ROB 498/599: Deep Learning for Robot Perception (DeepRob)

Lecture 8: CNN Architectures

02/05/2025

https://deeprob.org/w25/

ROBOTICS

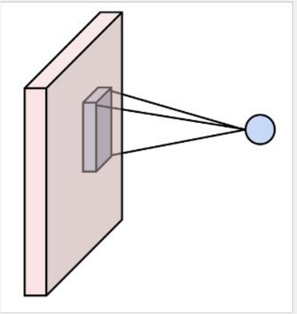
Today

- Feedback and Recap (5min)
- Convolutional Neural Network Architecture
  - LeNet, AlexNet, VGG, GoogLeNet (40min)
  - Residual Network (30min)
- Summary and Takeaways (5min)

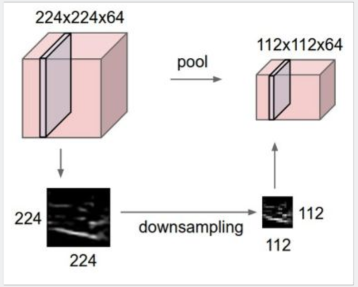
Recap: Components of Convolutional Networks

P2 released, due Feb. 16, 2025 - start NOW!!!

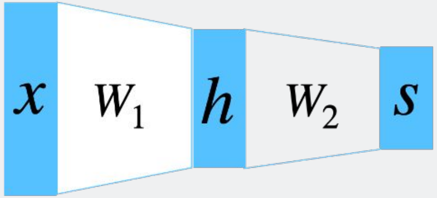
Convolution Layers



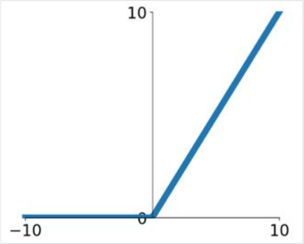
Pooling Layers



Fully-Connected Layers



Activation Function



Normalization

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

Question: How should we put them together?

ROBOTICS

Logistics - Vis Studio Rules

Room 1401 Duderstadt Center (https://xr.engin.umich.edu/visualization-studio/)
Desktop lab computer w/ 4090 GPU

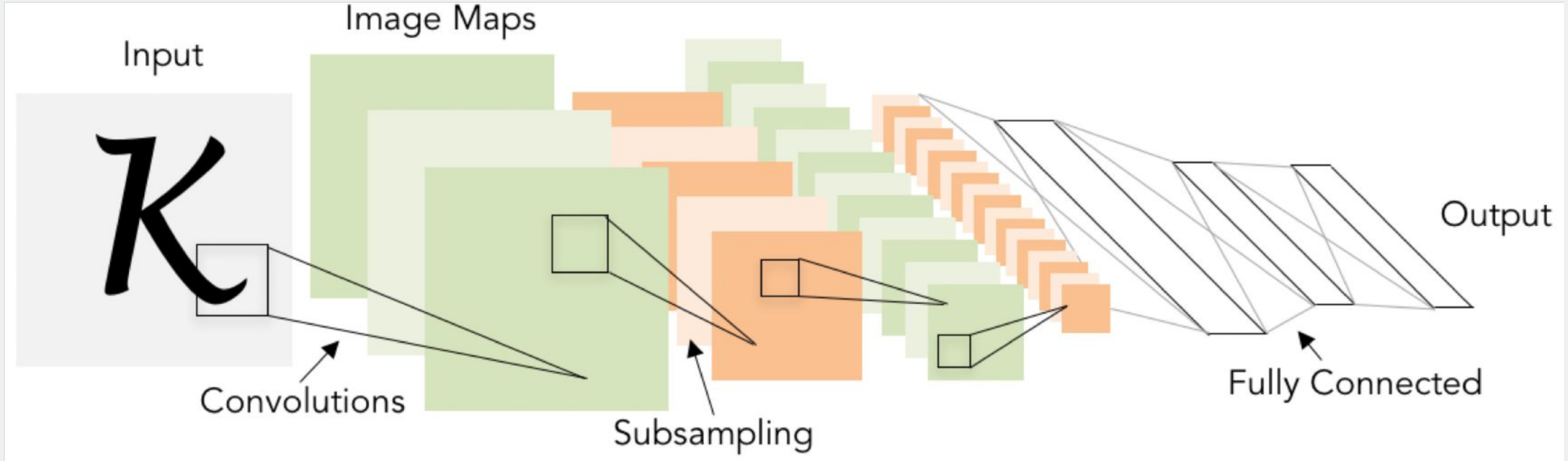
RULES:

1. Classes already scheduled and XR/VR/AR Projects have priority. When it is open hours/after hours/ weekends, it is walk-in.
2. Physically be there (e.g., during training) - Your account may be logged out after idle time.
3. Packages you download may only be local and temporary - re-download next time.

Convolutional Neural Networks

Classic architecture: [Conv, ReLU, Pool] x N, flatten, [FC, ReLU] x N, FC

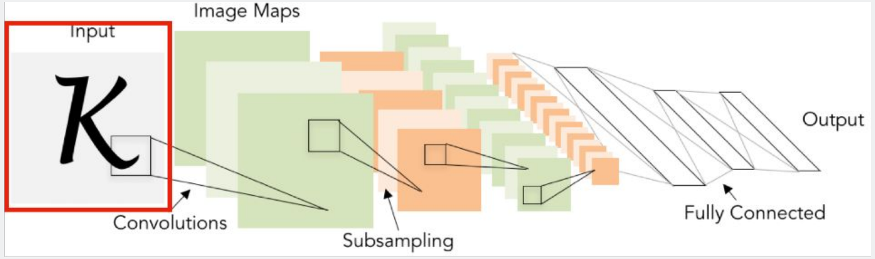
Example: LeNet-5



Lecun et al., “Gradient-based learning applied to document recognition”, 1998
http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf

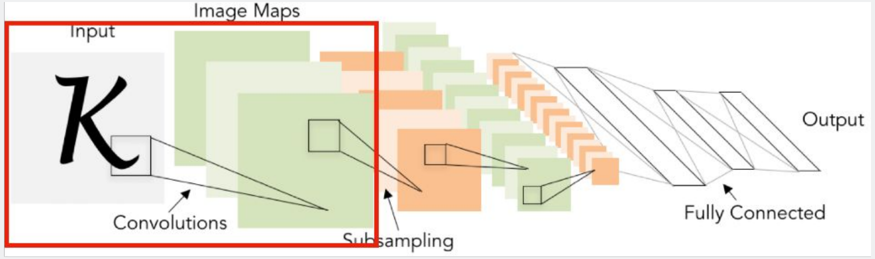
Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	



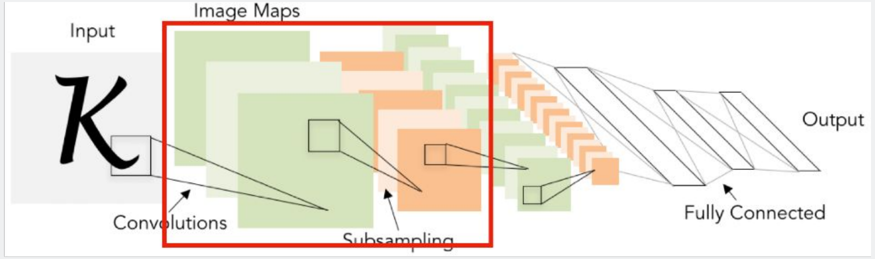
Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	



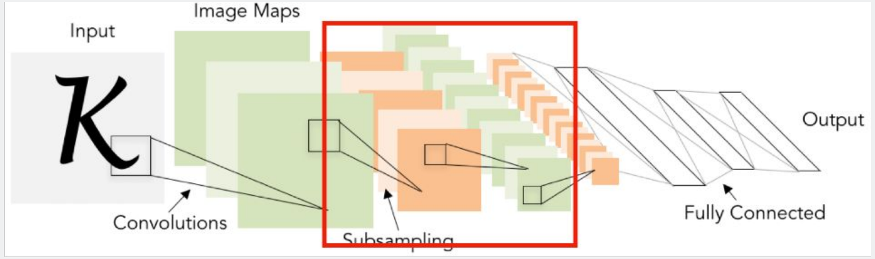
Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	



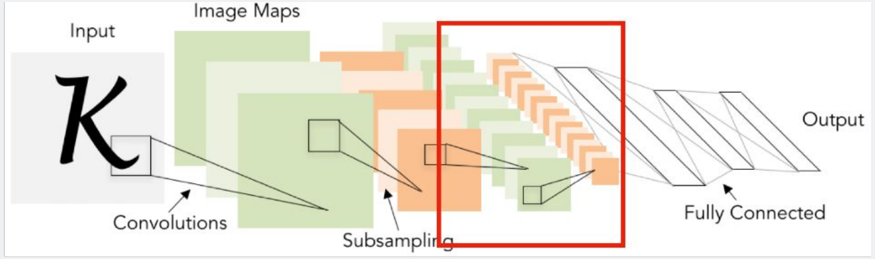
Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	



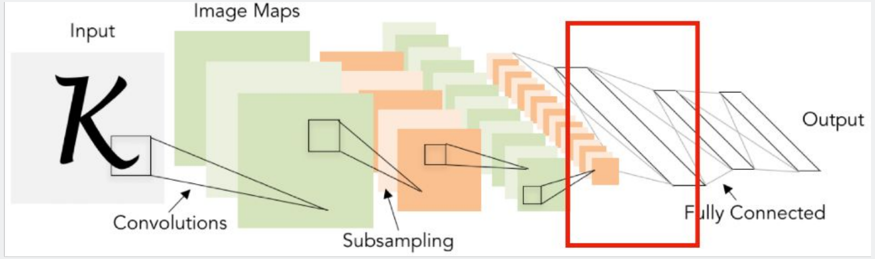
Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	



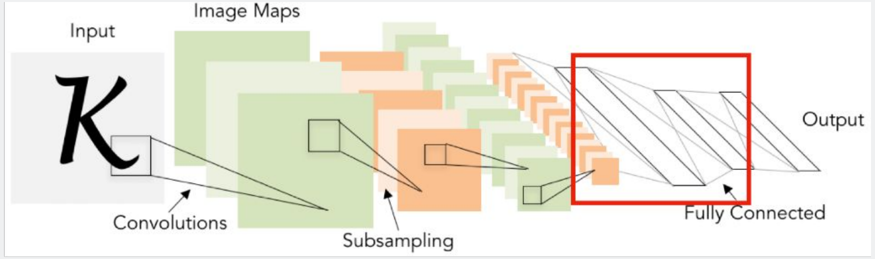
Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	



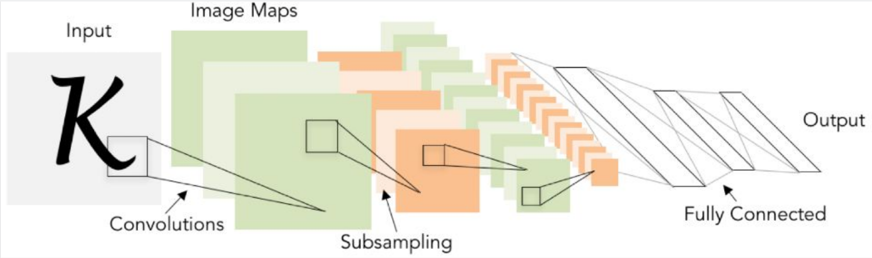
Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	
Linear (2450 -> 500)	500	2450 x 500
ReLU	500	



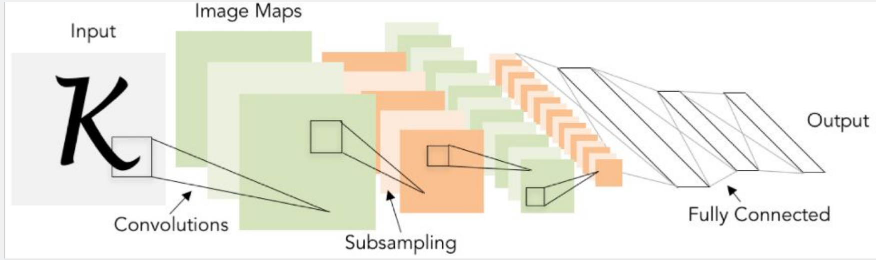
Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	
Linear (2450 -> 500)	500	2450 x 500
ReLU	500	
Linear (500 -> 10)	10	500 x 10



Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	
Linear (2450 -> 500)	500	2450 x 500
ReLU	500	
Linear (500 -> 10)	10	500 x 10



As we progress through the network:

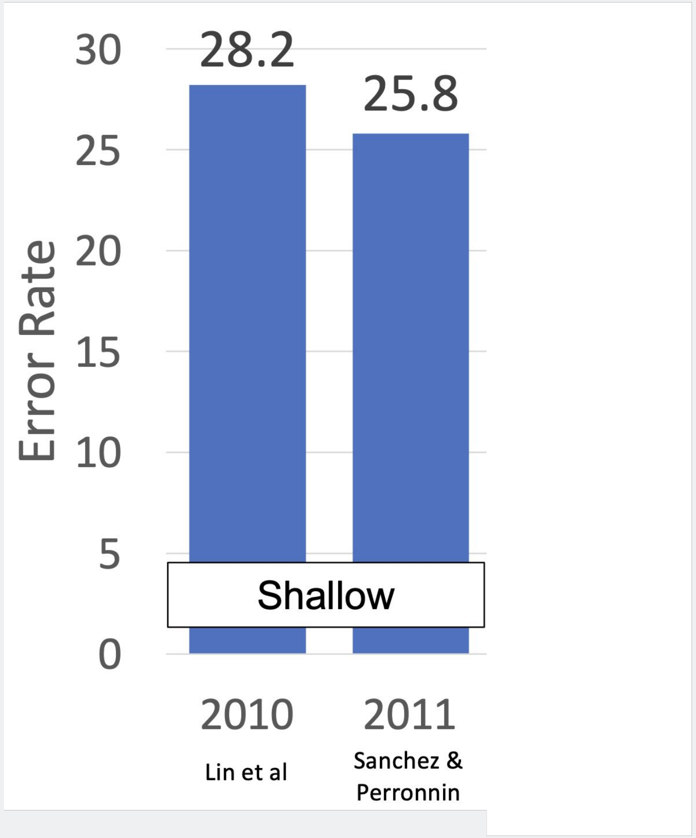
Spatial size **decreases**
(using pooling or striped convolution)

Number of channels **increases**
(total “volume” is preserved!)

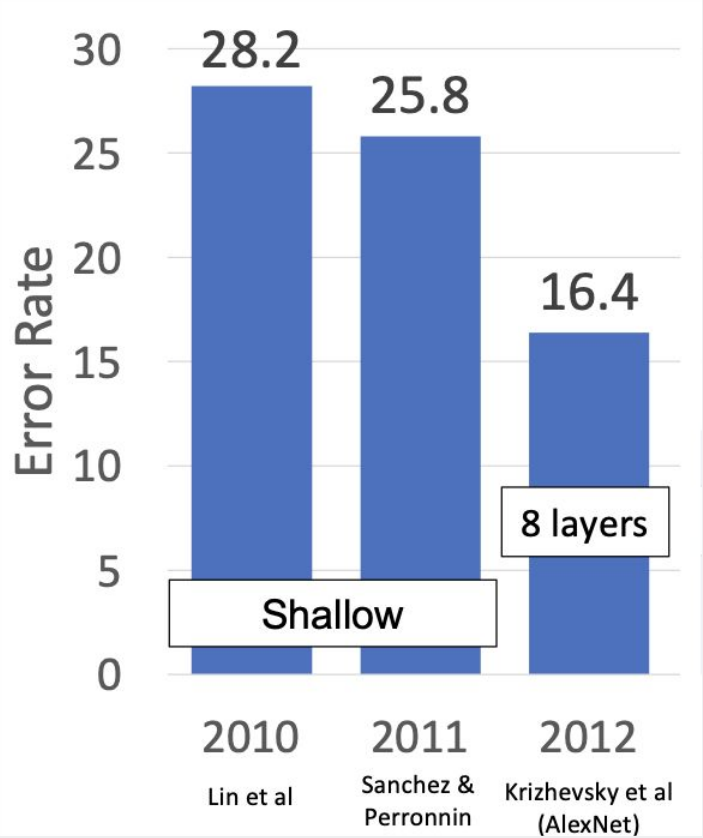
Some modern architectures break this trend

ROBOTICS

ImageNet Classification Challenge



ImageNet Classification Challenge



AlexNet

https://pytorch.org/hub/pytorch_vision_alexnet/

Also, early implementation in Caffe https://caffe.berkeleyvision.org/

Slightly truncated figure

Slides 227x227input size?

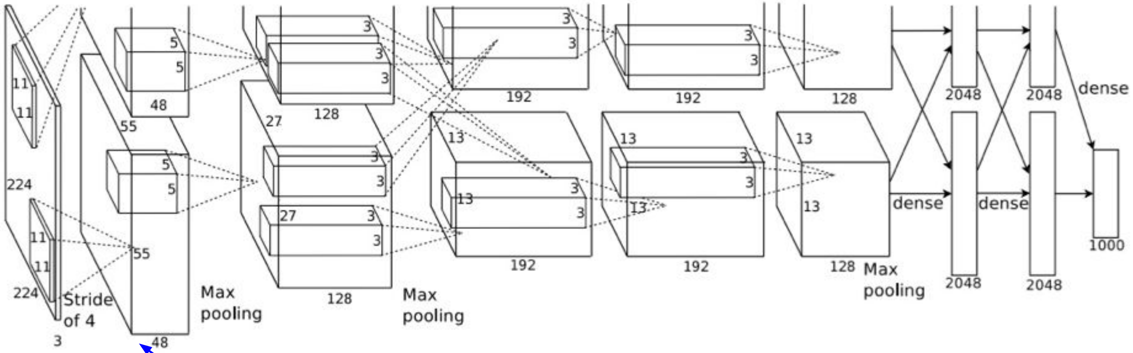
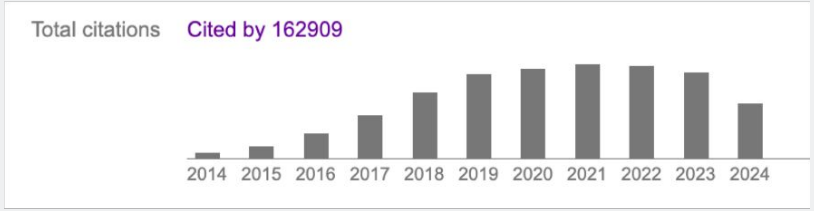


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input 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.

