CNN Architectures

Fiseha B. Tesema, PhD

Recap

- Introduction to deep learning
- Our Neural Network
- Modeling Neural networks
- Fully-Connected Network
- Convolutional Neural Networks (CNNs)
  - Activation functions
  - Fully-Connected layer to at a final stage: making a decision
  - Convolution layers
  - Pooling layers
  - Normalization

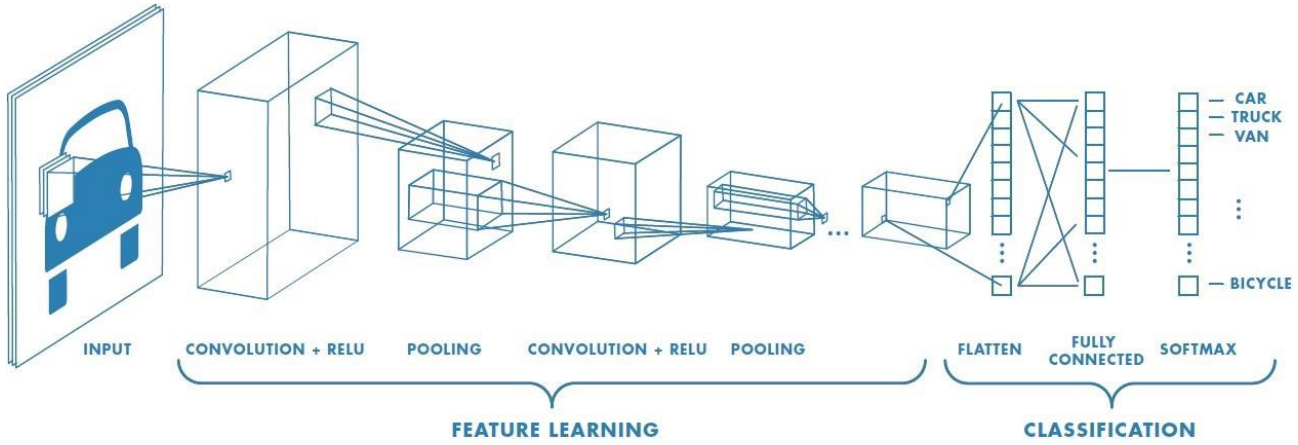
Outline

- Training CNN
- CNN Architecture
  - AlexNet
  - ZFNet: A Bigger AlexNet
  - VGG
  - GoogLeNet
  - ResNet
- Which Architecture should I use?
- How to use it?

Slide Credit

- Majority of the slide is complied from Deep Learning for computer vision: University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Training a CNN

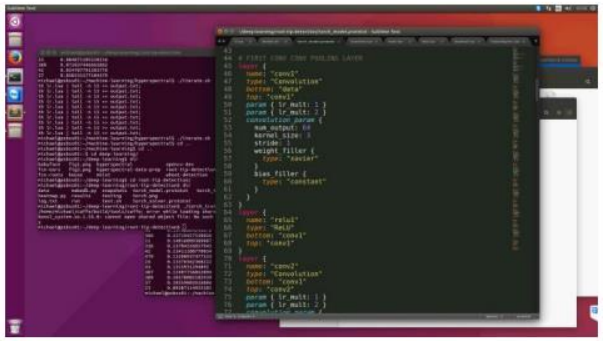


- Back-propagation algorithm
  - Stochastic gradient ascent - minimise error between net output and a training set
  - 5 convolutional layers, 3 layers in top neural network -> 500,000 neurons, 50,000,000 parameters,1 week to train (GPUs)

How this works in practice

1. Capture and annotate dataset

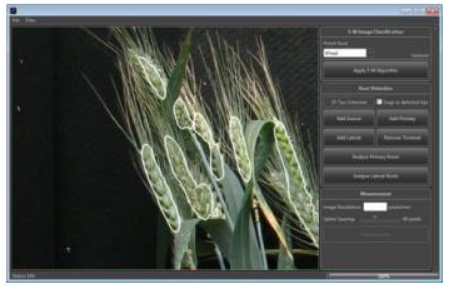
2. Design network architecture



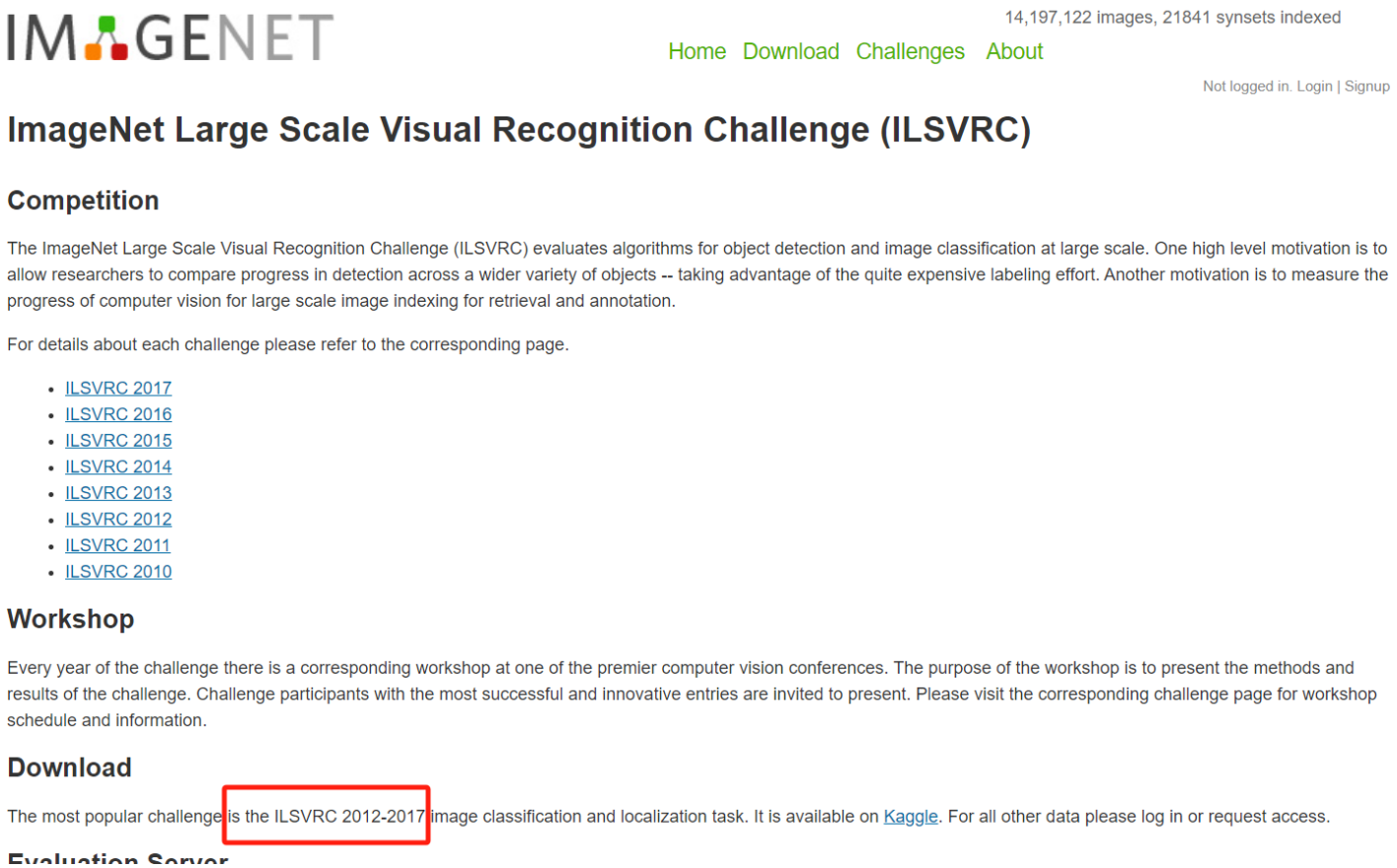
3. Train Network

4. Test Network

5. Deploy



Introduce ImageNEt



IMAGENET

14,197,122 images, 21841 synsets indexed

Home Download Challenges About

Not logged in. Login | Signup

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

Competition

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) evaluates algorithms for object detection and image classification at large scale. One high level motivation is to allow researchers to compare progress in detection across a wider variety of objects -- taking advantage of the quite expensive labeling effort. Another motivation is to measure the progress of computer vision for large scale image indexing for retrieval and annotation.

For details about each challenge please refer to the corresponding page.

- ILSVRC 2017
- ILSVRC 2016
- ILSVRC 2015
- ILSVRC 2014
- ILSVRC 2013
- ILSVRC 2012
- ILSVRC 2011
- ILSVRC 2010

Workshop

Every year of the challenge there is a corresponding workshop at one of the premier computer vision conferences. The purpose of the workshop is to present the methods and results of the challenge. Challenge participants with the most successful and innovative entries are invited to present. Please visit the corresponding challenge page for workshop schedule and information.

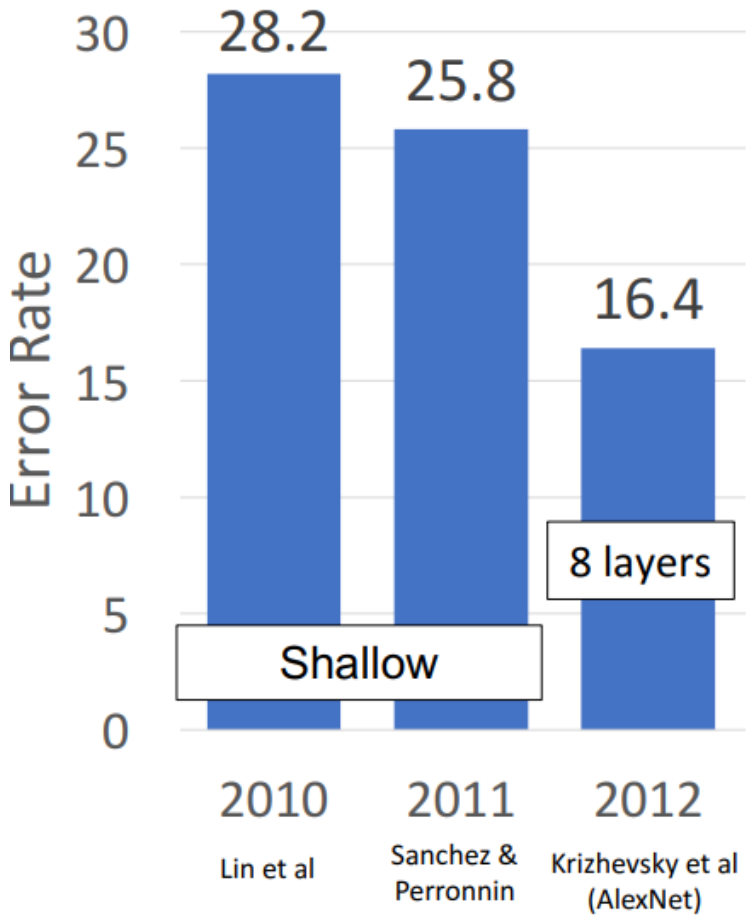
Download

The most popular challenge is the ILSVRC 2012-2017 image classification and localization task. It is available on Kaggle. For all other data please log in or request access.

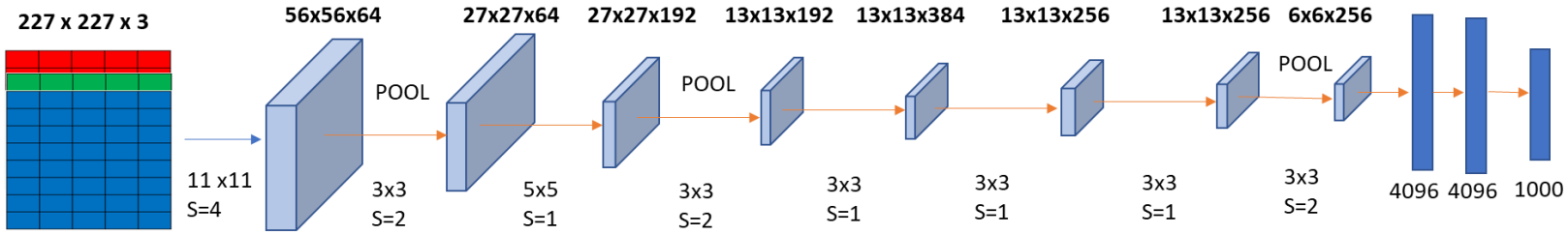
Evaluation Server

https://www.image-net.org/challenges/LSVRC/index.php

ImageNet Classification Challenge

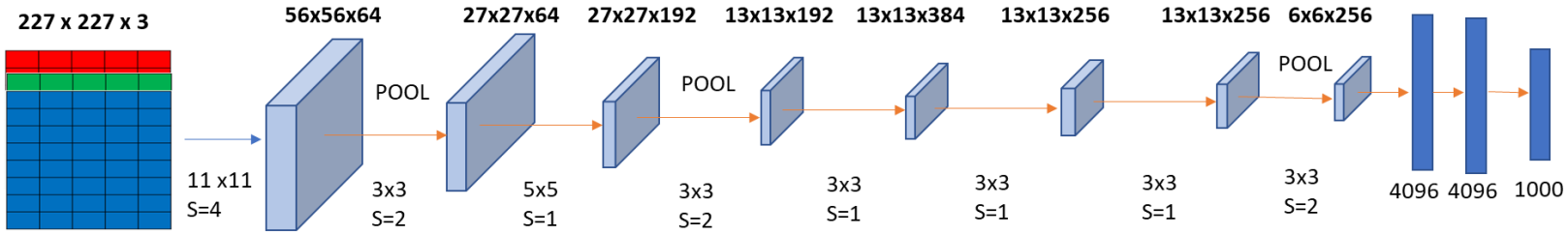


AlexNet



- 227 x 227 inputs
- 5 Convolutional layers
- Max pooling
- 3 fully-connected layers
- The first CNN that **ReLU nonlinearities**

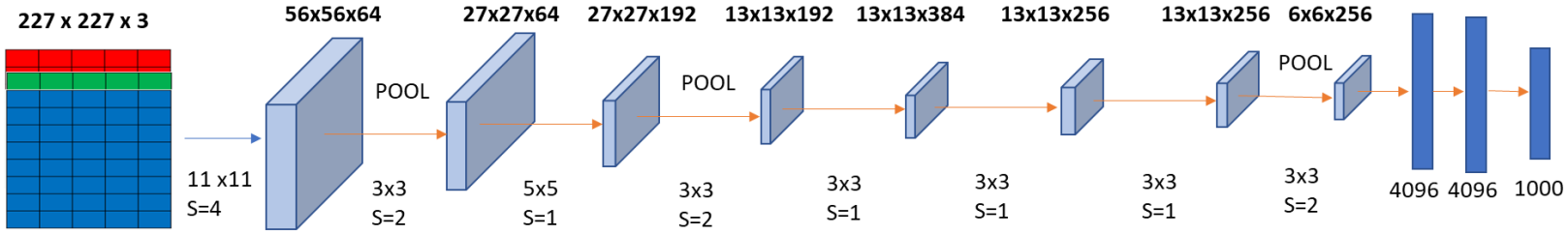
AlexNet



- 227 x 227x3 inputs
- 5 Convolutional layers
- 3 Max pooling
- 3 fully-connected layers
- The first CNN that **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

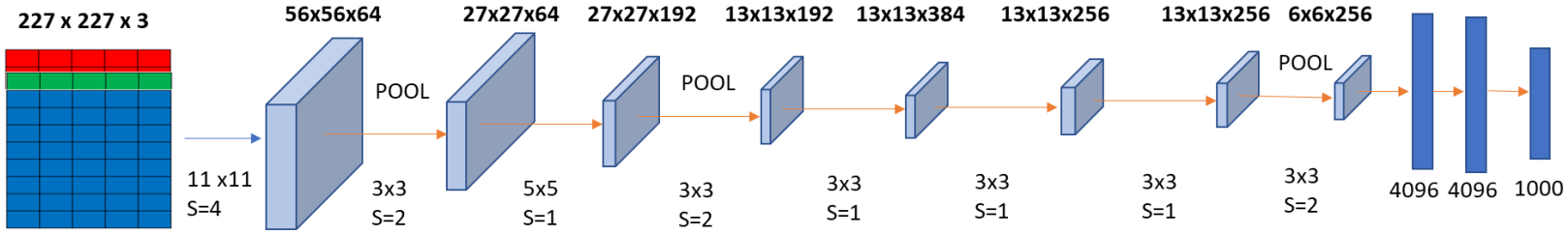
AlexNet



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

Recall: Output channels = number of filters

AlexNet



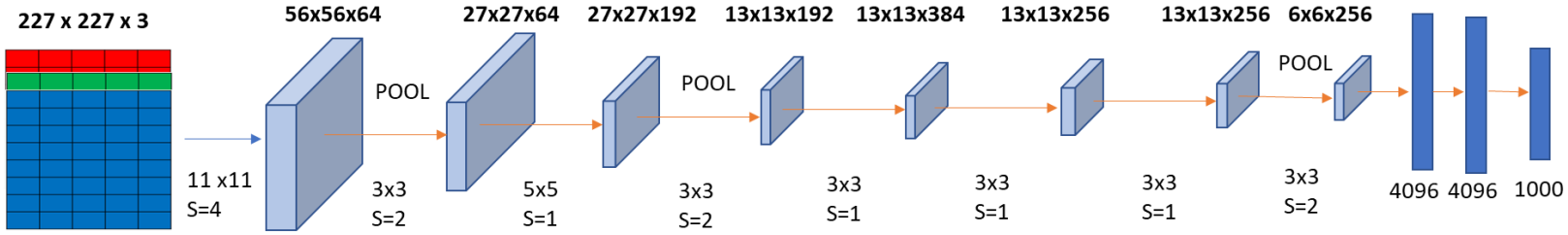
	Input size			Layer				Output size	
Layer	C	H / W		filters	kernel	stride	pad	C	H / W
conv1	3	227		64	11	4	2	64	56

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

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

$= 220/4 + 1 = 56$

AlexNet



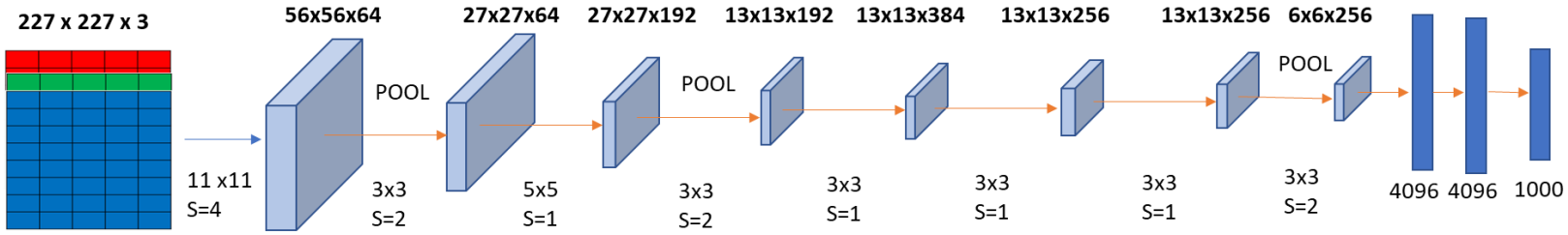
	Input size			Layer			Output size		
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)
conv1	3	227	64	11	4	2	64	56	784

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**

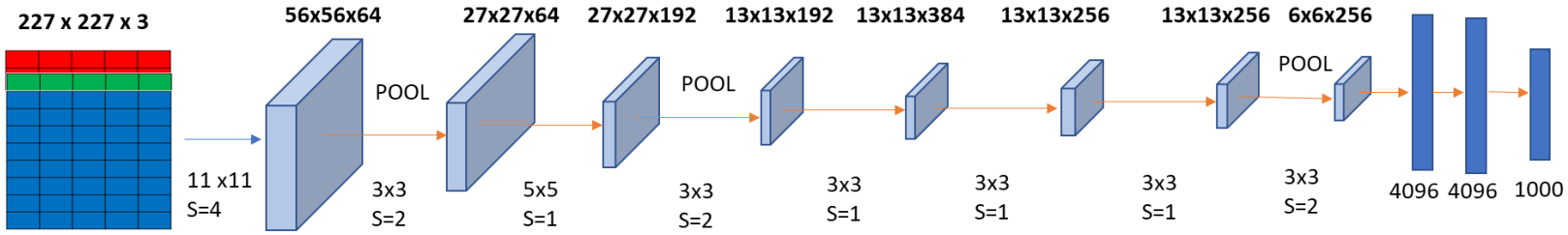
AlexNet



	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	?

