20. CNN Architectures

STA3142 Statistical Machine Learning

Kibok Lee

Assistant Professor of

Applied Statistics / Statistics and Data Science

May 28, 2024

** Slides adapted from EECS498/598 @ Univ. of Michigan by Justin Johnson*

연세대학교
YONSEI UNIVERSITY

Assignment 5 (Final Exam Replacement)

- Due **Friday 6/14, 11:59pm**
- Topic: Convolutional Neural Networks
  - Derive gradients for NN layers
  - Implement layers for CNNs
  - Train a CNN classifier for MNIST digit recognition
- Please read the instruction carefully!
  - Submit one pdf and one zip file separately
  - Write your code only in the designated spaces
  - Do not import additional libraries
  - …
- If you feel difficult, consider to take option 2.

Assignment Options

- **Option 1**: Submit your own work
  - You must not copy/refer to others' solution/code.
  - You will get **[F]** if any plagiarism is found.
- **Option 2**: Refer to others' solution/code
  - You need to **cite** references clearly.
    - Website address, query to LLMs, your study group members
    - **[No point]** if you do not provide references properly
  - For codes, **your comments** is required/graded.
    - **[No point]** if you do not provide any comments
  - Your score will be downscaled to **[70%]**.
    - e.g., if you take this option for 3rd, 4th, and the final assignments, the maximum score you can get is 12.5*(2 + 2*.7) + 15*.7 = 53 (out of 65)
  - You can apply this question-wise (Not subquestion-wise)

Can I use large language models?

- You can get an assistance by large language models (LLMs) like ChatGPT, which falls into **option 2**.
- Clearly note that you used LLMs and provide your queries together.
- This policy can be changed for later assignments.
  - If we found it makes you complete assignments without understanding

Note on Cheating

- If your solution is affected by LLM or other references, it falls into **option 2**.
  - This is cheating, but we allow you to have such assistance given that you properly acknowledge it and show your understanding with some comments.
- Acknowledge your plagiarism with references to **TAs via LearnUS DM by Fri, May 31**
  - If you do so, you will get 0 point for the corresponding assignments
  - If not, you will get F for this course
- There are several cases we found:
  - Your solution is too similar to LLM-generated texts or other references
    - Note: This course is about the basics of ML/DL, and LLM is a direct application of ML/DL
  - Your solution ignores the instruction too much

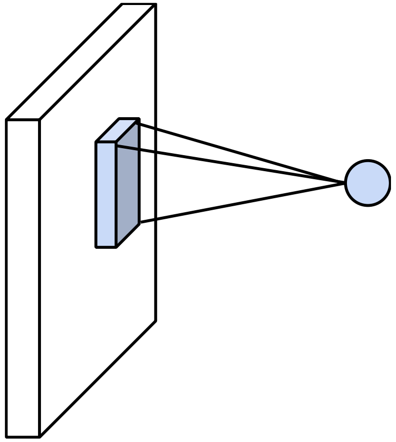
Note on Cheating

- Based on the honor code of this university, students who fake attendance receive **0 point (= F grade)** for the course and their **academic position is suspended** for a period of time.
  - Kor: https://www.yonsei.ac.kr/sc/support/training_course11.jsp
  - Eng: https://www.yonsei.ac.kr/en_sc/admission/academicregulations6.jsp

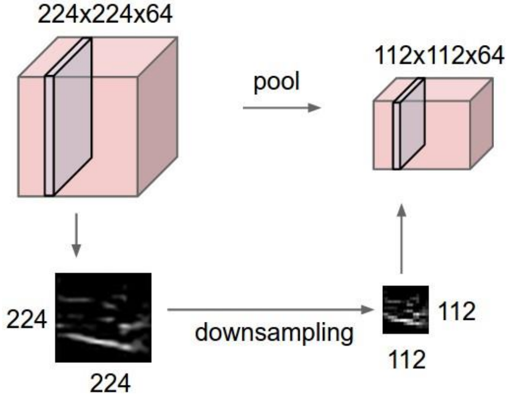
- Assuming that you didn't know about the academic honor code well, we will apply the following rule:
  - If you've faked attendance at least once so far, you have **no free absence** when calculating the attendance score.
  - From now on, if you fake attendance, you will get **F** for this course.

Recall: Components of Convolutional Networks

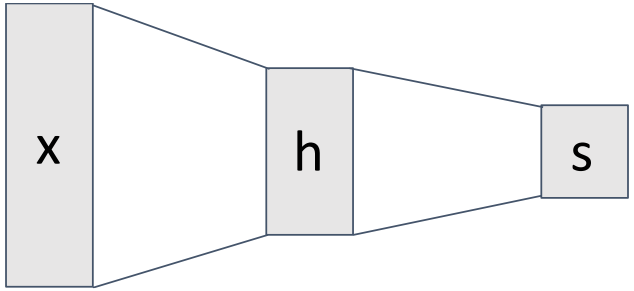
Convolution Layers



Pooling Layers

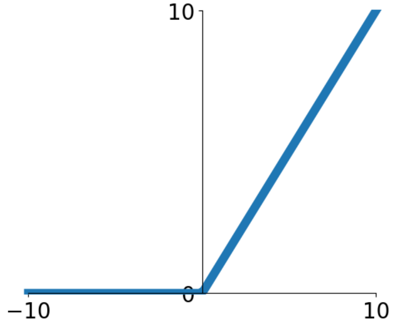


Fully-Connected Layers



Question: How should we put them together?

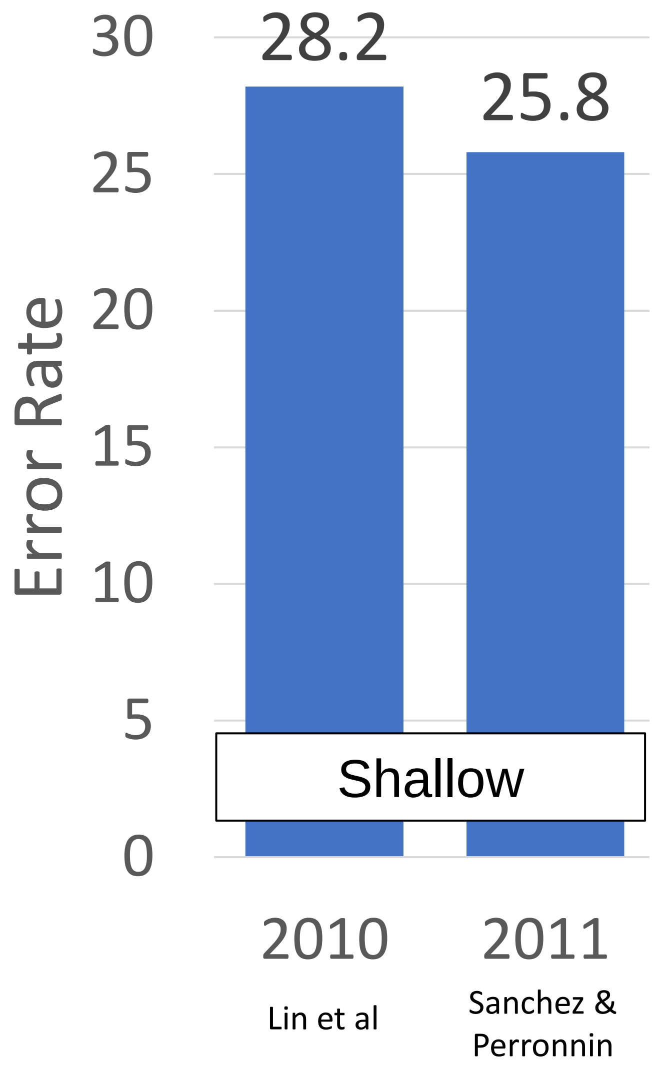
Activation Function



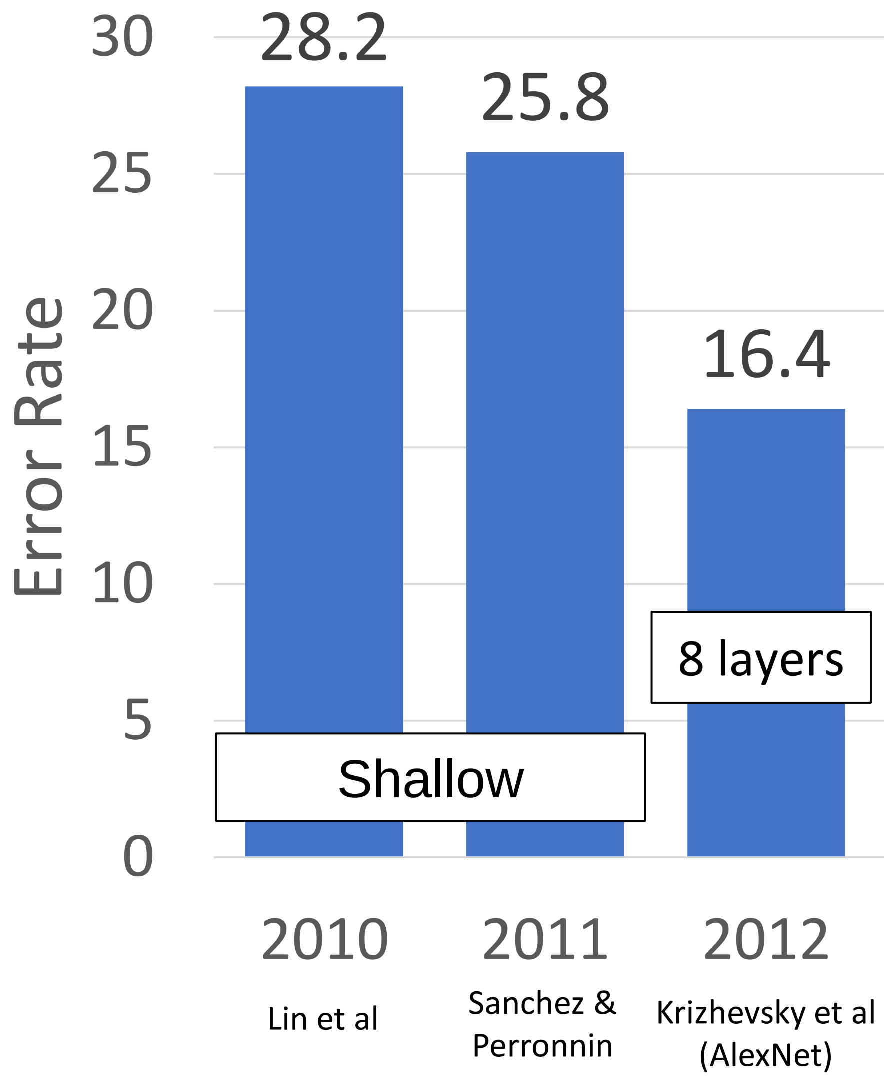
Normalization

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

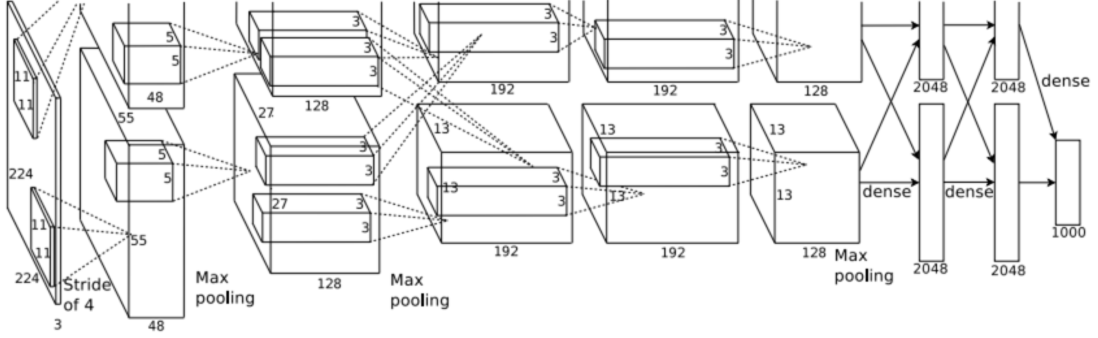
ImageNet Classification Challenge



ImageNet Classification Challenge



AlexNet

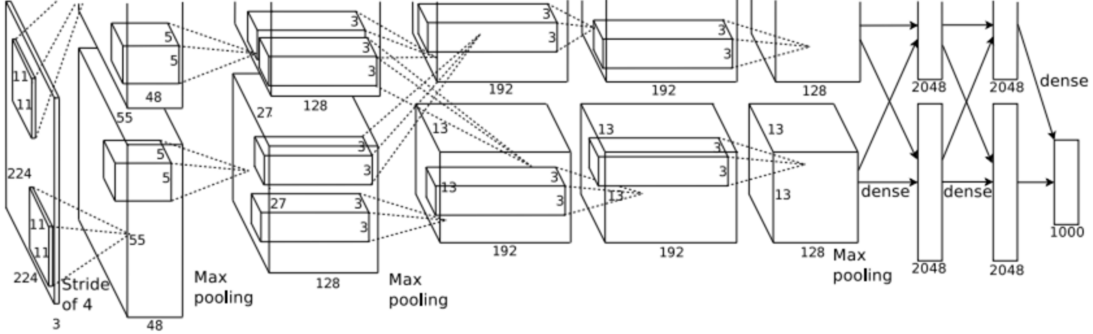


227 x 227 inputs
5 Convolutional layers
Max pooling
3 fully-connected layers
ReLU nonlinearities

Used “Local response normalization”; Not used anymore

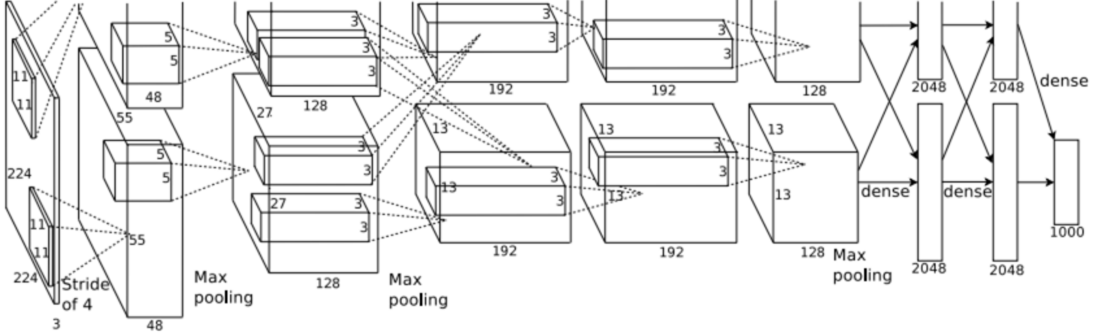
Trained on two GTX 580 GPUs – only 3GB of memory each! Model split over two GPUs

AlexNet



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

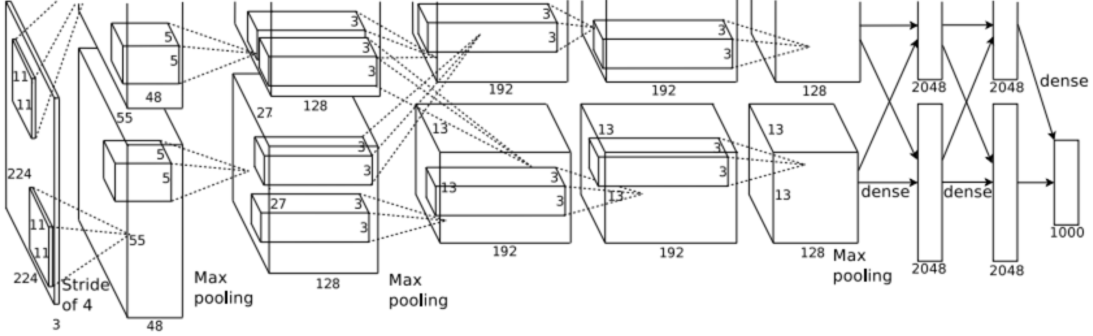
AlexNet



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

Recall: Output channels = number of filters

AlexNet



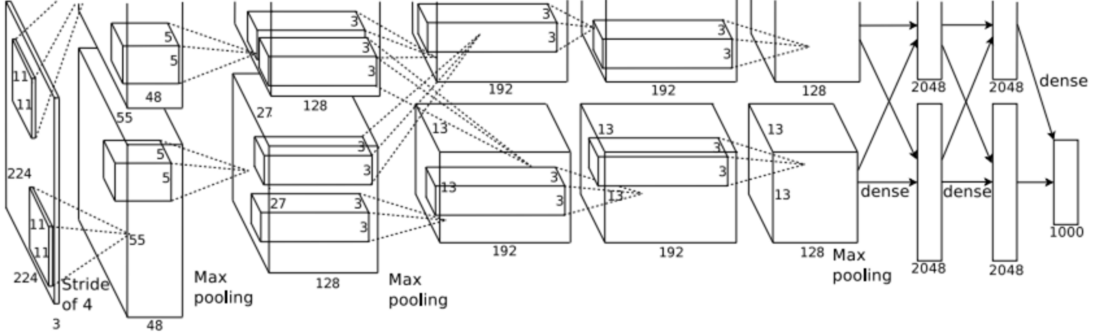
	Input size		Layer				Output size	
Layer	C	H / W	filters	KH / KW	stride	pad	C	H / W
conv1	3	227	64	11	4	2	64	56

Recall: $W' = (W - K + 2P) / S + 1$

$= (227 - 11 + 2*2) / 4 + 1$

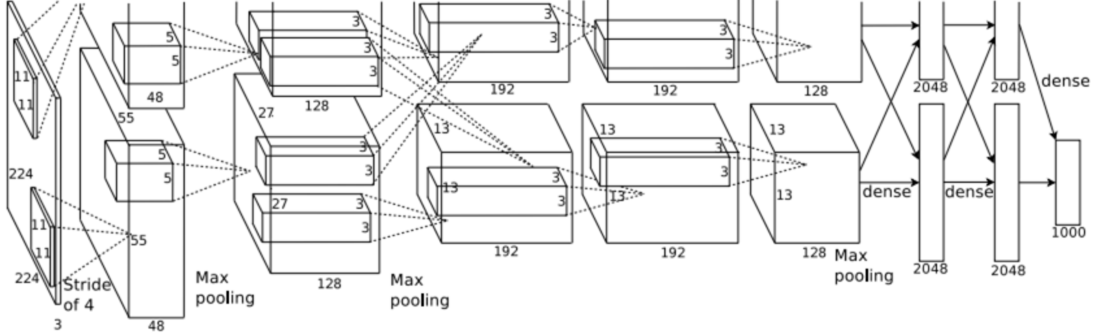
$= 220/4 + 1 = 56$

AlexNet



	Input size		Layer				Output size		
Layer	C	H / W	filters	KH / KW	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	KH / KW	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

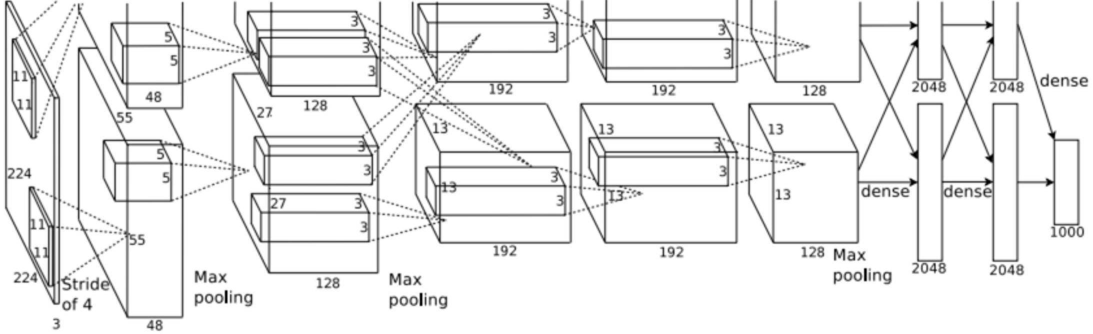
Bytes per element = 4 (for 32-bit floating point)

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

= 200704 * 4 / 1024

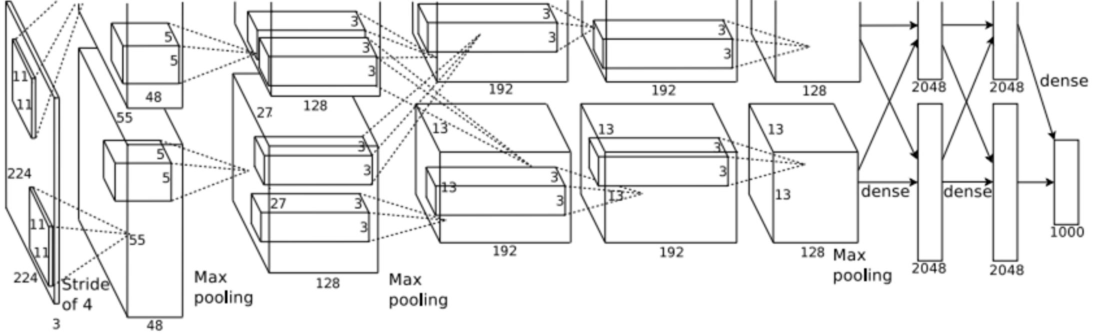
= **784**

AlexNet



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

AlexNet



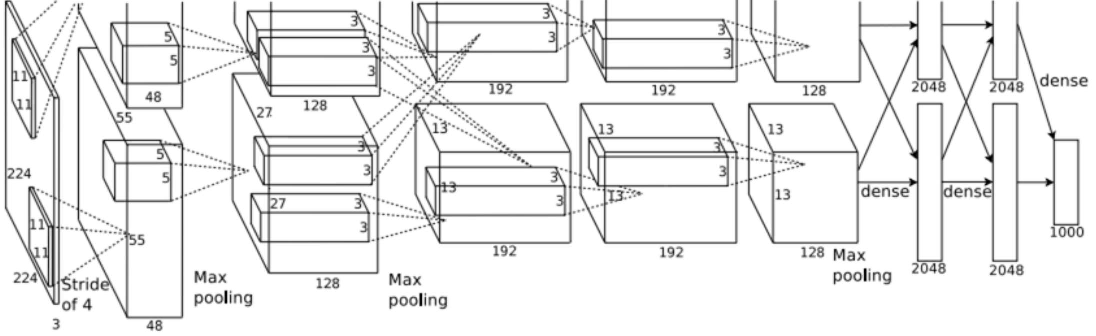
	Input size		Layer				Output size			
Layer	C	H / W	filters	KH / KW	stride	pad	C	H / W	memory (KB)	params (k)
conv1	3	227	64	11	4	2	64	56	784	23