Reimplementation version, 64 filters

AlexNet

AlexNet citations per year
(as of 09/30/2024)



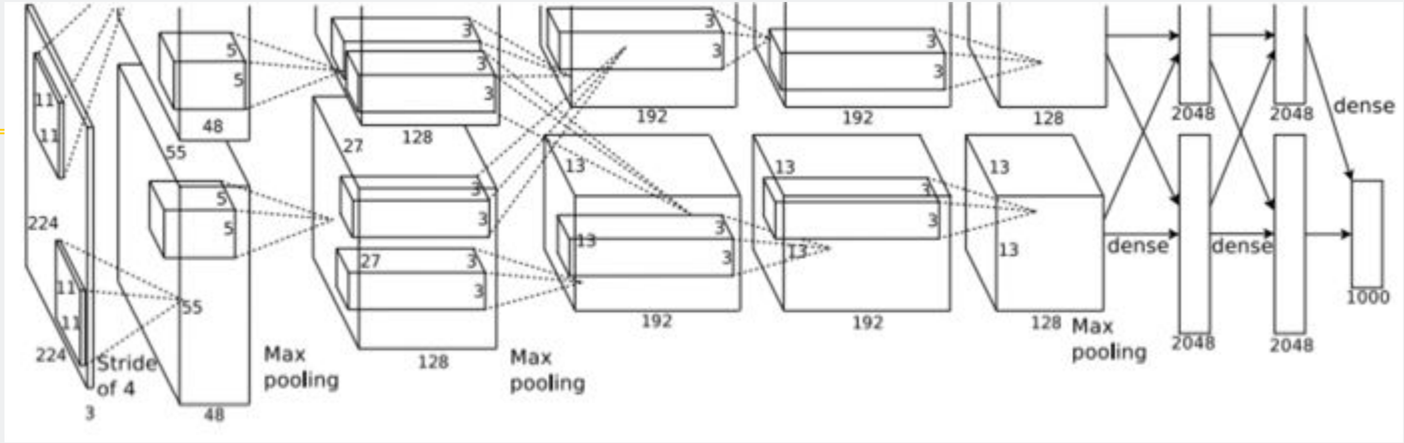
Total citations: **>160,000**

Citation Counts:

- Darwin, "On the origin of species," 1859: **60,117**
- Shannon, "A mathematical theory of communication," 1948: **156,791**
- Watson and Crick, "Molecular Structure of Nucleic Acids," 1953: **19,416**

Also Dropout paper ~55125 citations

AlexNet

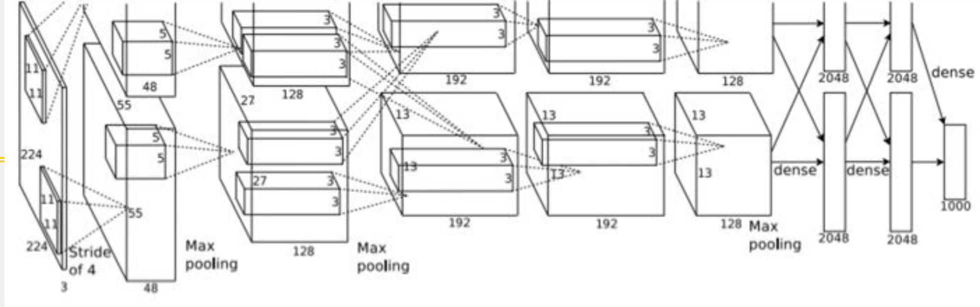


- 227 x 227 inputs
- 5 Convolutional Layers
- Max pooling
- 3 Fully-connected Layers
- ReLU nonlinearities

- Used “Local response normalization”; Not used anymore

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

AlexNet

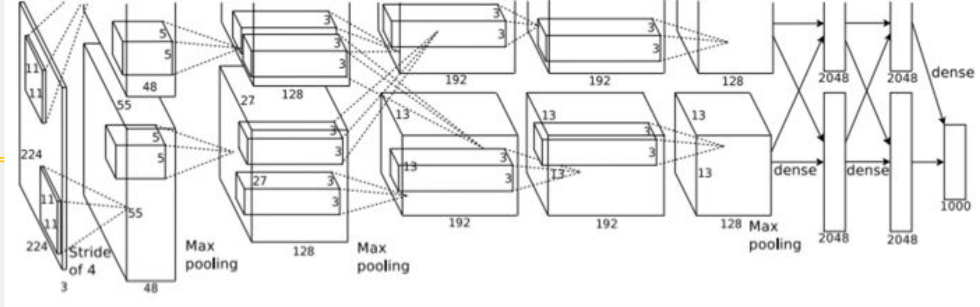


	Input size		Layer				Output size	
Layer	C	H/W	Filters	Kernel	Stride	Pad	C	H/W
Conv1	3	227	64	11	4	2	?	

Recall: Output channels = number of filters

Recall: W' = (W - K + 2P) / S + 1
= (227 - 11 + 2 x 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	?

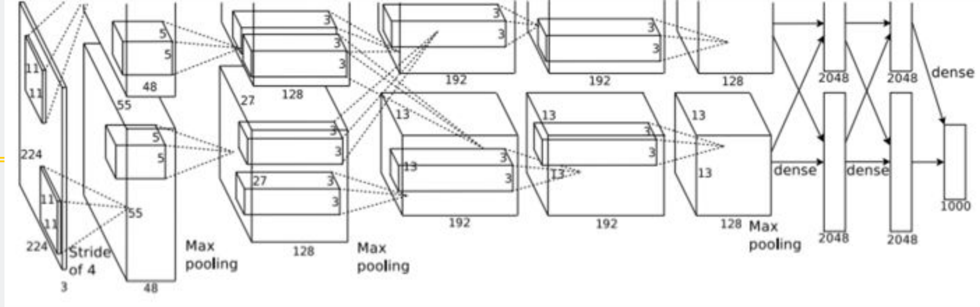
784

Number of output elements = C x H’ x W’
= 64 x 56 x 56 = 200,704

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

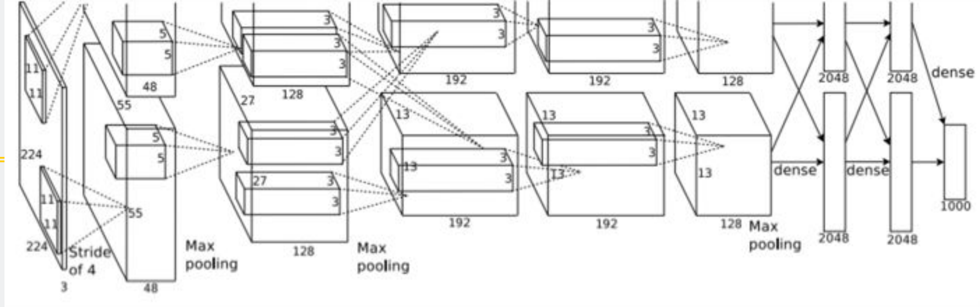
KB = (number of elements) x (bytes per elem) /1024
= 200704 x 4 / 1024
= 784

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	?	?

AlexNet



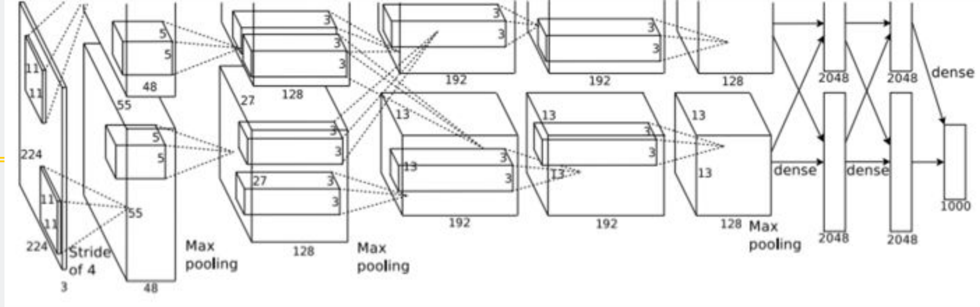
	Input size		Layer				Output size			
Layer	C	H/W	Filters	Kernel	Stride	Pad	C	H/W	Memory (KB)	Params (k)
Conv1	3	227	64	11	4	2	64	56	784	23

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

Bias shape = $C_{out} = 64$

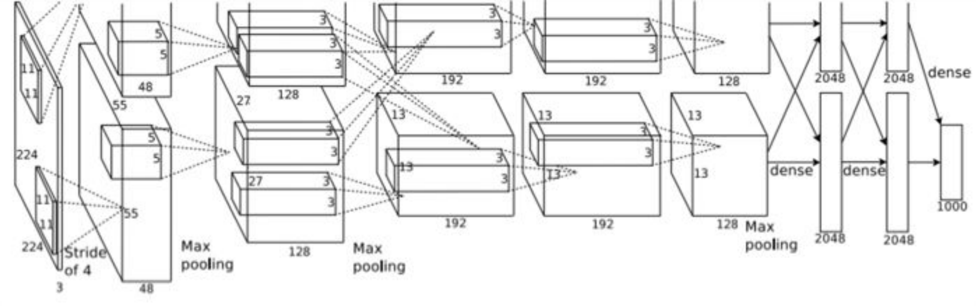
Number of weights = $64 \times 3 \times 11 \times 11 + 64$
= **23,296**

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)

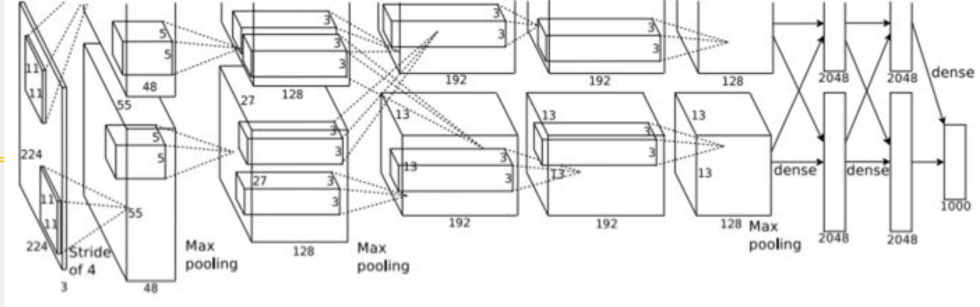
= $(C_{out} \times H' \times W') * (C_{in} \times K \times K)$

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

= 200,704 * 363

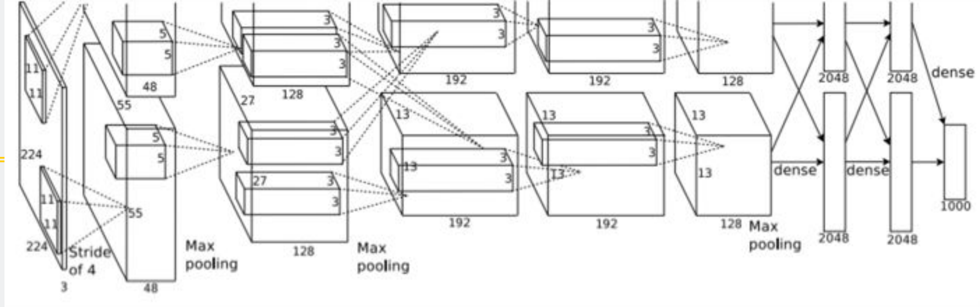
= **72,855,552**

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



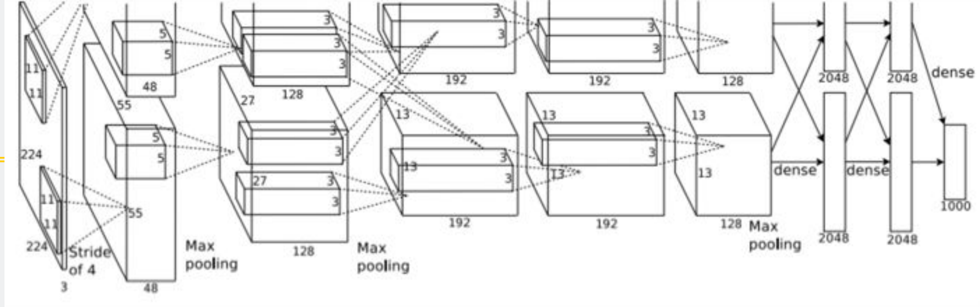
	Input size		Layer				Output size				
Layer	C	H/W	Filters	Kernel	Stride	Pad	C	H/W	Memory (KB)	Params (k)	Flop (M)
Conv1	3	227	64	11	4	2	64	56	784	23	73
Pool1	64	56		3	2	0	64	27			

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	?	

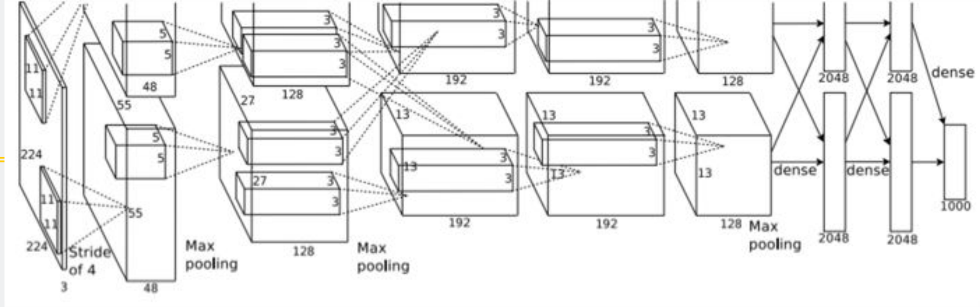
#output elms = C_{out} x H' x W'
Bytes per elem = 4
KB = C_{out} x H' x W' x 4 / 1024
= 64 * 27 * 27 * 4 / 1024
= **182.25**

Q: How many parameters for the pooling layer?

Aha Slides (In-class participation)

https://ahaslides.com/DFZE4

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

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

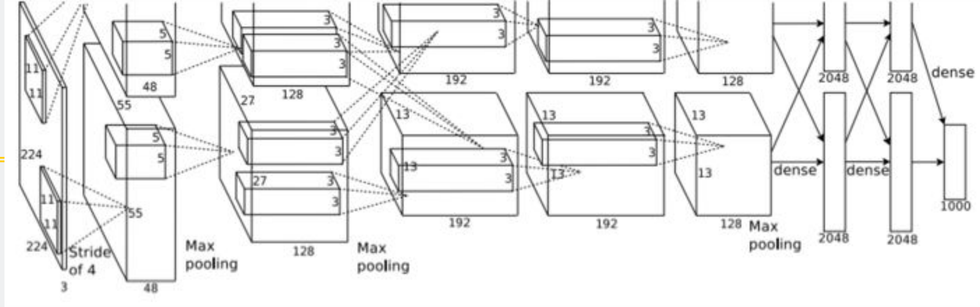
= $(C_{out} \times H' \times W') \times (K \times K)$

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

= 419,904

= **0.4 MFLOP**

AlexNet

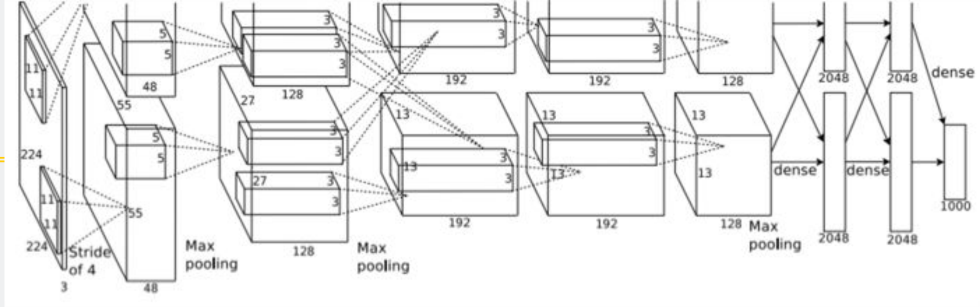


	Input size		Layer				Output size				
Layer	C	H/W	Filters	Kernel	Stride	Pad	C	H/W	Memory (KB)	Params (k)	Flop (M)
Conv1	3	227	64	11	4	2	64	56	784	23	73
Pool1	64	56		3	2	0	64	27	182	0	0
Conv2	64	27	192	5	1	2	192	27	547	307	224
Pool2	192	27		3	2	0	192	13	127	0	0
Conv3	192	13	384	3	1	1	384	13	254	664	112
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					???		36	0	0

Flatten output size https://ahaslides.com/DFZE4

ROBOTICS

AlexNet



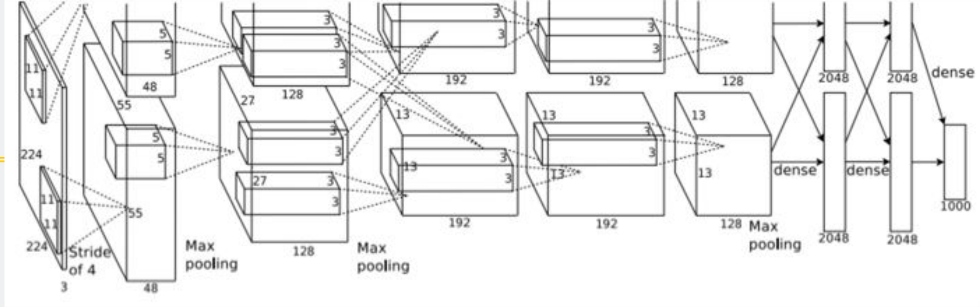
	Input size		Layer				Output size				
Layer	C	H/W	Filters	Kernel	Stride	Pad	C	H/W	Memory (KB)	Params (k)	Flop (M)
Conv1	3	227	64	11	4	2	64	56	784	23	73
Pool1	64	56		3	2	0	64	27	182	0	0
Conv2	64	27	192	5	1	2	192	27	547	307	224
Pool2	192	27		3	2	0	192	13	127	0	0
Conv3	192	13	384	3	1	1	384	13	254	664	112
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	37726	38

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

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

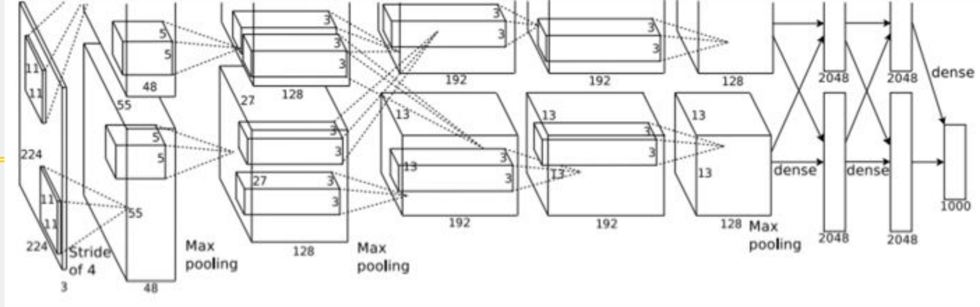
ROBOTICS

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	37726	38
FC7	4096		4096				4096		16	16777	17
FC8	4096		1000				1000		4	4096	4

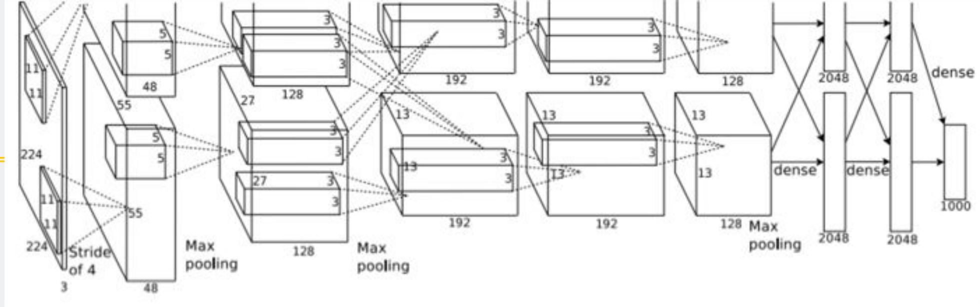
AlexNet



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	37726	38
FC7	4096		4096				4096		16	16777	17
FC8	4096		1000				1000		4	4096	4

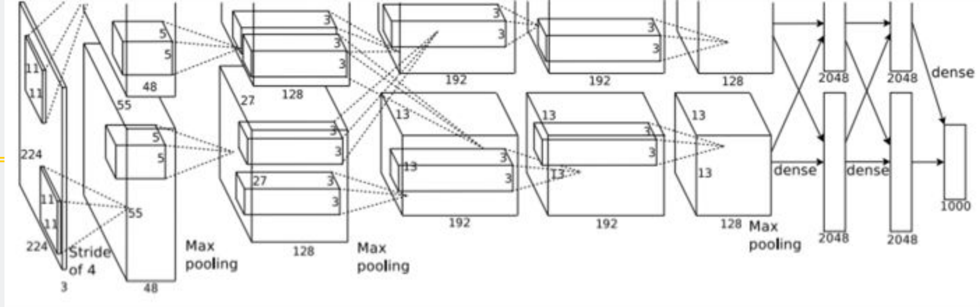
AlexNet



	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	37726	38
FC7	4096		4096				4096		16	16777	17
FC8	4096		1000				1000		4	4096	4

Interesting trends here!

