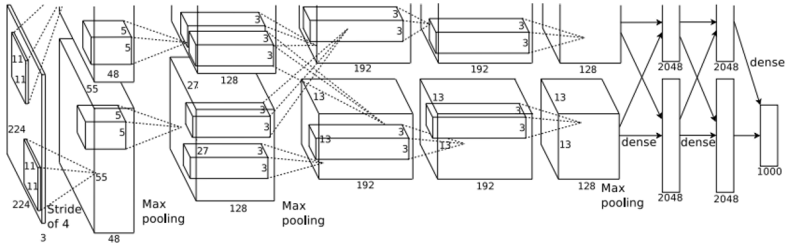
Network Architectures

Computer Vision (UW EE/CSE 576)



Richard Szeliski

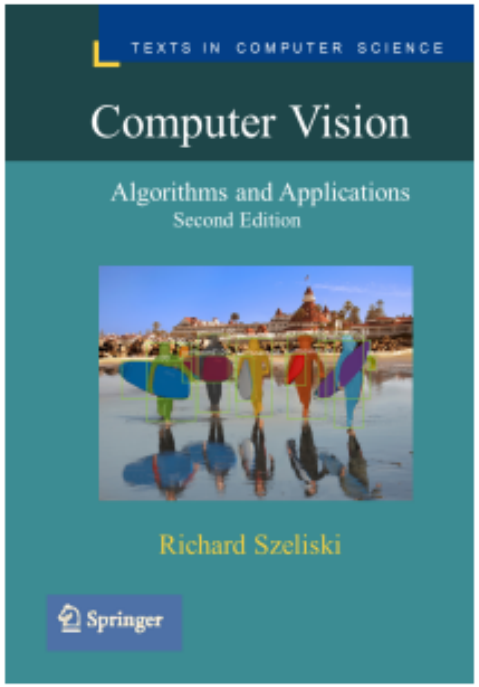
Facebook & UW

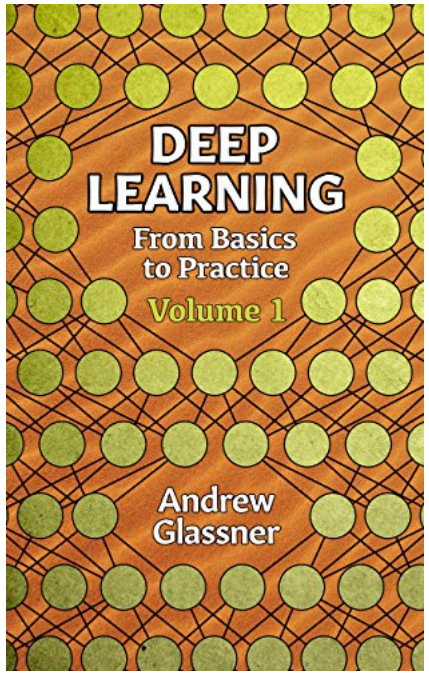
Lecture 9 – Apr 28, 2020

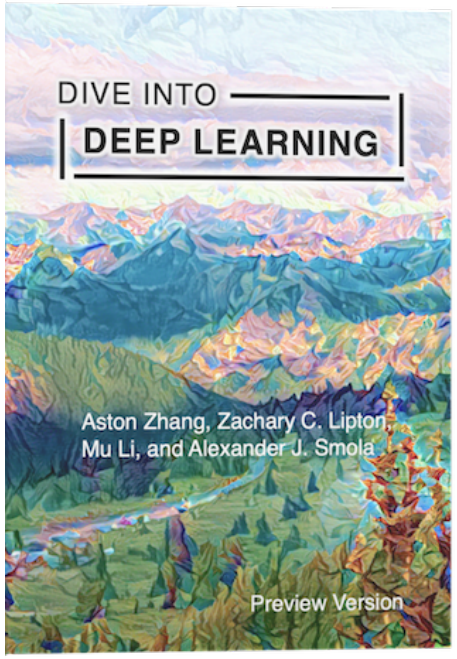
Class calendar

Date	Topic	Slides	Reading	Homework
April 9	Filters and convolutions	Google Slides	Szeliski, Chapter 3	HW1 due, HW2 assigned
April 14	Interpolation and Optimization	pdf, pptx	Szeliski, Chapter 4	
April 16	Machine Learning	pdf, pptx	Szeliski, Chapter 5.1-5.2	
April 21	Deep Neural Networks	pdf, pptx	Szeliski, Chapter 5.3	
April 23	Convolutional Neural Networks		Szeliski, Chapter 5.4	HW2 due, HW3 assigned
April 28	Network Architectures		Szeliski, Chapter 5.4	
April 30	Object Detection		Szeliski, Chapter 6.3	
May 5	Detection and Instance Segmentation		Szeliski, Chapter 6.4	
May 7	Edges, features, matching, RANSAC		Szeliski, Chapter 7.1-7.2, 8.1-8.2	HW3 due, HW4 assigned

References







https://d2l.ai/

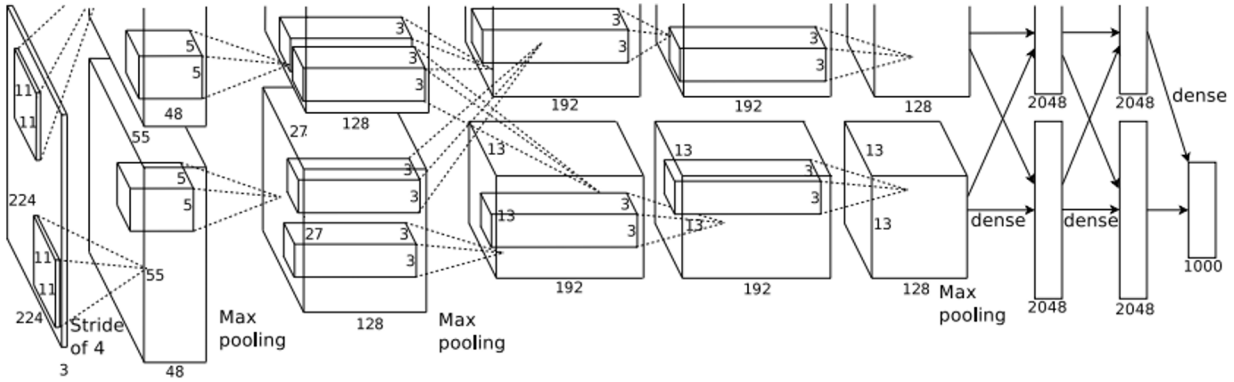
Readings

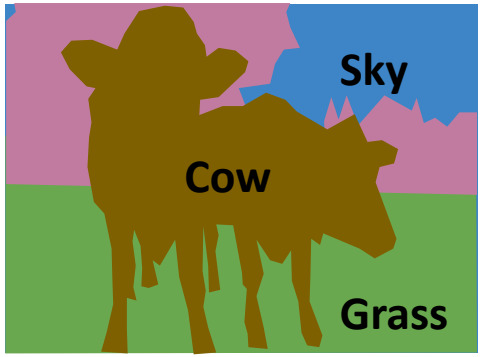
Chapter 5

Deep Learning

Network architectures

- Batch normalization
- CNN architectures
- Visualization
- Adversarial examples
- Deconvolution and U-Nets
- ~~Deep learning software~~





As before, I'm borrowing slides from

UNIVERSITY OF MICHIGAN

EECS 498-007 / 598-005
Deep Learning for Computer Vision
Fall 2019

Course Description

Computer Vision has become ubiquitous in our society, with applications in search, image understanding, apps, mapping, medicine, drones, and self-driving cars. Core to many of these applications are visual recognition tasks such as image classification and object detection. Recent developments in neural network approaches have greatly advanced the performance of these state-of-the-art visual recognition systems. This course is a deep dive into details of neural-network based deep learning methods for computer vision. During this course, students will learn to implement, train and debug their own neural networks and gain a detailed understanding of cutting-edge research in computer vision. We will cover learning algorithms, neural network architectures, and practical engineering tricks for training and fine-tuning networks for visual recognition tasks.

Instructor

Justin Johnson

Graduate Student Instructors

Yunseok Jang

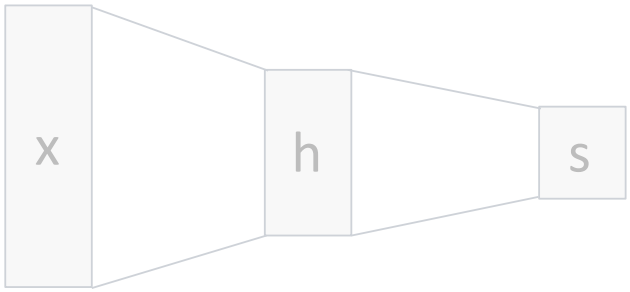
Kibok Lee

Luowei Zhou

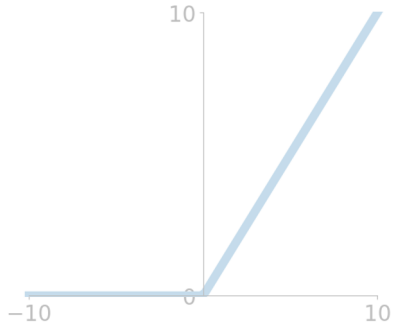
Lecture 7: Convolutional Networks

Components of a Convolutional Network

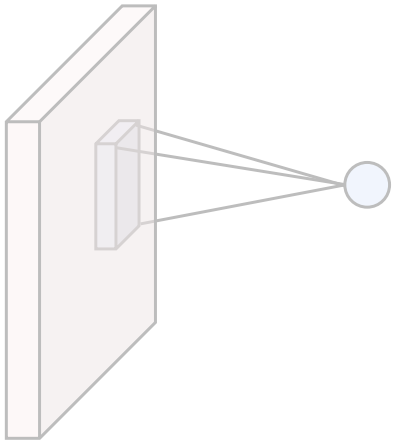
Fully-Connected Layers



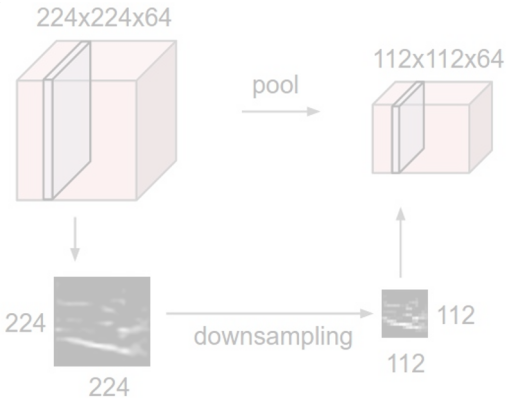
Activation Function



Convolution Layers



Pooling Layers



Normalization

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

Batch Normalization

Idea: “Normalize” the outputs of a layer so they have zero mean and unit variance. **Why?**

Why? Helps reduce “internal covariate shift”, improves optimization

We can normalize a batch of activations like this:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

This is a **differentiable function**, so we can use it as an operator in our networks and backprop through it!

Ioffe and Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift”, ICML 2015

Activation and weight scaling

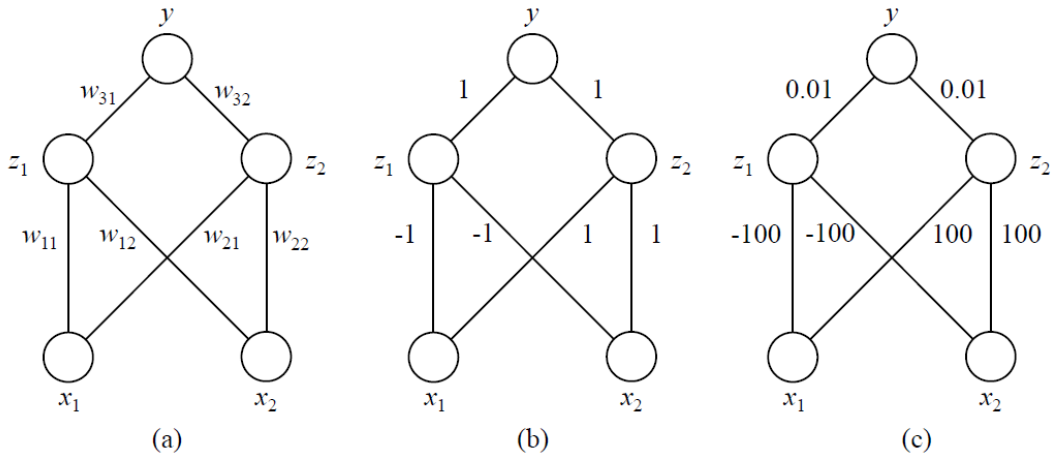


Figure 5.54 *Simple two hidden unit network with a ReLU activation function and no bias parameters for regressing the function $y = |x_1 + 1.1x_2|$: (a) can you guess a set of weights would fit this function? (b) a reasonable set of starting weights; (c) a poorly scaled set of weights.*

Ex 5.1: Activation and weight scaling Consider the two hidden unit network shown in Figure 5.54 which uses ReLU activation functions and has no additive bias parameters. Your task is to find a set of weights that will fit the function

$$y = |x_1 + 1.1x_2|. \tag{5.91}$$

1. Can you guess a set of weights that will fit this function?
2. Starting with the weights shown in column b, compute the activations for the hidden and final units as well as the regression loss for the nine input values $(x_1, x_2) \in \{-1, 0, 1\} \times \{-1, 0, 1\}$.
3. Now compute the gradients of the squared loss with respect to all six weights using the backpropagation chain rule equations (5.79–5.82) and sum them up across the training samples to get a final gradient.
4. What step size should you take in the gradient direction, and what would your update squared loss become?
5. Repeat this exercise for the initial weights in column (c) of Figure 5.54.
6. Given this new set of weights, how much worse is your error decrease, and how many iterations would you expect it to take to achieve a reasonalbe solution?
7. Would batch normalization help in this case?

Activation and weight scaling

x1	x2	w11	w12	w21	w22	z1	z2	w31	w32	y	target	L2 loss
-1	-1	-1	-1	1	1	2	0	1	1	2	2.1	0.01
-1	0					1	0			1	1	0
-1	1					0	0			0	0.1	0.01
0	-1					1	0			1	1.1	0.01
0	0					0	0			0	0	0
0	1					0	1			1	1.1	0.01
1	-1					0	0			0	0.1	0.01
1	0					0	1			1	1	0
1	1					0	2			2	2.1	0.01
												0.06

x1	x2	d_w11	d_w12	d_w21	d_w22	d_z1	d_z2	d_w31	d_w32	g_y
-1	-1	0.1	0.1	0	0	-0.1	-0.1	-0.2	0	-0.1
-1	0	0	0	0	0	0	0	0	0	0
-1	1	0	0	0	0	0	0	0	0	-0.1
0	-1	0	0.1	0	0	-0.1	-0.1	-0.1	0	-0.1
0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	-0.1	-0.1	-0.1	0	-0.1	-0.1
1	-1	0	0	0	0	0	0	0	0	-0.1
1	0	0	0	0	0	0	0	0	0	0
1	1	0	0	-0.1	-0.1	-0.1	-0.1	0	-0.2	-0.1
grad-:		-0.1	-0.2	0.1	0.2			0.3	0.3	

Balanced | Imbalanced

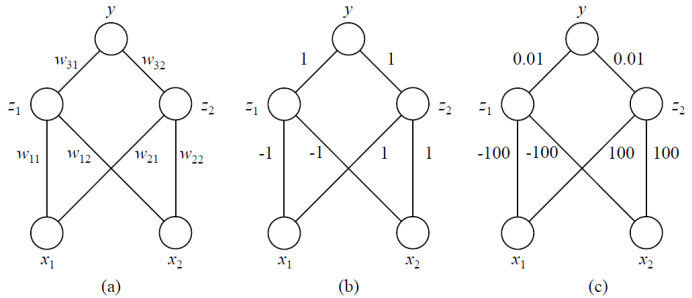
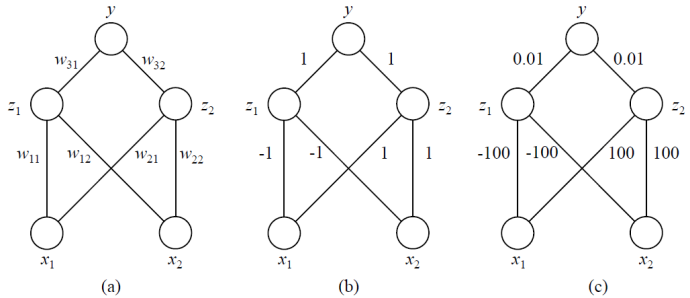


Figure 5.54 *Simple two hidden unit network with a ReLU activation function and no bias parameters for regressing the function $y = |x_1 + 1.1x_2|$: (a) can you guess a set of weights would fit this function? (b) a reasonable set of starting weights; (c) a poorly scaled set of weights.*

Activation and weight scaling



x1	x2	w11	w12	w21	w22	z1	z2	w31	w32	y	target	L2 loss
-1	-1	-1	-1	1	1	2	0	1	1	2	2.1	0.01
-1	0											
-1	1											
0	-1											
0	0											
0	1											
1	-1											
1	0											
1	1											

x1	x2	d_w11	d_w12	d_w21	d_w22	d_z1
-1	-1	0.1	0.1	0	0	
-1	0	0	0	0	0	
-1	1	0	0	0	0	
0	-1	0	0.1	0	0	
0	0	0	0	0	0	
0	1	0	0	0	-0.1	
1	-1	0	0	0	0	
1	0	0	0	0	0	
1	1	0	0	-0.1	-0.1	
grad-:		-0.1	-0.2	0.1	0.2	

Balanced | Imbalanced

x1	x2	w11	w12	w21	w22	z1	z2	w31	w32	y	target	L2 loss
-1	-1	-100	-100	100	100	200	0	0.01	0.01	2	2.1	0.01
-1	0					100	0			1	1	0
-1	1					0	0			0	0.1	0.01
0	-1					100	0			1	1.1	0.01
0	0					0	0			0	0	0
0	1					0	100			1	1.1	0.01
1	-1					0	0			0	0.1	0.01
1	0					0	100			1	1	0
1	1					0	200			2	2.1	0.01
												0.06

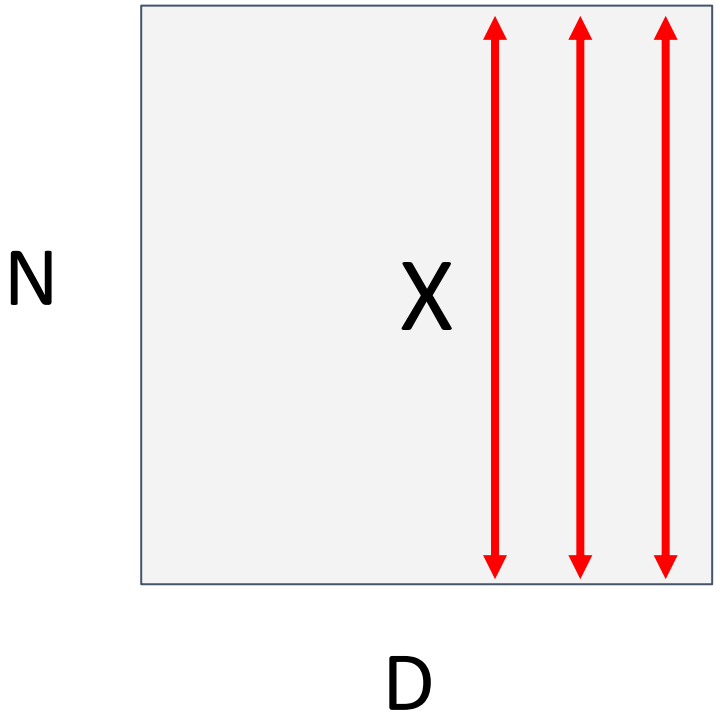
x1	x2	d_w11	d_w12	d_w21	d_w22	d_z1	d_z2	d_w31	d_w32	g_y
-1	-1	0.001	0.001	0	0	-0.001	-0.001	-20	0	-0.1
-1	0	0	0	0	0	0	0	0	0	0
-1	1	0	0	0	0	0	0	0	0	-0.1
0	-1	0	0.001	0	0	-0.001	-0.001	-10	0	-0.1
0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	-0.001	-0.001	-0.001	0	-10	-0.1
1	-1	0	0	0	0	0	0	0	0	-0.1
1	0	0	0	0	0	0	0	0	0	0
1	1	0	0	-0.001	-0.001	-0.001	-0.001	0	-20	-0.1
grad-:		-0.001	-0.002	0.001	0.002			30	30	

Balanced | Imbalanced

und no bias
of weights
caled set of

Batch Normalization

Input: $x : N \times D$



$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

Per-channel std, shape is D

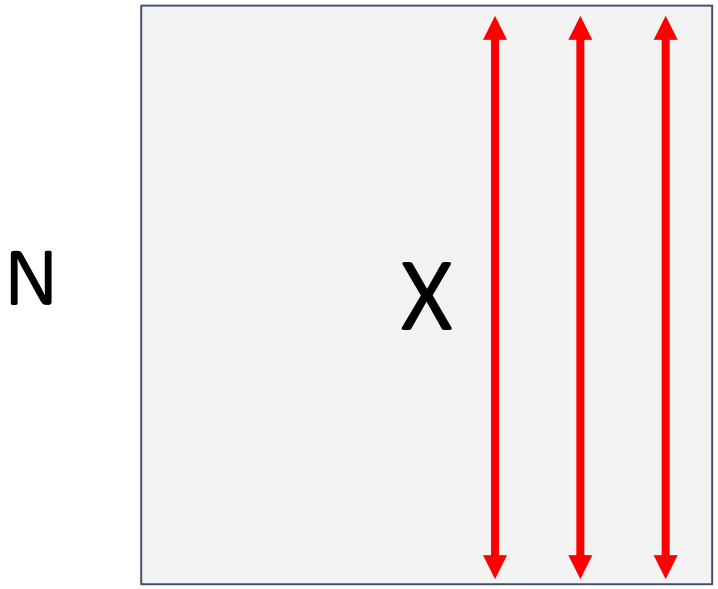
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Batch Normalization

Input: $x : N \times D$



$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

Per-channel std, shape is D

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

Normalized x, Shape is N x D

Problem: What if zero-mean, unit variance is too hard of a constraint?

Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Batch Normalization

Input: $x : N \times D$

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function!

$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

Per-channel std, shape is D

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

Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

Batch Normalization: Test-Time

Problem: Estimates depend on minibatch; can't do this at test-time

Input: $x : N \times D$

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function!

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

Per-channel std, shape is D

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

Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

Batch Normalization: Test-Time

Input: $x : N \times D$

Learnable scale and shift parameters:

$\gamma, \beta : D$

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function!

$$\mu_j = \text{(Running) average of values seen during training}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \text{(Running) average of values seen during training}$$

Per-channel std, shape is D

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

Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

Batch Normalization: Test-Time

Input: $x : N \times D$

Learnable scale and shift parameters:

$\gamma, \beta : D$

During testing batchnorm becomes a linear operator.
Can be fused with the previous fully-connected or conv layer

$$\mu_j = \text{(Running) average of values seen during training}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \text{(Running) average of values seen during training}$$

Per-channel std, shape is D

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

Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

Batch Normalization for ConvNets

Batch Normalization for
fully-connected networks

$x: N \times D$

Normalize

$\mu, \sigma: 1 \times D$

$\gamma, \beta: 1 \times D$

$y = \gamma(x-\mu)/\sigma+\beta$

Batch Normalization for
convolutional networks
(Spatial Batchnorm, BatchNorm2D)

$x: N \times C \times H \times W$

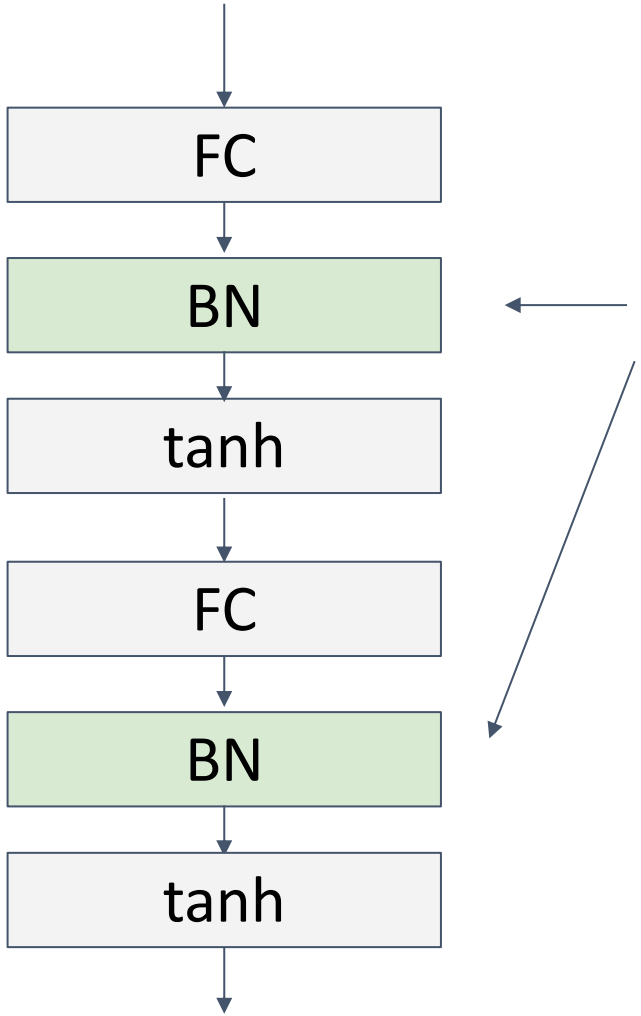
Normalize

$\mu, \sigma: 1 \times C \times 1 \times 1$

$\gamma, \beta: 1 \times C \times 1 \times 1$

$y = \gamma(x-\mu)/\sigma+\beta$

Batch Normalization

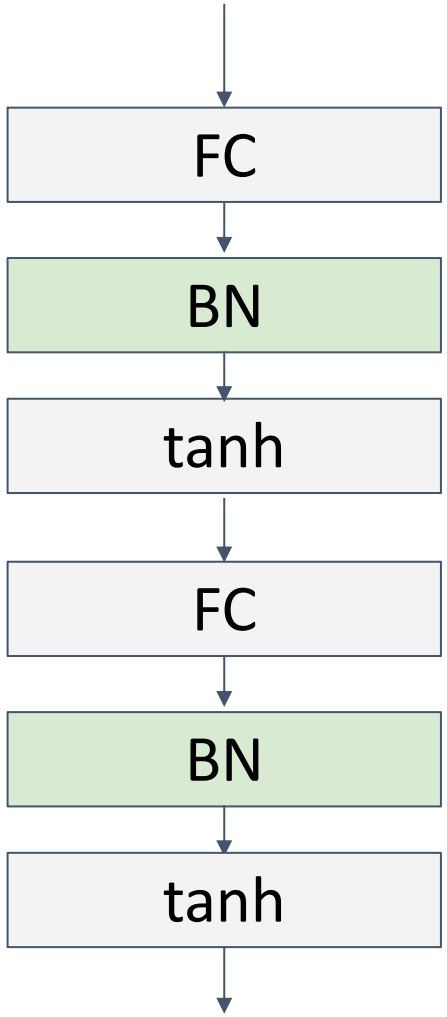


Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

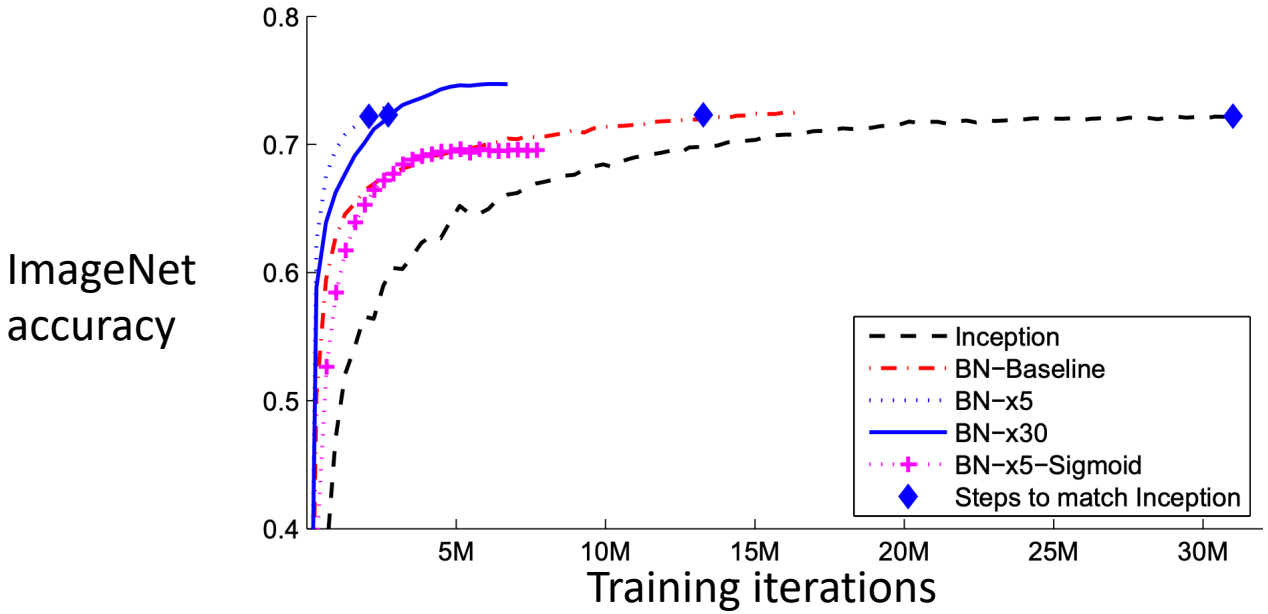
$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Batch Normalization

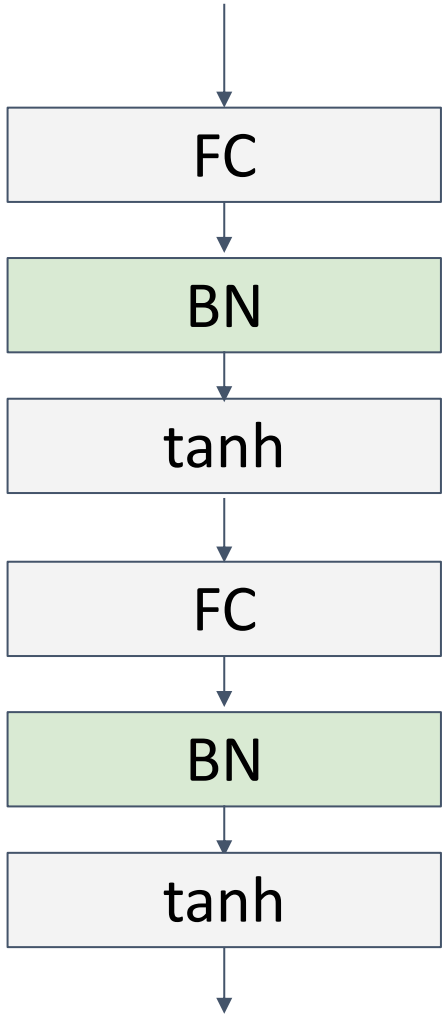


- Makes deep networks **much** easier to train!
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!



Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

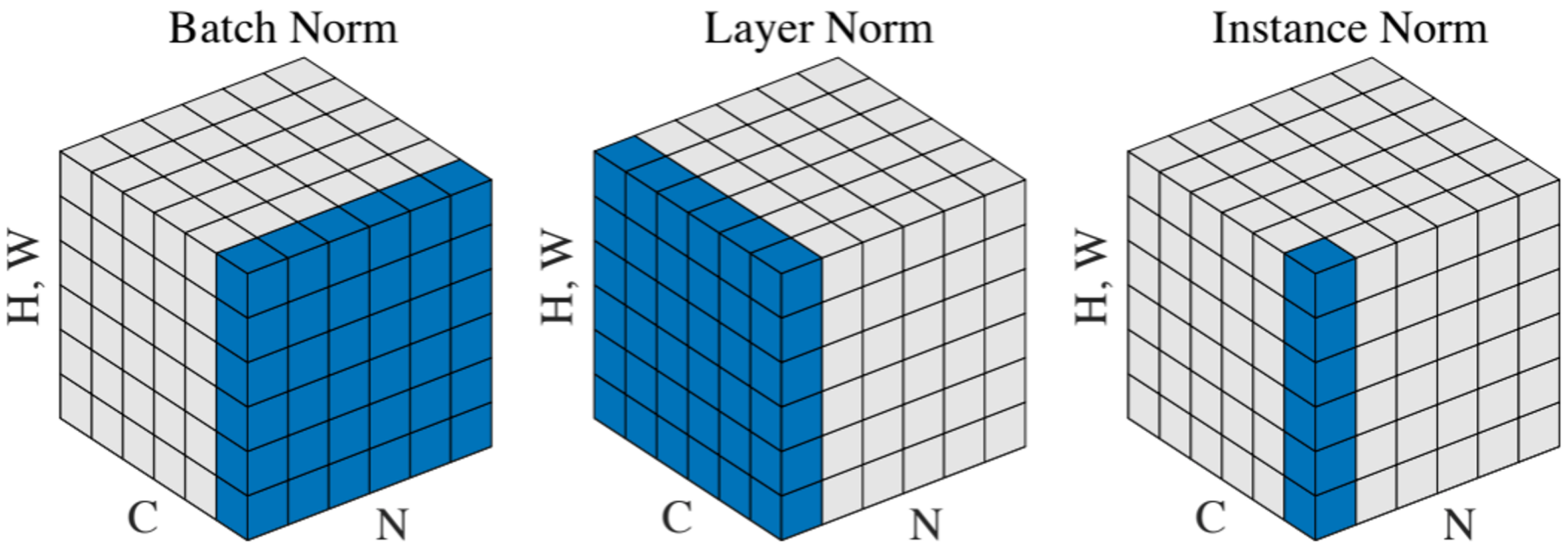
Batch Normalization



- Makes deep networks **much** easier to train!
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!
- Not well-understood theoretically (yet)
- Behaves differently during training and testing: this is a very common source of bugs!

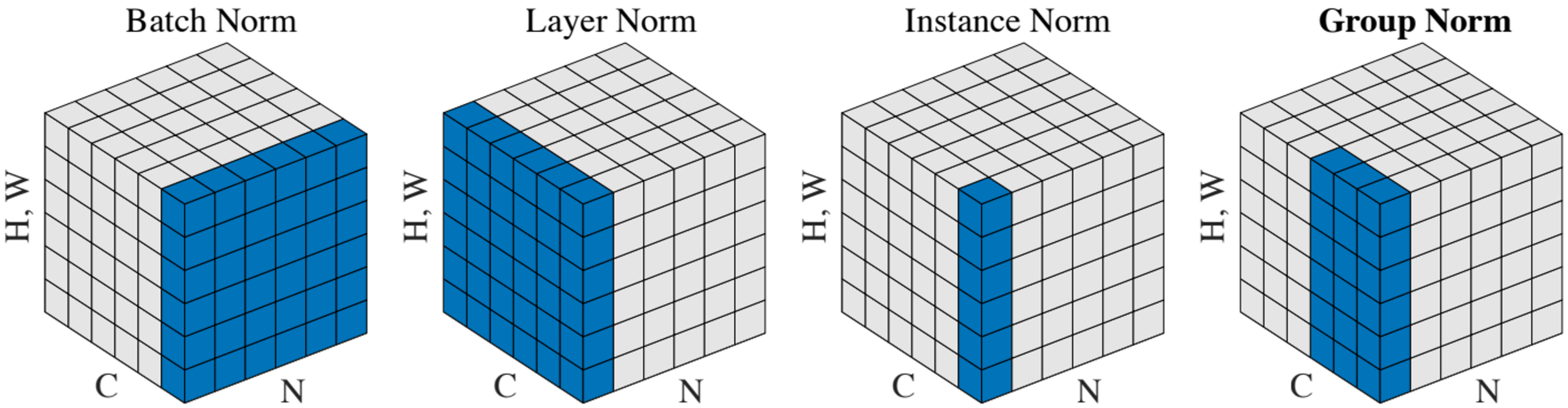
Ioffe and Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", ICML 2015

Comparison of Normalization Layers



Wu and He, “Group Normalization”, ECCV 2018

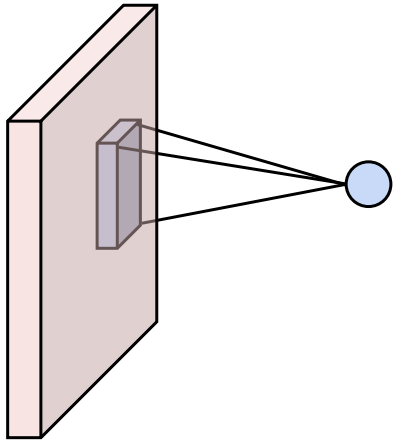
Group Normalization



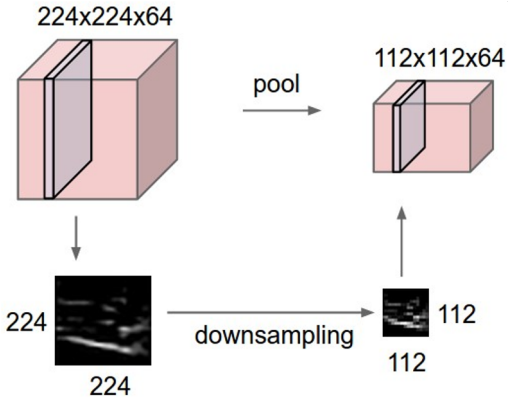
Wu and He, “Group Normalization”, ECCV 2018

Components of a Convolutional Network

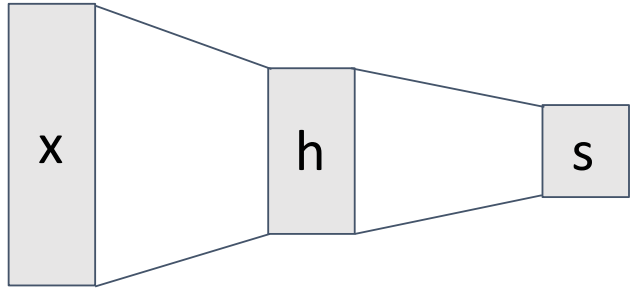
Convolution Layers



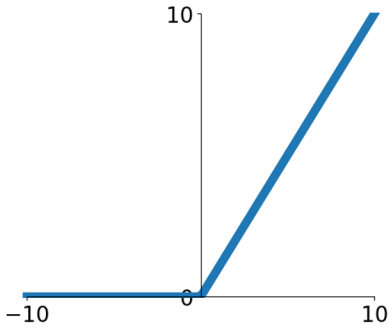
Pooling Layers



Fully-Connected Layers



Activation Function

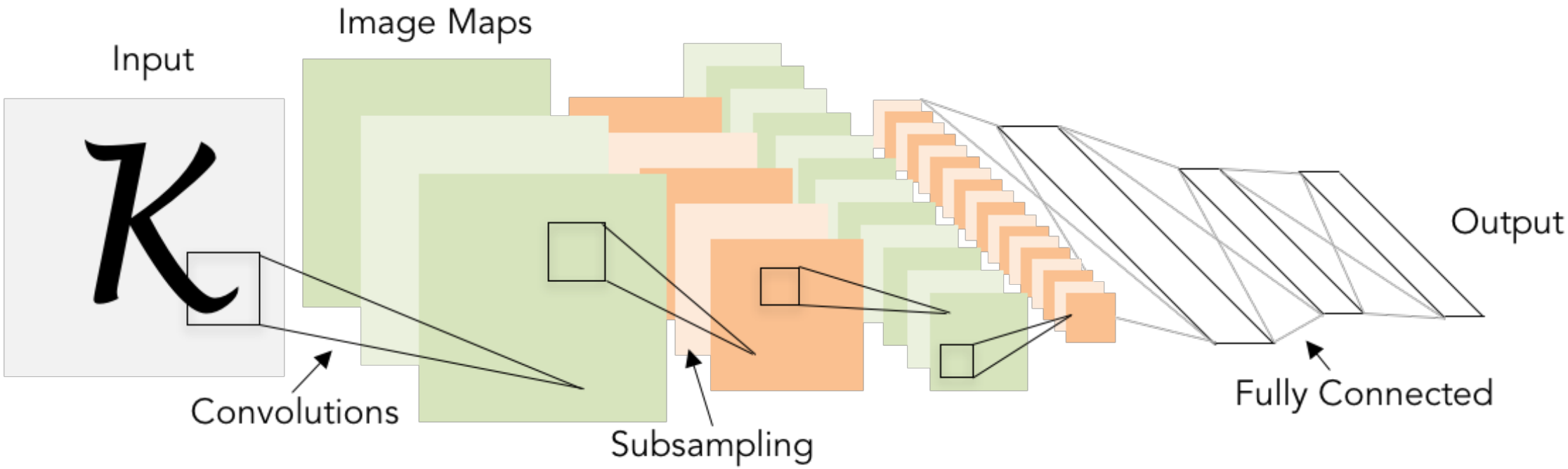


Normalization

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

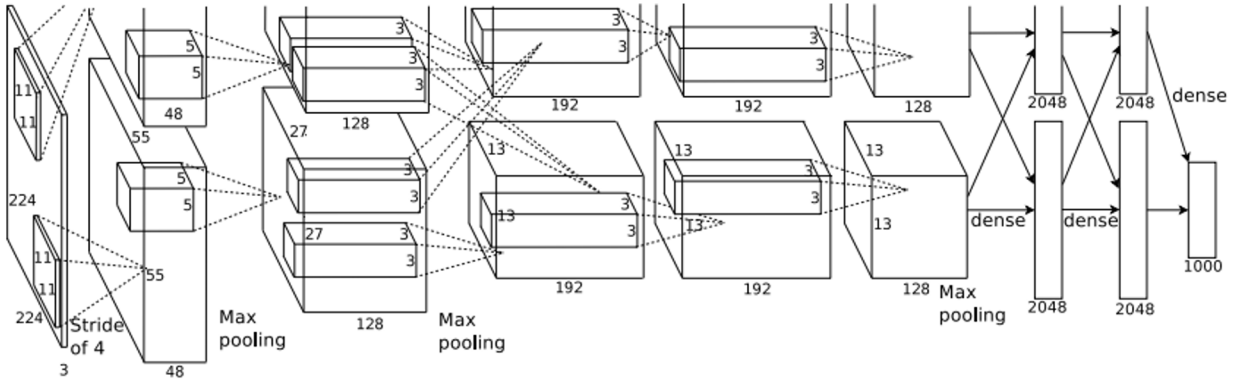
Components of a Convolutional Network

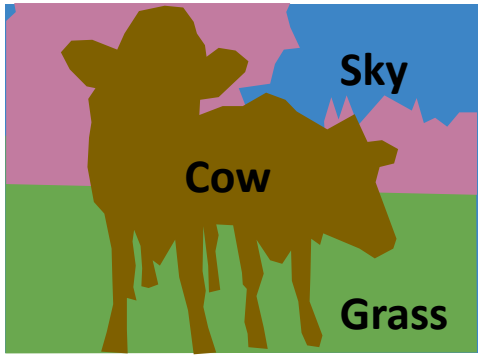
Problem: What is the right way to combine all these components?



Network architectures

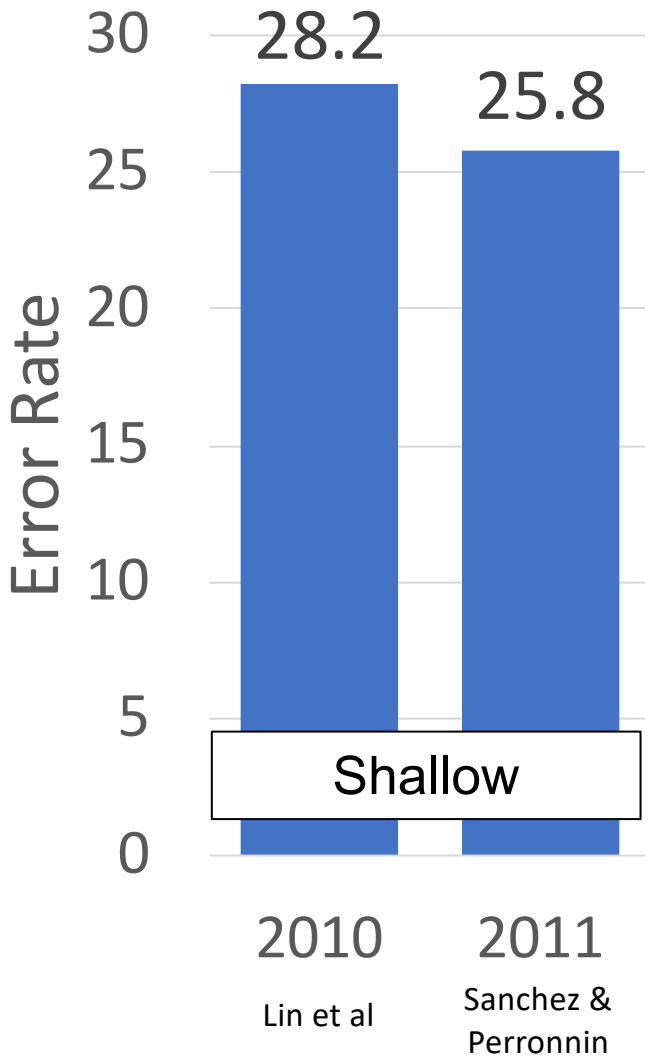
- Batch normalization
- **CNN architectures**
- Visualization
- Adversarial examples
- Deconvolution and U-Nets
- ~~Deep learning software~~



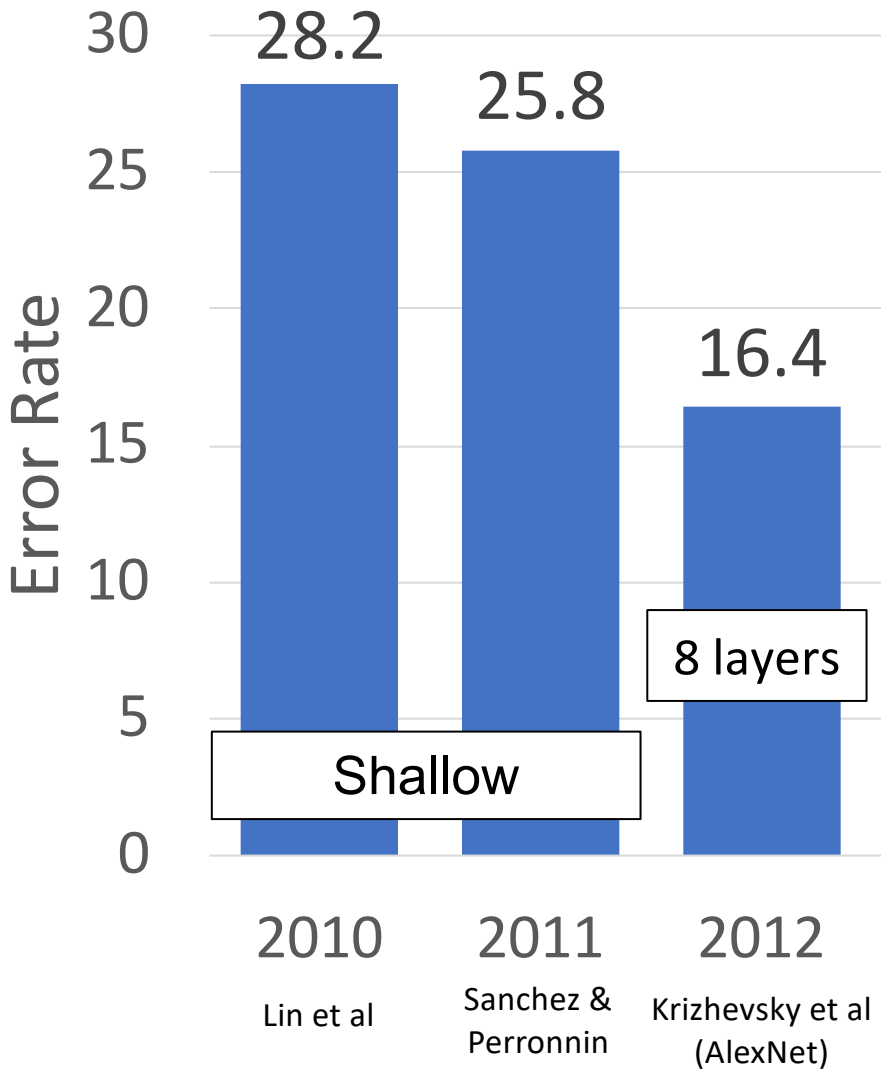


Lecture 8:
CNN Architectures

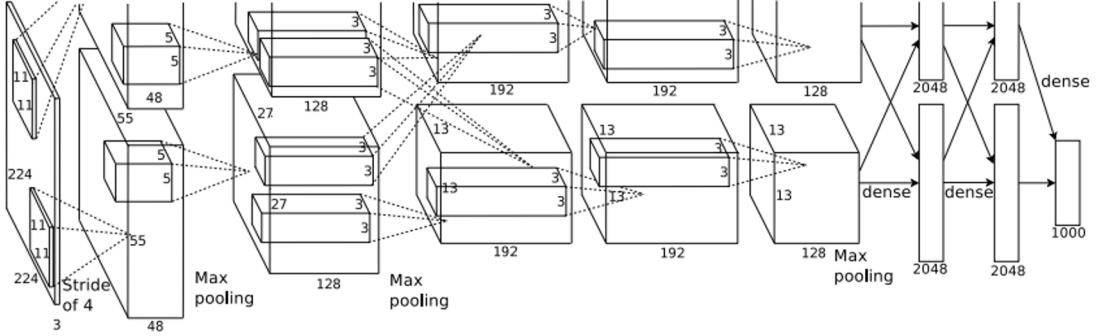
ImageNet Classification Challenge



ImageNet Classification Challenge

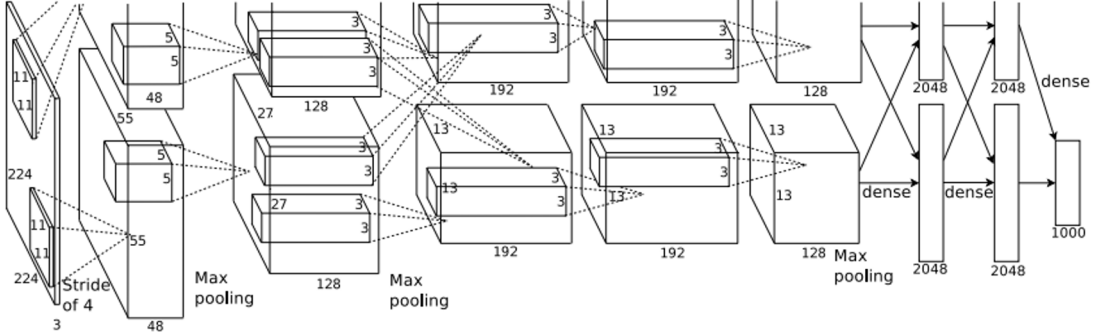


AlexNet



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

AlexNet



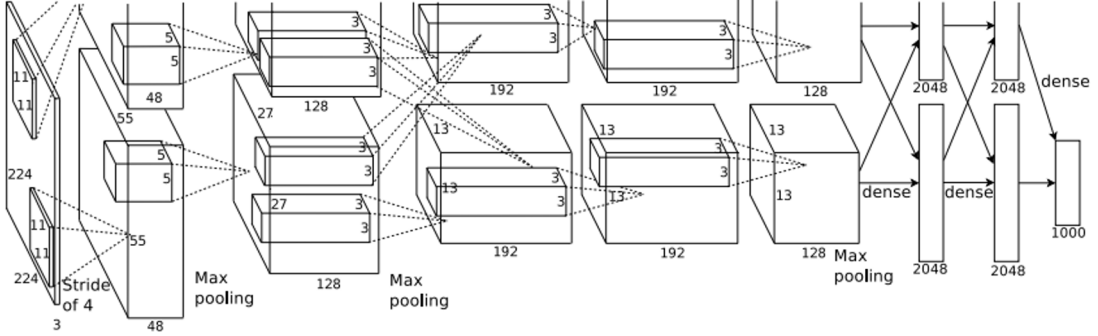
227 x 227 inputs
5 Convolutional layers
Max pooling
3 fully-connected layers
ReLU nonlinearities

Used “Local response normalization”; Not used anymore

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

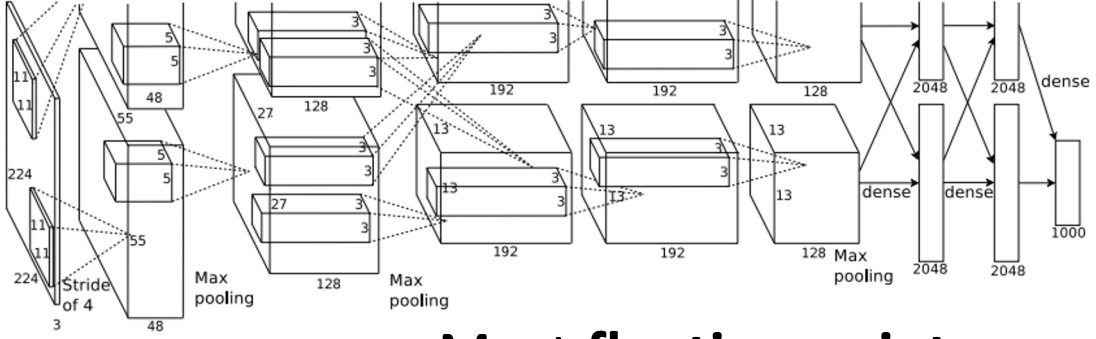
AlexNet

Interesting trends here!



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

AlexNet

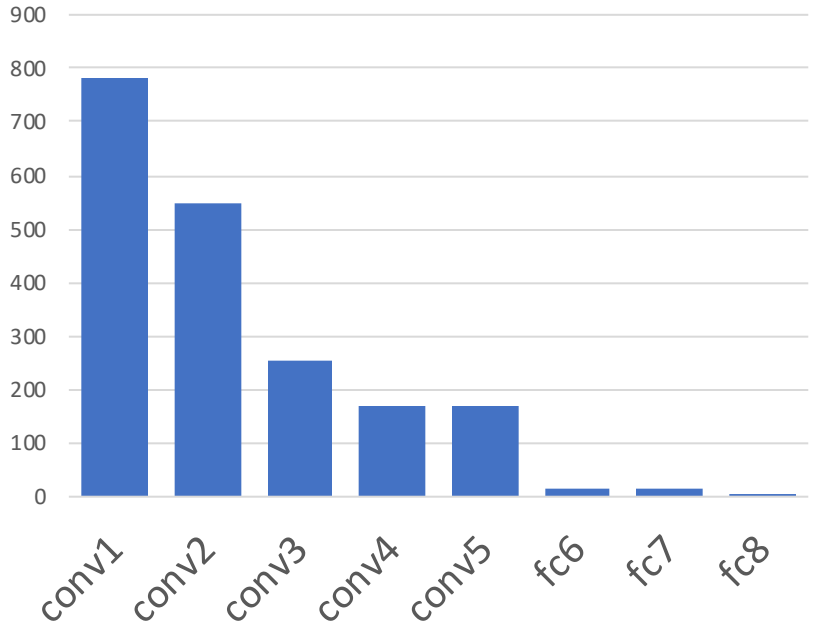


Most of the **memory usage** is in the early convolution layers

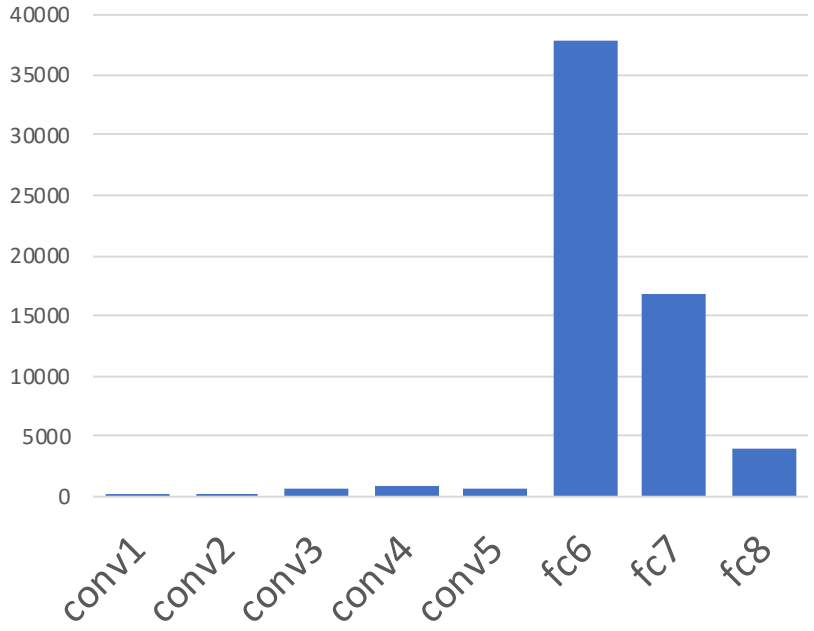
Nearly all **parameters** are in the fully-connected layers

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

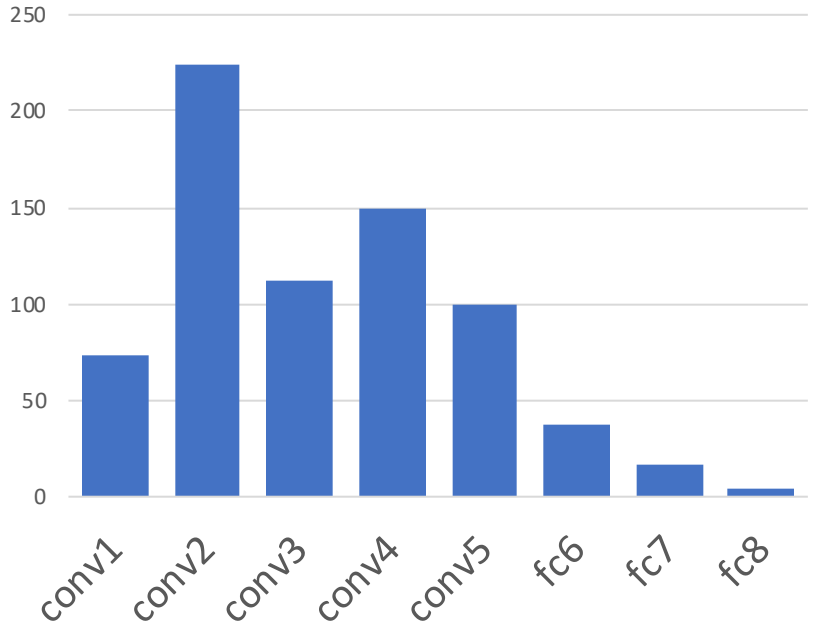
Memory (KB)



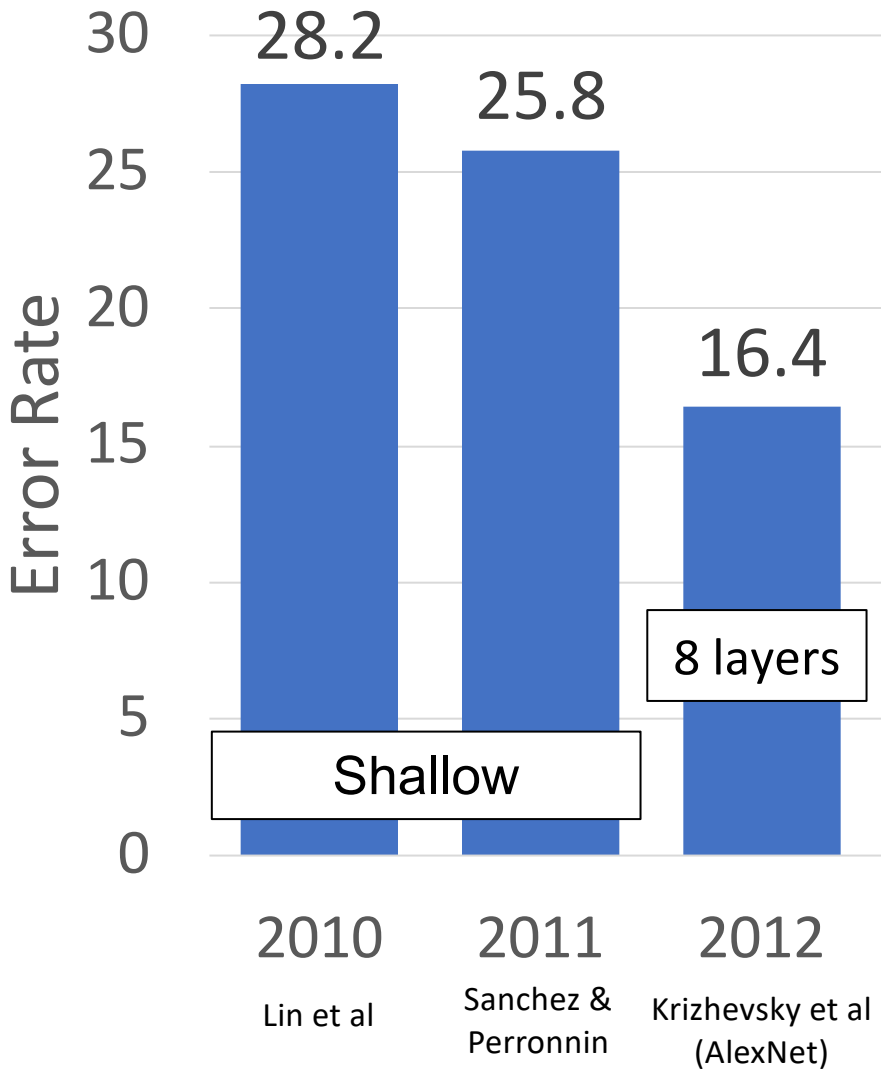
Params (K)



