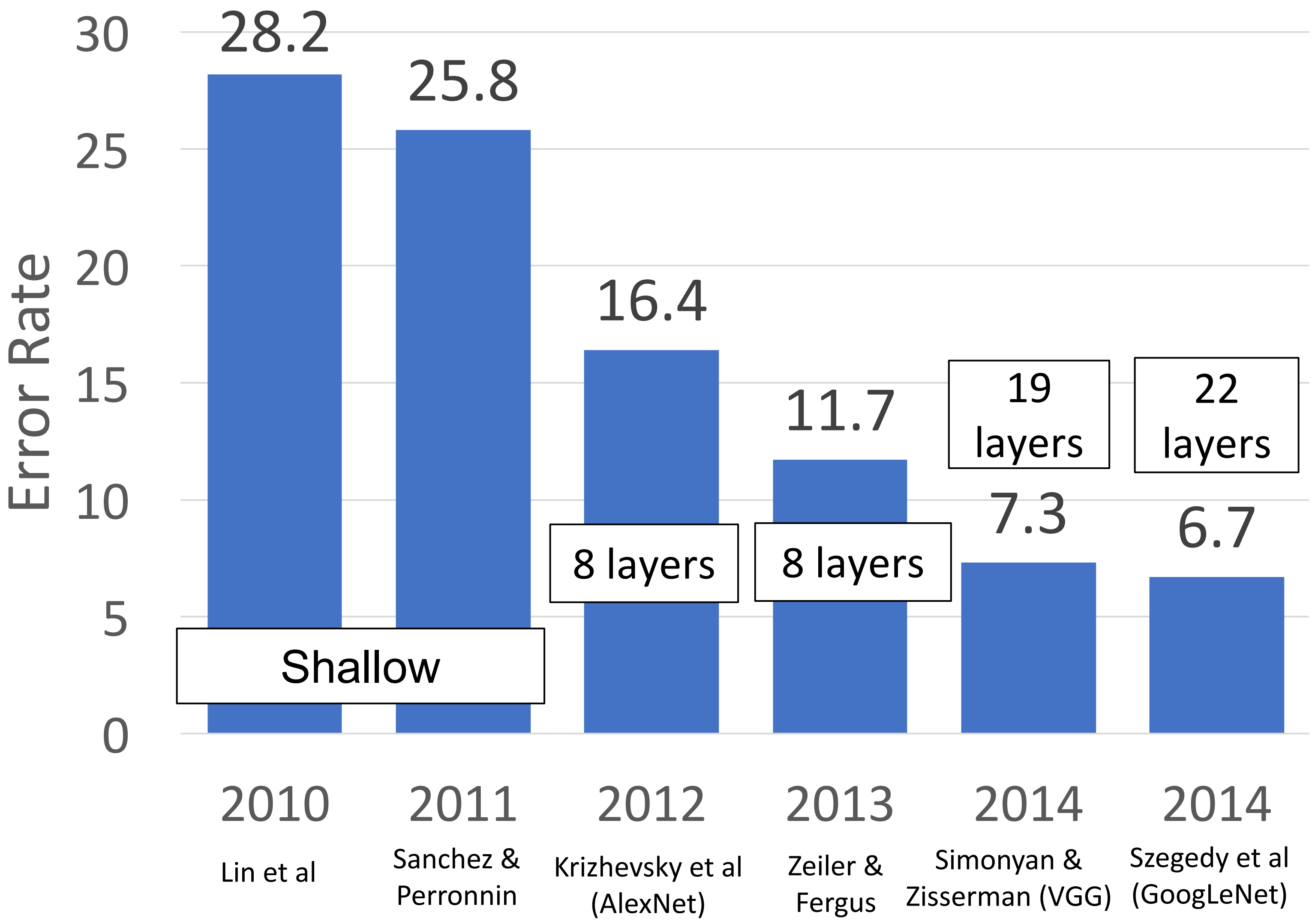
VGG-16 total: 13.6 GFLOP (19.4x)

ImageNet Classification Challenge

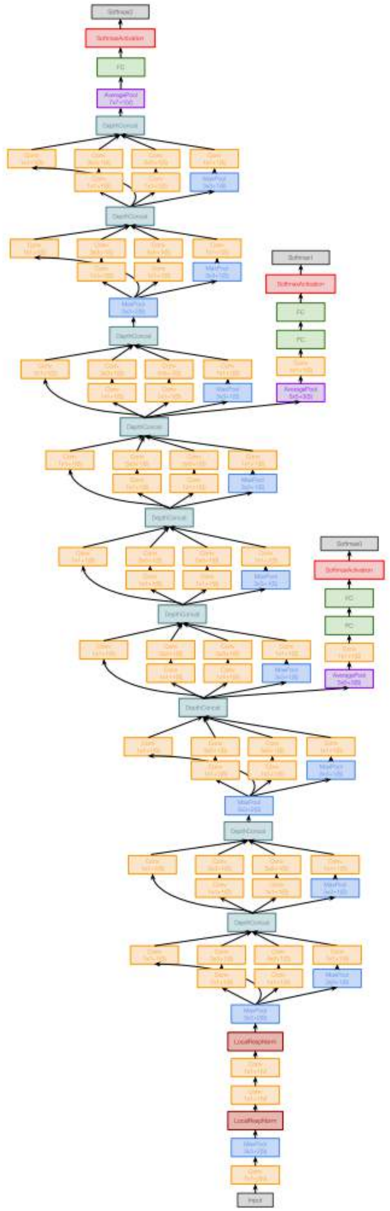




ImageNet Classification Challenge



GoogLeNet: Focus on Efficiency



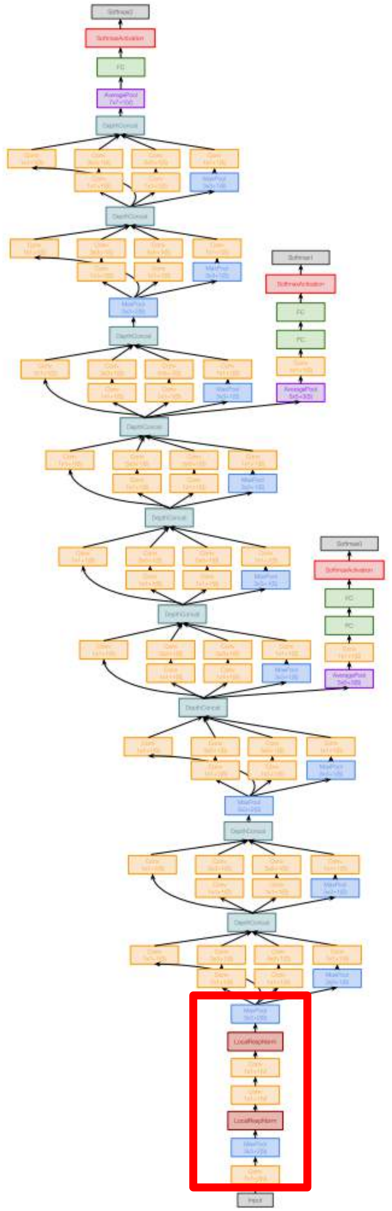
Many innovations for efficiency: reduce parameter count, memory usage, and computation

Szegedy et al, "Going deeper with convolutions", CVPR 2015

GoogLeNet: Aggressive Stem

Stem network at the start aggressively downsamples input
(Recall in VGG-16: Most of the compute was at the start)

very quickly.

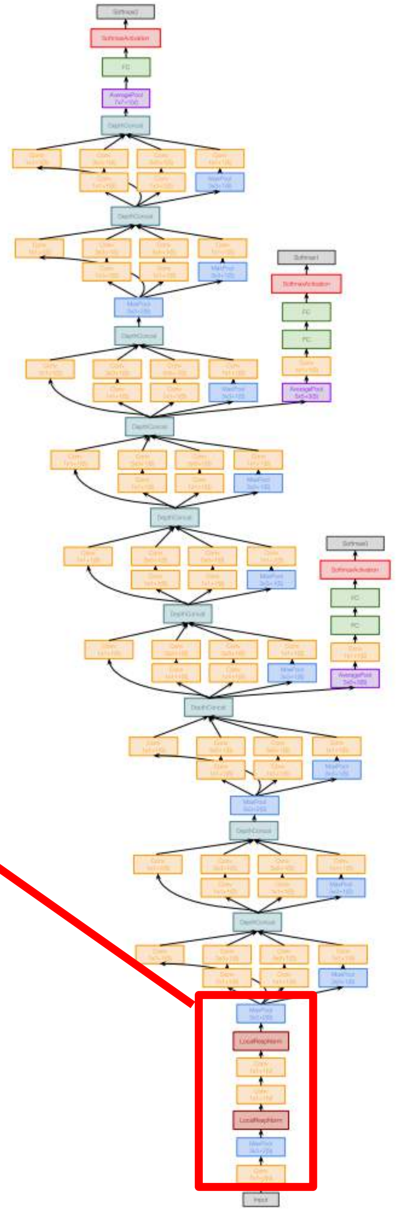


Szegedy et al, "Going deeper with convolutions", CVPR 2015

GoogLeNet: Aggressive Stem

Stem network at the start aggressively downsamples input
(Recall in VGG-16: Most of the compute was at the start)

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (K)	flop (M)
conv	3	224	64	7	2	3	64	112	3136	9	118
max-pool	64	112		3	2	1	64	56	784	0	2
conv	64	56	64	1	1	0	64	56	784	4	13
conv	64	56	192	3	1	1	192	56	2352	111	347
max-pool	192	56		3	2	1	192	28	588	0	1



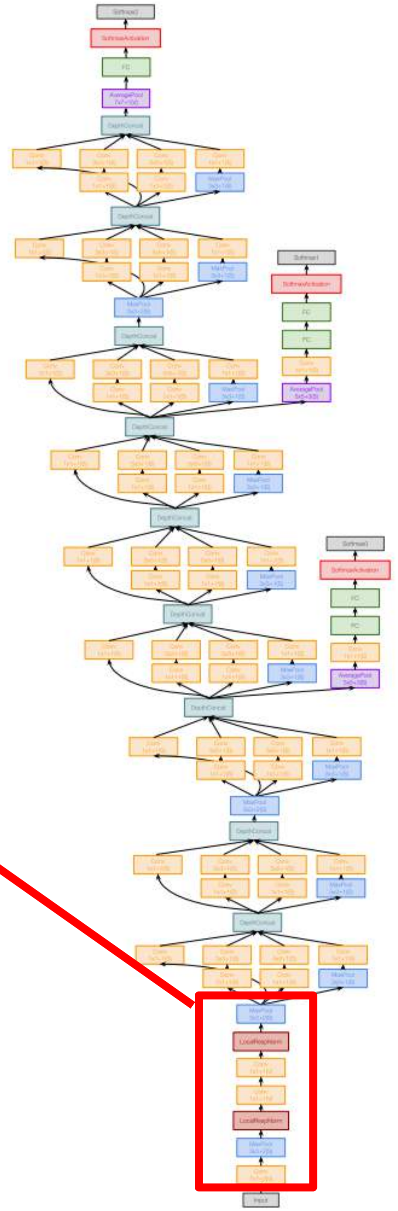
Total from 224 to 28 spatial resolution:
Memory: 7.5 MB
Params: 124K
MFLOP: 418

Szegedy et al, “Going deeper with convolutions”, CVPR 2015

GoogLeNet: Aggressive Stem

Stem network at the start aggressively downsamples input
(Recall in VGG-16: Most of the compute was at the start)

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (K)	flop (M)
conv	3	224	64	7	2	3	64	112	3136	9	118
max-pool	64	112		3	2	1	64	56	784	0	2
conv	64	56	64	1	1	0	64	56	784	4	13
conv	64	56	192	3	1	1	192	56	2352	111	347
max-pool	192	56		3	2	1	192	28	588	0	1



Total from 224 to 28 spatial resolution:
Memory: 7.5 MB
Params: 124K
MFLOP: 418

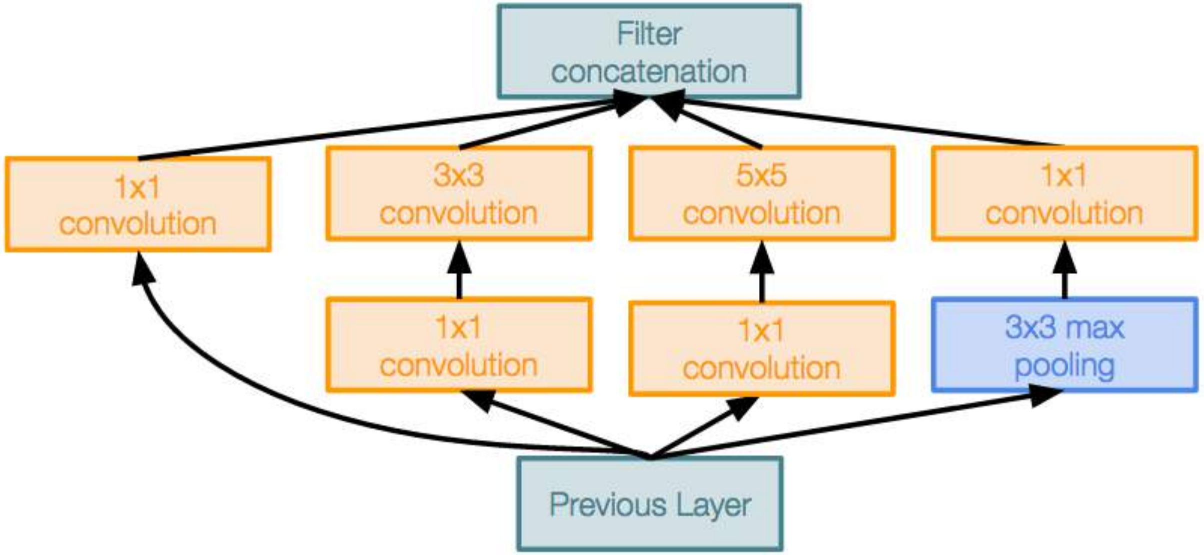
Compare VGG-16:
Memory: 42.9 MB (5.7x)
Params: 1.1M (8.9x)
MFLOP: 7485 (17.8x)

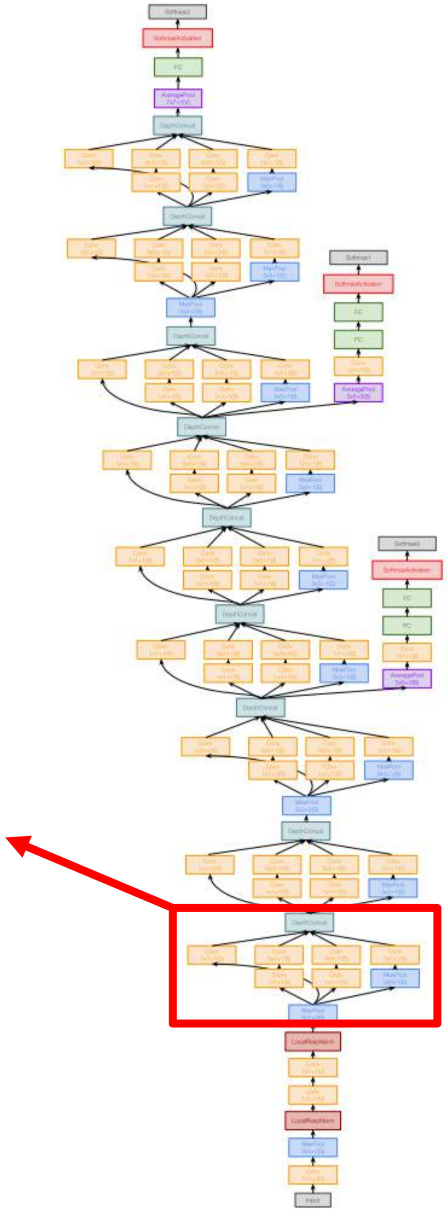
Szegedy et al, "Going deeper with convolutions", CVPR 2015

GoogLeNet: Inception Module

Inception module
Local unit with parallel branches

Local structure repeated many times throughout the network





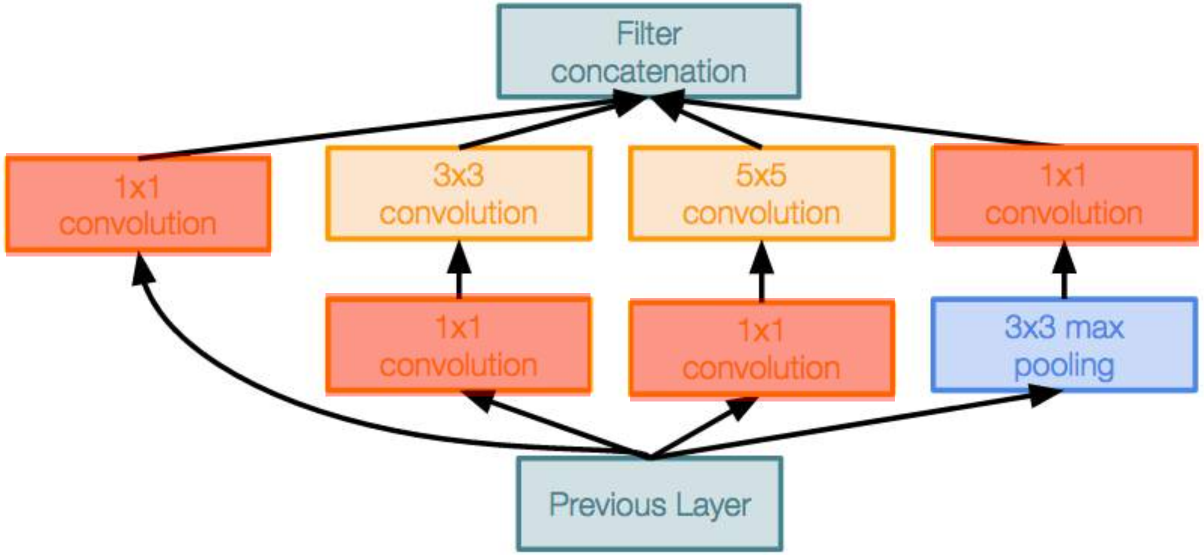
Szegedy et al, “Going deeper with convolutions”, CVPR 2015

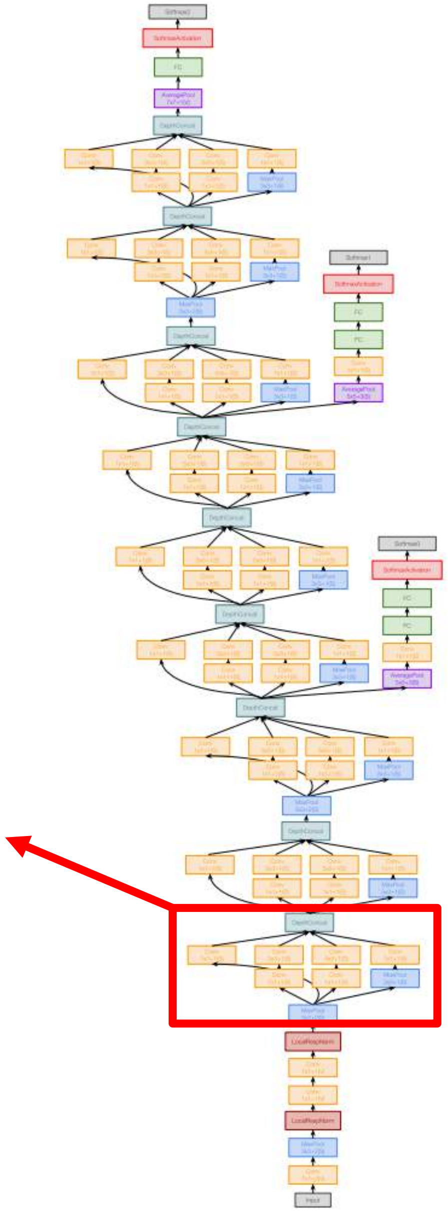
GoogLeNet: Inception Module

Inception module
Local unit with
parallel branches

Local structure repeated
many times throughout the
network

Uses 1x1 "Bottleneck"
layers to reduce channel
dimension before
expensive conv (we will
revisit this with ResNet!)



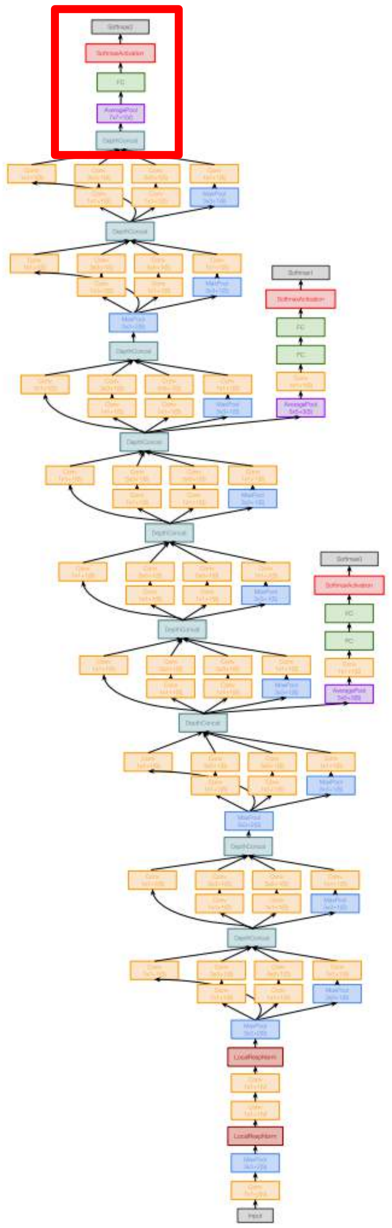


Szegedy et al, "Going deeper with convolutions", CVPR 2015

GoogLeNet: Global Average Pooling

No large FC layers at the end! Instead uses **global average pooling** to collapse spatial dimensions, and one linear layer to produce class scores
(Recall VGG-16: Most parameters were in the FC layers!)

	Input size		Layer				Output size				
Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (k)	flop (M)
avg-pool	1024	7		7	1	0	1024	1	4	0	0
fc	1024		1000				1000		0	1025	1



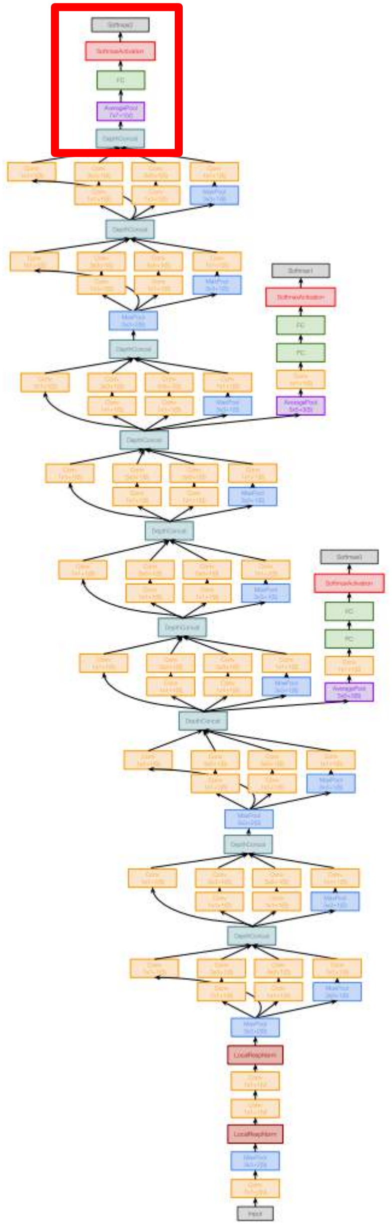
GoogLeNet: Global Average Pooling

No large FC layers at the end! Instead uses "global average pooling" to collapse spatial dimensions, and one linear layer to produce class scores
(Recall VGG-16: Most parameters were in the FC layers!)