Weight shape = $C_{out} \times C_{in} \times K \times K$
$= 64 \times 3 \times 11 \times 11$

Bias shape = $C_{out} = 64$

Number of weights = $64*3*11*11 + 64$
$= \mathbf{23{,}296}$

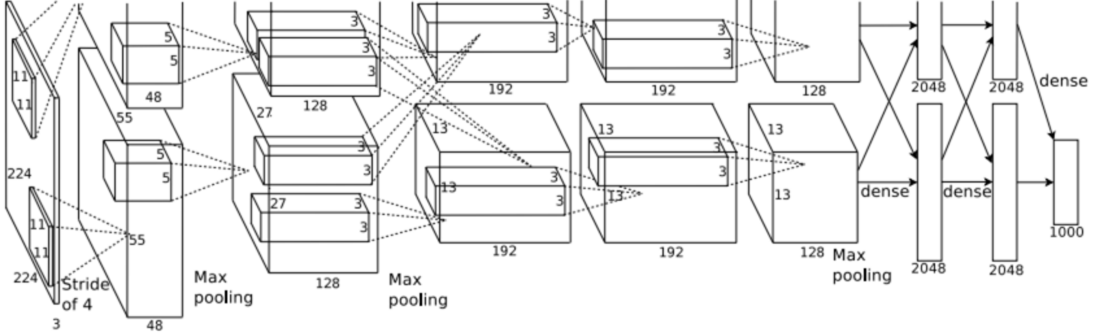
AlexNet



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

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

AlexNet



	Input size		Layer				Output size				
Layer	C	H / W	filters	KH / KW	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)

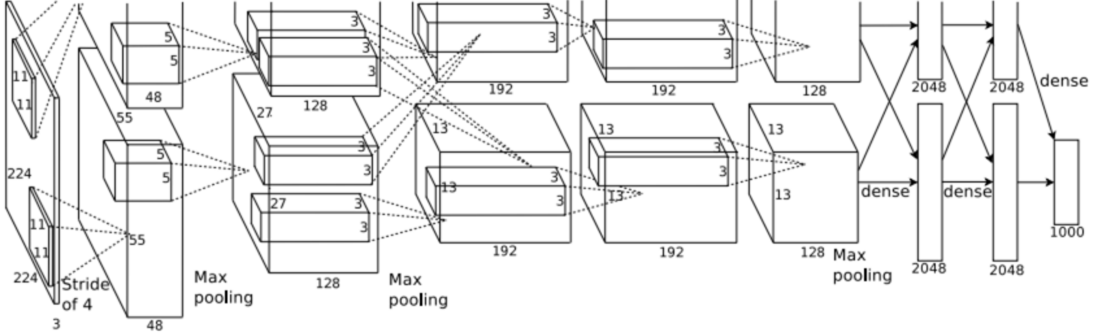
= $(C_{out} \times H' \times W') * (C_{in} \times K \times K)$

= (64 * 56 * 56) * (3 * 11 * 11)

= 200,704 * 363

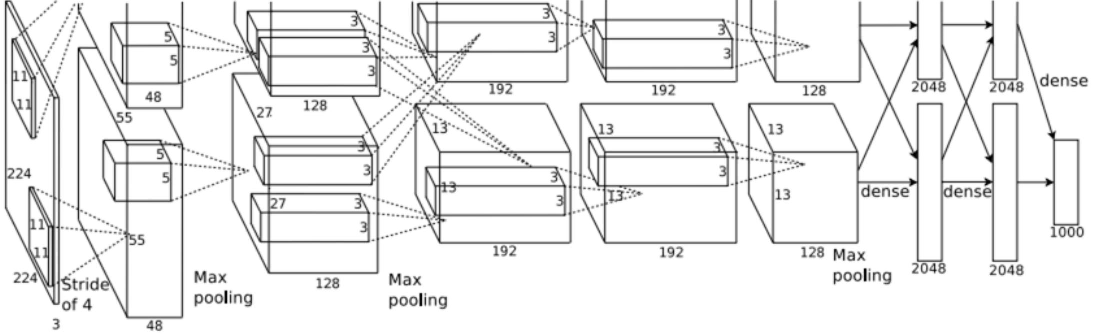
= **72,855,552**

AlexNet



	Input size		Layer				Output size				
Layer	C	H / W	filters	KH / KW	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



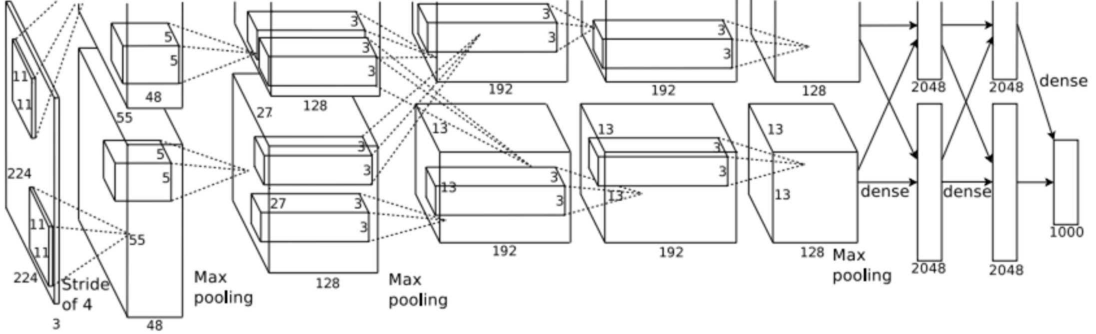
	Input size		Layer				Output size				
Layer	C	H / W	filters	KH / KW	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27			

For pooling layer:

#output channels = #input channels = 64

W' = floor((W – K) / S + 1)
= floor(53 / 2 + 1) = floor(27.5) = **27**

AlexNet



	Input size		Layer				Output size				
Layer	C	H / W	filters	KH / KW	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'

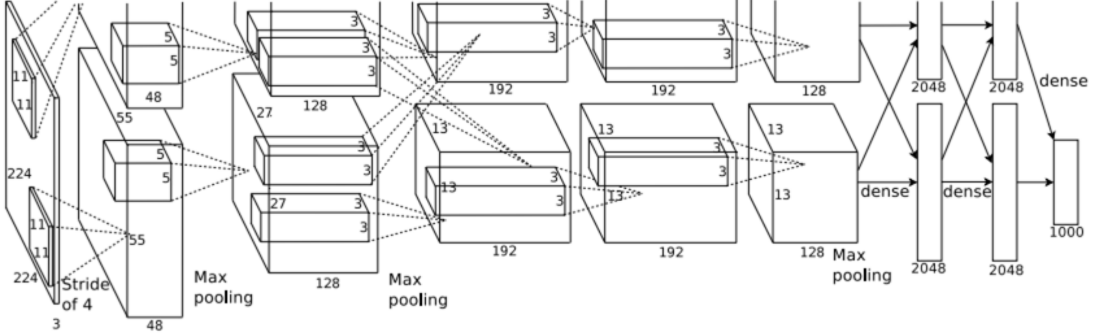
Bytes per elem = 4

KB = C_{out} * H' * W' * 4 / 1024

= 64 * 27 * 27 * 4 / 1024

= **182.25**

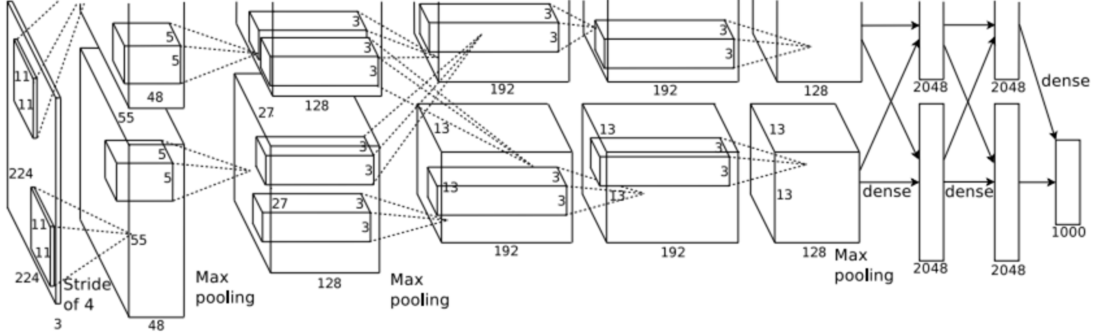
AlexNet



	Input size		Layer				Output size				
Layer	C	H / W	filters	KH / KW	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	KH / KW	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)

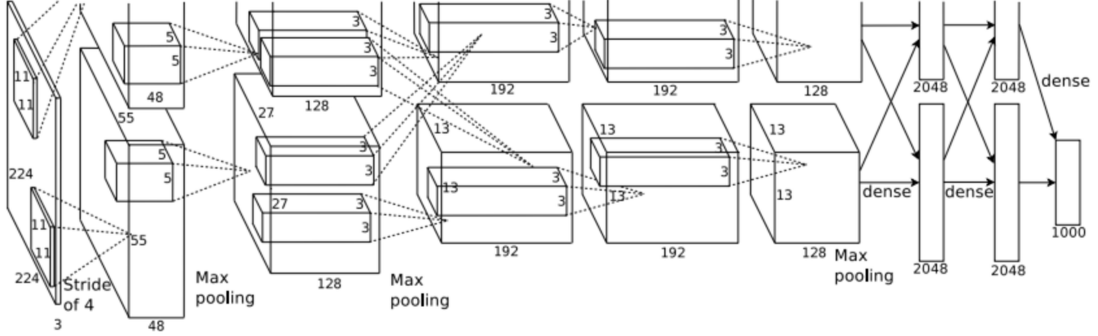
= $(C_{out} * H' * W') * (K * K)$

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

= 419,904

= **0.4 MFLOP**

AlexNet



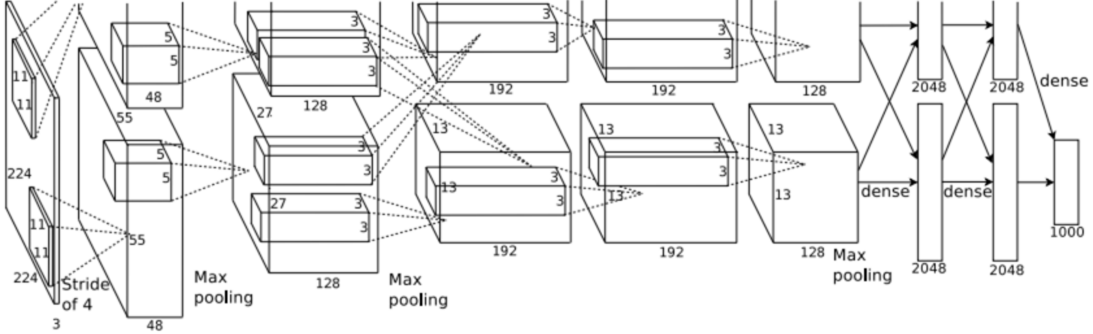
	Input size		Layer				Output size				
Layer	C	H / W	filters	KH / KW	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664	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

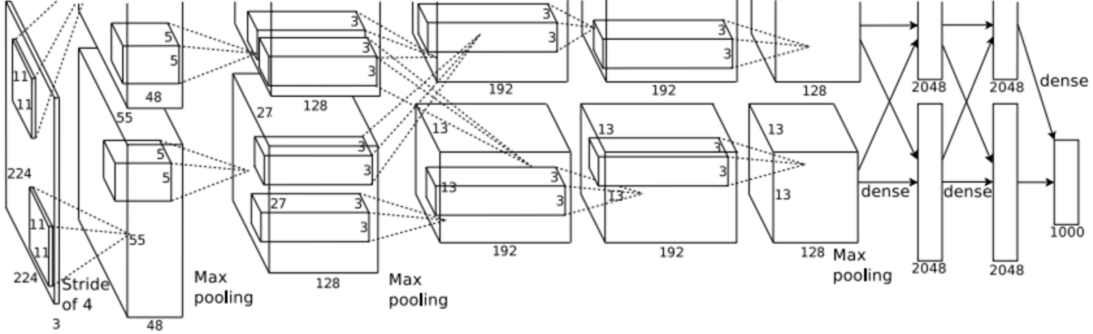


	Input size		Layer				Output size				
Layer	C	H / W	filters	KH / KW	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664	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	37,753	38

FC params $= C_{in} * C_{out} + C_{out}$
$= 9216 * 4096 + 4096$
$= 37{,}752{,}832$

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

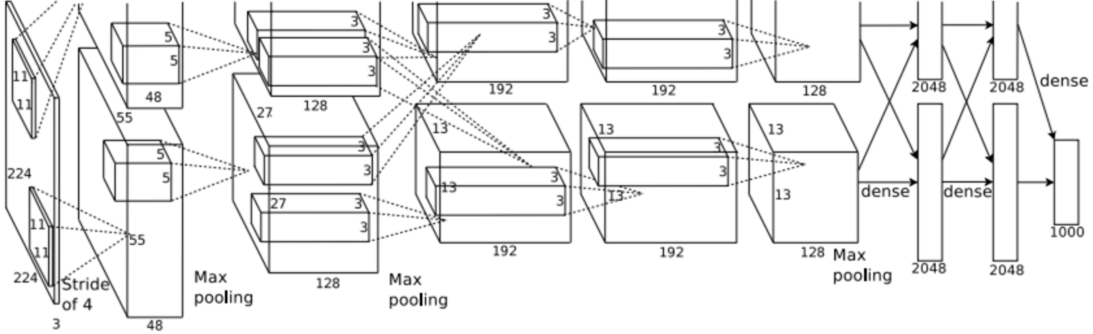
AlexNet



	Input size		Layer				Output size				
Layer	C	H / W	filters	KH / KW	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	37,753	38
fc7	4096		4096				4096		16	16,781	17
fc8	4096		1000				1000		4	4,097	4

AlexNet

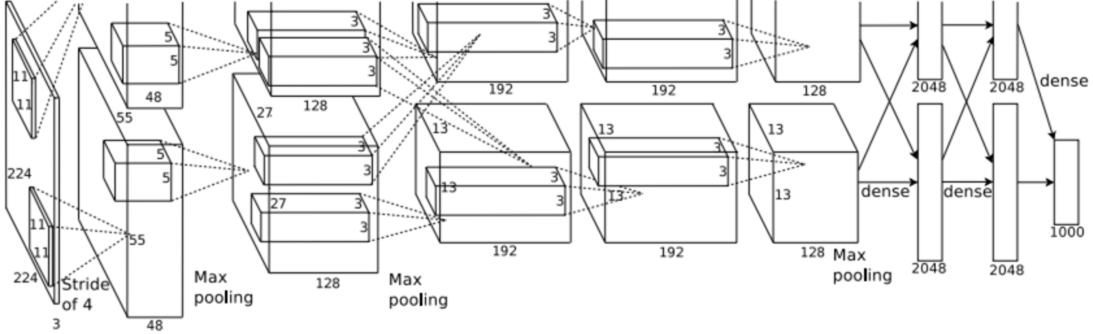
How to choose this?
Trial and error =(



	Input size		Layer				Output size				
Layer	C	H / W	filters	KH / KW	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	37,753	38
fc7	4096		4096				4096		16	16,781	17
fc8	4096		1000				1000		4	4,097	4

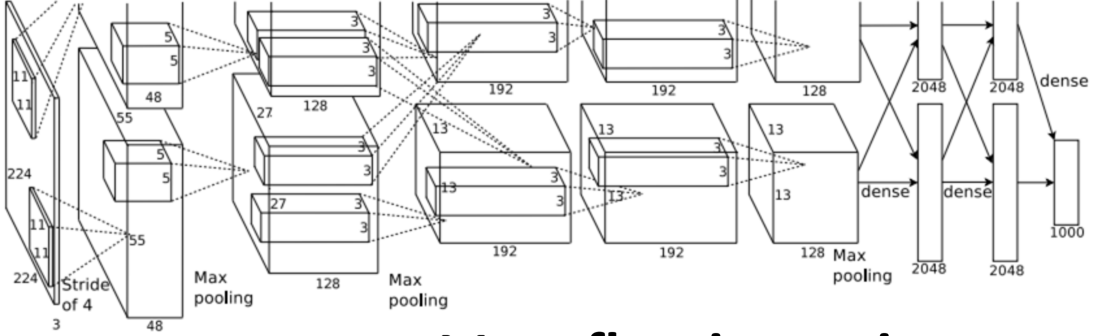
AlexNet

Interesting trends here!



	Input size		Layer				Output size				
Layer	C	H / W	filters	KH / KW	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	37,753	38
fc7	4096		4096				4096		16	16,781	17
fc8	4096		1000				1000		4	4,097	4

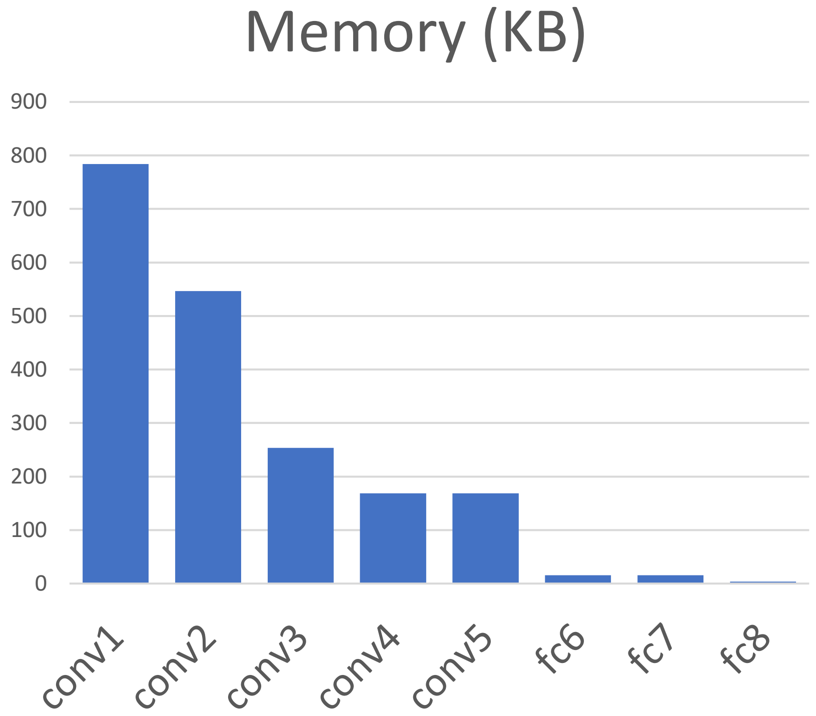
AlexNet

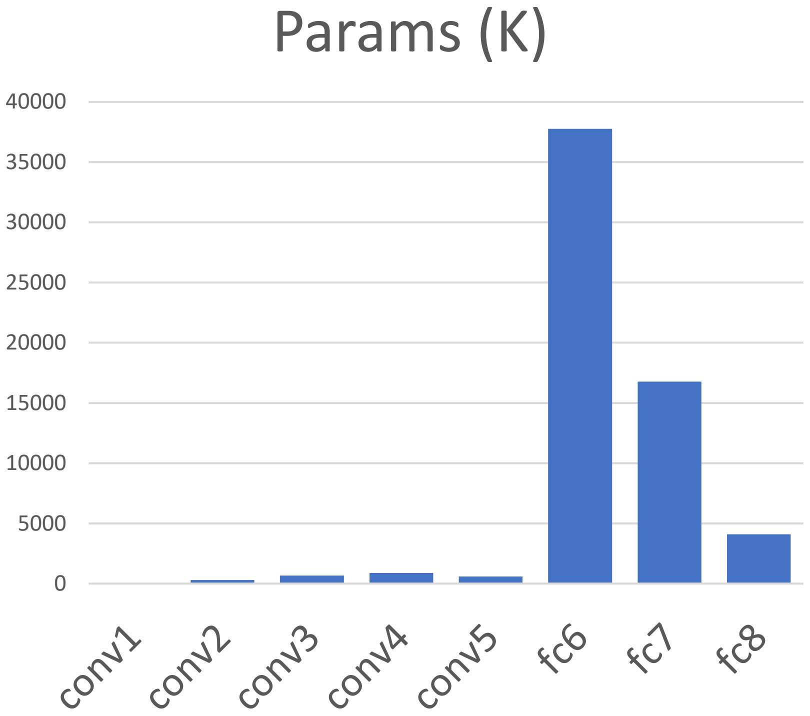


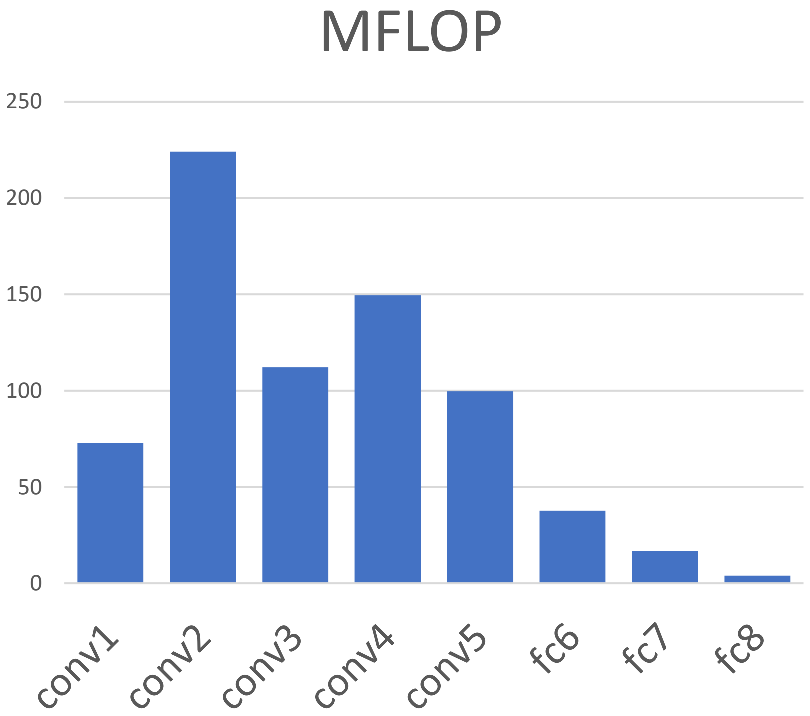
Most of the **memory usage** is in the early convolution layers