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

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

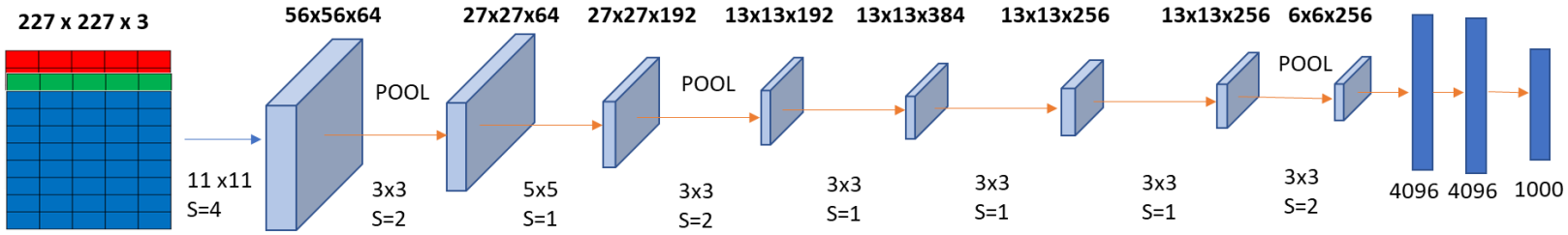
Bias shape = C_{out} = 64

Number of weights = 64*3*11*11 + 64

= **23,296**

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

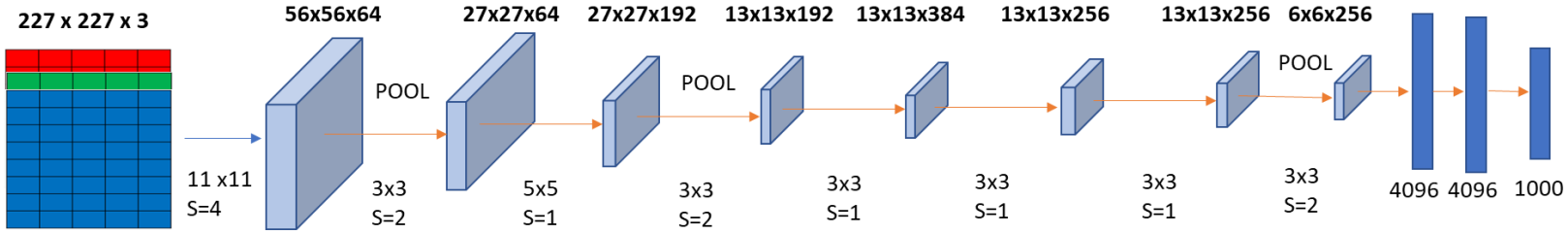
AlexNet



	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	?

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

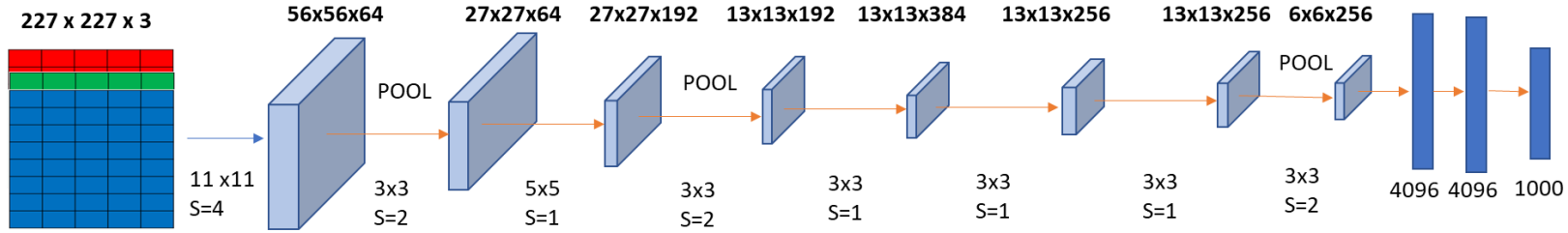
AlexNet



	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} x H' x W') * (C_{in} x K x K)
= (64 * 56 * 56) * (3 * 11 * 11)
= 200,704 * 363
= **72,855,552**

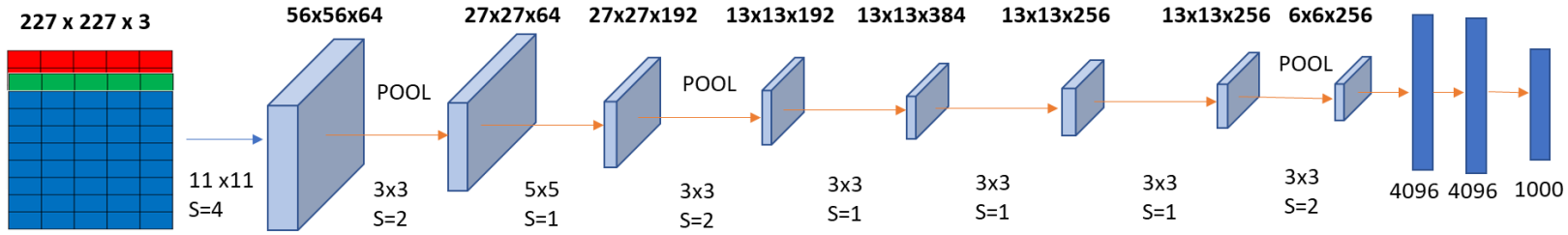
AlexNet



	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		?			

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

AlexNet



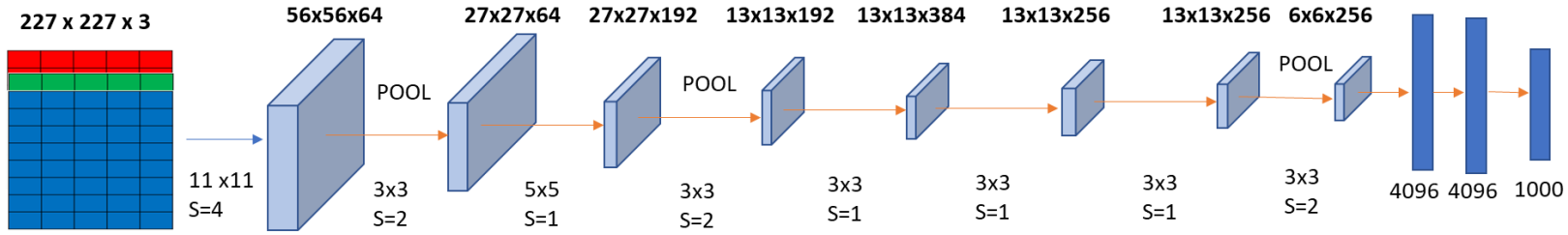
	Input size			Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27			

For pooling layer:

#output channels = #input channels = 64

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

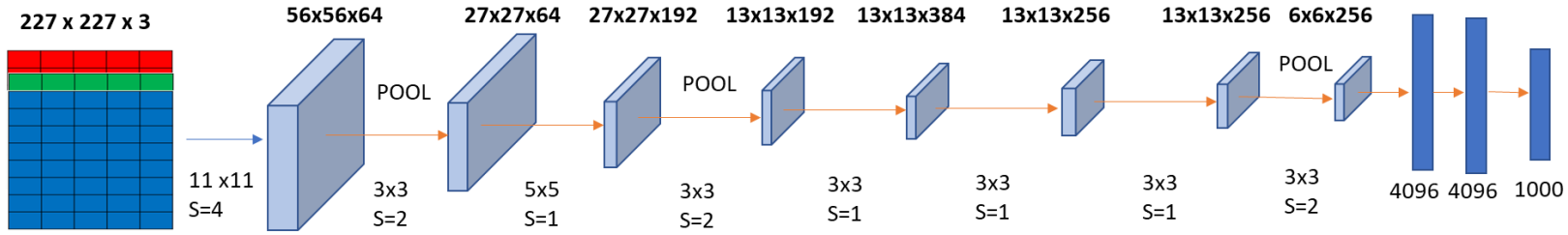
Bytes per elem = 4

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

= 64 * 27 * 27 * 4 / 1024

= **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	?	

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

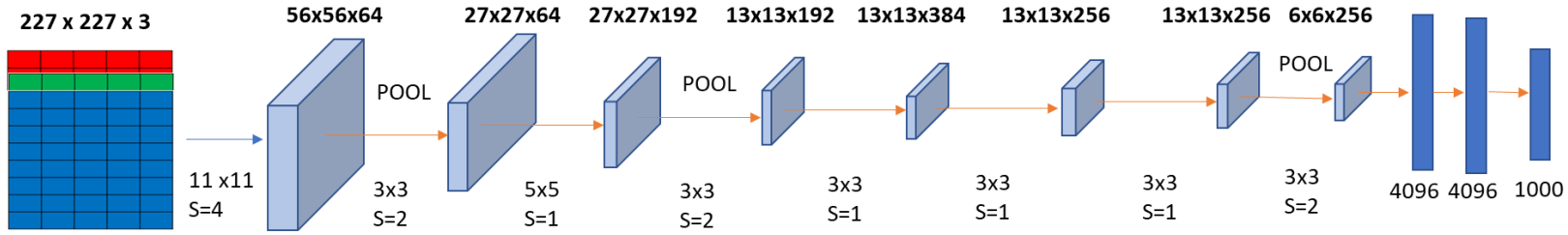
Bytes per elem = 4

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

= 64 * 27 * 27 * 4 / 1024

= **182.25**

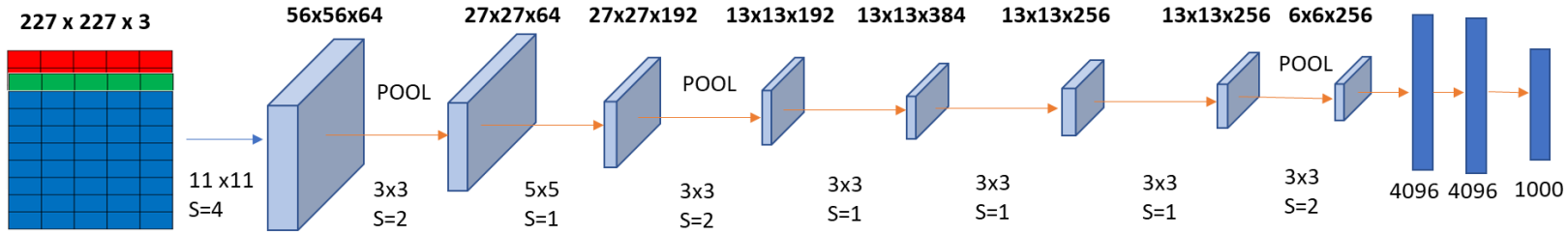
AlexNet



	Input size			Layer			Output size				
Layer	C	H / W	filters	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!

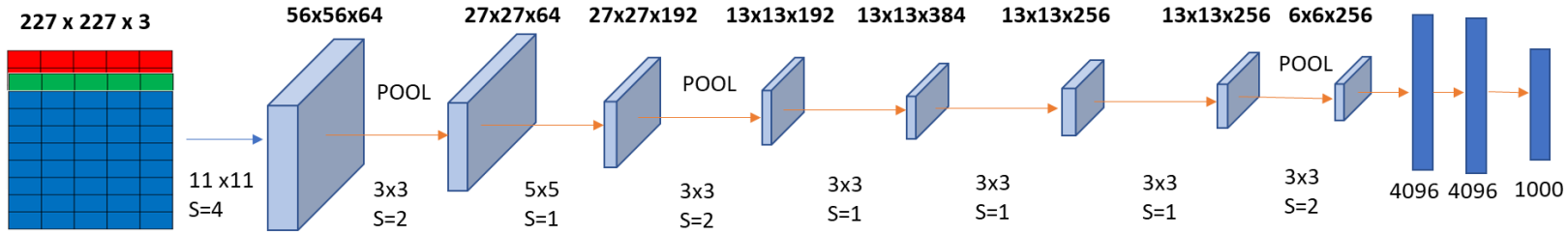
AlexNet



	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



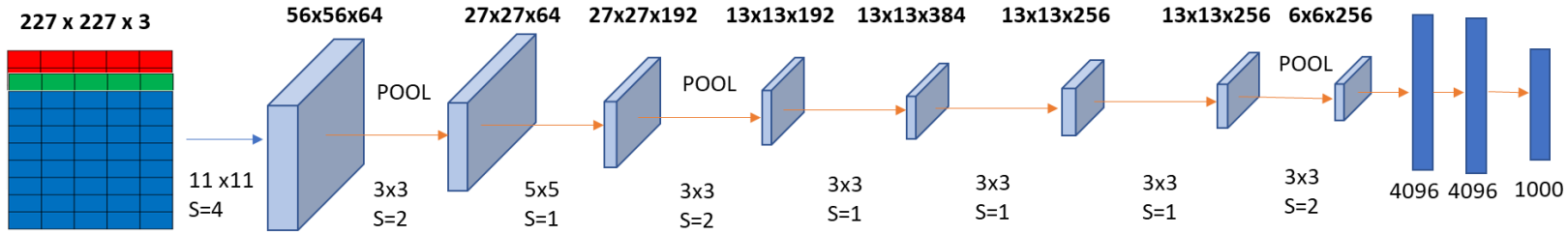
	Input size			Layer			Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664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

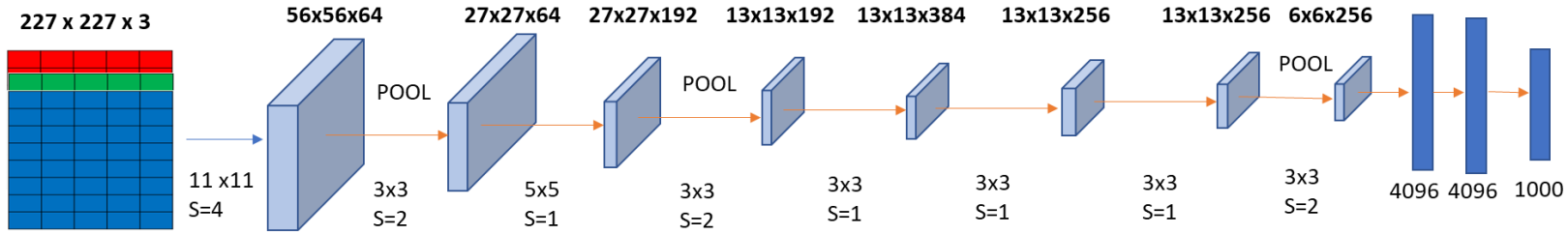


	Input size			Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664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

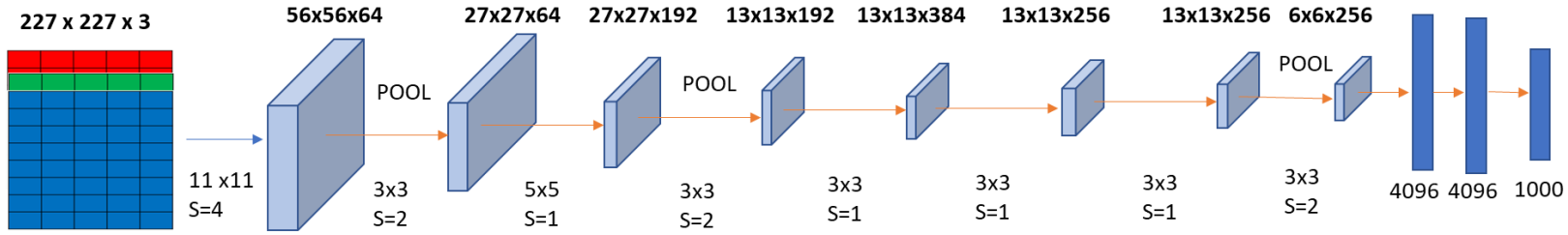
FC flops = $C_{in} * C_{out}$
= 9216 * 4096
= 37,748,736