	Input size		Layer				Output size				
Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (k)	flop (M)
avg-pool	1024	7		7	1	0	1024	1	4	0	0
fc	1024		1000				1000		0	1025	1

Compare with VGG-16:

Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (K)	flop (M)
flatten	512	7					25088		98		
fc6	25088			4096			4096		16	102760	103
fc7	4096			4096			4096		16	16777	17
fc8	4096			1000			1000		4	4096	4

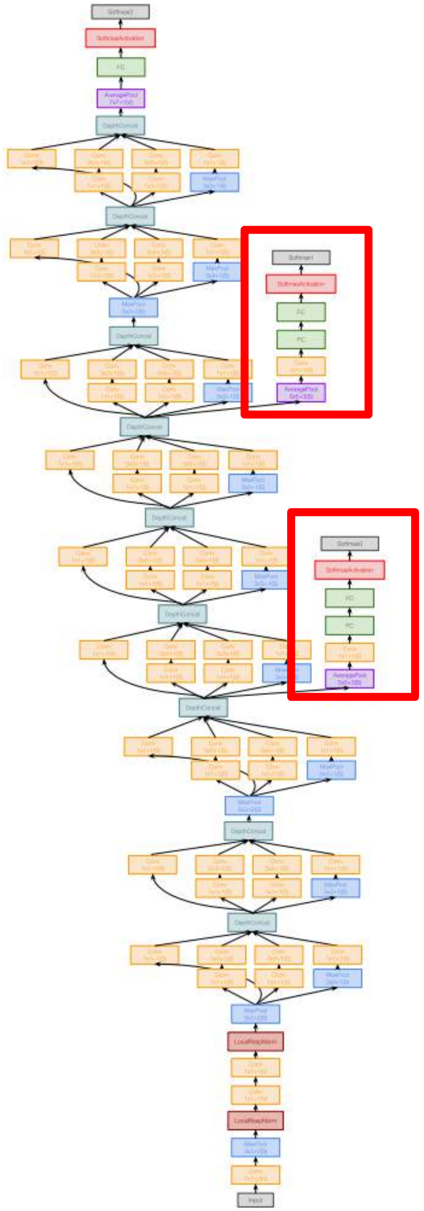


GoogLeNet: Auxiliary Classifiers

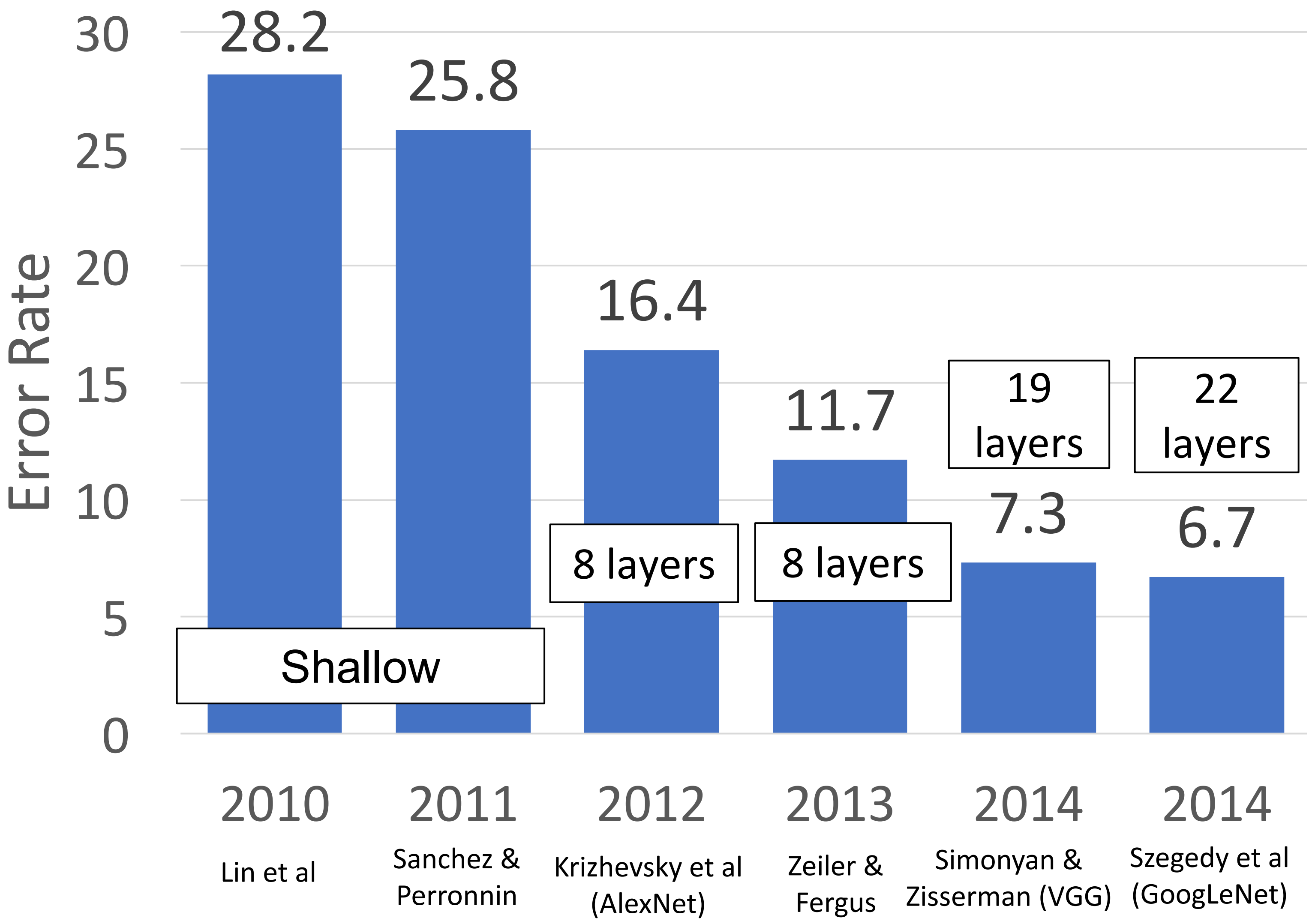
Training using loss at the end of the network didn't work well: Network is too deep, gradients don't propagate cleanly

As a hack, attach "auxiliary classifiers" at several intermediate points in the network that also try to classify the image and receive loss

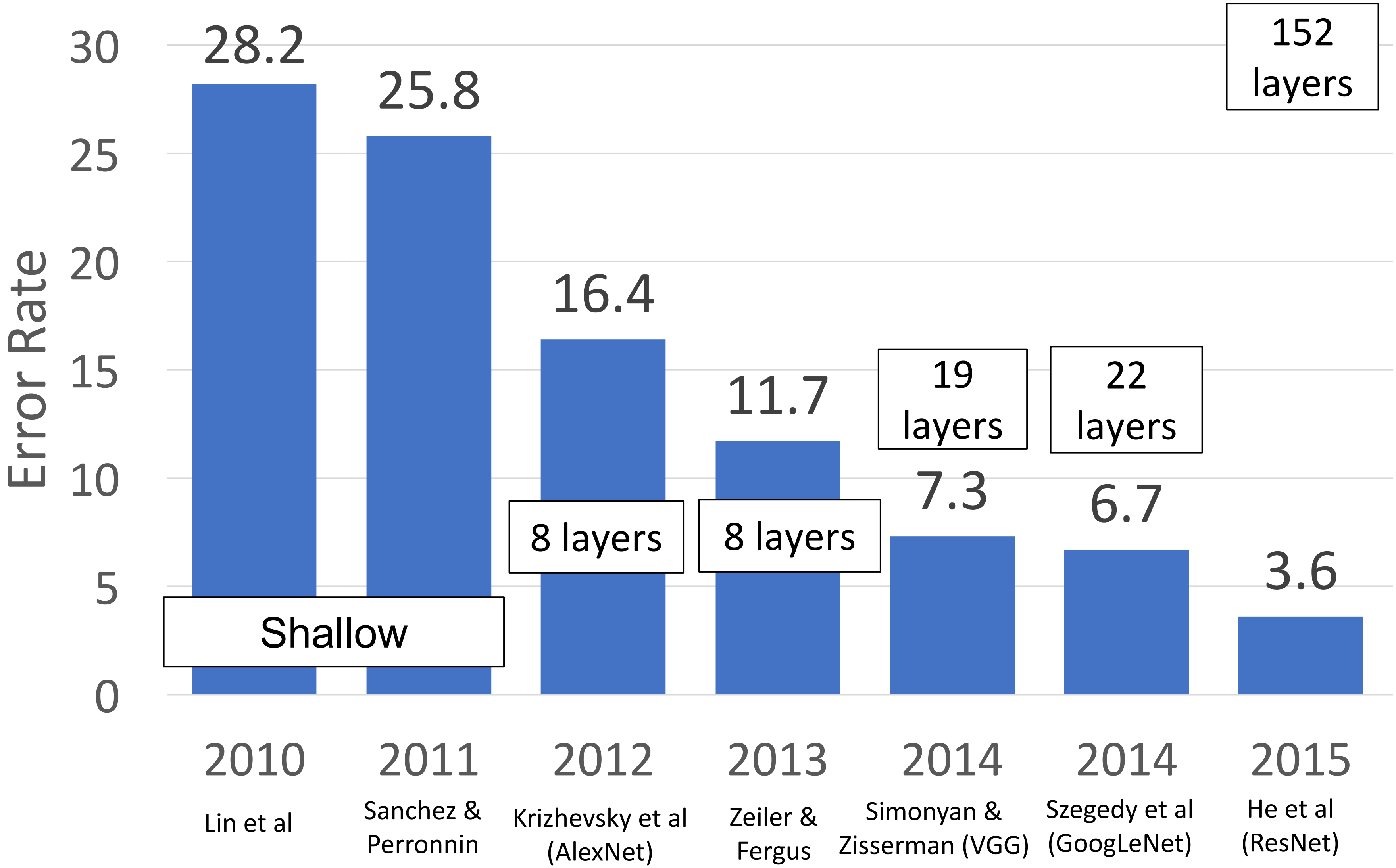
GoogLeNet was before batch normalization! With BatchNorm no longer need to use this trick



ImageNet Classification Challenge



ImageNet Classification Challenge



Residual Networks

Once we have Batch Normalization, we can train networks with 10+ layers. What happens as we go deeper?

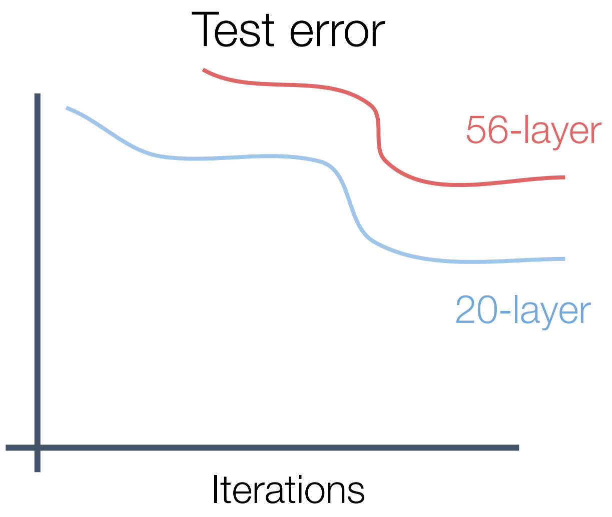
He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Residual Networks

Once we have Batch Normalization, we can train networks with 10+ layers. What happens as we go deeper?

Deeper model does worse than shallow model!

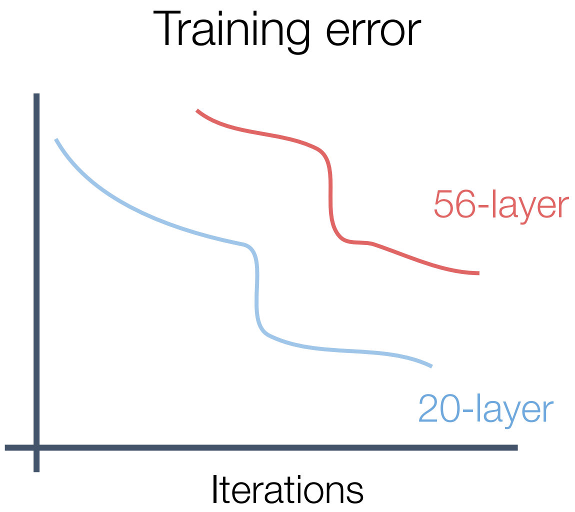
Initial guess: Deep model is **overfitting** since it is much bigger than the other model

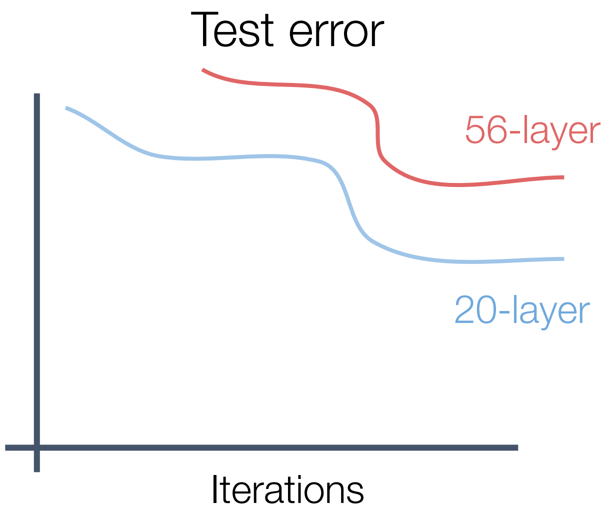


He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Residual Networks

Once we have Batch Normalization, we can train networks with 10+ layers. What happens as we go deeper?





In fact the deep model seems to be **underfitting** since it also performs worse than the shallow model on the training set! It is actually **underfitting**

He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Residual Networks

A deeper model can <u>emulate</u> a shallower model: copy layers from shallower model, set extra layers to identity

Thus deeper models should do at least as good as shallow models

Hypothesis: This is an <u>optimization</u> problem. Deeper models are harder to optimize, and in particular don't learn identity functions to emulate shallow models

He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Residual Networks

A deeper model can emulate a shallower model: copy layers from shallower model, set extra layers to identity

Thus deeper models should do at least as good as shallow models

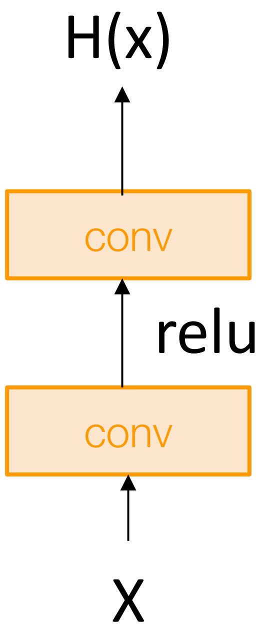
Hypothesis: This is an optimization problem. Deeper models are harder to optimize, and in particular don't learn identity functions to emulate shallow models

Solution: Change the network so learning identity functions with extra layers is easy!

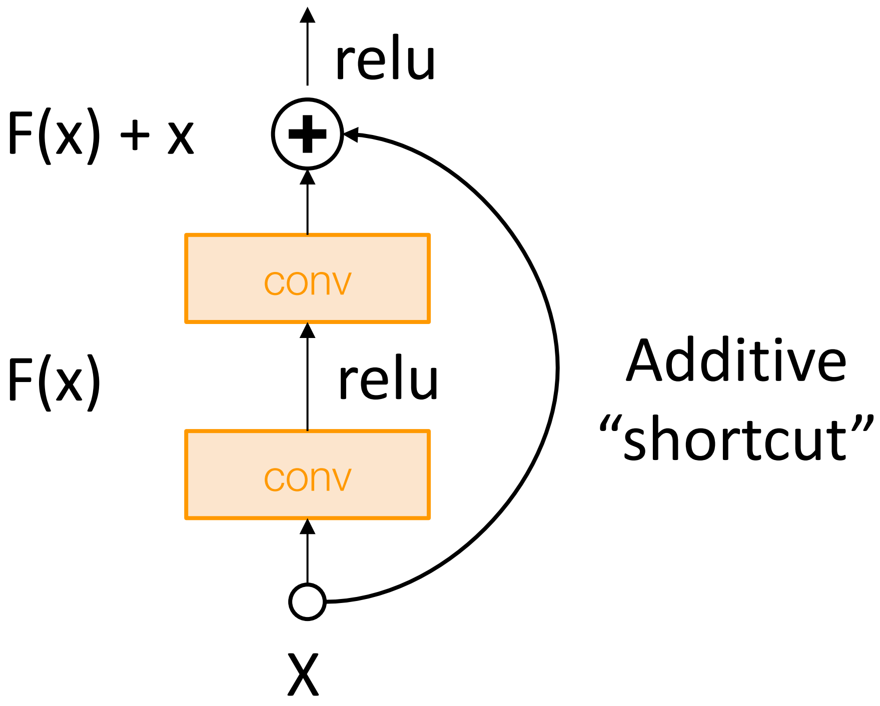
He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Residual Networks

Solution: Change the network so learning identity functions with extra layers is easy!



"Plain" block

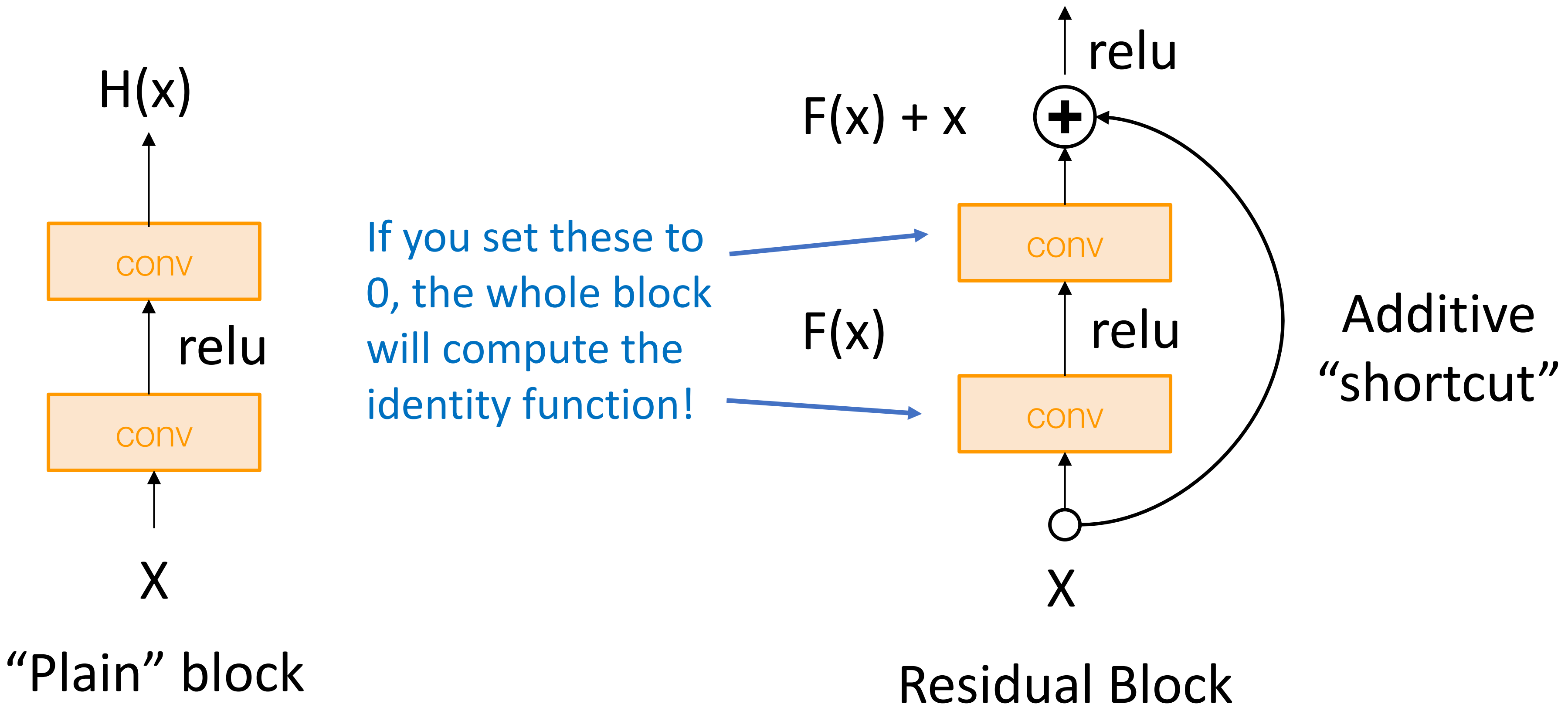


Residual Block

He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Residual Networks

Solution: Change the network so learning identity functions with extra layers is easy!