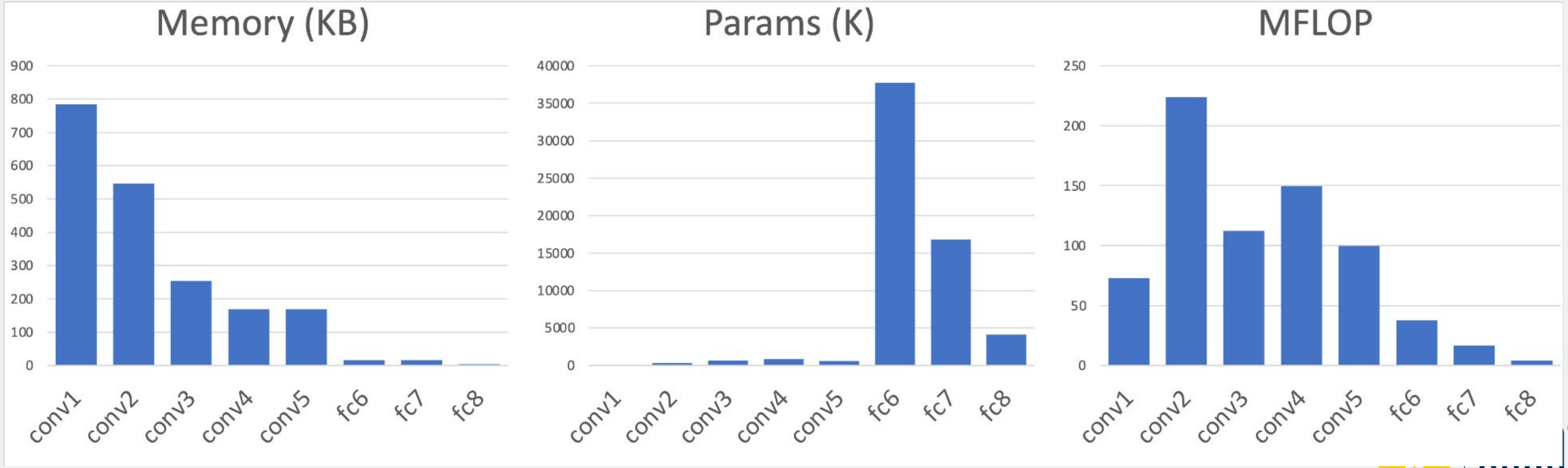
AlexNet



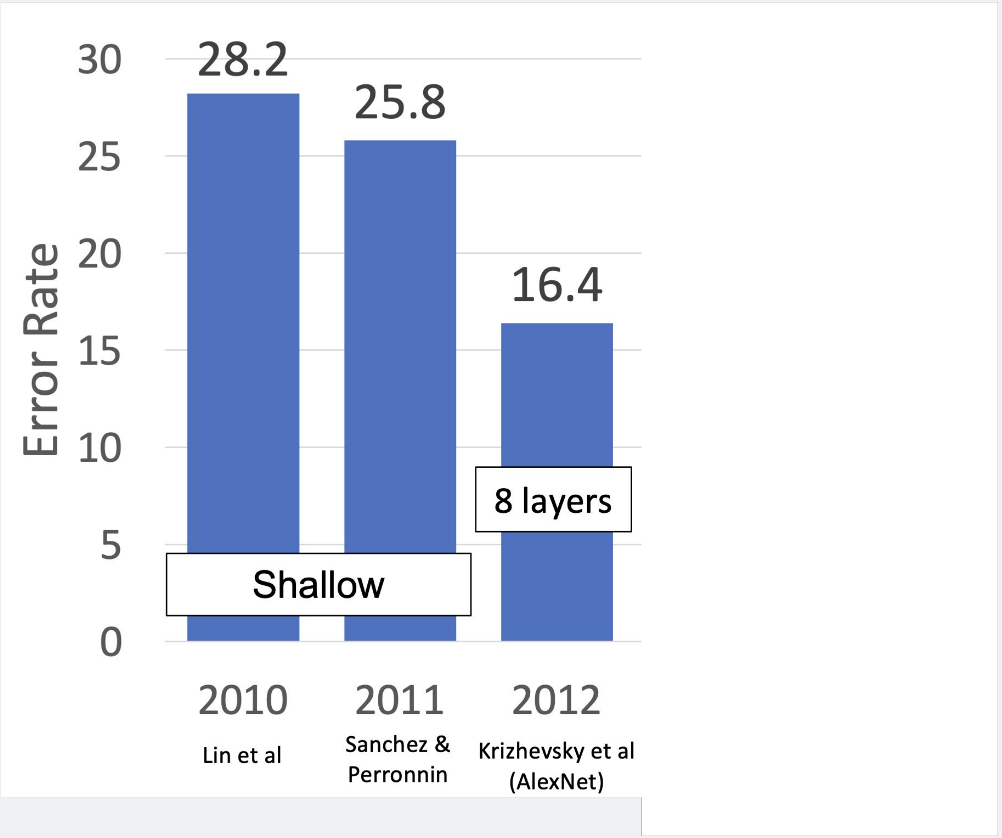
Most of the **memory usage** in the early convolution layers

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

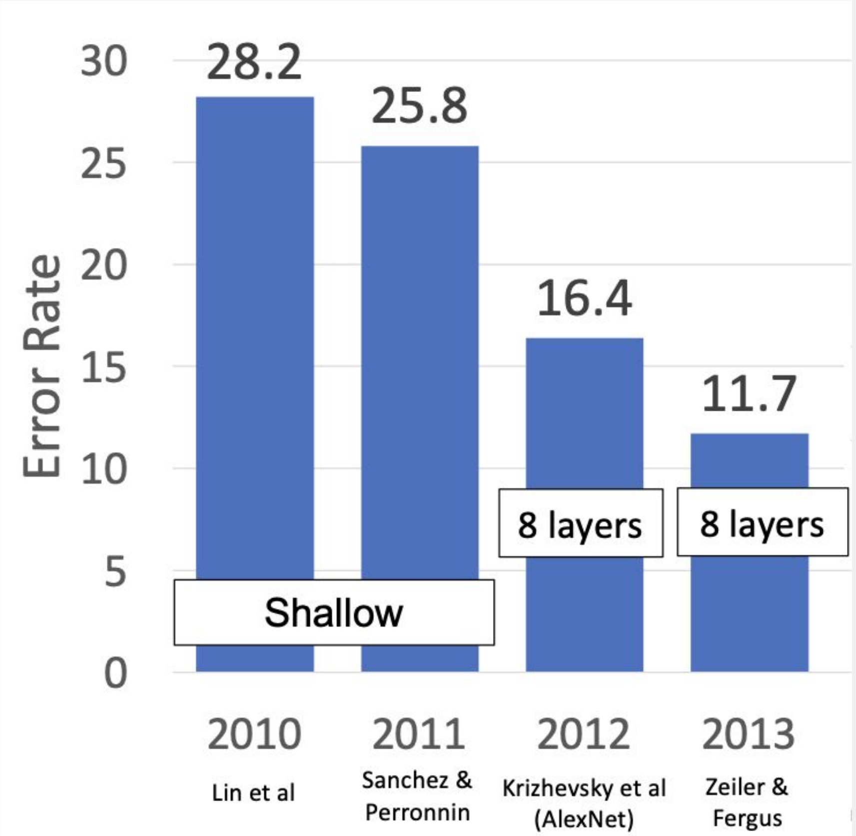
Most **floating-point ops** occur in the convolution layers



ImageNet Classification Challenge

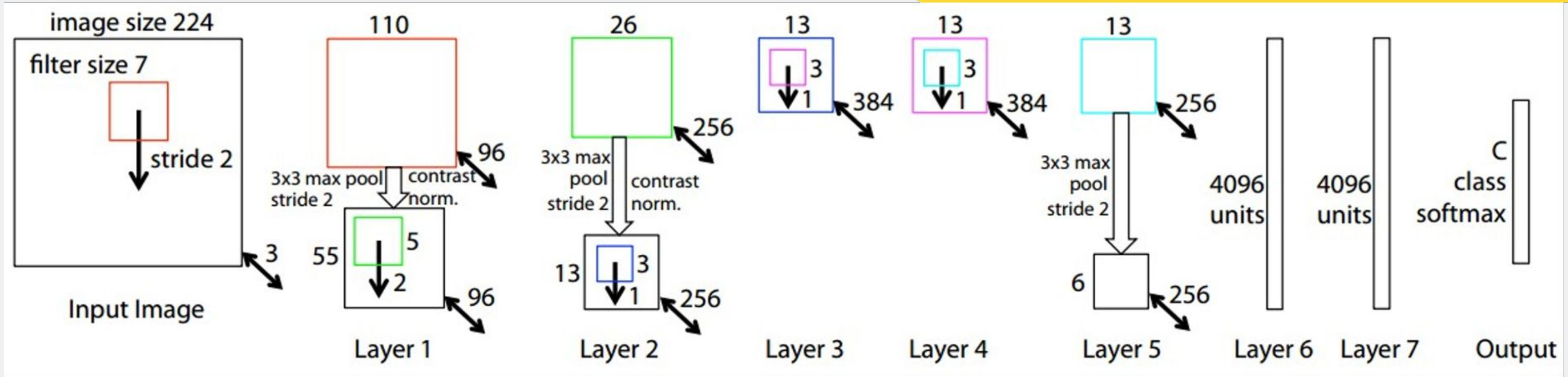


ImageNet Classification Challenge



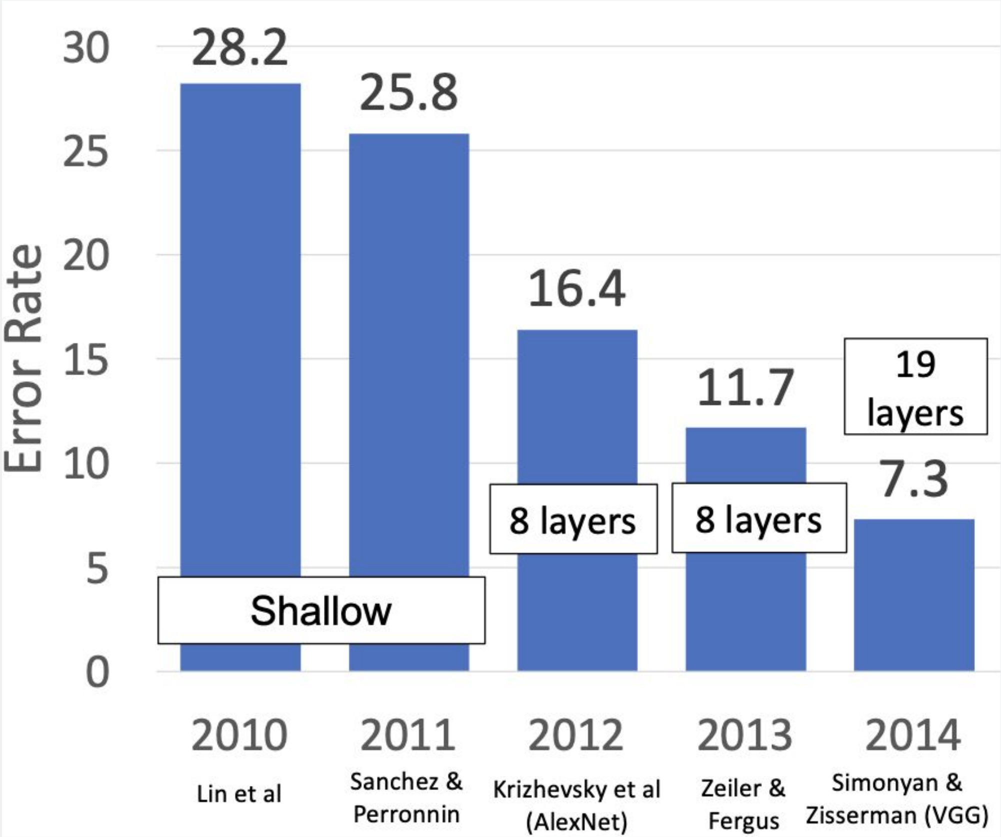
ZFNet: A Bigger AlexNet

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 :(

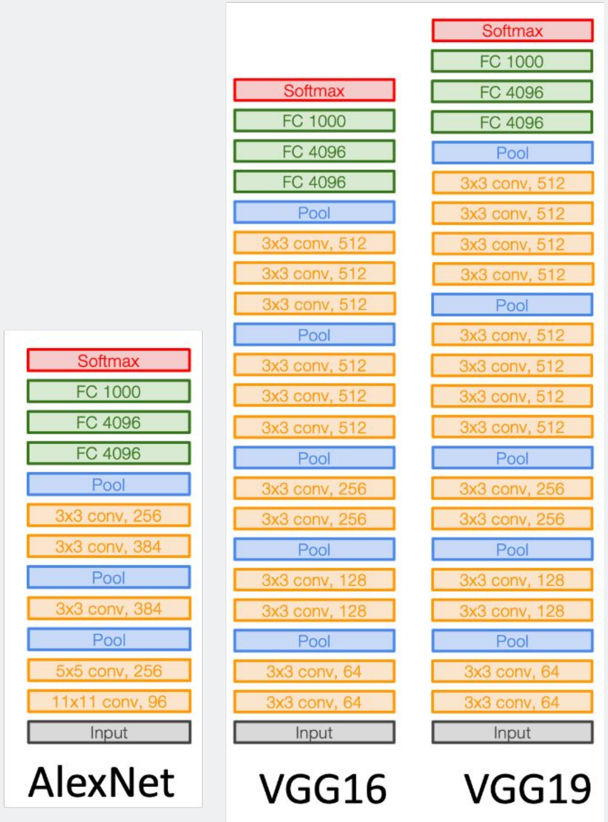
ImageNet Classification Challenge



ROBOTICS

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



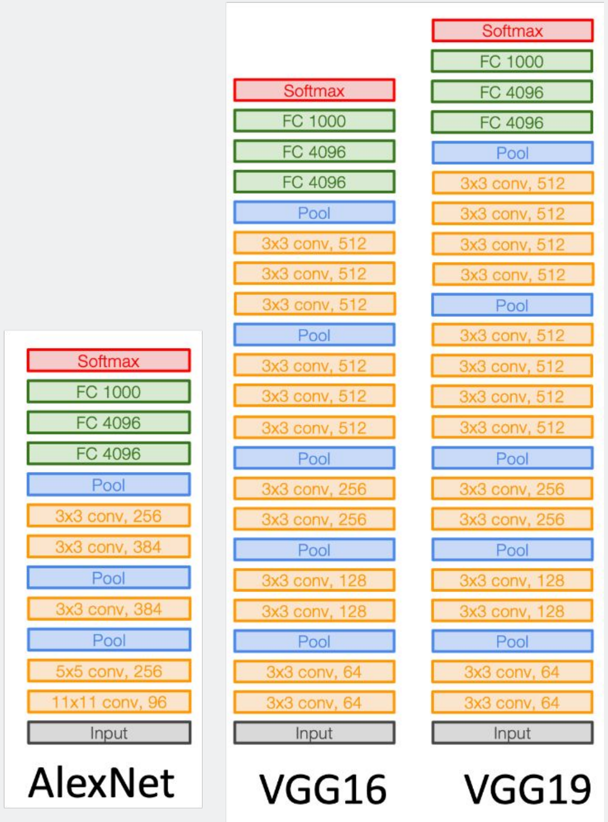
Simonyan and Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR 2015 https://arxiv.org/abs/1409.1556

ROBOTICS

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 convolution **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: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

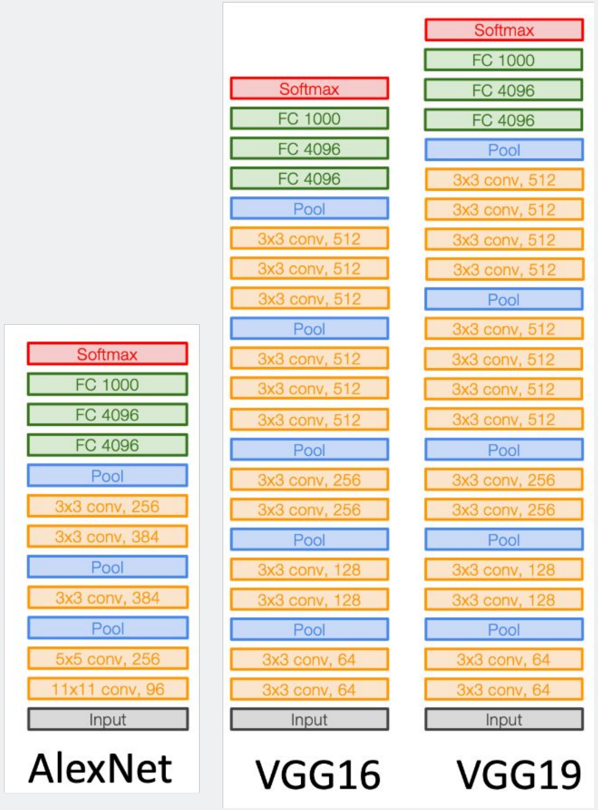
After pool, double #channels

<u>Option 1:</u>

Conv(5x5, C->C)

Q: How many parameters?