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

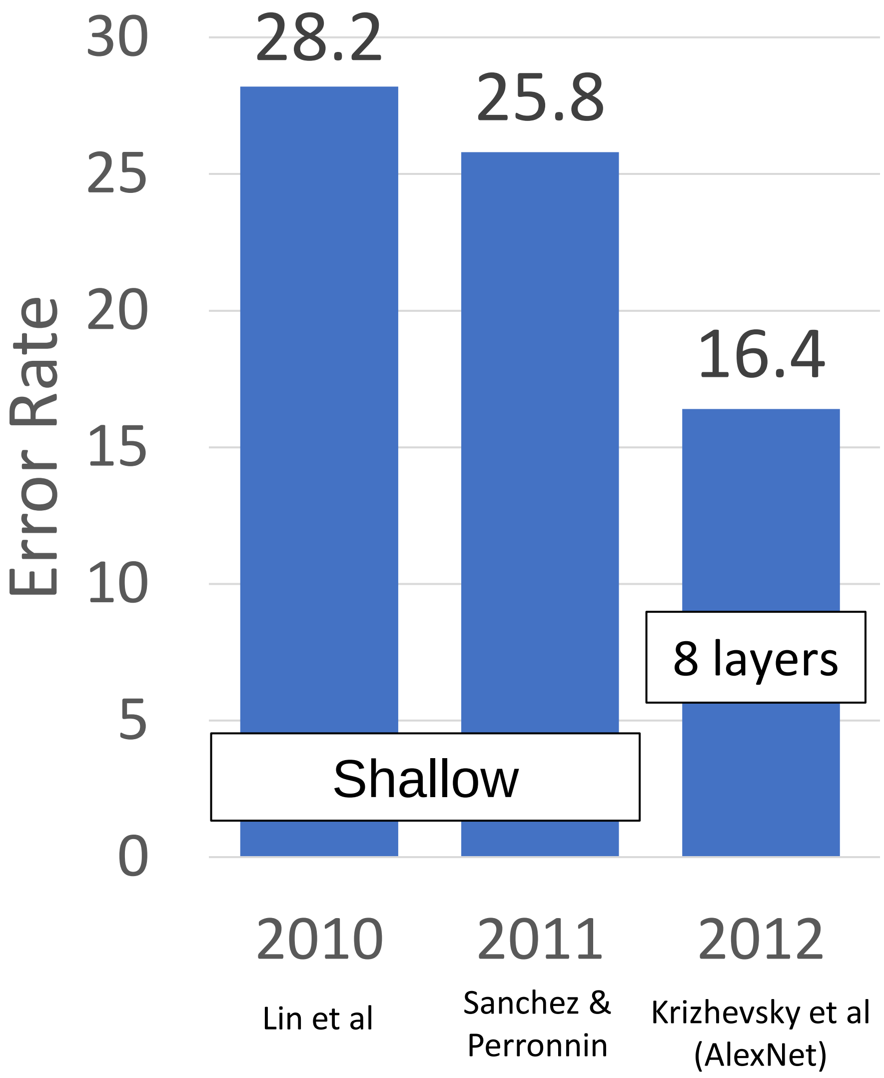
Most **floating-point ops** occur in the convolution layers



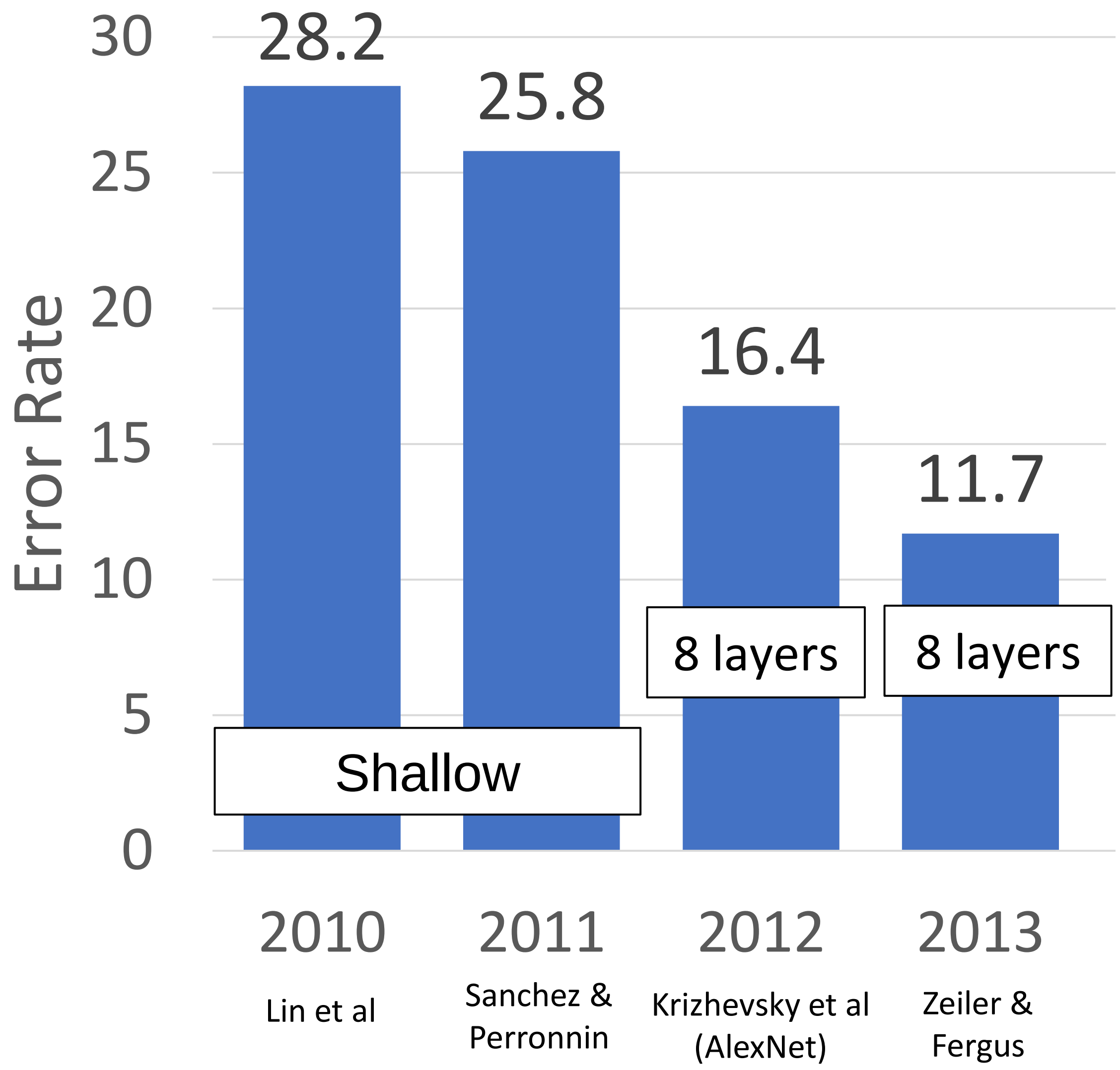




ImageNet Classification Challenge

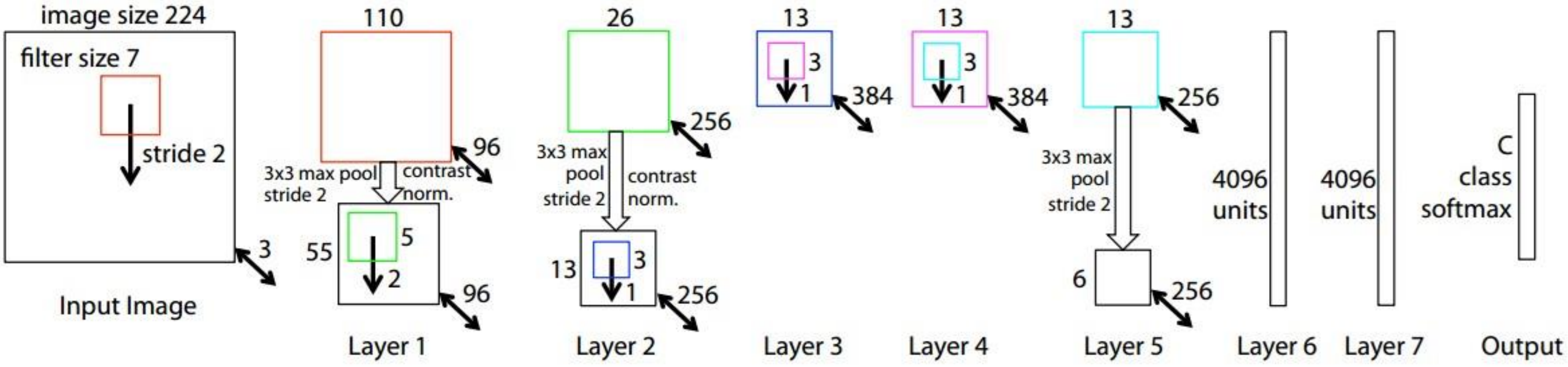


ImageNet Classification Challenge



ZFNet: A Bigger AlexNet

ImageNet top 5 error: 16.4% -> 11.7%



AlexNet but:

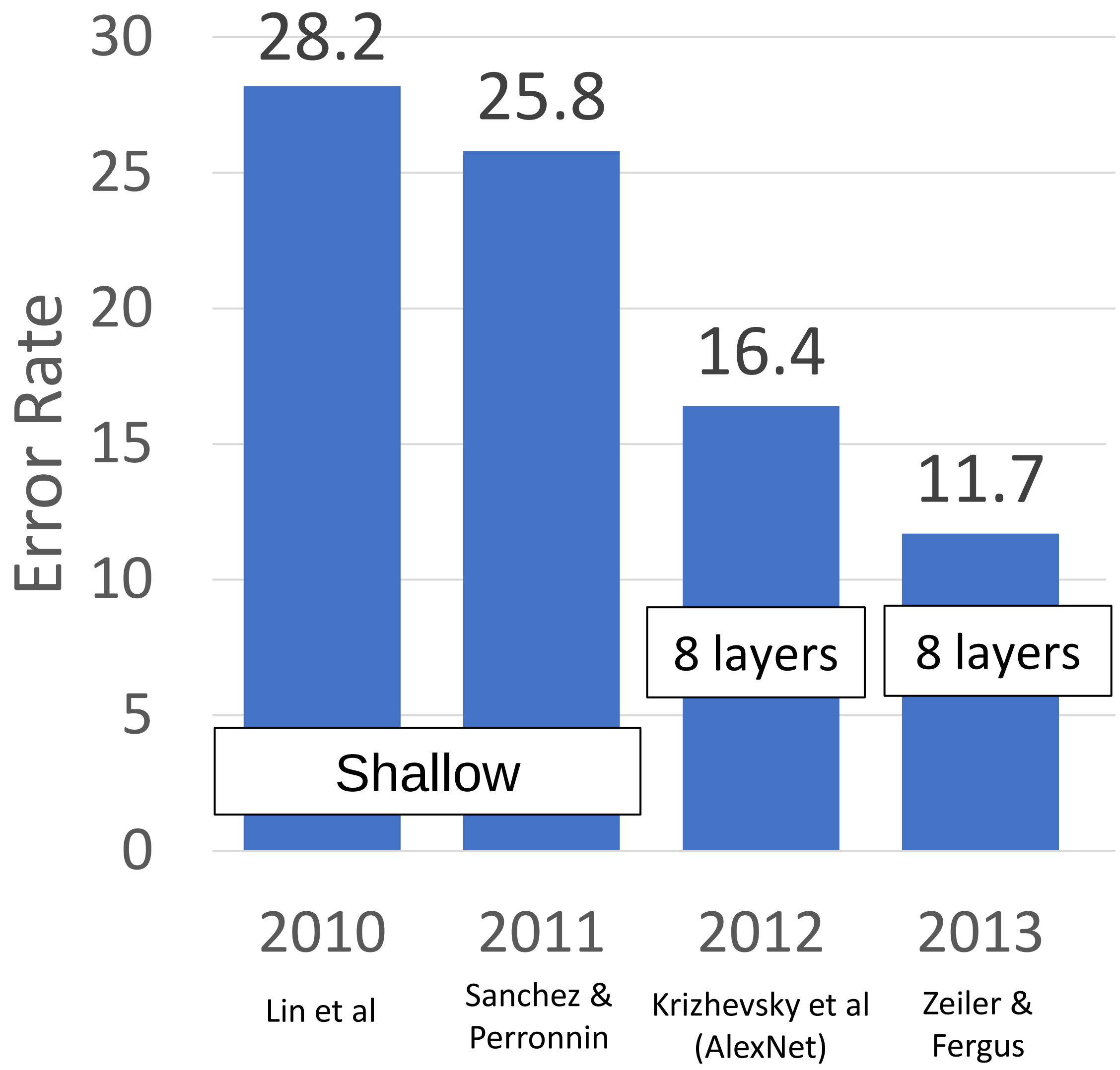
CONV1: change from (11x11 stride 4) to (7x7 stride 2)

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

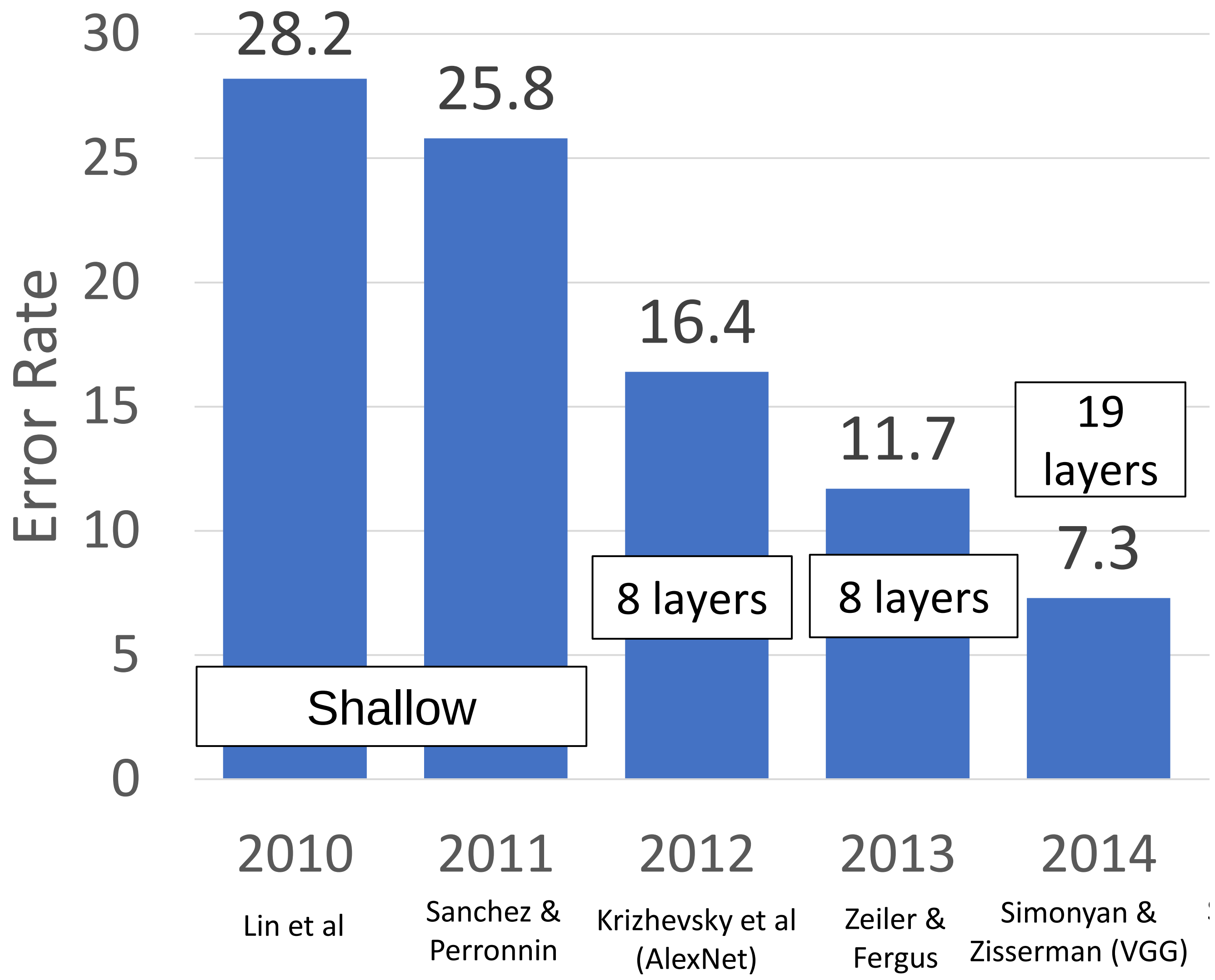
More trial and error =(

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

Network has 5 convolutional **stages**:

Stage 1: conv-conv-pool

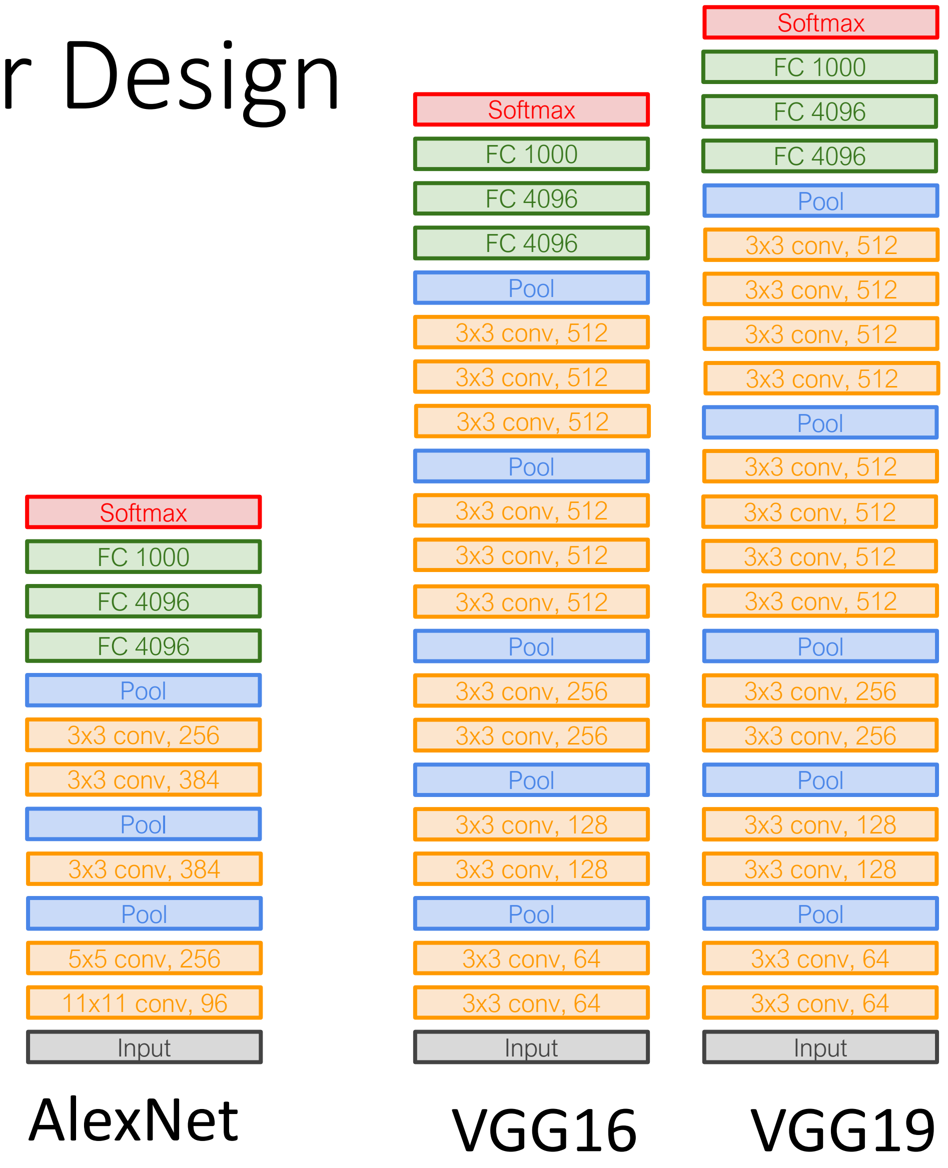
Stage 2: conv-conv-pool

Stage 3: conv-conv-conv-[conv]-pool

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

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

(VGG-19 has 4 conv in stage 3--5)



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

VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

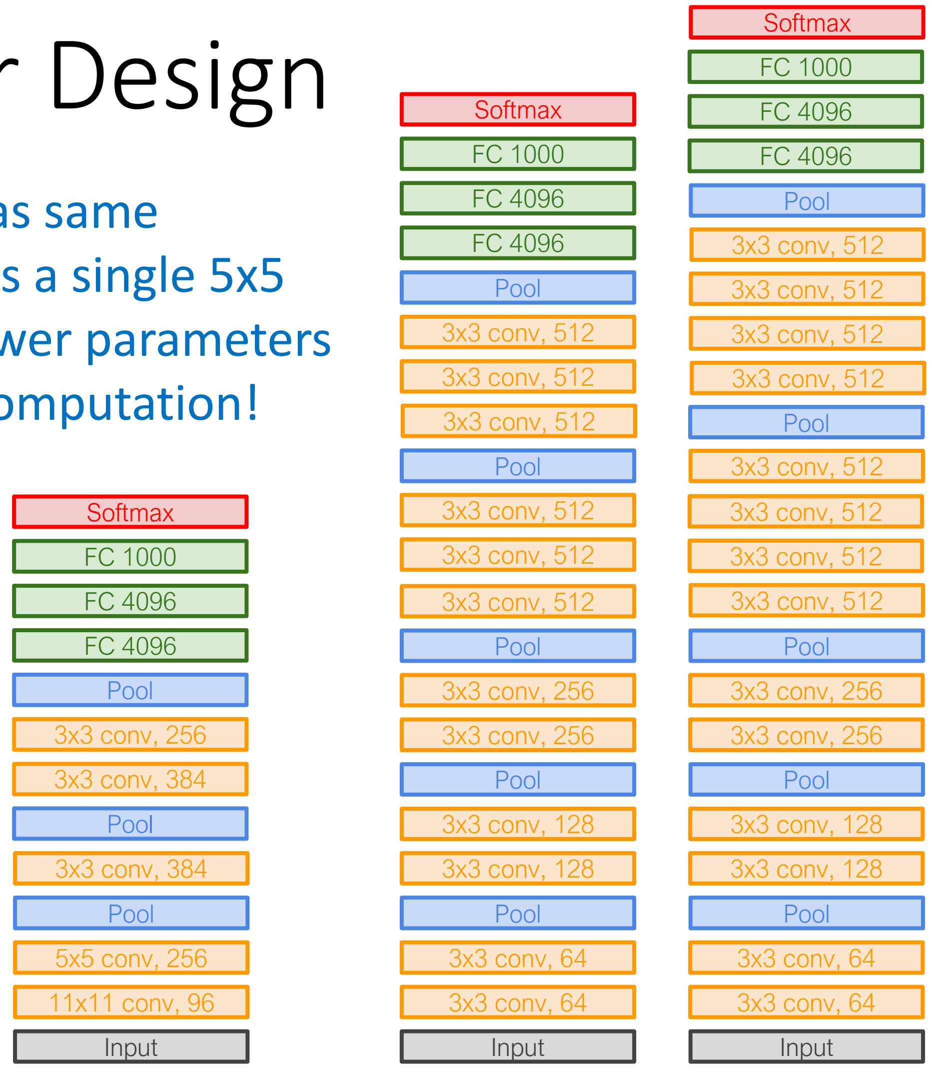
All conv are 3x3 stride 1 pad 1

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!