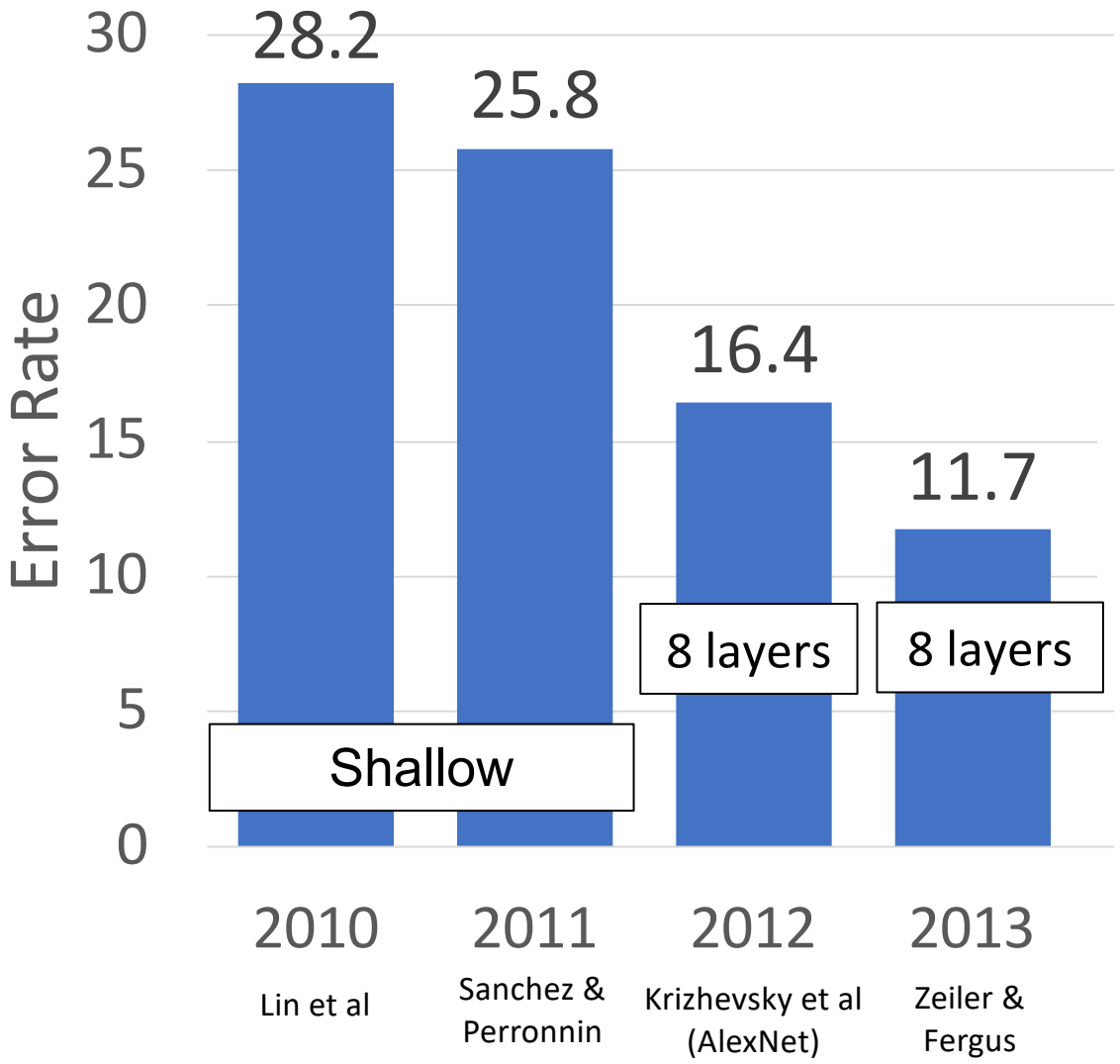
MFLOP



ImageNet Classification Challenge

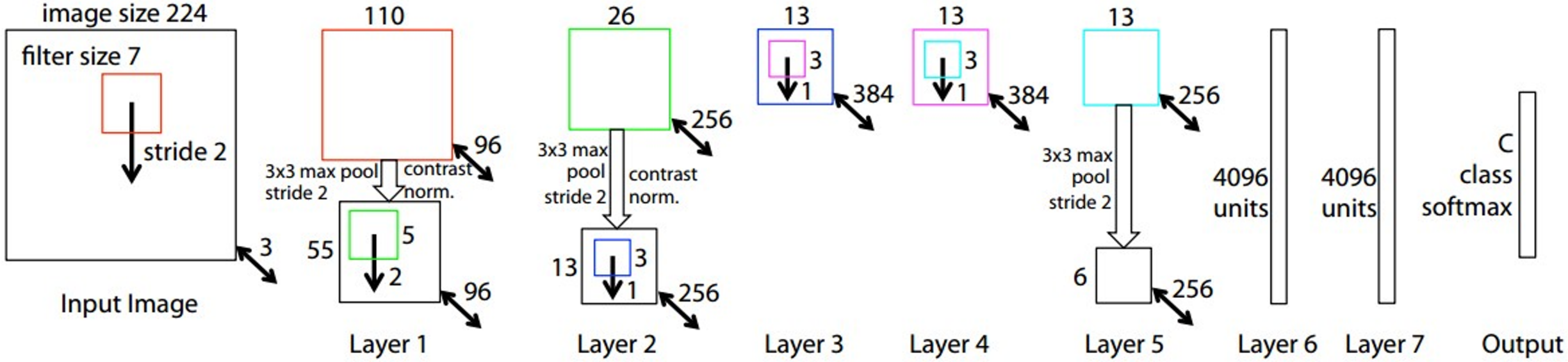


ImageNet Classification Challenge



ZFNet: A Bigger AlexNet

ImageNet top 5 error: 16.4% -> 11.7%



AlexNet but:

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

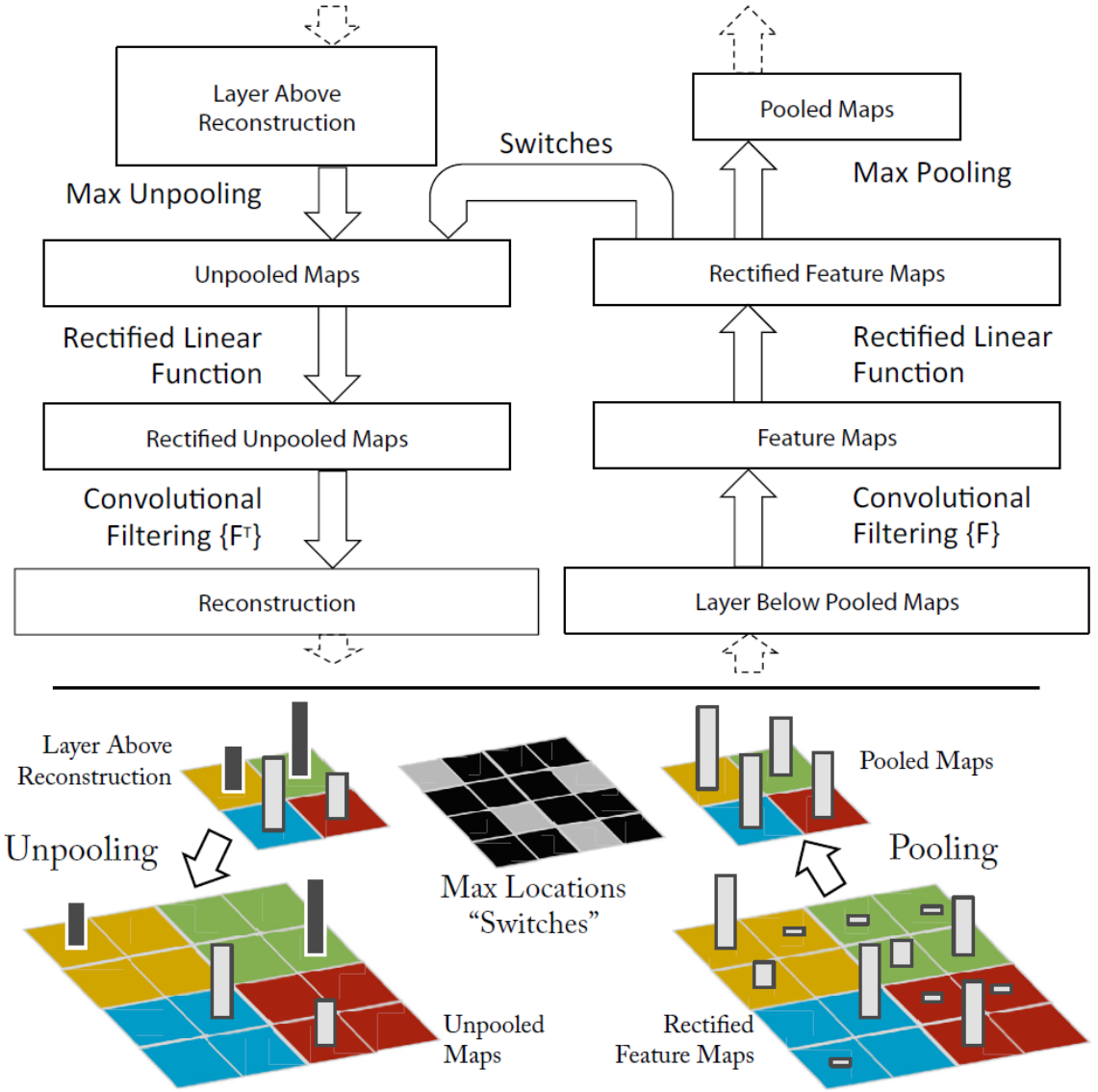
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

More trial and error =(

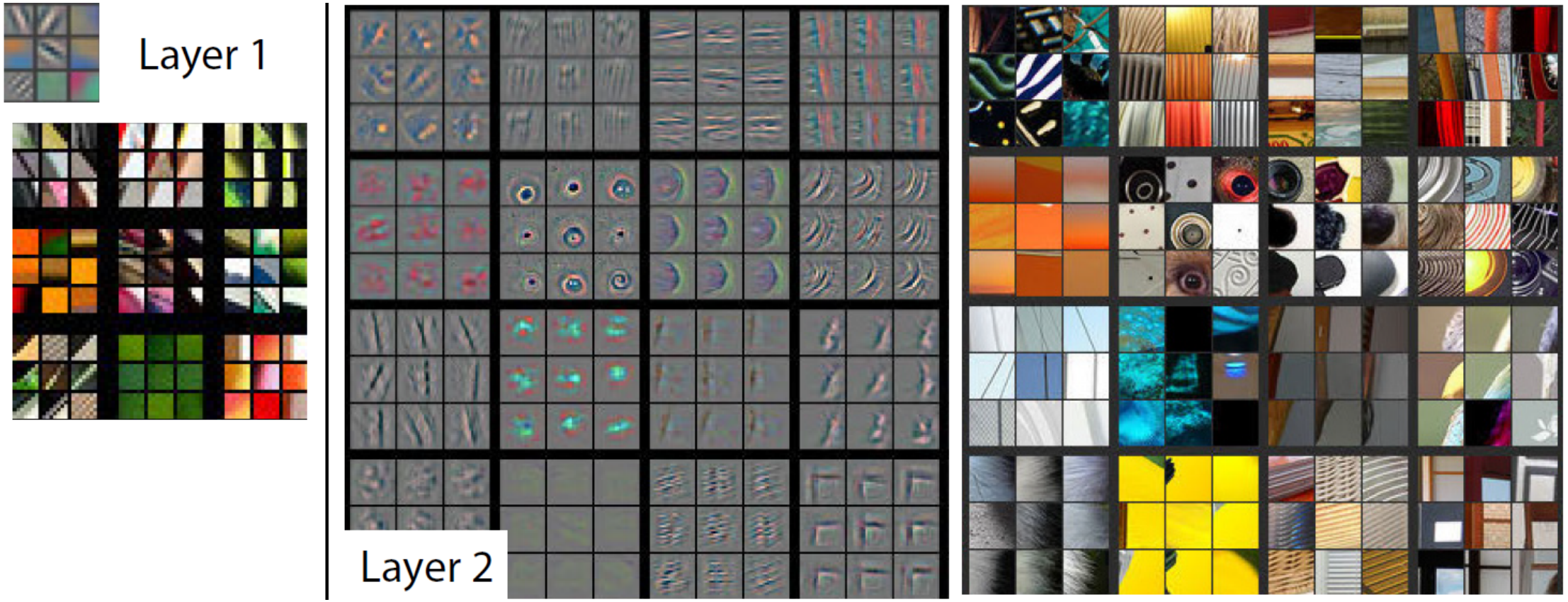
Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014

The really cool part

- Visualizing features with backprojection (``deconvnet'')
- How could you do this?
- What about max pooling?



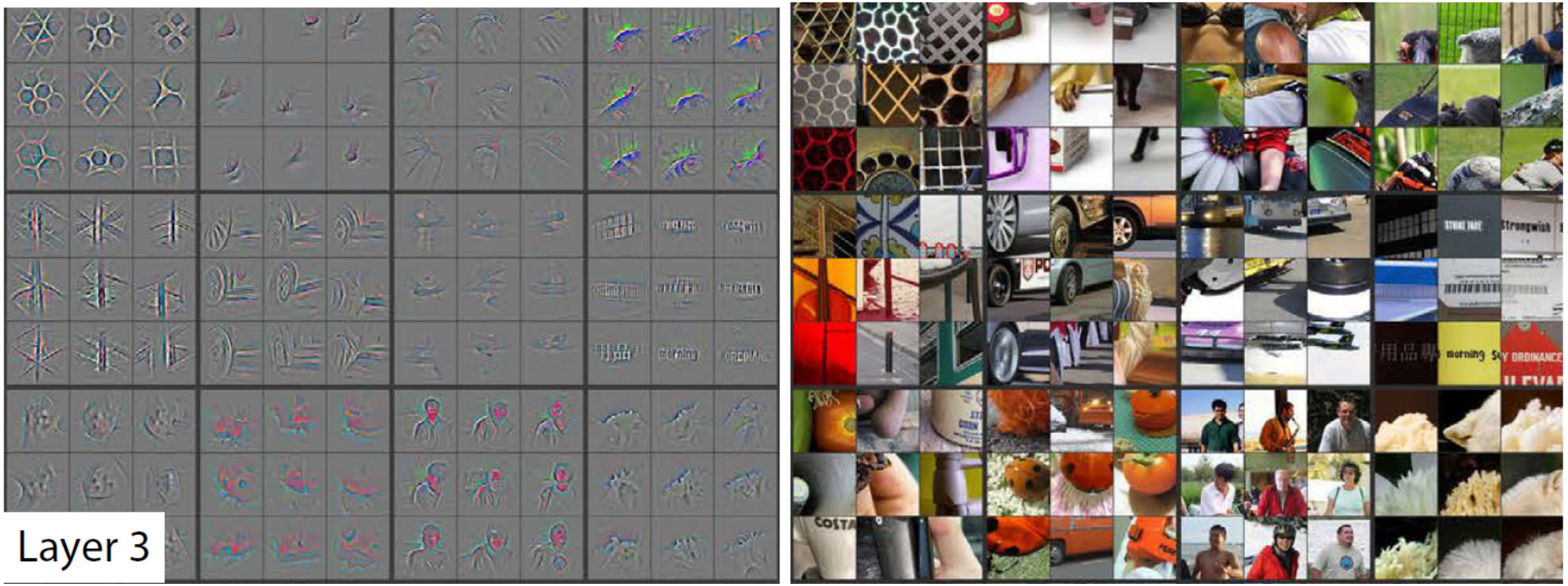
Visualizing features through backprojection



Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

gray: max stimulus, color: patch

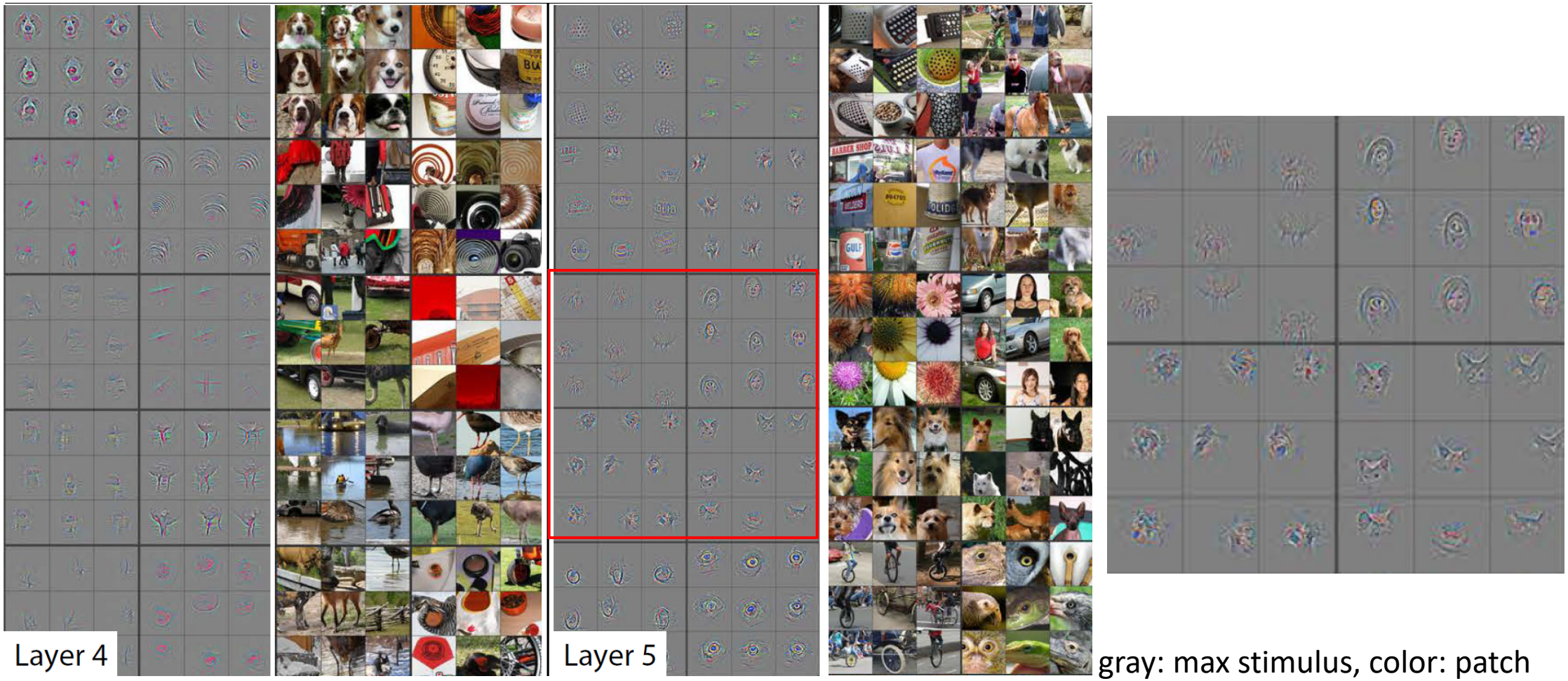
Visualizing features through backprojection



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

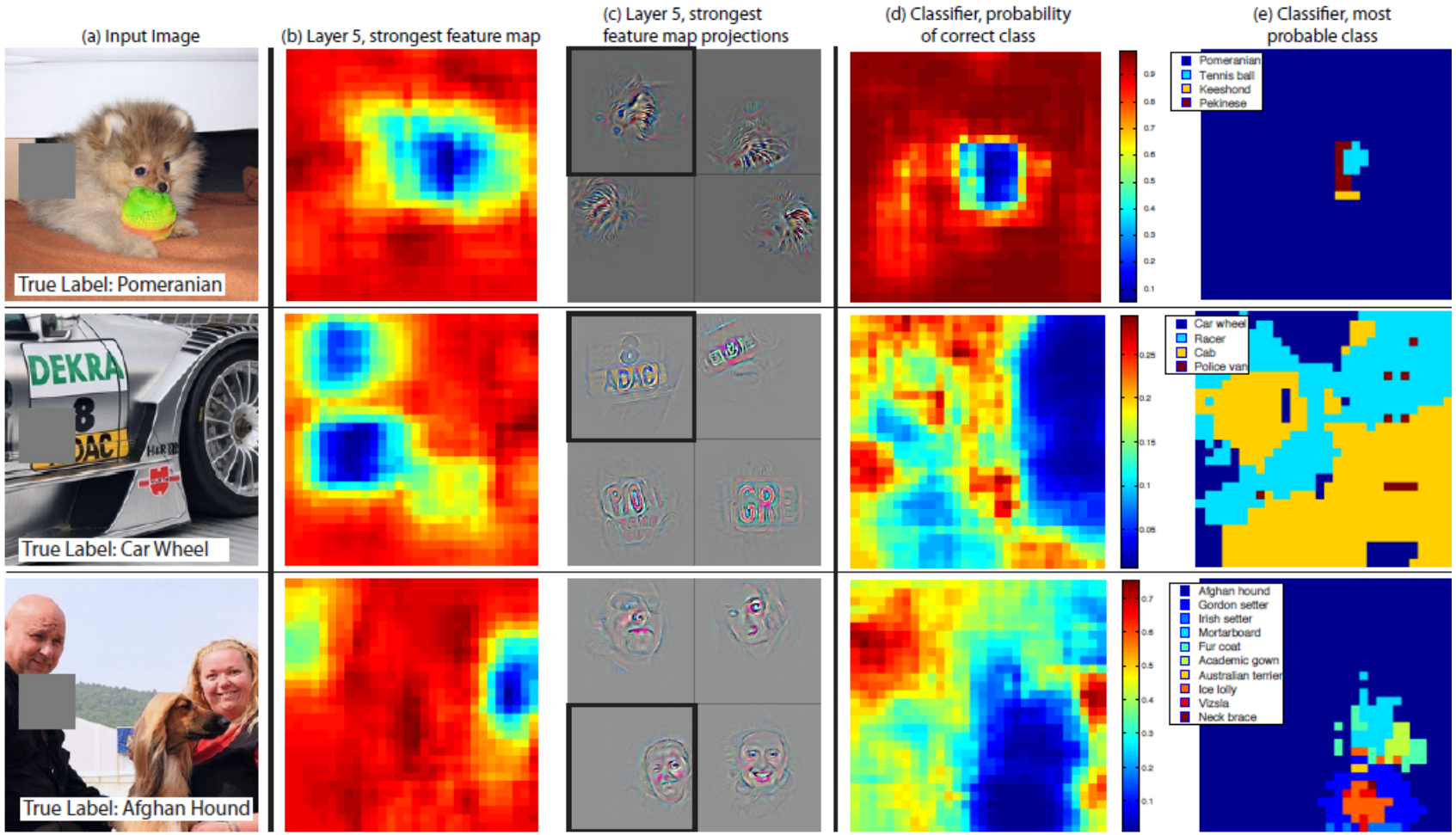
gray: max stimulus, color: patch

Visualizing features through backprojection

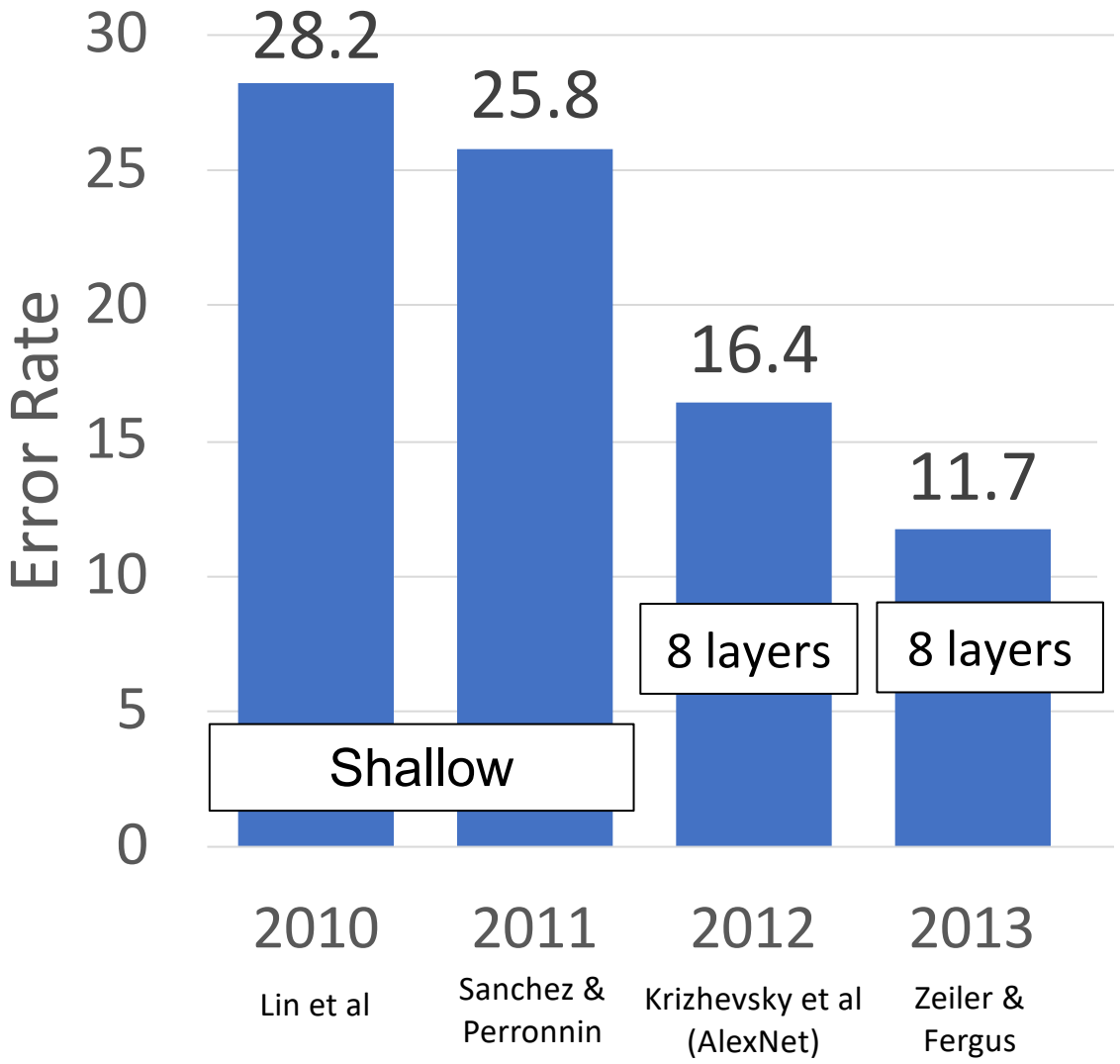


gray: max stimulus, color: patch

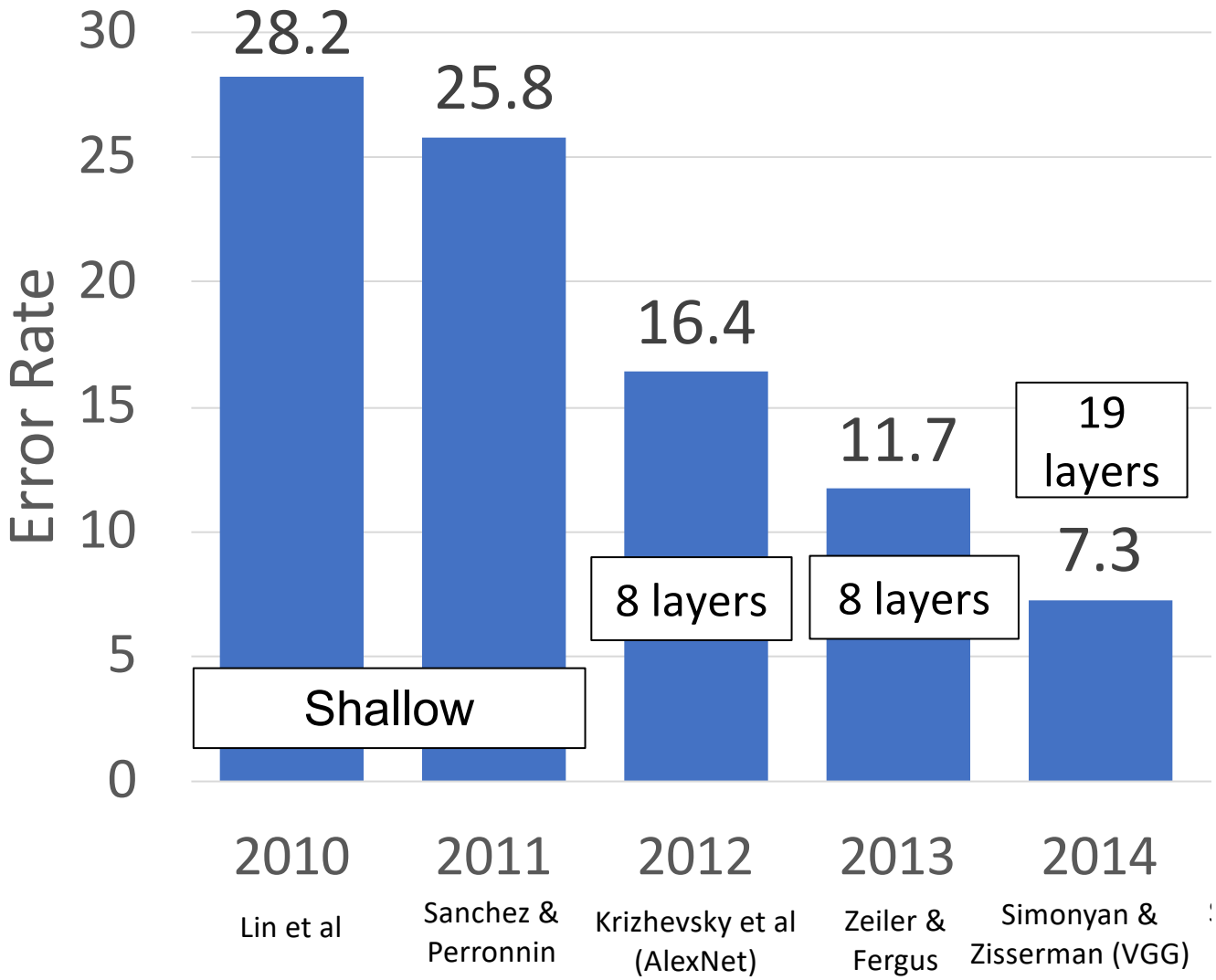
Attribution: saliency via occlusions



ImageNet Classification Challenge



ImageNet Classification Challenge



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

Network has 5 convolutional **stages**:

Stage 1: conv-conv-pool

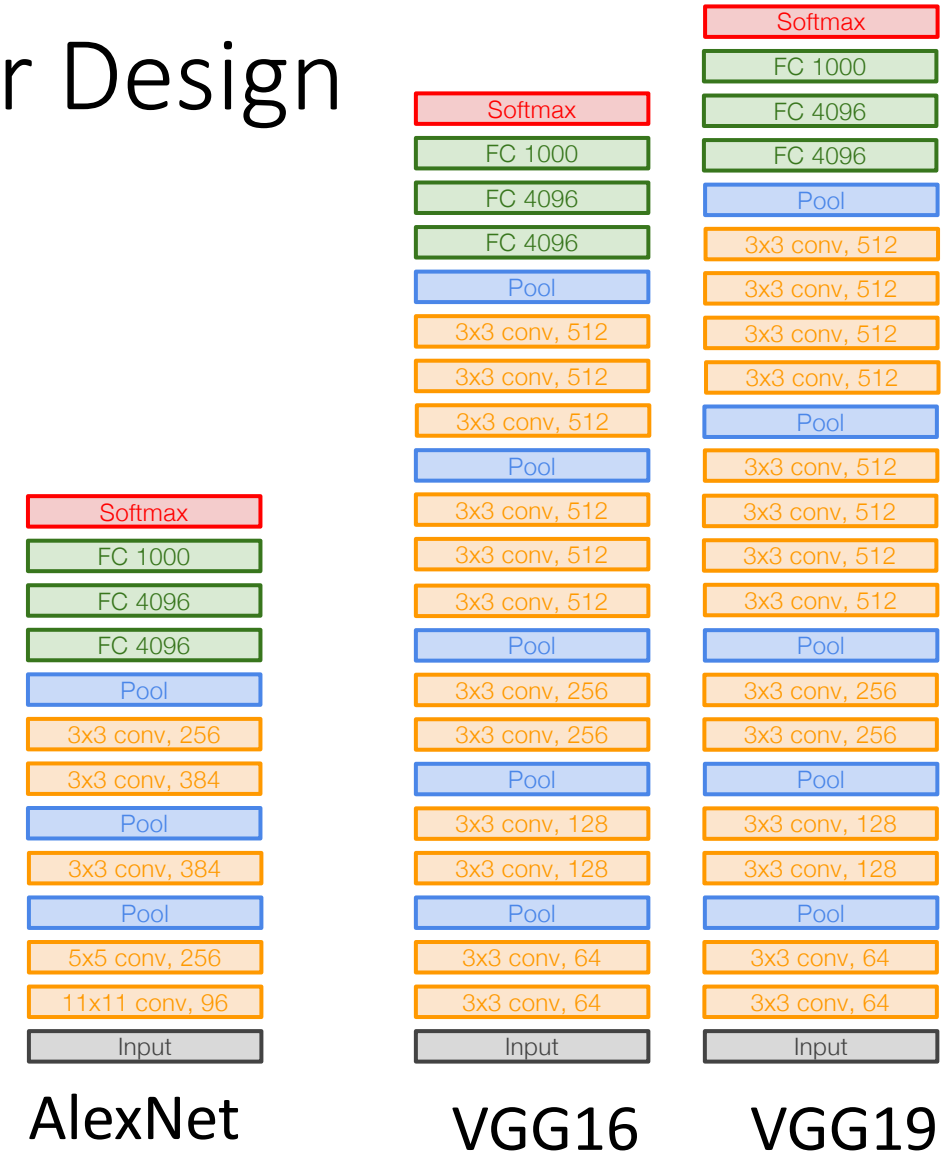
Stage 2: conv-conv-pool

Stage 3: conv-conv-pool

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

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

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



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

VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

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

<u>Option 1:</u>

Conv(5x5, C -> C)

Params: $25C^2$

FLOPs: $25C^2HW$

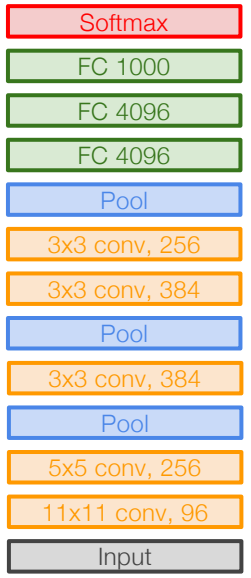
<u>Option 2:</u>

Conv(3x3, C -> C)