Q: How many FLOPs?

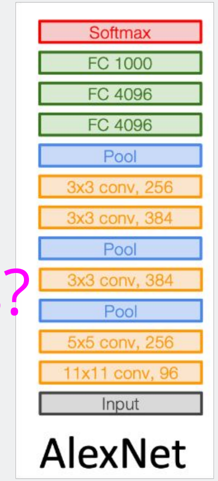


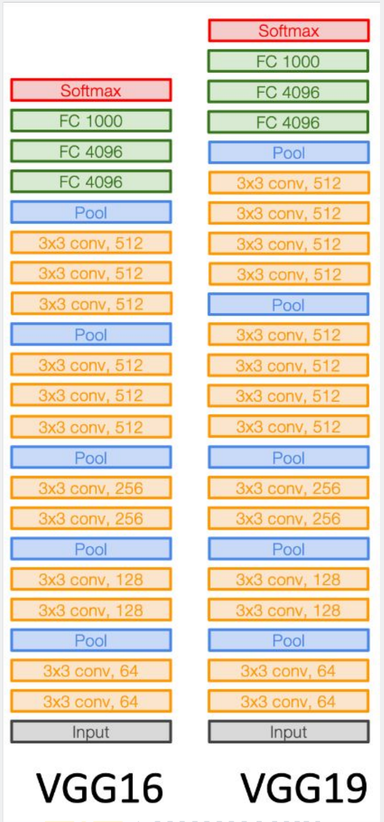
VGG: Deeper Networks, Regular Design

VGG Design rules:
All conv are 3x3 stride 1 pad 1
All max pool are 2x2 stride 2
After pool, double #channels

Option 2:
Conv(3x3, C->C)
Conv(3x3, C->C)

Q: How many parameters?
Q: How many FLOPs?





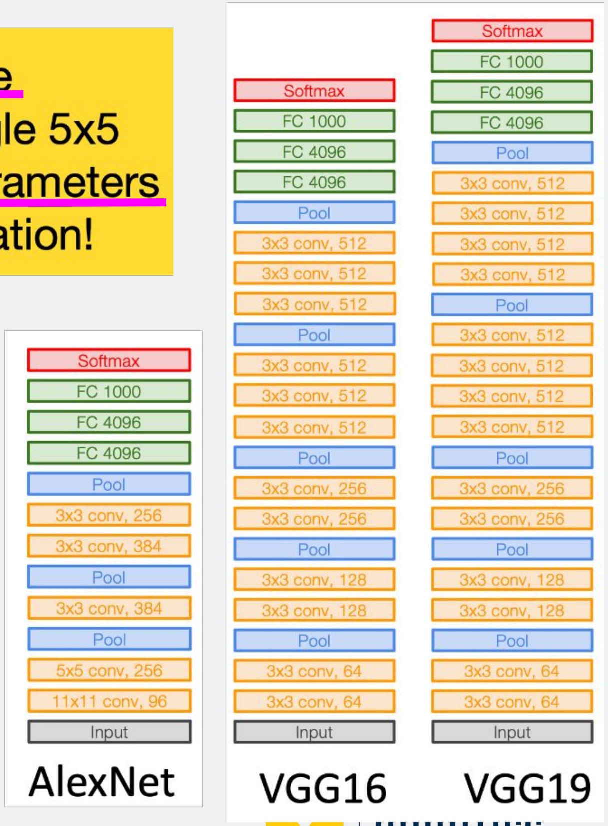
VGG: Deeper Networks, Regular Design

VGG Design rules:
All conv are 3x3 stride 1 pad 1
All max pool are 2x2 stride 2
After pool, double #channels

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

Option 1:
Conv(5x5, C->C)

Option 2:
Conv(3x3, C->C)
Conv(3x3, C->C)

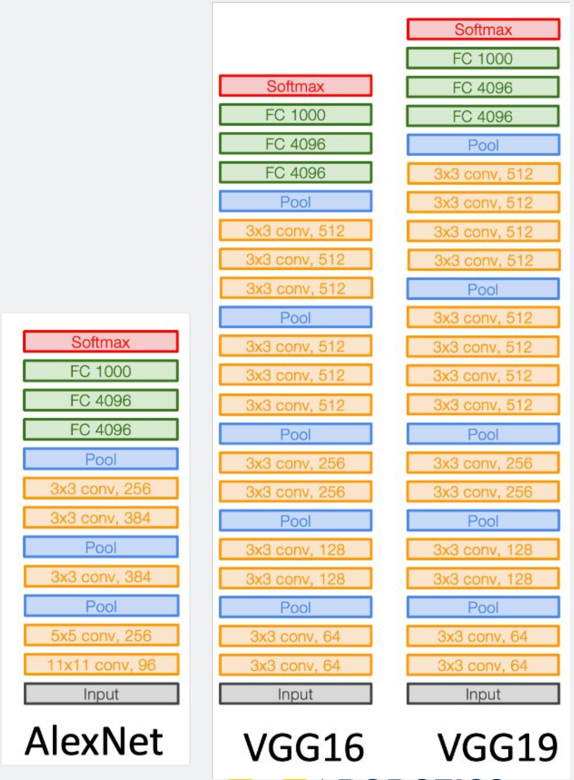


VGG: Deeper Networks, Regular Design

VGG Design rules:
All conv are 3x3 stride 1 pad 1
All max pool are 2x2 stride 2
After pool, double #channels

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

Memory: 4HWC
Params: $9C^2$
FLOPs: $36HWC^2$



VGG: Deeper Networks, Regular Design

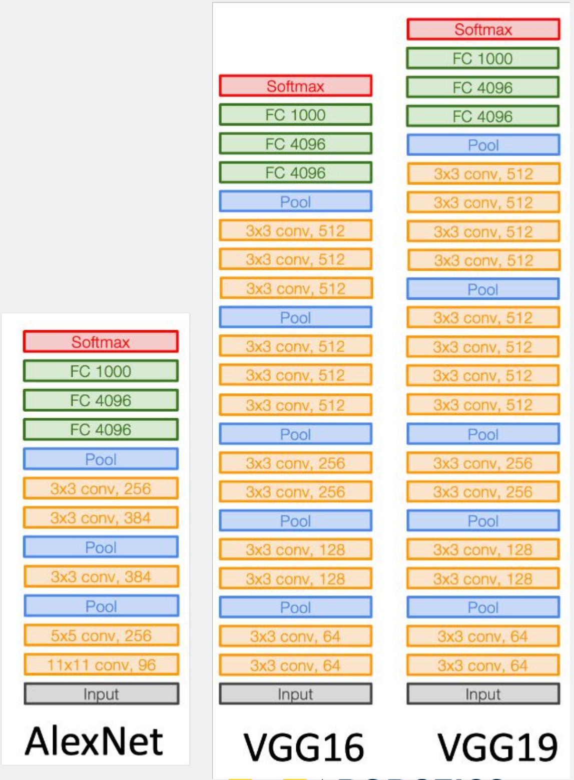
<u>VGG Design rules:</u>
All conv are 3x3 stride 1 pad 1
All max pool are 2x2 stride 2
After pool, double #channels

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

Memory: 4HWC
Params: $9C^2$
FLOPs: $36HWC^2$

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

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



ROBOTICS

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

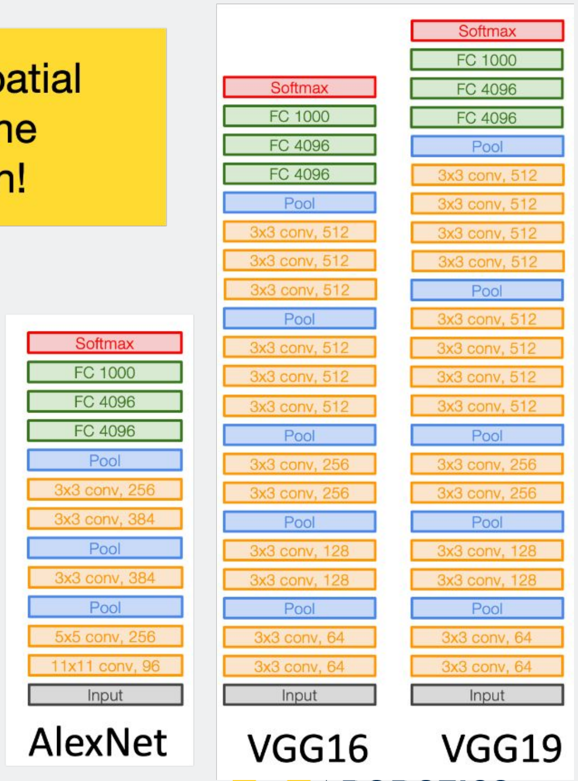
Conv layers at each spatial resolution take the same amount of computation!

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

Memory: 4HWC
Params: $9C^2$
FLOPs: $36HWC^2$

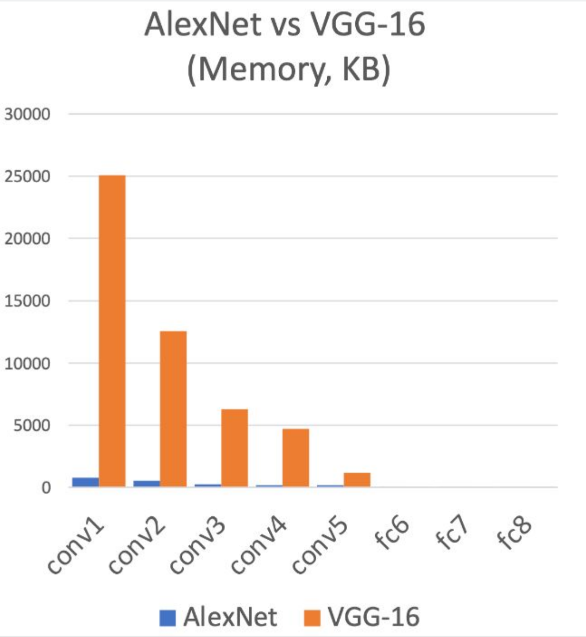
<u>Input:</u> 2C x H x W
<u>Layer:</u> Conv(3x3, 2C->2C)

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

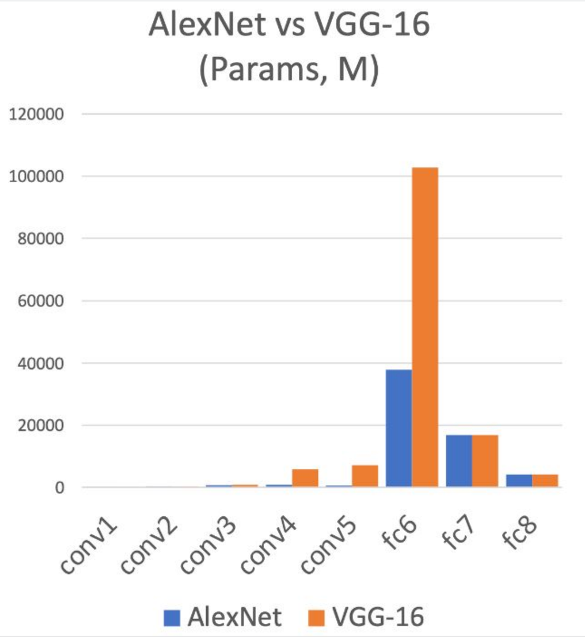


AlexNet vs VGG-16

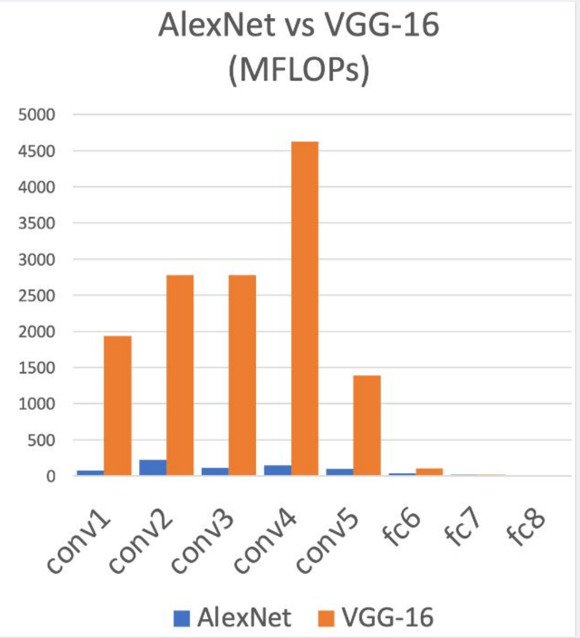
Much bigger network!



AlexNet total: 1.9MB
VGG-16 total: 48.6MB (25x)

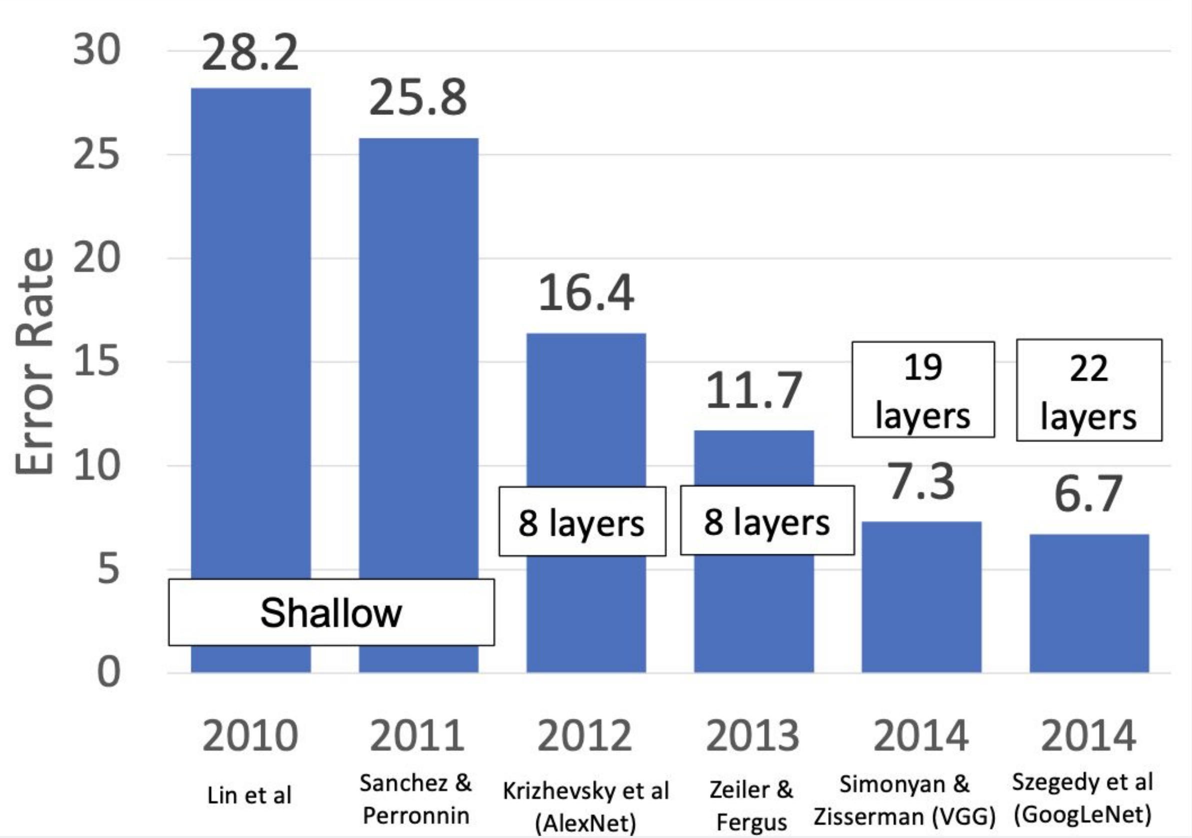


AlexNet total: 61M
VGG-16 total: 138M (2.3x)



AlexNet total: 0.7 GFLOP
VGG-16 total: 13.6 GFLOP (19.4x)

ImageNet Classification Challenge

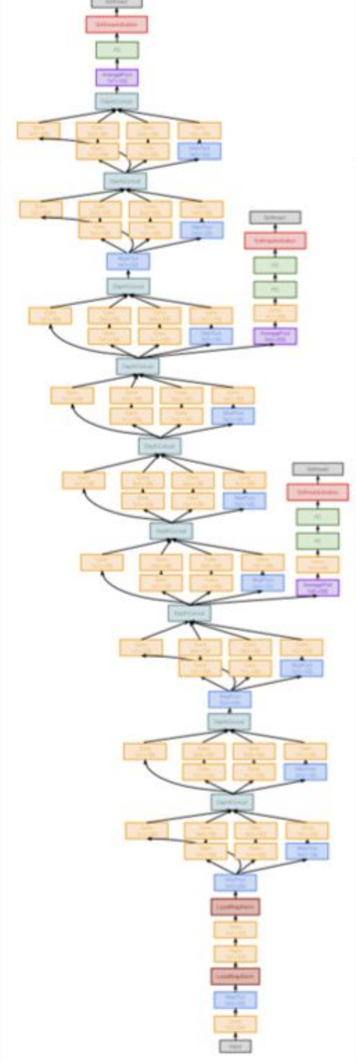


GoogLeNet: Focus on Efficiency

Many innovations for efficiency:

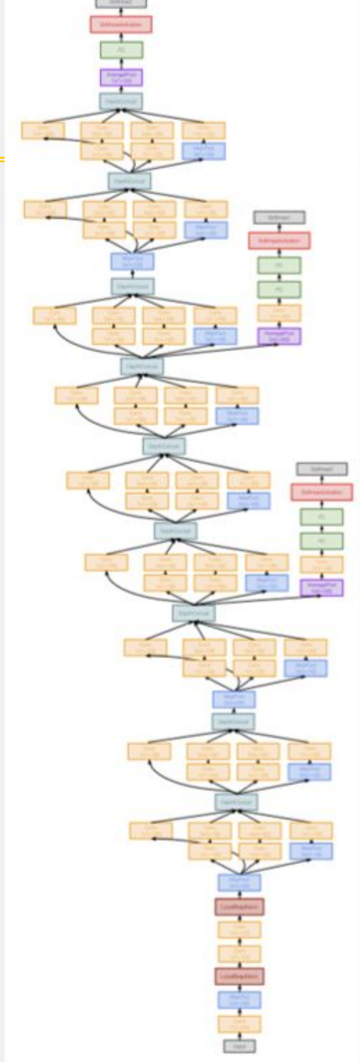
- reduce parameter count
- memory usage
- computation

Szegedy et al, “Going deeper with convolutions”,
CVPR 2015
https://arxiv.org/abs/1409.4842



GoogLeNet: Focus on Efficiency

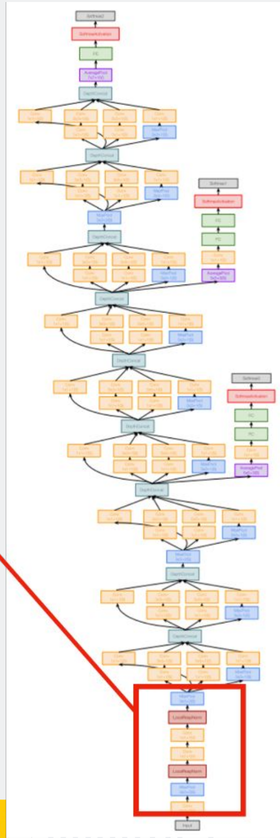
Stem network at the start aggressively downsamples input
(Recall in VGG-16: Most of the compute was at the start)



GoogLeNet: Focus on Efficiency

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	Strid	Pad	C	H/W	Memory	Params	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



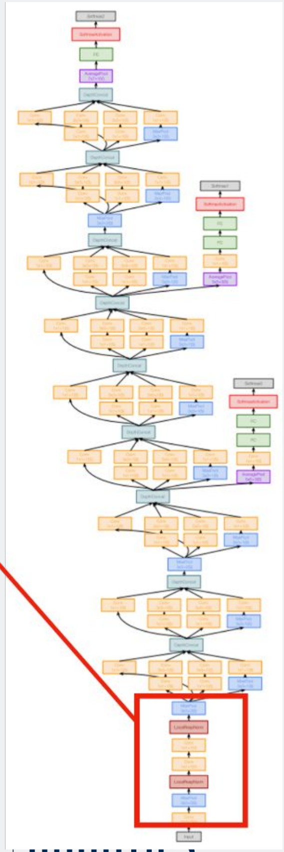
Total from 224 to 28 spatial resolution:
Memory: 7.5 MB
Params: 124K
MFLOP: 418

Quickly downsampling by a factor of 8

GoogLeNet: Focus on Efficiency

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	Strid	Pad	C	H/W	Memory	Params	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



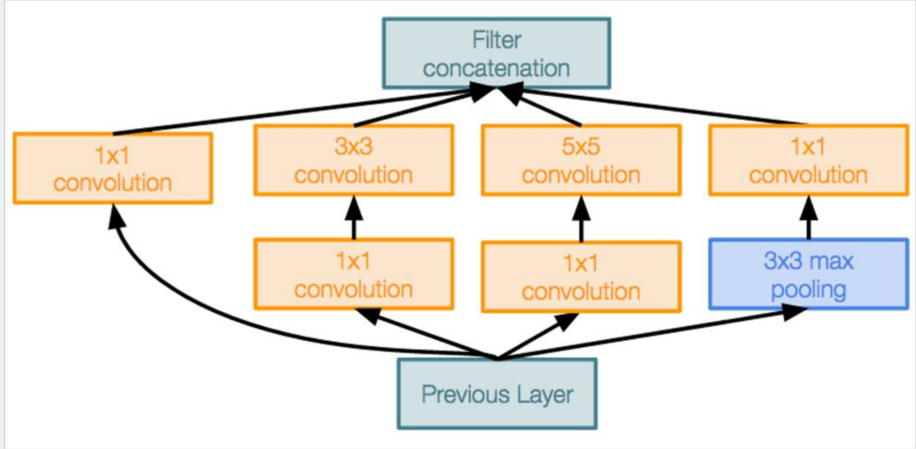
Total from 224 to 28 spatial resolution:
Memory: 7.5 MB
Params: 124K
MFLOP: 418

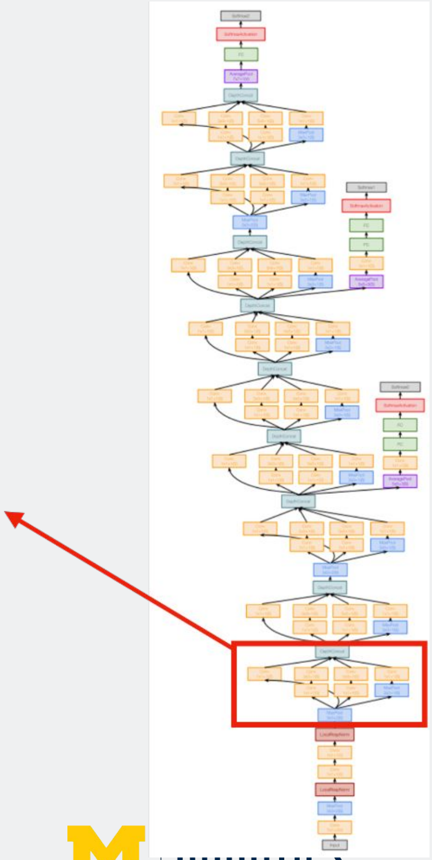
Compare VGG-16:
Memory: 42.9 MB (5.7x)
Params: 1.1M (8.9x)
MFLOP: 7485 (17.8x)

GoogLeNet: Focus on Efficiency

Inception module: Local unit with parallel branches

Local structure repeated many times throughout the network



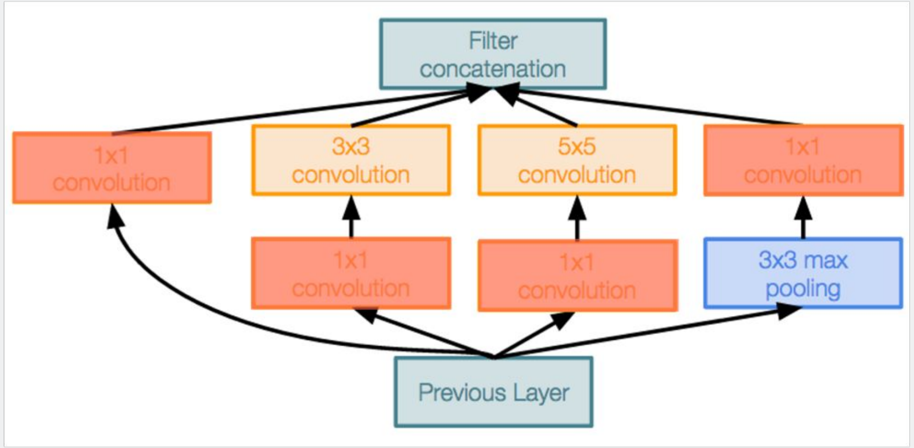


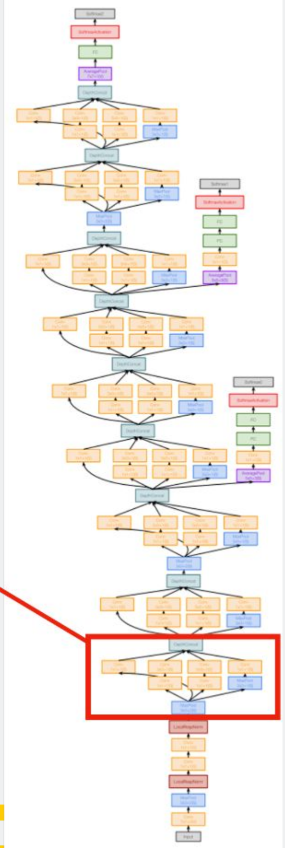
GoogLeNet: Focus on Efficiency

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!)



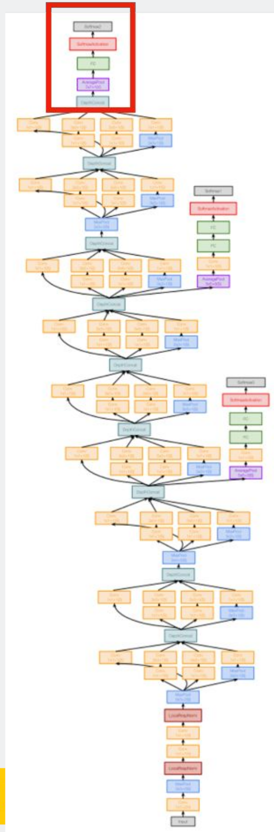


GoogLeNet: Global Average Pooling

No large FC layers at the end!

Instead use **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	Flop (M)
avg-pool	1024	7		7	1	0	1024	1	4	0	0
fc	1024		1000				1000	0	0	1025	1



GoogLeNet: Global Average Pooling

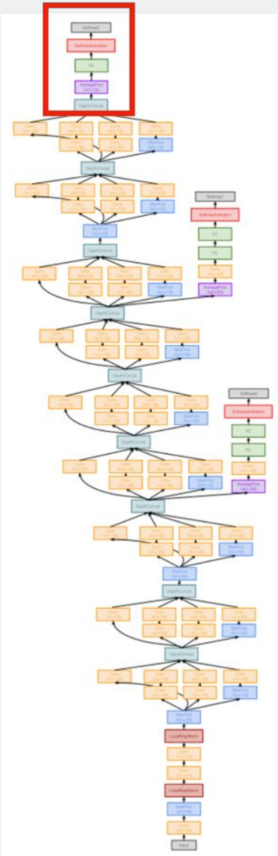
No large FC layers at the end!

Instead use **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	Flop (M)
avg-pool	1024	7		7	1	0	1024	1	4	0	0
fc	1024		1000				1000	0	0	1025	1

Compare with VGG-16:

	Input size		Layer				Output size				
Layer	C	H/W	Filters	Kernel	Stride	Pad	C	H/W	Memory (KB)	Params	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

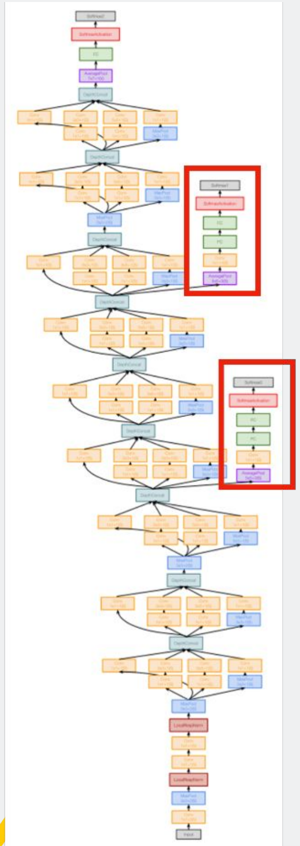


GoogLeNet: Auxiliary Classifiers

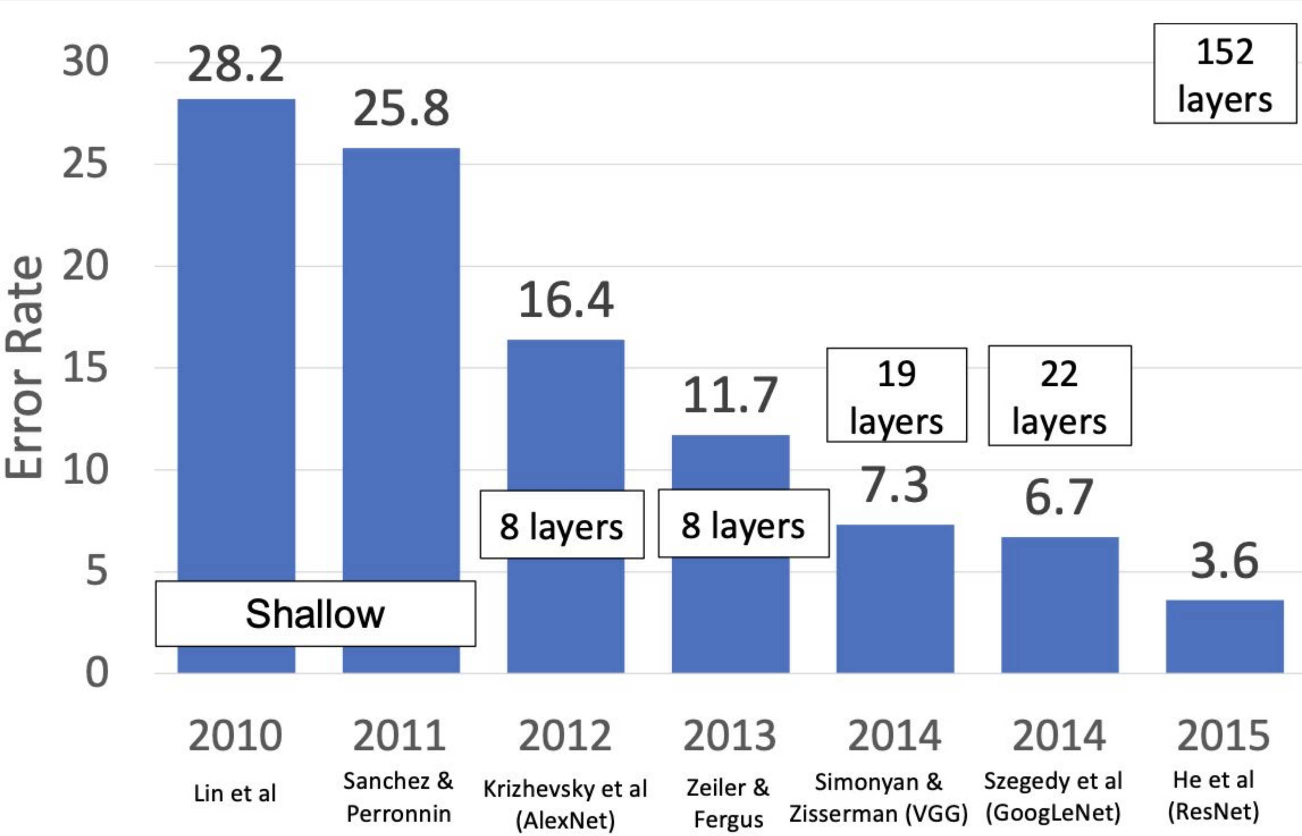
Training using loss at the end of the network didn't work well: Network is too deep, gradients don't propagate cleanly

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, we no longer need to use this trick



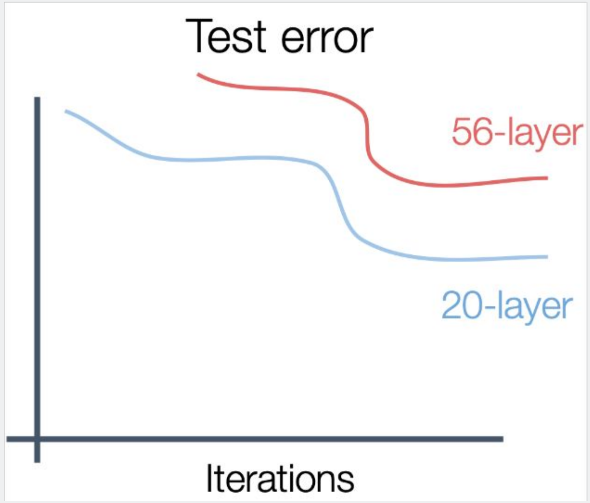
ImageNet Classification Challenge



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!



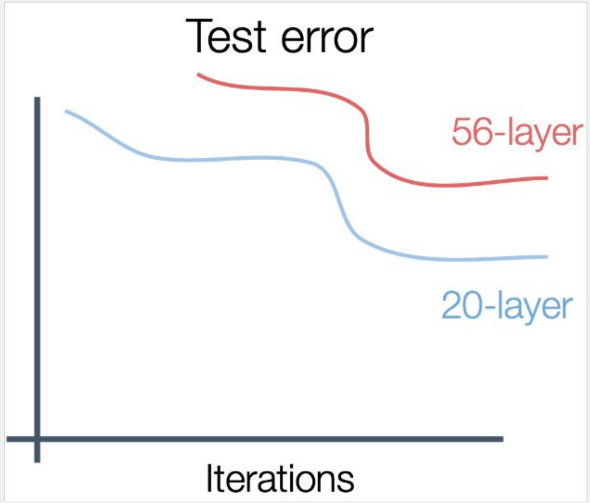
He et al, "Deep Residual Learning for Image Recognition", CVPR 2016
https://arxiv.org/abs/1512.03385

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!

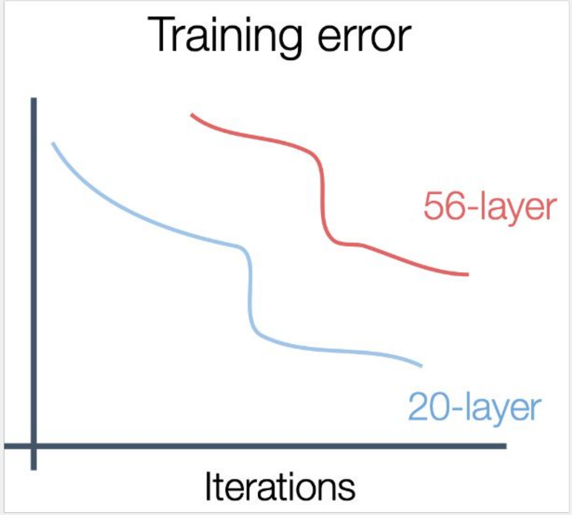
Initial guess: Deep model is **overfitting** since it is much bigger than the other model (?)