AlexNet

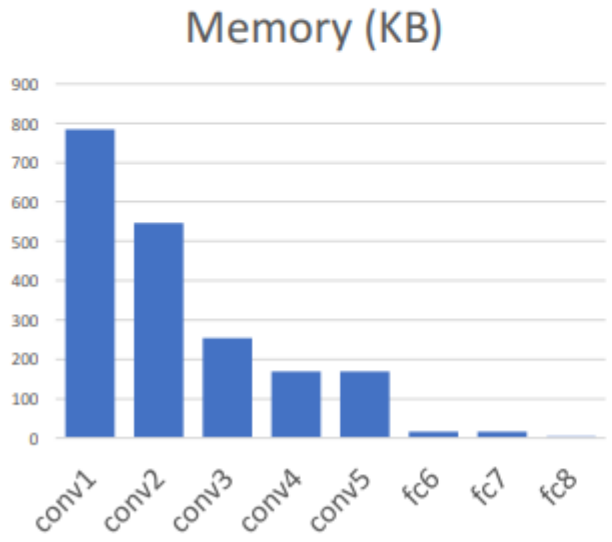


	Input size			Layer				Output size					
Layer	C	H / W	filters	kernel	stride	pad		C	H / W	memory (KB)	params (k)	flop (M)	
conv1	3	227	64	11	4	2		64	56	784	23	73	
pool1	64	56		3	2	0		64	27	182	0	0	
conv2	64	27	192	5	1	2		192	27	547	307	224	
pool2	192	27		3	2	0		192	13	127	0	0	
conv3	192	13	384	3	1	1		384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

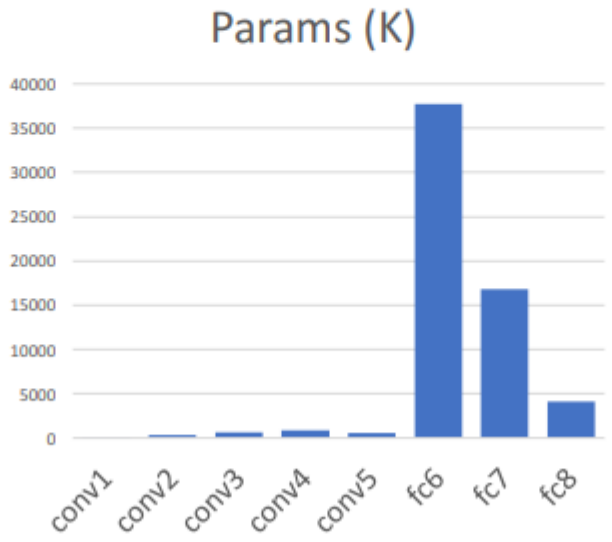
AlexNet



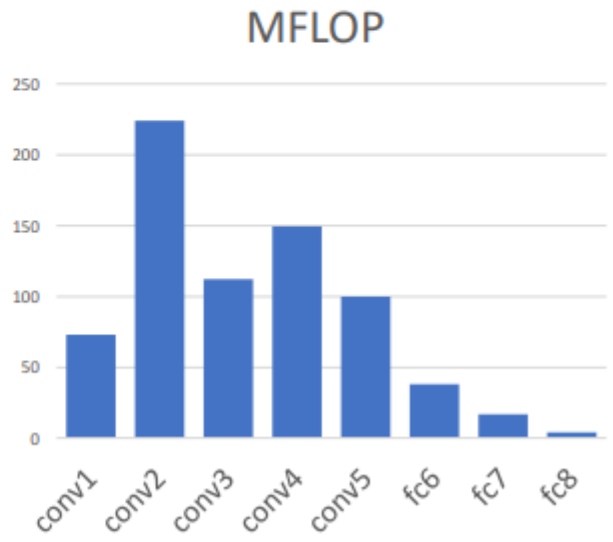
Most of the **memory usage** is in the early convolution layers



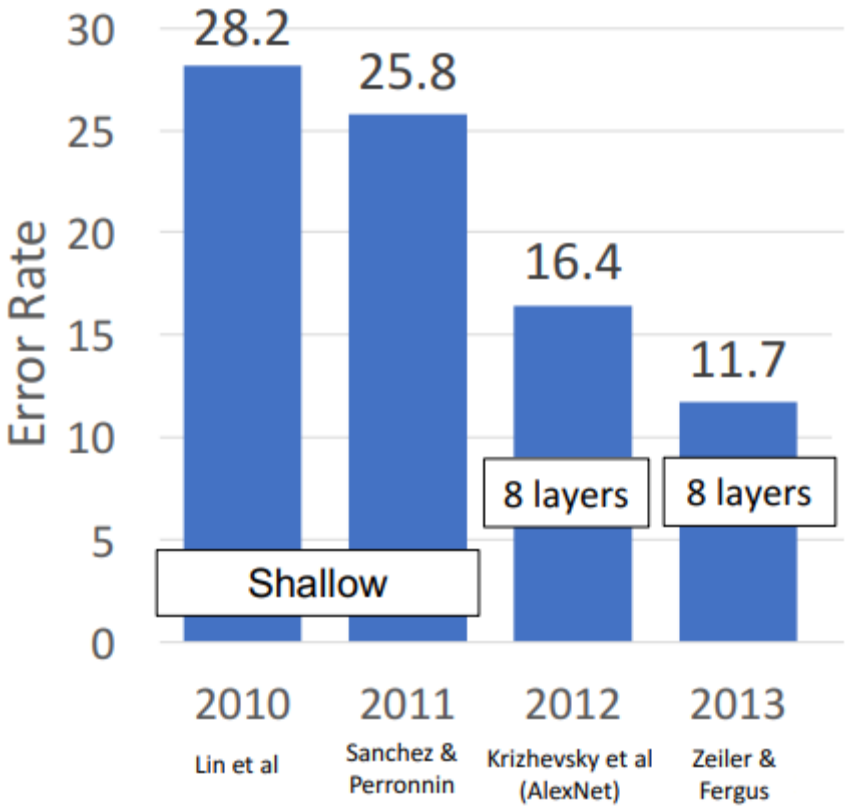
Nearly all **parameters** are in the fully-connected layers



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



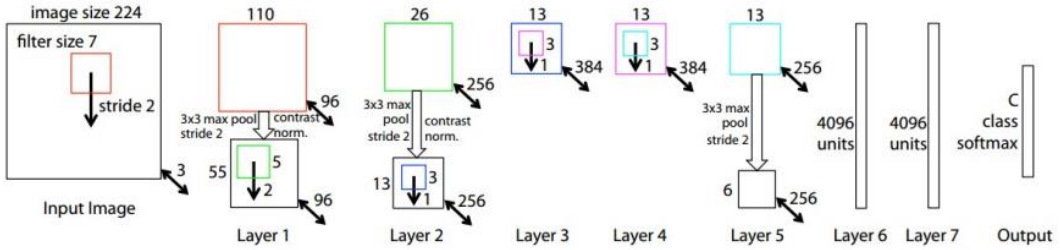
ImageNet Classification Challenge



ImageNet Classification Challenge

- ZFNet: A Bigger AlexNet (2013):

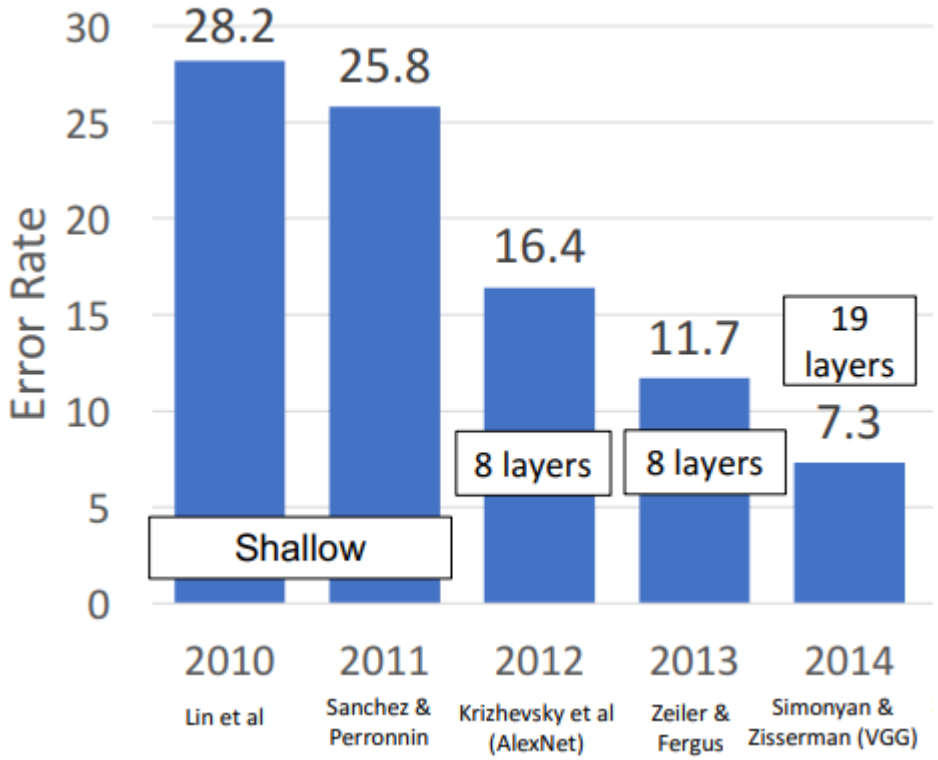
ImageNet top 5 error: 16.4% -> 11.7%



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

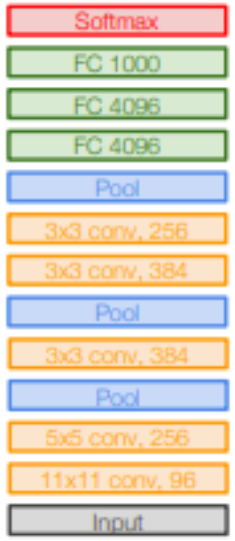
Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014

ImageNet Classification Challenge



VGG: Deeper Networks, Regular Design

- The **Visual Geometry Group (VGG)** at Oxford University proposed the VGGNet architecture
- **VGG Design rules:**
  - All conv are 3x3 stride 1 pad 1
  - All max pool are 2x2 stride 2
  - After pool, double #channels
- Network has 5 convolutional **stages**:



AlexNet

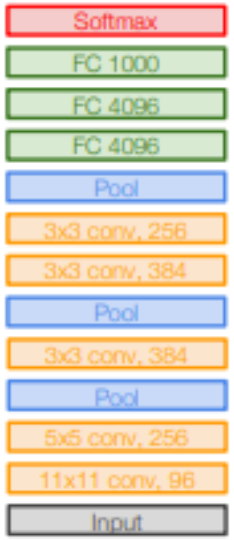
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

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

VGG

- **VGG Design rules:**
  - **All conv are 3x3 stride 1 pad 1**
  - All max pool are 2x2 stride 2
  - After pool, double #channels
- **Why 2 Double layers?**
  - Assume Conv layer input C channel and output C channel.

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: $18C^2HW$



AlexNet

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

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

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

VGG

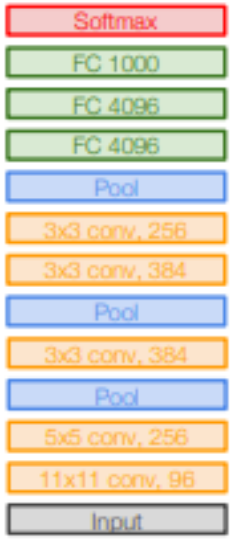
- **VGG Design rules:**
  - All conv are 3x3 stride 1 pad 1
  - **All max pool are 2x2 stride 2**
  - **After pool, double #channels**

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

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

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

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



AlexNet

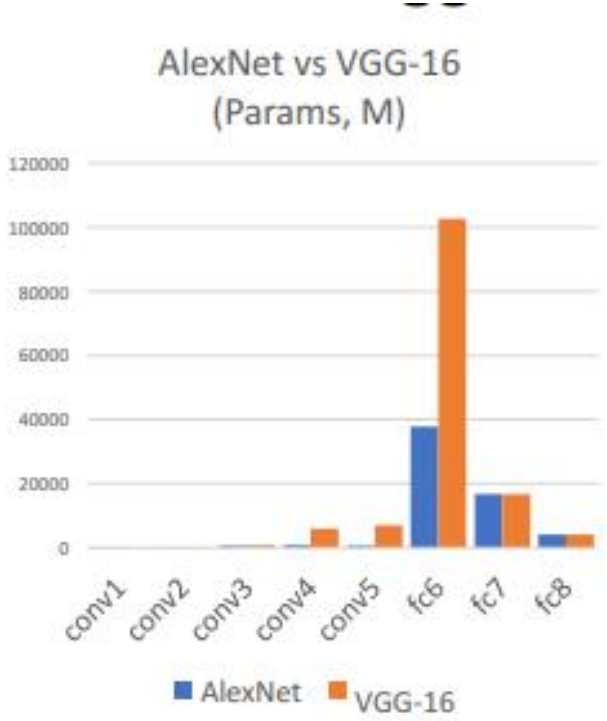
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

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

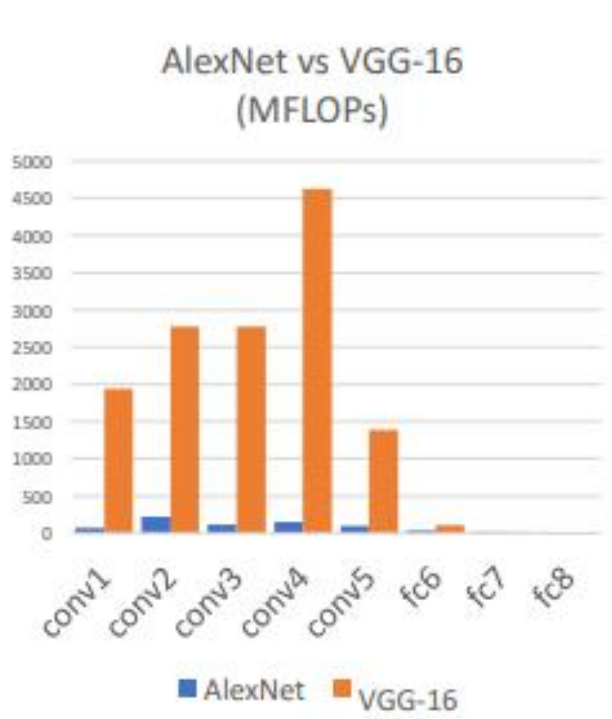
Simonyan and Zissermann, “Very Deep Convolutional Networks for Large-Scale Image Recognition”, ICLR 2015

AlexNet vs VGG-16: Much bigger network!

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

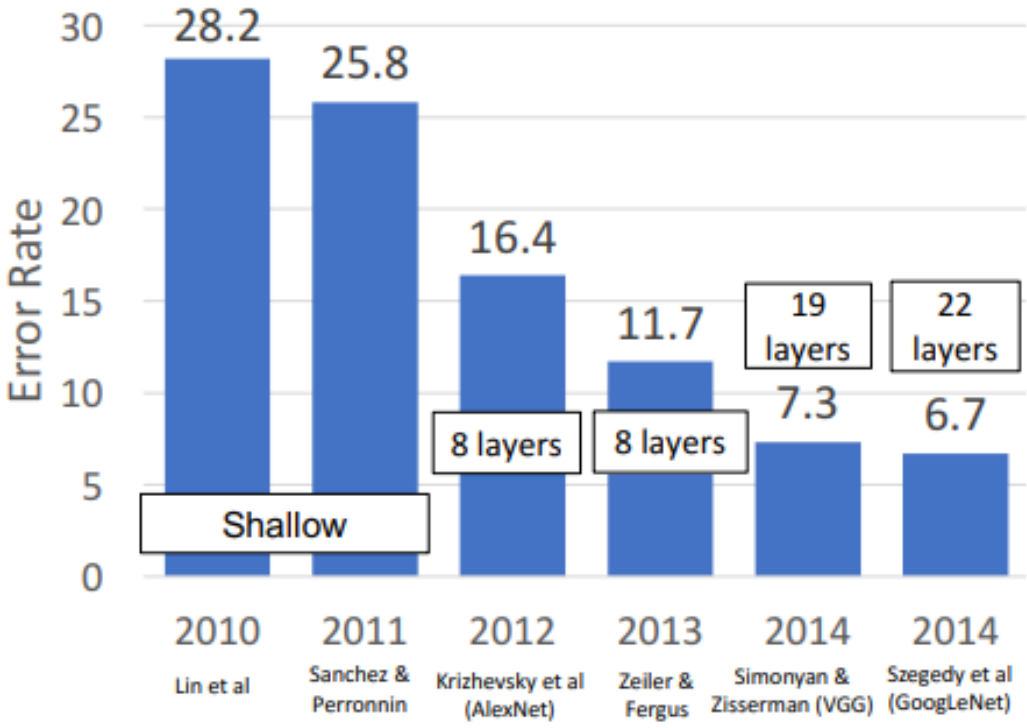


AlexNet total: 61M
VGG-16 total: 138M (2.3x)



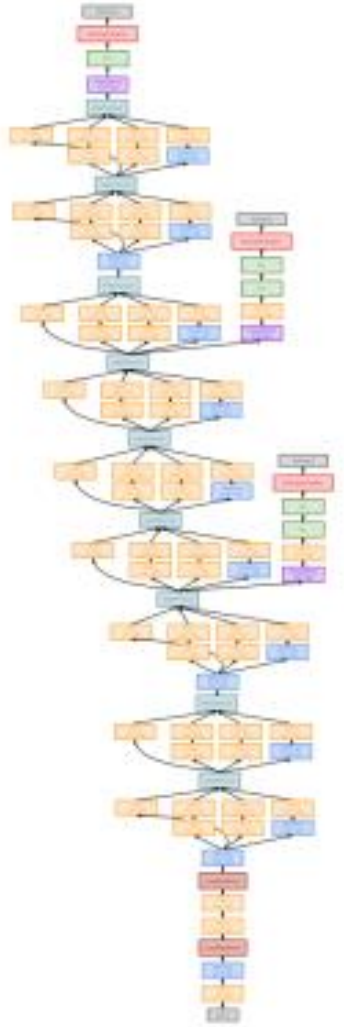
AlexNet total: 0.7 GFLOP
VGG-16 total: 13.6 GFLOP (19.4x)