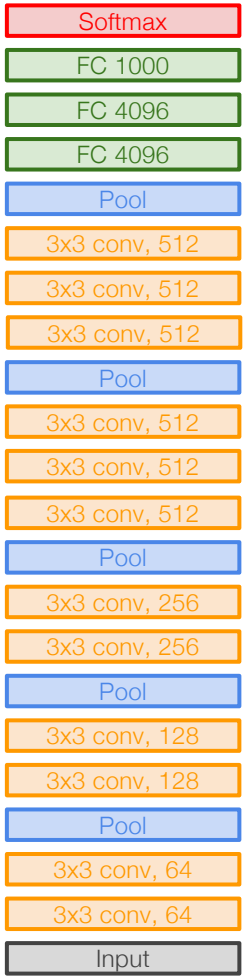
Conv(3x3, C -> C)

Params: $18C^2$

FLOPs: $18C^2HW$



AlexNet

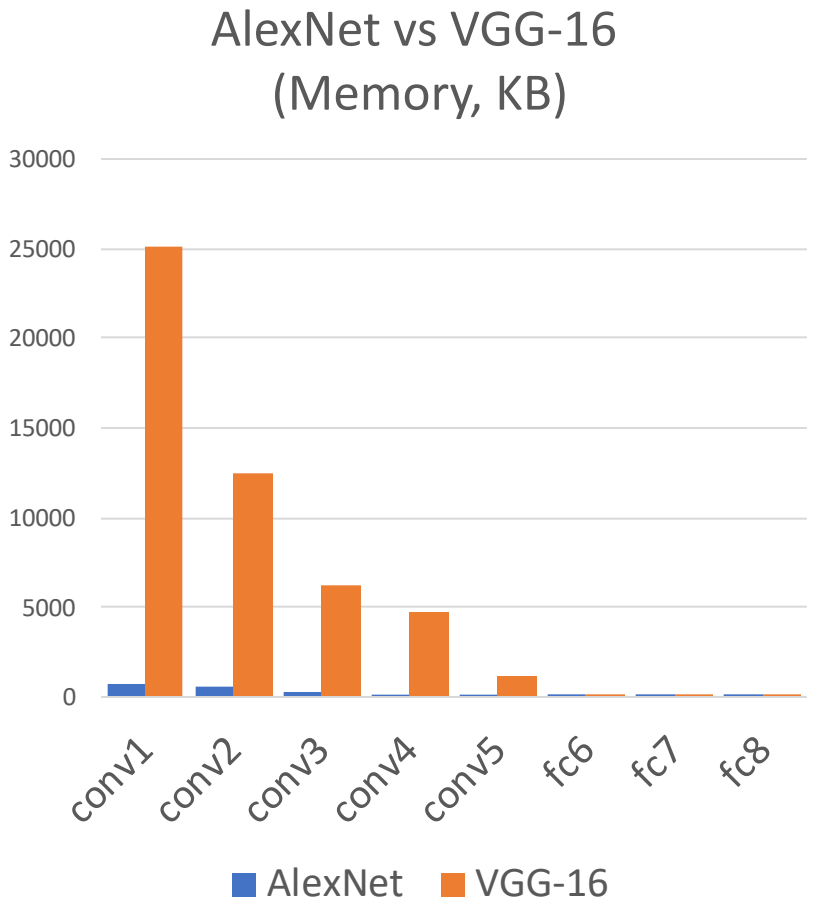


VGG16

VGG19

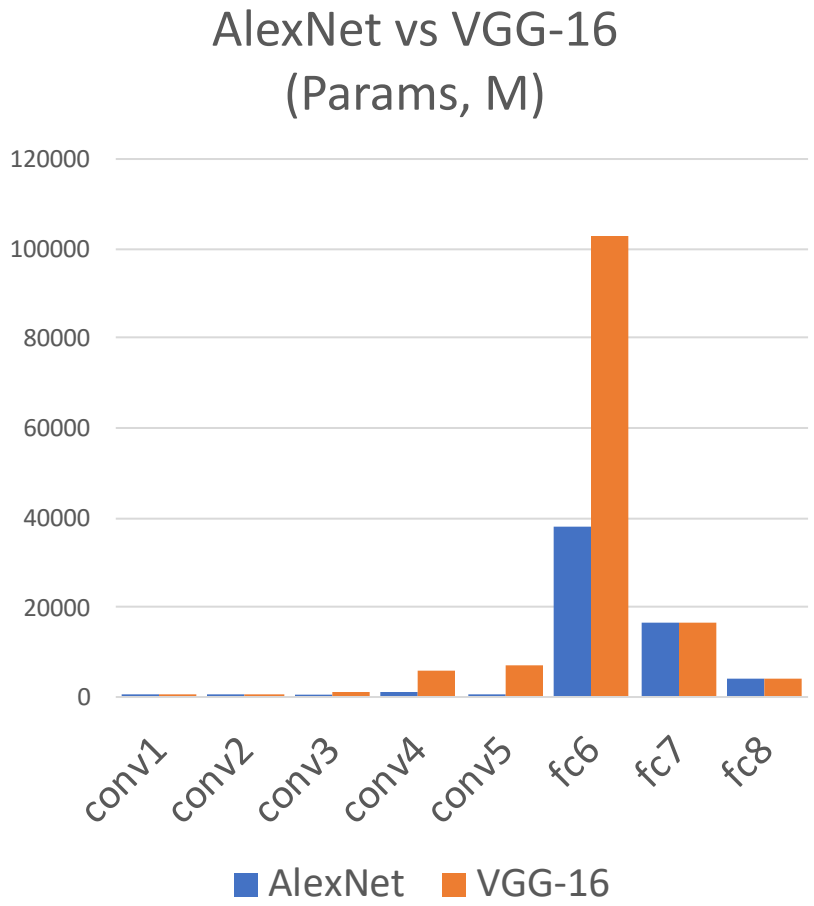
Simonyan and Zissermann, "Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR 2015

AlexNet vs VGG-16: Much bigger network!



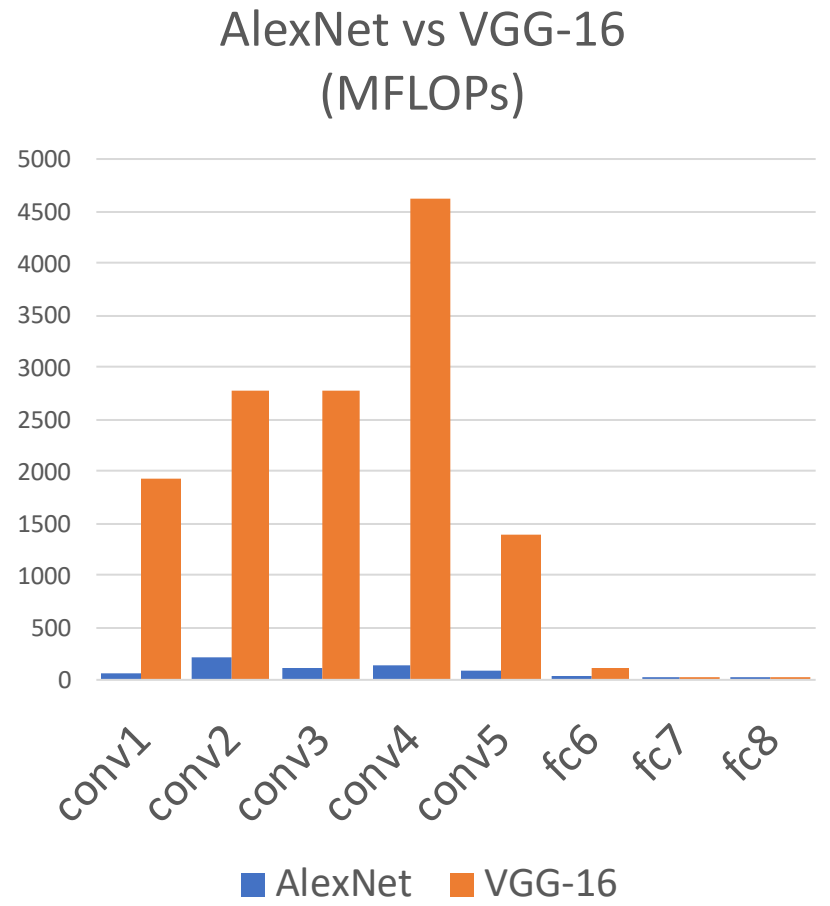
AlexNet total: 1.9 MB

VGG-16 total: 48.6 MB (25x)



AlexNet total: 61M

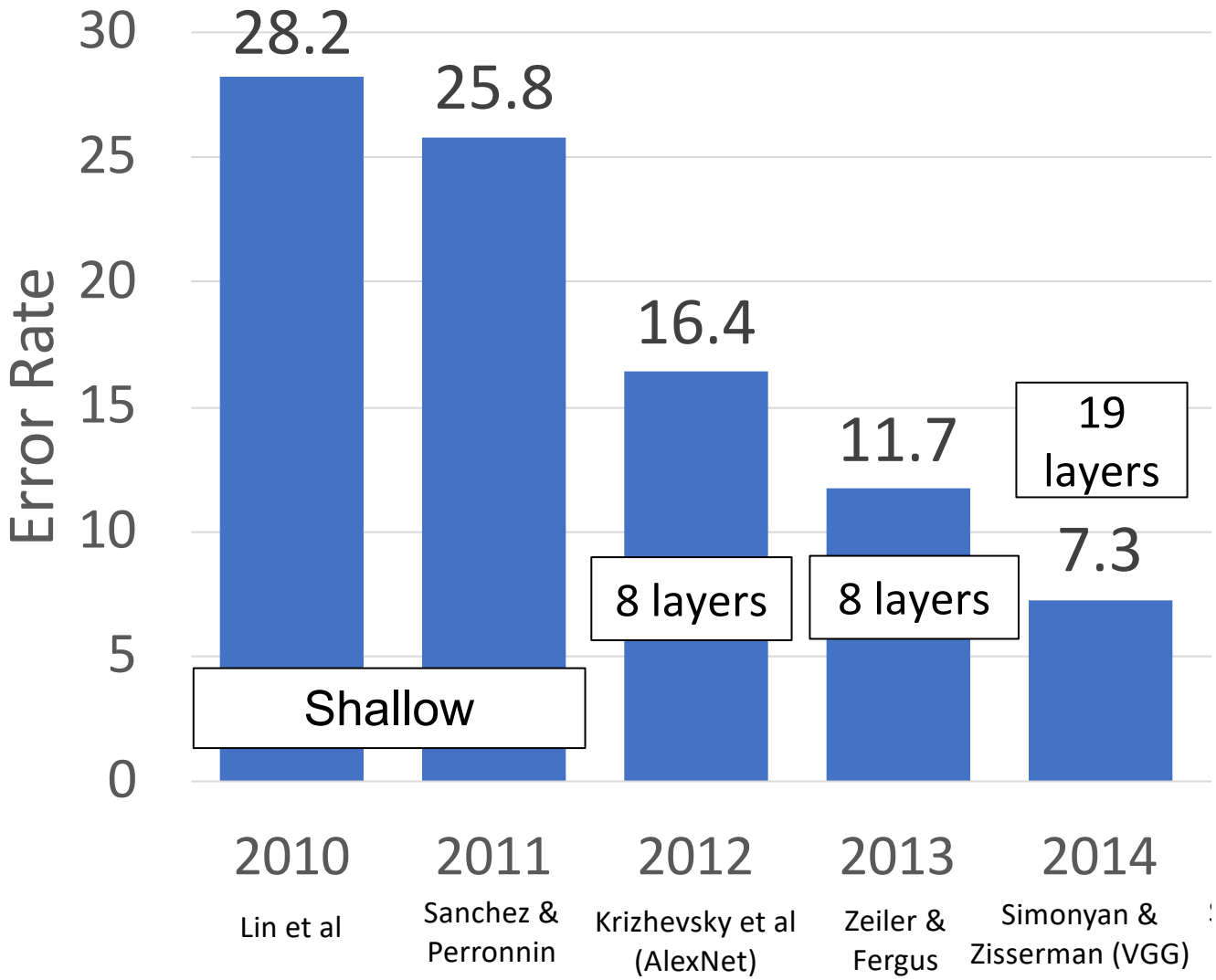
VGG-16 total: 138M (2.3x)



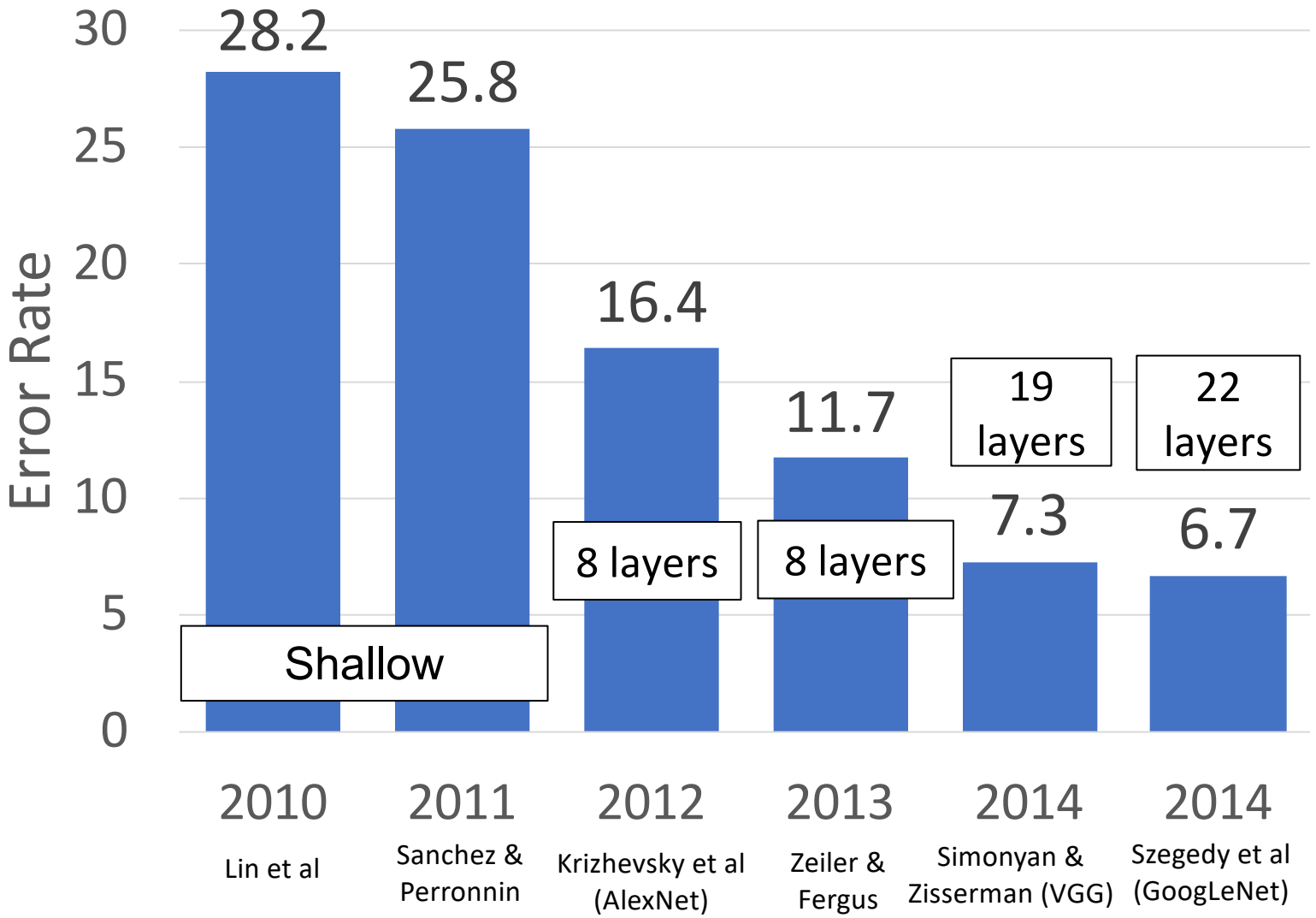
AlexNet total: 0.7 GFLOP

VGG-16 total: 13.6 GFLOP (19.4x)

ImageNet Classification Challenge

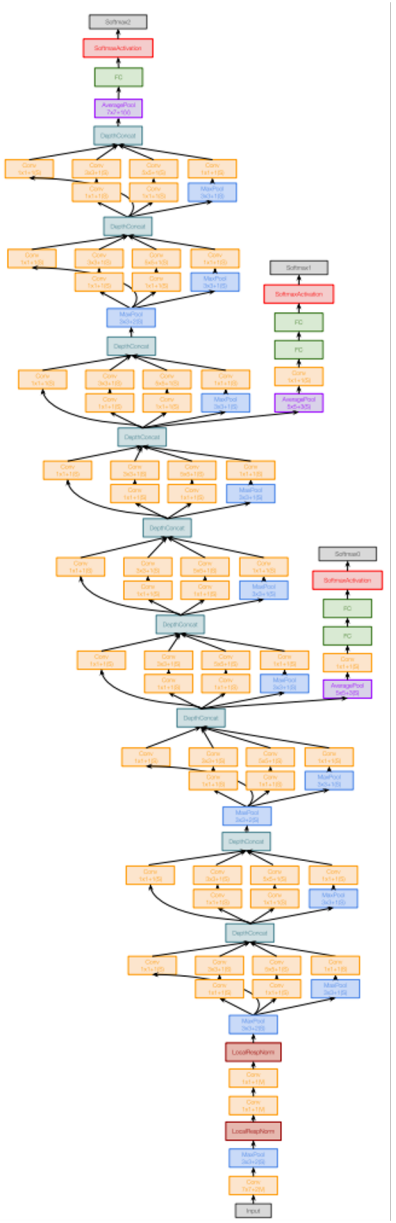


ImageNet Classification Challenge



GoogLeNet: Focus on Efficiency

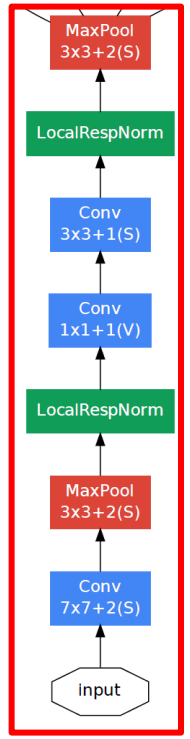
Many innovations for efficiency: reduce parameter count, memory usage, and computation

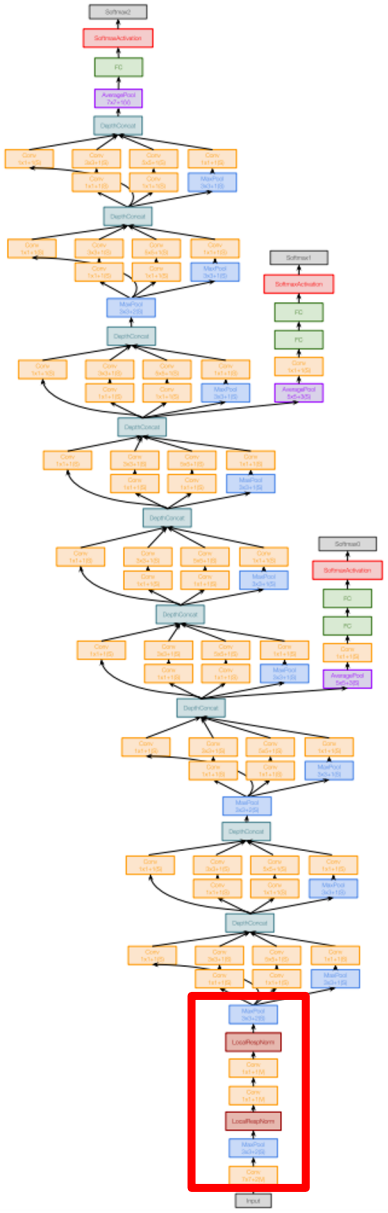


Szegedy et al, “Going deeper with convolutions”, CVPR 2015

GoogLeNet: Aggressive Stem

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



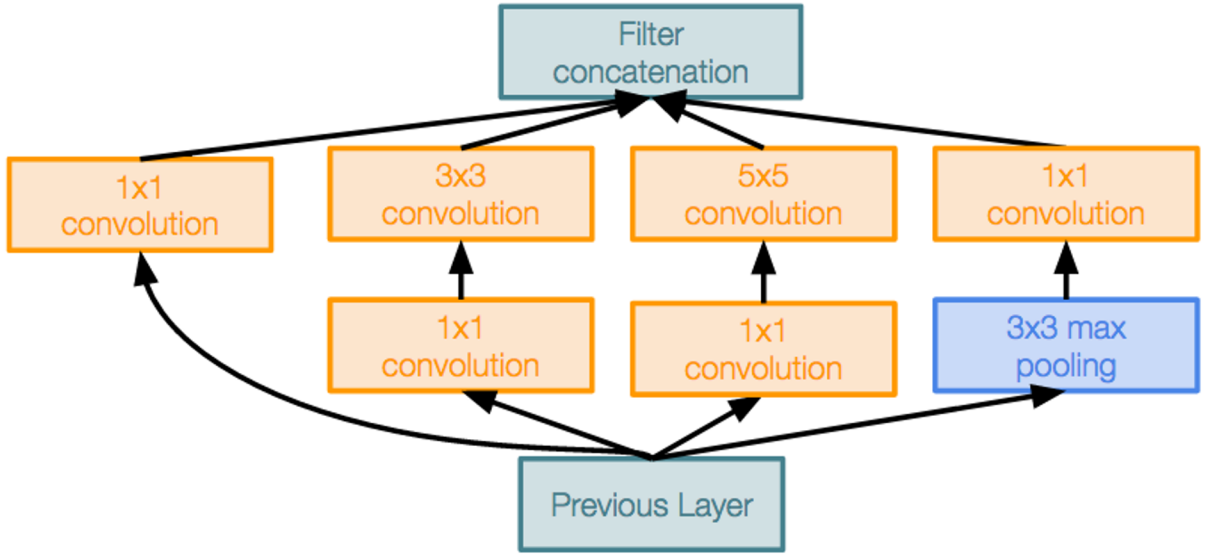


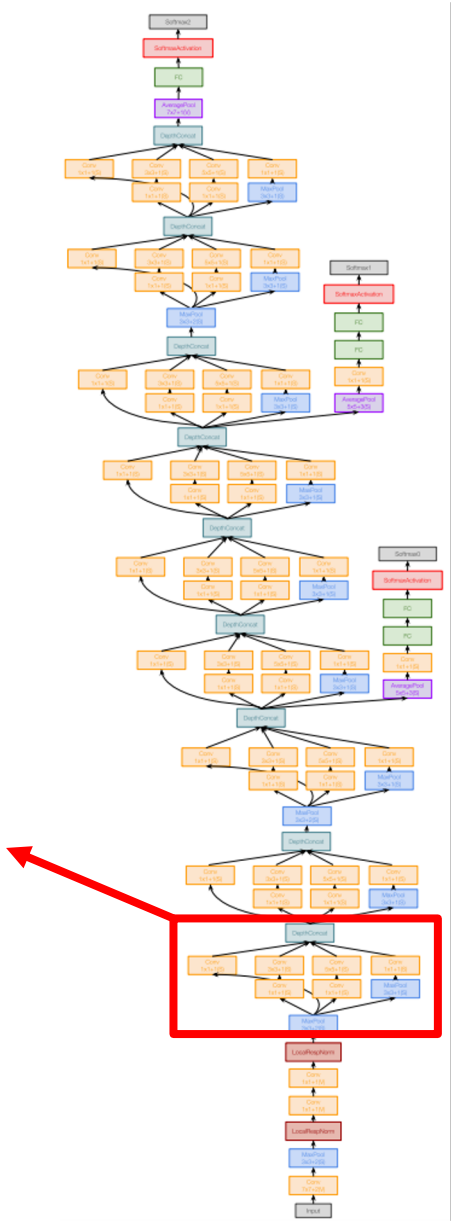
Szegedy et al, “Going deeper with convolutions”, CVPR 2015

GoogLeNet: Inception Module

Inception module
Local unit with
parallel branches

Local structure repeated
many times throughout the
network





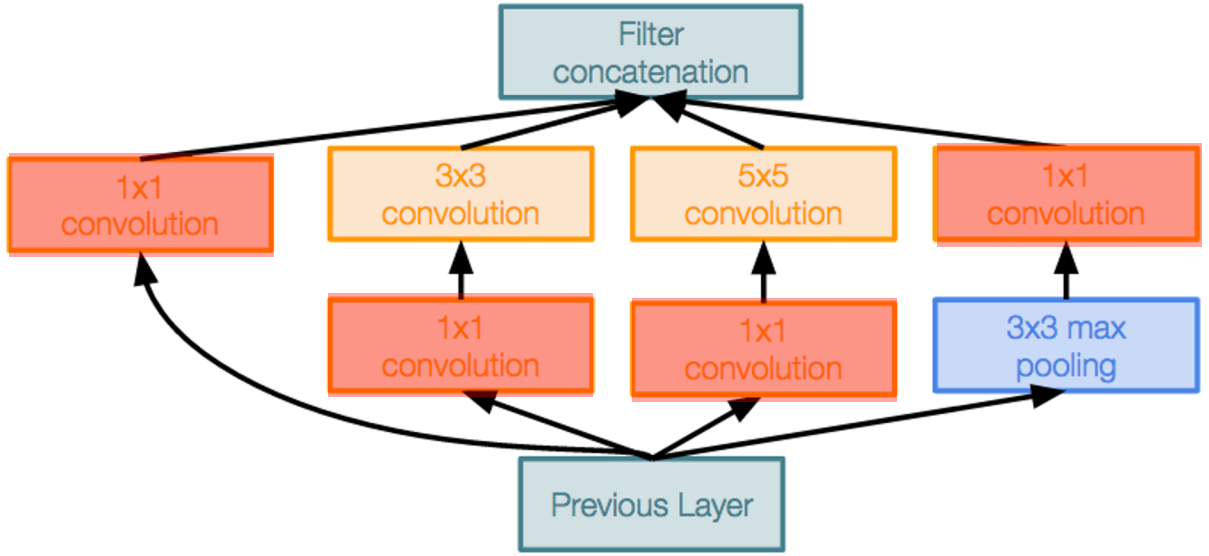
Szegedy et al, “Going deeper with convolutions”, CVPR 2015

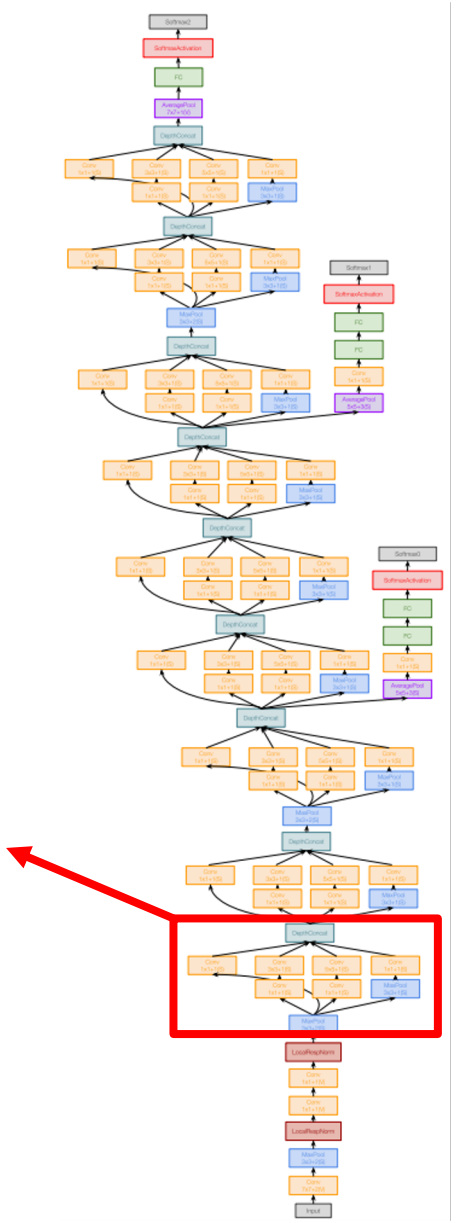
GoogLeNet: Inception Module

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!)





Related to (sum of) separable filters

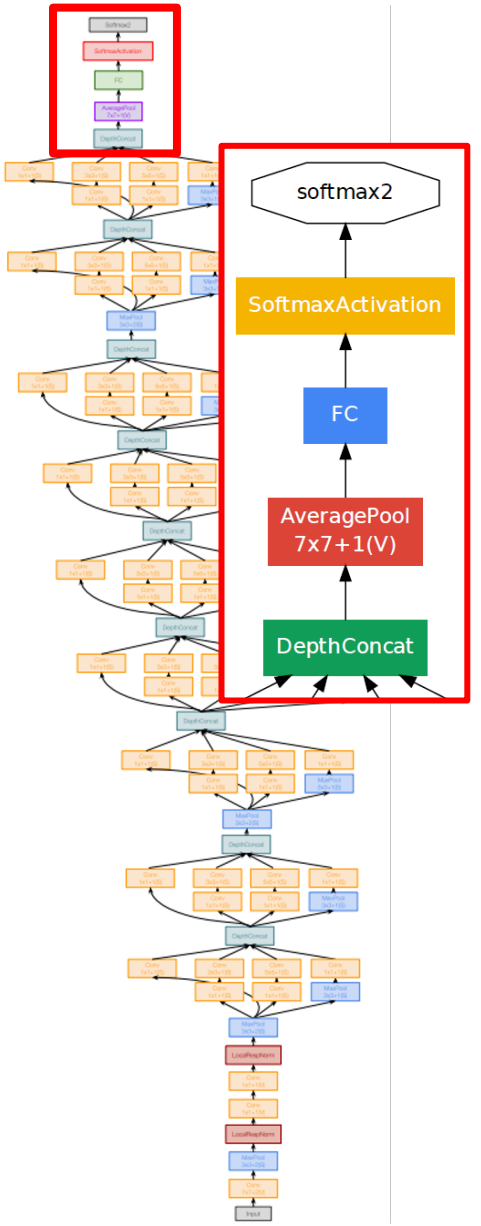
Perona, “Deformable kernels for early vision”. PAMI 1995

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

GoogLeNet: Global Average Pooling

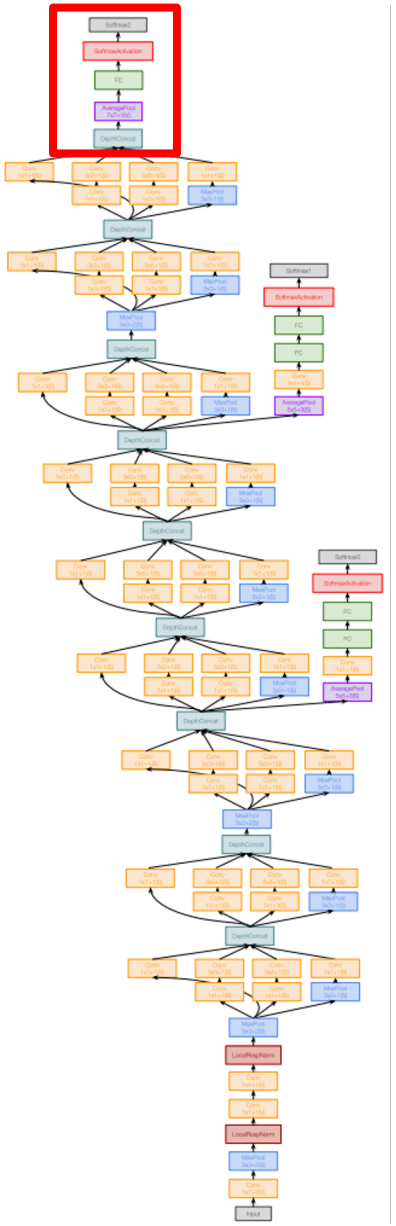
No large FC layers at the end! Instead uses **global average pooling** to collapse spatial dimensions, and one linear layer to produce class scores (Recall VGG-16: Most parameters were in the FC layers!)