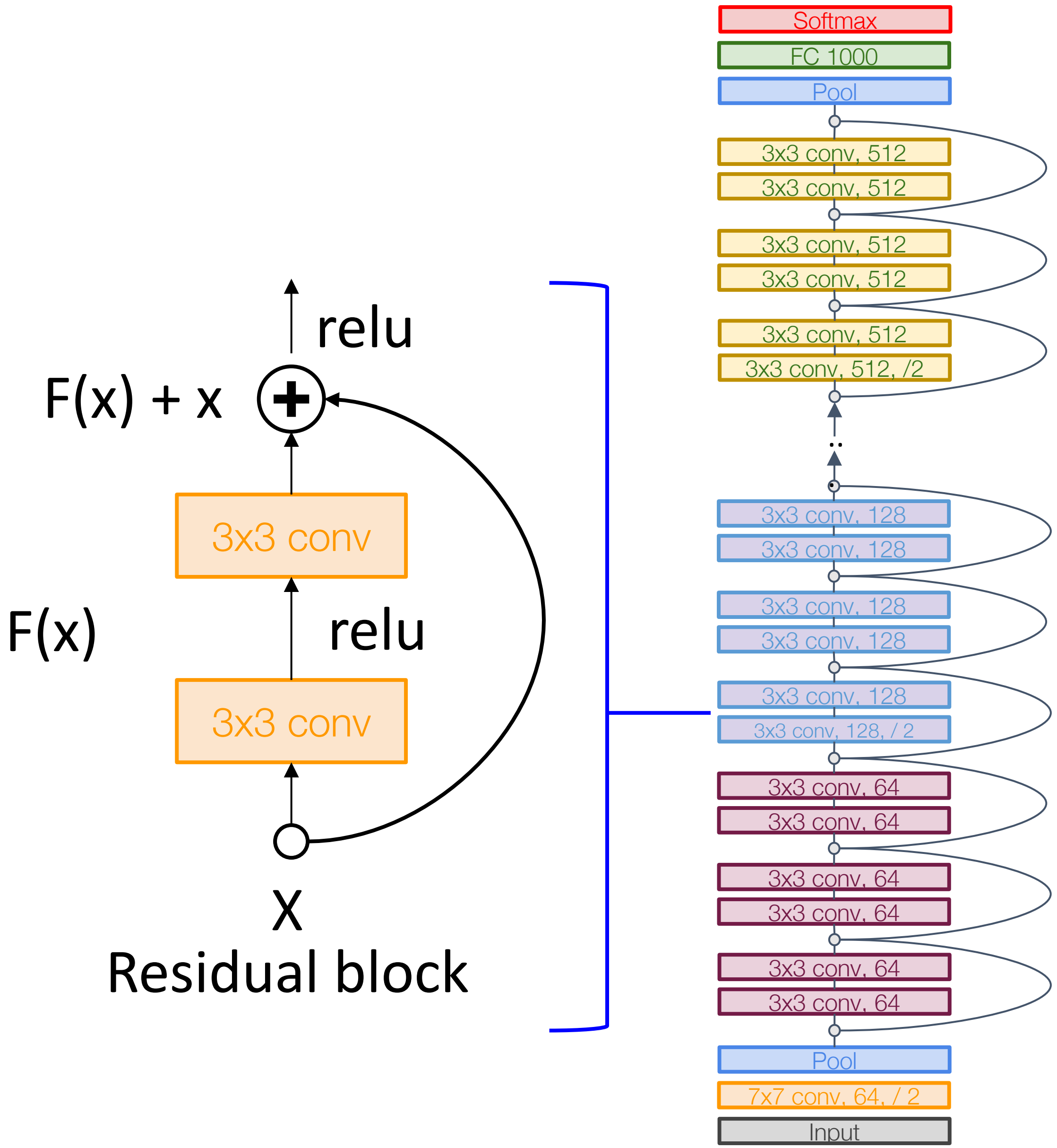
He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Residual Networks

A residual network is a stack of many residual blocks

Regular design, like VGG: each residual block has two 3x3 conv

Network is divided into **stages**: the first block of each stage halves the resolution (with stride-2 conv) and doubles the number of channels

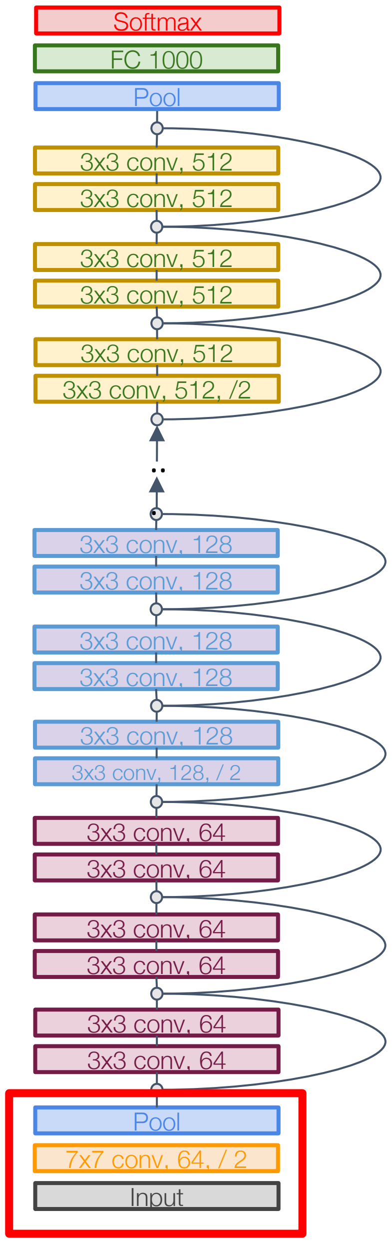


He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Residual Networks

Uses the same aggressive **stem** as GoogleNet to downsample the input 4x before applying residual blocks:

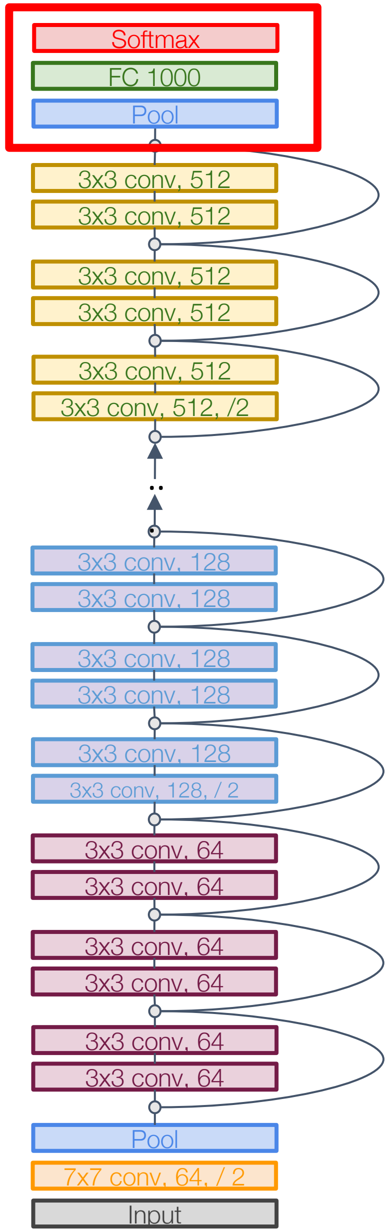
	Input size		Layer				Output size				
Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (k)	flop (M)
conv	3	224	64	7	2	3	64	112	3136	9	118
max-pool	64	112		3	2	1	64	56	784	0	2



He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Residual Networks

Like GoogLeNet, no big fully-connected-layers: instead use **global average pooling** and a single linear layer at the end

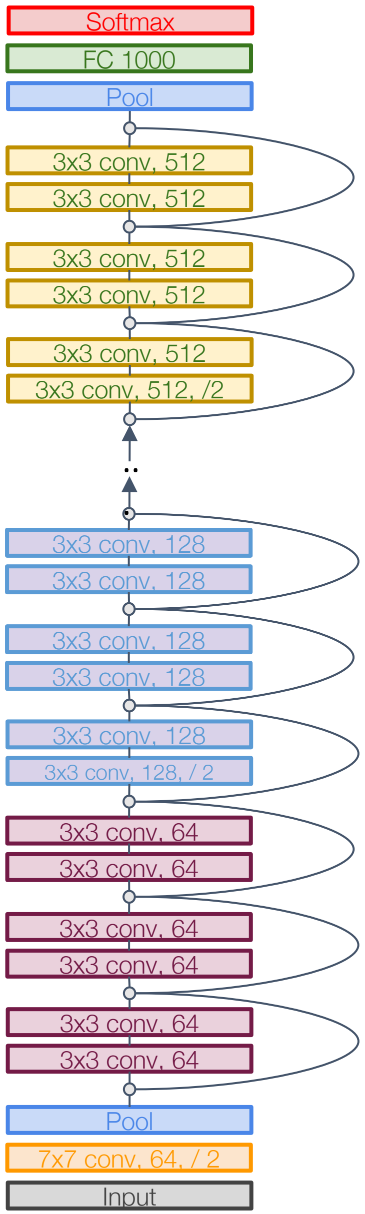


He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Residual Networks

ResNet-18:
Stem: 1 conv layer
Stage 1 (C=64): 2 res. block = 4 conv
Stage 2 (C=128): 2 res. block = 4 conv
Stage 3 (C=256): 2 res. block = 4 conv
Stage 4 (C=512): 2 res. block = 4 conv
Linear

ImageNet top-5 error: 10.92
GFLOP: 1.8



He et al, "Deep Residual Learning for Image Recognition", CVPR 2016
Error rates are 224x224 single-crop testing, reported by torchvision

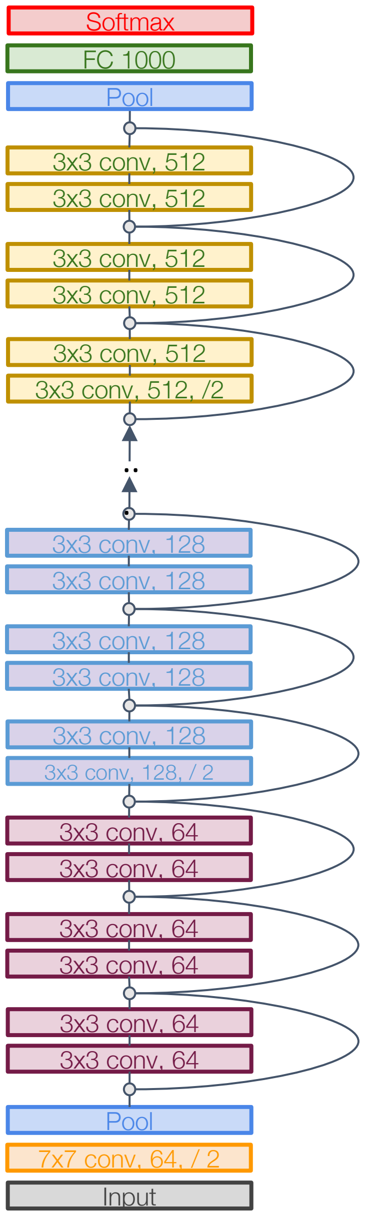
Residual Networks

ResNet-18:
Stem: 1 conv layer
Stage 1 (C=64): 2 res. block = 4 conv
Stage 2 (C=128): 2 res. block = 4 conv
Stage 3 (C=256): 2 res. block = 4 conv
Stage 4 (C=512): 2 res. block = 4 conv
Linear

ImageNet top-5 error: 10.92
GFLOP: 1.8

ResNet-34:
Stem: 1 conv layer
Stage 1: 3 res. block = 6 conv
Stage 2: 4 res. block = 8 conv
Stage 3: 6 res. block = 12 conv
Stage 4: 3 res. block = 6 conv
Linear

ImageNet top-5 error: 8.58
GFLOP: 3.6



He et al, "Deep Residual Learning for Image Recognition", CVPR 2016
Error rates are 224x224 single-crop testing, reported by torchvision

Residual Networks

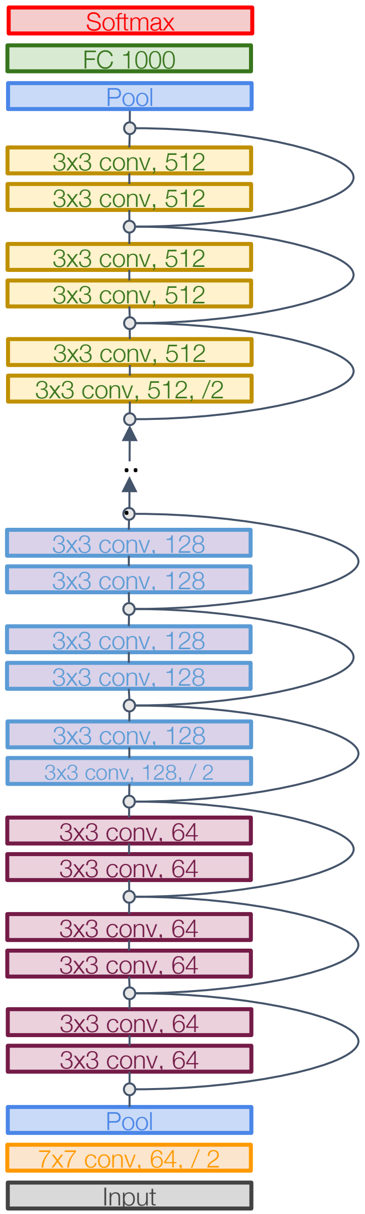
ResNet-18:
Stem: 1 conv layer
Stage 1 (C=64): 2 res. block = 4 conv
Stage 2 (C=128): 2 res. block = 4 conv
Stage 3 (C=256): 2 res. block = 4 conv
Stage 4 (C=512): 2 res. block = 4 conv
Linear

ImageNet top-5 error: 10.92
GFLOP: 1.8

ResNet-34:
Stem: 1 conv layer
Stage 1: 3 res. block = 6 conv
Stage 2: 4 res. block = 8 conv
Stage 3: 6 res. block = 12 conv
Stage 4: 3 res. block = 6 conv
Linear

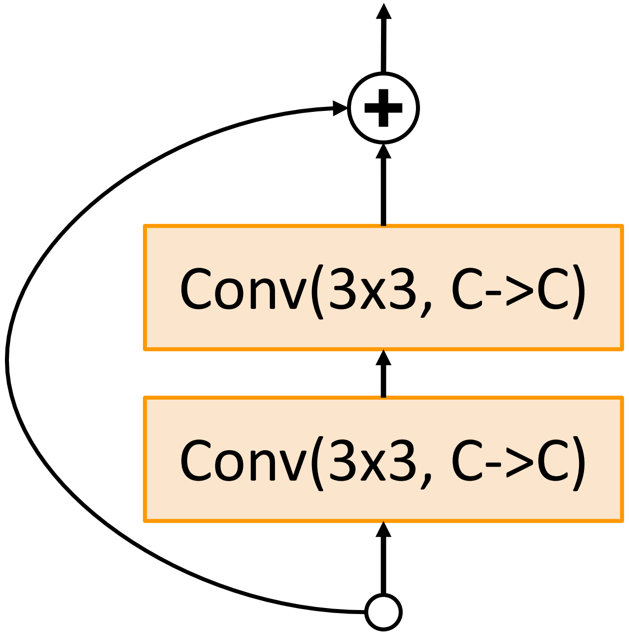
ImageNet top-5 error: 8.58
GFLOP: 3.6

VGG-16:
ImageNet top-5 error: 9.62
GFLOP: 13.6



He et al, "Deep Residual Learning for Image Recognition", CVPR 2016
Error rates are 224x224 single-crop testing, reported by torchvision

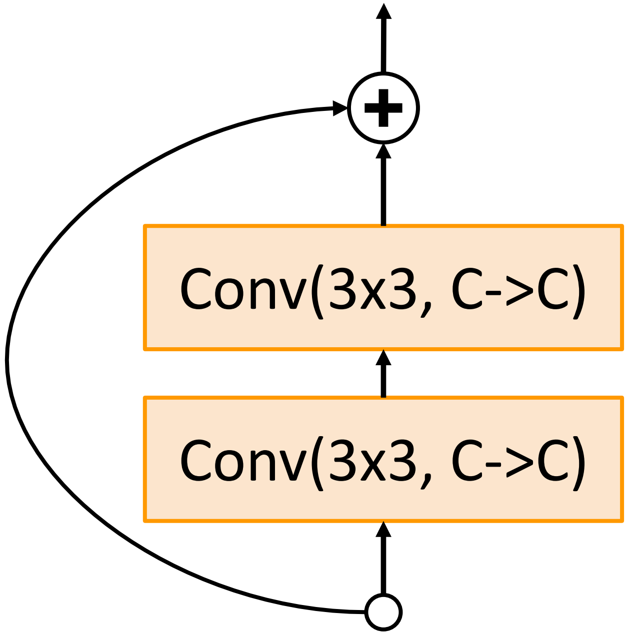
Residual Networks: Basic Block



"Basic"
Residual block

He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Residual Networks: Basic Block



FLOPs: $9HWC^2$

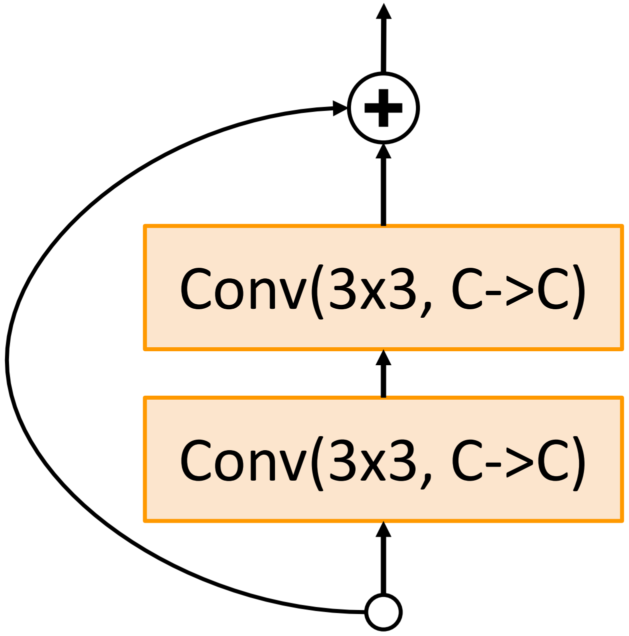
FLOPs: $9HWC^2$

“Basic”
Residual block

Total FLOPs:
$18HWC^2$

He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016

Residual Networks: Bottleneck Block

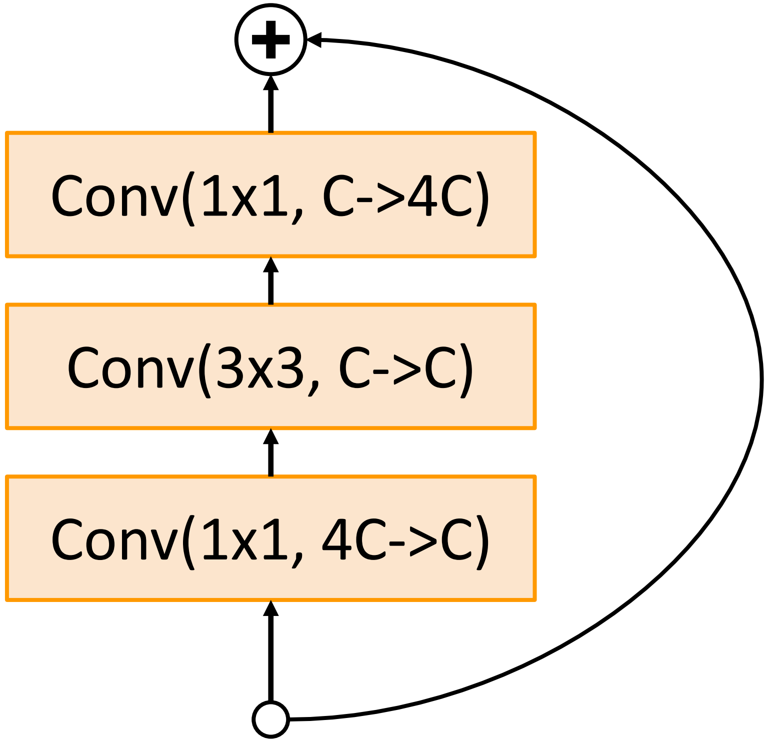


FLOPs: $9HWC^2$

FLOPs: $9HWC^2$

"Basic"
Residual block

Total FLOPs:
$18HWC^2$



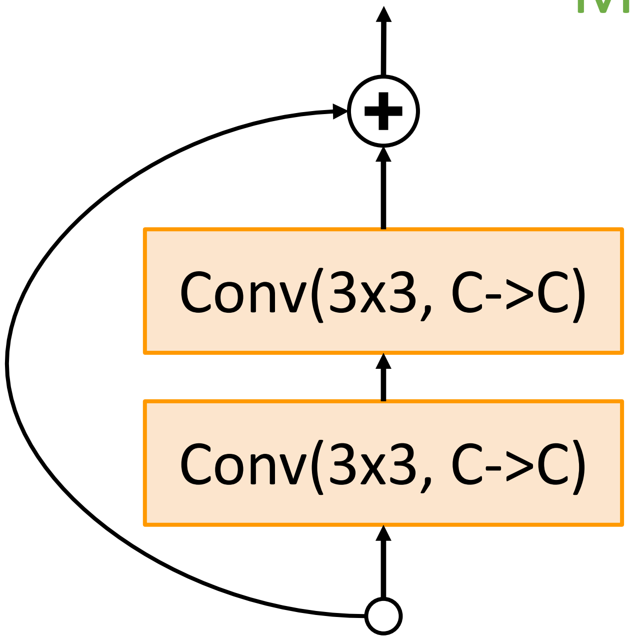
"Bottleneck"
Residual block

He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Residual Networks: Bottleneck Block

More layers, less computational cost!

more nonlinearity.



FLOPs: $9HWC^2$

FLOPs: $9HWC^2$

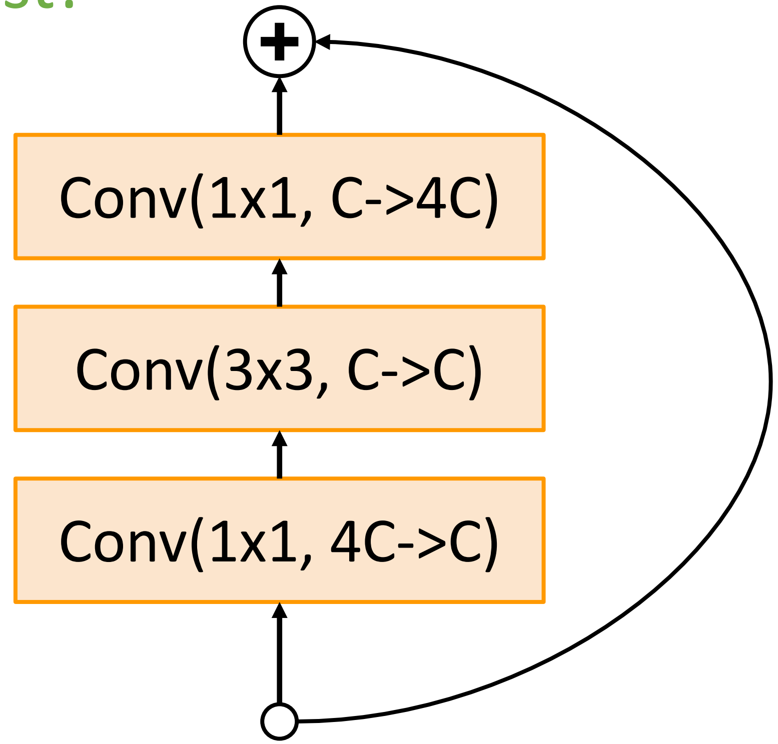
"Basic"
Residual block

Total FLOPs:
$18HWC^2$

FLOPs: $4HWC^2$

FLOPs: $9HWC^2$

FLOPs: $4HWC^2$



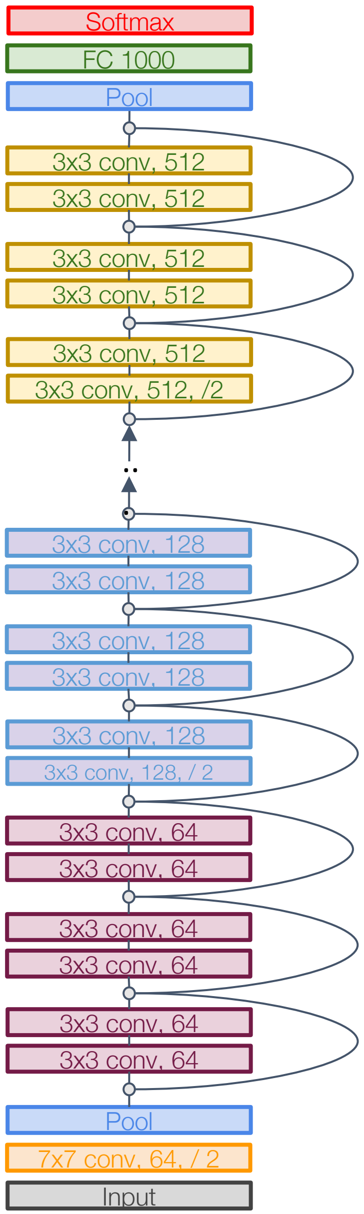
Total FLOPs:
$17HWC^2$

"Bottleneck"
Residual block

He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Residual Networks

	Block type	Stem layers	Stage 1		Stage 2		Stage 3		Stage 4		FC layers	GFLOP	ImageNet top-5 error
			Blocks	Layers	Blocks	Layers	Blocks	Layers	Blocks	Layers			
ResNet-18	Basic	1	2	4	2	4	2	4	2	4	1	1.8	10.92
ResNet-34	Basic	1	3	6	4	8	6	12	3	6	1	3.6	8.58

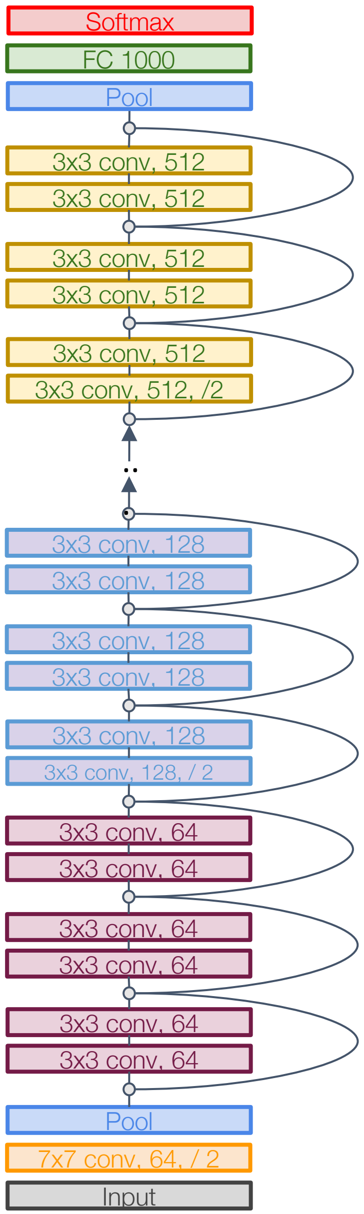


He et al, "Deep Residual Learning for Image Recognition", CVPR 2016
Error rates are 224x224 single-crop testing, reported by torchvision

Residual Networks

ResNet-50 is the same as ResNet-34, but replaces Basic blocks with Bottleneck Blocks.
This is a great baseline architecture for many tasks even today!