ImageNet Classification Challenge



GoogLeNet: Focus on Efficiency

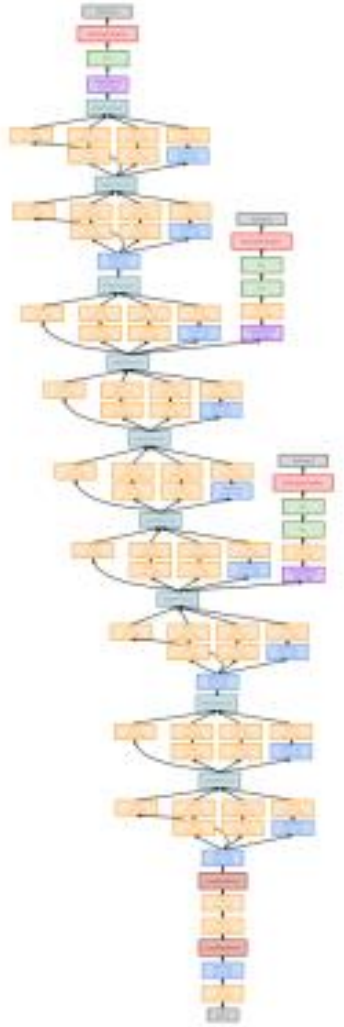
- Many innovations for efficiency:
  reduce parameter count,
  memory usage, and computation



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

GoogLeNet: Focus on Efficiency

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



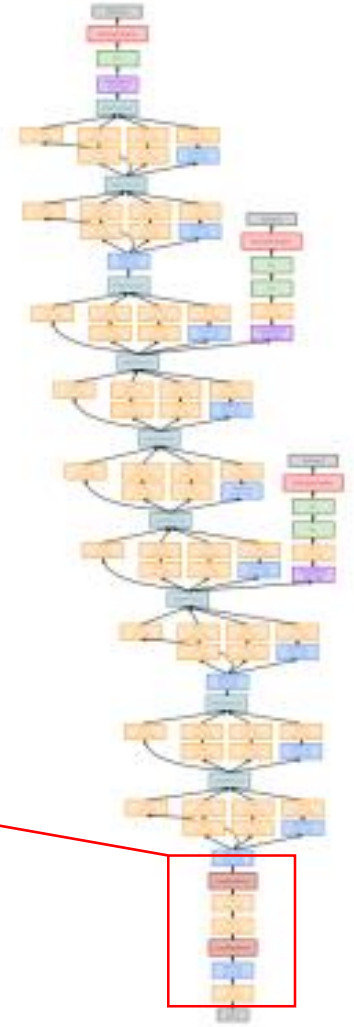
Szegedy et al, “Going deeper with convolutions”, CVPR 2015

GoogLeNet: Aggressive Stem

- **Stem network** at the start aggressively down samples 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



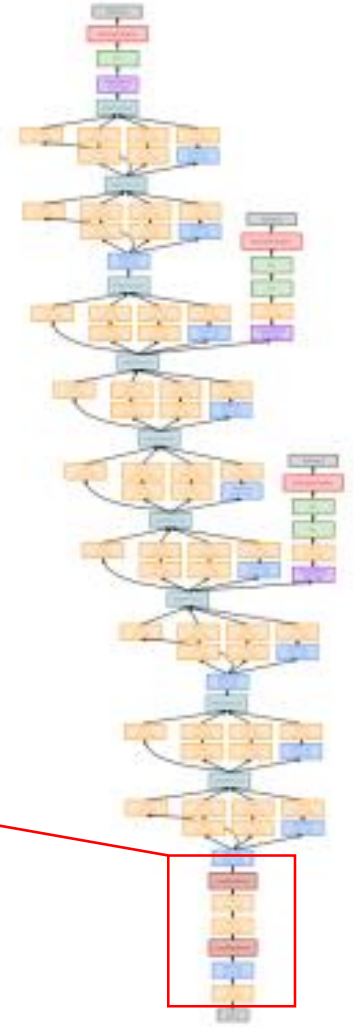
Szegedy et al, “Going deeper with convolutions”, CVPR 2015

GoogLeNet: Aggressive Stem

- **Stem network** at the start aggressively down samples 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

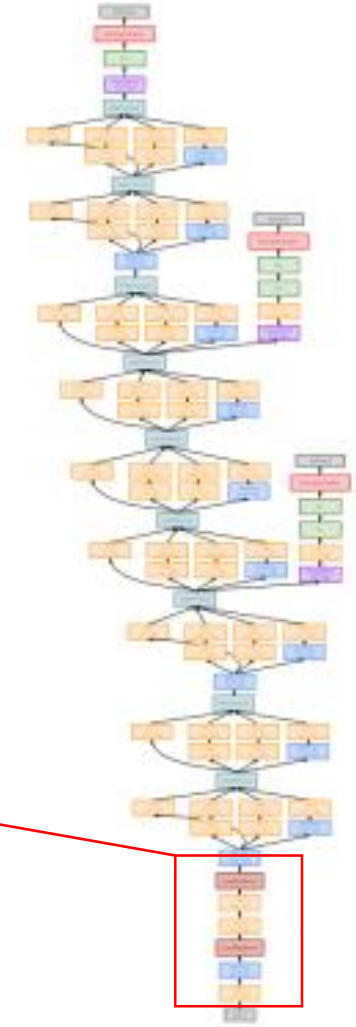


Szegedy et al, “Going deeper with convolutions”, CVPR 2015

GoogLeNet: Aggressive Stem

- **Stem network** at the start aggressively down samples 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

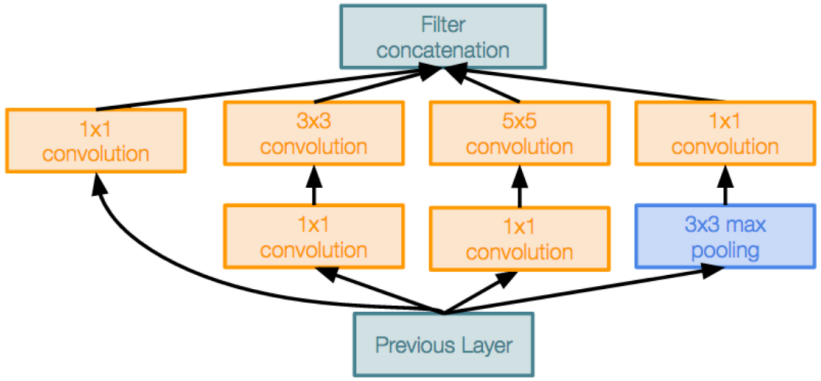


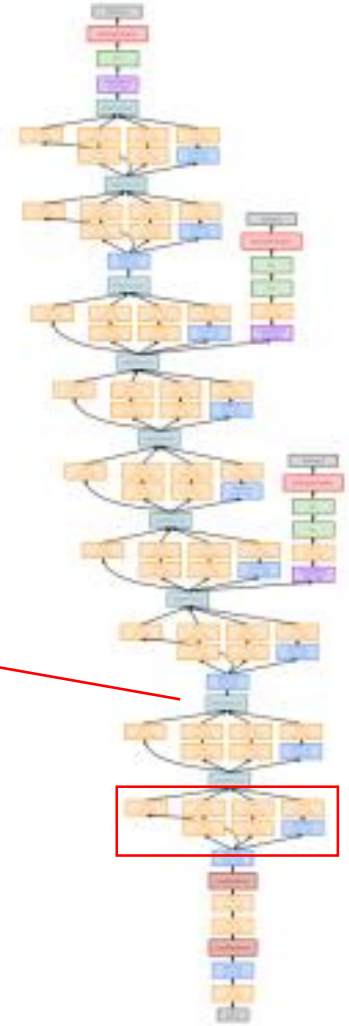
Total from 224 to 28 spatial resolution:
Memory: 7.5 MB
Params: 124K
MFLOP: 418

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



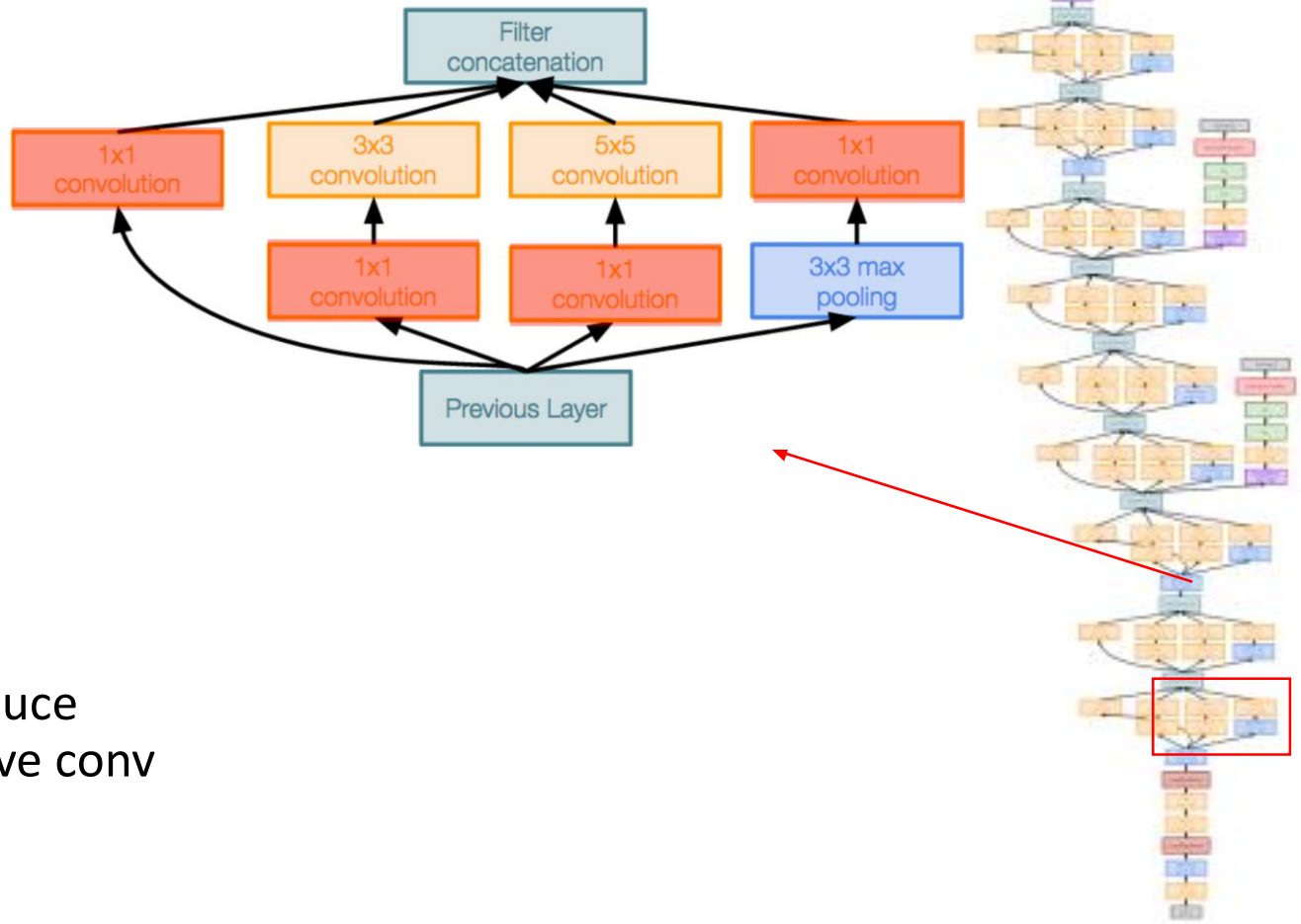


- **Inception module**
  - Local unit with **parallel branches**
  - Local structure **repeated** many times throughout the network

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



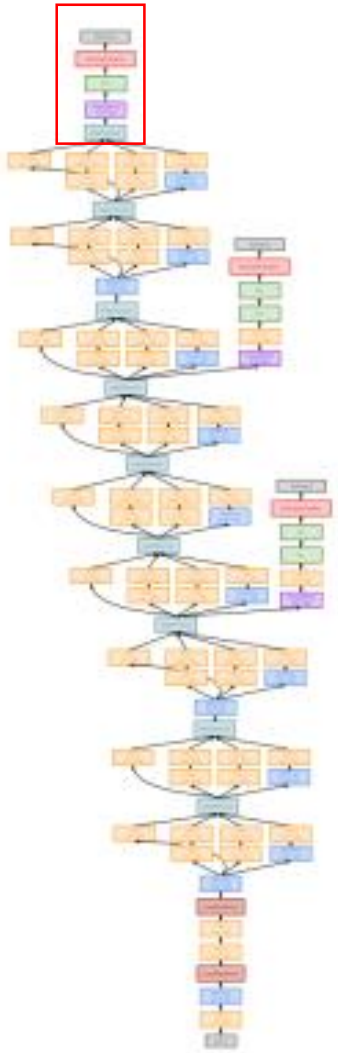
- Uses 1x1 "Bottleneck" layers to reduce channel dimension before expensive conv (we will revisit this with ResNet!)

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



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

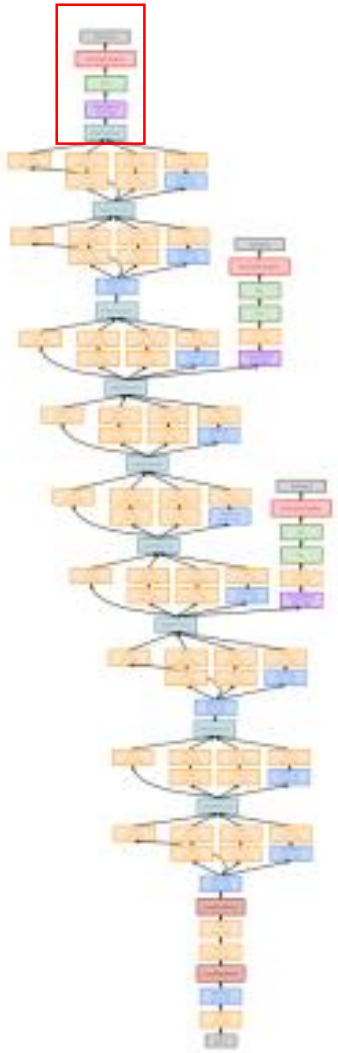
GoogLeNet: Global Average Pooling

- **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

Compare with VGG-16:

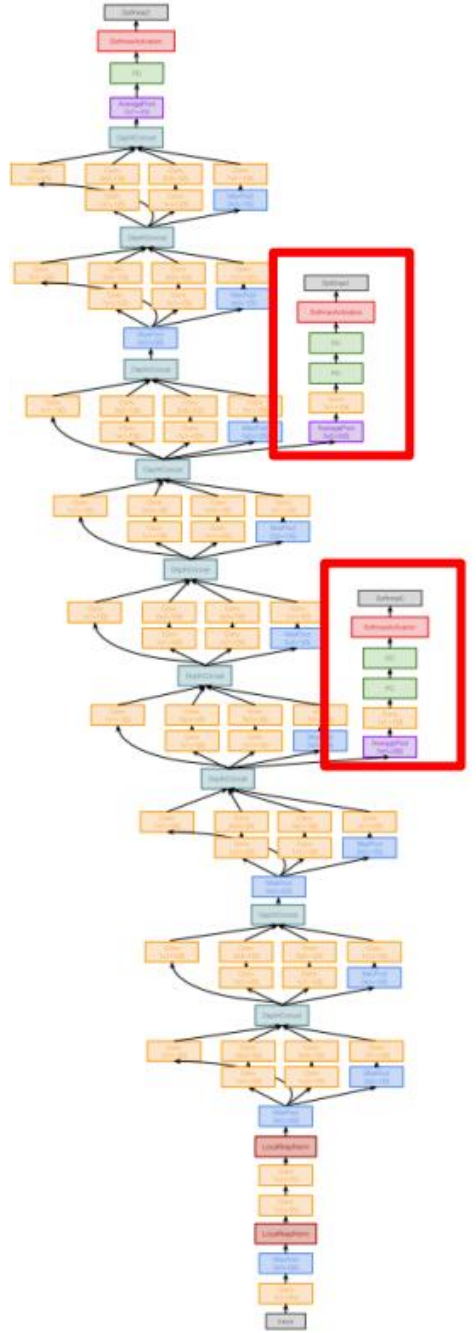
Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (K)	flop (M)
flatten	512	7					25088		98		
fc6	25088			4096			4096		16	102760	103
fc7	4096			4096			4096		16	16777	17
fc8	4096			1000			1000		4	4096	4



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

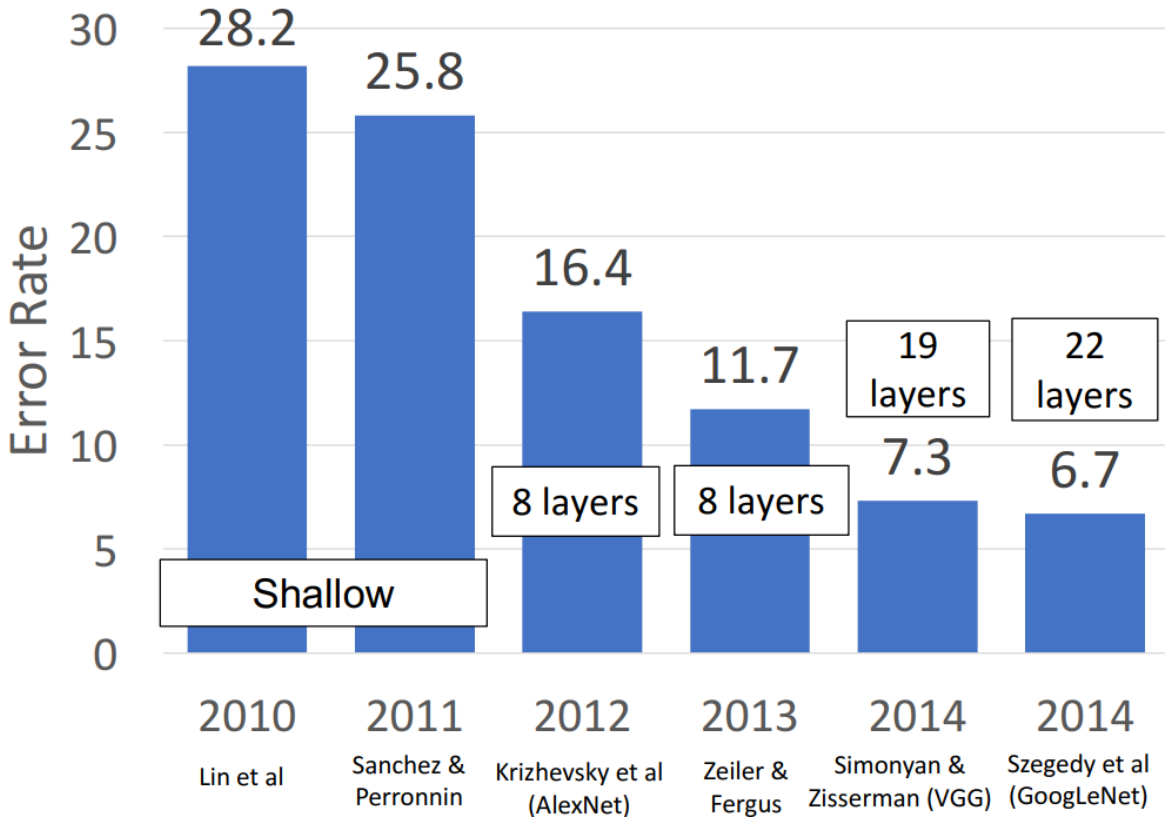
GoogLeNet: Auxiliary Classifiers

- 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.