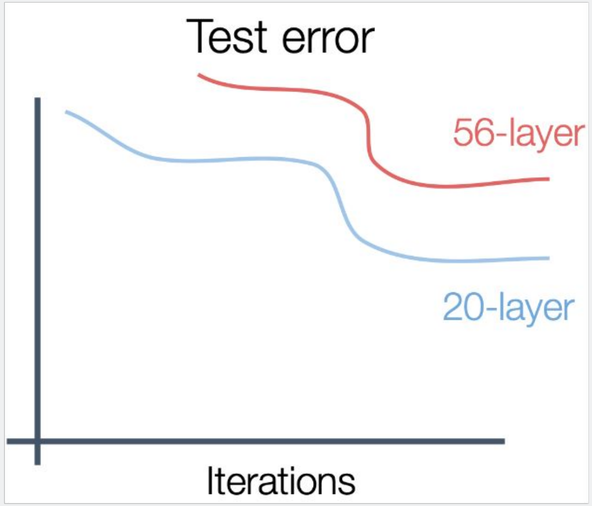


He et al, "Deep Residual Learning for Image Recognition", CVPR 2016
https://arxiv.org/abs/1512.03385

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**

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*

Residual Networks

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

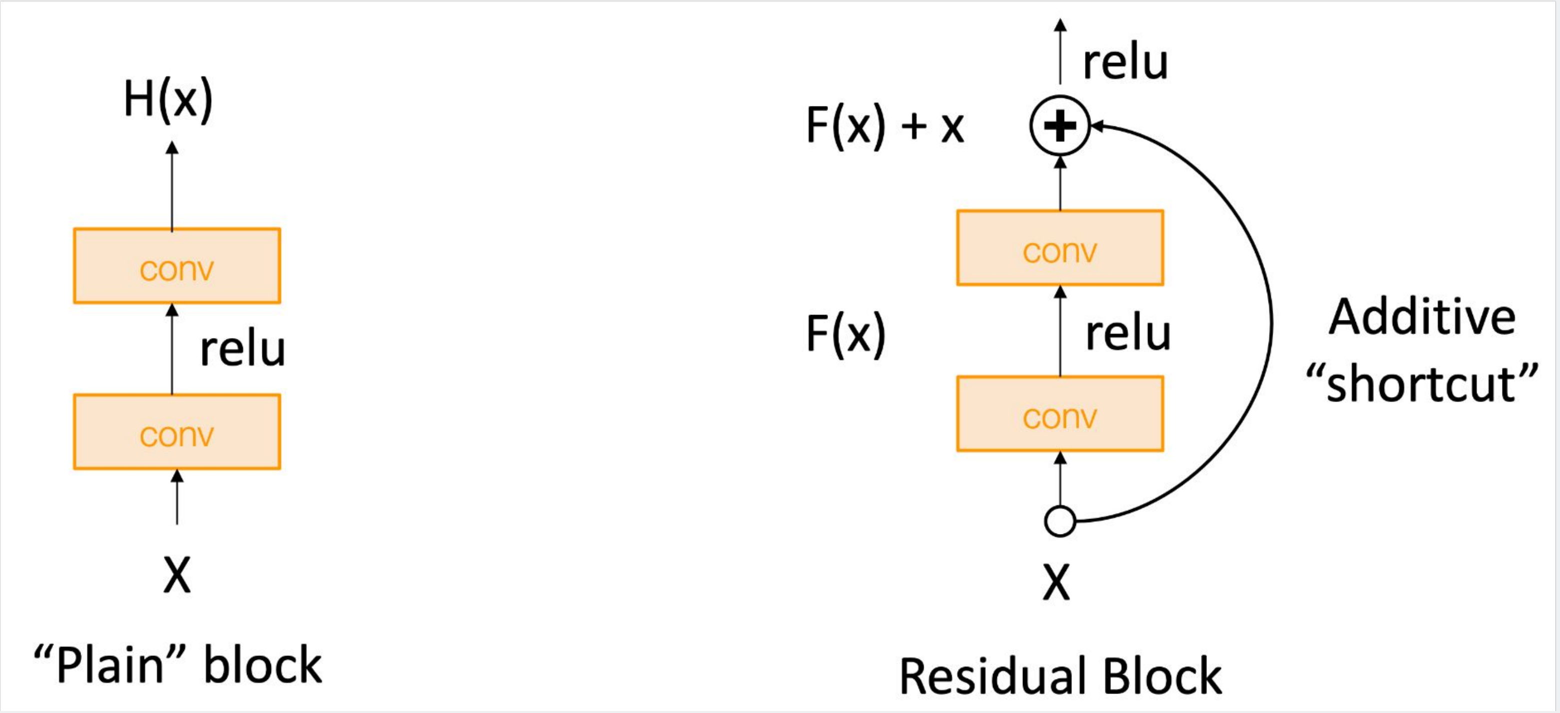
Thus deeper models *should do at least as good as shallow models*

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!

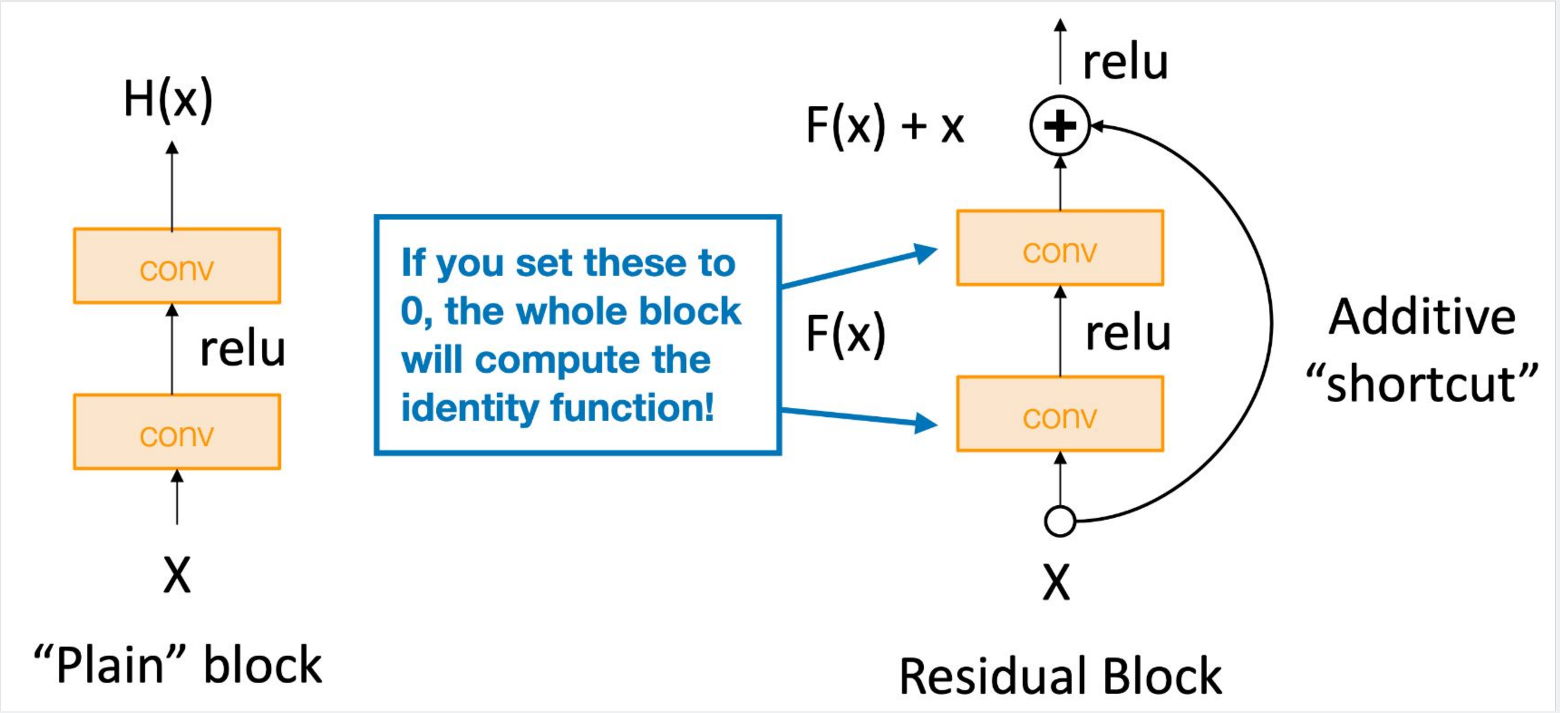
Residual Networks

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



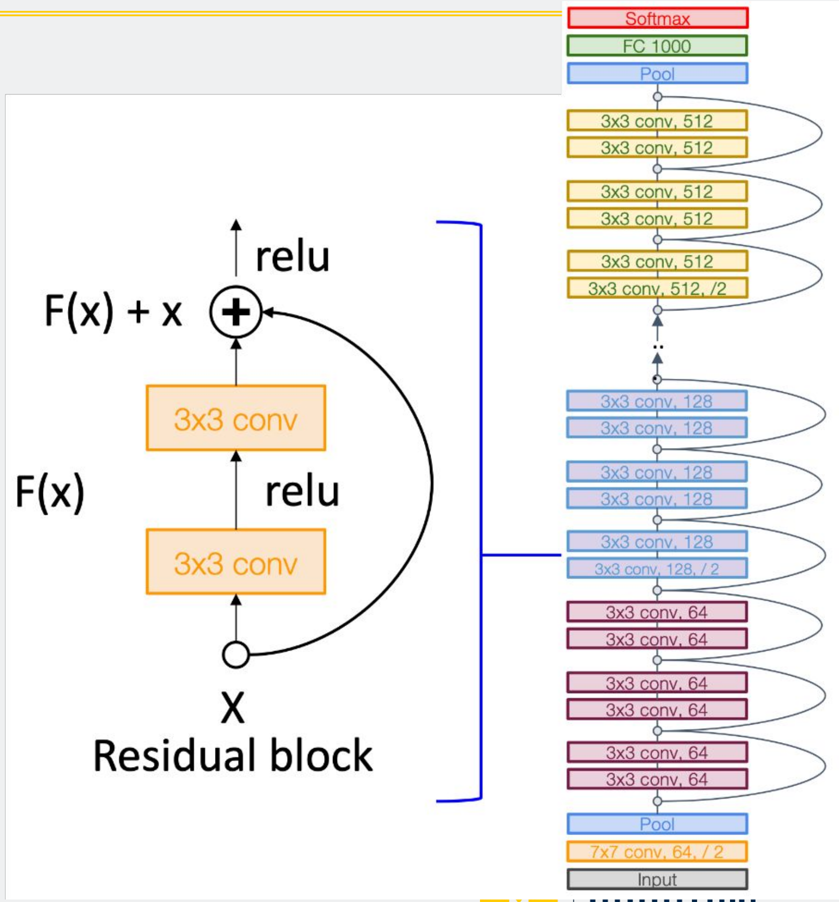
Residual Networks

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



Residual Networks

A residual network is a stack of many residual blocks

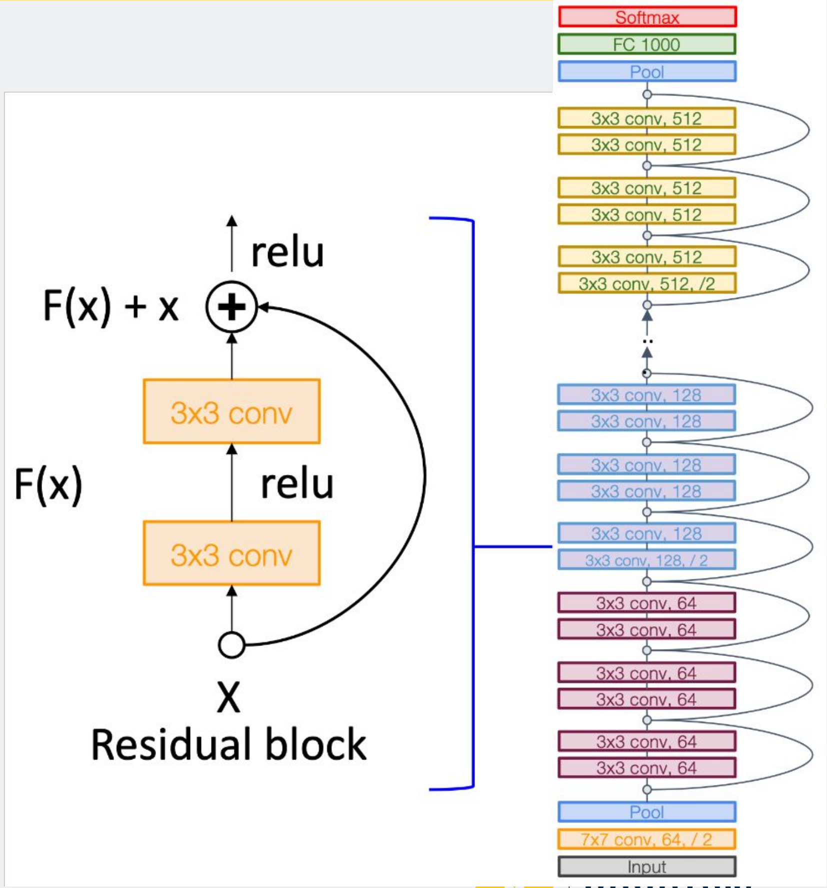


Residual Networks

A residual network is a stack of many residual blocks

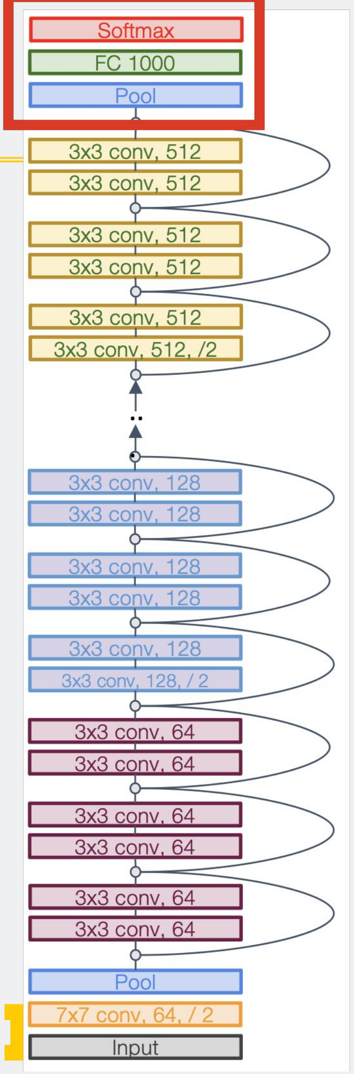
Regular design, like VGG: each residual block has two 3x3 conv

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



Residual Networks

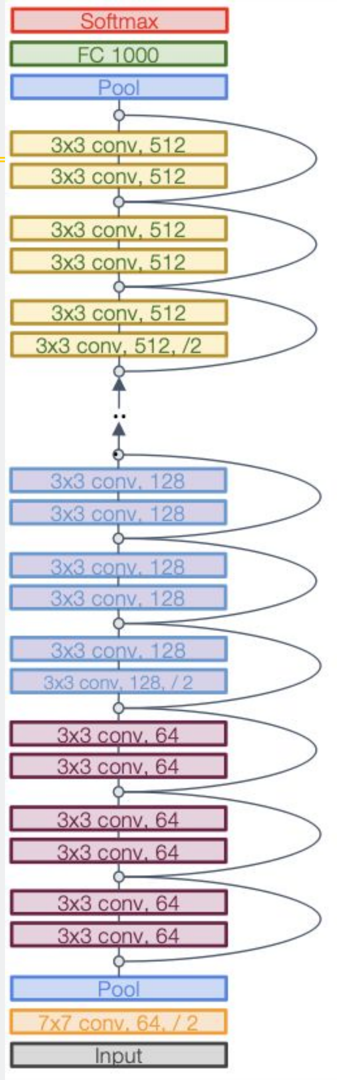
Like GoogLeNet, no big fully-connected-layers: Instead use global average pooling and a single linear layer at the end



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

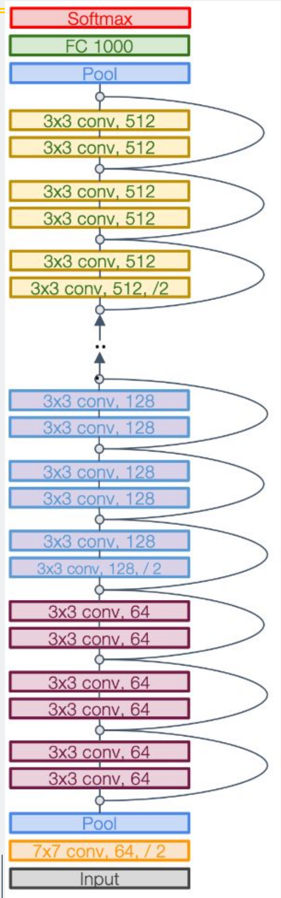
ResNet-18:
Stem: 1 conv layer
Stage 1 (C=64): 2 res. block = 4 conv
Stage 2 (C=128): 2 res. block = 4 conv
Stage 3 (C=256): 2 res. block = 4 conv
Stage 4 (C=512): 2 res. block = 4 conv
Linear

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

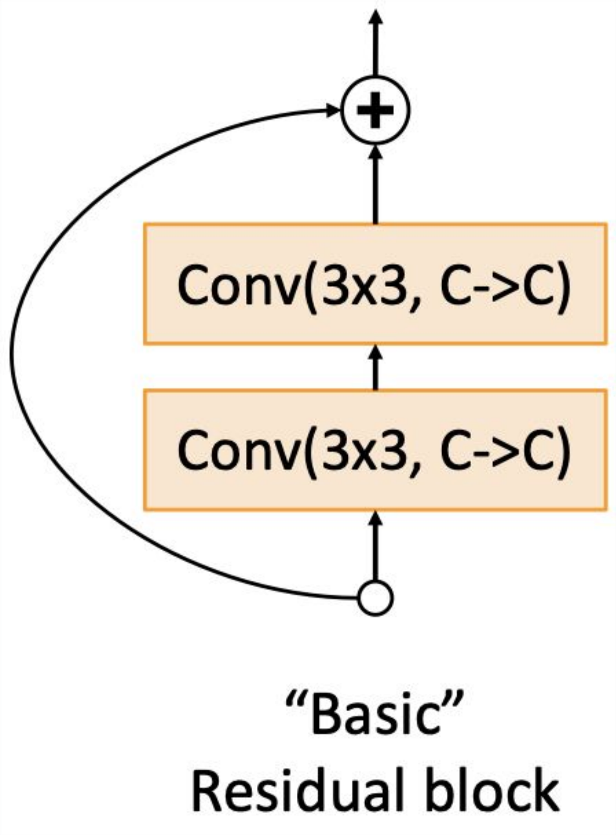
VGG-16:
ImageNet top-5 error: 9.62
GFLOP: 13.6