			Stage 1		Stage 2		Stage 3		Stage 4				
	Block type	Stem layers	Blocks	Layers	Blocks	Layers	Blocks	Layers	Blocks	Layers	FC layers	GFLOP	ImageNet top-5 error
ResNet-18	Basic	1	2	4	2	4	2	4	2	4	1	1.8	10.92
ResNet-34	Basic	1	3	6	4	8	6	12	3	6	1	3.6	8.58
ResNet-50	Bottle	1	3	9	4	12	6	18	3	9	1	3.8	7.13

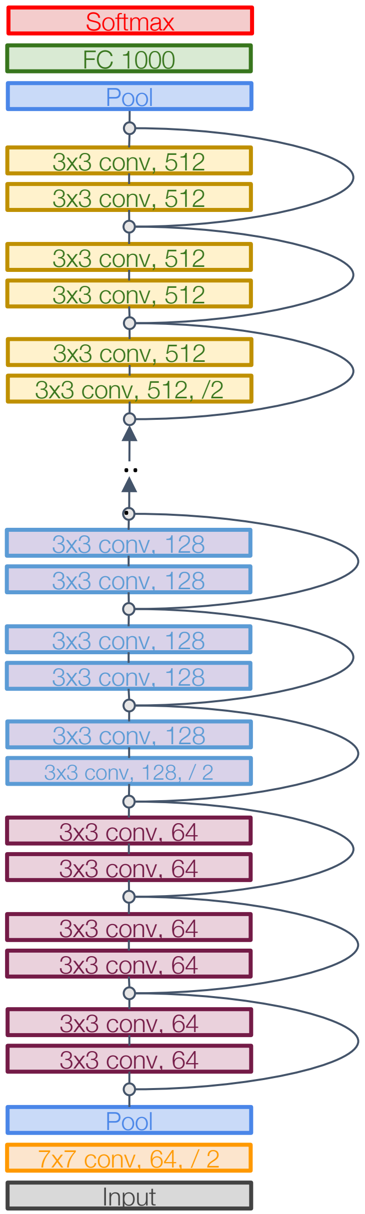


He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016
Error rates are 224x224 single-crop testing, reported by torchvision

Residual Networks

Deeper ResNet-101 and ResNet-152 models are more accurate, but also more computationally heavy

			Stage 1		Stage 2		Stage 3		Stage 4				
	Block type	Stem layers	Blocks	Layers	Blocks	Layers	Blocks	Layers	Blocks	Layers	FC layers	GFLOP	ImageNet top-5 error
ResNet-18	Basic	1	2	4	2	4	2	4	2	4	1	1.8	10.92
ResNet-34	Basic	1	3	6	4	8	6	12	3	6	1	3.6	8.58
ResNet-50	Bottle	1	3	9	4	12	6	18	3	9	1	3.8	7.13
ResNet-101	Bottle	1	3	9	4	12	23	69	3	9	1	7.6	6.44
ResNet-152	Bottle	1	3	9	8	24	36	108	3	9	1	11.3	5.94



He et al, "Deep Residual Learning for Image Recognition", CVPR 2016
Error rates are 224x224 single-crop testing, reported by torchvision

Residual Networks

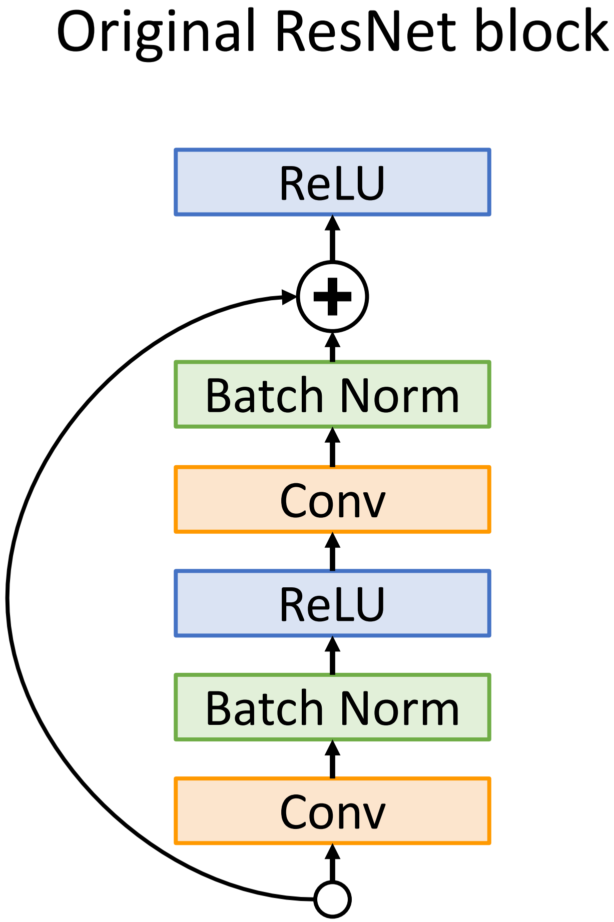
- Able to train very deep networks
- Deeper networks do better than shallow networks (as expected)
- Swept 1st place in all ILSVRC and COCO 2015 competitions
- Still widely used today!

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

He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Improving Residual Networks: Block Design

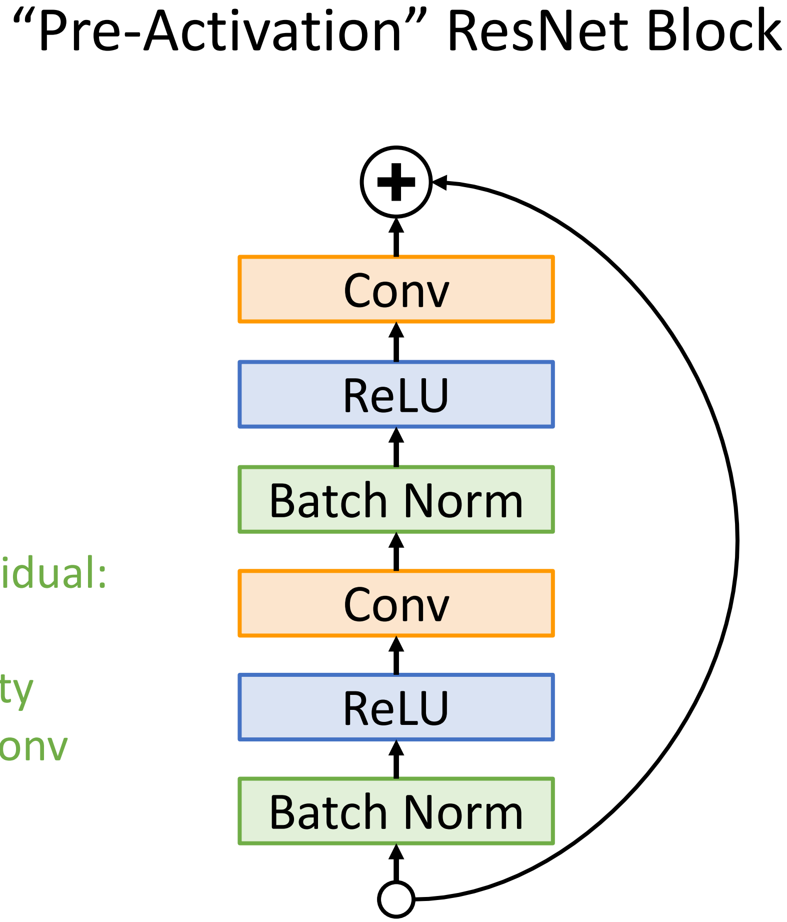


Note ReLU **after** residual:

Cannot actually learn identity function since outputs are nonnegative!

Note ReLU **inside** residual:

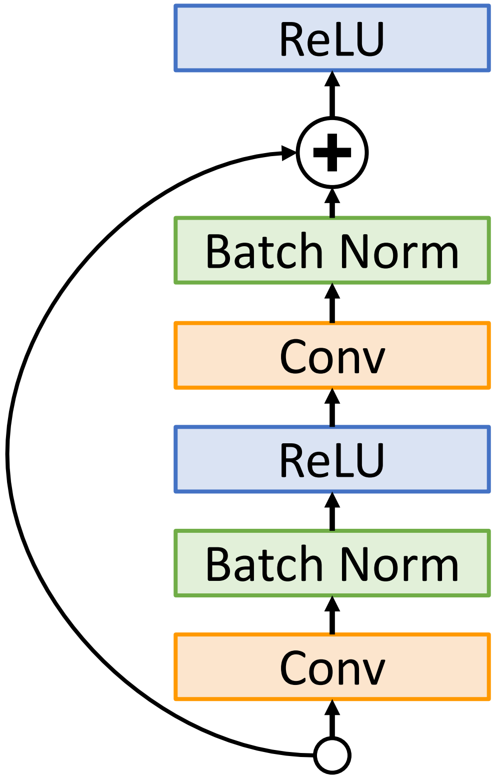
Can learn true identity function by setting Conv weights to zero!



He et al, "Identity mappings in deep residual networks", ECCV 2016

Improving Residual Networks: Block Design

Original ResNet block

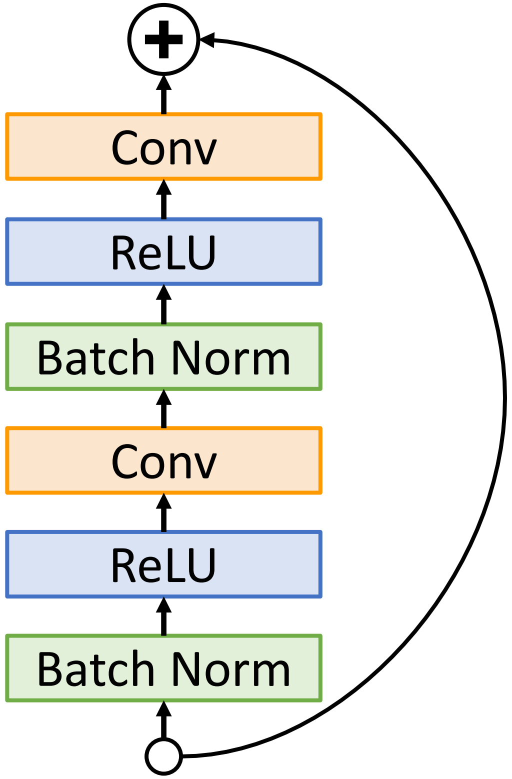


Slight improvement in accuracy (ImageNet top-1 error)

ResNet-152: 21.3 vs **21.1**
ResNet-200: 21.8 vs **20.7**

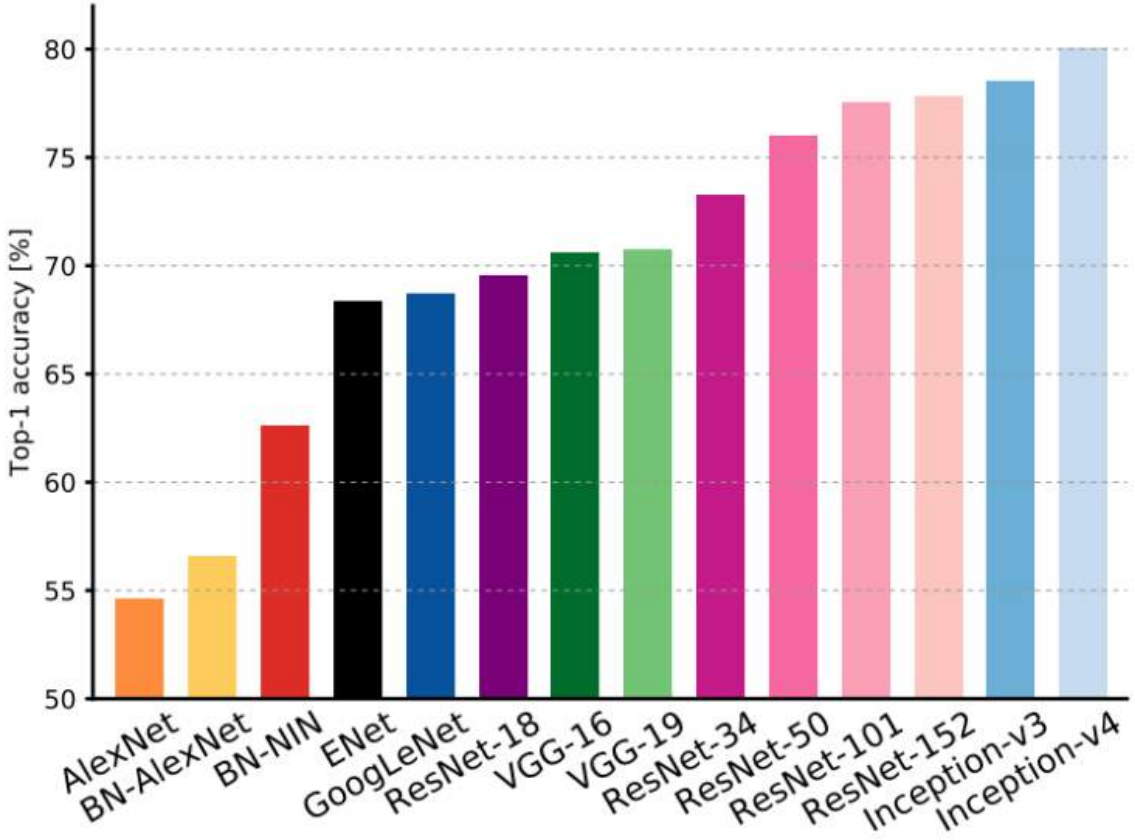
Not actually used that much in practice

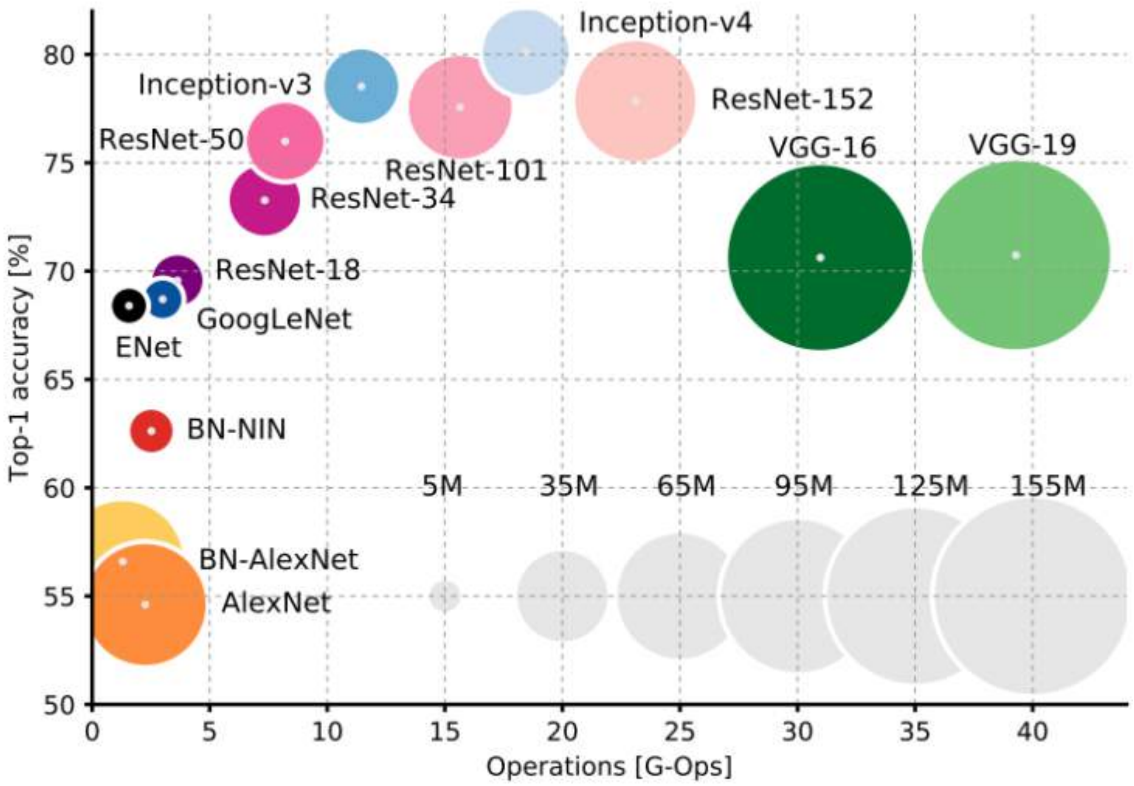
"Pre-Activation" ResNet Block



He et al, "Identity mappings in deep residual networks", ECCV 2016

Comparing Complexity

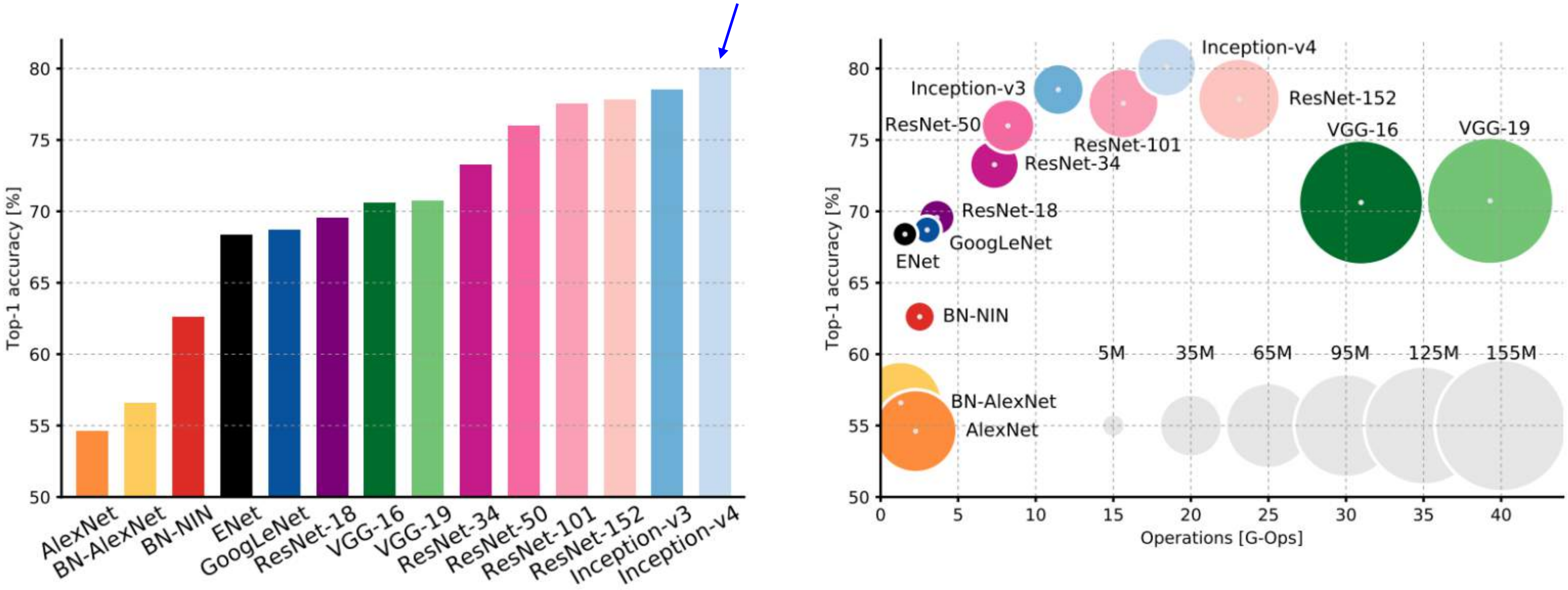




Canziani et al, "An analysis of deep neural network models for practical applications", 2017

Comparing Complexity

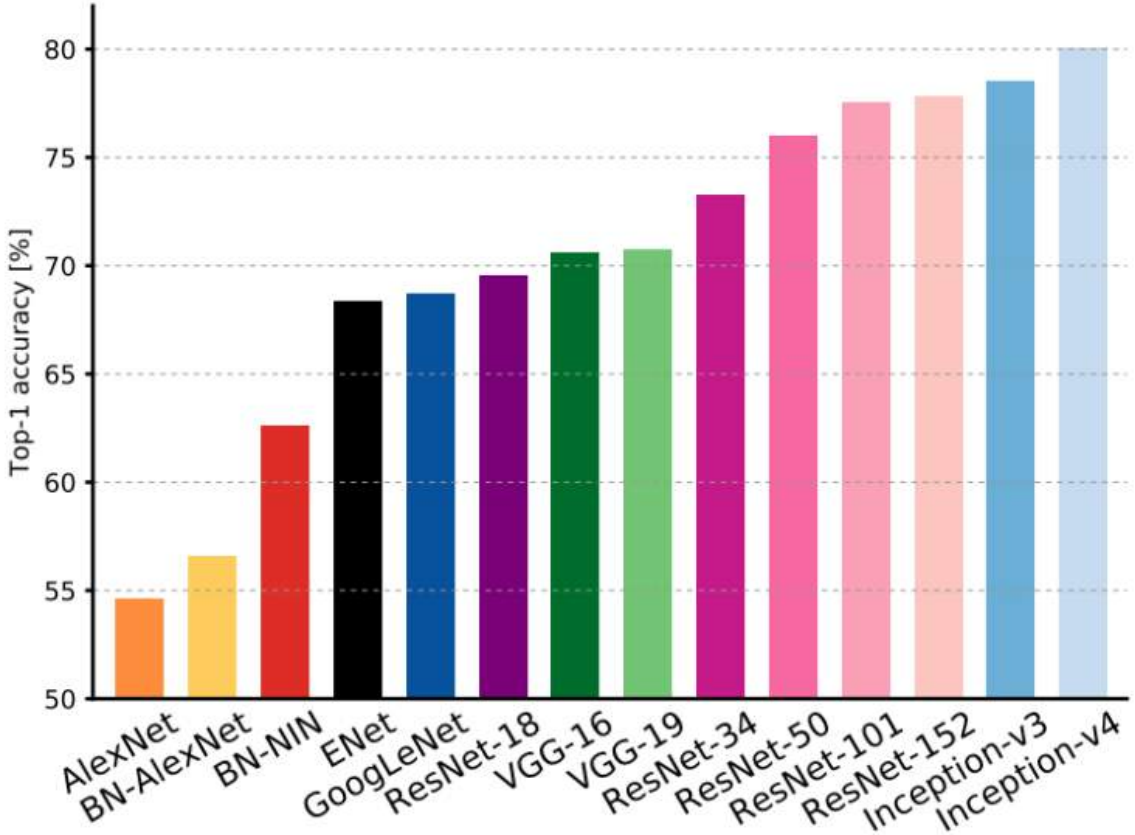
Inception-v4: Resnet + Inception!