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



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>

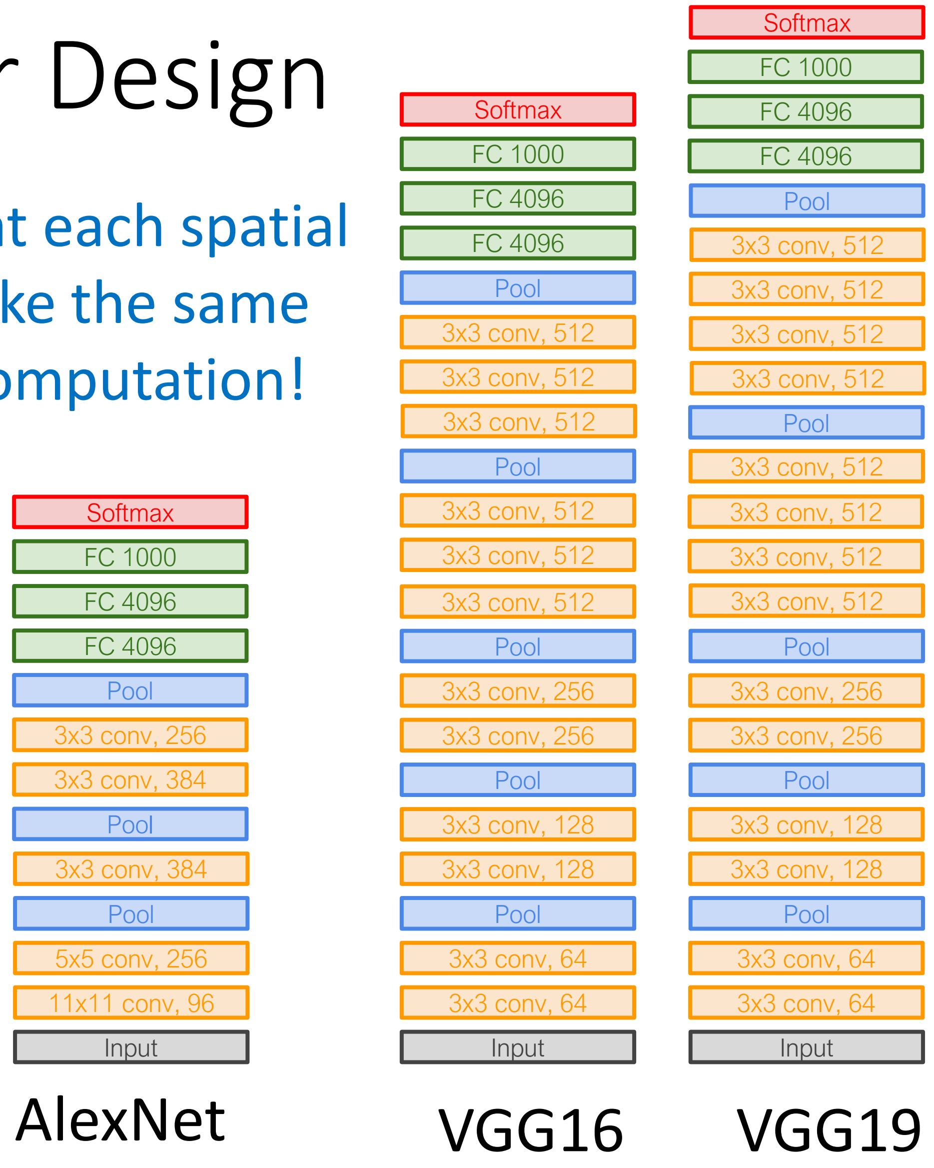
All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

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: $36HWC^2$	FLOPs: $36HWC^2$

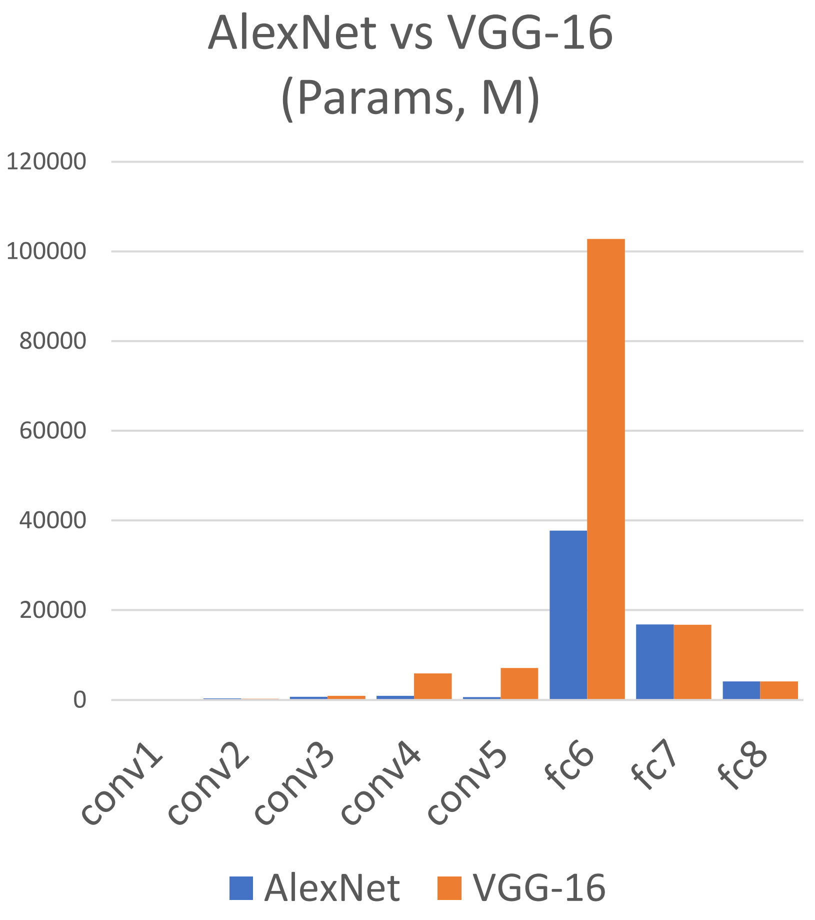


Simonyan and Zissermann, "Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR 2015

AlexNet vs VGG-16: Much bigger network!

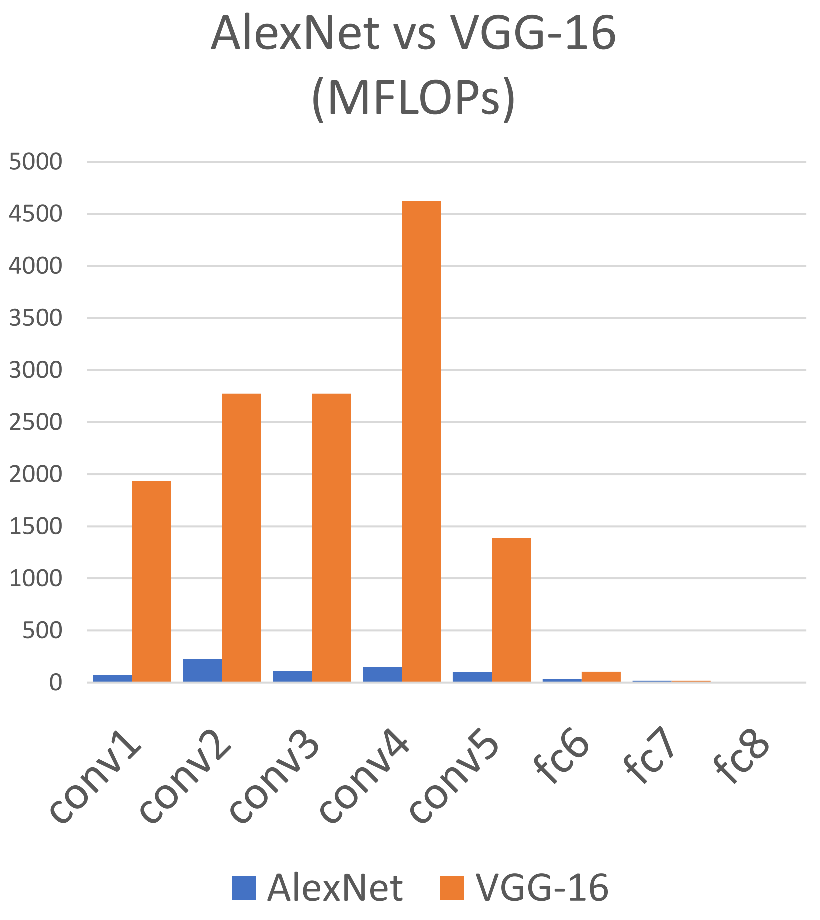
AlexNet total: 1.9 MB

VGG-16 total: 48.6 MB (25x)



AlexNet total: 61M

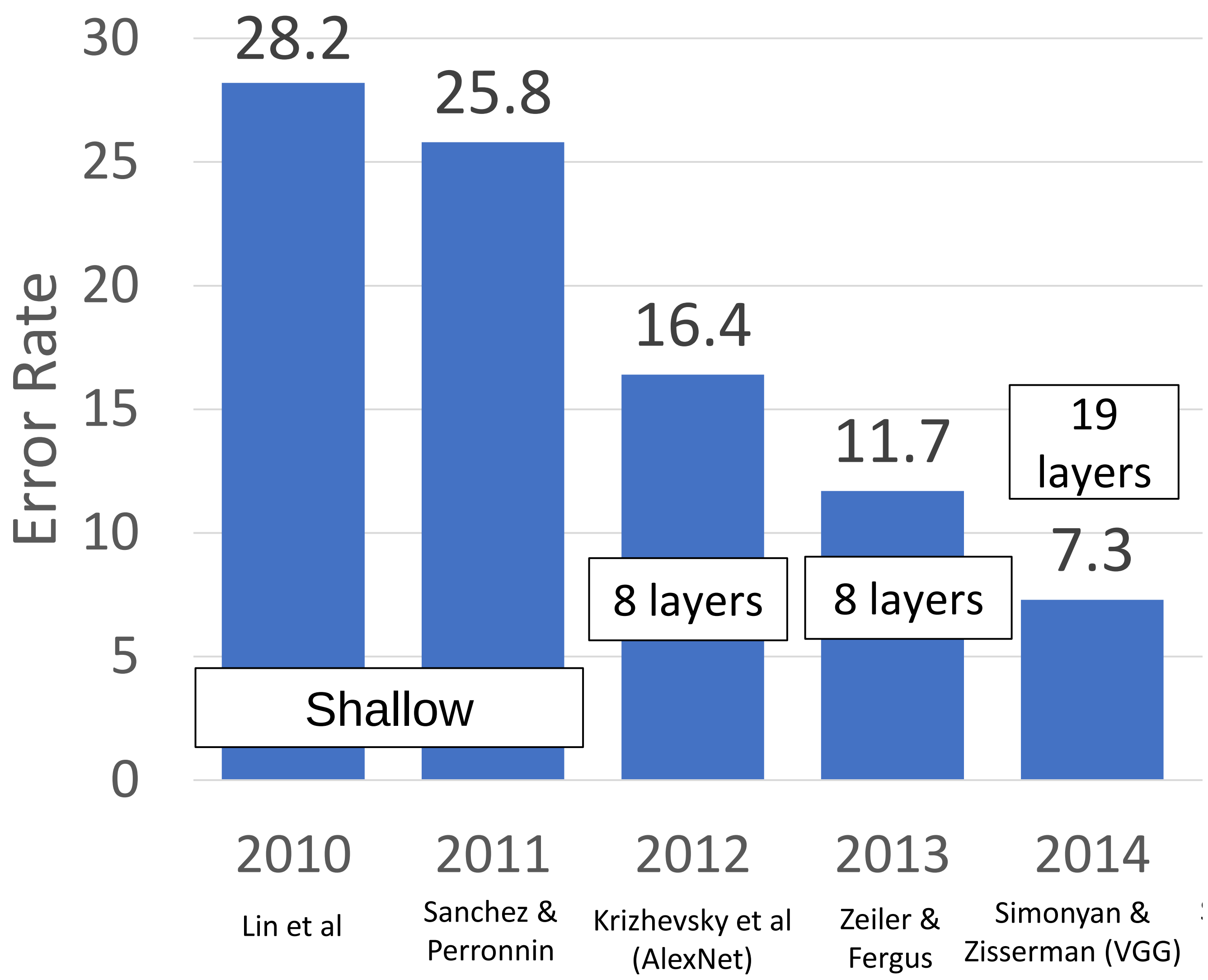
VGG-16 total: 138M (2.3x)



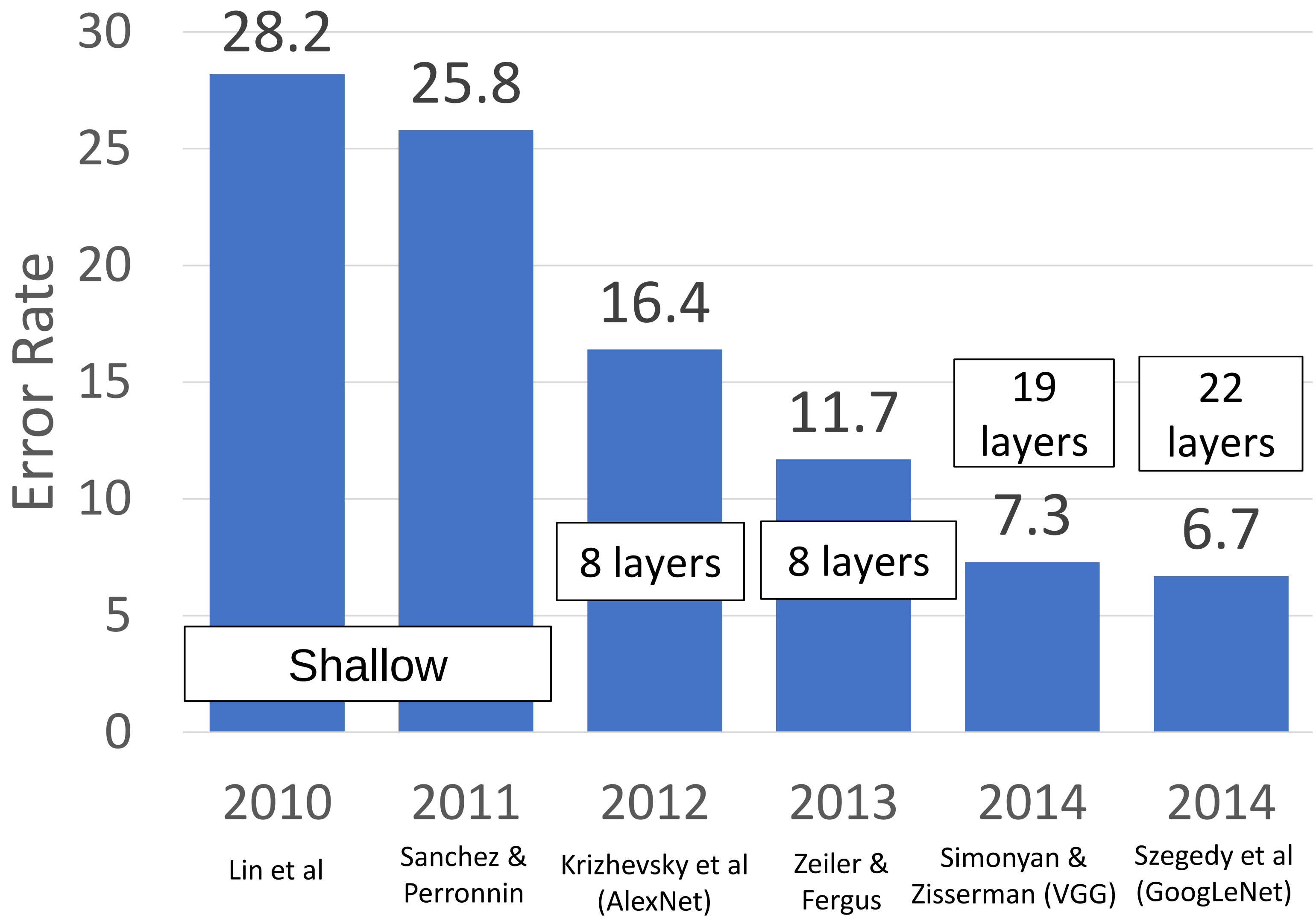
AlexNet total: 0.7 GFLOP

VGG-16 total: 13.6 GFLOP (19.4x)

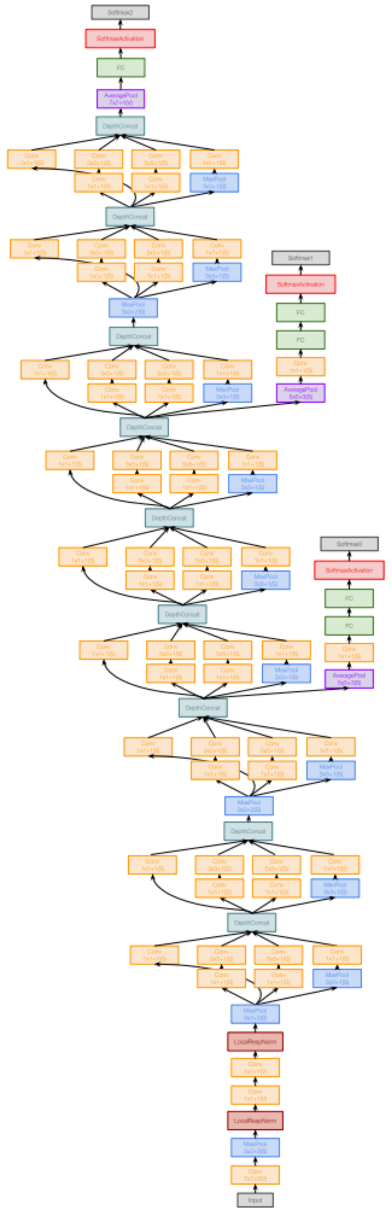
ImageNet Classification Challenge



ImageNet Classification Challenge



GoogLeNet: Focus on Efficiency



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

- Aggressive Stem
- Inception Module
- Global Average Pooling
- Auxiliary Classifiers

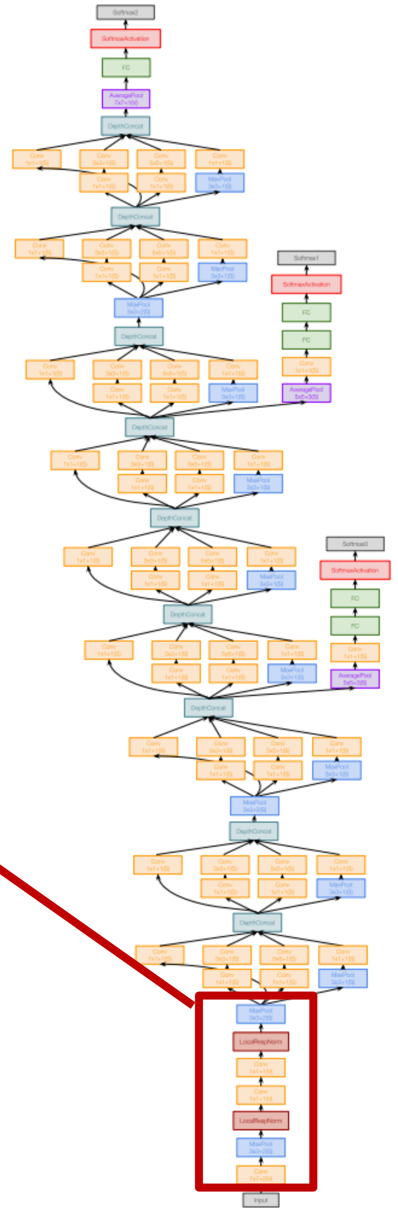
(Commonly used in modern CNN architectures)

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	KH/KW	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