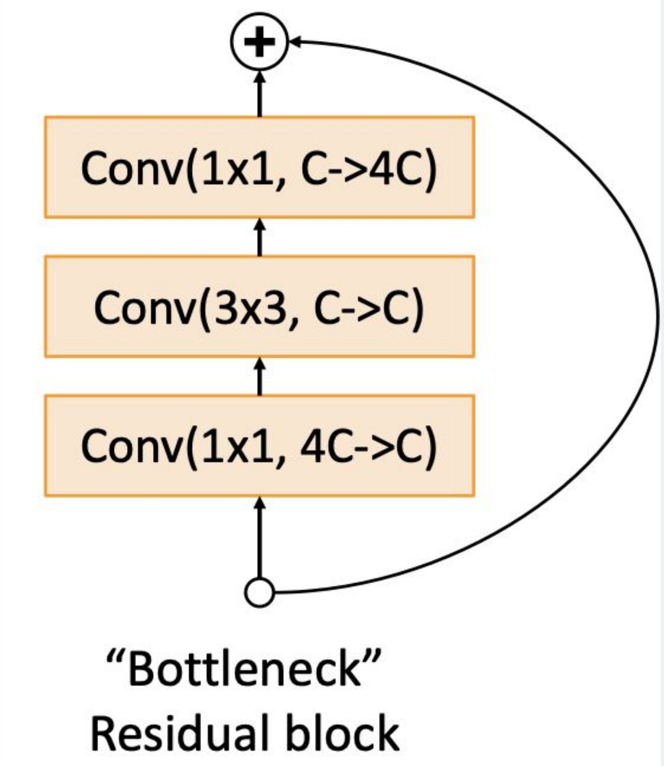
Aha Slides (In-class participation)

https://ahaslides.com/DFZE4

Residual Networks: Basic and Bottleneck Block



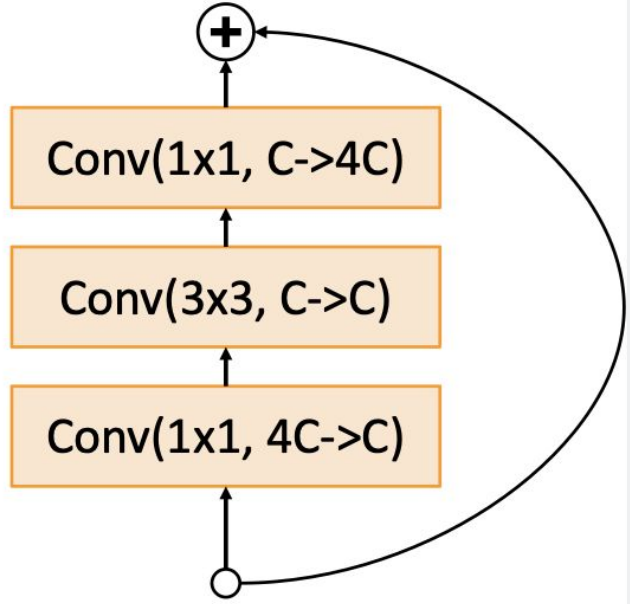
Hint:
How
many
FLOPs?



ROBOTICS

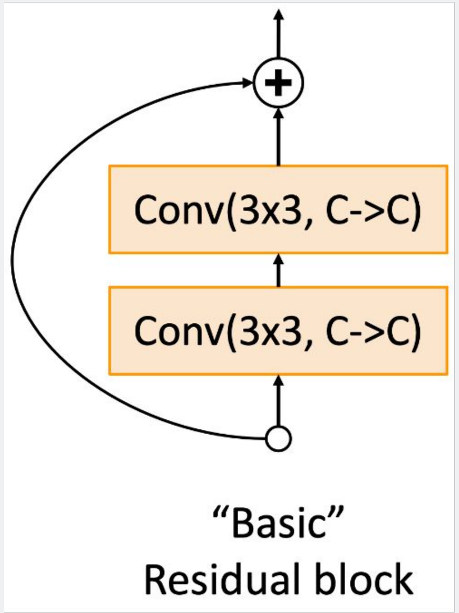
Residual Networks: Bottleneck Block

Q: How many FLOPs?



"Bottleneck"
Residual block

Residual Networks: Bottleneck Block



More layers, less computational cost!

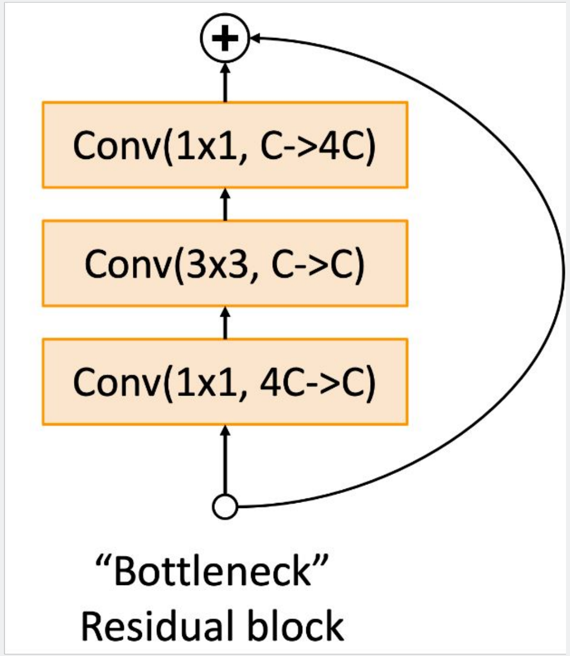
FLOPs: $9HWC^2$

FLOPs: $9HWC^2$

FLOPs: $4HWC^2$

FLOPs: $9HWC^2$

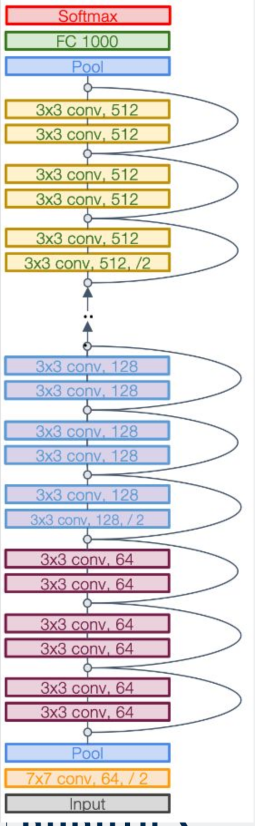
FLOPs: $4HWC^2$



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!

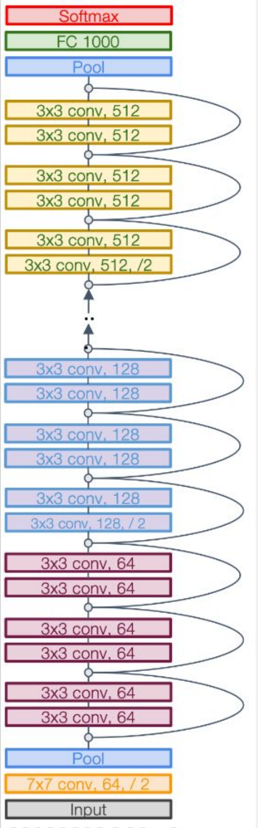
			Stage 1		Stage 2		Stage 3		Stage 4				
	Block type	Stem layers	Blocks	Layers	Blocks	Layers	Blocks	Layers	Blocks	Layers	FC Layers	GFLOP	ImageNet
ResNet-18	Basic	1	2	4	2	4	2	4	2	4	1	1.8	10.92
ResNet-34	Basic	1	3	6	4	8	6	12	3	6	1	3.6	8.58
ResNet-50	Bottle	1	3	9	4	12	6	18	3	9	1	3.8	7.13



Residual Networks

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

	Block type	Stem layers	Stage 1		Stage 2		Stage 3		Stage 4				
			Blocks	Layers	Blocks	Layers	Blocks	Layers	Blocks	Layers	FC Layers	GFLOP	ImageNet
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



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

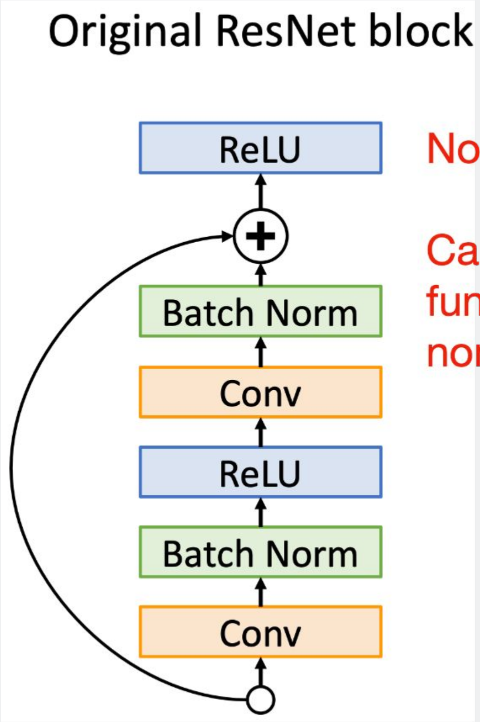
MSRA @ ILSVRC & COCO 2015 Competitions

- **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

Examples:

- ResNet for brain age prediction - Nature, Scientific Reports (2024) https://www.nature.com/articles/s41598-024-61915-5
- Bridging ResNet and Vision Transformers - CVPR (2024) https://arxiv.org/abs/2403.14302

Improving Residual Networks: Block Design

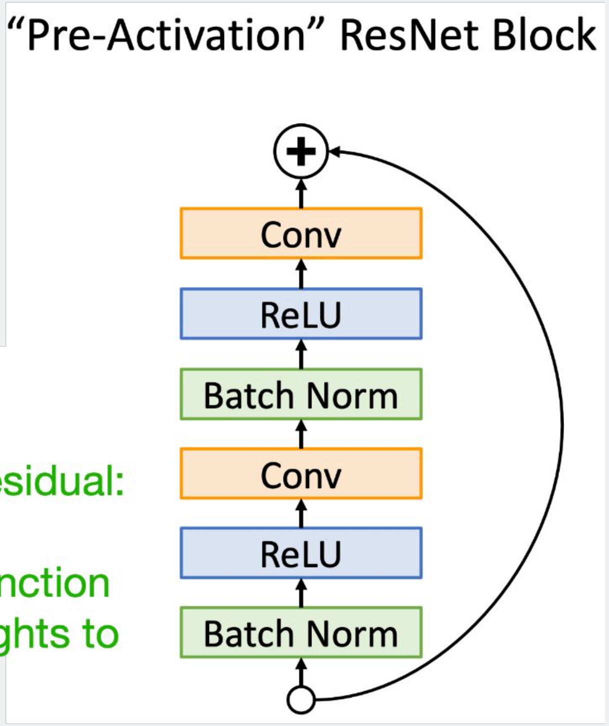


Note ReLU **after** residual:

Cannot actually learn identity function since outputs are nonnegative!

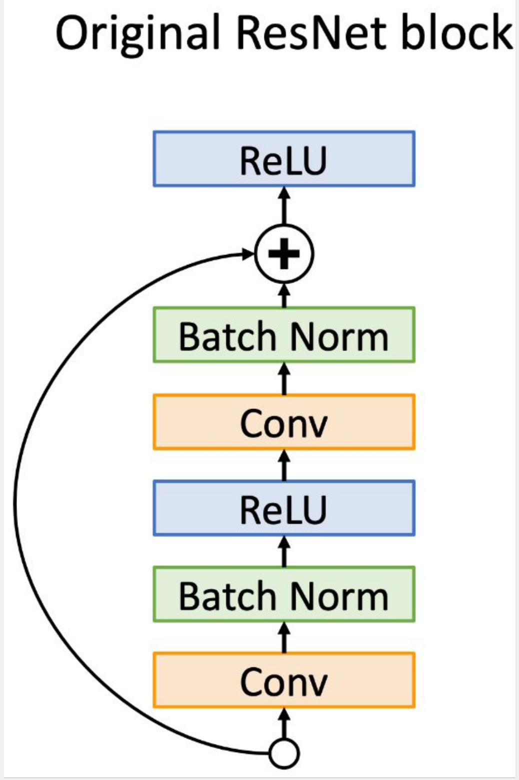
Note ReLU **inside** residual:

Can learn identity function by setting Conv weights to zero



He et al, "Identity mappings in deep residual networks", ECCV 2016
https://arxiv.org/abs/1603.05027

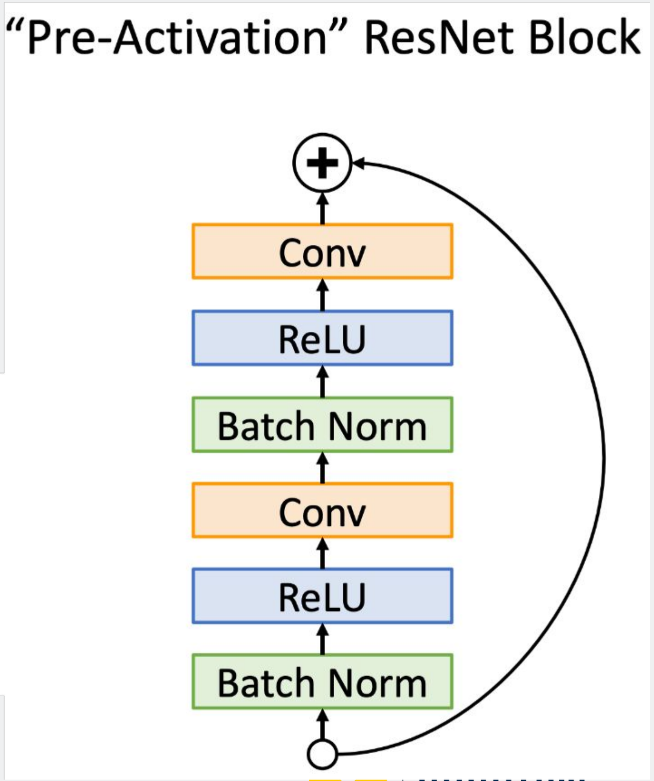
Improving Residual Networks: Block Design



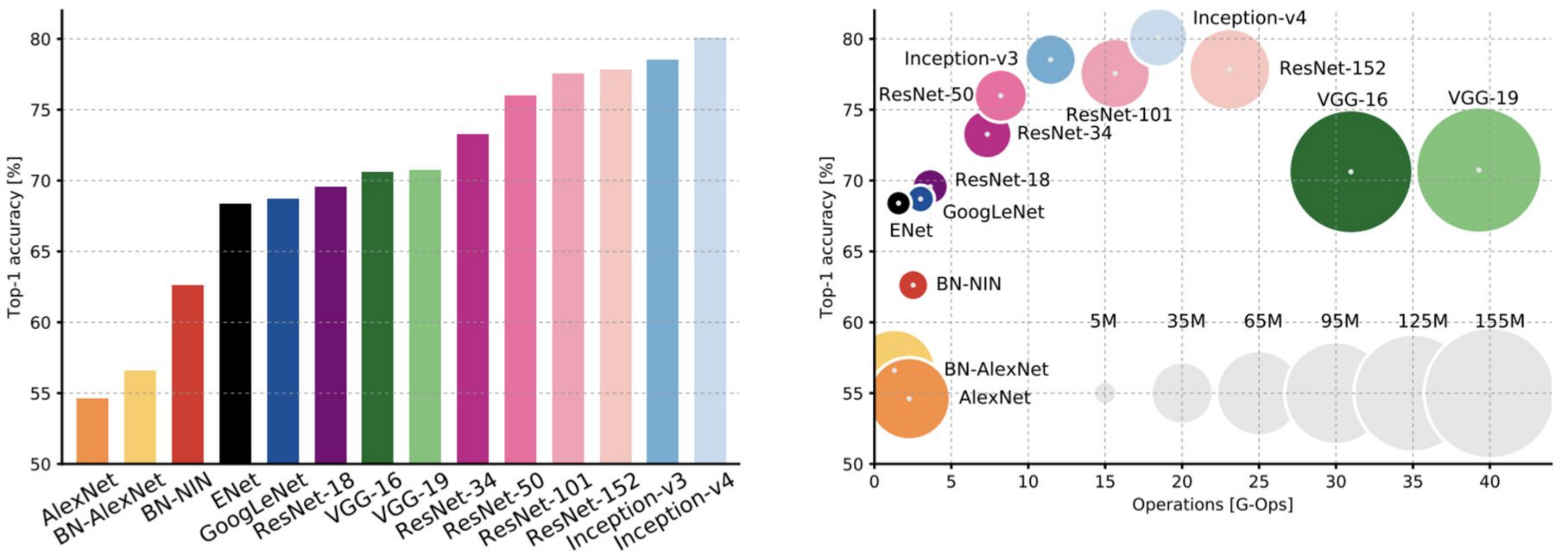
Slight improvement in accuracy (ImageNet top-1 error)