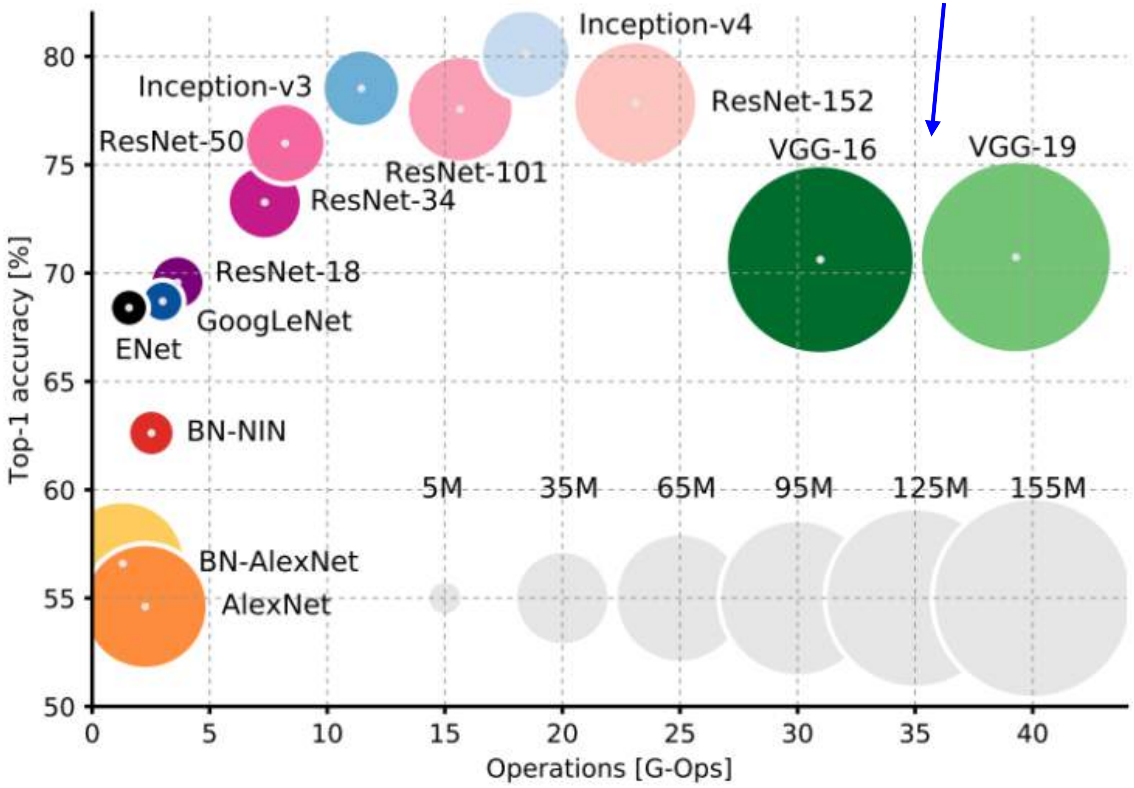


Canziani et al, "An analysis of deep neural network models for practical applications", 2017

Comparing Complexity

VGG: Highest memory, most operations

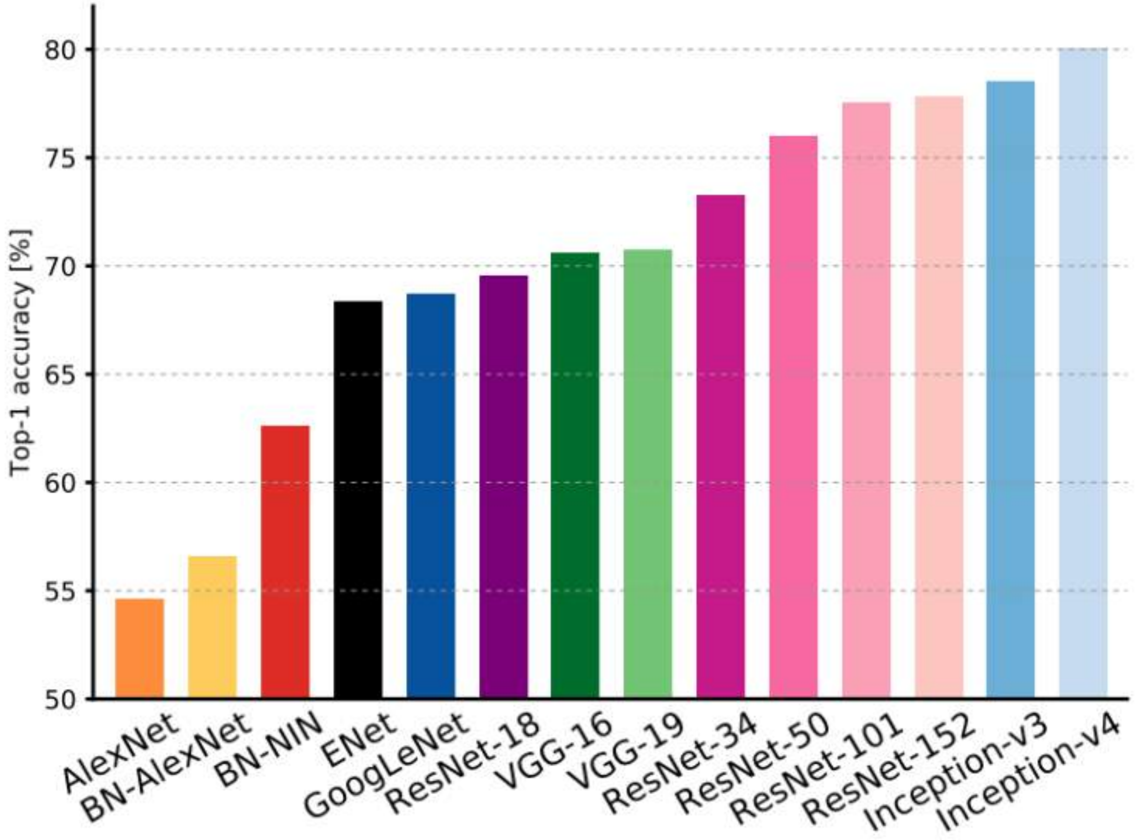


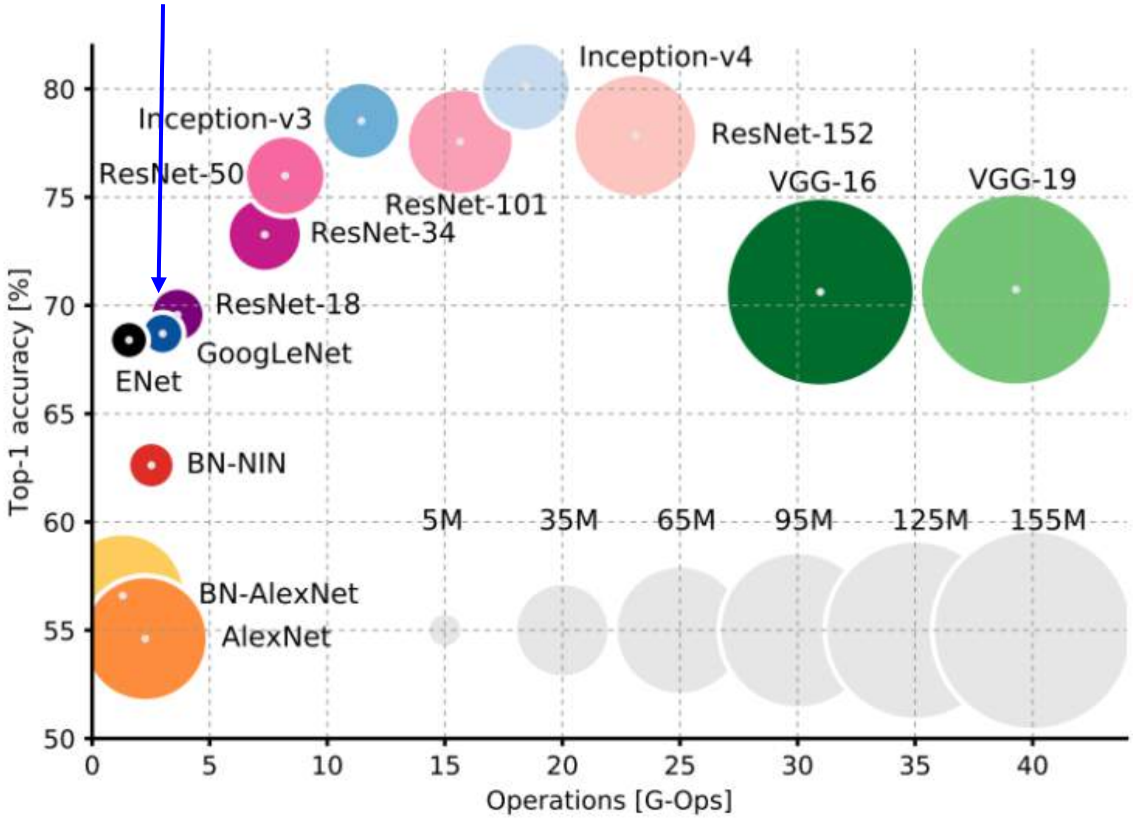


Canziani et al, "An analysis of deep neural network models for practical applications", 2017

Comparing Complexity

GoogLeNet:
Very efficient!

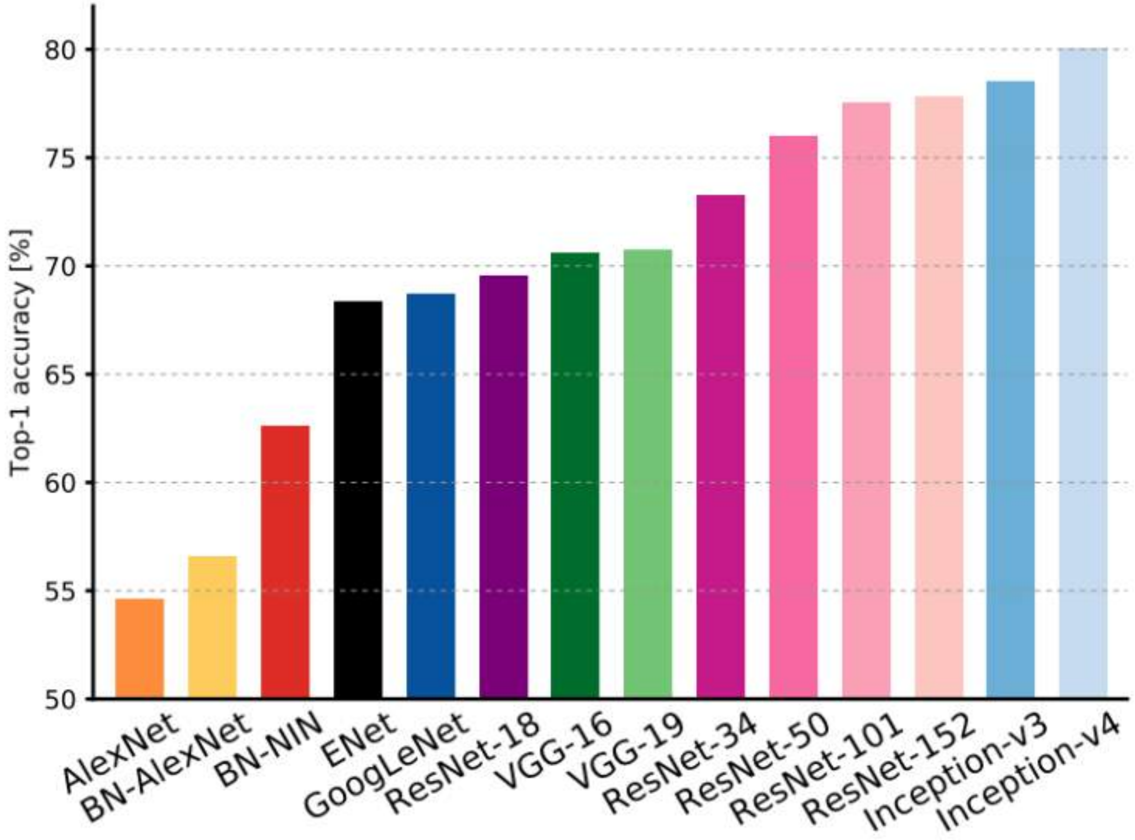


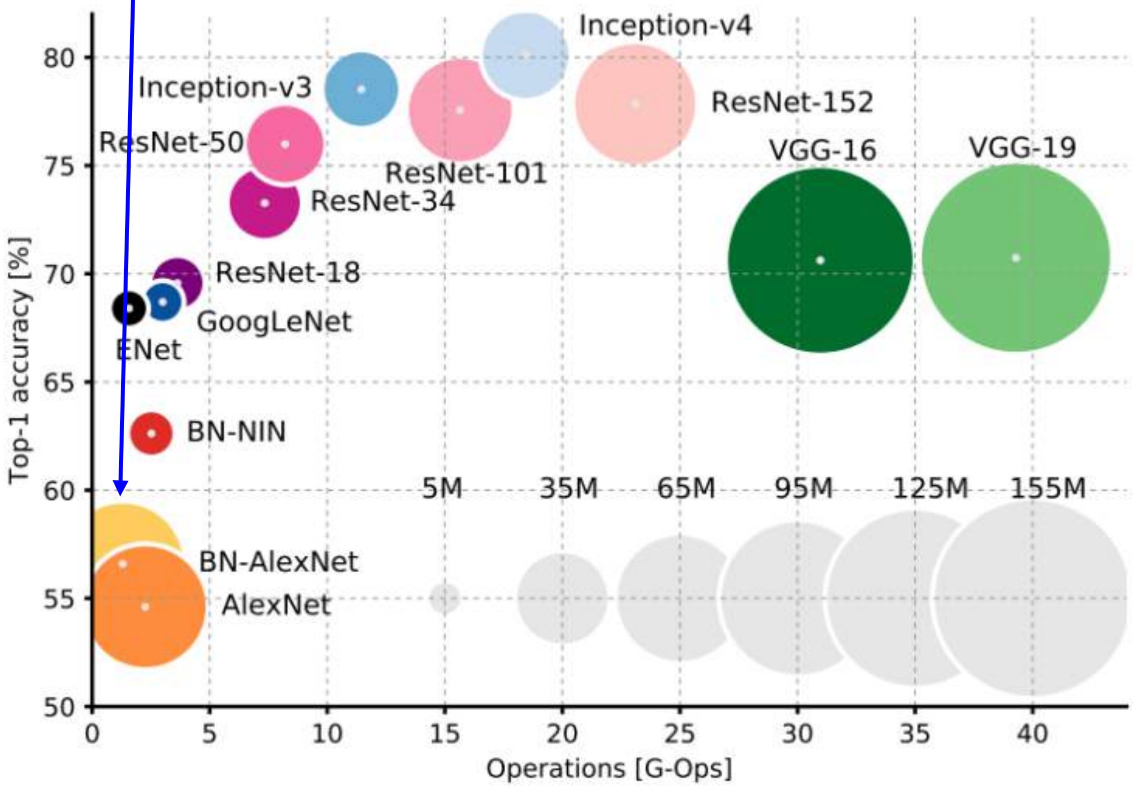


Canziani et al, "An analysis of deep neural network models for practical applications", 2017

Comparing Complexity

AlexNet: Low compute, lots of parameters

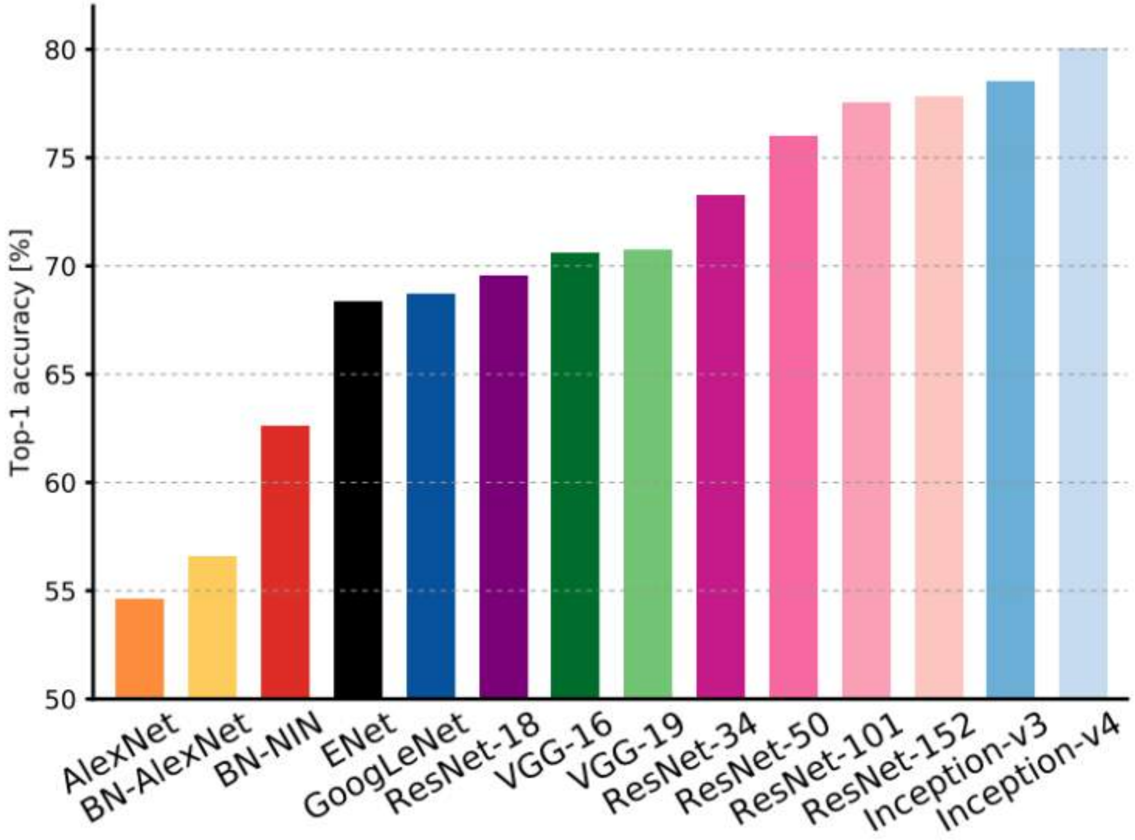


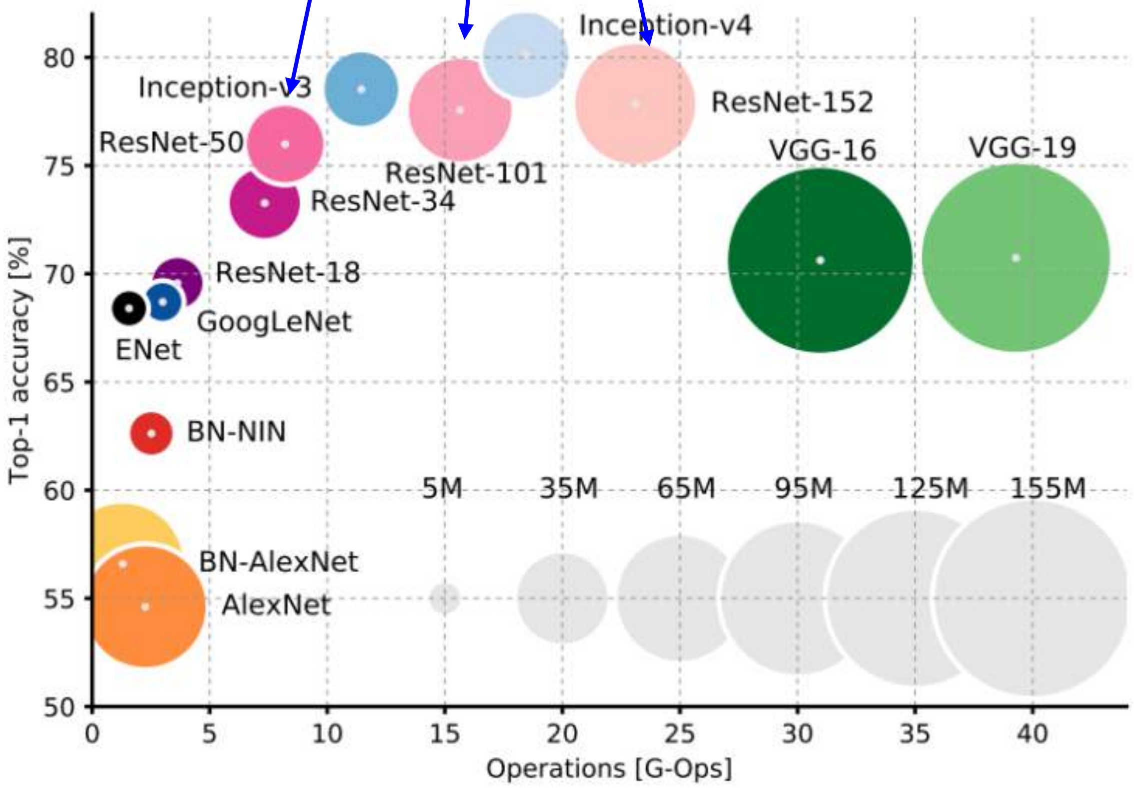


Canziani et al, “An analysis of deep neural network models for practical applications”, 2017