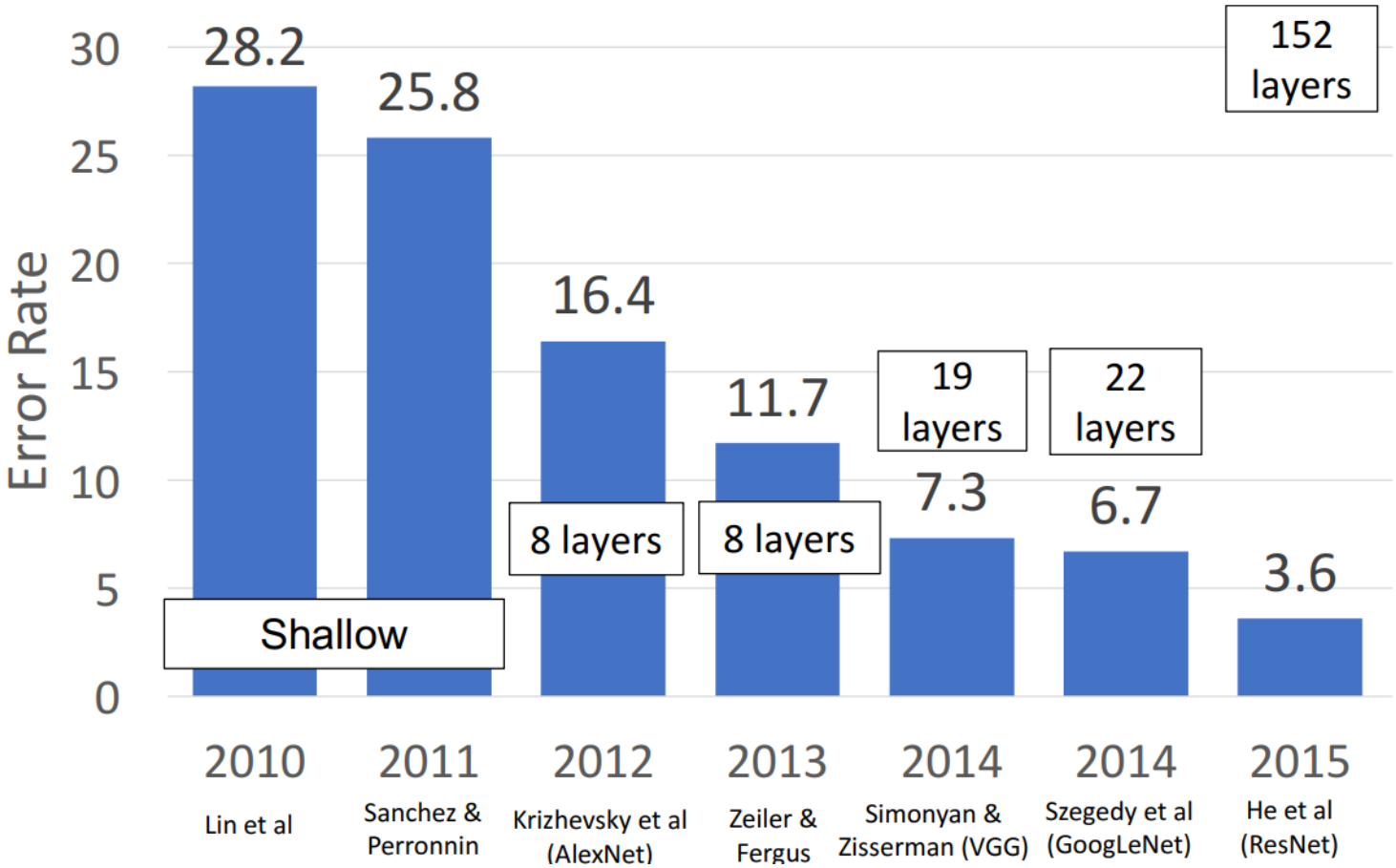


Szegedy et al, "Going deeper with convolutions", CVPR 2015

ImageNet Classification Challenge



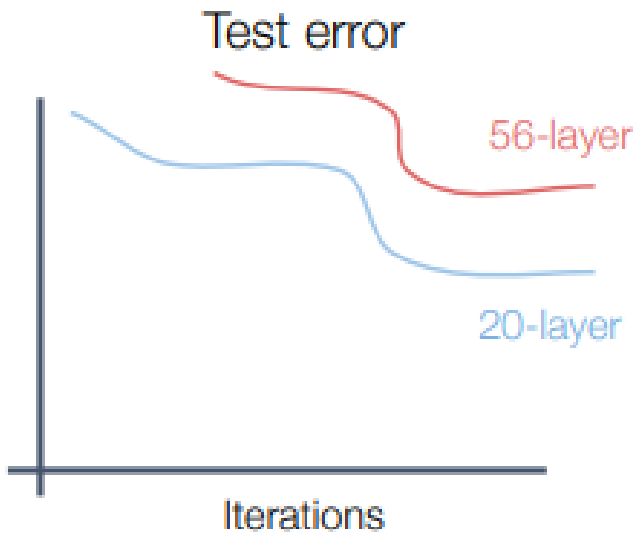
ImageNet Classification Challenge



Residual Networks

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

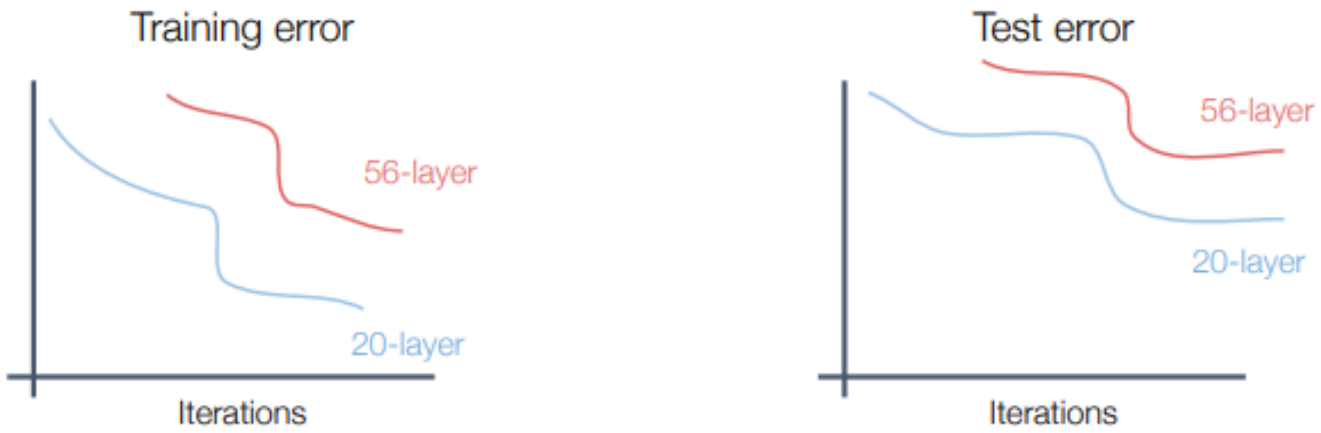
  - Deeper model does worse than shallow model!
  - Initial guess: Deep model is **overfitting** since it is much bigger than the other model.



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

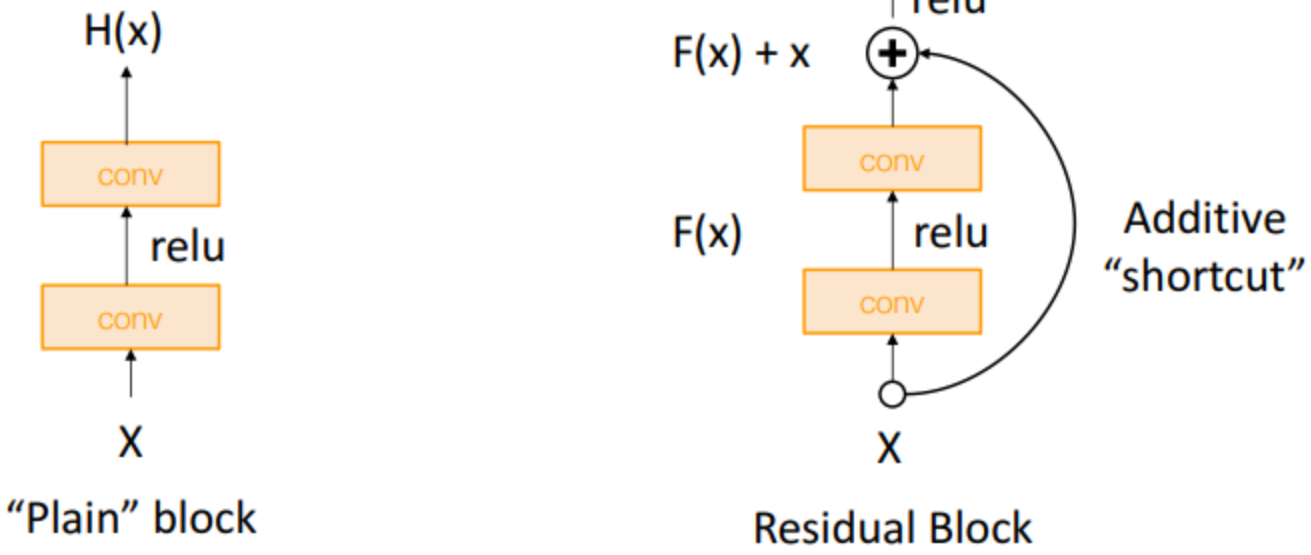
Residual Networks

- 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!

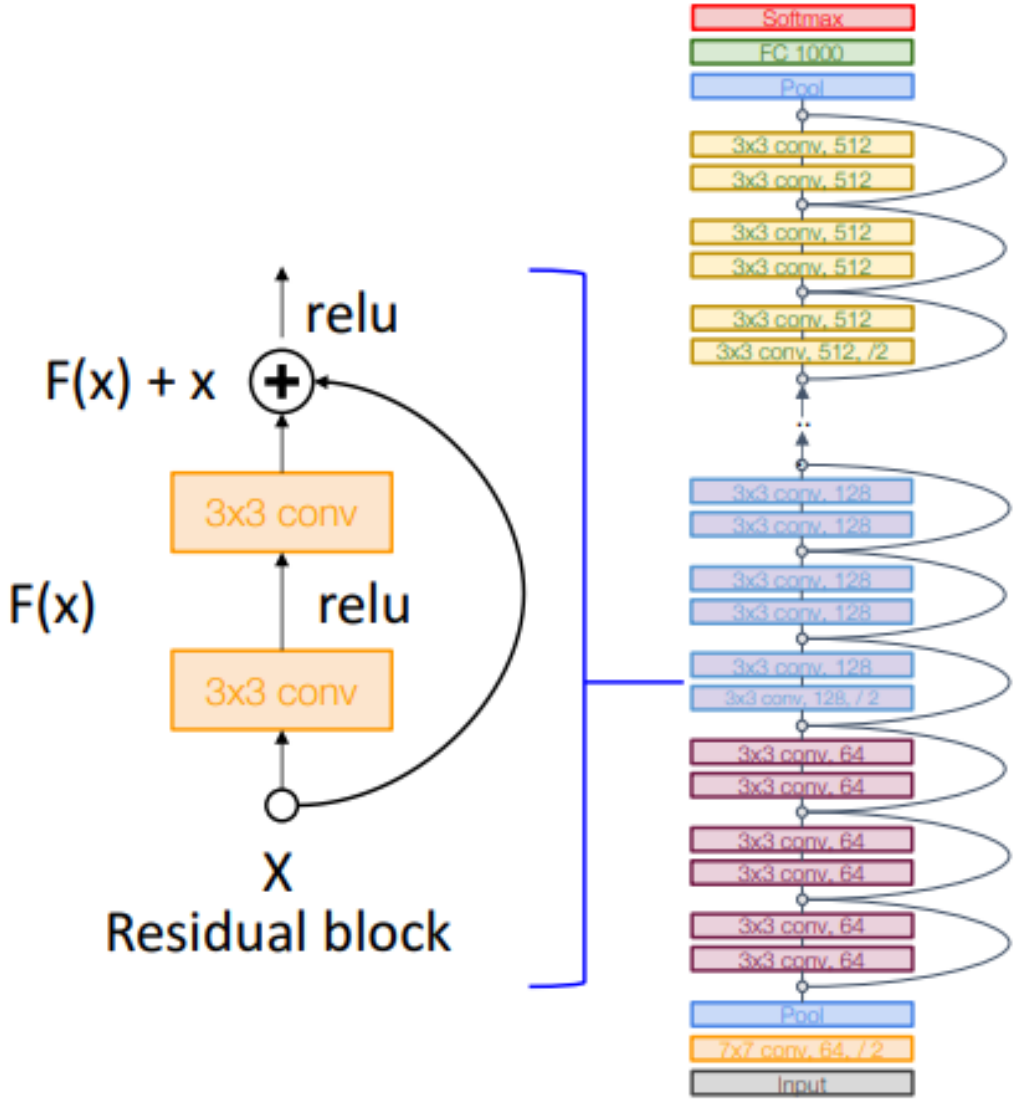


Gradient can propagate faster backwards since the addition gate map gradients equally through its outgoing edges, and gradients on the residual edges can "skip" other layers.

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

Residual Networks

- Resnet:
  - Is a stack of many **residual blocks.**
  - Inspired by 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

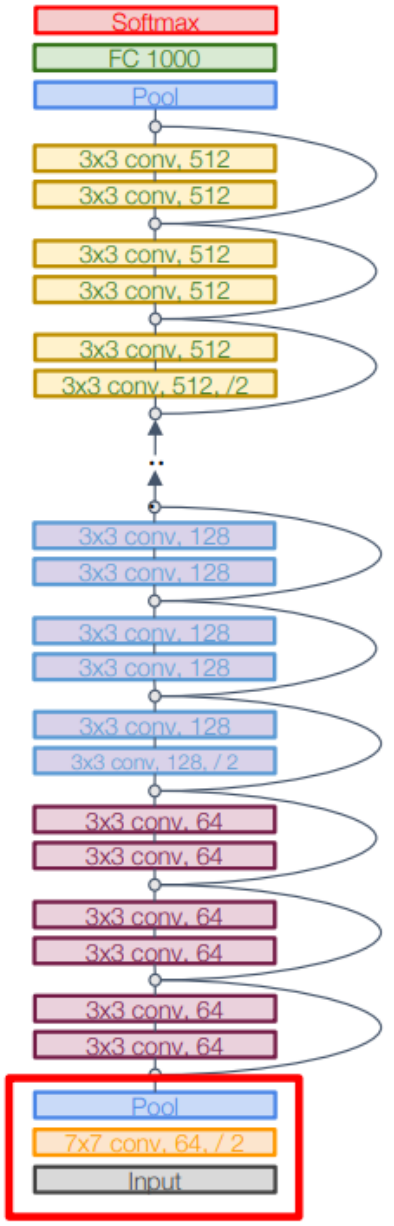
Residual Networks

- Residual learning in ResNet simplifies training deep networks by using skip connections to learn the difference (residual) between input and output, rather than the full mapping.
  - **Mitigates vanishing gradients** by providing direct paths for gradient flow.
  - **Solves degradation** by allowing identity mappings, preventing performance loss as depth increases.
  - **Improves feature learning** by focusing on refining features rather than relearning them.
- Residual blocks enable very deep networks (e.g., ResNet-152) to train effectively and achieve state-of-the-art performance.