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

Not actually used that much in practice



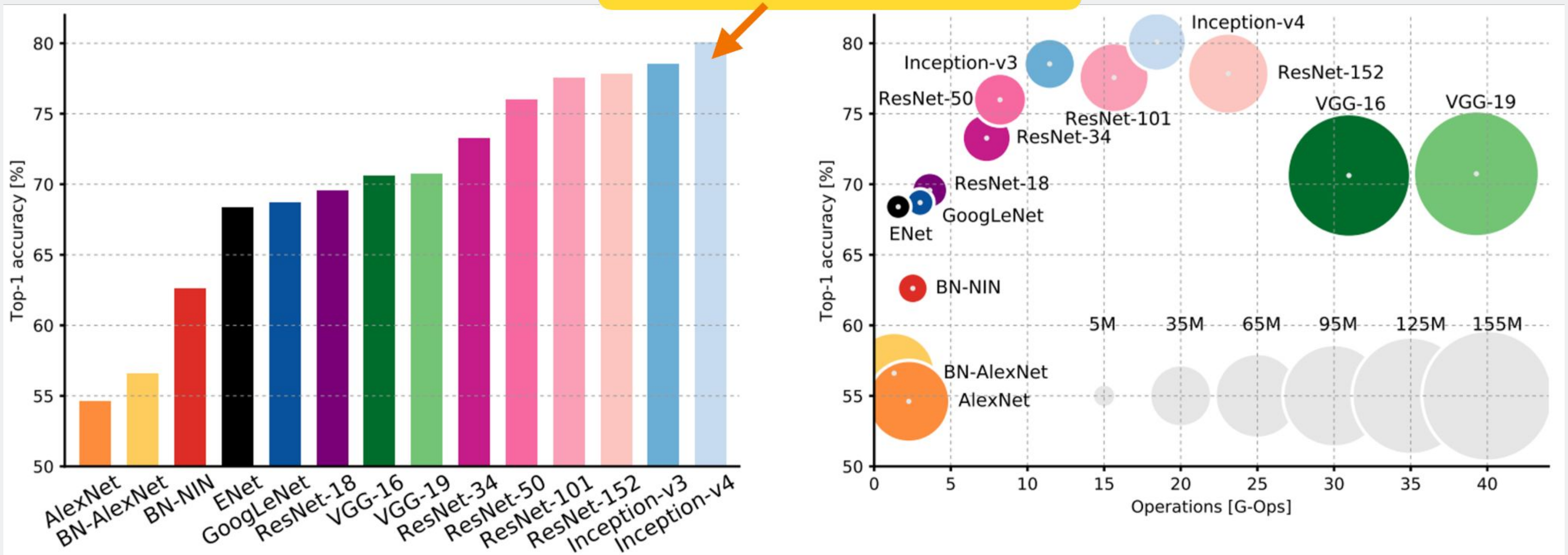
Comparing Complexity



ROBOTICS

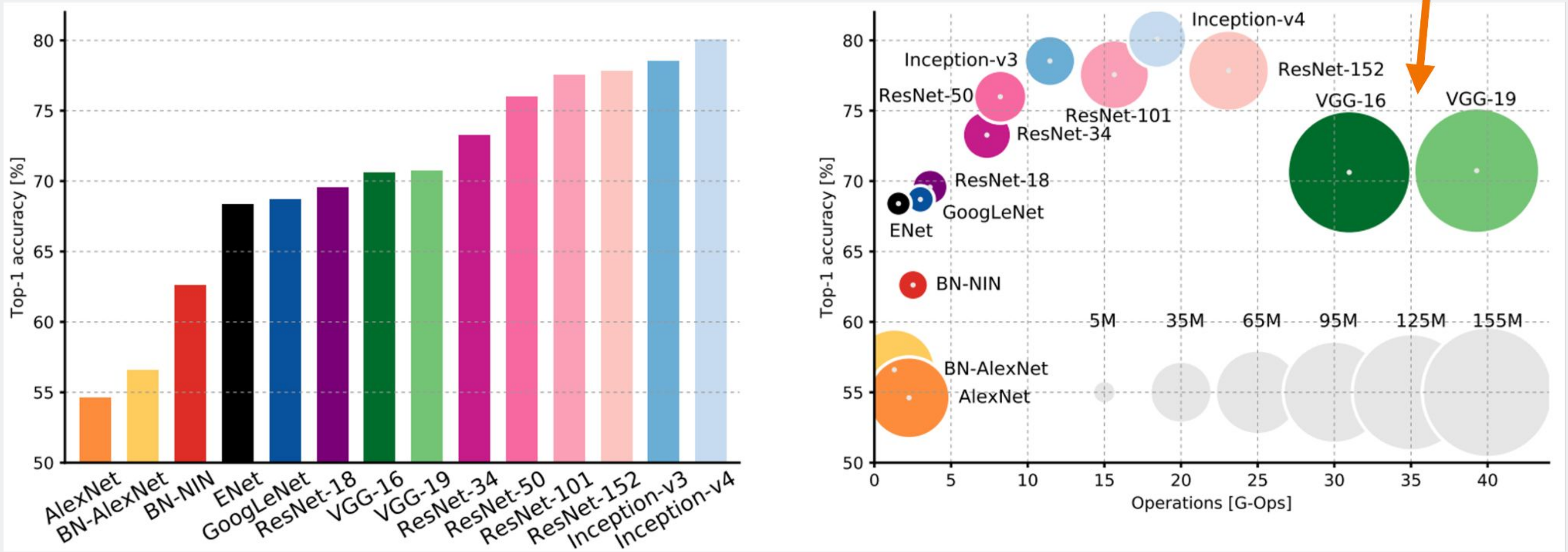
Comparing Complexity

Inception-v4: ResNet + Inception!



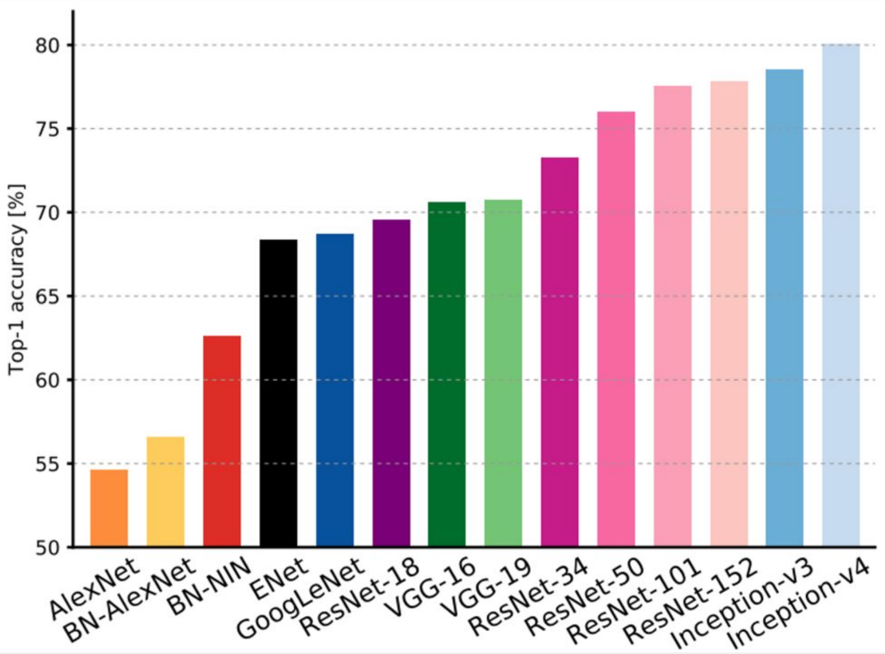
Comparing Complexity

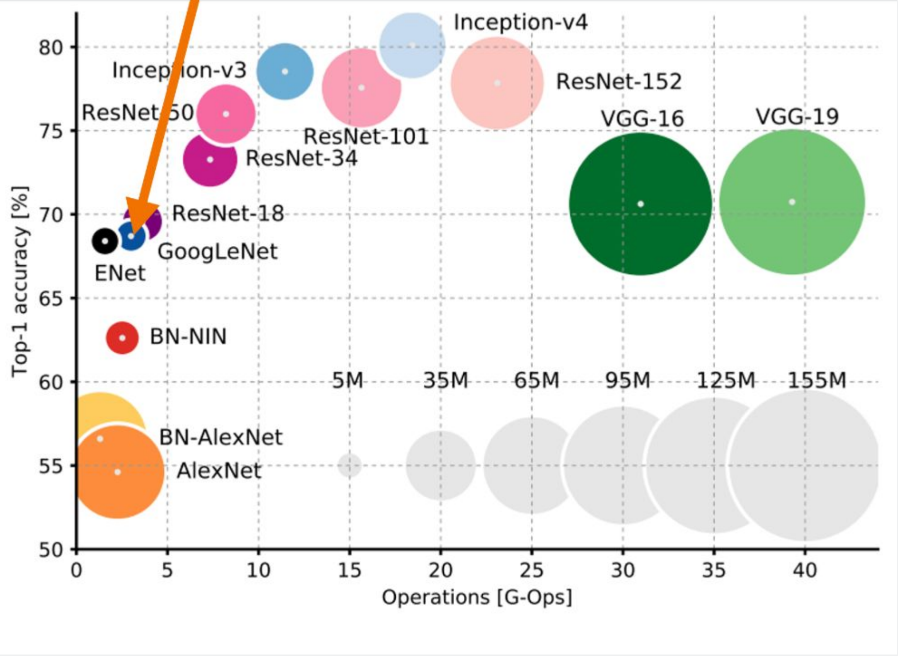
VGG:
Highest memory,
most operations



Comparing Complexity

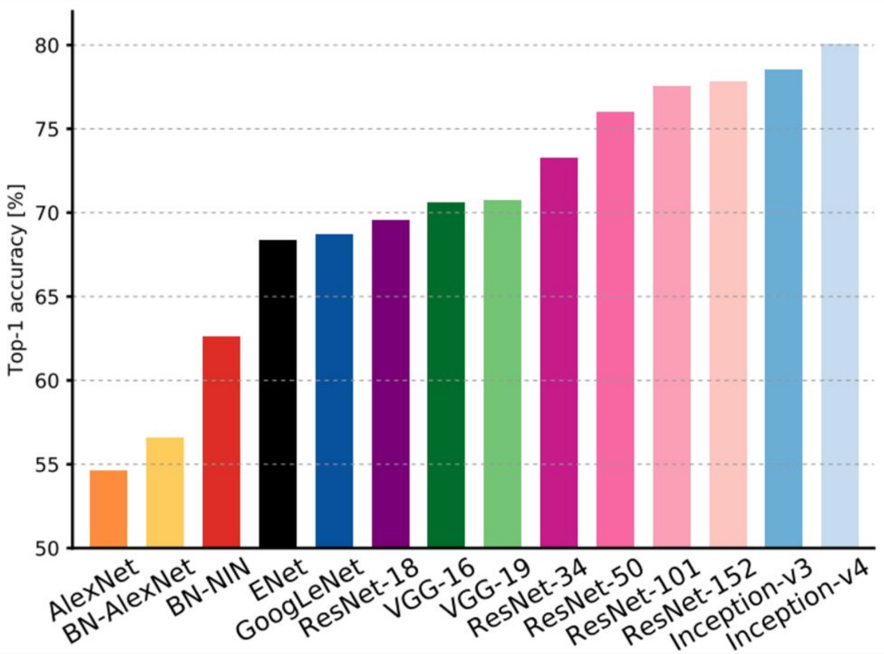
GoogLeNet:
Very efficient!

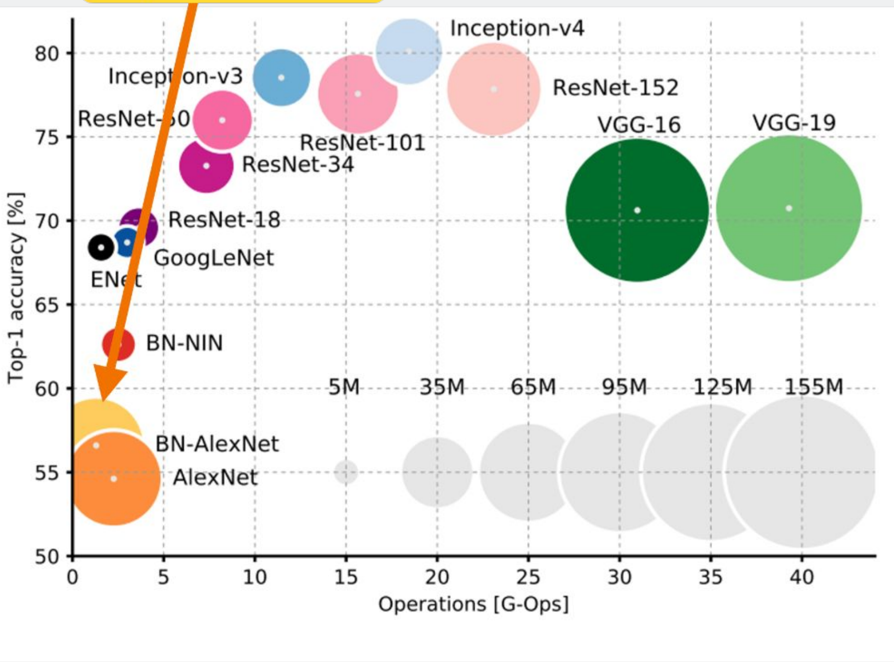




Comparing Complexity

AlexNet: Low compute, lots of parameters

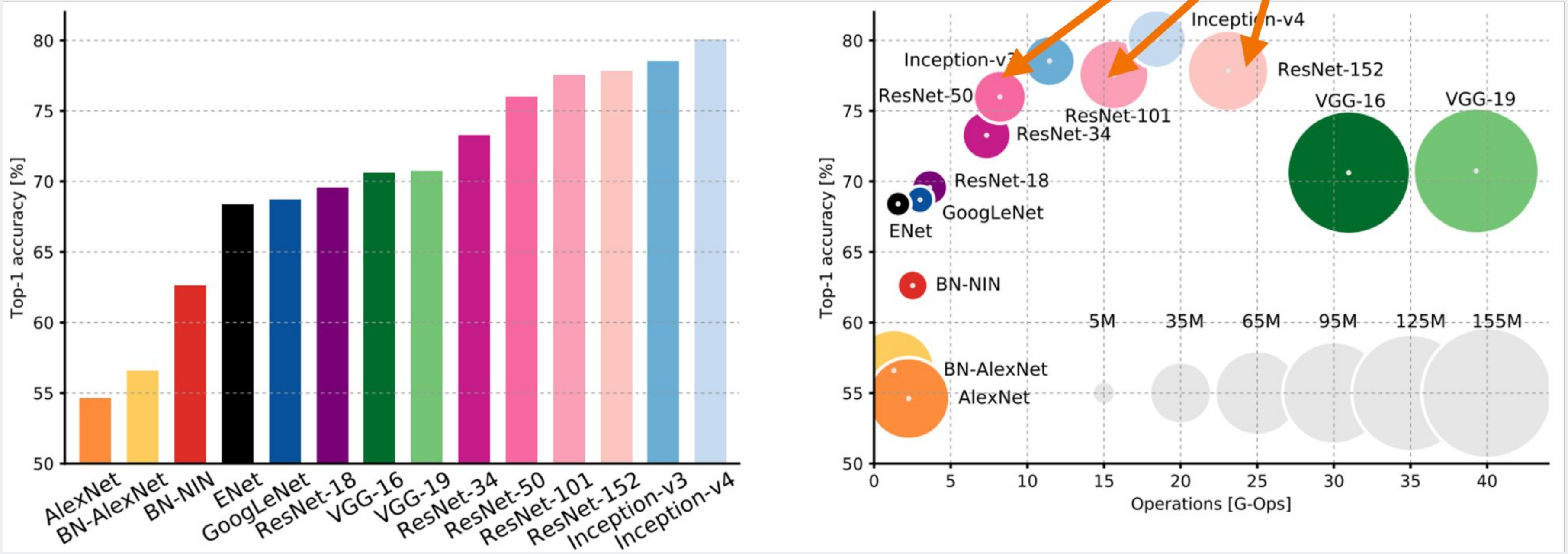




ROBOTICS

Comparing Complexity

ResNet: Simple design, moderate efficiency, high accuracy



What happens to ImageNet NOW?

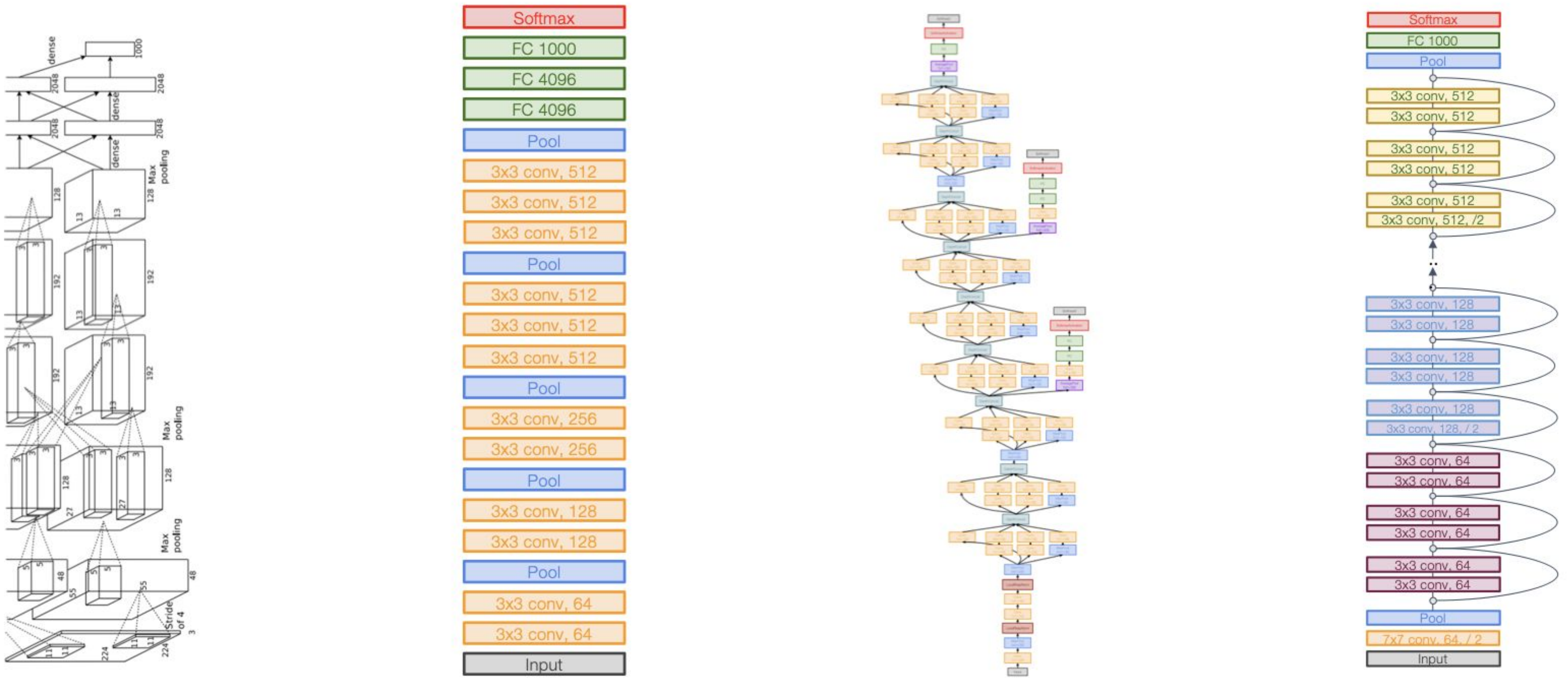
Original ImageNet challenge - (discontinued 2017)
Still a large-scale and valid benchmark dataset!
Also commonly used for weight initializations
https://www.image-net.org/

ImageNet 3D - (NeurIPS 2024)
General-Purpose Object-Level 3D Understanding
https://arxiv.org/abs/2406.09613
https://github.com/wufeim/imagenet3d?tab=readme-ov-file

Also, research in tiny networks

Summary

Canvas Quiz 20250205 released, due Feb. 9, 2025



AlexNet VGG GoogLeNet ResNet