Comparing Complexity

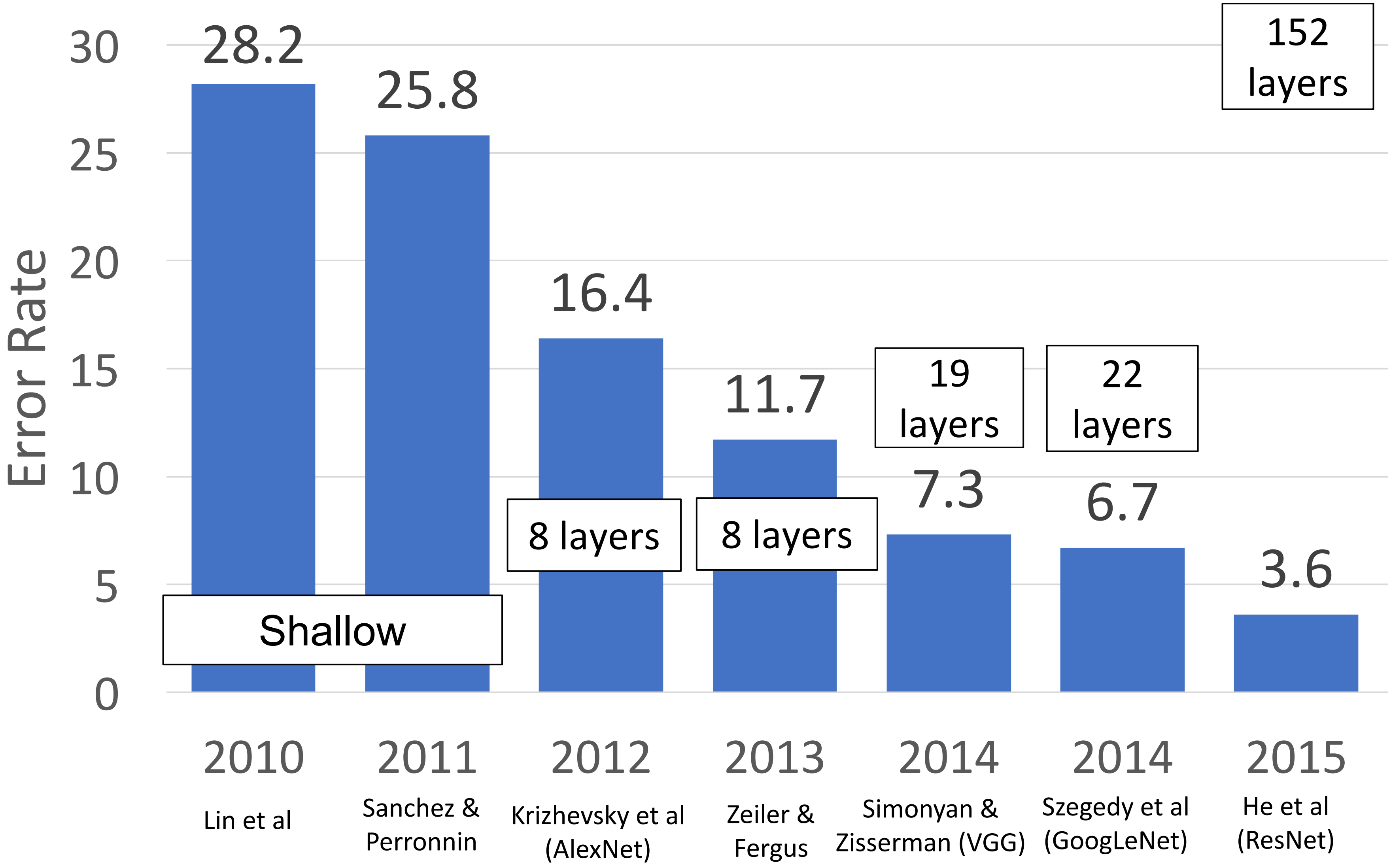
ResNet: Simple design,
moderate efficiency,
high accuracy



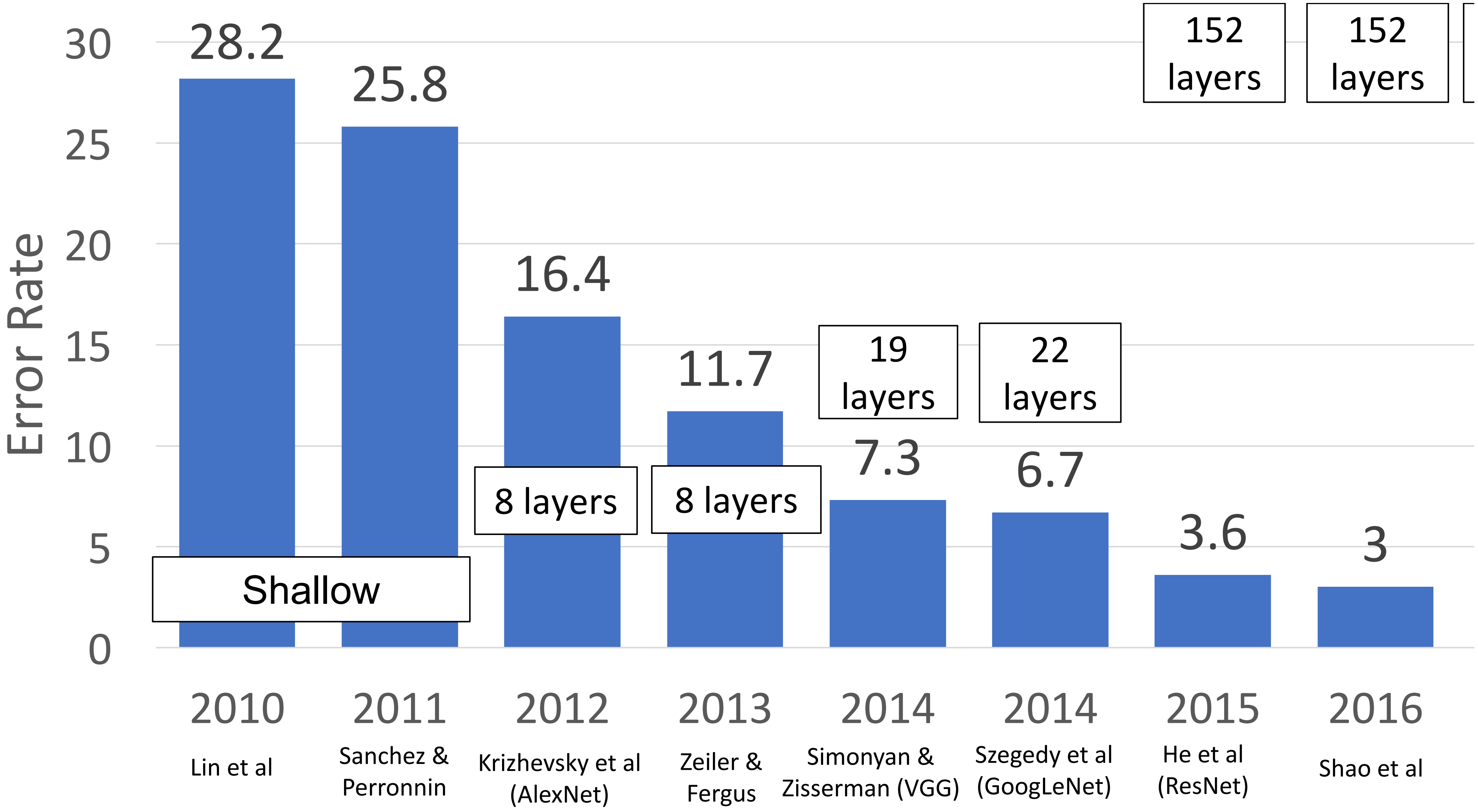


Canziani et al, "An analysis of deep neural network models for practical applications", 2017

ImageNet Classification Challenge



ImageNet Classification Challenge



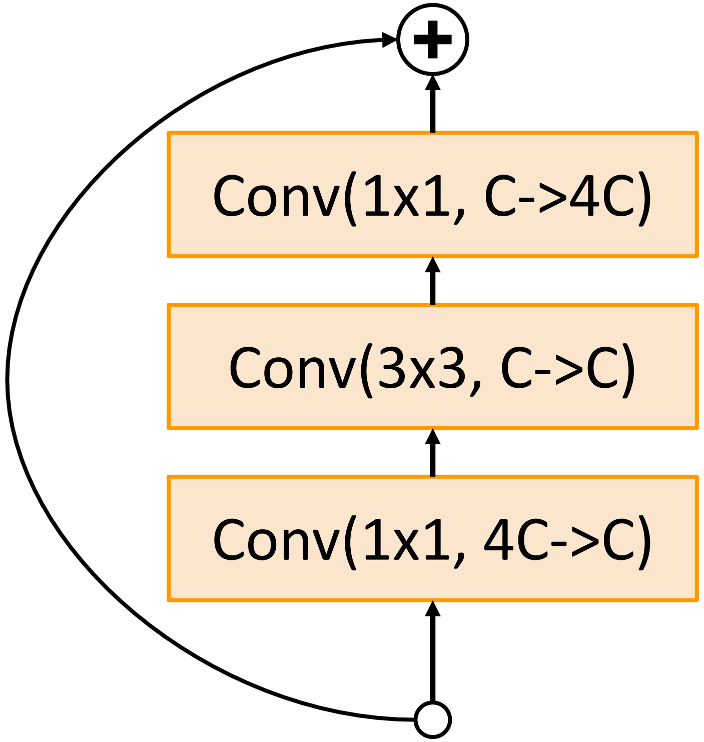
ImageNet 2016 winner: **Model Ensembles**

Multi-scale ensemble of Inception, Inception-Resnet, Resnet, Wide Resnet models

	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

Shao et al, 2016

Improving ResNets



FLOPs: $4HWC^2$

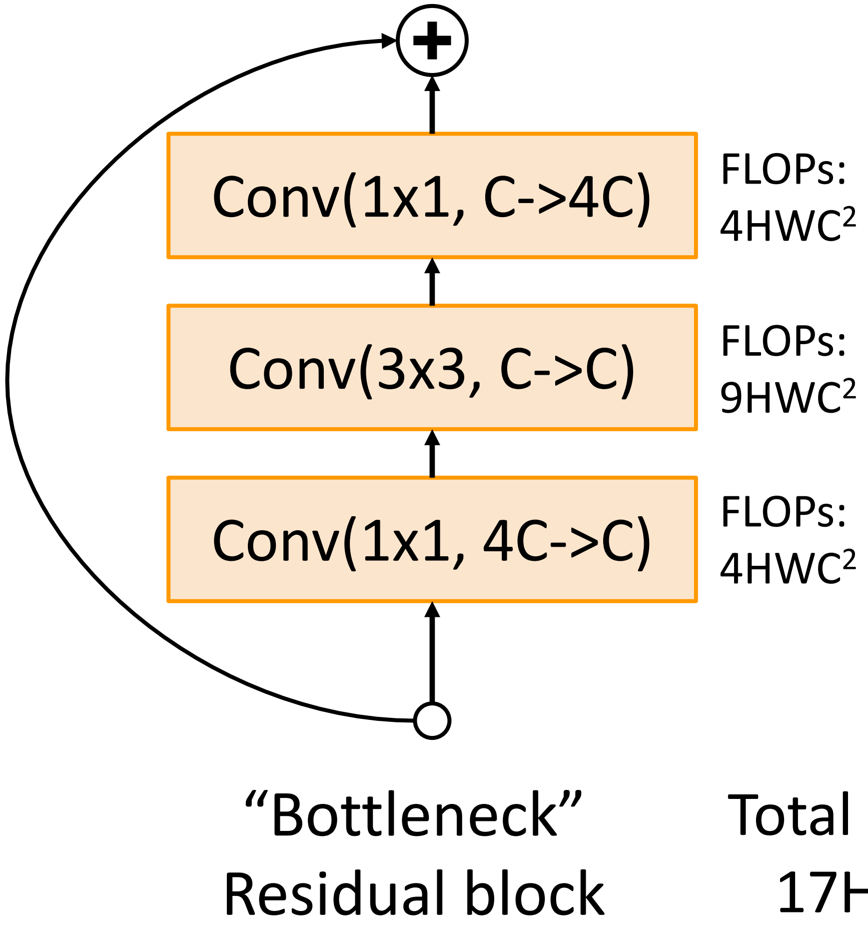
FLOPs: $9HWC^2$

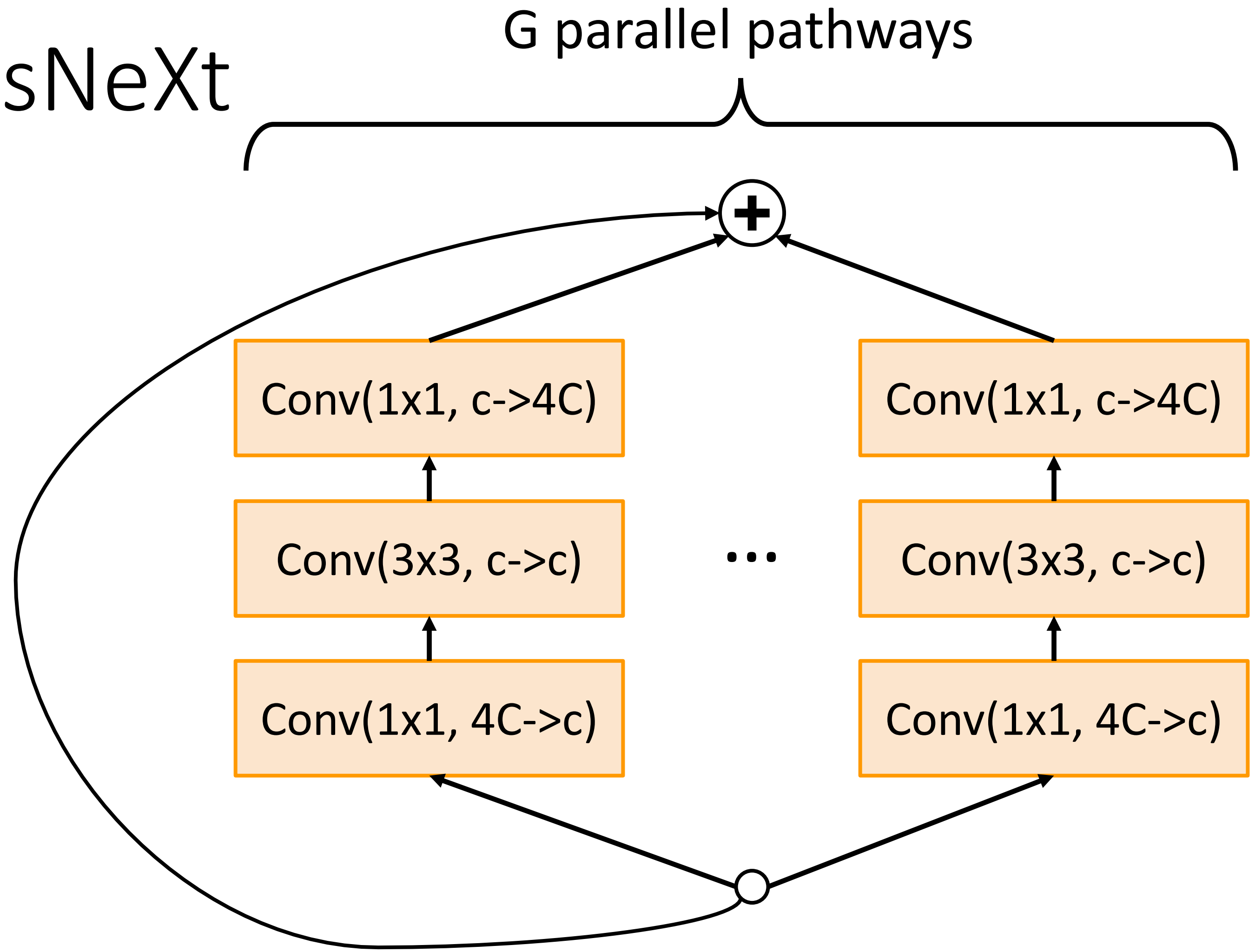
FLOPs: $4HWC^2$

"Bottleneck" Residual block

Total FLOPs: $17HWC^2$

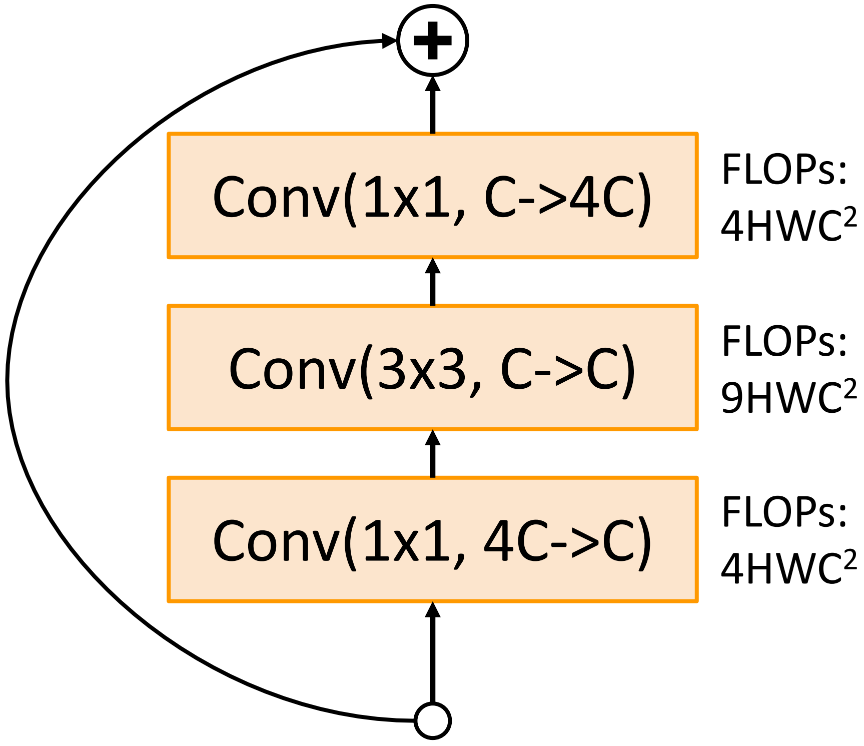
Improving ResNets: ResNeXt





Xie et al, "Aggregated residual transformations for deep neural networks", CVPR 2017