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



GoogLeNet: Global Average Pooling

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



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

Compare with VGG-16:

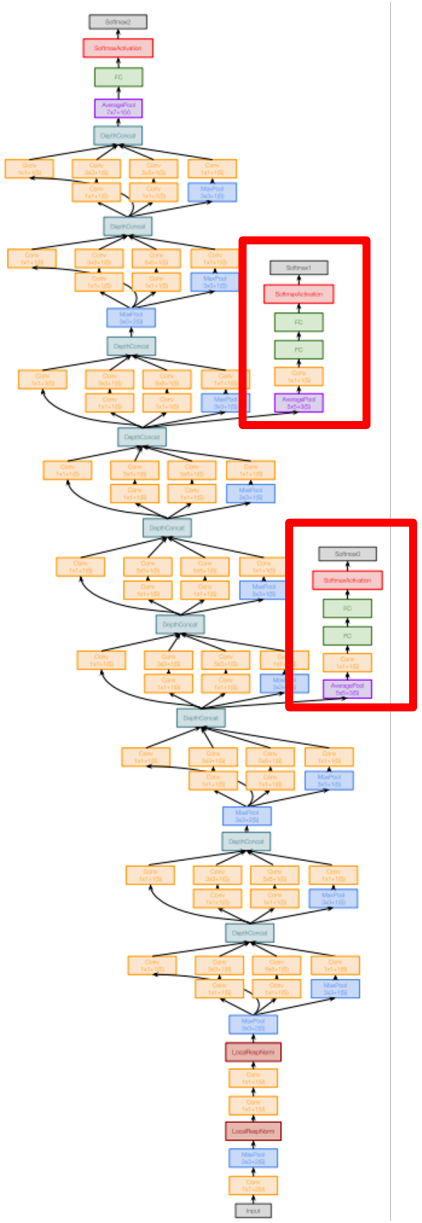
Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (K)	flop (M)
flatten	512	7					25088		98		
fc6	25088			4096			4096		16	102760	103
fc7	4096			4096			4096		16	16777	17
fc8	4096			1000			1000		4	4096	4

GoogLeNet: Auxiliary Classifiers

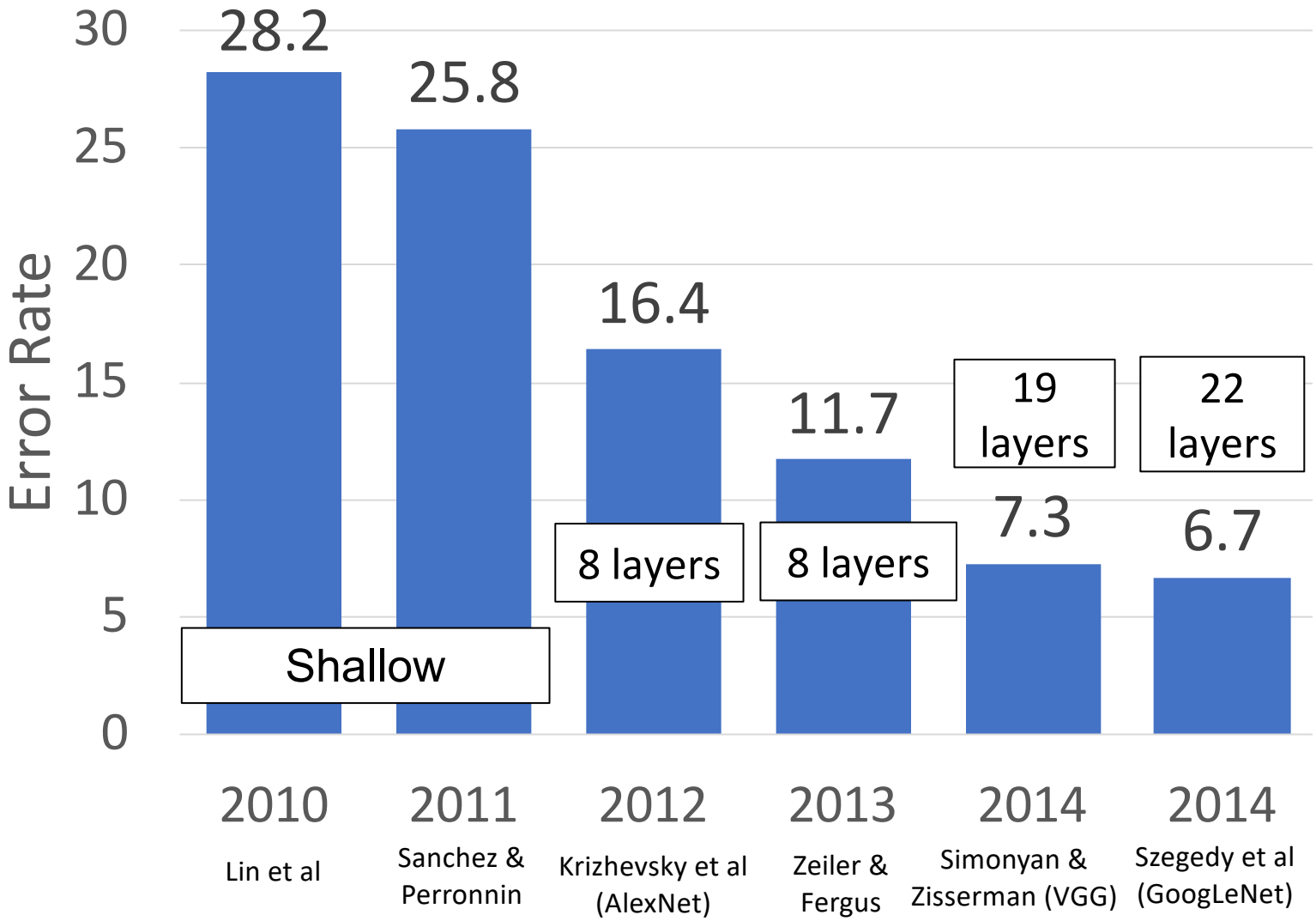
Training using loss at the end of the network didn't work well:
Network is too deep, gradients don't propagate cleanly

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

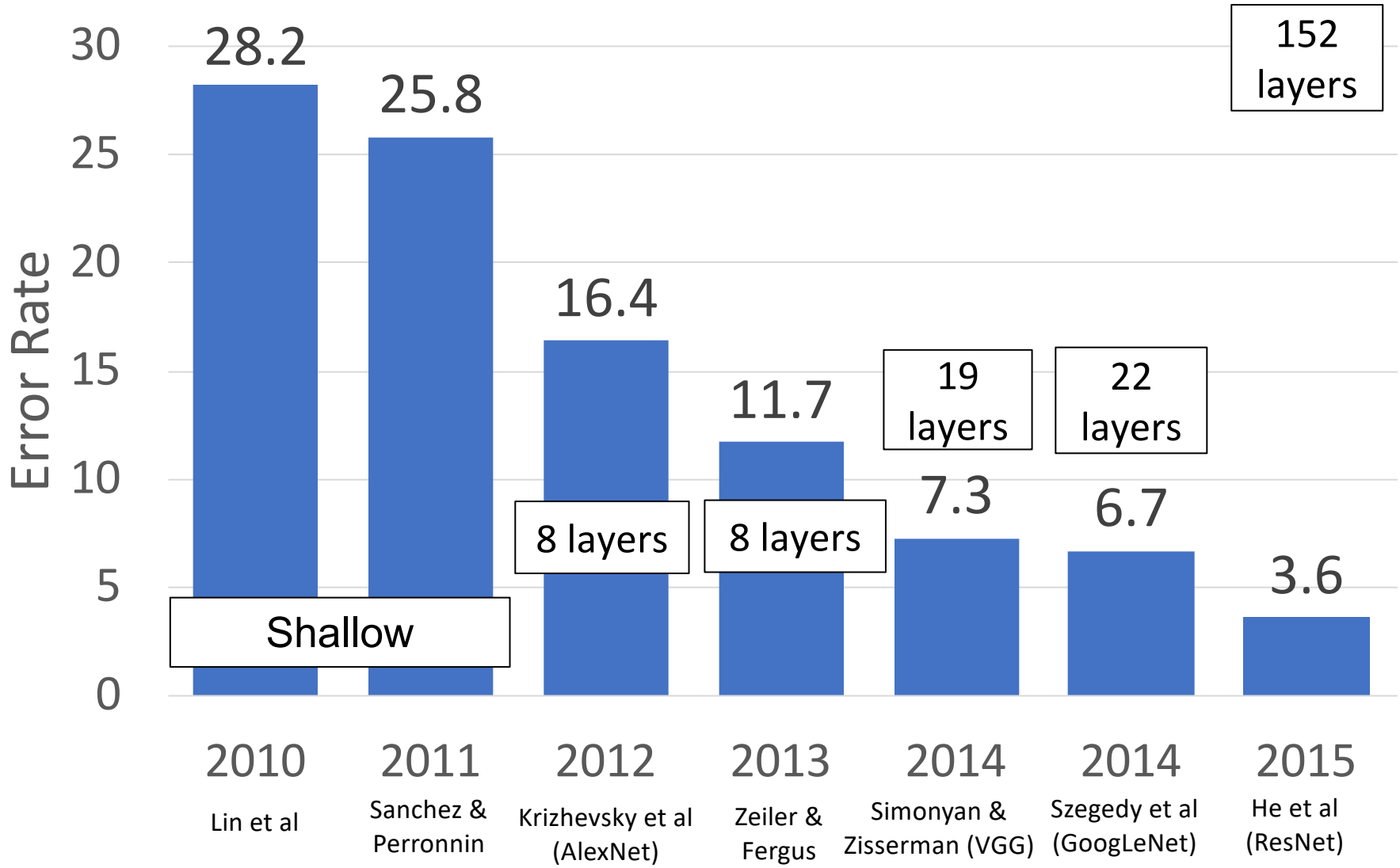
GoogLeNet was before batch normalization! With BatchNorm no longer need to use this trick



ImageNet Classification Challenge



ImageNet Classification Challenge



Residual Networks

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

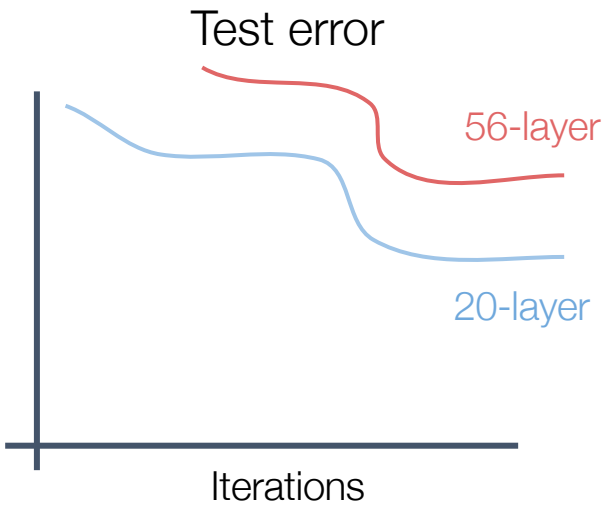
He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016

Residual Networks

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

Deeper model does worse than shallow model!

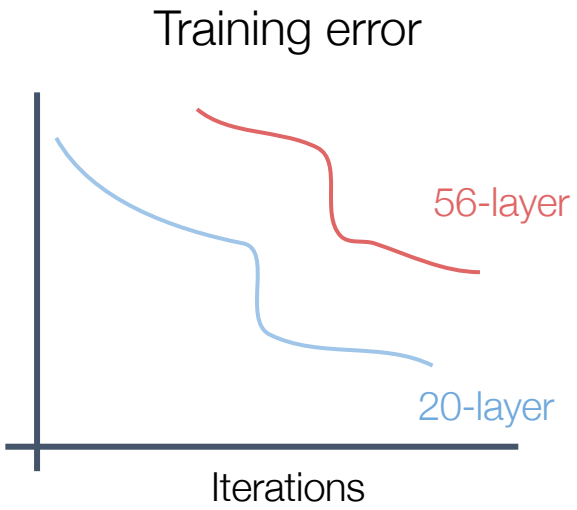
Initial guess: Deep model is **overfitting** since it is much bigger than the other model

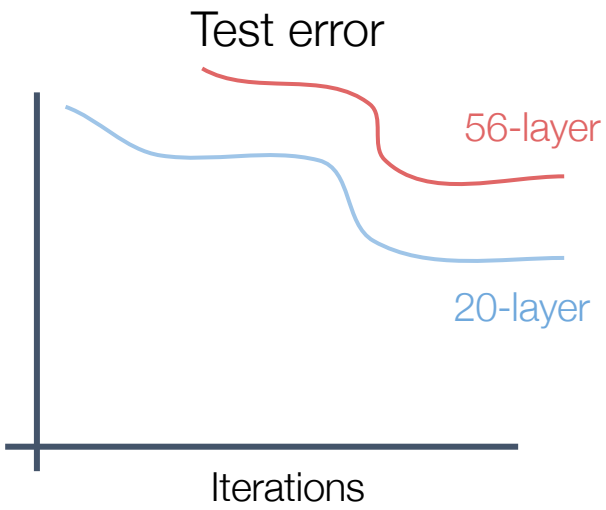


He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016

Residual Networks

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





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

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

Residual Networks

A deeper model can 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

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

Residual Networks

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

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

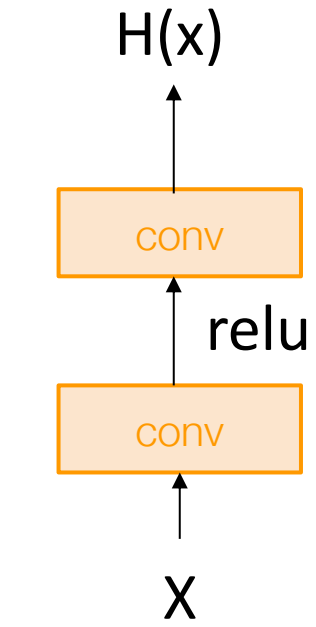
Hypothesis: This is an optimization problem. Deeper models are harder to optimize, and in particular don't learn identity functions to emulate shallow models

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

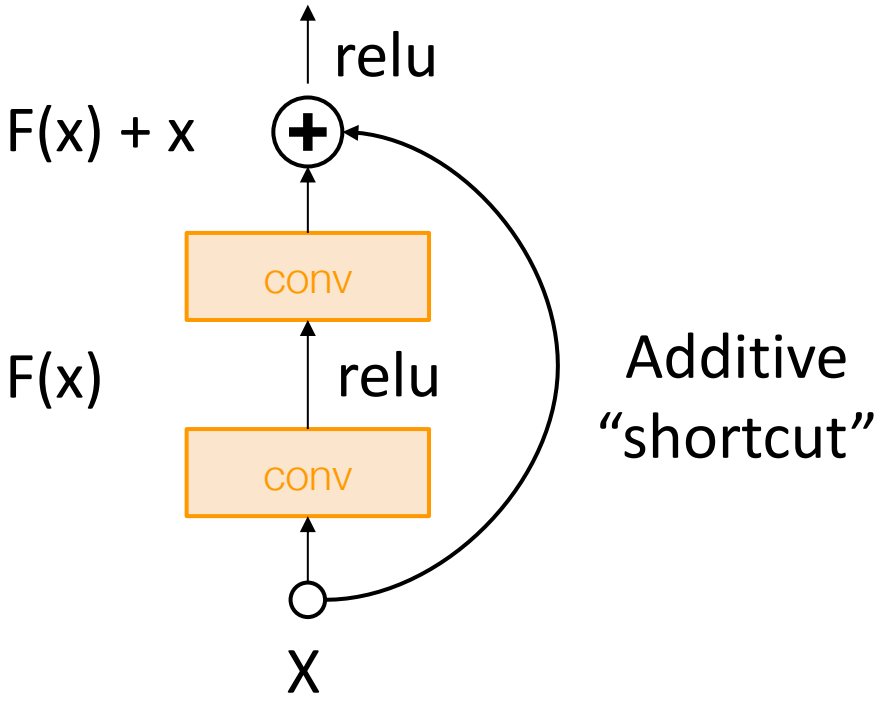
He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Residual Networks

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



"Plain" block

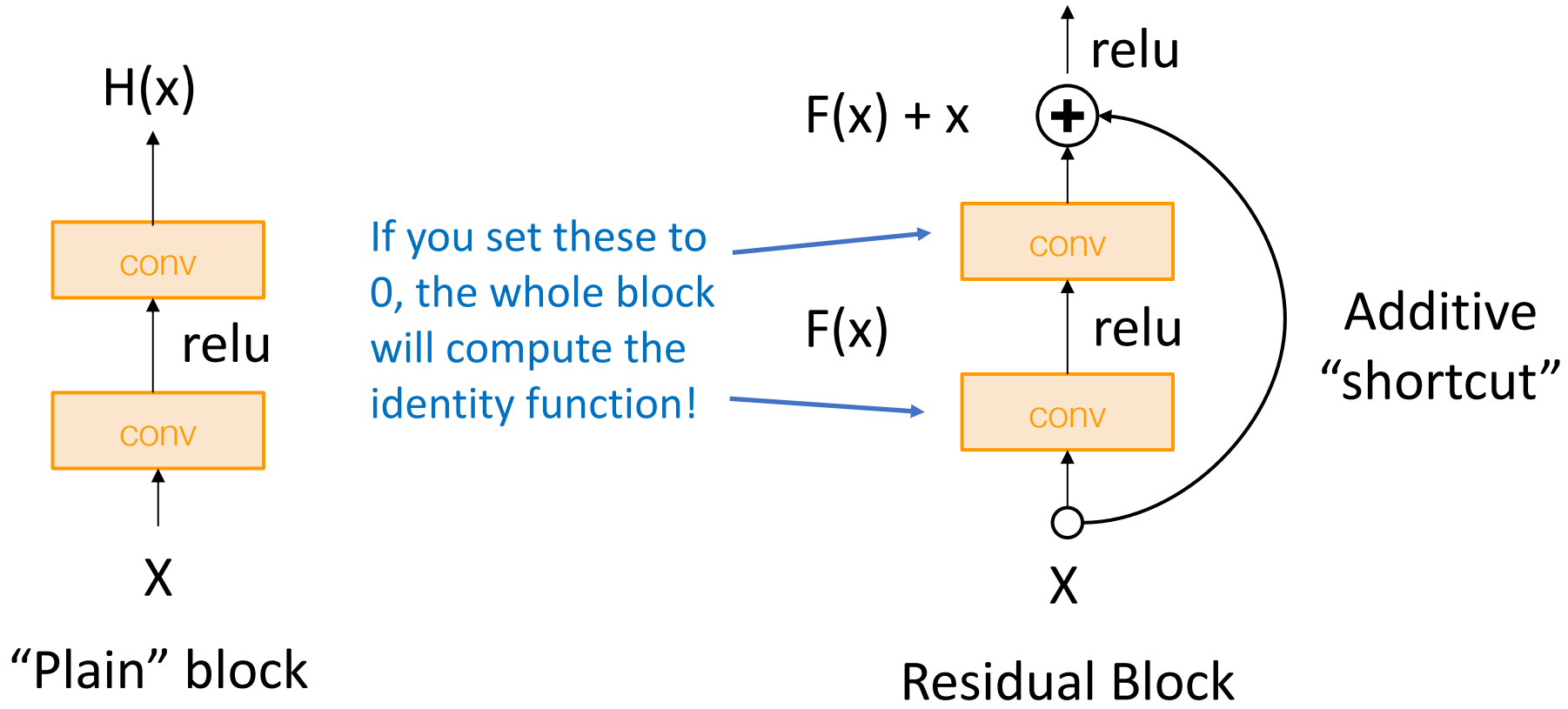


Residual Block

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

Residual Networks

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



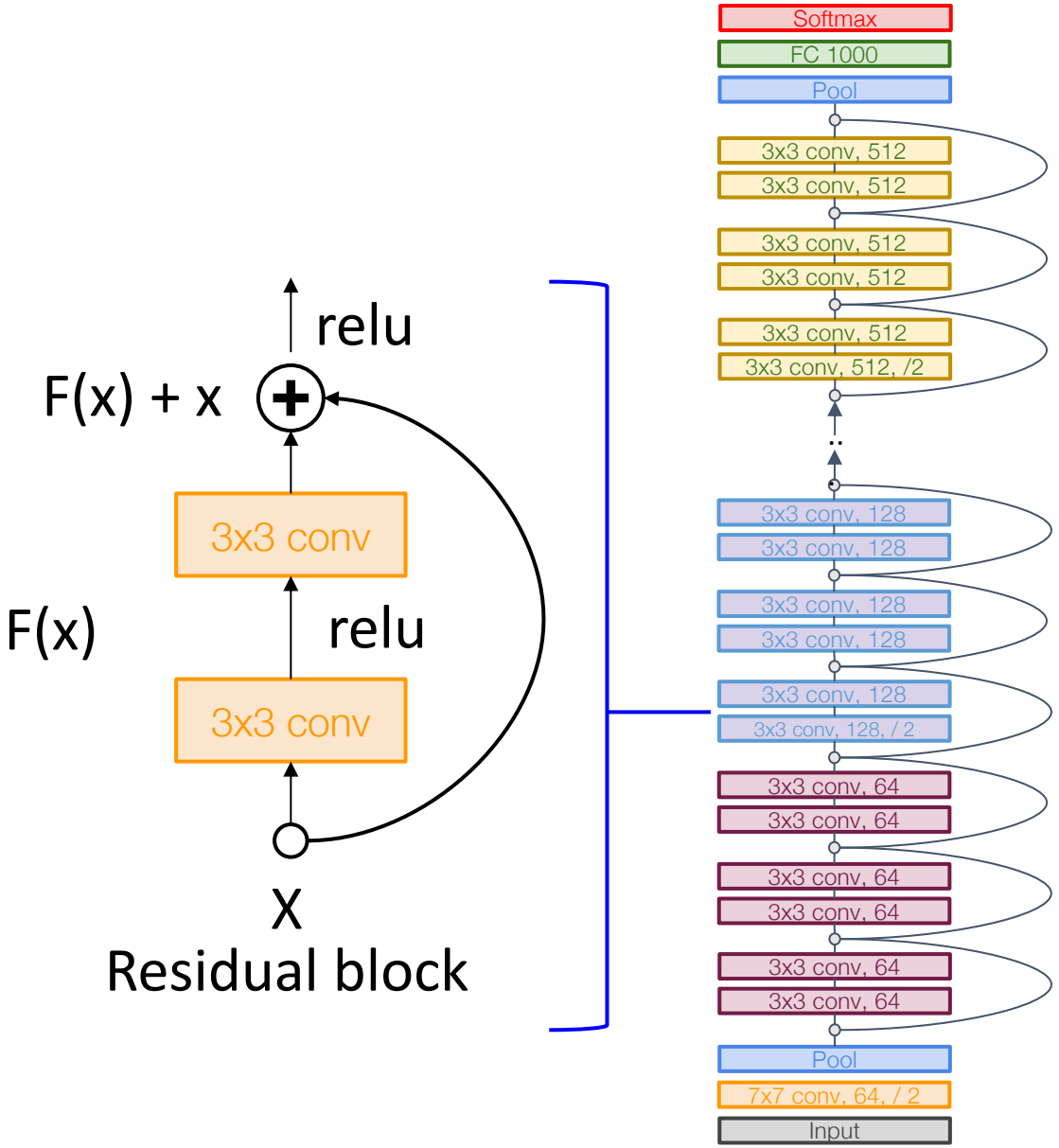
He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Residual Networks

A residual network is a stack of many residual blocks

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

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

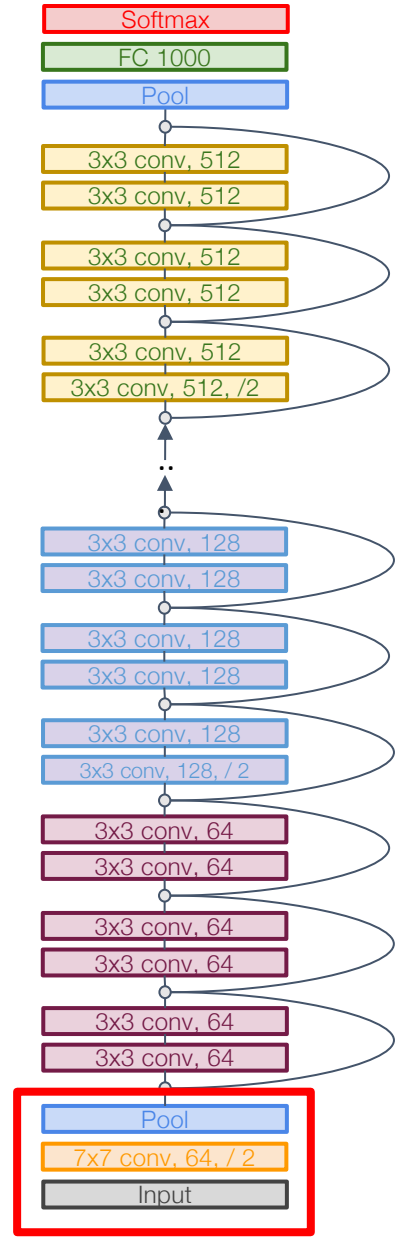


He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Residual Networks

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

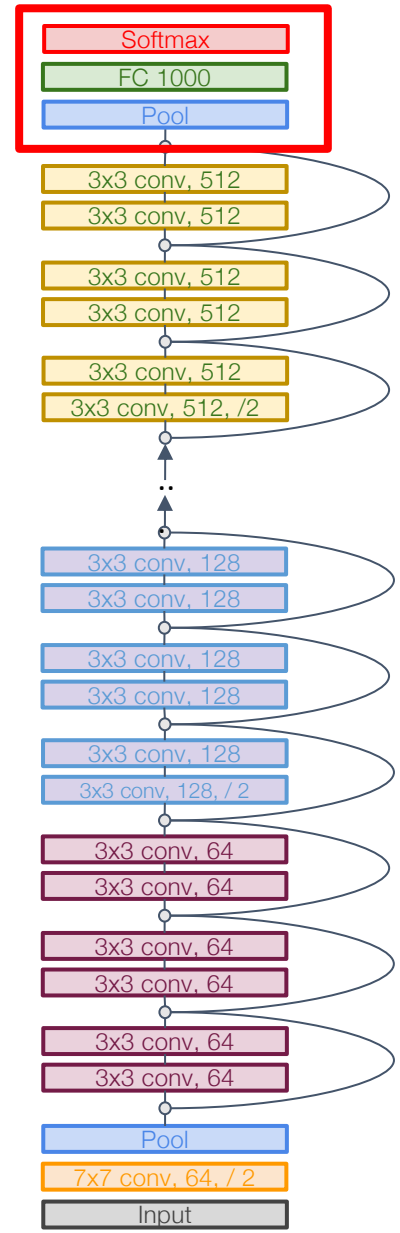
	Input size		Layer				Output size				
Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (k)	flop (M)
conv	3	224	64	7	2	3	64	112	3136	9	118
max-pool	64	112		3	2	1	64	56	784	0	2



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

Residual Networks

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



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

Residual Networks

ResNet-18:

Stem: 1 conv layer

Stage 1 (C=64): 2 res. block = 4 conv

Stage 2 (C=128): 2 res. block = 4 conv

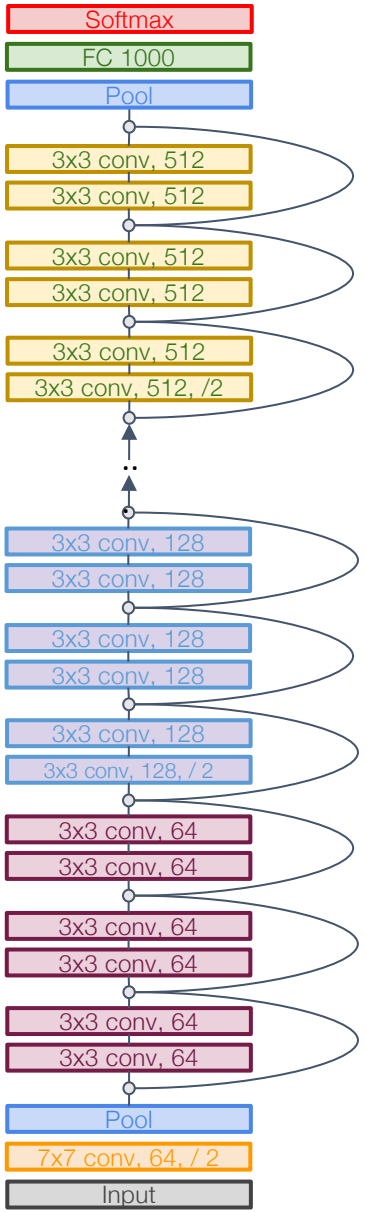
Stage 3 (C=256): 2 res. block = 4 conv

Stage 4 (C=512): 2 res. block = 4 conv

Linear

ImageNet top-5 error: 10.92

GFLOP: 1.8



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

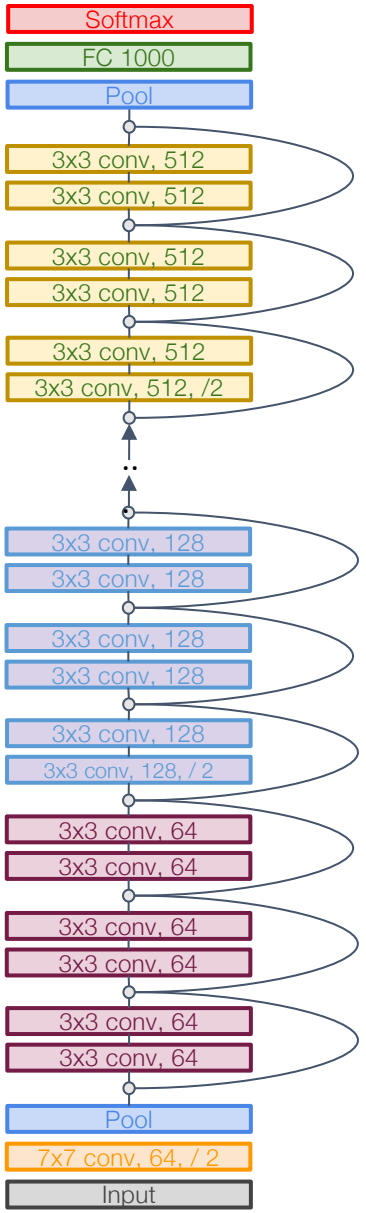
Residual Networks

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

ImageNet top-5 error: 10.92
GFLOP: 1.8

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

ImageNet top-5 error: 8.58
GFLOP: 3.6



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

Residual Networks

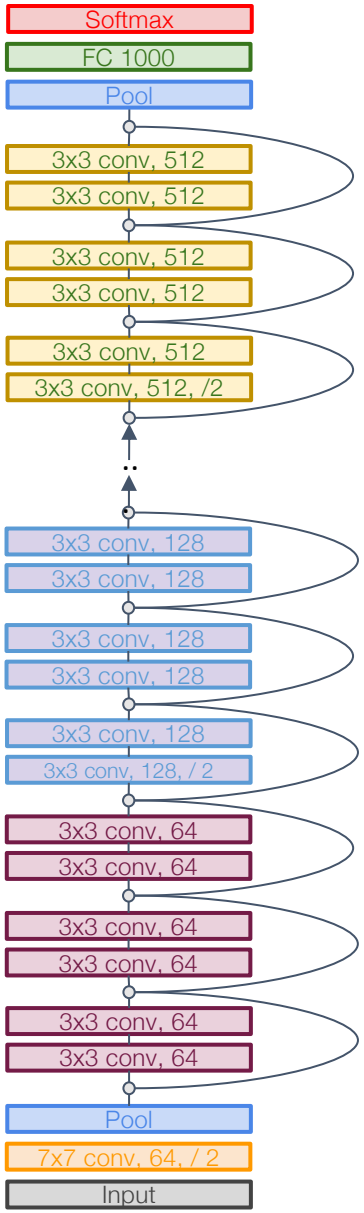
ResNet-18:
Stem: 1 conv layer
Stage 1 (C=64): 2 res. block = 4 conv
Stage 2 (C=128): 2 res. block = 4 conv
Stage 3 (C=256): 2 res. block = 4 conv
Stage 4 (C=512): 2 res. block = 4 conv
Linear