Residual Networks

- Resnet also uses the same aggressive stem as GoogleNet to down sample the input 4x before applying residual blocks:

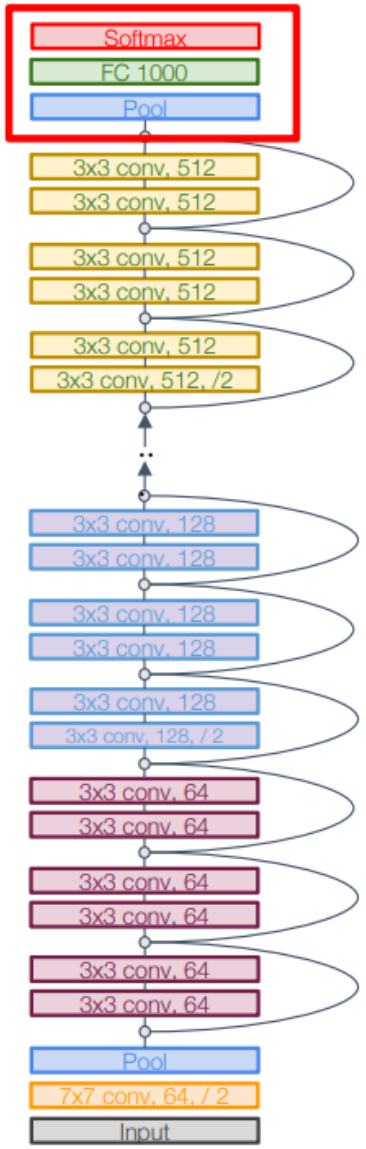
	Input size		Layer				Output size				
Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (k)	flop (M)
conv	3	224	64	7	2	3	64	112	3136	9	118
max-pool	64	112		3	2	1	64	56	784	0	2



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

Residual Networks

- Resent us the idea from 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

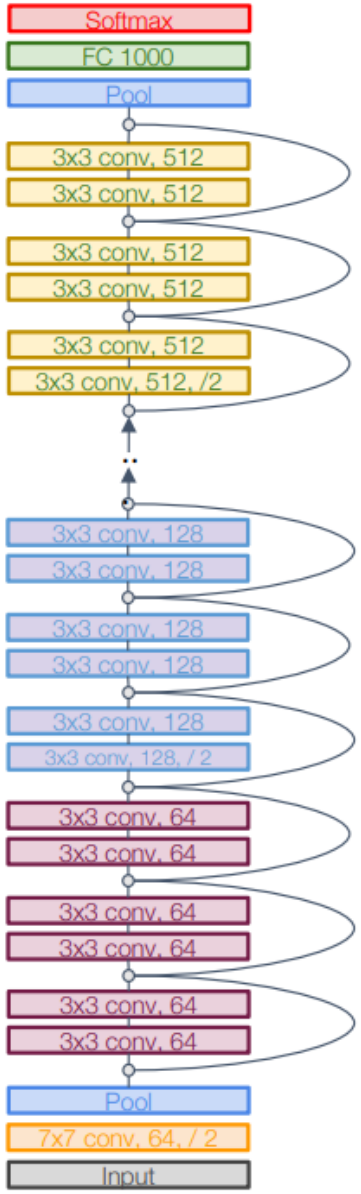
ResNet-18:
Stem: 1 conv layer
Stage 1 (C=64): 2 res. block = 4 conv
Stage 2 (C=128): 2 res. block = 4 conv
Stage 3 (C=256): 2 res. block = 4 conv
Stage 4 (C=512): 2 res. block = 4 conv
Linear

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

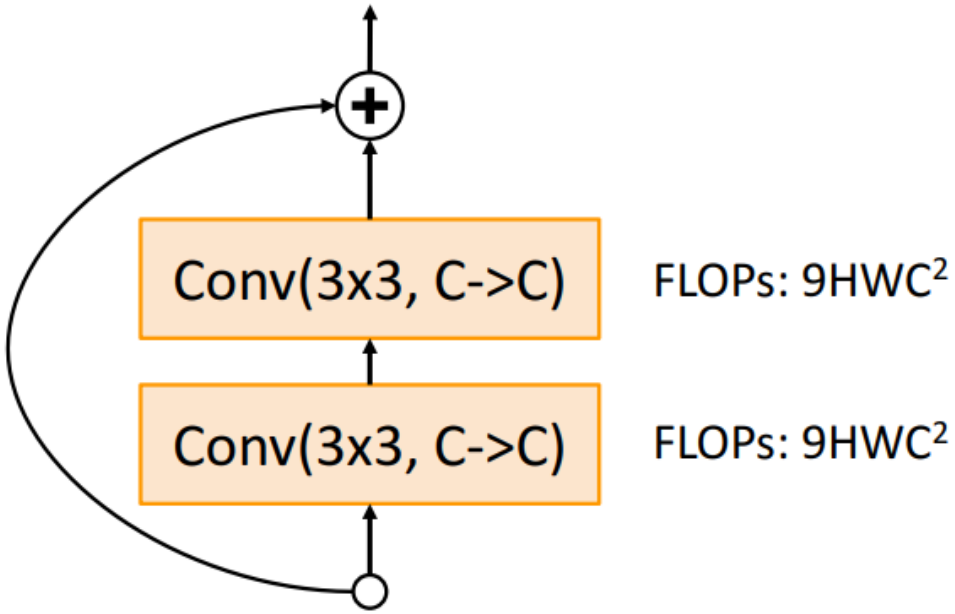
ImageNet top-5 error: 8.58
GFLOP: 3.6

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

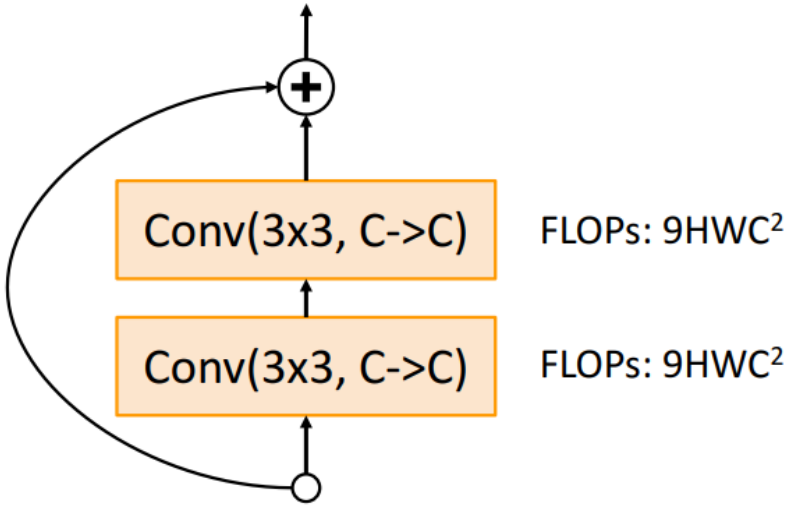


"Basic"
Residual block

Total FLOPs:
18HWC²

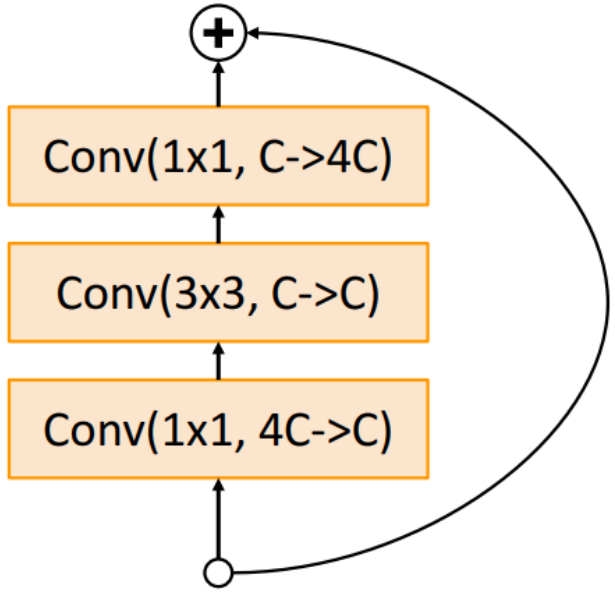
He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Residual Networks: Bottleneck Block



"Basic"
Residual block

Total FLOPs:
18HWC²

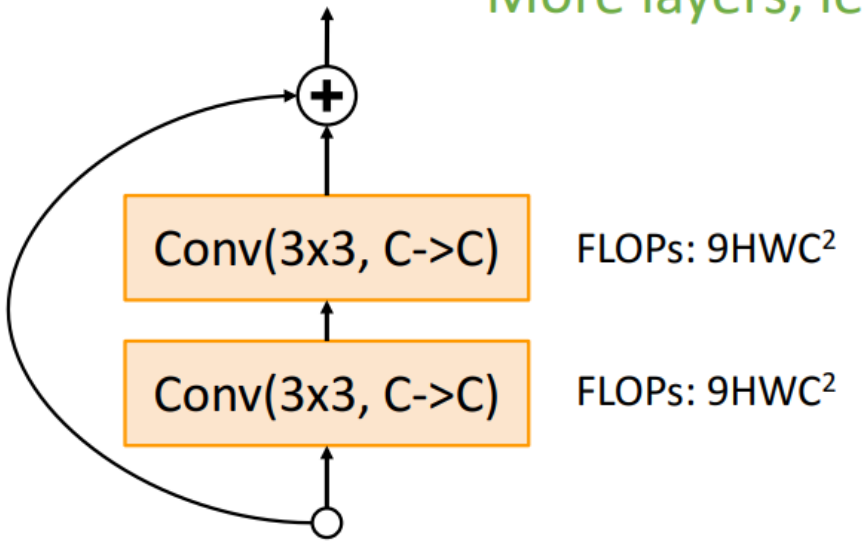


"Bottleneck"
Residual block

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

Residual Networks: Bottleneck Block

More layers, less computational cost!

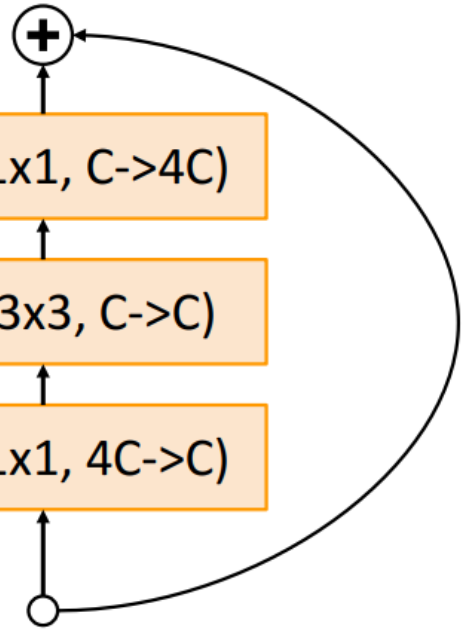


FLOPs: $9HWC^2$

FLOPs: $9HWC^2$

"Basic" Residual block

Total FLOPs: $18HWC^2$



FLOPs: $4HWC^2$

FLOPs: $9HWC^2$

FLOPs: $4HWC^2$

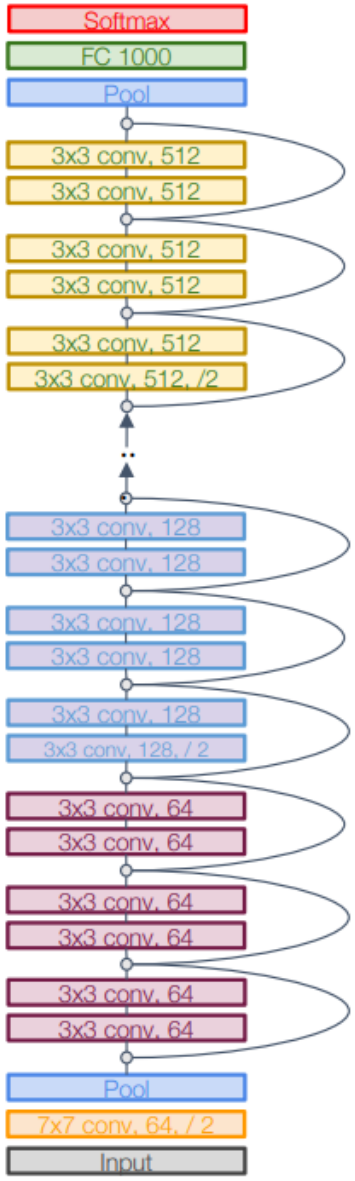
Total FLOPs: $17HWC^2$

"Bottleneck" Residual block

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

Residual Networks

			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

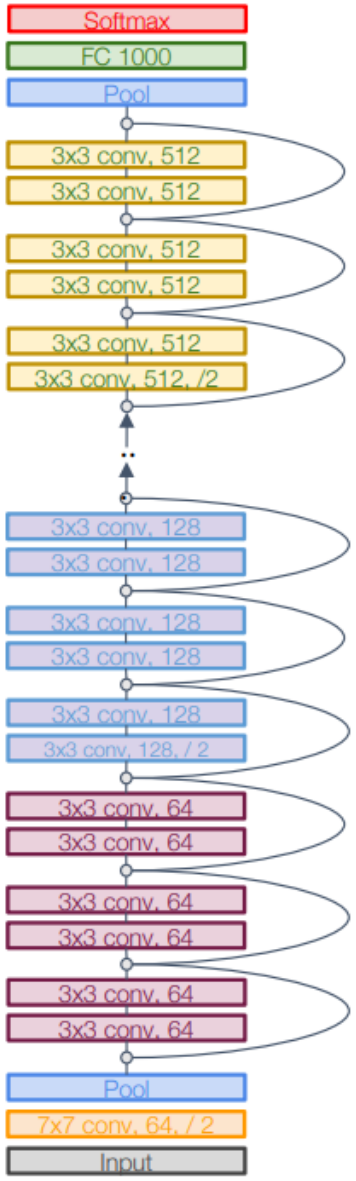


He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016
Error rates are 224x224 single-crop testing, reported by torchvision

Residual Networks

- ResNet-50 is the same as ResNet-34, but replaces Basic blocks with Bottleneck Blocks.
- The error decreases! By simply making it deeper without increasing computation.
- 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

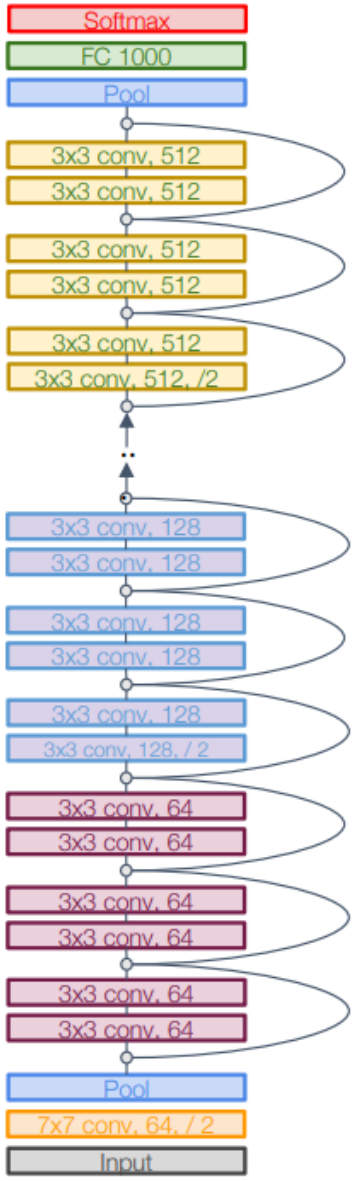


He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016

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

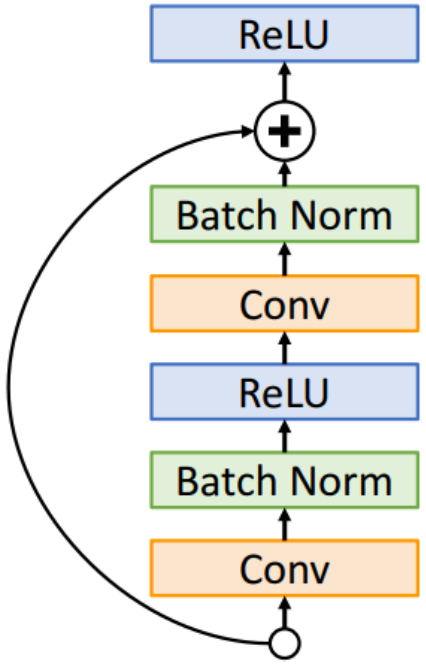
Residual Networks

- 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



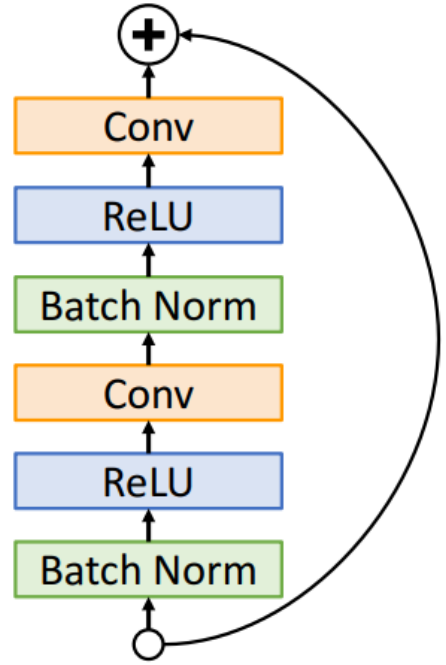
Note ReLU **after** residual:

Cannot actually learn identity function since outputs are nonnegative!