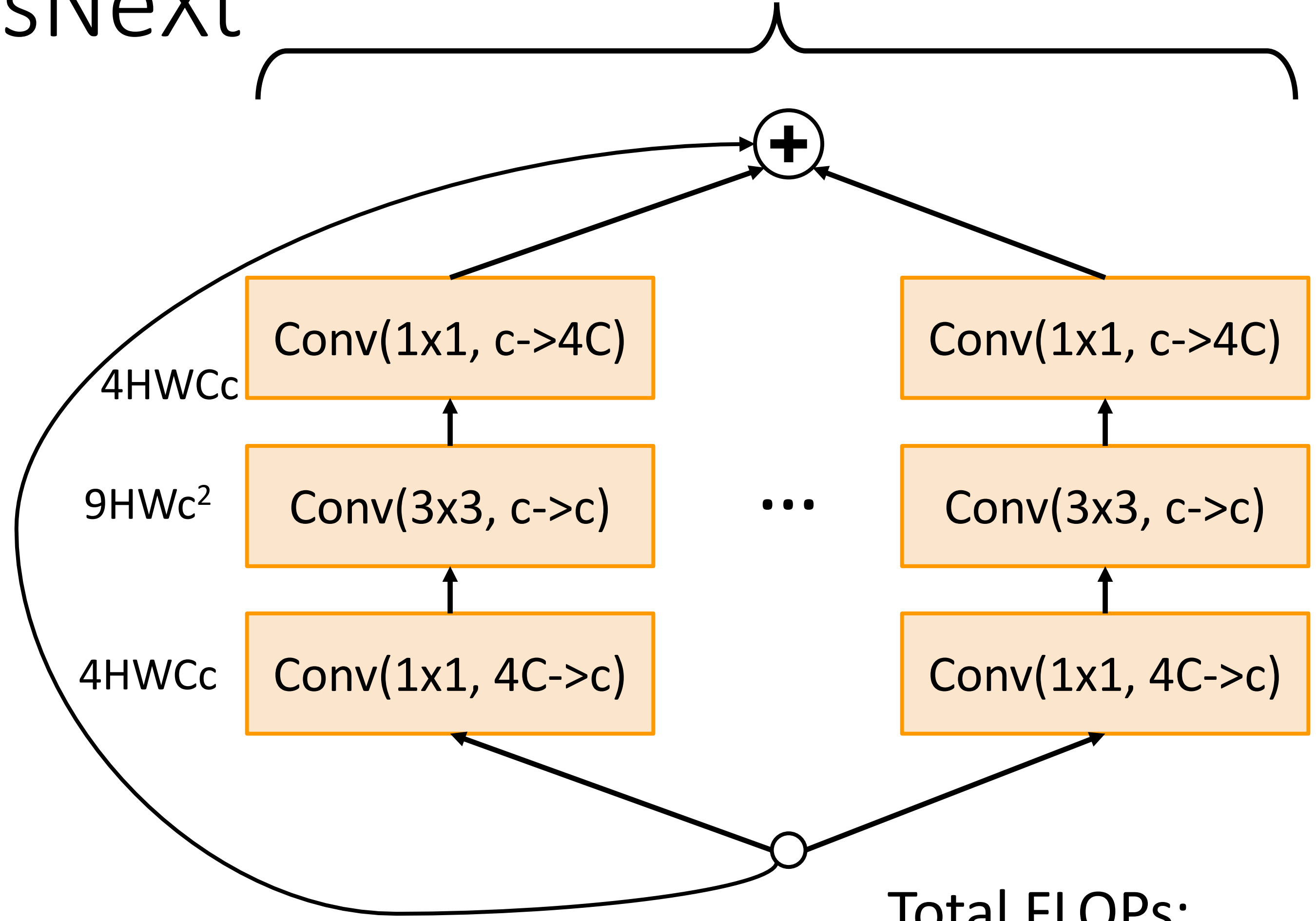
Improving ResNets: ResNeXt



"Bottleneck"
Residual block

Total FLOPs:
$17HWC^2$

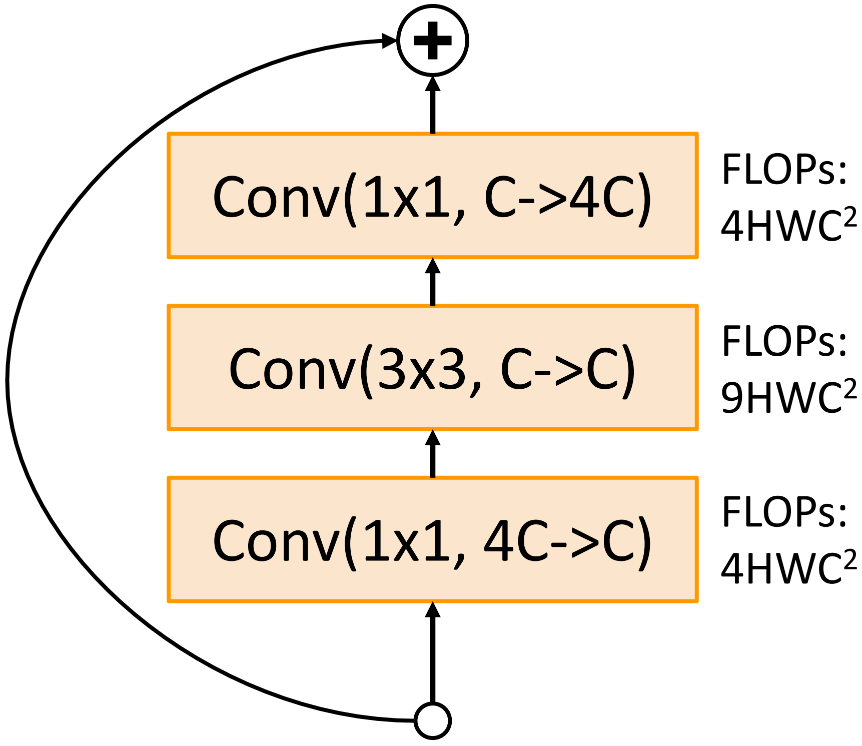
G parallel pathways



Total FLOPs:
$(8Cc + 9c^2)*HWG$

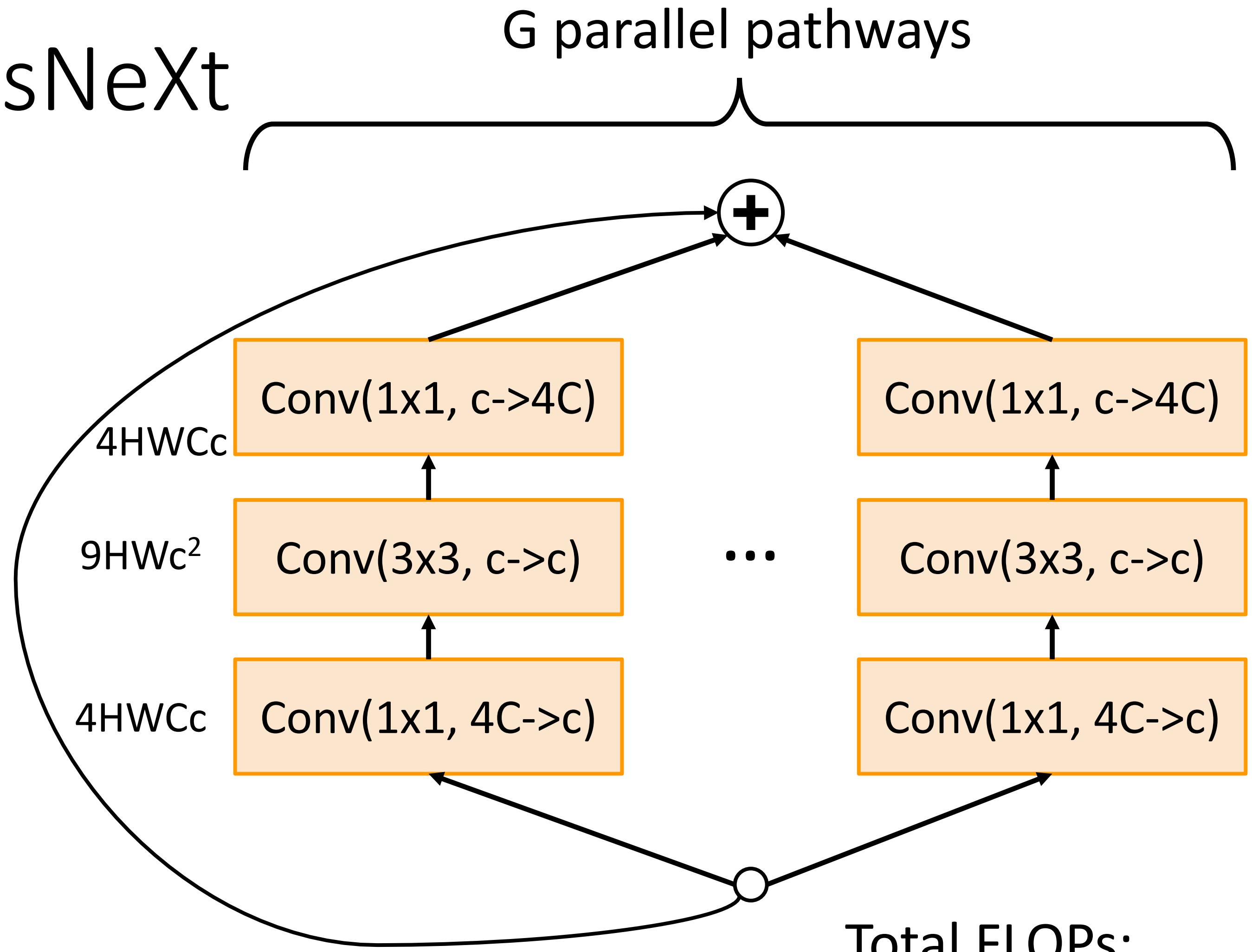
Xie et al, "Aggregated residual transformations for deep neural networks", CVPR 2017

Improving ResNets: ResNeXt



"Bottleneck"
Residual block

Total FLOPs:
$17HWC^2$



Total FLOPs:
$(8Cc + 9c^2)*HWG$

Equal cost when
$9Gc^2 + 8GCc - 17C^2 = 0$

Example: C=64, G=4, c=24; C=64, G=32, c=4

Xie et al, "Aggregated residual transformations for deep neural networks", CVPR 2017

Grouped Convolution

Convolution with groups=1:
Normal convolution

Input: C_{in} x H x W
Weight: C_{out} x C_{in} x K x K
Output: C_{out} x H' x W'
FLOPs: $C_{out}C_{in}K^2HW$

All convolutional kernels touch all C_{in} channels of the input

Grouped Convolution

Convolution with groups=1:
Normal convolution

Input: C_{in} x H x W
Weight: C_{out} x C_{in} x K x K
Output: C_{out} x H' x W'
FLOPs: $C_{out}C_{in}K^2HW$

All convolutional kernels touch all C_{in} channels of the input

Convolution with groups=2:
Two parallel convolution layers that work on half the channels

Input: C_{in} x H x W

Split

Group 1:
(C_{in} / 2) x H x W

Conv(K x K, C_{in}/2 -> C_{out}/2)

Out 1:
(C_{out} / 2) x H' x W'

Group 2:
(C_{in} / 2) x H x W

Conv(K x K, C_{in}/2 -> C_{out}/2)

Out 2:
(C_{out} / 2) x H' x W'

Concat

Output: C_{out} x H' x W'

Grouped Convolution

Convolution with groups=1:
Normal convolution

Input: C_{in} x H x W
Weight: C_{out} x C_{in} x K x K
Output: C_{out} x H' x W'
FLOPs: $C_{out}C_{in}K^2HW$

All convolutional kernels touch all C_{in} channels of the input

Convolution with groups=G:
G parallel conv layers; each "sees" C_{in}/G input channels and produces C_{out}/G output channels

Input: C_{in} x H x W
Split to G x [(C_{in} / G) x H x W]
Weight: G x (C_{out} / G) x (C_{in} x G) x K x K
G parallel convolutions
Output: G x [(C_{out} / G) x H' x W']
Concat to C_{out} x H' x W'
FLOPs: $C_{out}C_{in}K^2HW/G$

Grouped Convolution

<u>Convolution with groups=1</u>:
Normal convolution

Input: C_{in} x H x W
Weight: C_{out} x C_{in} x K x K
Output: C_{out} x H' x W'
FLOPs: $C_{out}C_{in}K^2HW$

All convolutional kernels touch all C_{in} channels of the input

Depthwise Convolution
Special case: $G=C_{in}$, $C_{out} = nC_{in}$
Each input channel is convolved with n different K x K filters to produce n output channels

<u>Convolution with groups=G</u>:
G parallel conv layers; each "sees" C_{in}/G input channels and produces C_{out}/G output channels

Input: C_{in} x H x W
Split to G x [(C_{in} / G) x H x W]
Weight: G x (C_{out} / G) x (C_{in} x G) x K x K
G parallel convolutions
Output: G x [(C_{out} / G) x H' x W']
Concat to C_{out} x H' x W'
FLOPs: $C_{out}C_{in}K^2HW/G$

Grouped Convolution in PyTorch

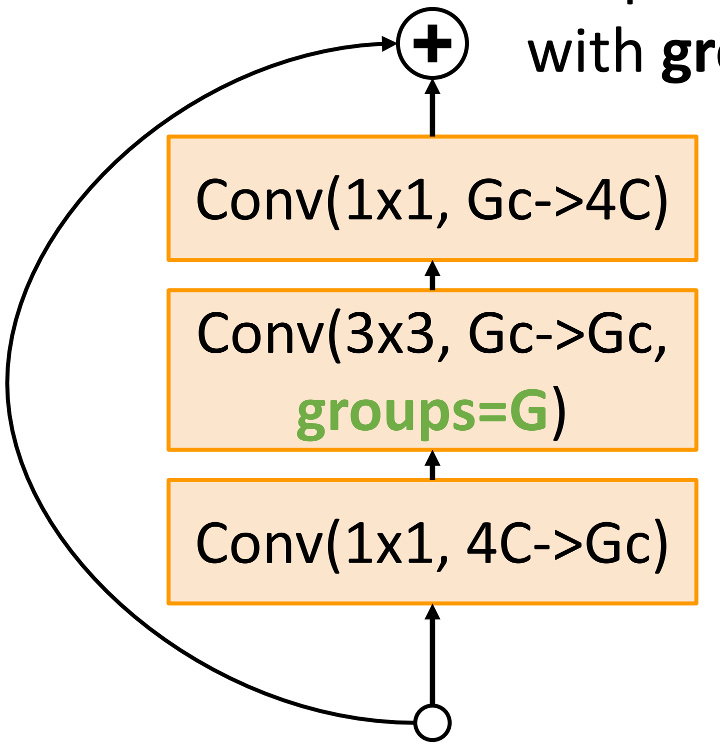
PyTorch convolution gives an option for groups!

Conv2d

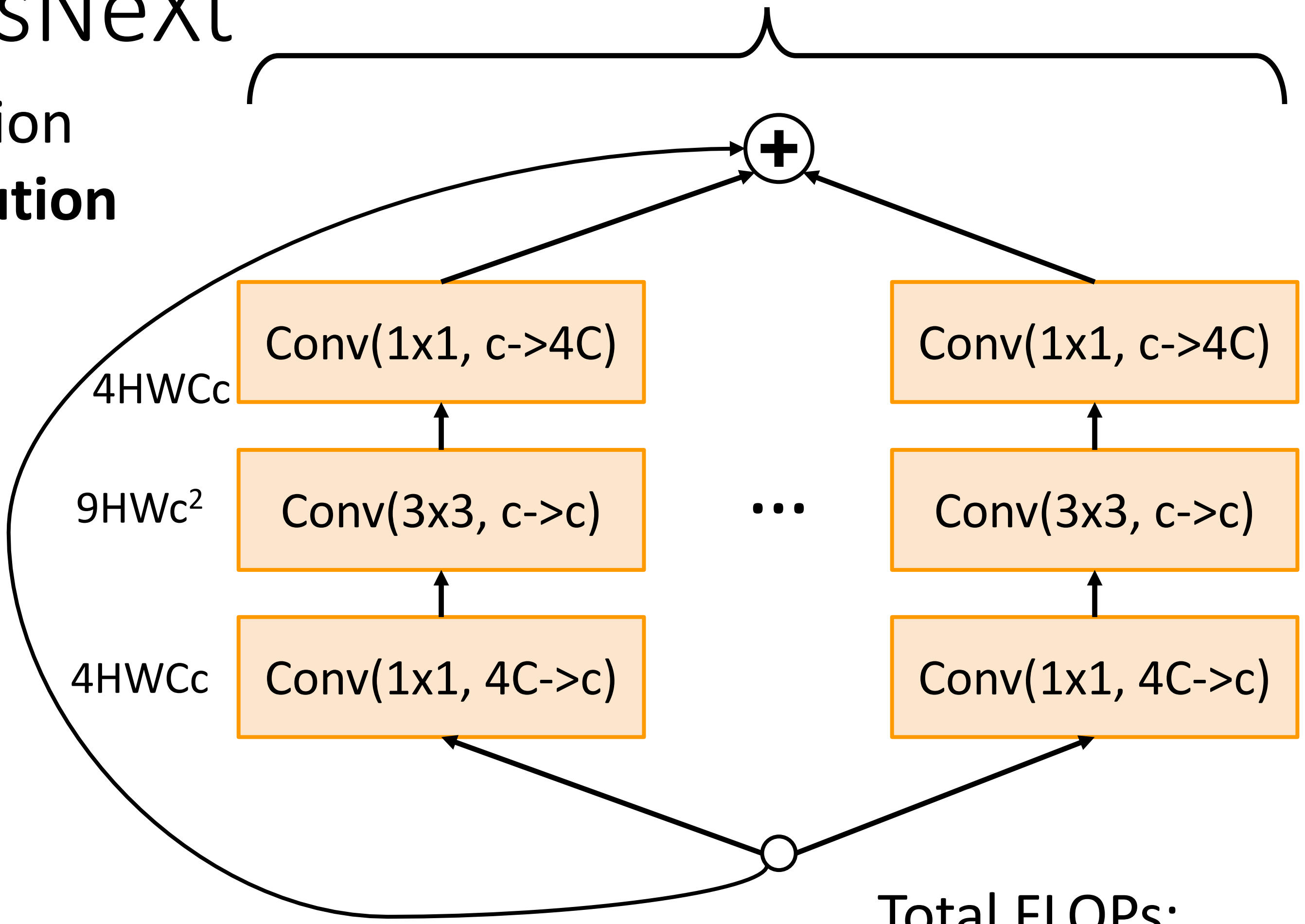
CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [SOURCE]

Improving ResNets: ResNeXt

Equivalent formulation with **grouped convolution**



ResNeXt block:
Grouped convolution



Total FLOPs:
$(8Cc + 9c^2)*HWG$

Equal cost when
$9Gc^2 + 8GCc - 17C^2 = 0$

Example: C=64, G=4, c=24; C=64, G=32, c=4

Xie et al, "Aggregated residual transformations for deep neural networks", CVPR 2017

ResNeXt: Maintain computation by adding groups!

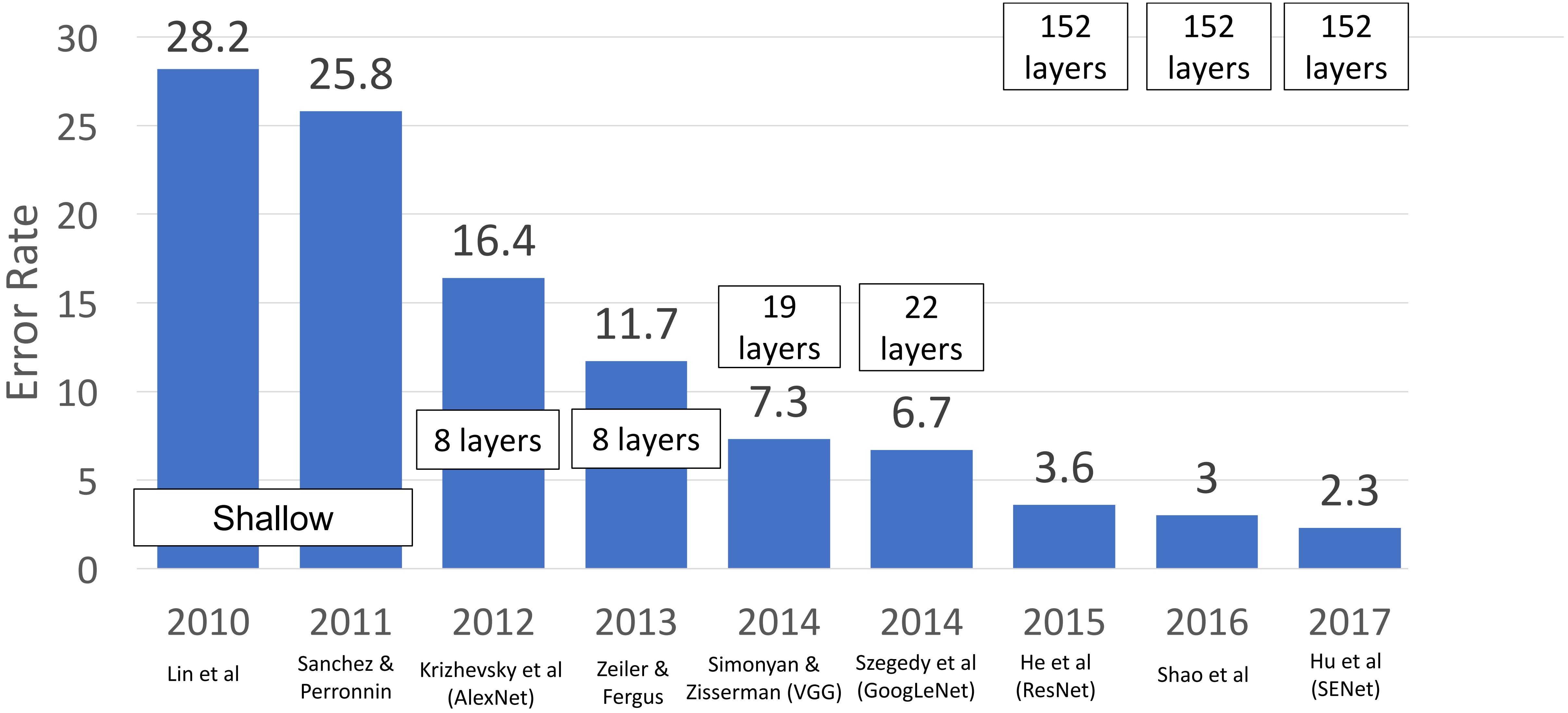
Model	Groups	Group width	Top-1 Error
ResNet-50	1	64	23.9
ResNeXt-50	2	40	23
ResNeXt-50	4	24	22.6
ResNeXt-50	8	14	22.3
ResNeXt-50	32	4	22.2

Model	Groups	Group width	Top-1 Error
ResNet-101	1	64	22.0
ResNeXt-101	2	40	21.7
ResNeXt-101	4	24	21.4
ResNeXt-101	8	14	21.3
ResNeXt-101	32	4	21.2

Adding groups improves performance **with same computational complexity!**

Xie et al, "Aggregated residual transformations for deep neural networks", CVPR 2017

ImageNet Classification Challenge

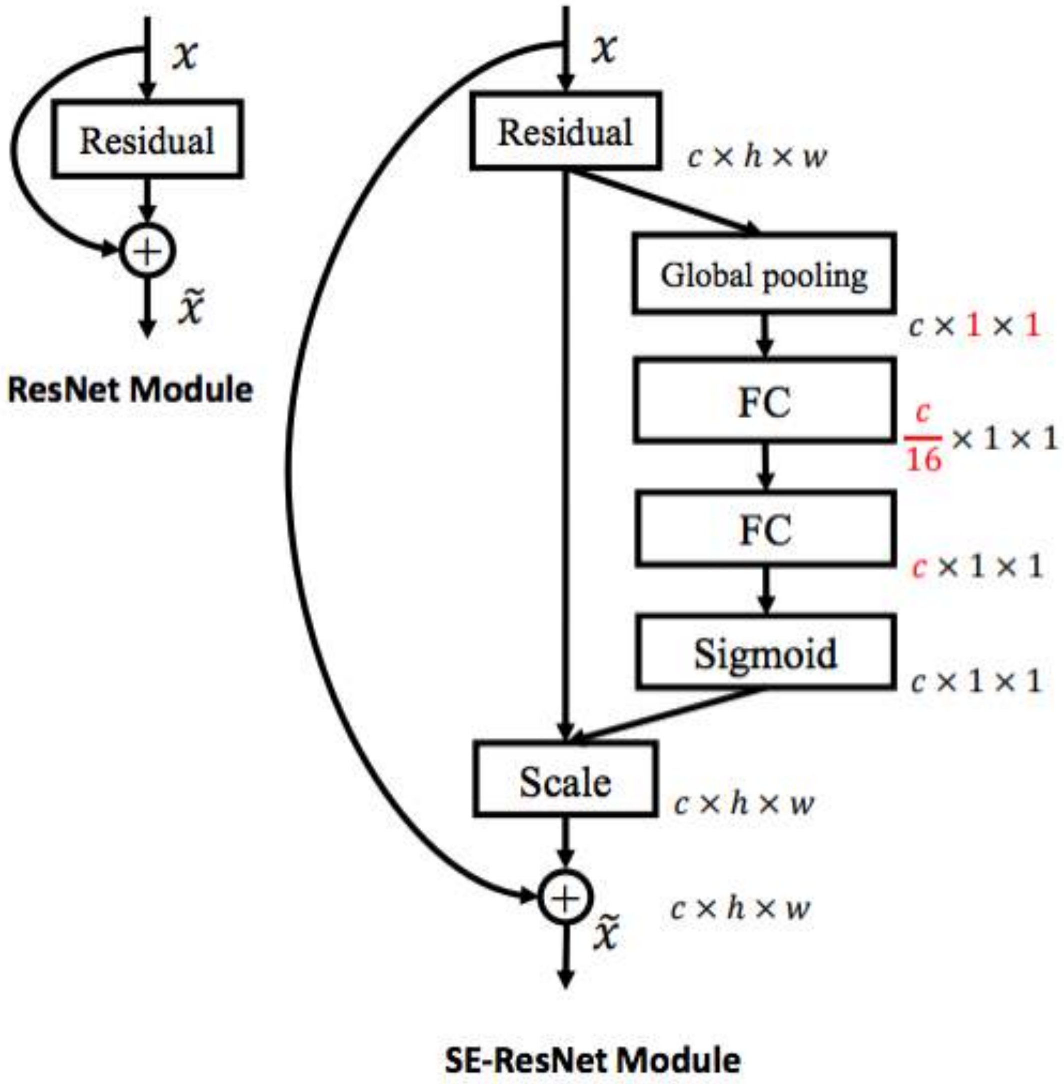


Squeeze-and-Excitation Networks

Adds a ”Squeeze-and-excite” branch to each residual block that performs global pooling, full-connected layers, and multiplies back onto feature map

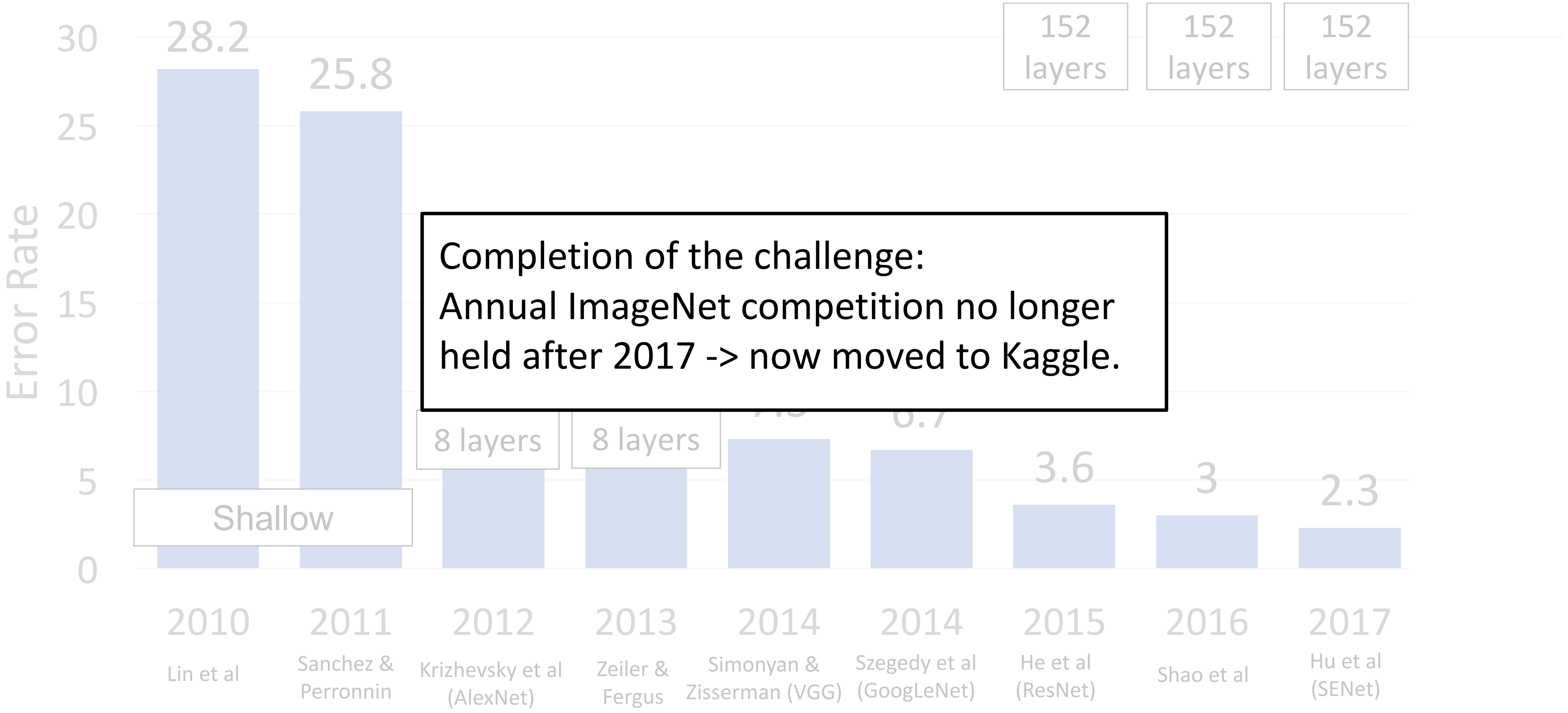
Adds **global context** to each residual block!