ImageNet top-5 error: 10.92
GFLOP: 1.8

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

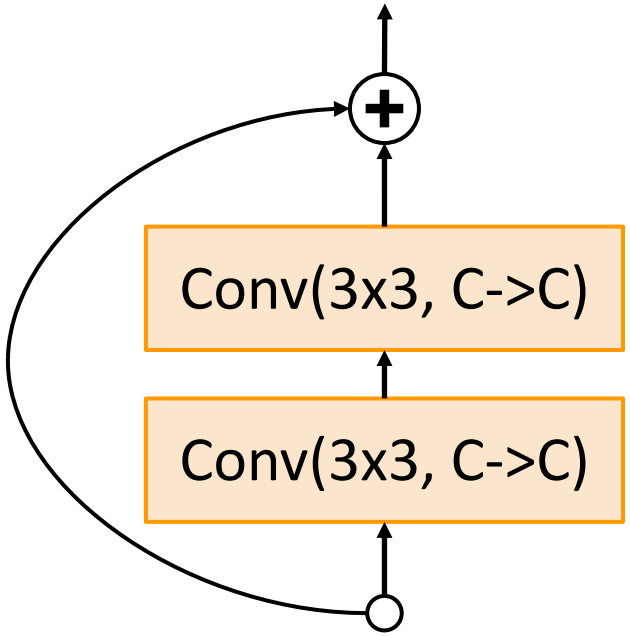
ImageNet top-5 error: 8.58
GFLOP: 3.6

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



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

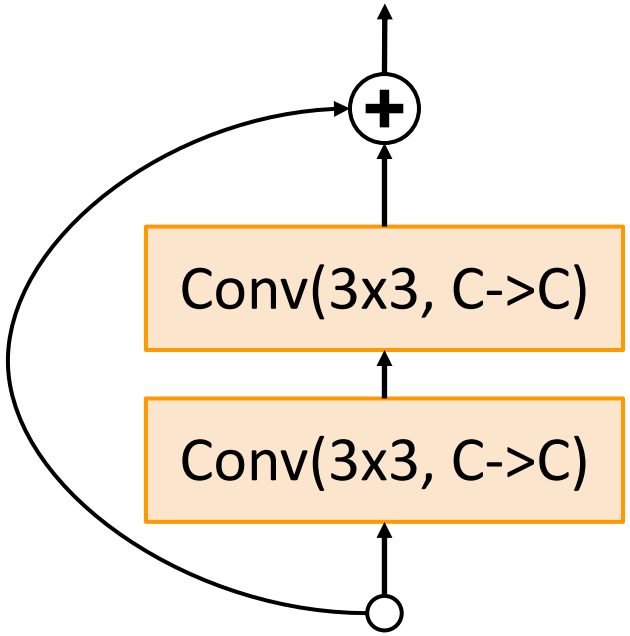
Residual Networks: Basic Block



"Basic"
Residual block

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

Residual Networks: Basic Block



FLOPs: $9HWC^2$

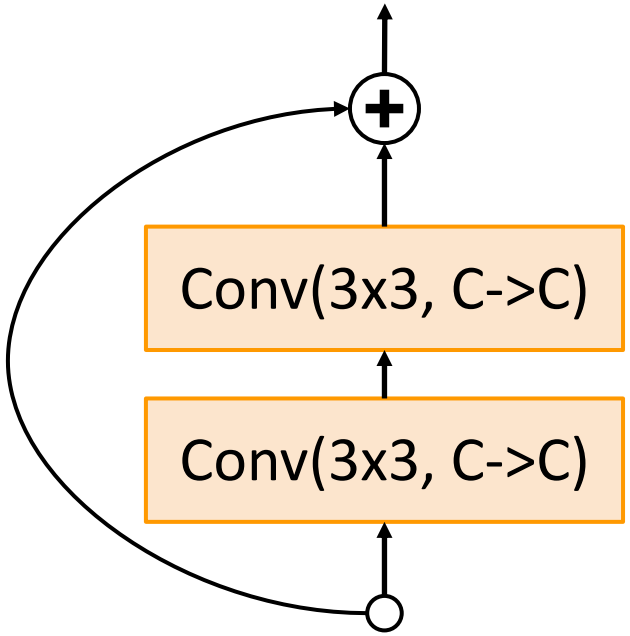
FLOPs: $9HWC^2$

"Basic"
Residual block

Total FLOPs:
$18HWC^2$

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

Residual Networks: Bottleneck Block

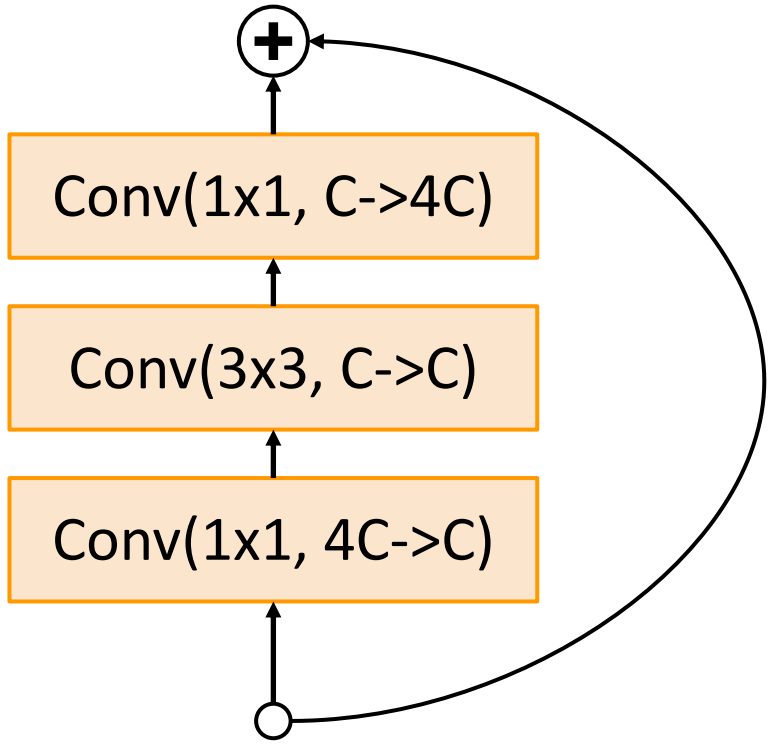


FLOPs: $9HWC^2$

FLOPs: $9HWC^2$

"Basic"
Residual block

Total FLOPs:
$18HWC^2$

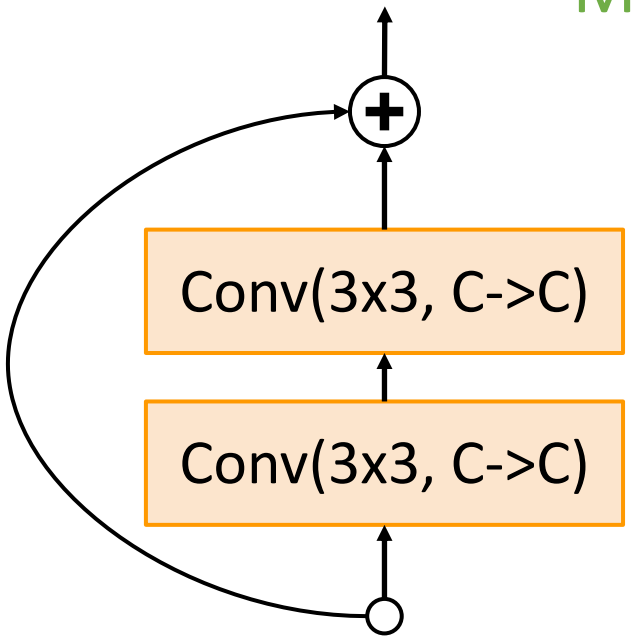


"Bottleneck"
Residual block

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

Residual Networks: Bottleneck Block

More layers, less computational cost!



FLOPs: $9HWC^2$

FLOPs: $9HWC^2$

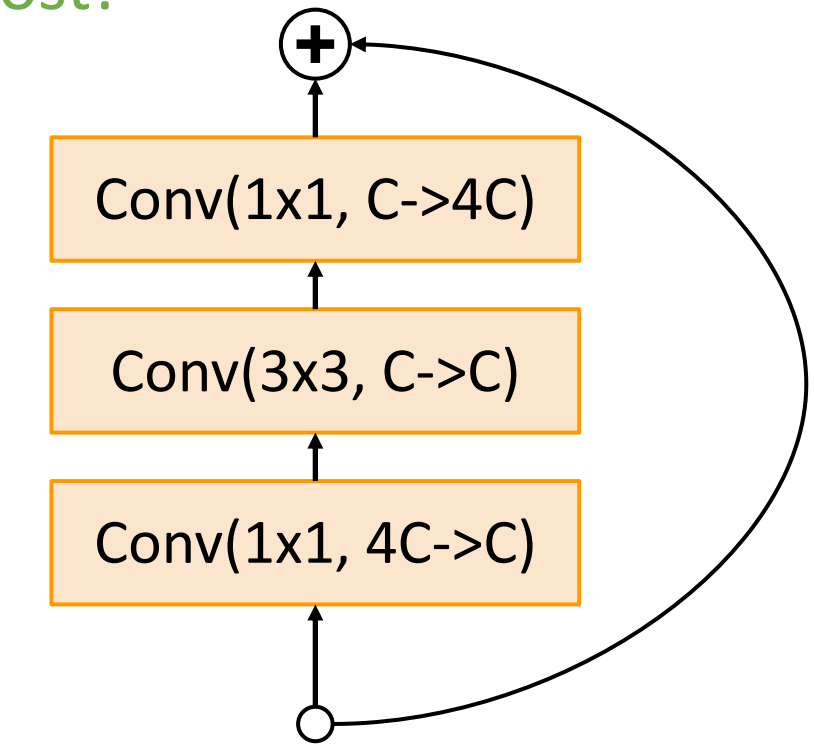
"Basic" Residual block

Total FLOPs: $18HWC^2$

FLOPs: $4HWC^2$

FLOPs: $9HWC^2$

FLOPs: $4HWC^2$



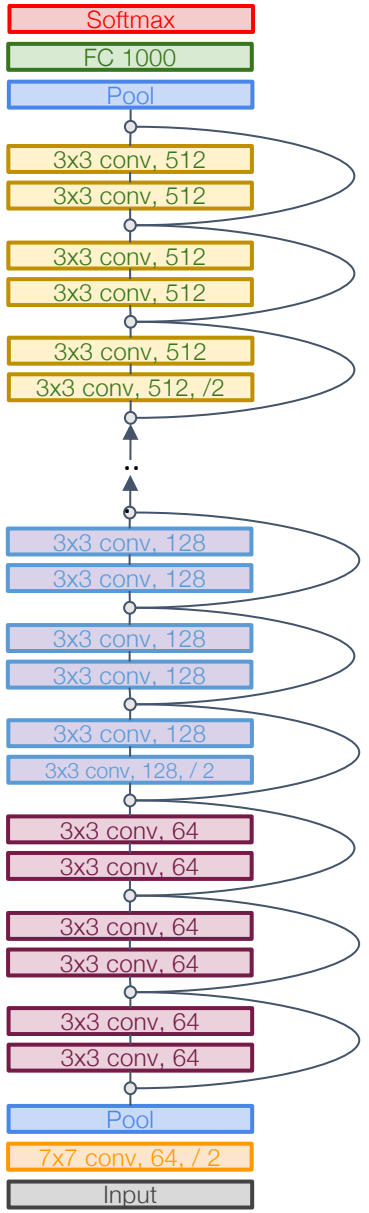
Total FLOPs: $17HWC^2$

"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

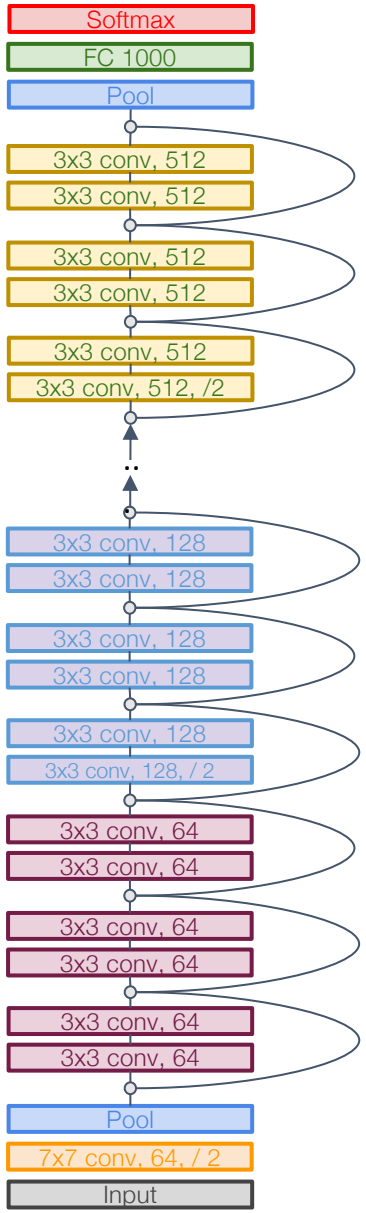


He et al, "Deep Residual Learning for Image Recognition", CVPR 2016
Error rates are 224x224 single-crop testing, reported by torchvision

Residual Networks

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

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

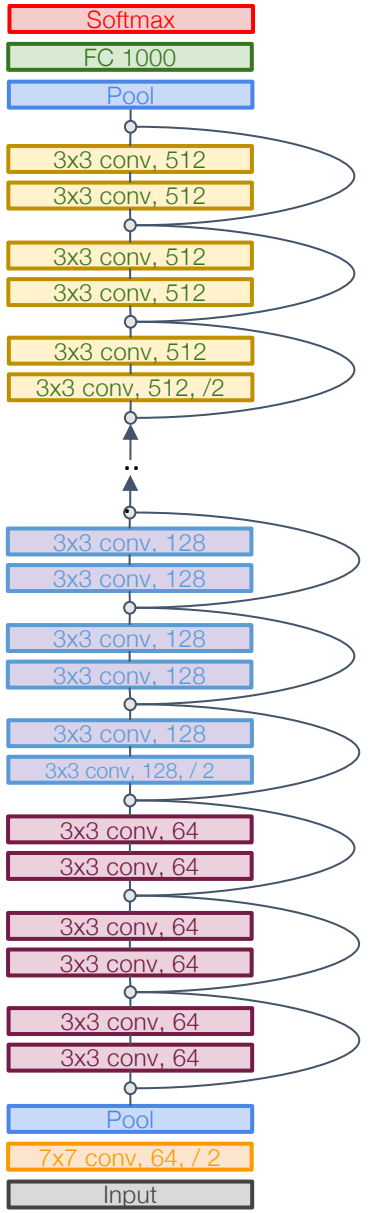


He et al, "Deep Residual Learning for Image Recognition", CVPR 2016
Error rates are 224x224 single-crop testing, reported by torchvision

Residual Networks

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

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



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

Residual Networks

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

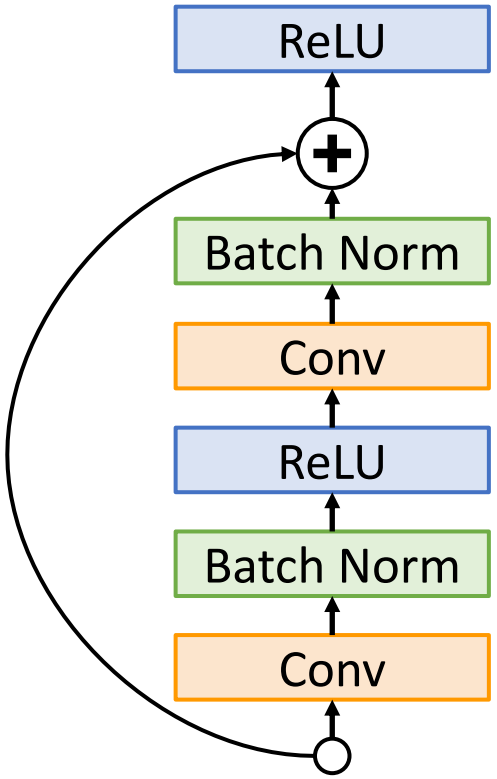
MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: "*Ultra-deep*" (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

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

Improving Residual Networks: Block Design

Original ResNet block



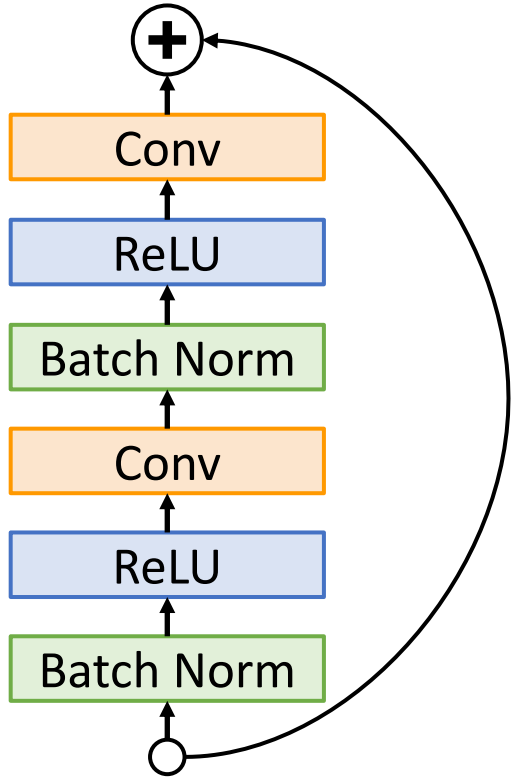
Note ReLU **after** residual:

Cannot actually learn identity function since outputs are nonnegative!

Note ReLU **inside** residual:

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

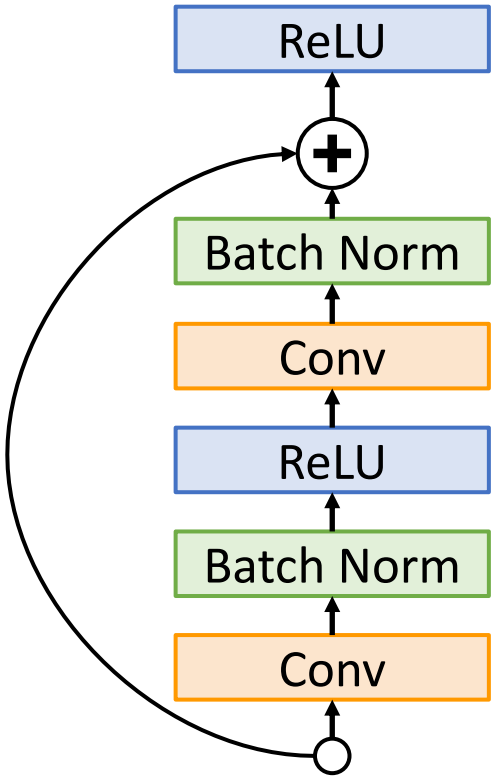
"Pre-Activation" ResNet Block



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

Improving Residual Networks: Block Design

Original ResNet block

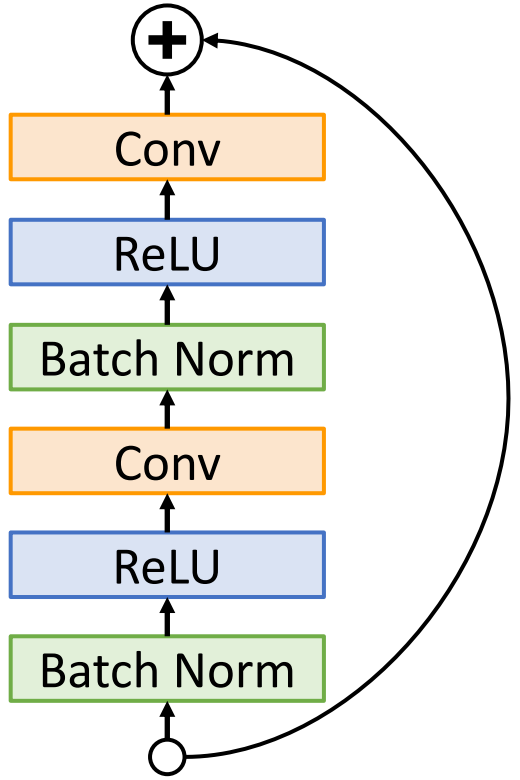


Slight improvement in accuracy (ImageNet top-1 error)

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

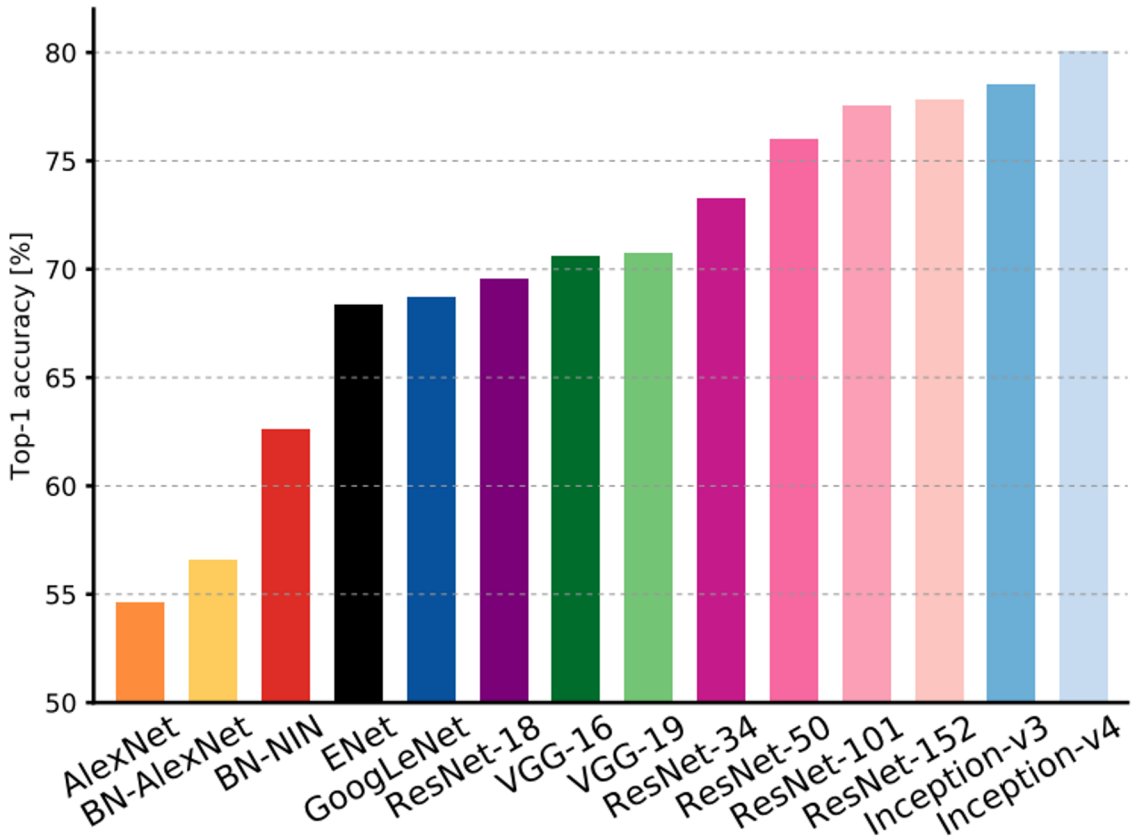
Not actually used that much in practice

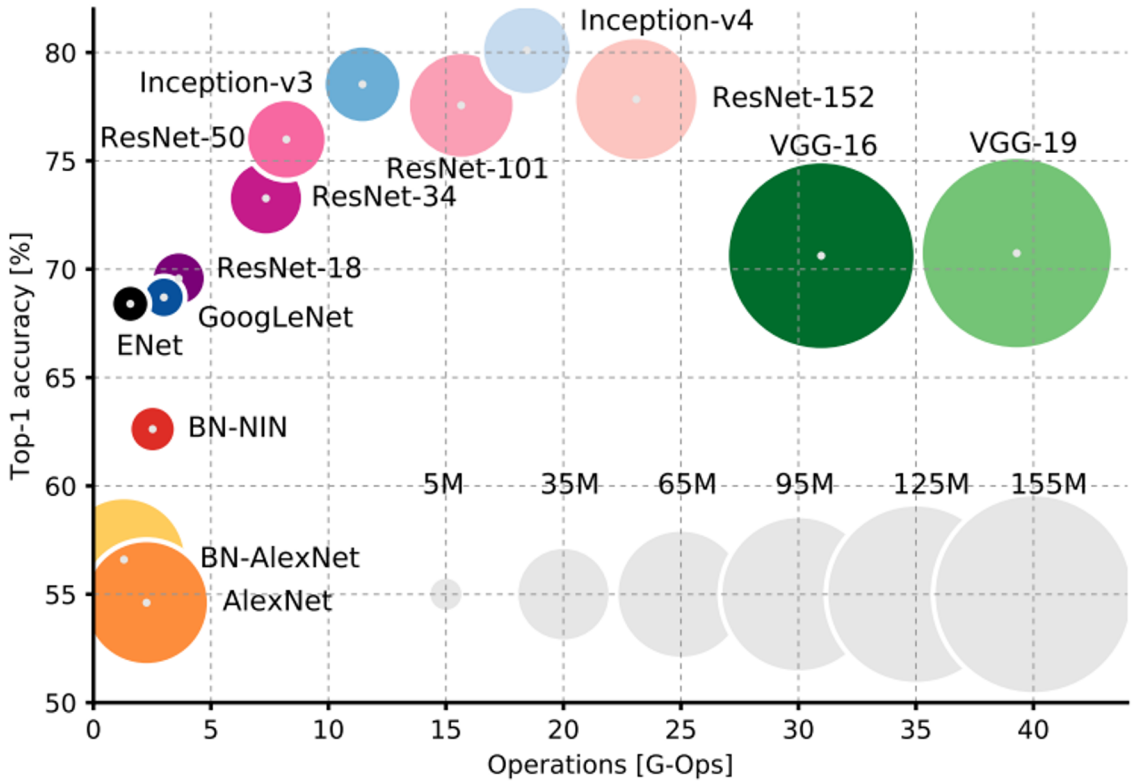
"Pre-Activation" ResNet Block



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

Comparing Complexity



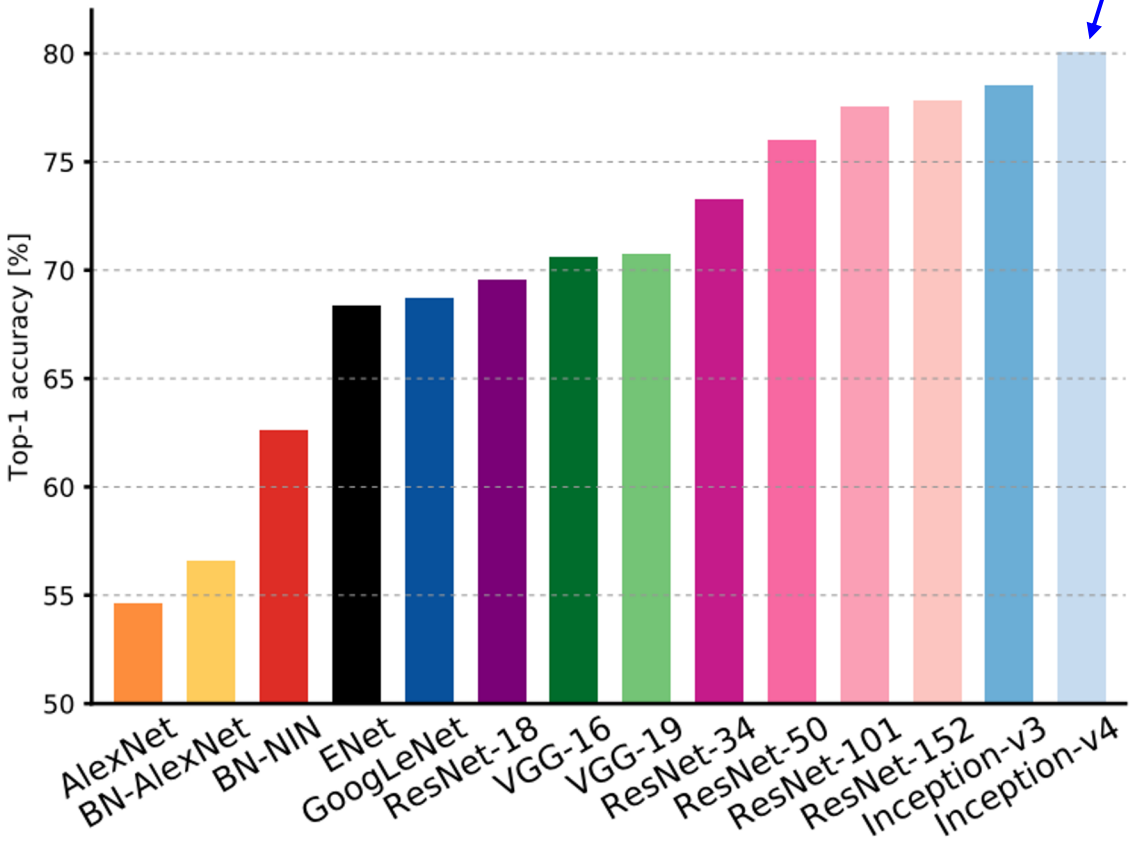


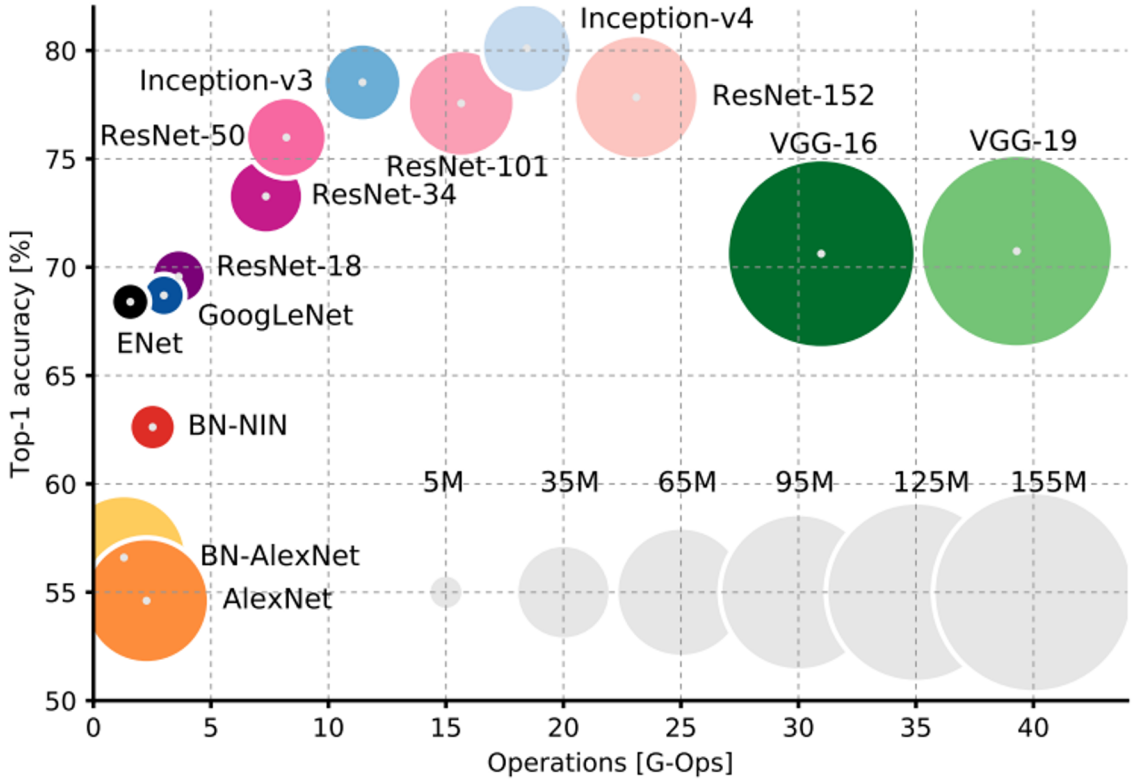
BN: batch normalized version

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

Comparing Complexity

Inception-v4: Resnet + Inception!