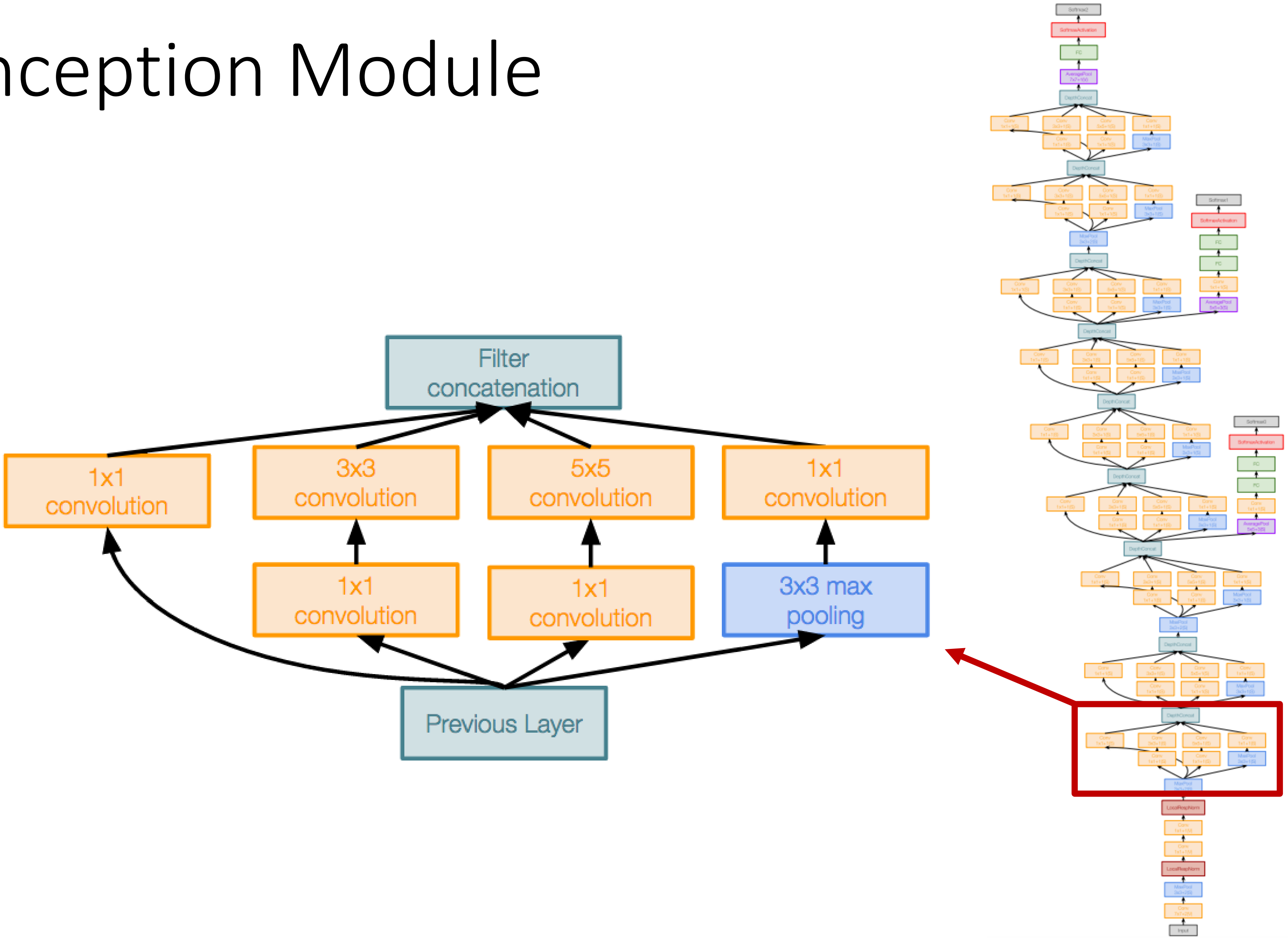
Compare VGG-16:
Memory: 42.9 MB (5.7x)
Params: 1.1M (8.9x)
MFLOP: 7485 (17.8x)

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



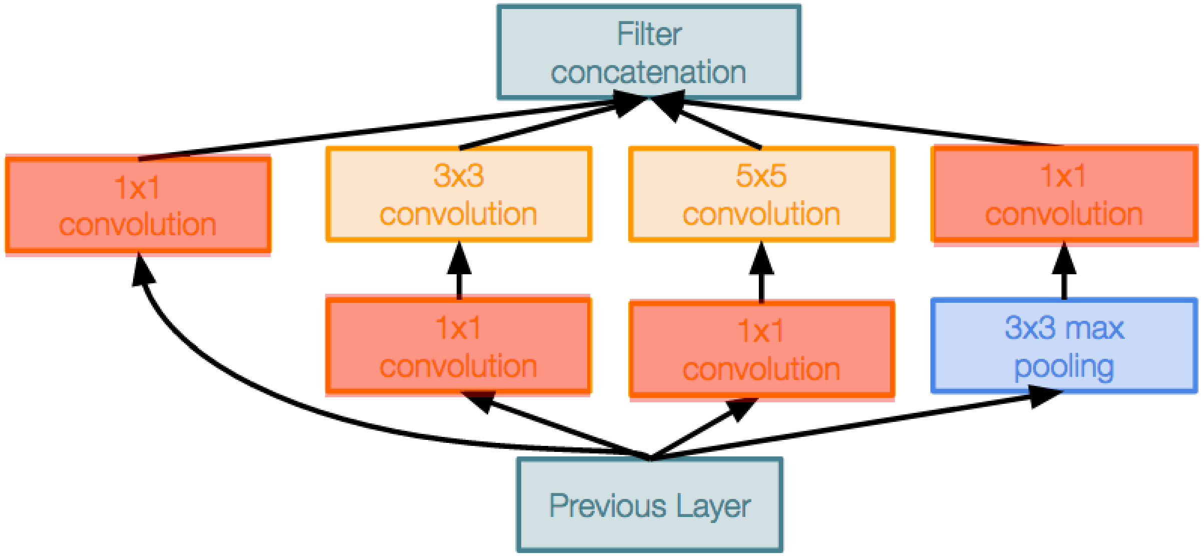
Szegedy et al, "Going deeper with convolutions", CVPR 2015

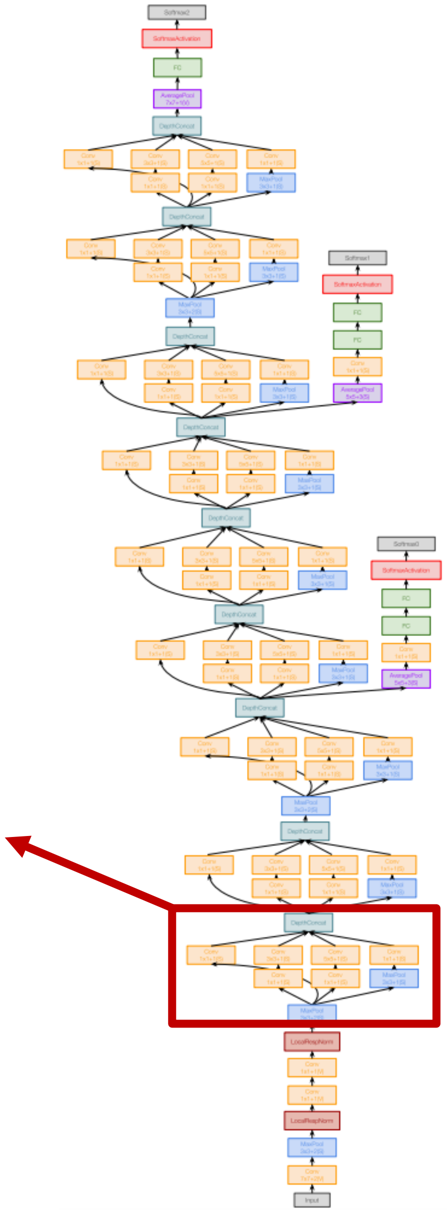
GoogLeNet: Inception Module

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!)



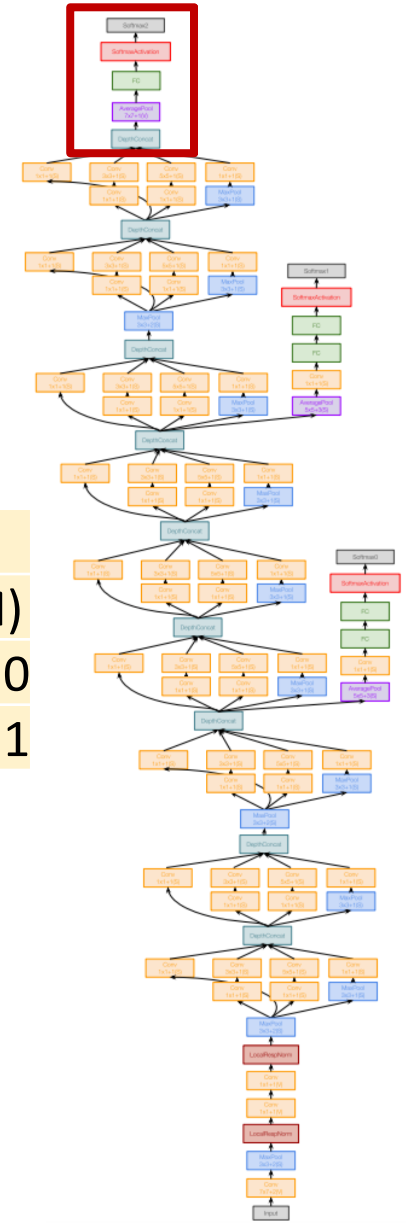


Szegedy et al, "Going deeper with convolutions", CVPR 2015

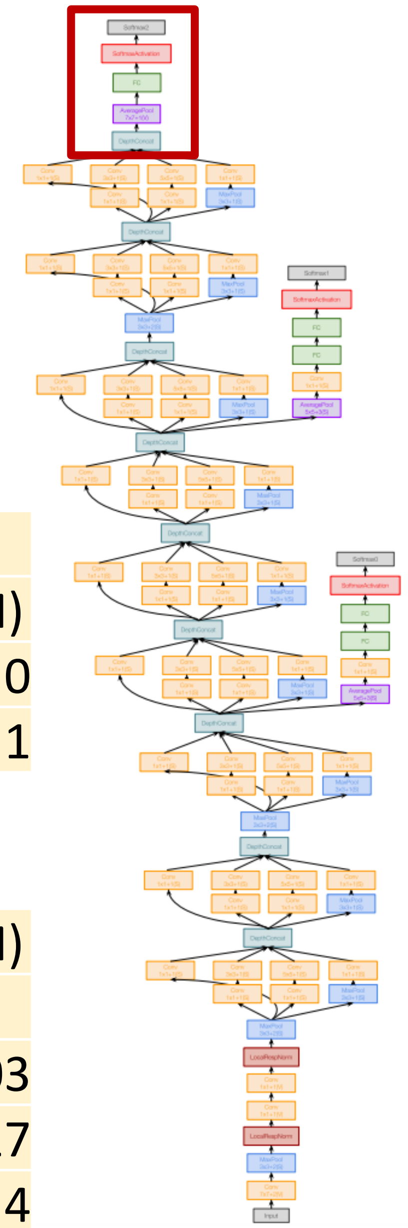
GoogLeNet: Global Average Pooling

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	KH/KW	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	KH/KW	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:

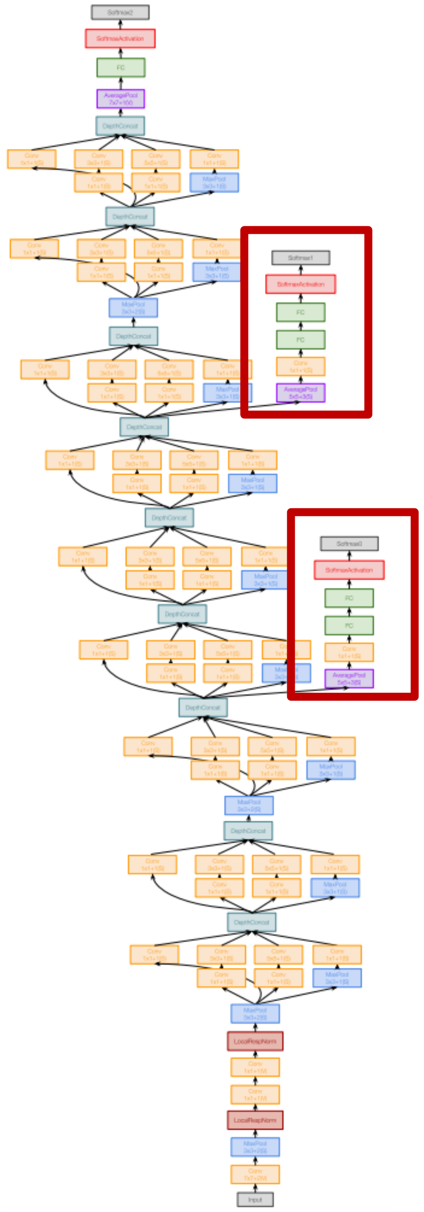
Layer	C	H/W	filters	KH/KW	stride	pad	C	H/W	memory (KB)	params (K)	flop (M)
flatten	512	7					25088		98		
fc6	25088			4096			4096		16	102760	103
fc7	4096			4096			4096		16	16777	17
fc8	4096			1000			1000		4	4096	4

GoogLeNet: Auxiliary Classifiers

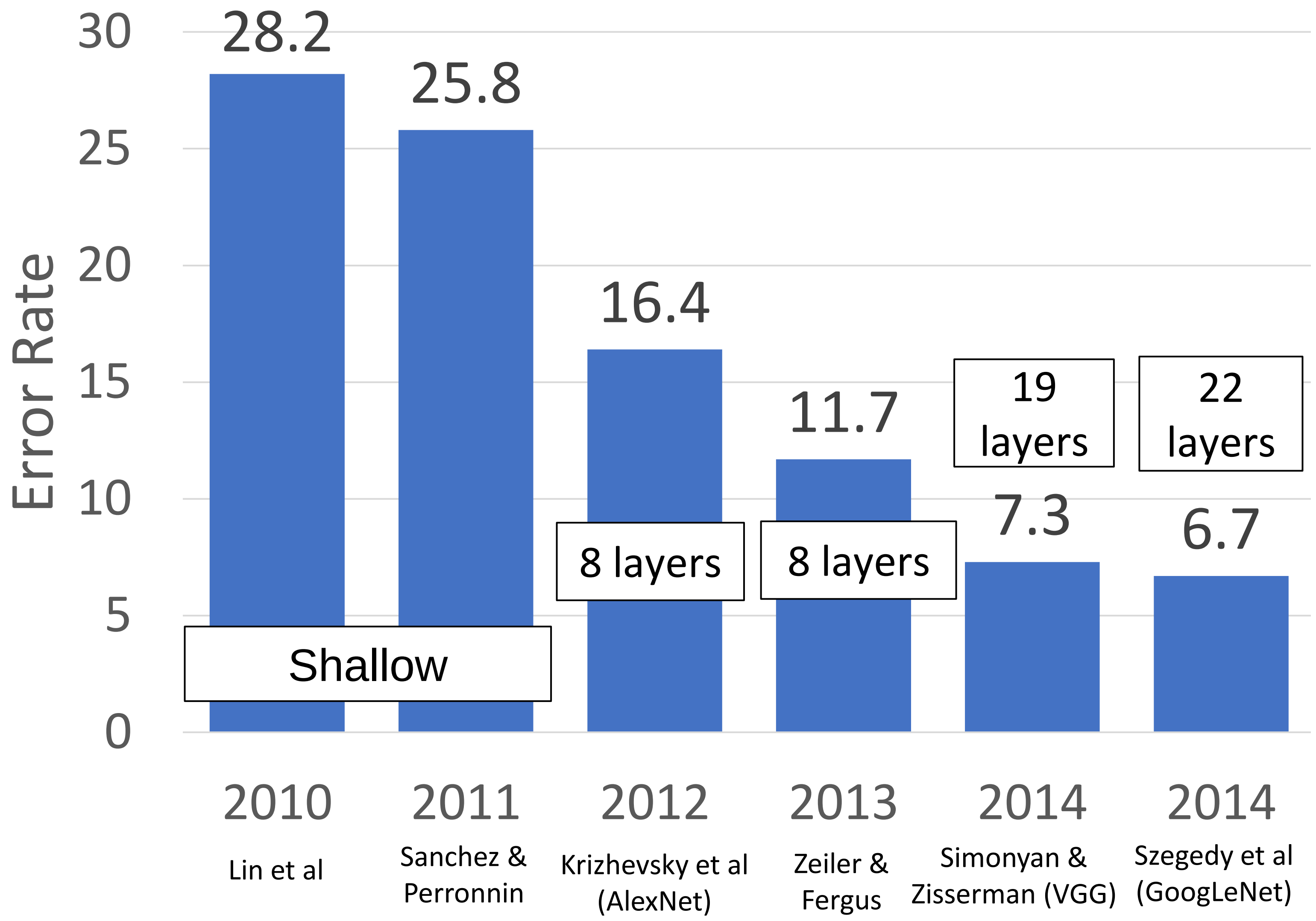
Training using loss at the end of the network didn't work well: Network is too deep, gradients don't propagate cleanly

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

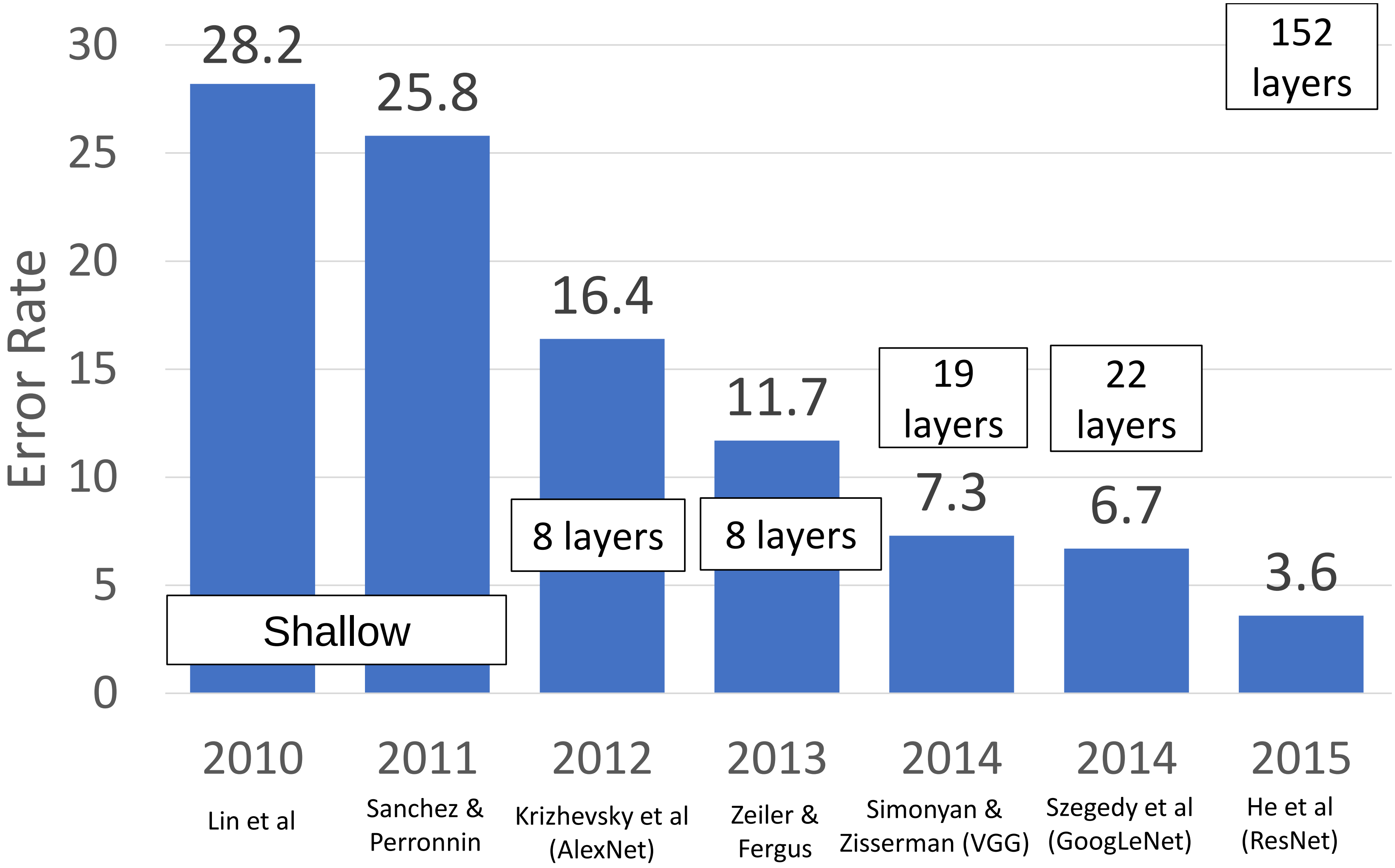
GoogLeNet was before batch normalization! With BatchNorm no longer need to use this trick



ImageNet Classification Challenge



ImageNet Classification Challenge

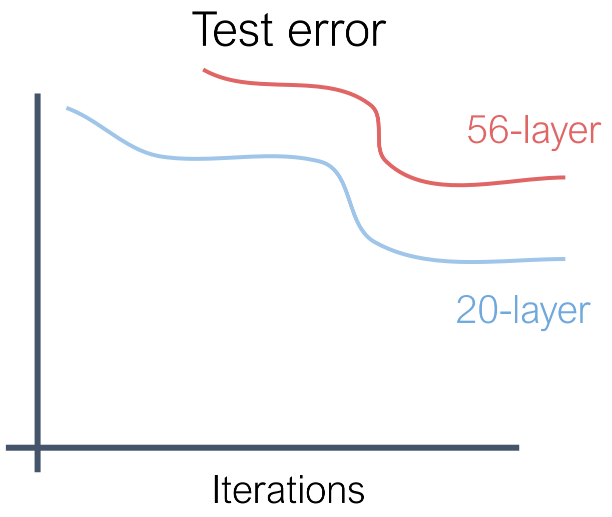


Residual Networks

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

Deeper model does worse than shallow model!