Note ReLU **inside** residual:

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

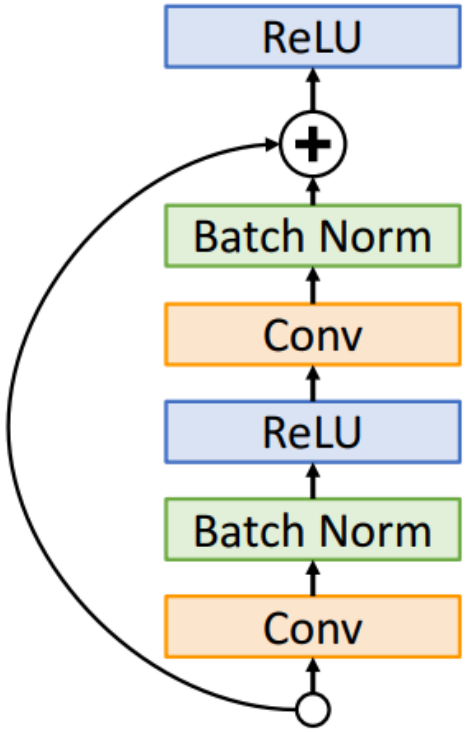
"Pre-Activation" ResNet Block



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

Improving Residual Networks: Block Design

Original ResNet block

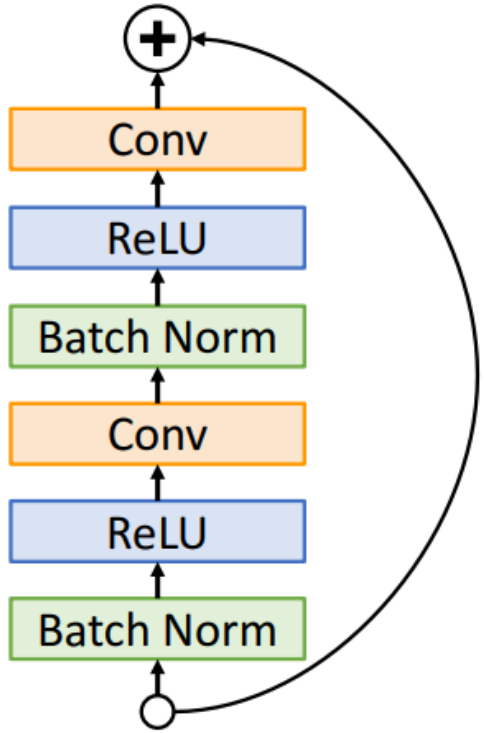


Slight improvement in accuracy (ImageNet top-1 error)

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

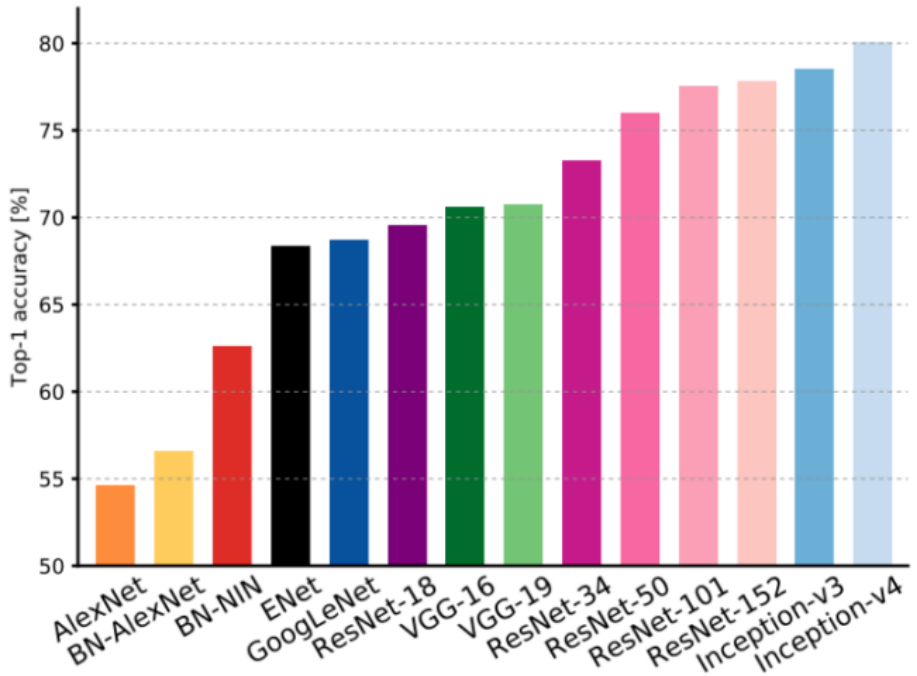
Not actually used that much in practice

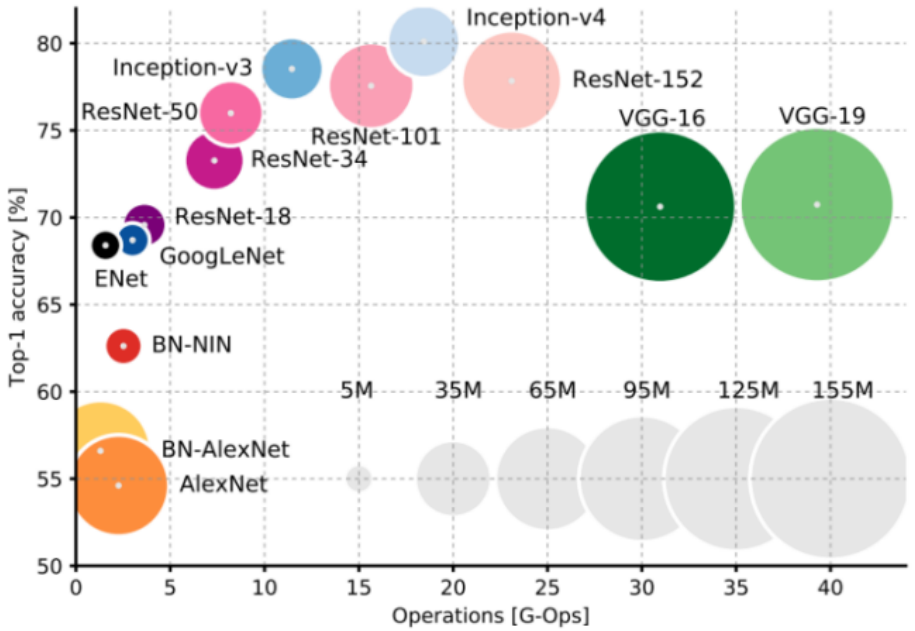
"Pre-Activation" ResNet Block



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

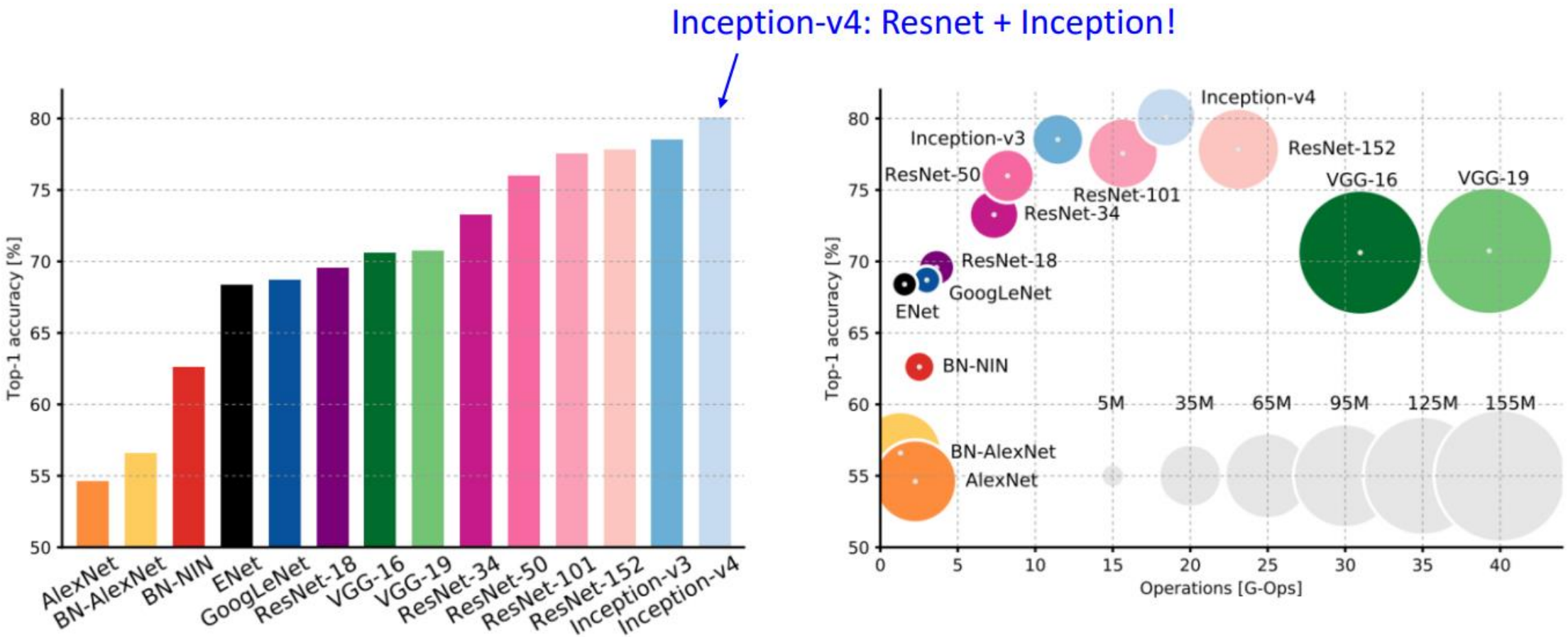
Comparing Complexity





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

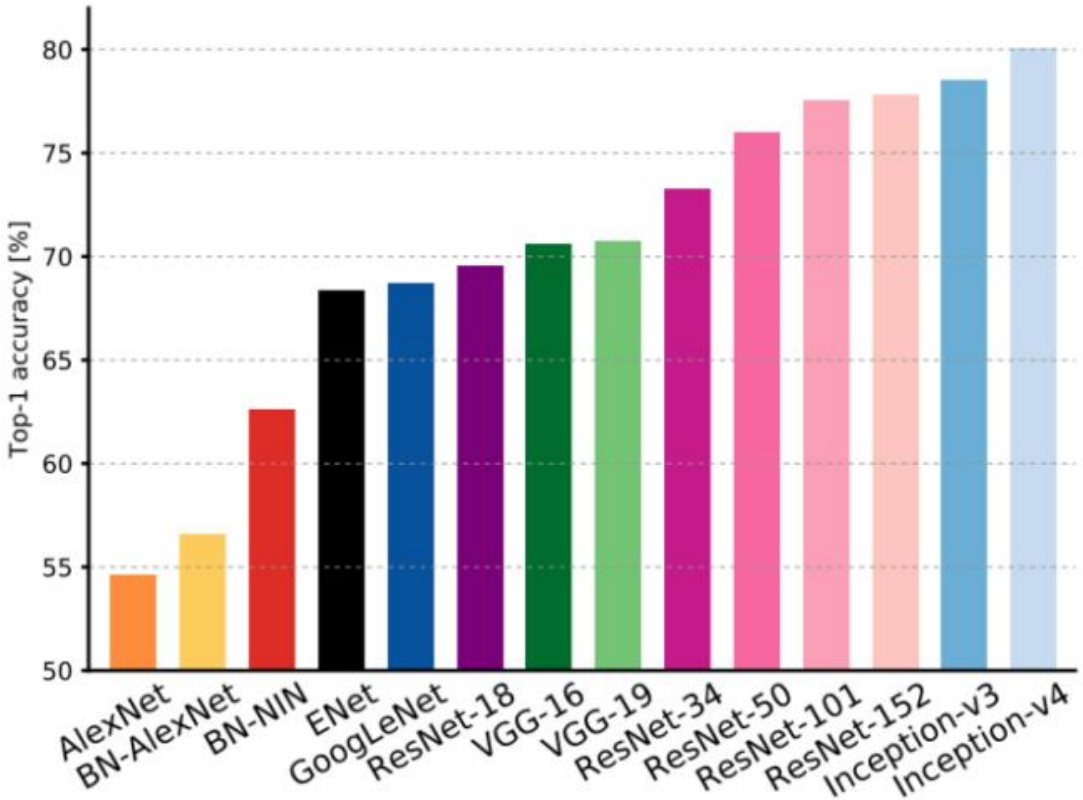
Comparing Complexity

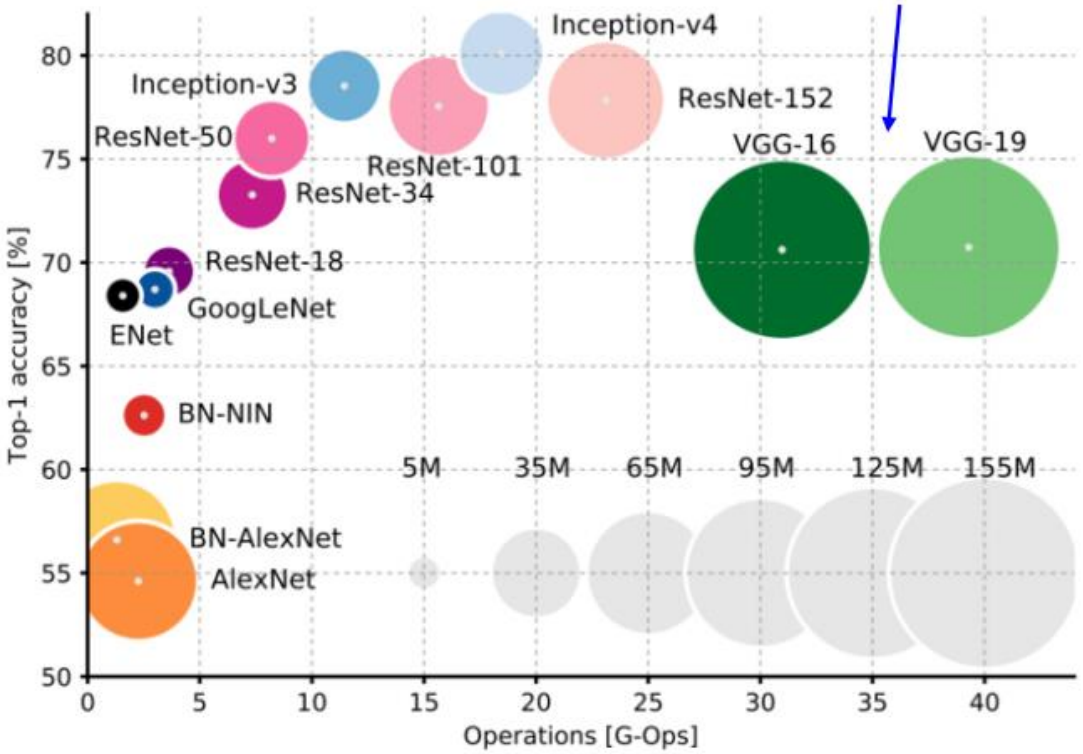


Canziani et al, “An analysis of deep neural network models for practical applications”, 2017

Comparing Complexity

VGG: Highest memory, most operations

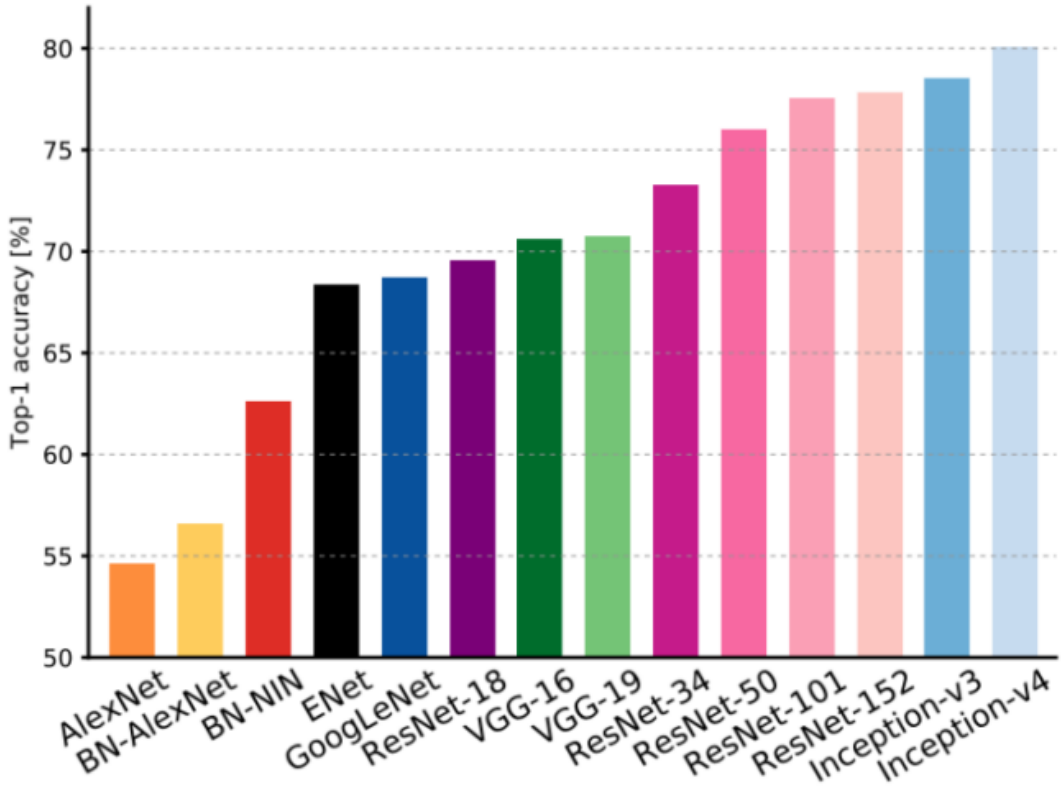


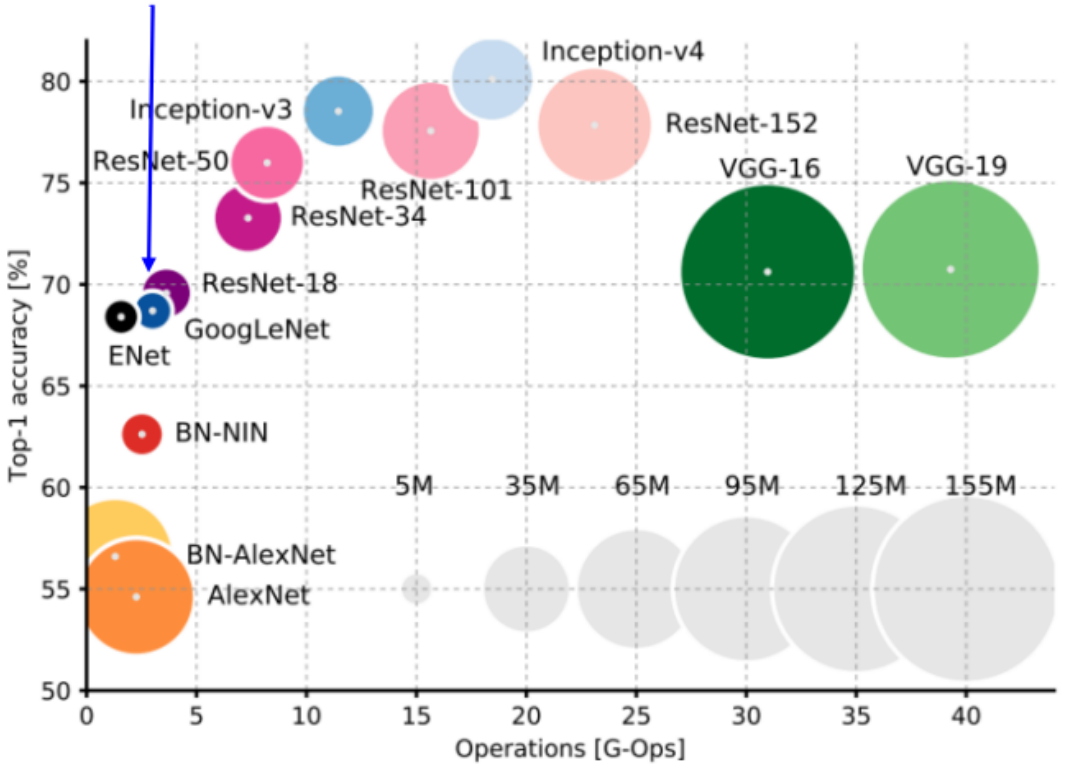


Canziani et al, “An analysis of deep neural network models for practical applications”, 2017