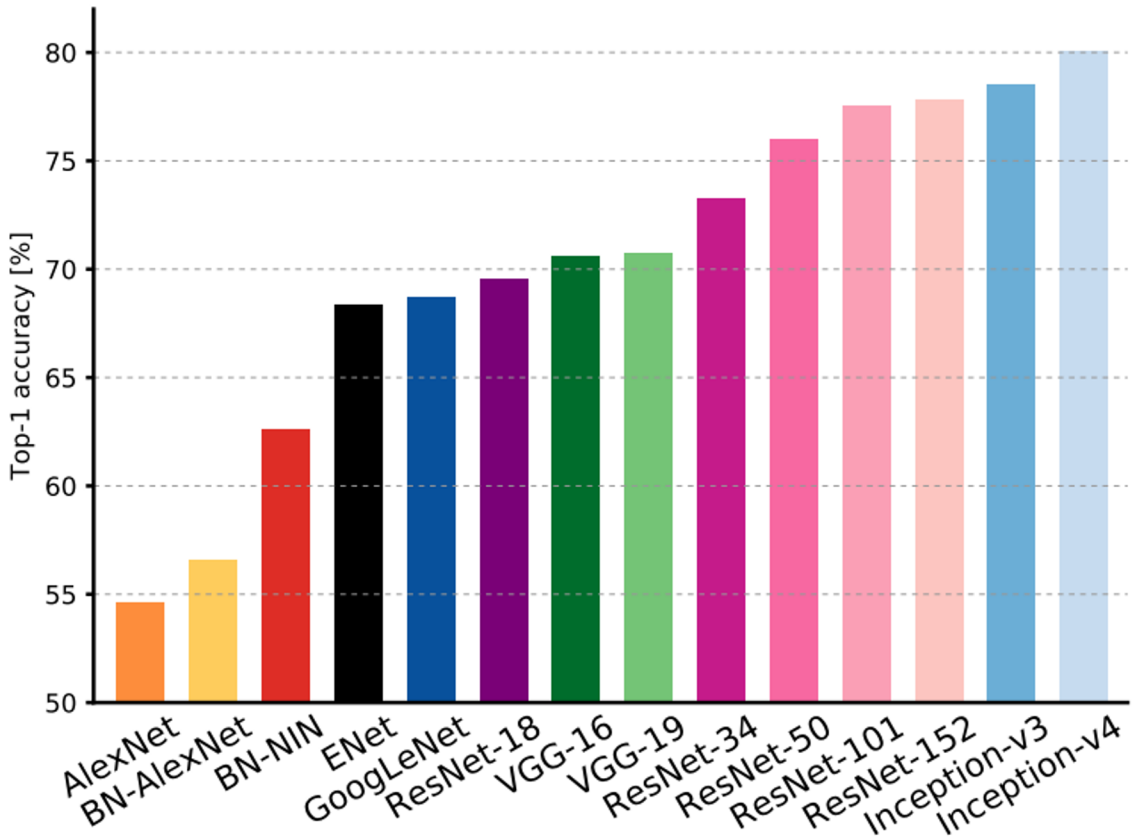




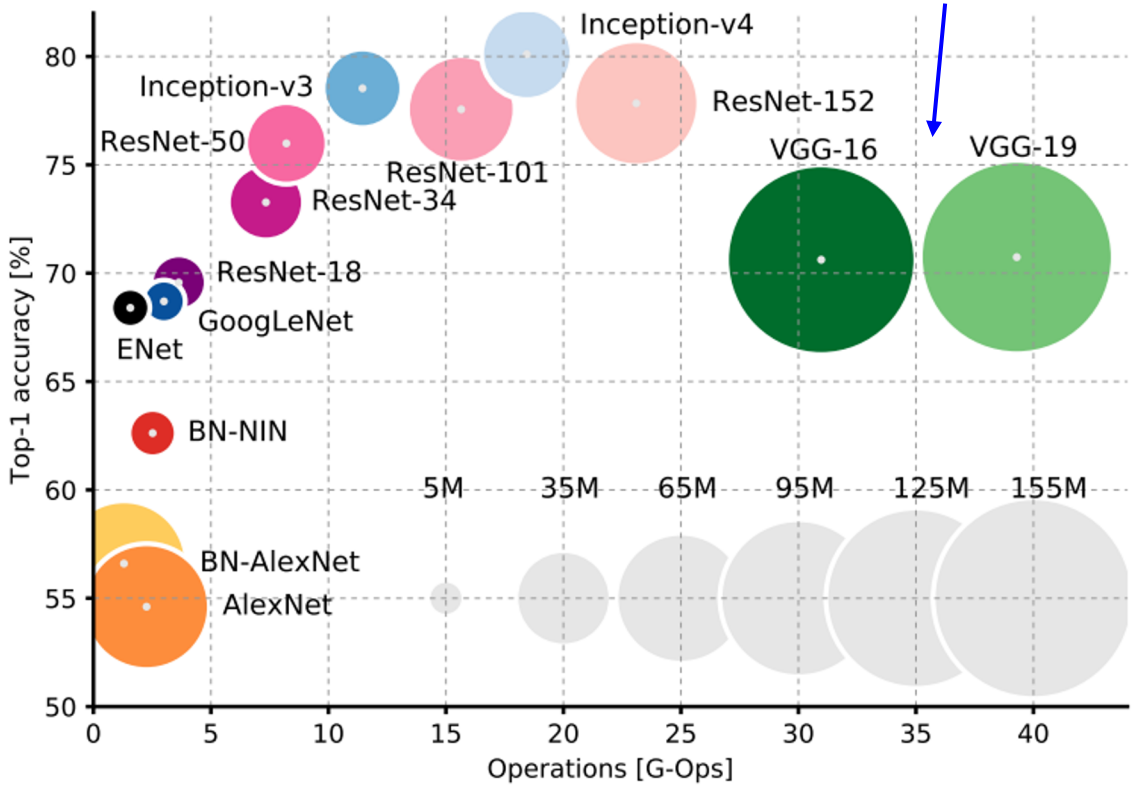
BN: batch normalized version

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

Comparing Complexity



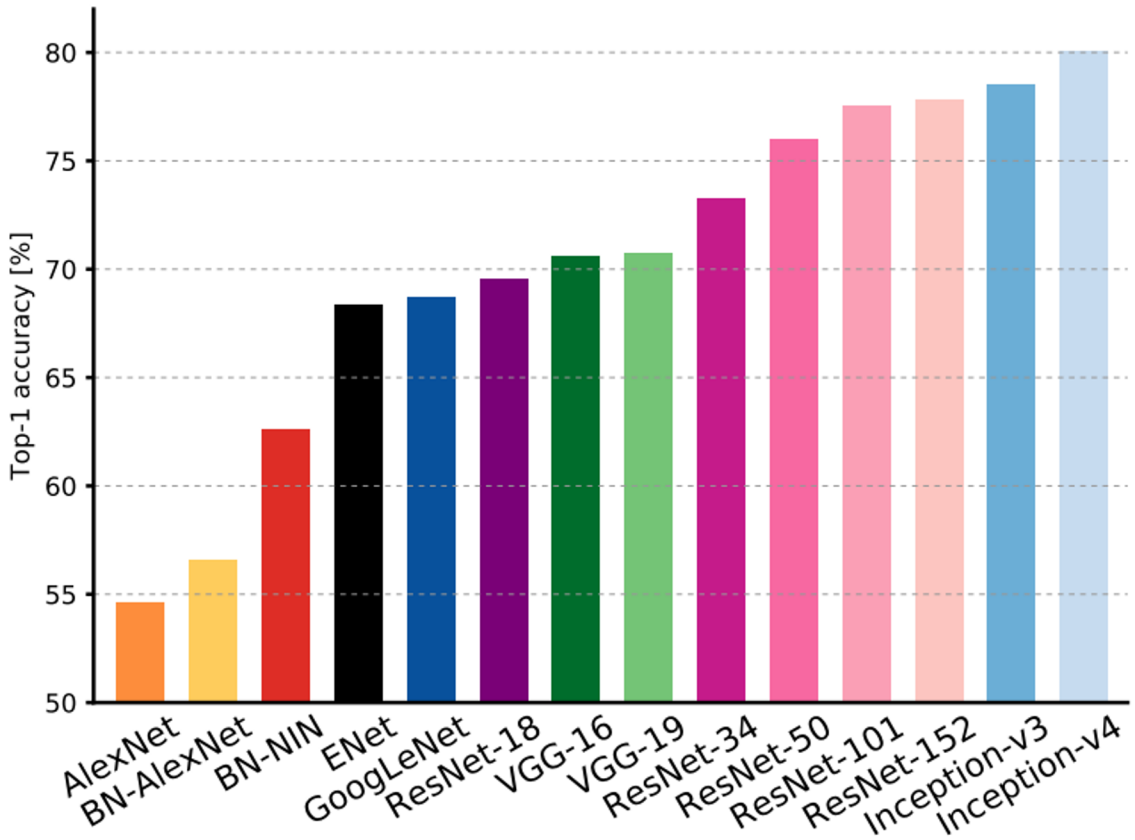
VGG: Highest memory, most operations



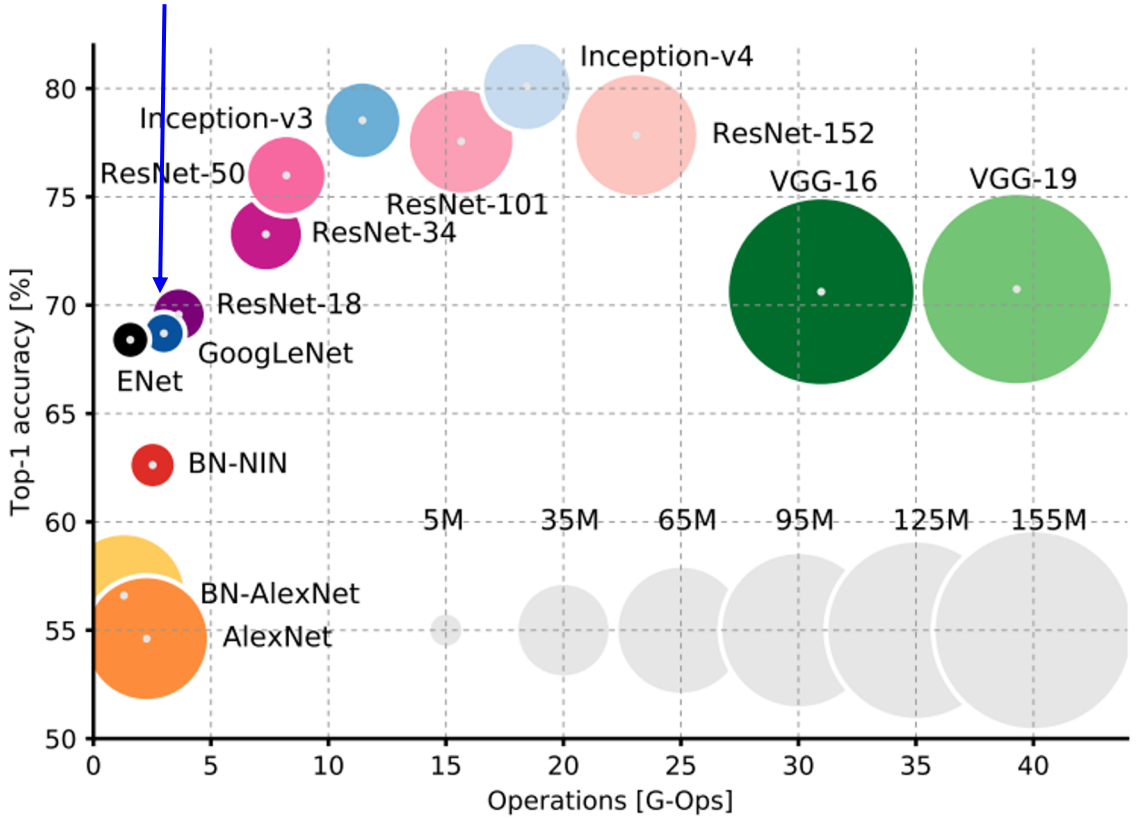
BN: batch normalized version

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

Comparing Complexity



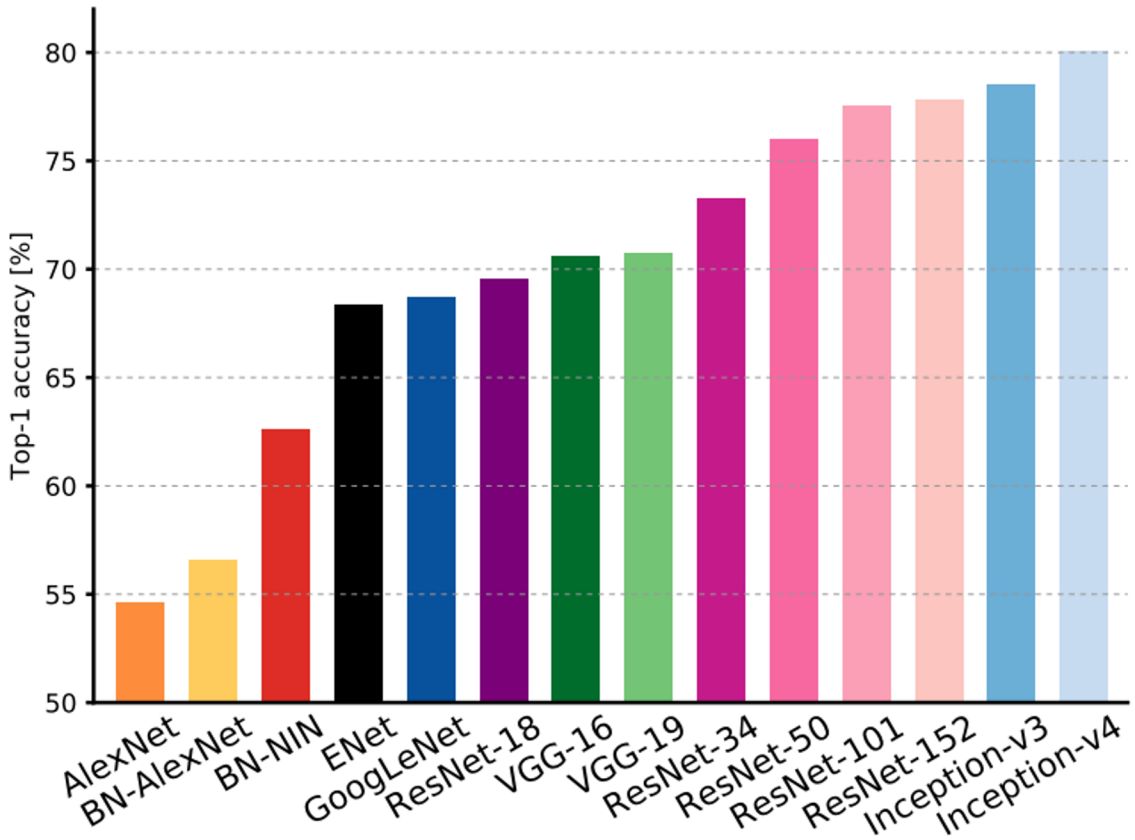
GoogLeNet:
Very efficient!



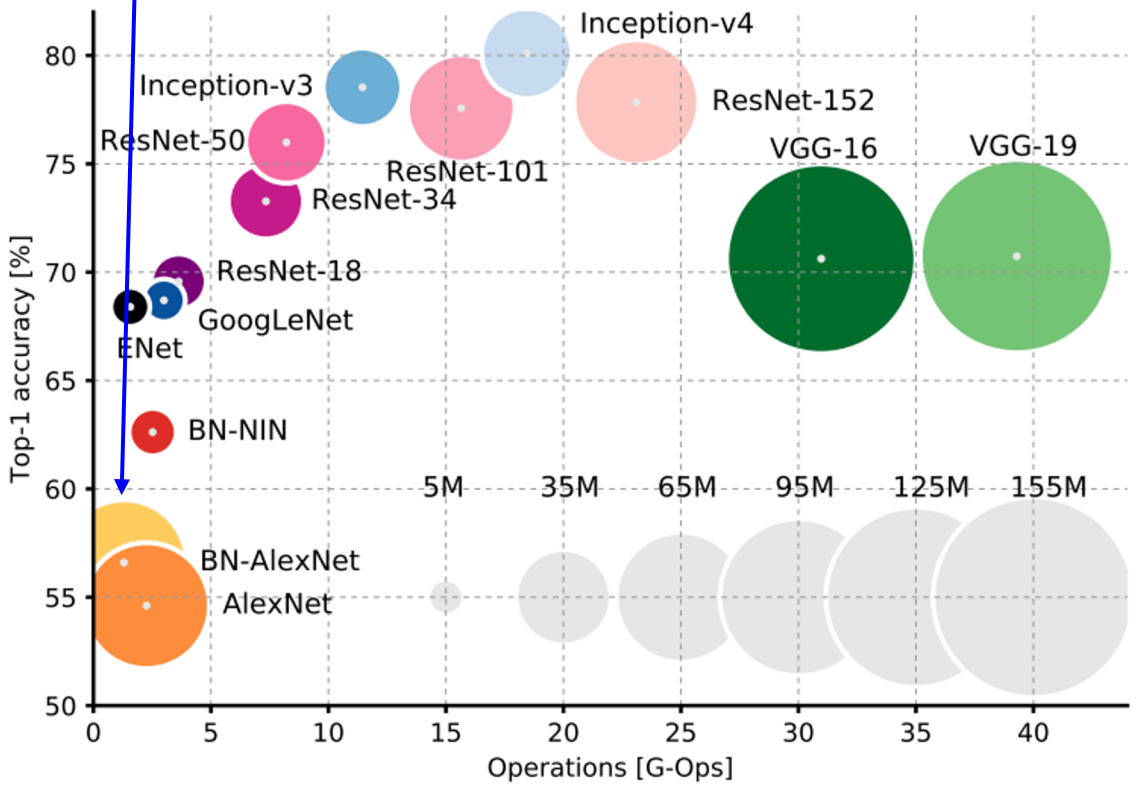
BN: batch normalized version

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

Comparing Complexity



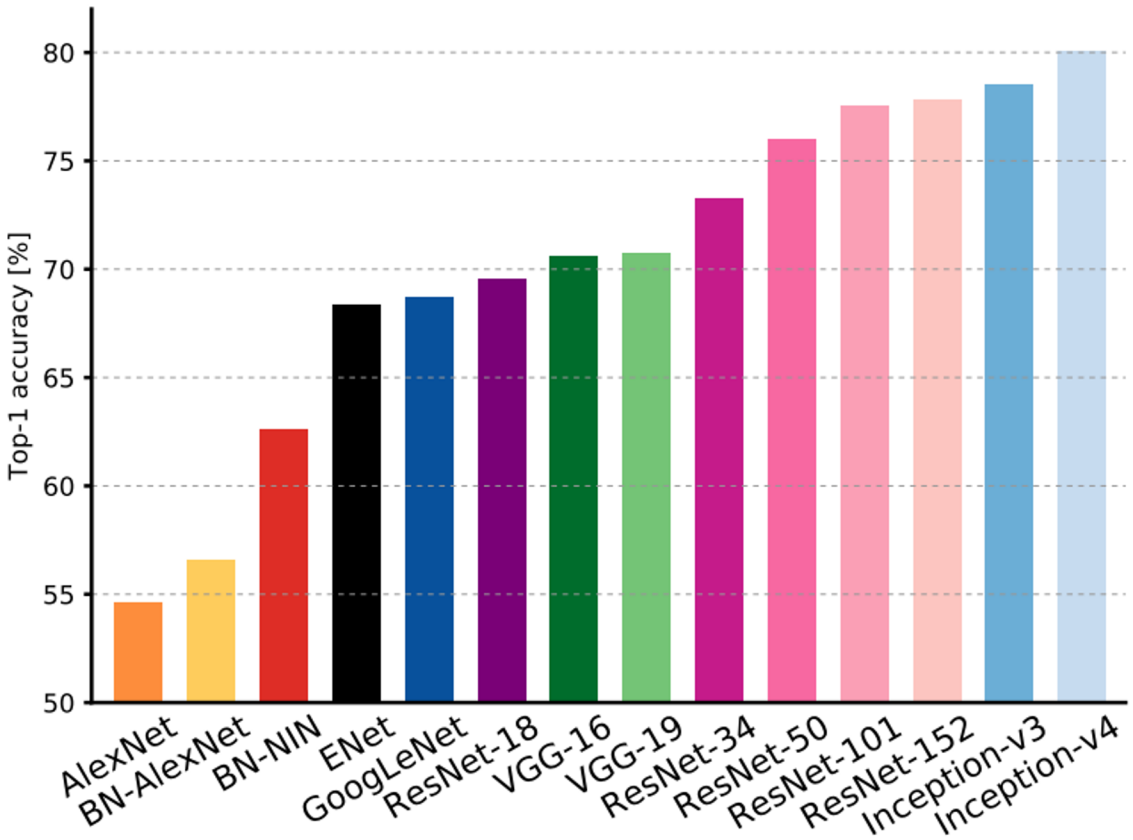
AlexNet: Low compute, lots of parameters



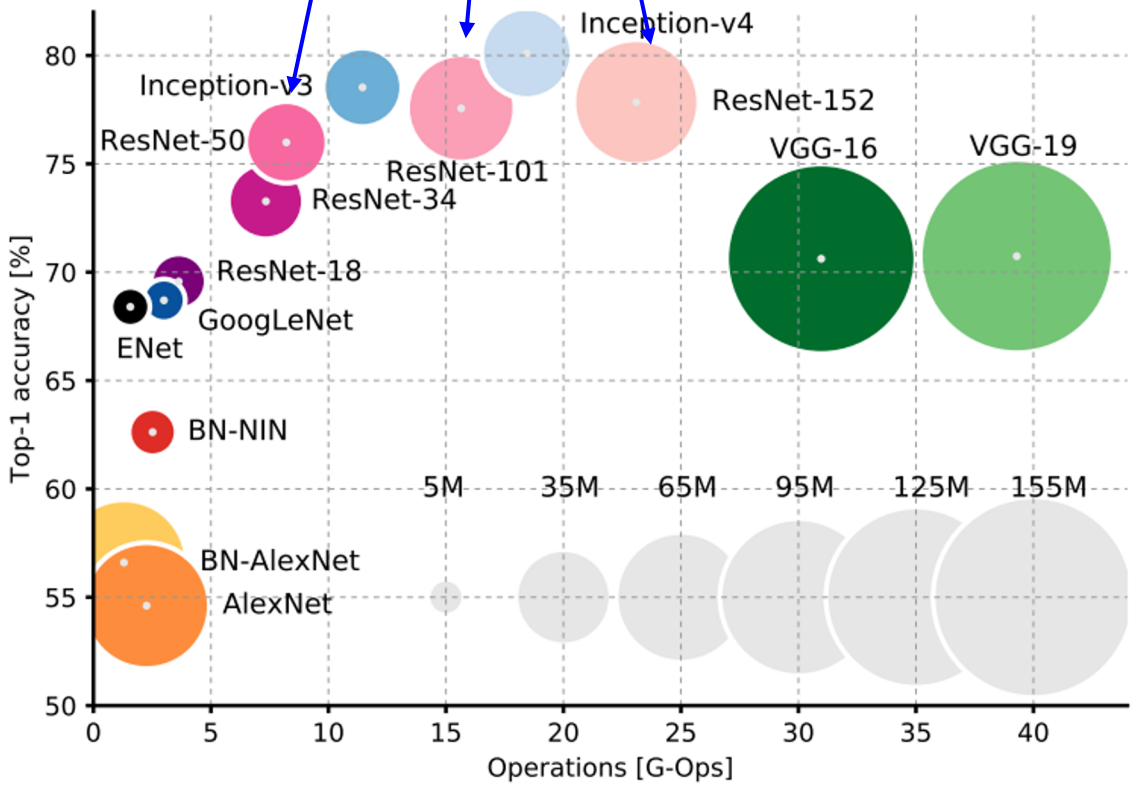
BN: batch normalized version

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

Comparing Complexity



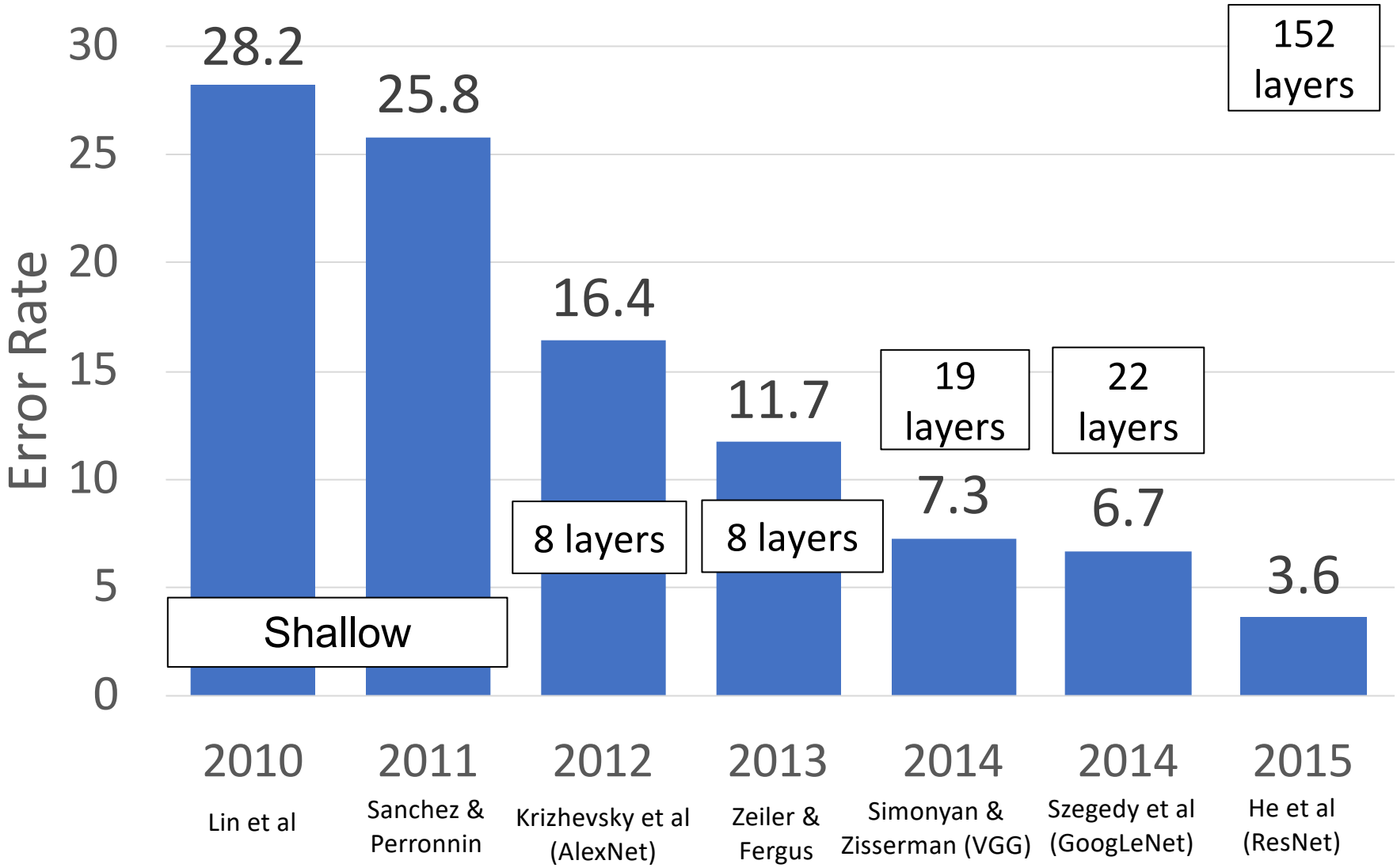
ResNet: Simple design, moderate efficiency, high accuracy



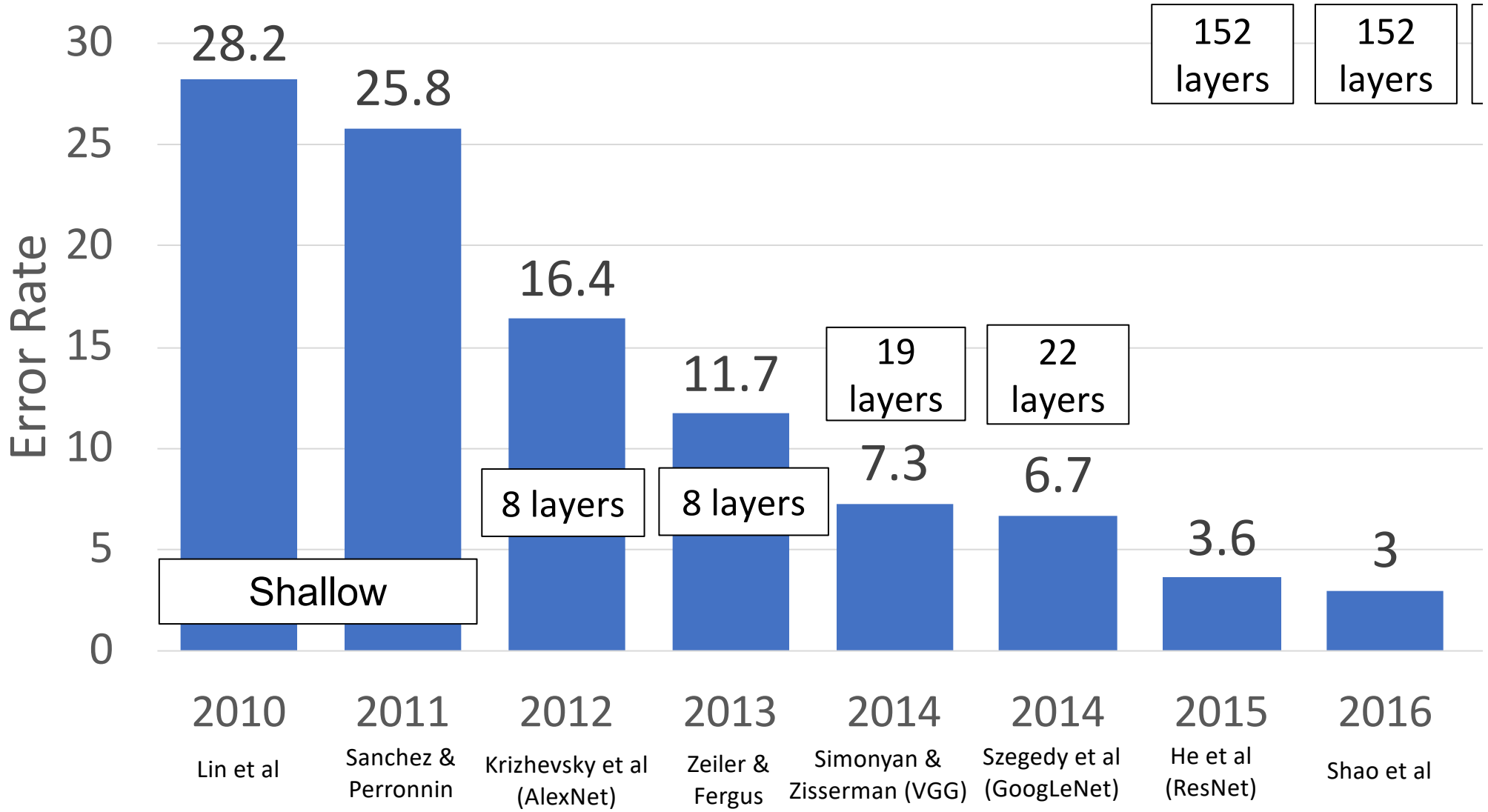
BN: batch normalized version

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

ImageNet Classification Challenge



ImageNet Classification Challenge



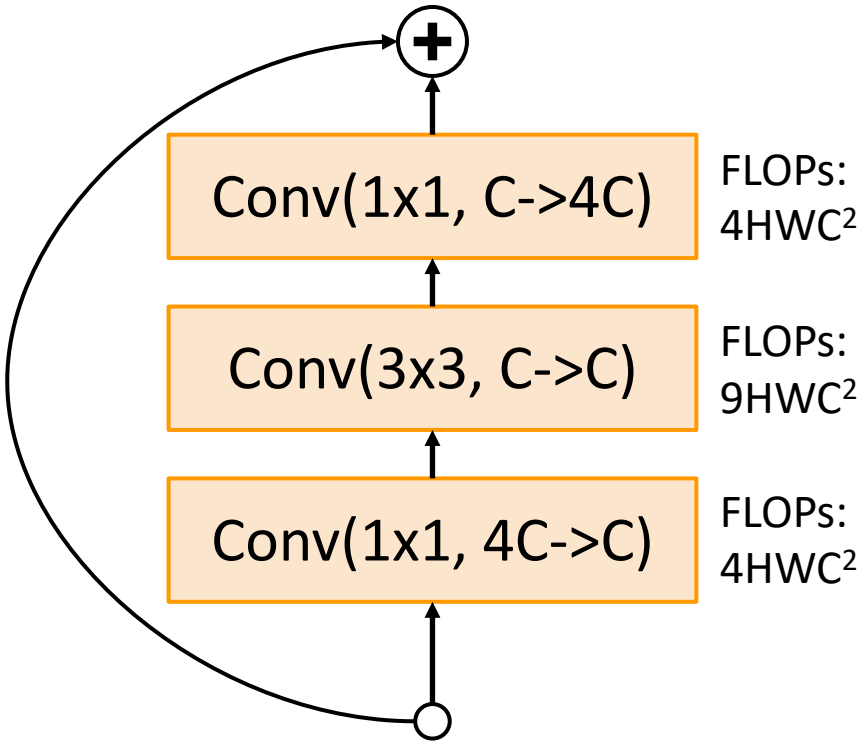
ImageNet 2016 winner: **Model Ensembles**