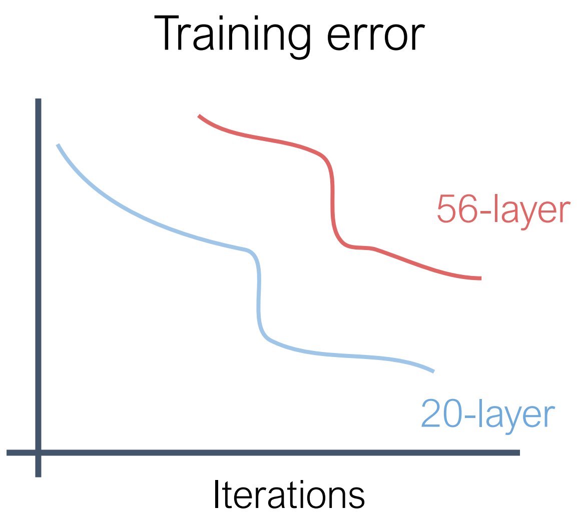
Initial guess: Deep model is **overfitting** since it is much bigger than the other model

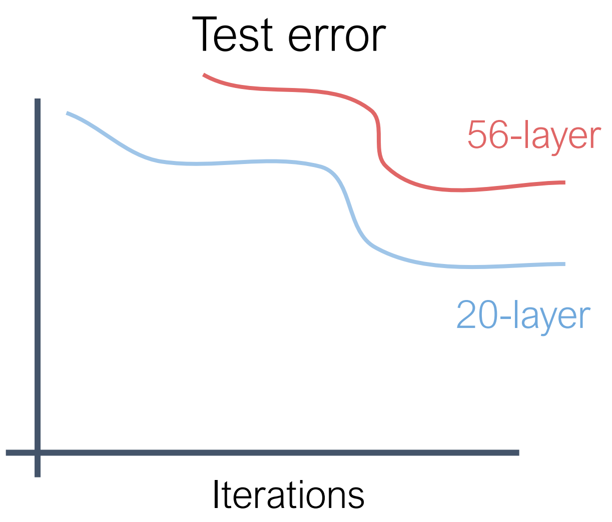


He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

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 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

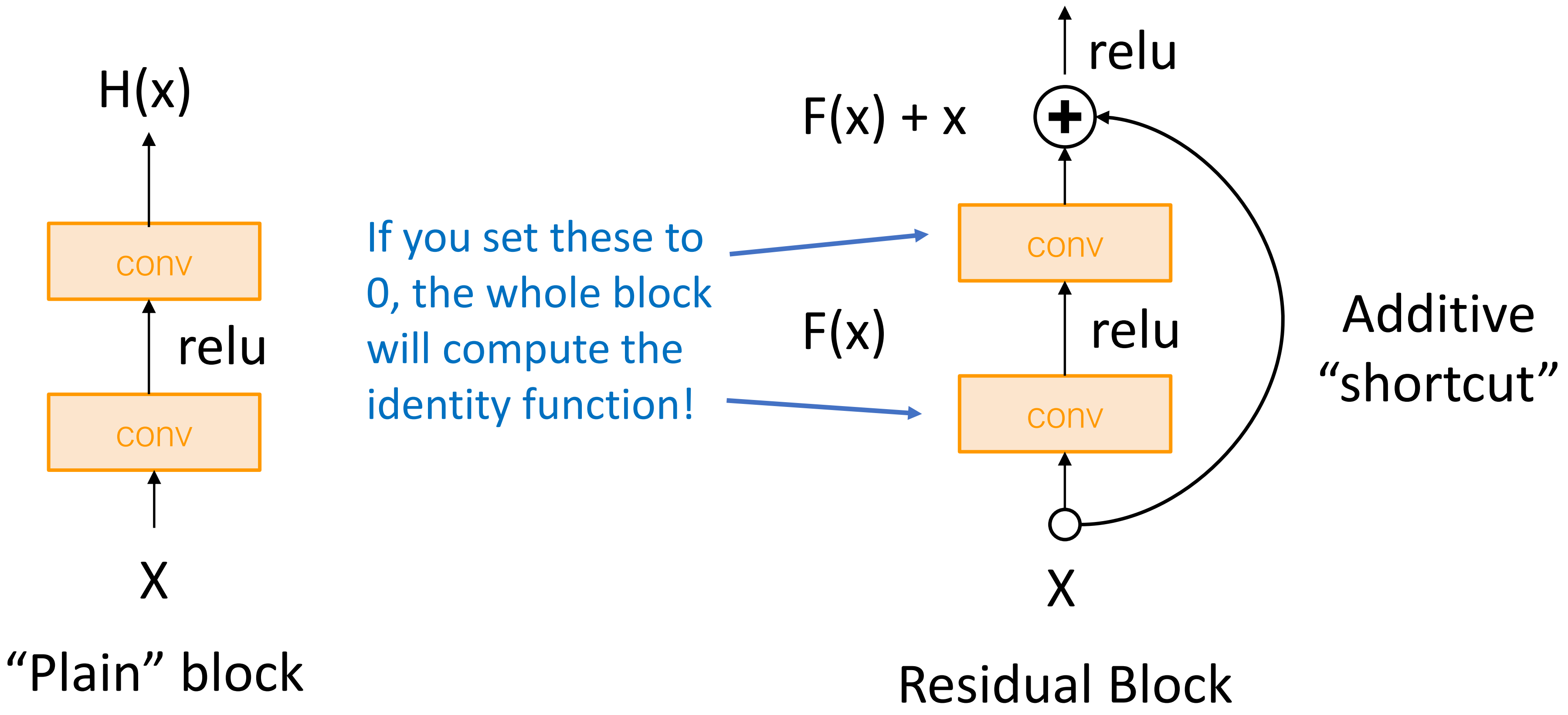
Hypothesis: This is an optimization problem. Deeper models are harder to optimize, and in particular don't learn identity functions to emulate shallow models

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!



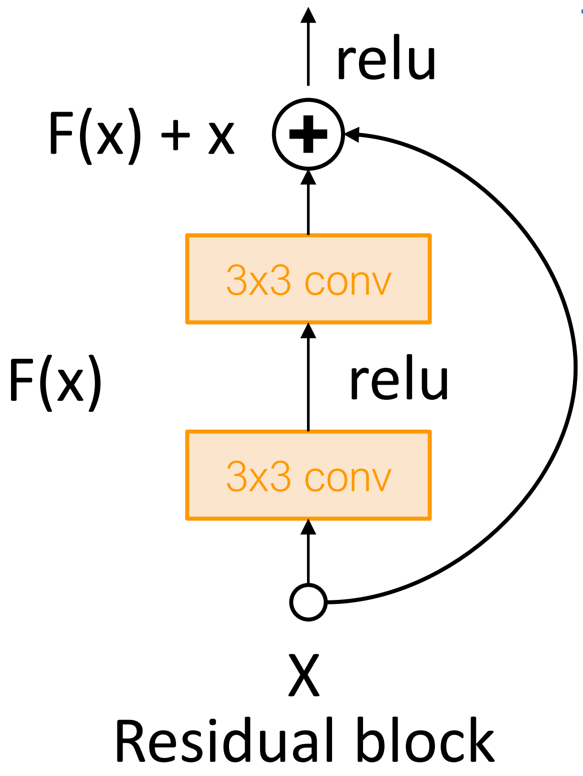
He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

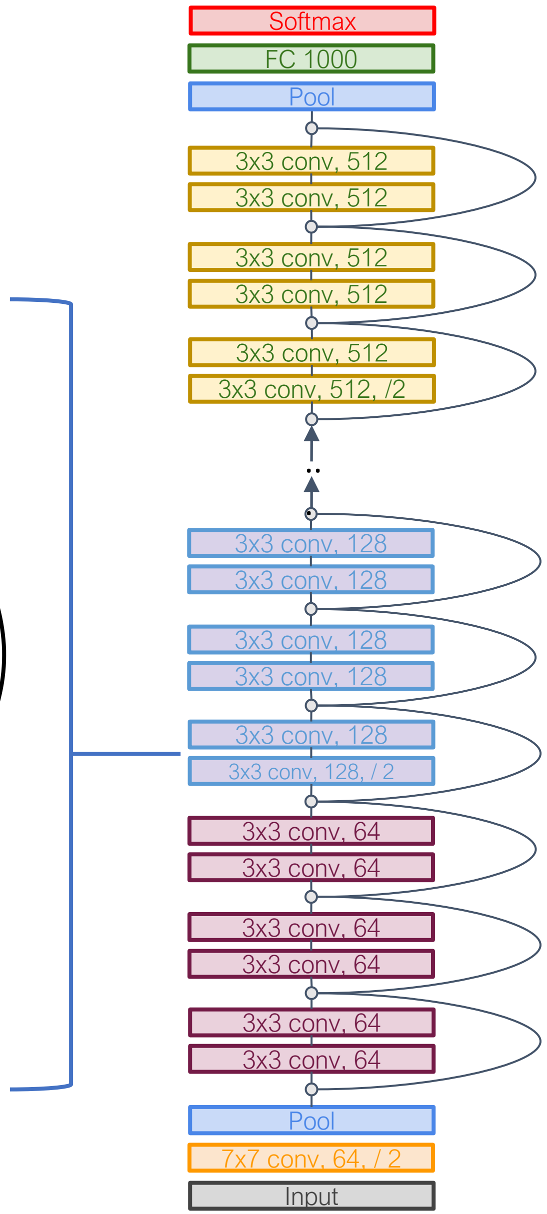
Residual Networks

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



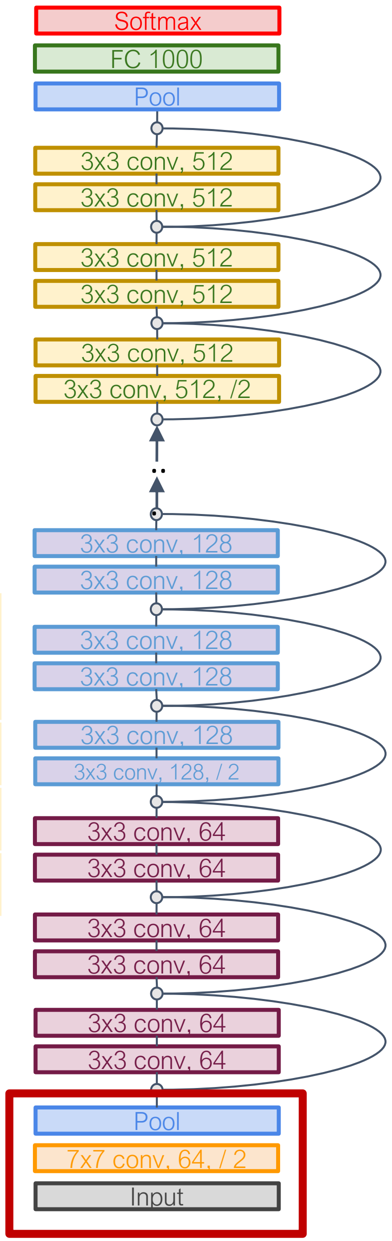


He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016

Residual Networks

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

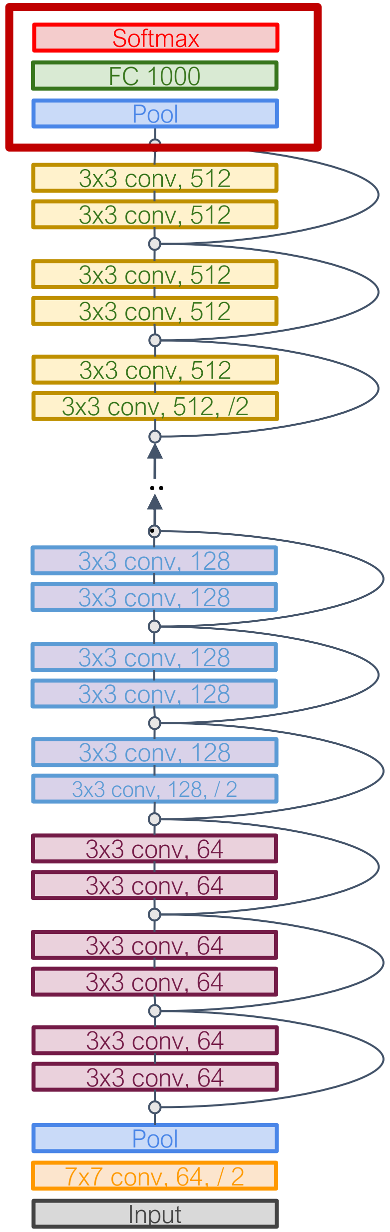
	Input size		Layer				Output size				
Layer	C	H/W	filters	KH/KW	stride	pad	C	H/W	memory (KB)	params (k)	flop (M)
conv	3	224	64	7	2	3	64	112	3136	9	118
max-pool	64	112		3	2	1	64	56	784	0	2



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

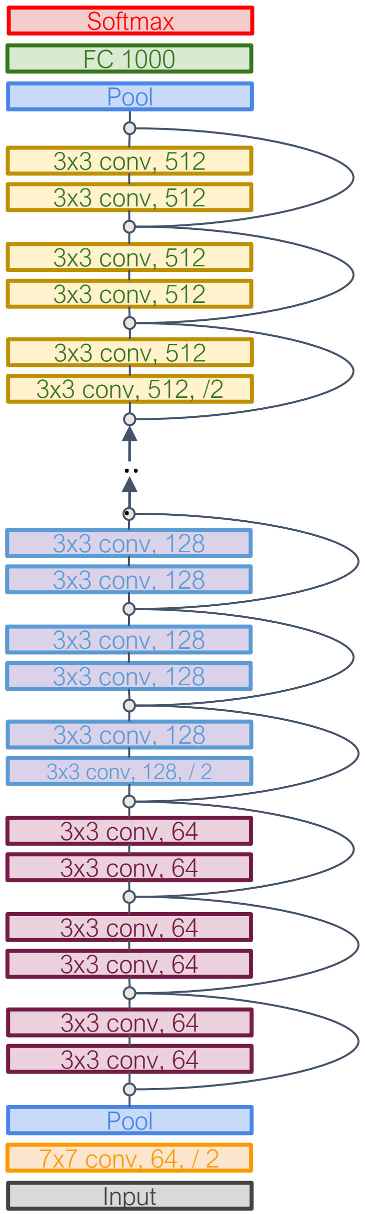
ResNet-18:
Stem: 1 conv layer
Stage 1 (C=64): 2 res. block = 4 conv
Stage 2 (C=128): 2 res. block = 4 conv
Stage 3 (C=256): 2 res. block = 4 conv
Stage 4 (C=512): 2 res. block = 4 conv
Linear

ImageNet top-5 error: 10.92
GFLOP: 1.8

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

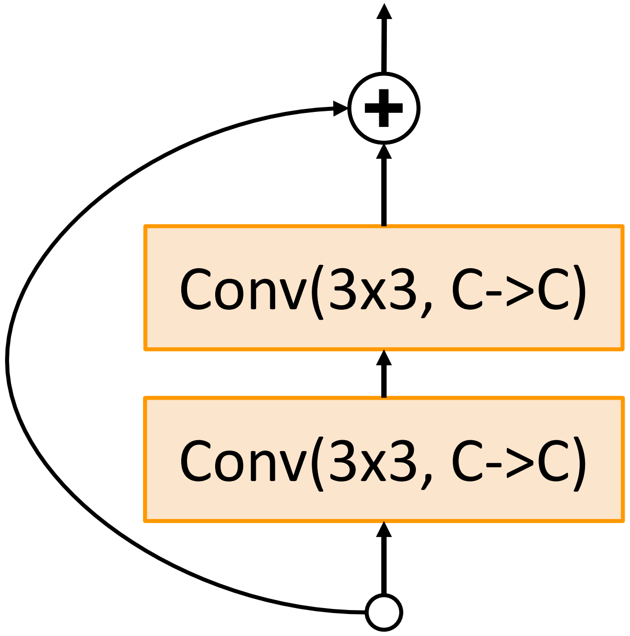
ImageNet top-5 error: 8.58
GFLOP: 3.6

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



FLOPs: $9HWC^2$

FLOPs: $9HWC^2$

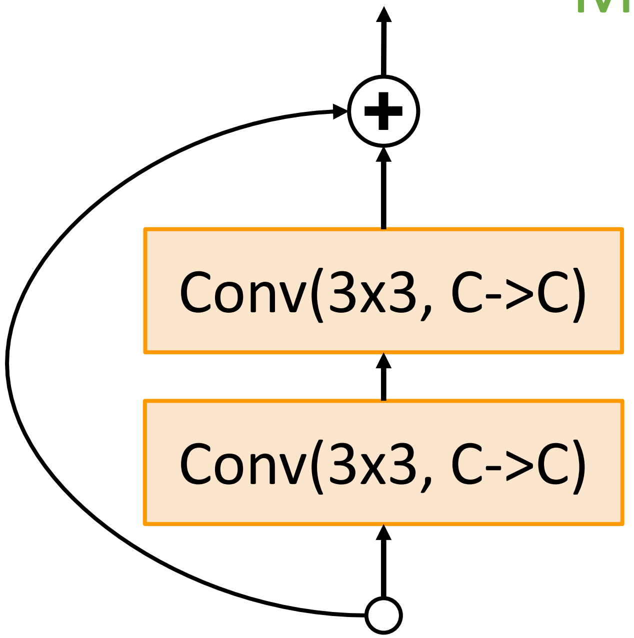
"Basic" Residual block

Total FLOPs: $18HWC^2$

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

Residual Networks: Bottleneck Block

More layers, less computational cost!



FLOPs: $9HWC^2$

FLOPs: $9HWC^2$

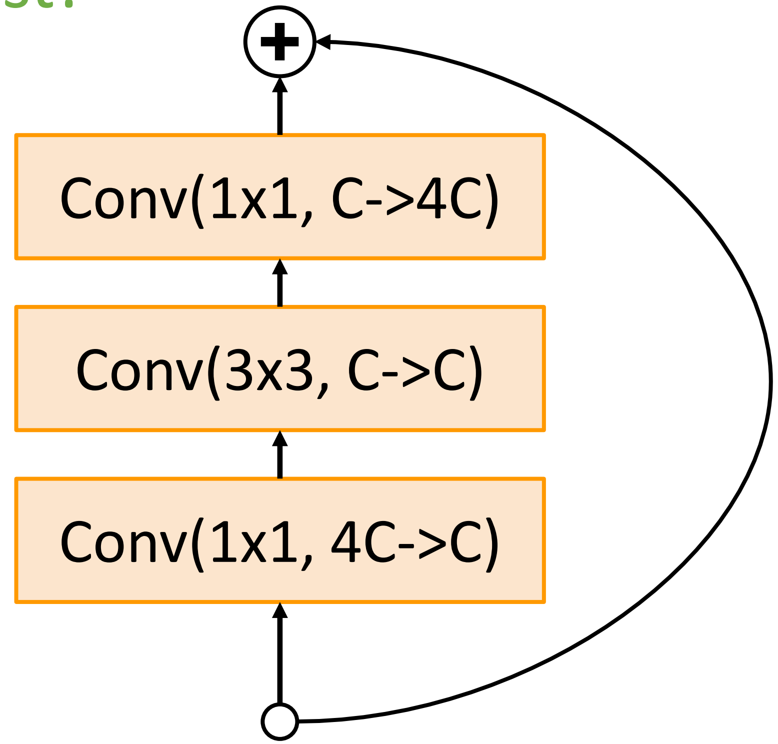
"Basic"
Residual block

Total FLOPs:
$18HWC^2$

FLOPs: $4HWC^2$

FLOPs: $9HWC^2$

FLOPs: $4HWC^2$



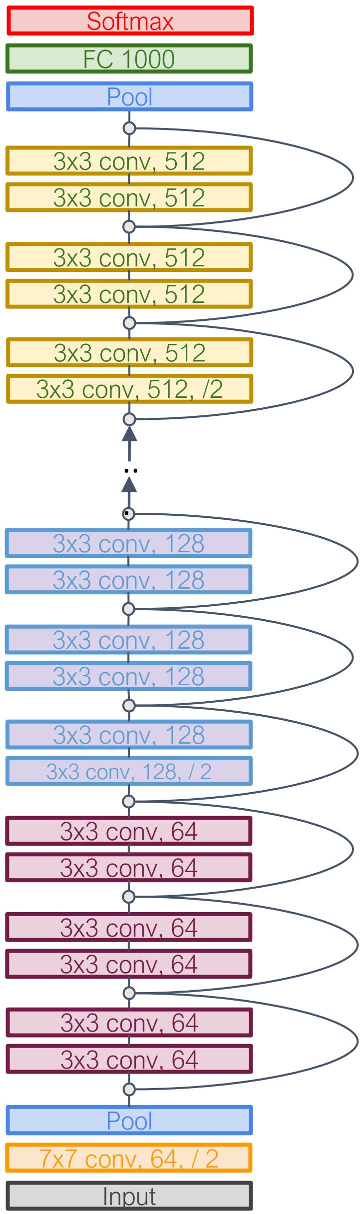
Total FLOPs:
$17HWC^2$

"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

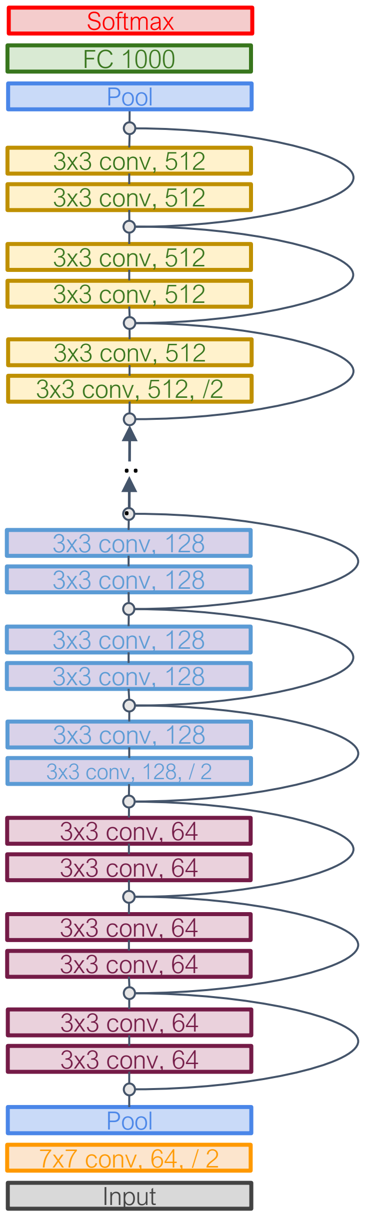


He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016
Error rates are 224x224 single-crop testing, reported by torchvision