Comparing Complexity

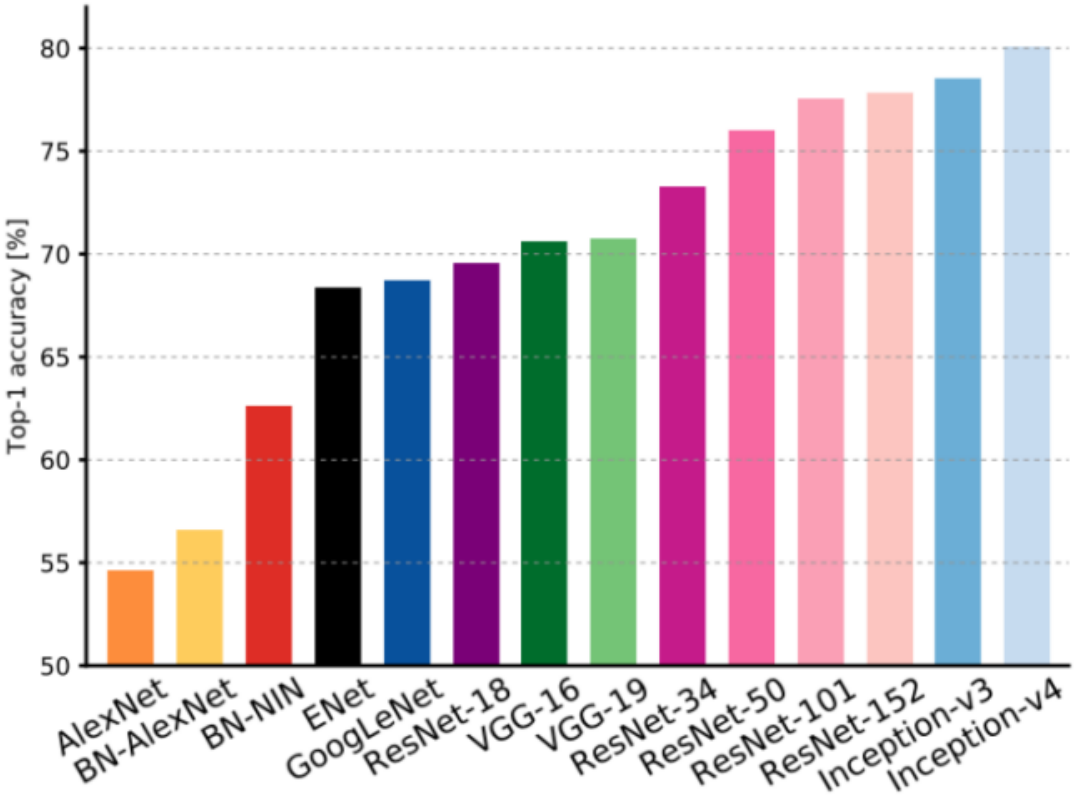
GoogLeNet:
Very efficient!



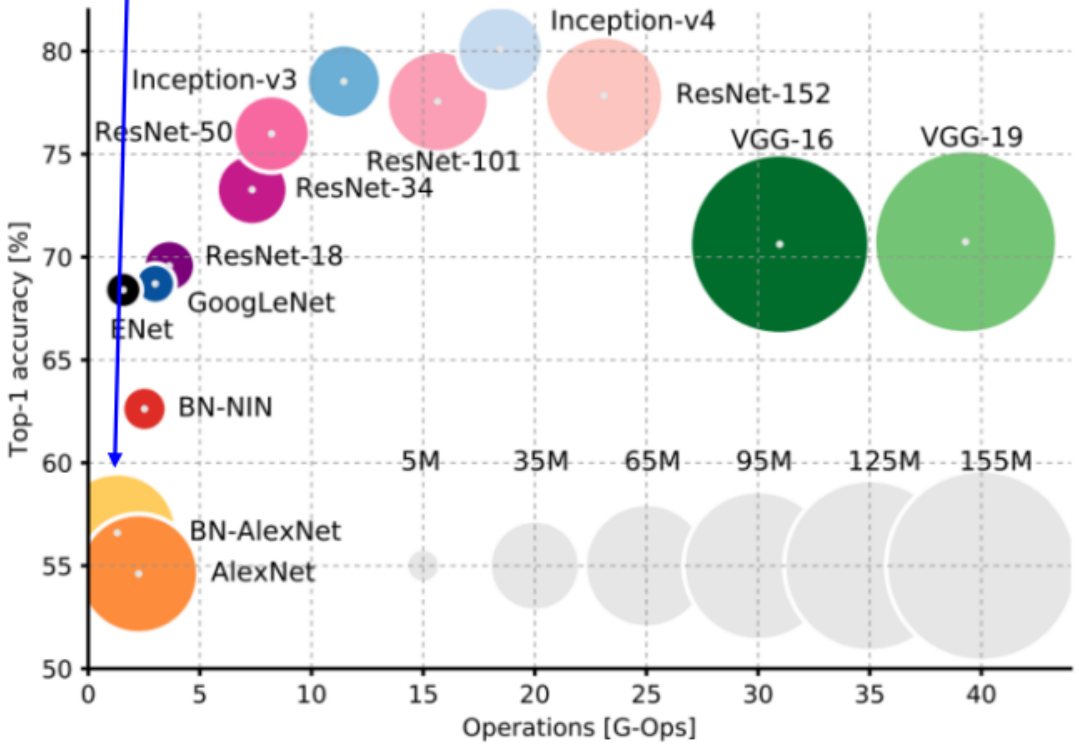


Canziani et al, “An analysis of deep neural network models for practical applications”, 2017

Comparing Complexity

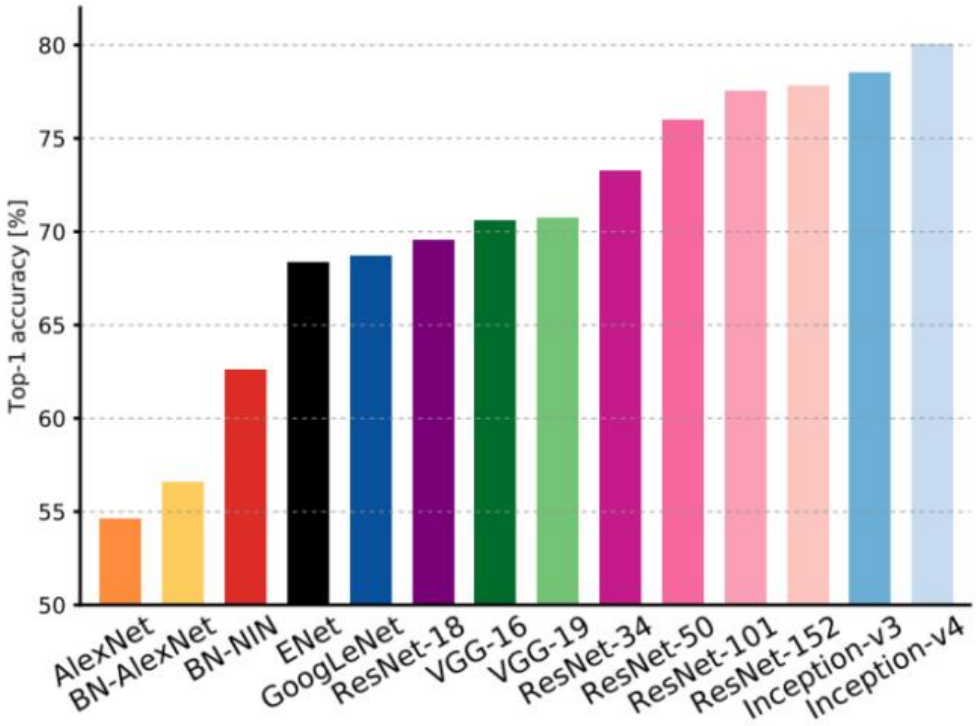


AlexNet: Low compute, lots of parameters

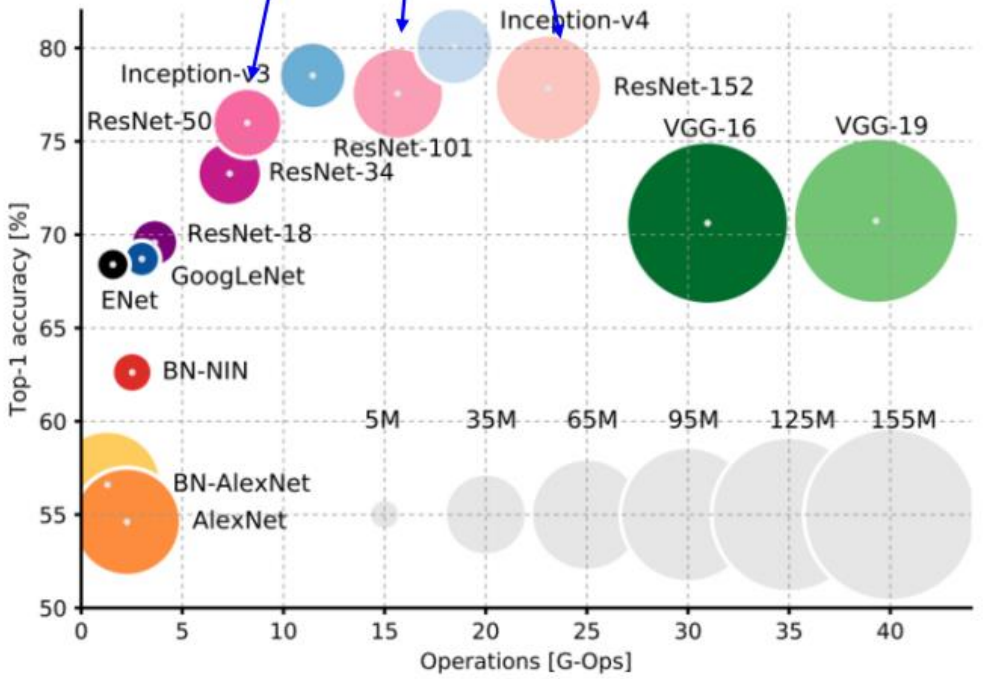


Canziani et al, “An analysis of deep neural network models for practical applications”, 2017

Comparing Complexity



ResNet: Simple design, moderate efficiency, high accuracy

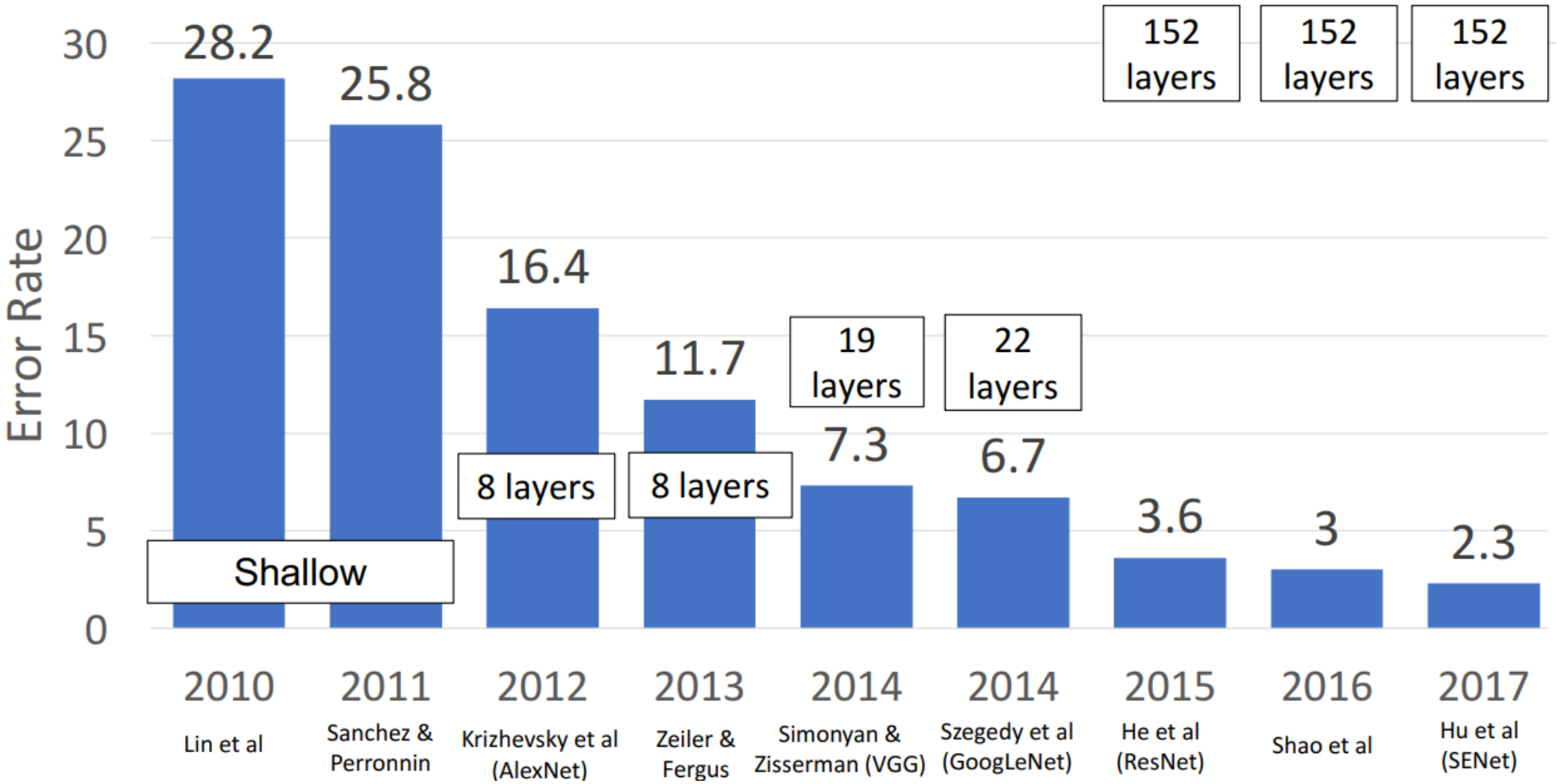


Canziani et al, “An analysis of deep neural network models for practical applications”, 2017

Other CNN Architectures

- ImageNet 2016 winner: Model Ensembles
- Improving ResNets: ResNeXt (The next generation of ResNet)
- Squeeze-and-Excitation Networks

ImageNet Classification Challenge



ImageNet Classification Challenge

Other CNN Architectures

- Densely Connected Neural Networks
- MobileNets: Tiny Networks (For Mobile Devices)
  - ShuffleNet: Zhang et al, CVPR 2018
  - MobileNetV2: Sandler et al, CVPR 2018
  - ShuffleNetV2: Ma et al, ECCV 2018

CNN Architectures Summary

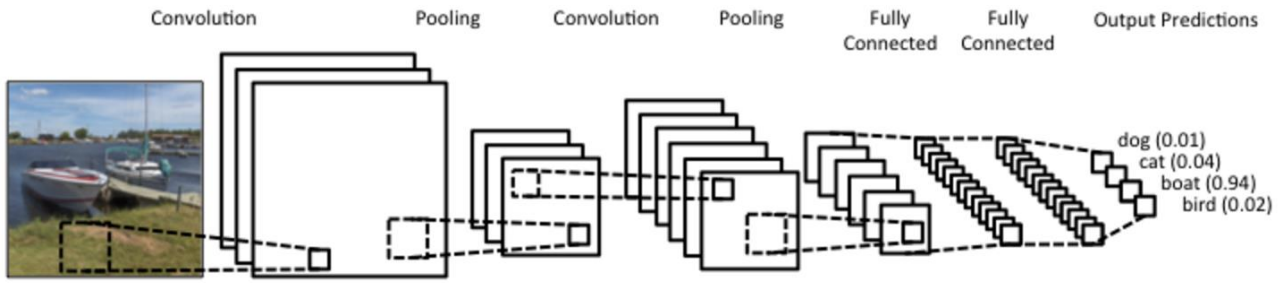
- Early work (AlexNet -> ZFNet -> VGG) shows that **bigger networks work better**
- **GoogLeNet** one of the first to focus on **efficiency** (aggressive stem, 1x1 bottleneck convolutions, global avg pool instead of FC layers)
- **ResNet** showed us how to train **extremely deep networks, residual block.**
- 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

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**

How to use it

- Transfer learning



- Take the vector of activations from one of the fully connected (FC) layers and treat it as an off-the-shelf feature
- Train a new classifier layer on top of the FC layer
- *Fine-tune* the whole network

References

- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. NIPS.
- Zeiler, M. D., & Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. ECCV.
- Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR.
- Szegedy, C., et al. (2015). Going deeper with convolutions. CVPR.
- He, K., et al. (2016). Deep Residual Learning for Image Recognition. CVPR.
- He, K., et al. (2016). Identity mappings in deep residual networks. ECCV.
- Canziani, A., Paszke, A., & Culurciello, E. (2017). An analysis of deep neural network models for practical applications.
- Howard, A. G., et al. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861.
- Hu, J., Shen, L., Albanie, S., Sun, G., & Wu, E. (2018). Squeeze-and-Excitation Networks. CVPR 2018.
- Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017). Aggregated Residual Transformations for Deep Neural Networks. CVPR 2017.
- Huang et al, “Densely connected neural networks”, CVPR 2017

Next:Generative Adversarial Networks(GANs)