Won ILSVRC 2017 with ResNeXt-152-SE



Hu et al, “Squeeze-and-Excitation networks”, CVPR 2018

ImageNet Classification Challenge

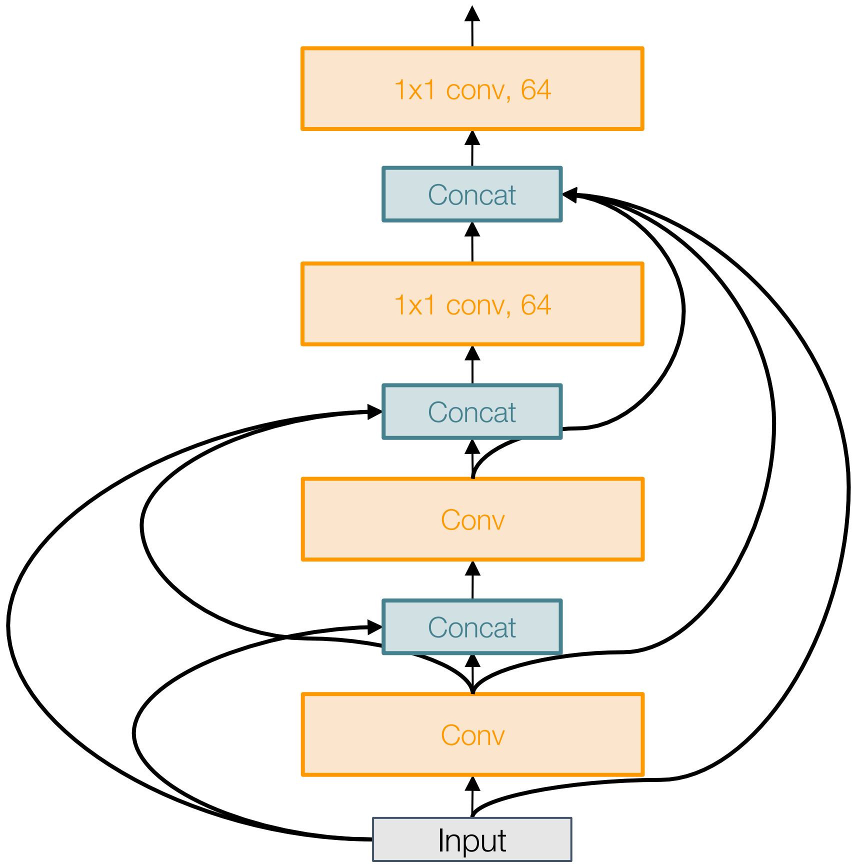


Completion of the challenge:
Annual ImageNet competition no longer held after 2017 -> now moved to Kaggle.

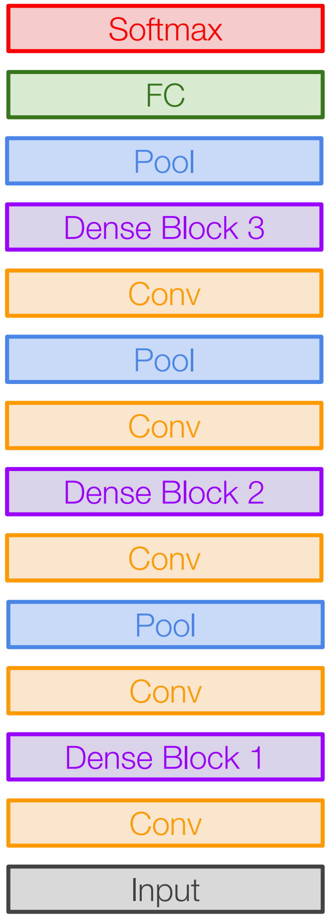
Densely Connected Neural Networks

Dense blocks where each layer is connected to every other layer in feedforward fashion

Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse



Dense Block

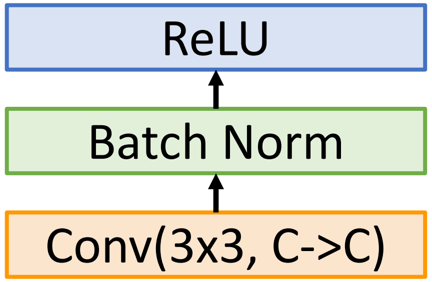


Huang et al, "Densely connected neural networks", CVPR 2017

MobileNets: Tiny Networks (For Mobile Devices)

Standard Convolution Block

Total cost: $9C^2HW$

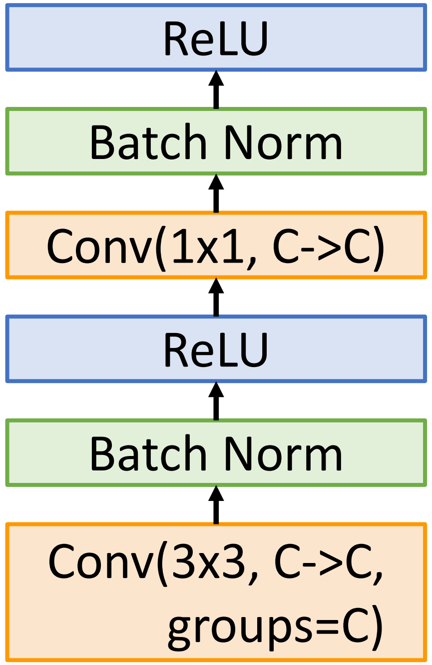


$9C^2HW$

$$\text{Speedup} = 9C^2/(9C+C^2)$$
$$= 9C/(9+C)$$
$$\Rightarrow 9 \text{ (as C->inf)}$$

Depthwise Separable Convolution

Total cost: $(9C + C^2)HW$



C^2HW "Pointwise Convolution"

$9CHW$ "Depthwise Convolution"

Howard et al, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", 2017

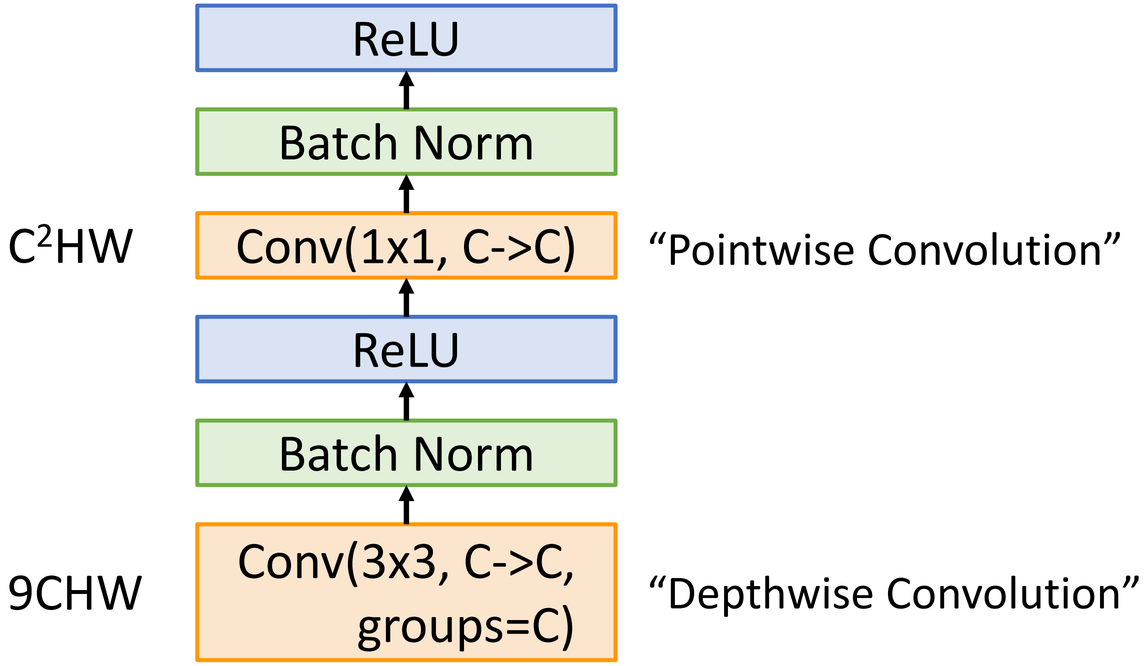
MobileNets: Tiny Networks (For Mobile Devices)

Also related:

ShuffleNet: Zhang et al, CVPR 2018
MobileNetV2: Sandler et al, CVPR 2018
ShuffleNetV2: Ma et al, ECCV 2018

Depthwise Separable Convolution

Total cost: $(9C + C^2)HW$

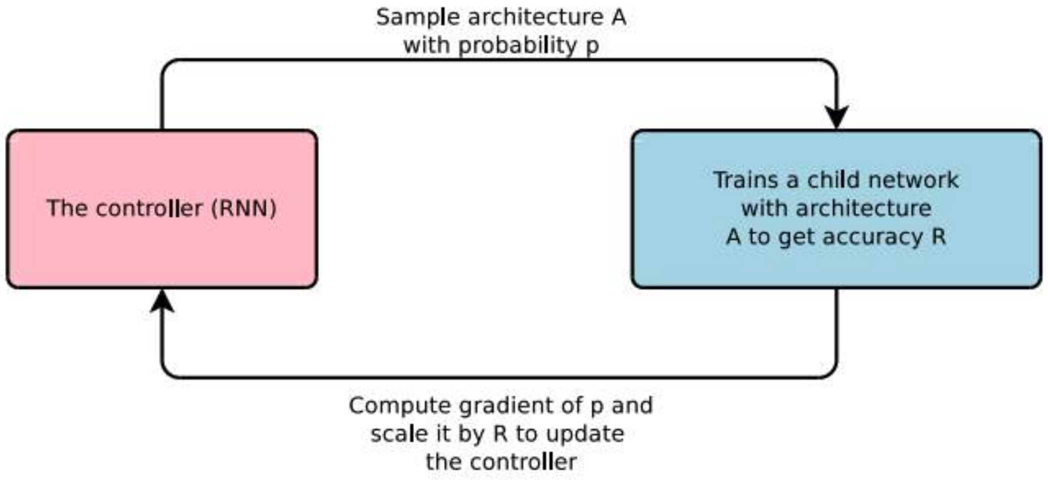


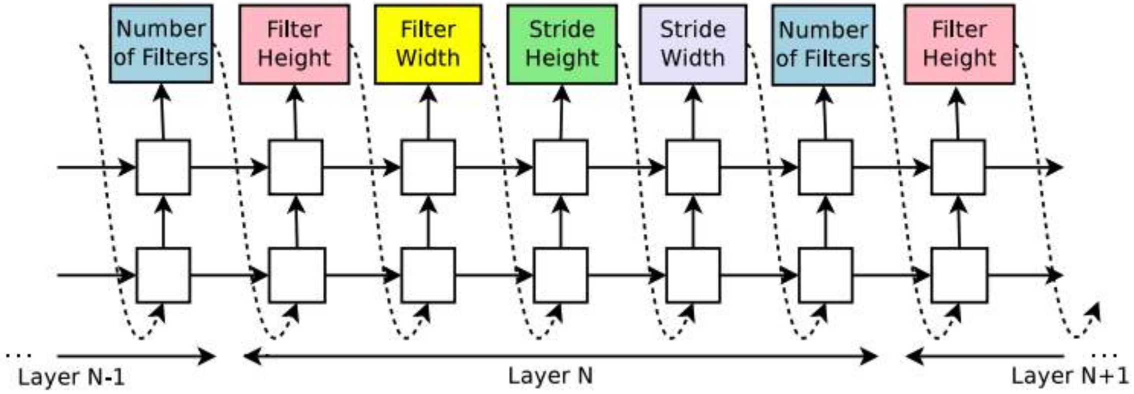
Howard et al, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", 2017

Neural Architecture Search

Designing neural network architectures is hard – let's automate it!

- One network (**controller**) outputs network architectures
- Sample **child networks** from controller and train them
- After training a batch of child networks, make a gradient step on controller network (Using **policy gradient**)
- Over time, controller learns to output good architectures!



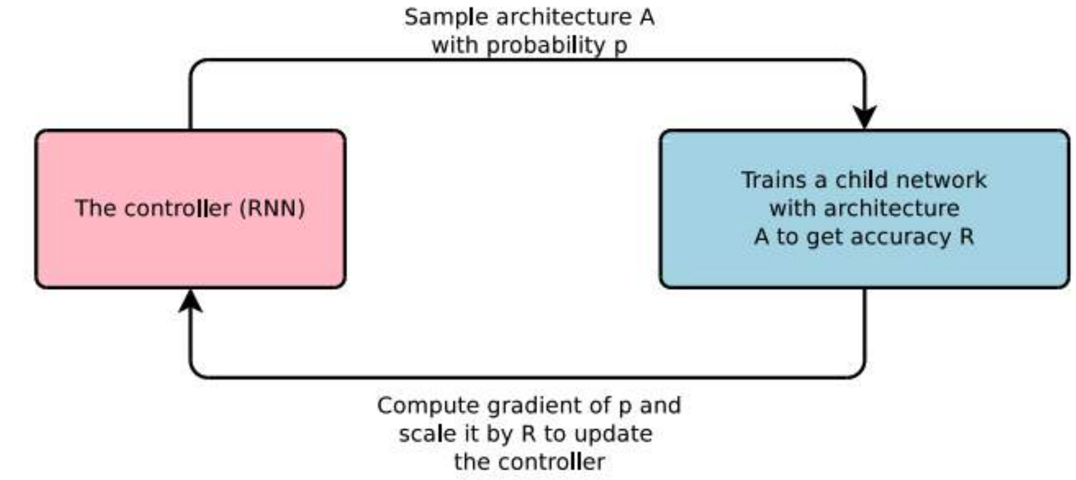


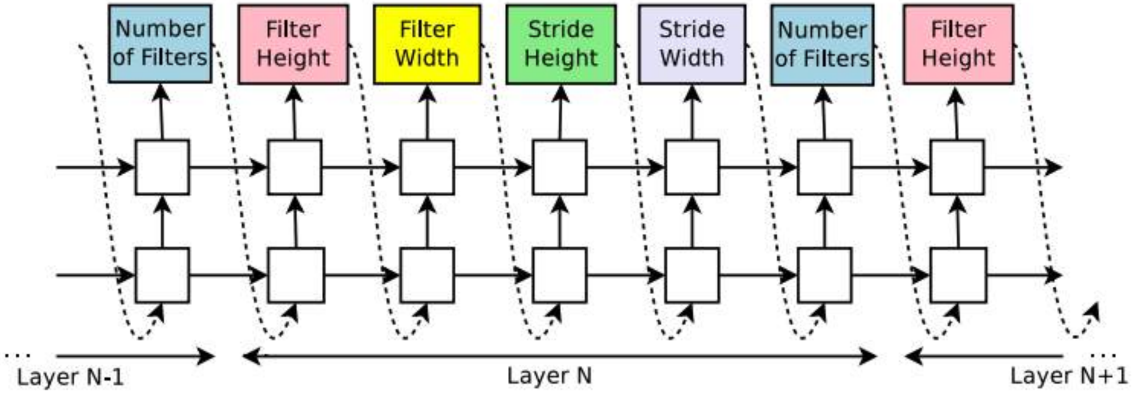
Zoph and Le, "Neural Architecture Search with Reinforcement Learning", ICLR 2017

Neural Architecture Search

Designing neural network architectures is hard – let's automate it!

- One network (**controller**) outputs network architectures
- Sample **child networks** from controller and train them
- After training a batch of child networks, make a gradient step on controller network (Using **policy gradient**)
- Over time, controller learns to output good architectures!
- VERY EXPENSIVE!! Each gradient step on controller requires training a batch of child models!
- Original paper trained on 800 GPUs for 28 days!
- Followup work has focused on efficient search

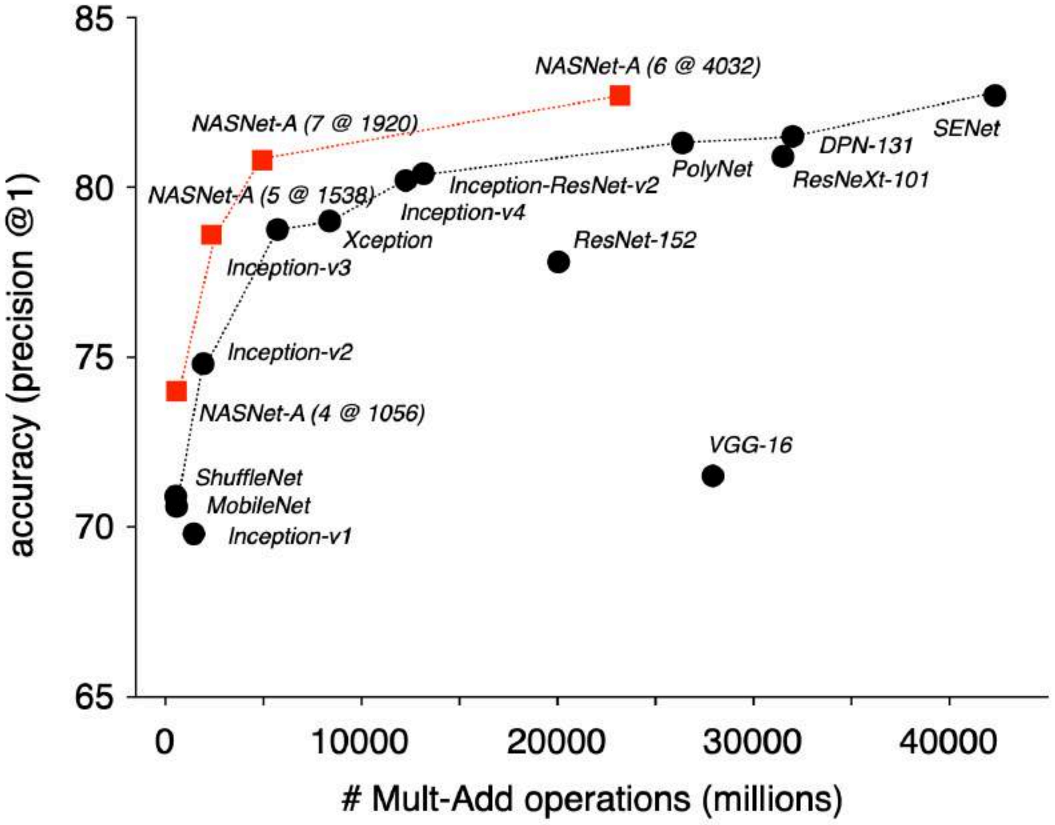


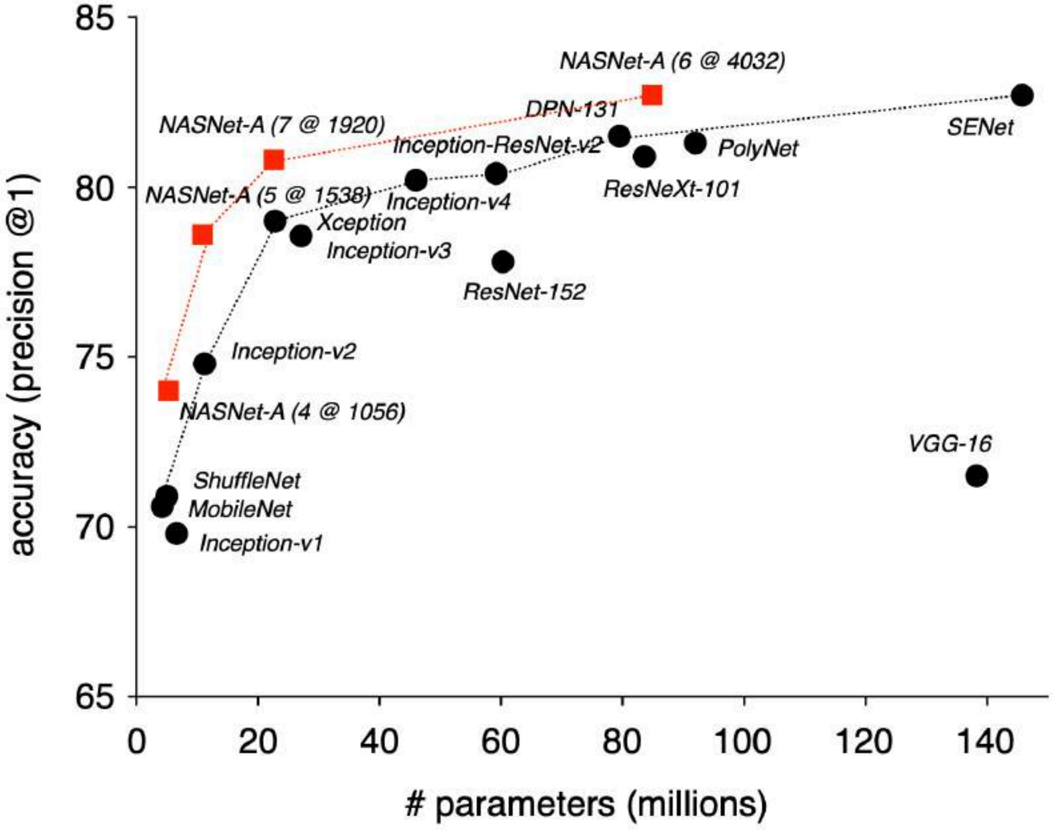


Zoph and Le, "Neural Architecture Search with Reinforcement Learning", ICLR 2017

Neural Architecture Search

Neural architecture search can be used to find efficient CNN architectures!





Zoph et al, “Learning Transferable Architectures for Scalable Image Recognition”, CVPR 2018

CNN Architectures Summary

Early work (AlexNet -> ZFNet -> VGG) shows that **bigger networks work better**

GoogLeNet one of the first to focus on **efficiency** (aggressive stem, 1x1 bottleneck convolutions, global avg pool instead of FC layers)

ResNet showed us how to train extremely deep networks – limited only by GPU memory! Started to show diminishing returns as networks got bigger

After ResNet: **Efficient networks** became central: how can we improve the accuracy without increasing the complexity?

Lots of **tiny networks** aimed at mobile devices: MobileNet, ShuffleNet, etc

Neural Architecture Search promises to automate architecture design

Which Architecture should I use?

Don’t be a hero. For most problems you should use an off-the-shelf architecture; don’t try to design your own!

If you just care about accuracy, **ResNet-50** or **ResNet-101** are great choices

If you want an efficient network (real-time, run on mobile, etc) try **MobileNets** and **ShuffleNets**

Next Time:
Deep Learning Hardware and Software