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

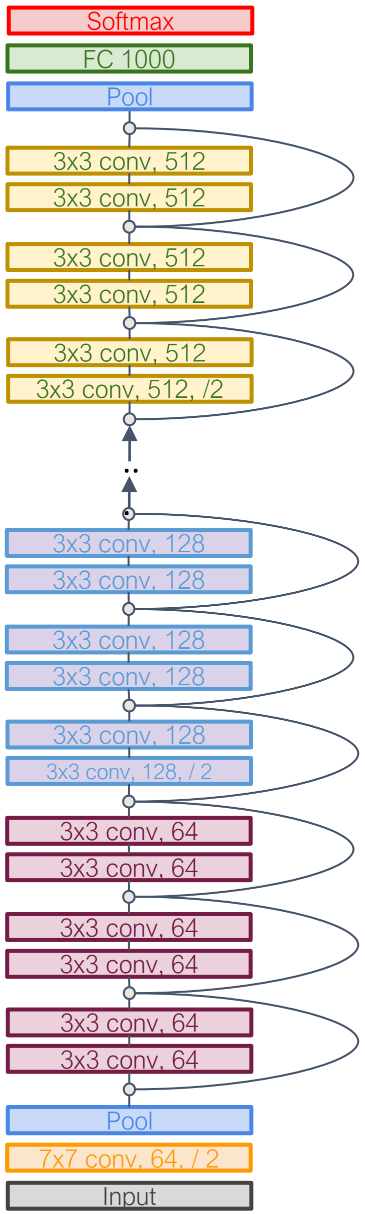


He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016
Error rates are 224x224 single-crop testing, reported by torchvision

Residual Networks

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

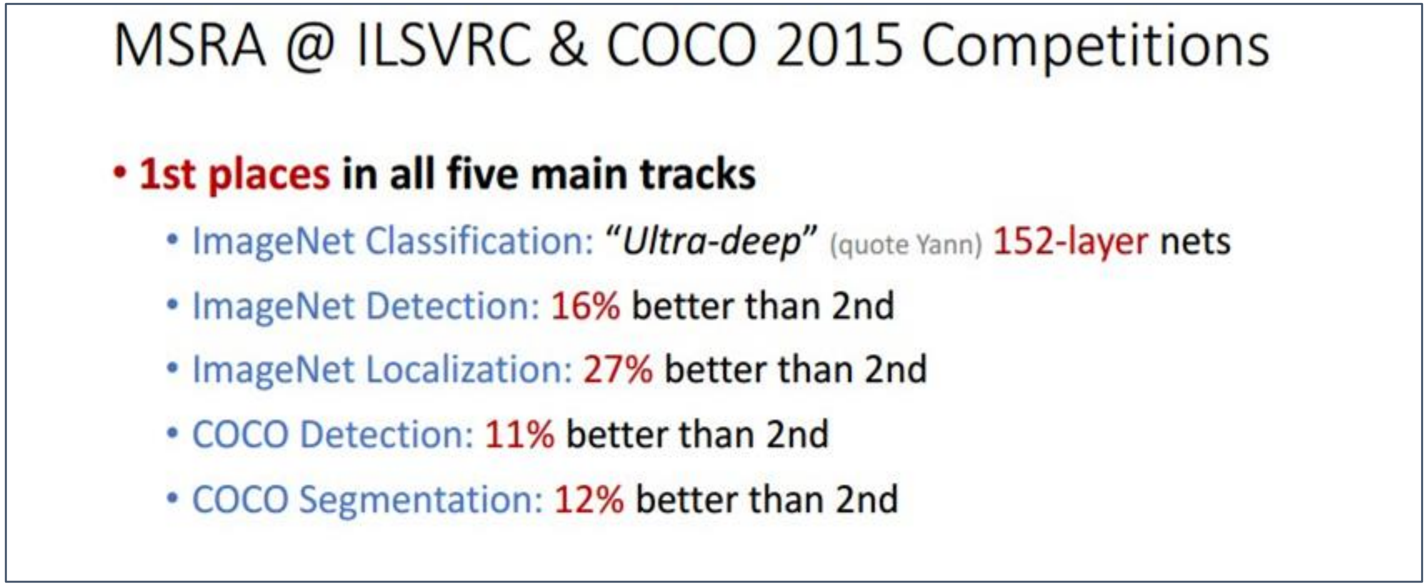
			Stage 1		Stage 2		Stage 3		Stage 4				
	Block type	Stem layers	Blocks	Layers	Blocks	Layers	Blocks	Layers	Blocks	Layers	FC layers	GFLOP	ImageNet top-5 error
ResNet-18	Basic	1	2	4	2	4	2	4	2	4	1	1.8	10.92
ResNet-34	Basic	1	3	6	4	8	6	12	3	6	1	3.6	8.58
ResNet-50	Bottle	1	3	9	4	12	6	18	3	9	1	3.8	7.13
ResNet-101	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

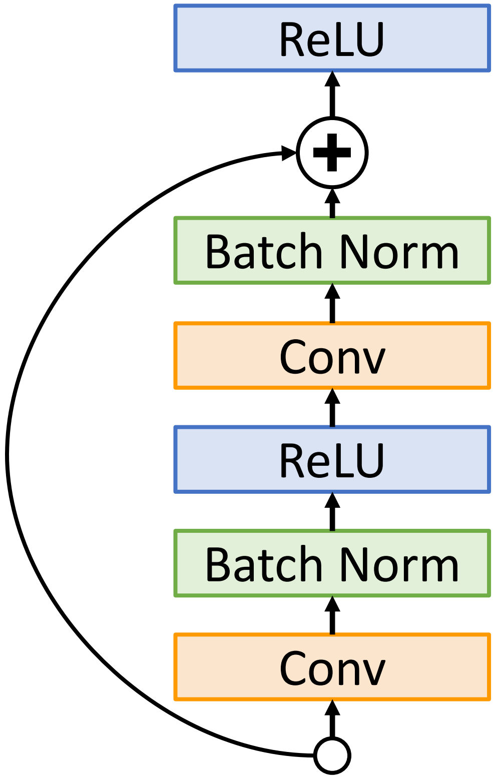
- Able to train very deep networks
- Deeper networks do better than shallow networks (as expected)
- Swept 1st place in all ILSVRC and COCO 2015 competitions
- Still widely used today!



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

Improving Residual Networks: Block Design

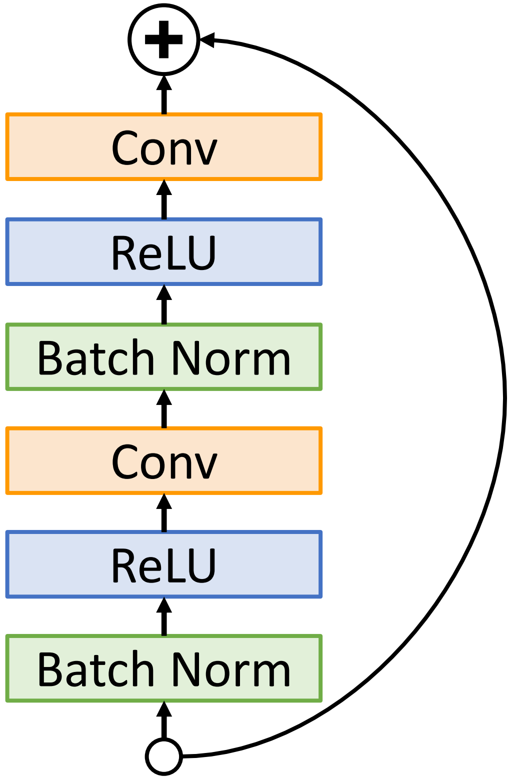
Original ResNet block



Note ReLU **after** residual:

Cannot actually learn identity function since outputs are nonnegative!

"Pre-Activation" ResNet Block



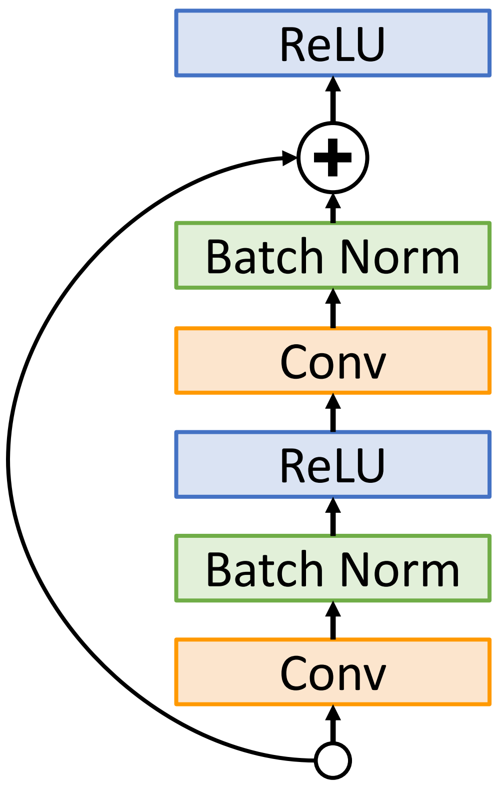
Note ReLU **inside** residual:

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

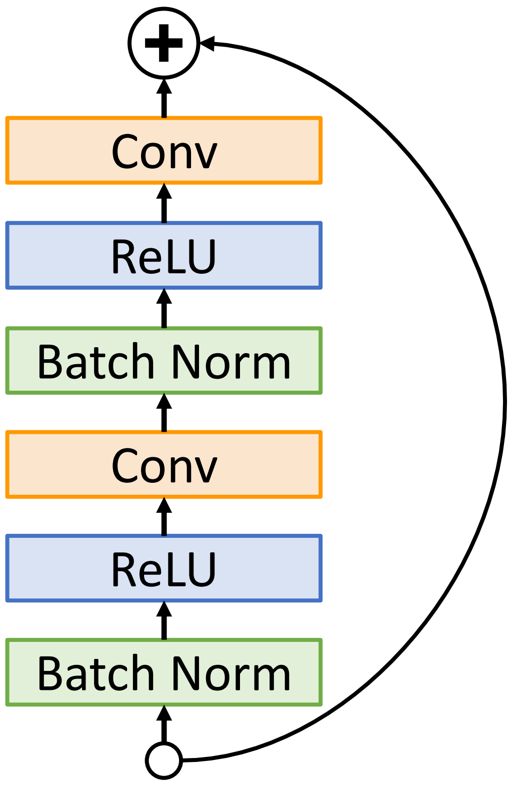


Slight improvement in accuracy (ImageNet top-1 error)

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

Not actually used that much in practice

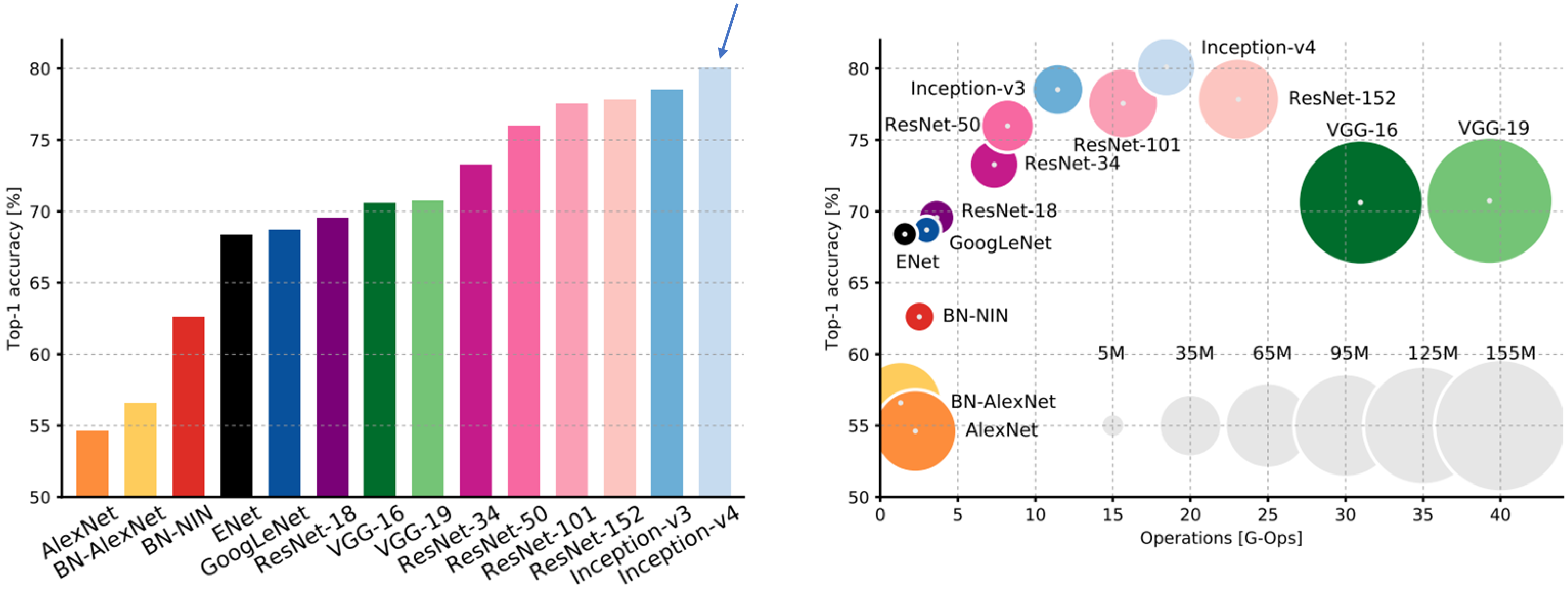
"Pre-Activation" ResNet Block



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

Comparing Complexity

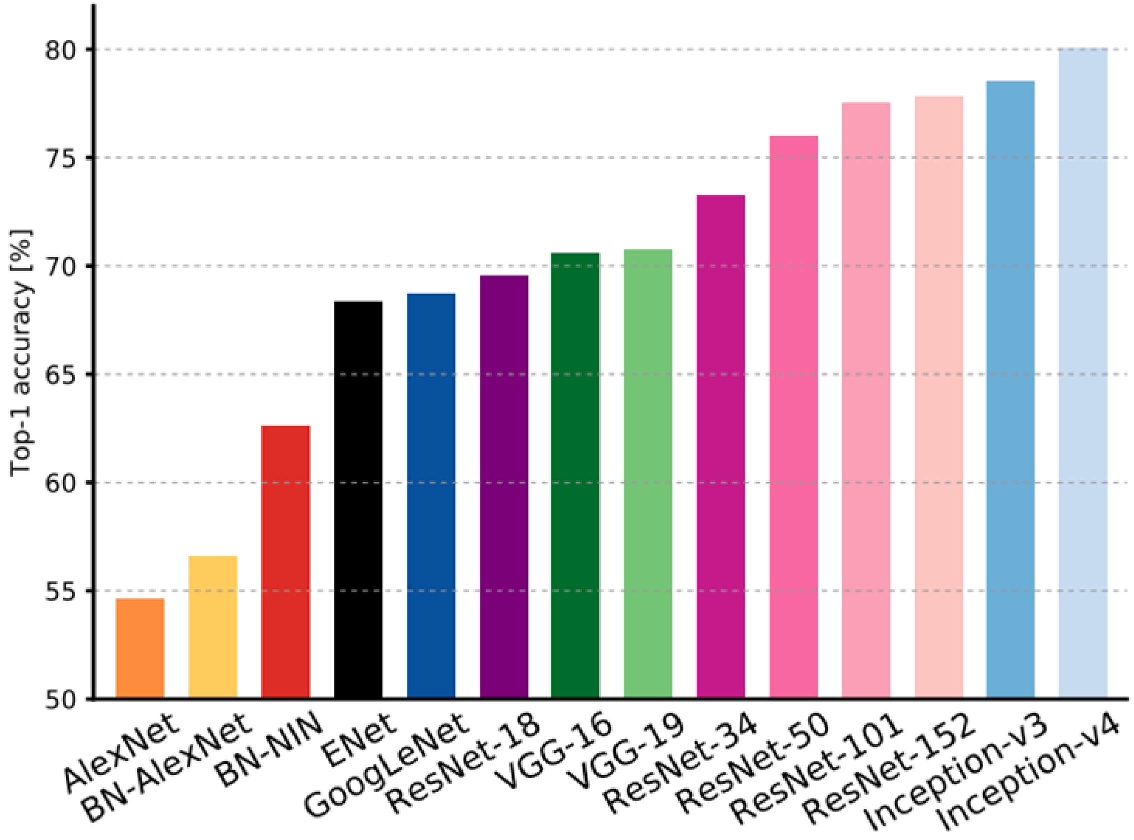
Inception-v4: Resnet + Inception!

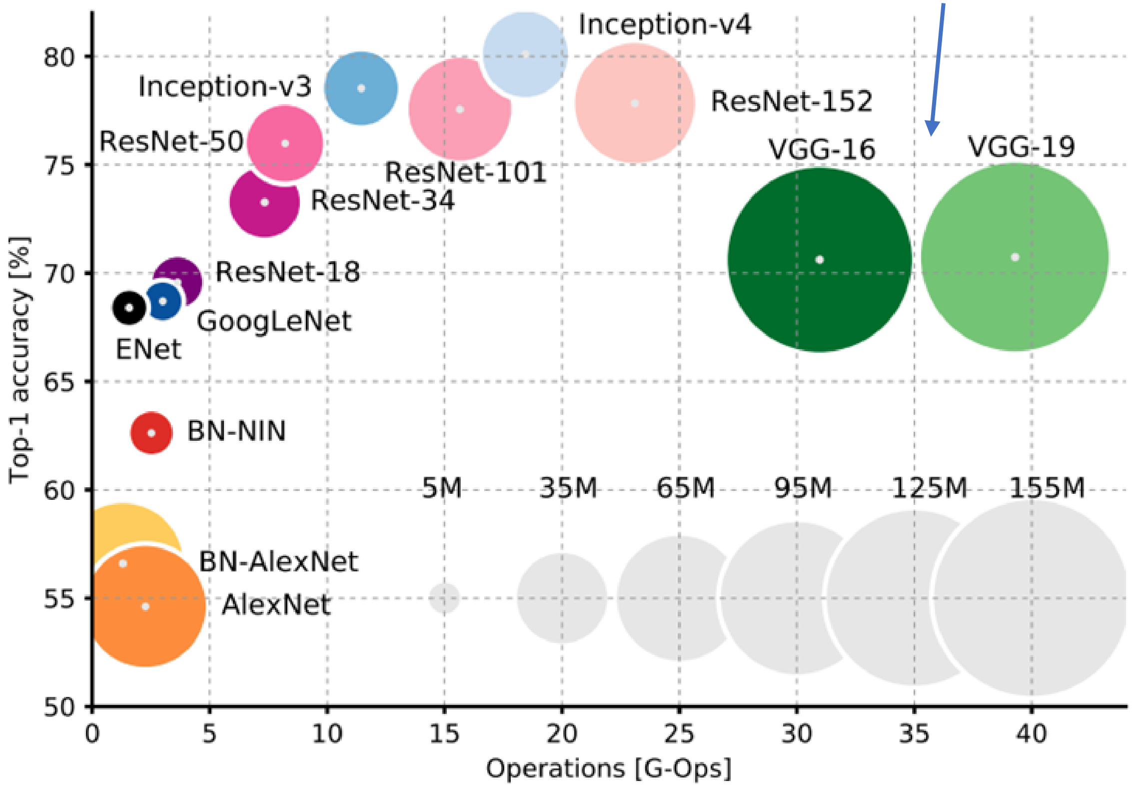


Canziani et al, “An analysis of deep neural network models for practical applications”, 2017

Comparing Complexity

VGG: Highest memory, most operations

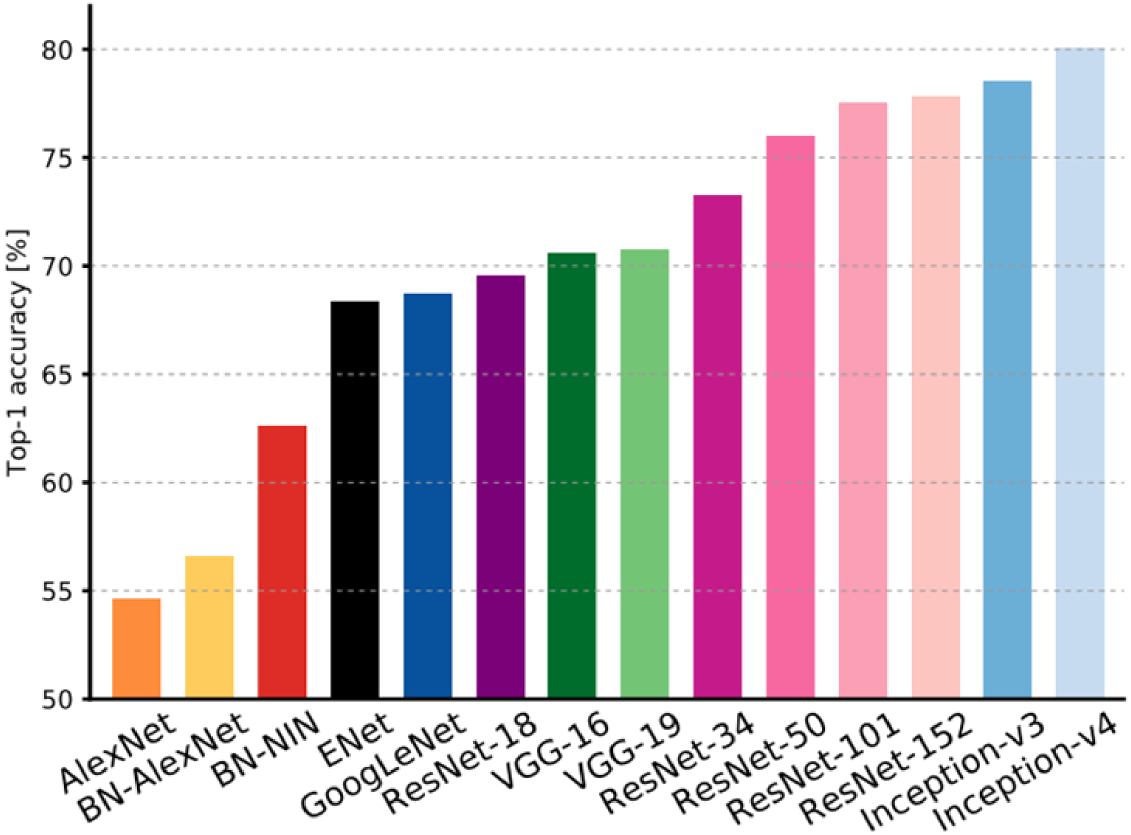


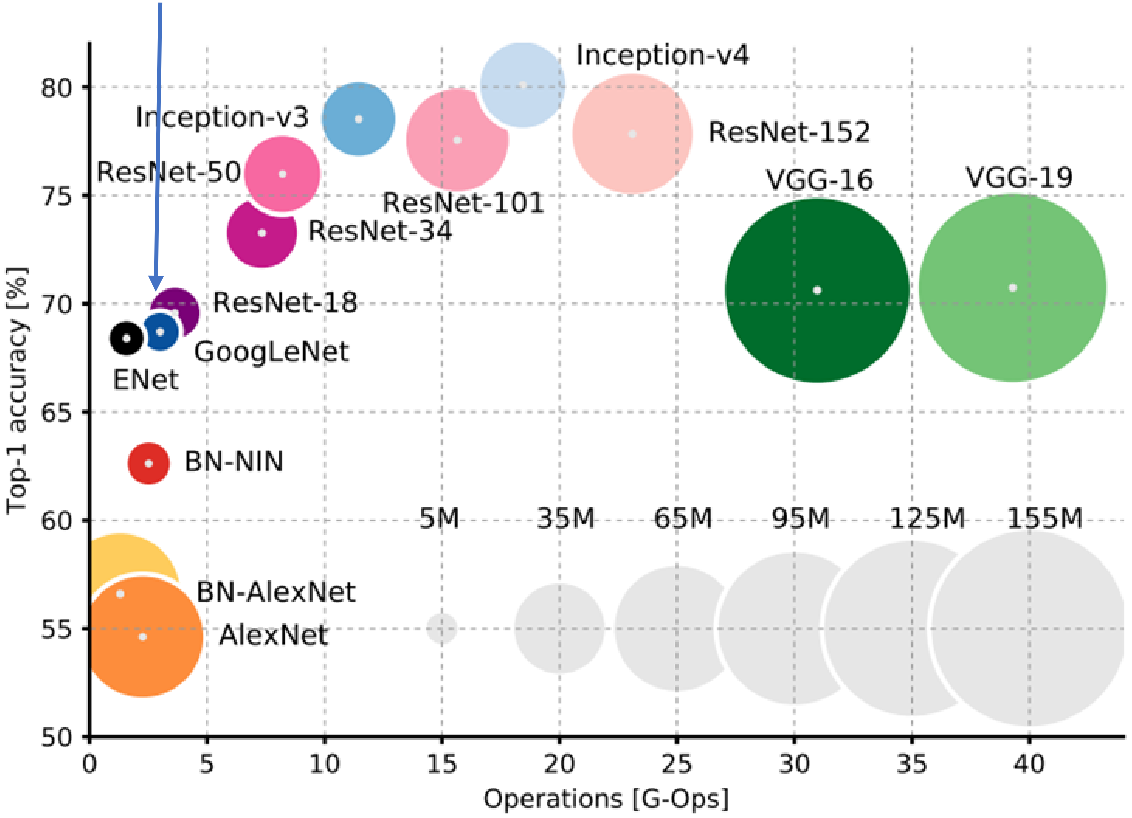


Canziani et al, "An analysis of deep neural network models for practical applications", 2017

Comparing Complexity

GoogLeNet:
Very efficient!

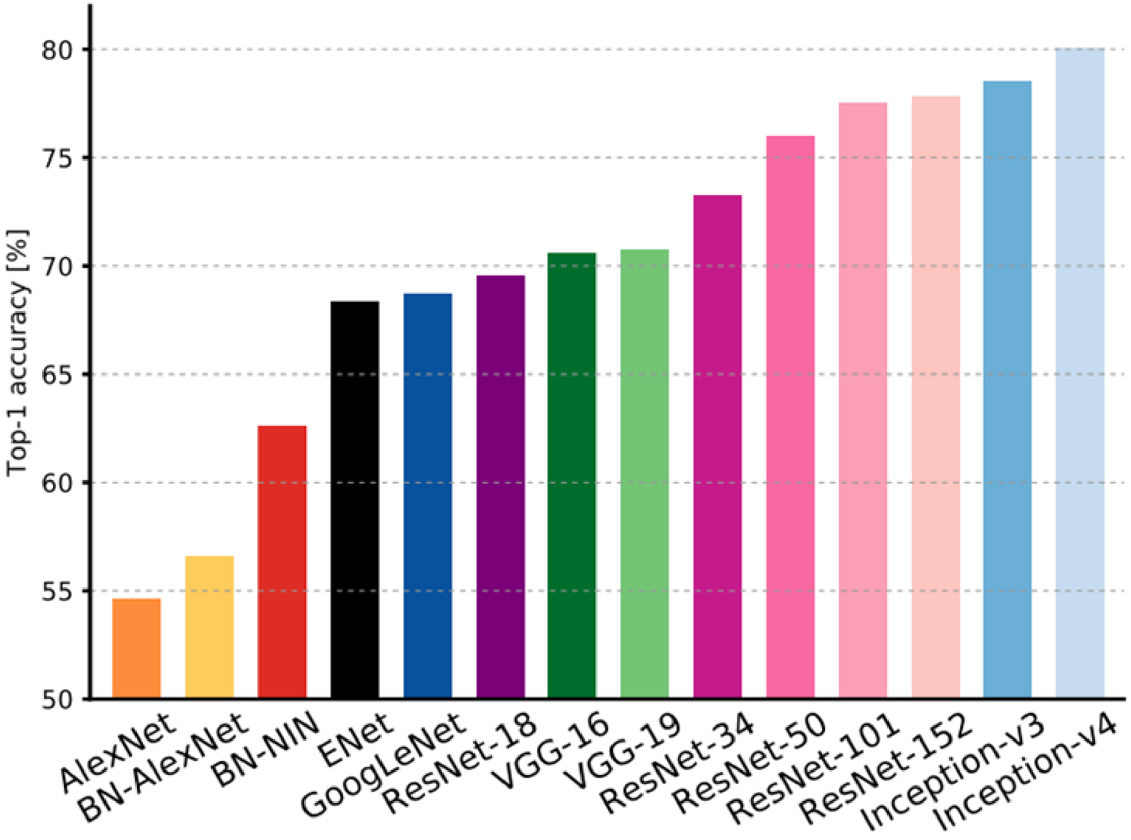


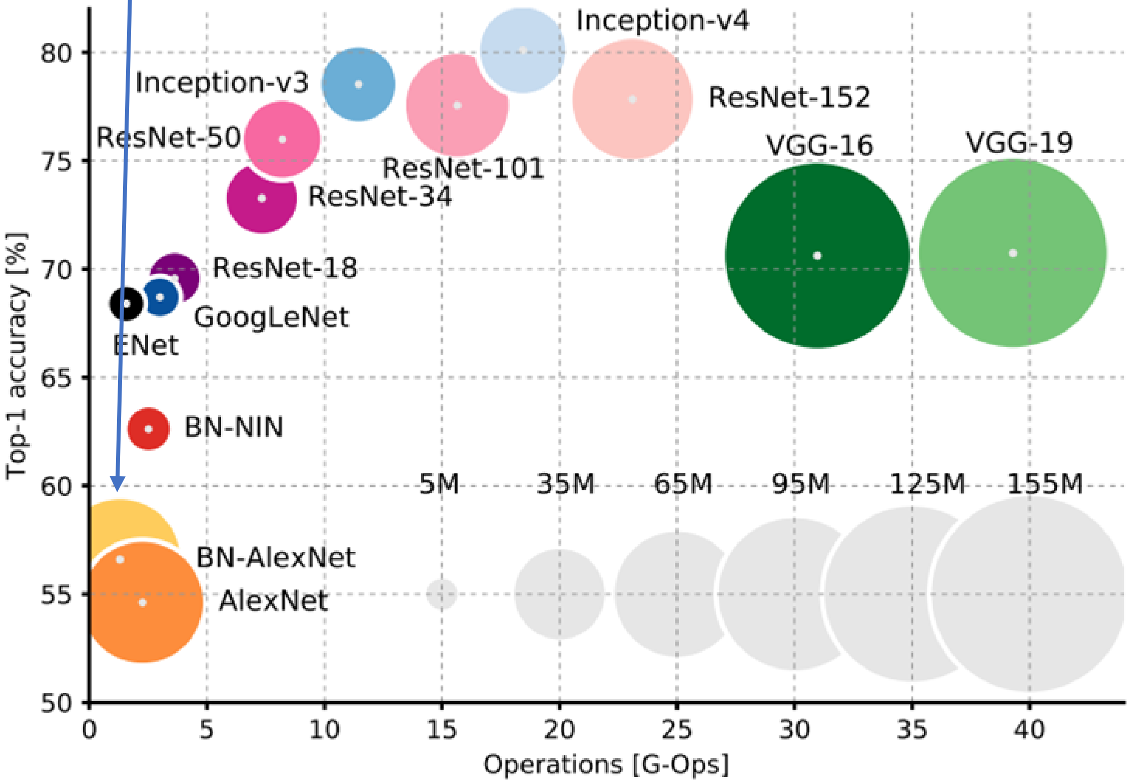


Canziani et al, “An analysis of deep neural network models for practical applications”, 2017

Comparing Complexity

AlexNet: Low compute, lots of parameters

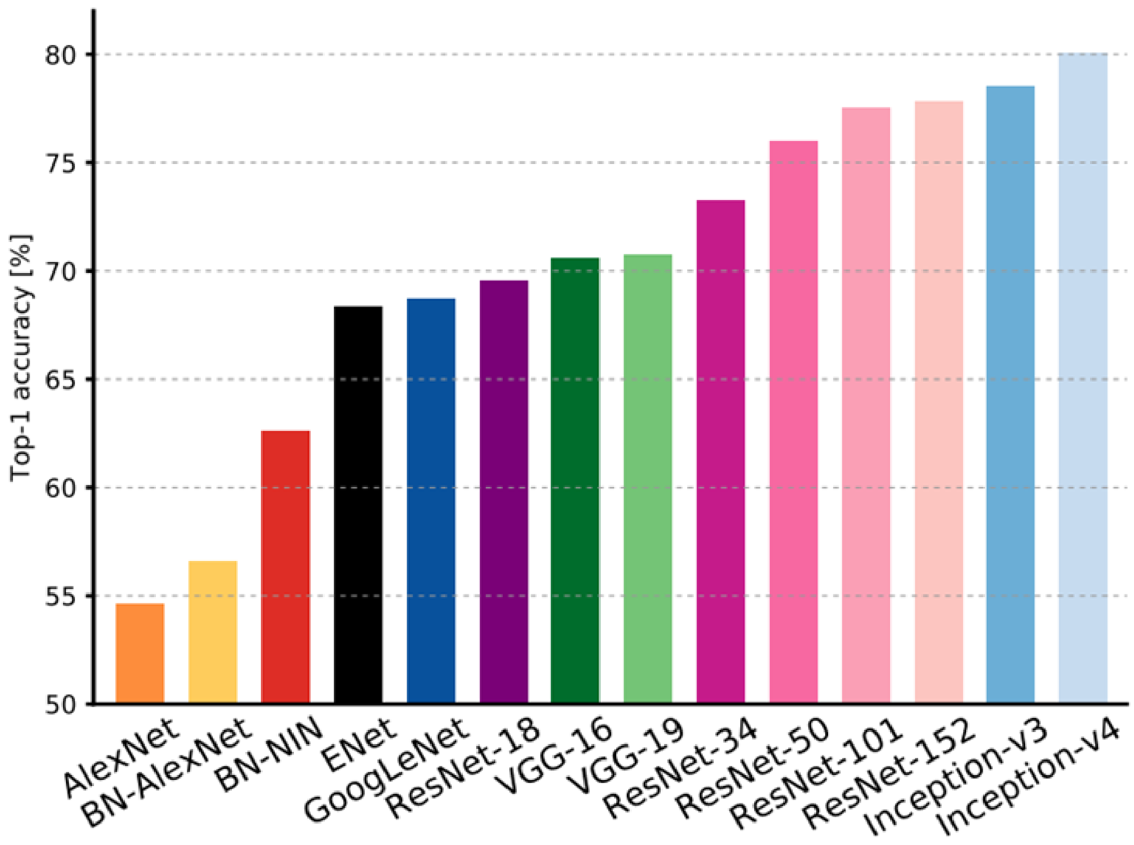


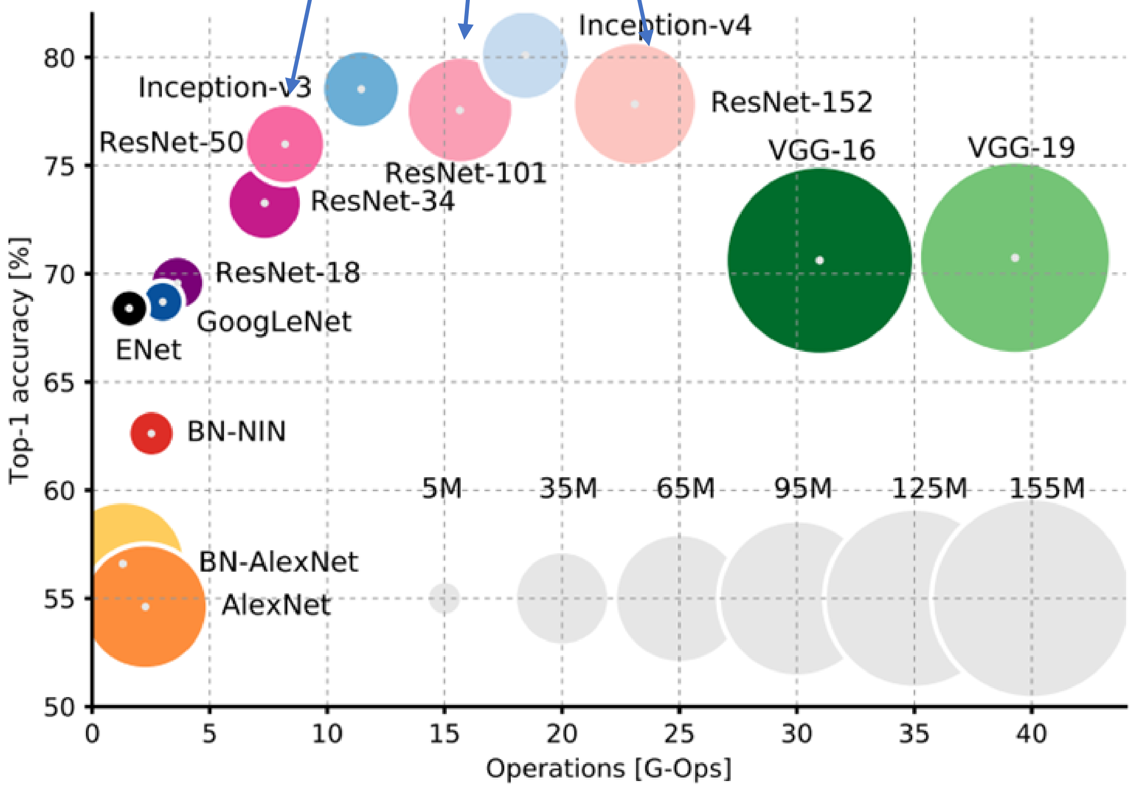


Canziani et al, "An analysis of deep neural network models for practical applications", 2017

Comparing Complexity

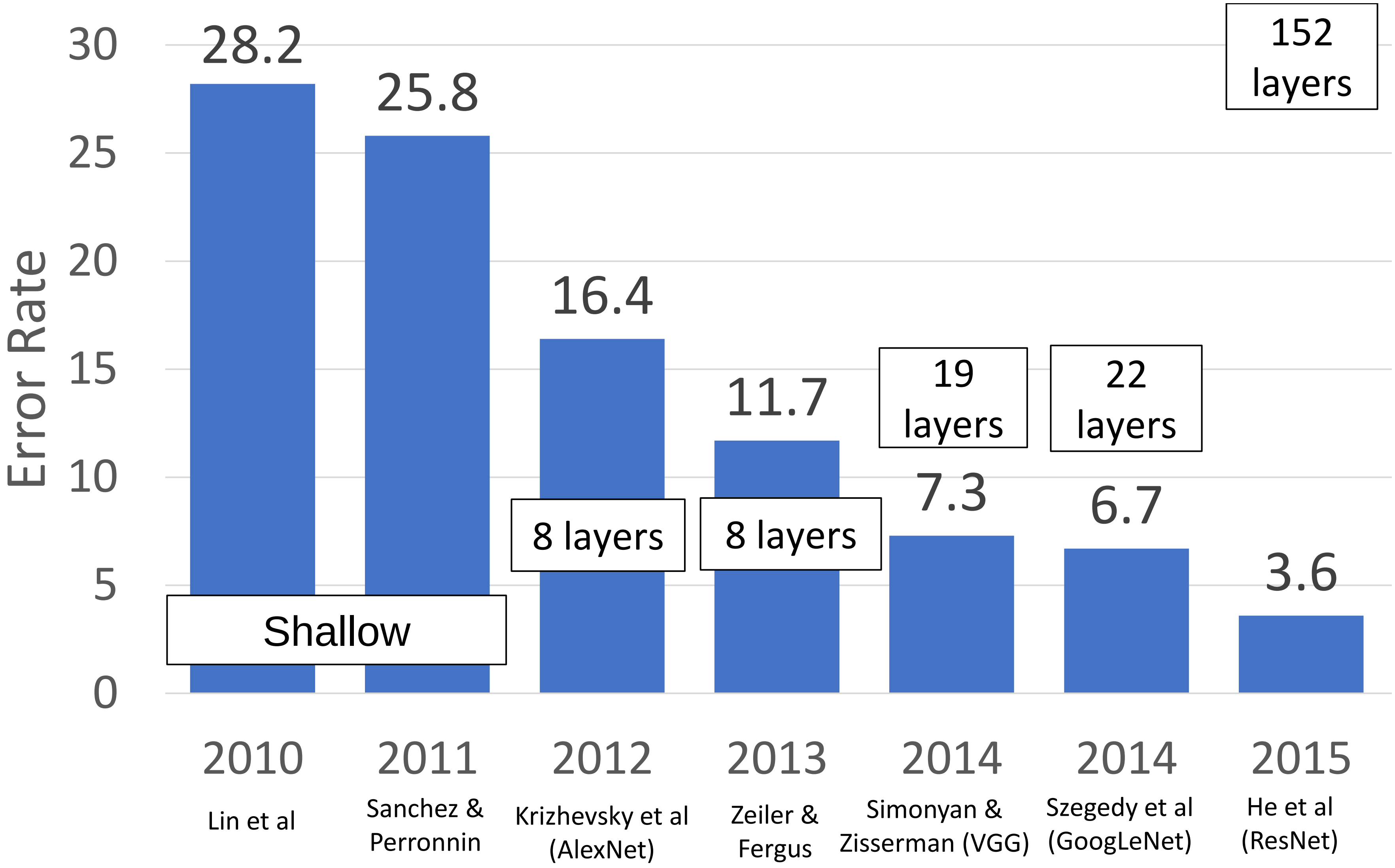
ResNet: Simple design, moderate efficiency, high accuracy





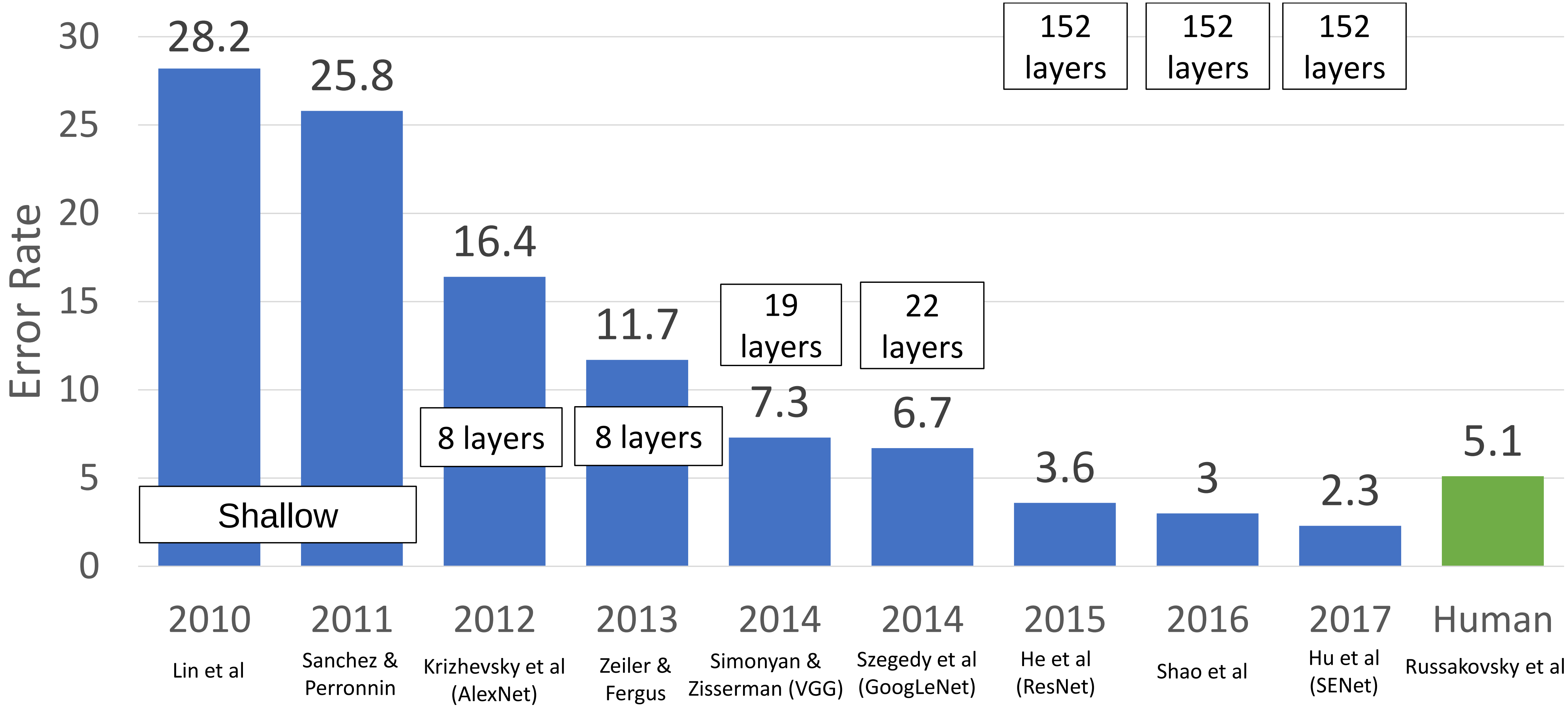
Canziani et al, “An analysis of deep neural network models for practical applications”, 2017

ImageNet Classification Challenge



CNN architectures have continued to evolve!

ImageNet Classification Challenge



Next: How to Train Your Networks