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

	Inception-v3	Inception-v4	Inception-Resnet-v2	Resnet-200	Wrn-68-3	Fusion（Val.）	Fusion（Test）
Err. (%)	4.20	4.01	3.52	4.26	4.65	2.92 (-0.6)	2.99

Shao et al, 2016

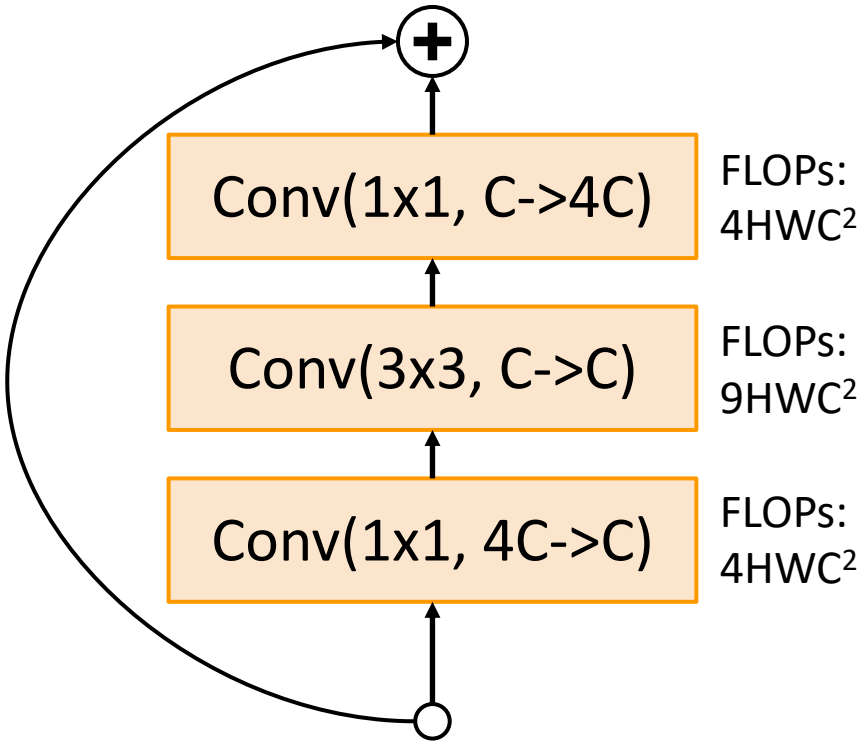
Improving ResNets



"Bottleneck" Residual block

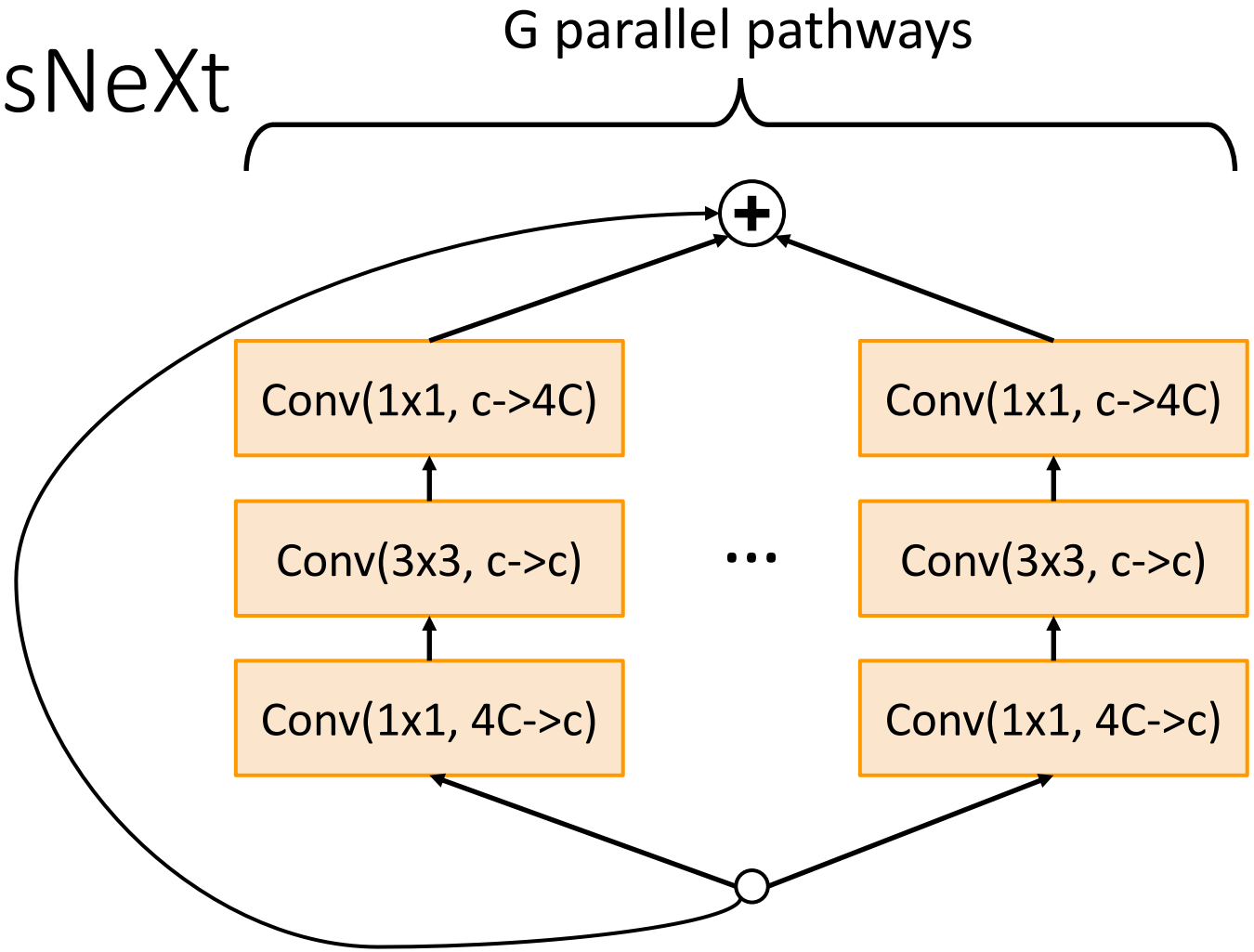
Total FLOPs:
$17HWC^2$

Improving ResNets: ResNeXt



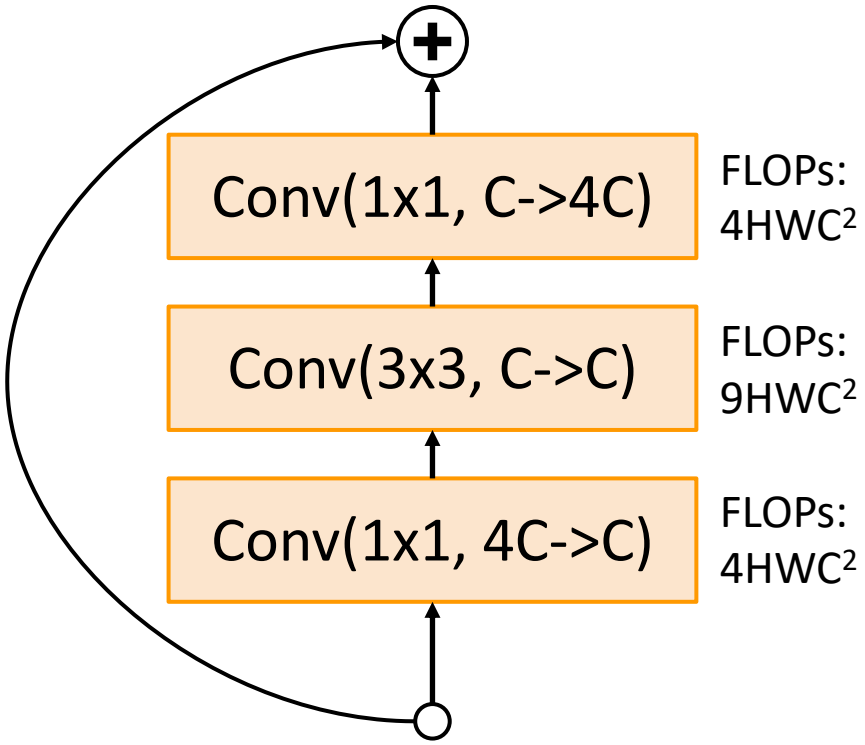
"Bottleneck" Residual block

Total FLOPs:
$17HWC^2$



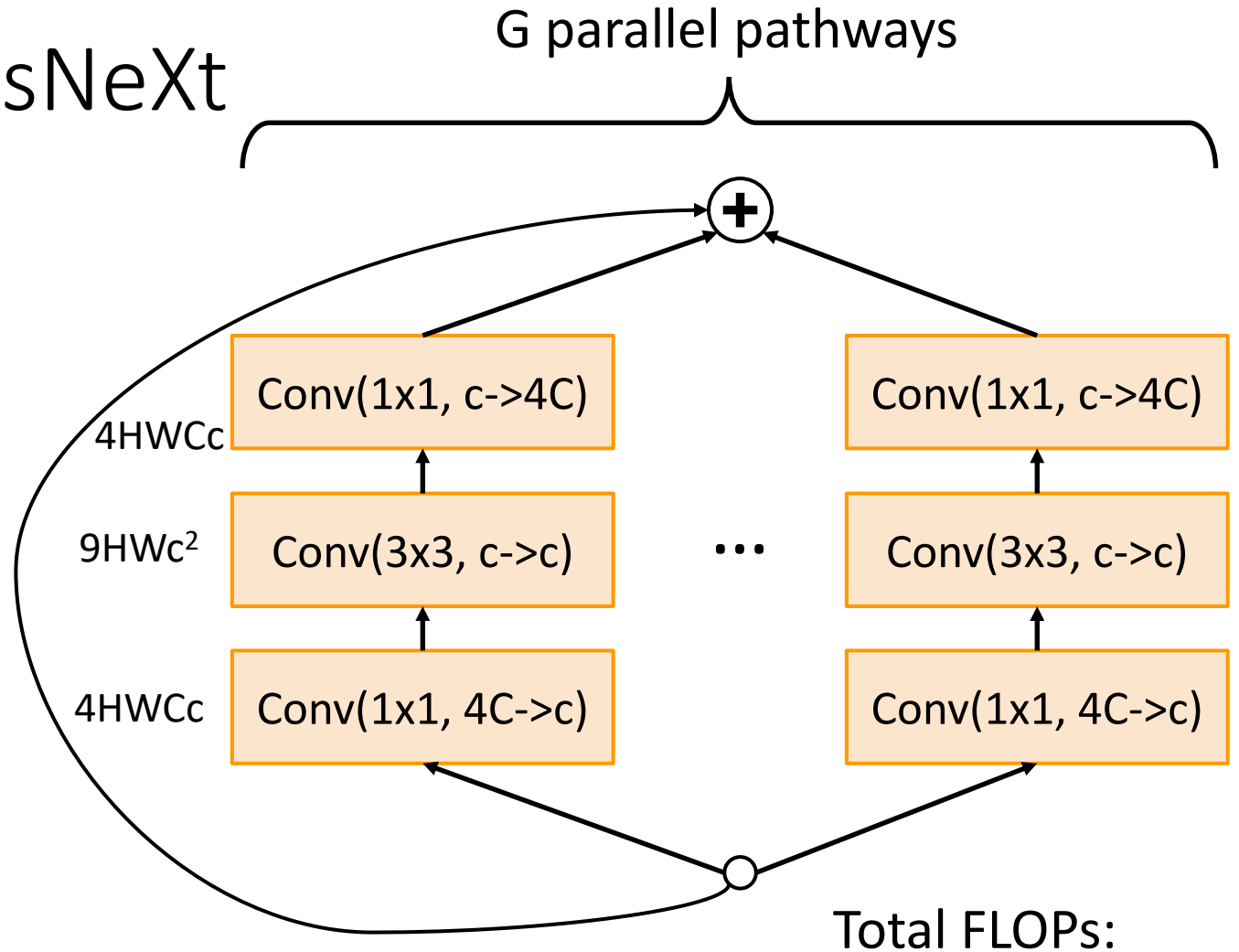
Xie et al, "Aggregated residual transformations for deep neural networks", CVPR 2017

Improving ResNets: ResNeXt



"Bottleneck" Residual block

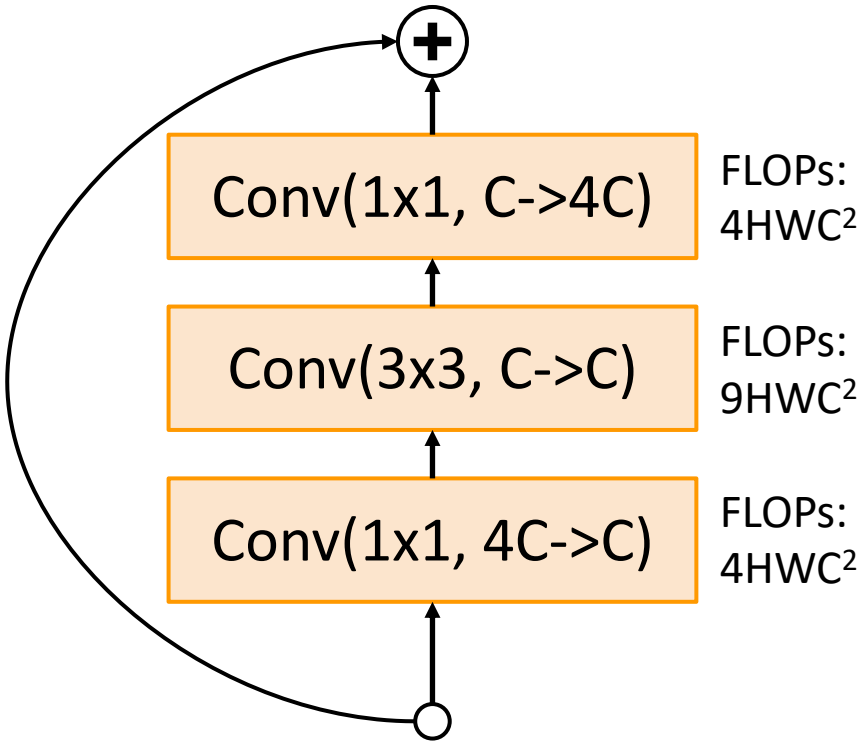
Total FLOPs:
$17HWC^2$



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

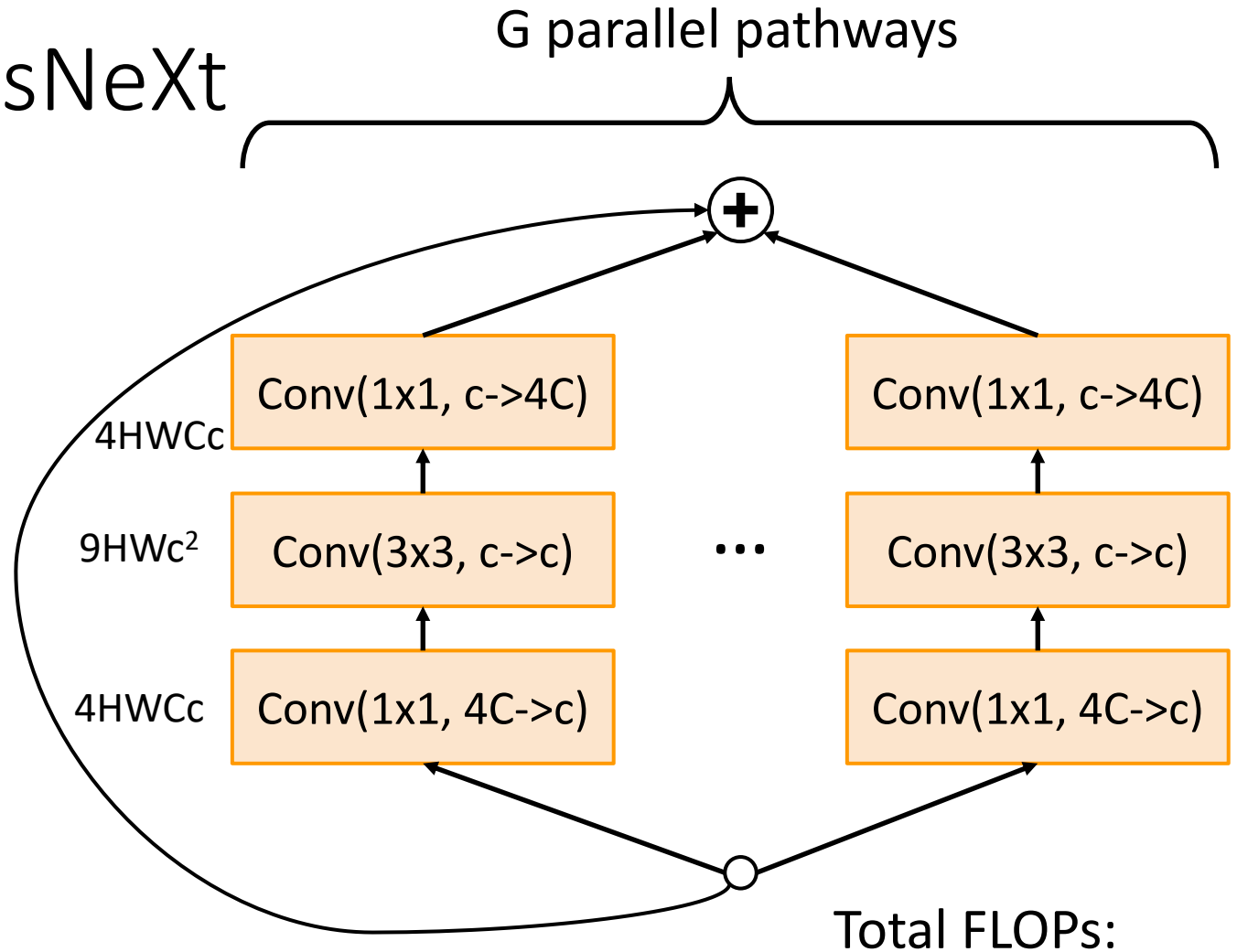
Xie et al, "Aggregated residual transformations for deep neural networks", CVPR 2017

Improving ResNets: ResNeXt



"Bottleneck" Residual block

Total FLOPs: $17HWC^2$



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

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

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

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

Grouped Convolution

Convolution with groups=1:
Normal convolution

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

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

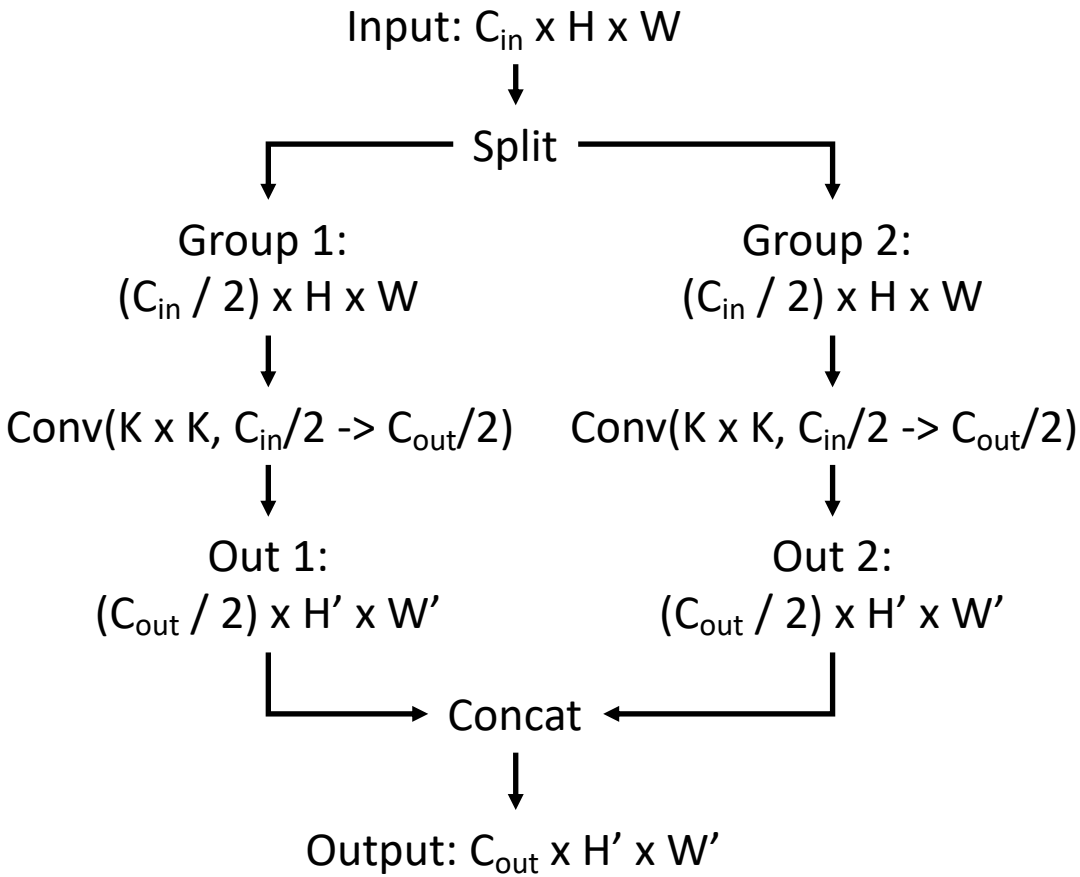
Grouped Convolution

Convolution with groups=1:
Normal convolution

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

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

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



Grouped Convolution

Convolution with groups=1:
Normal convolution

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

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

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

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

Grouped Convolution

Convolution with groups=1:
Normal convolution

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

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

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

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

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

Grouped Convolution in PyTorch

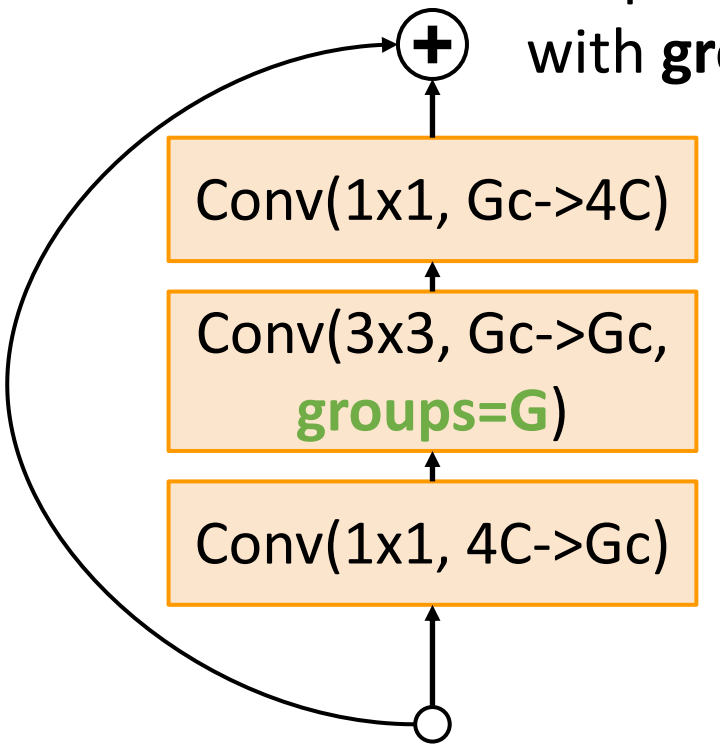
PyTorch convolution gives an option for groups!

Conv2d

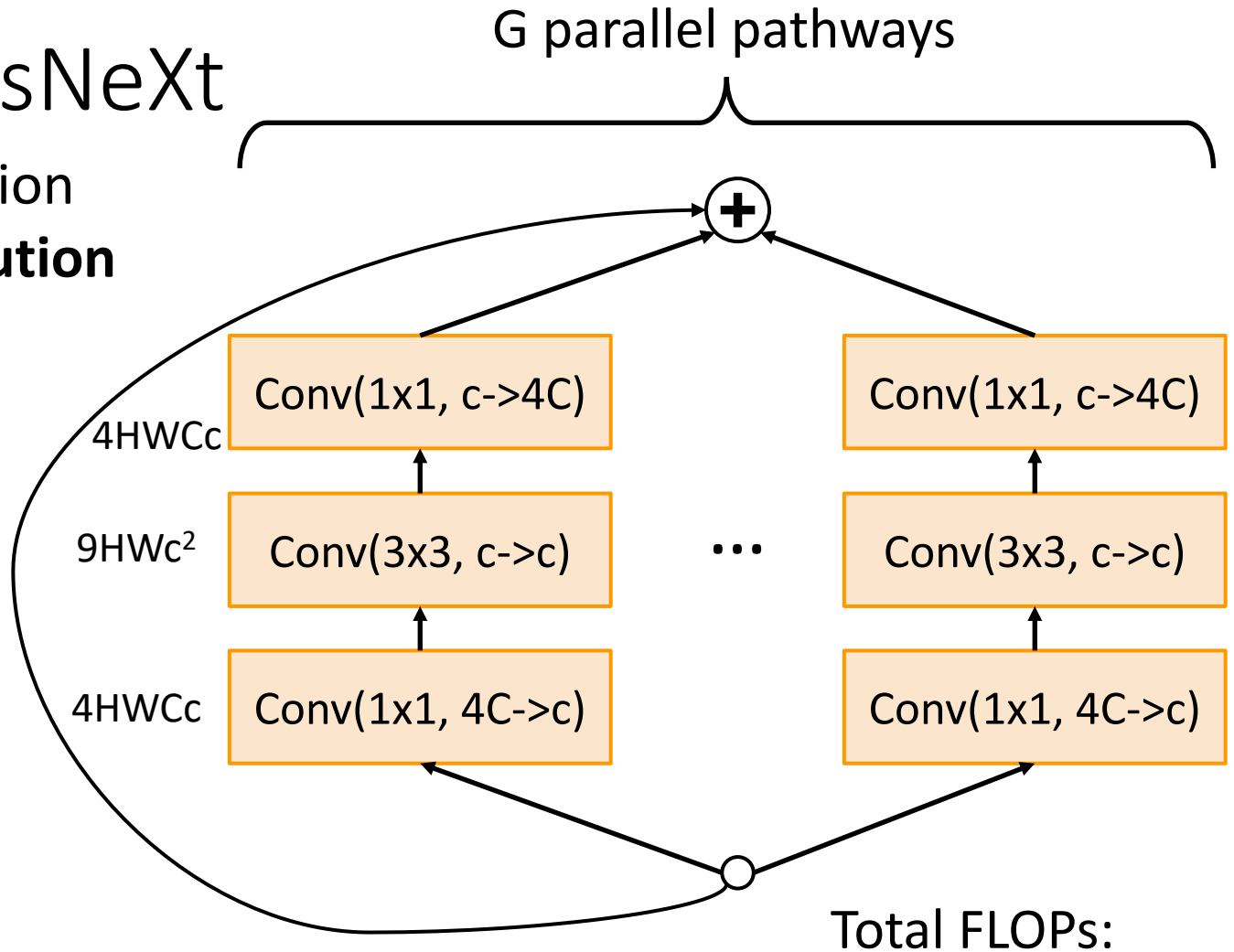
CLASS `torch.nn.Conv2d(`*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros'*`)` [SOURCE]

Improving ResNets: ResNeXt

Equivalent formulation
with **grouped convolution**



ResNeXt block:
Grouped convolution



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

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

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

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

ResNeXt: Maintain computation by adding groups!

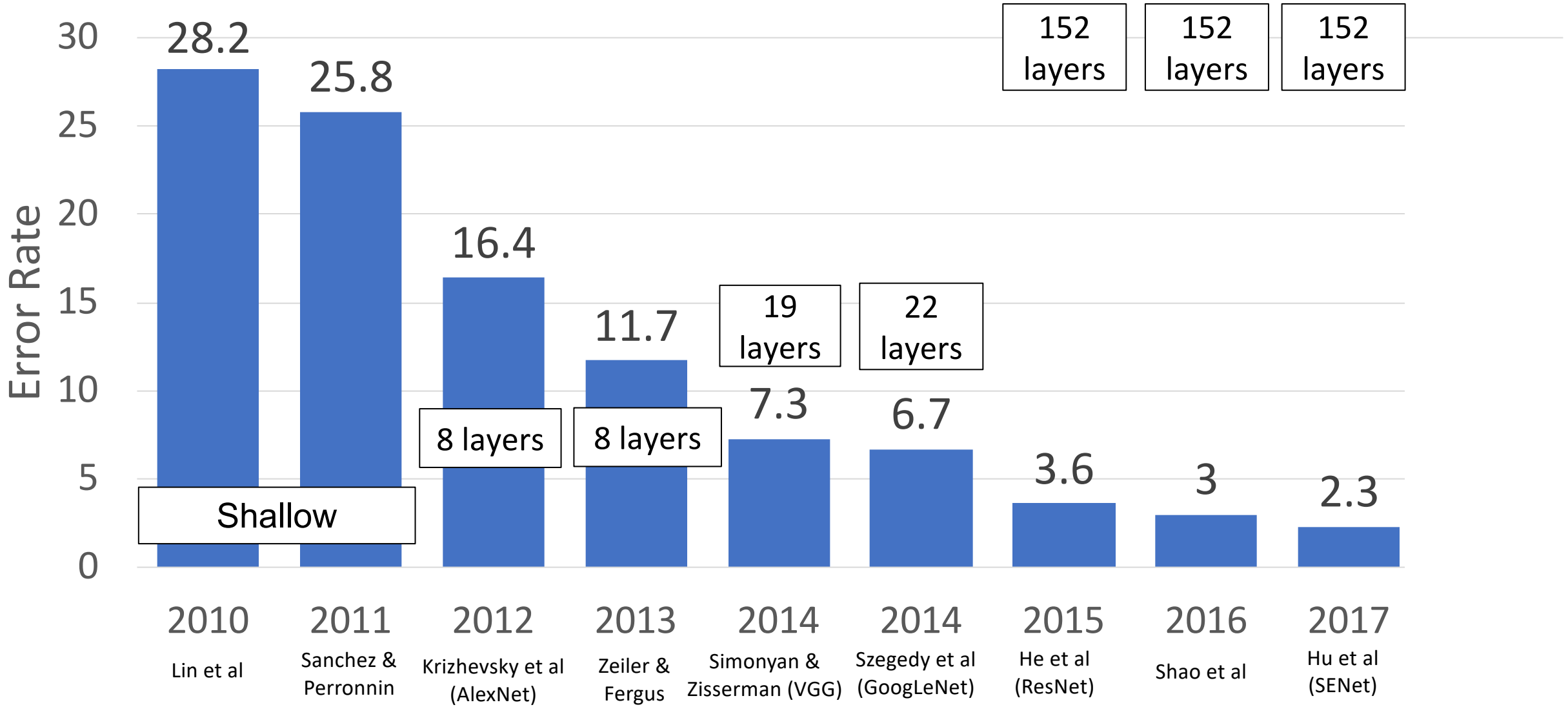
Model	Groups	Group width	Top-1 Error
ResNet-50	1	64	23.9
ResNeXt-50	2	40	23
ResNeXt-50	4	24	22.6
ResNeXt-50	8	14	22.3
ResNeXt-50	32	4	22.2

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

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

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

ImageNet Classification Challenge

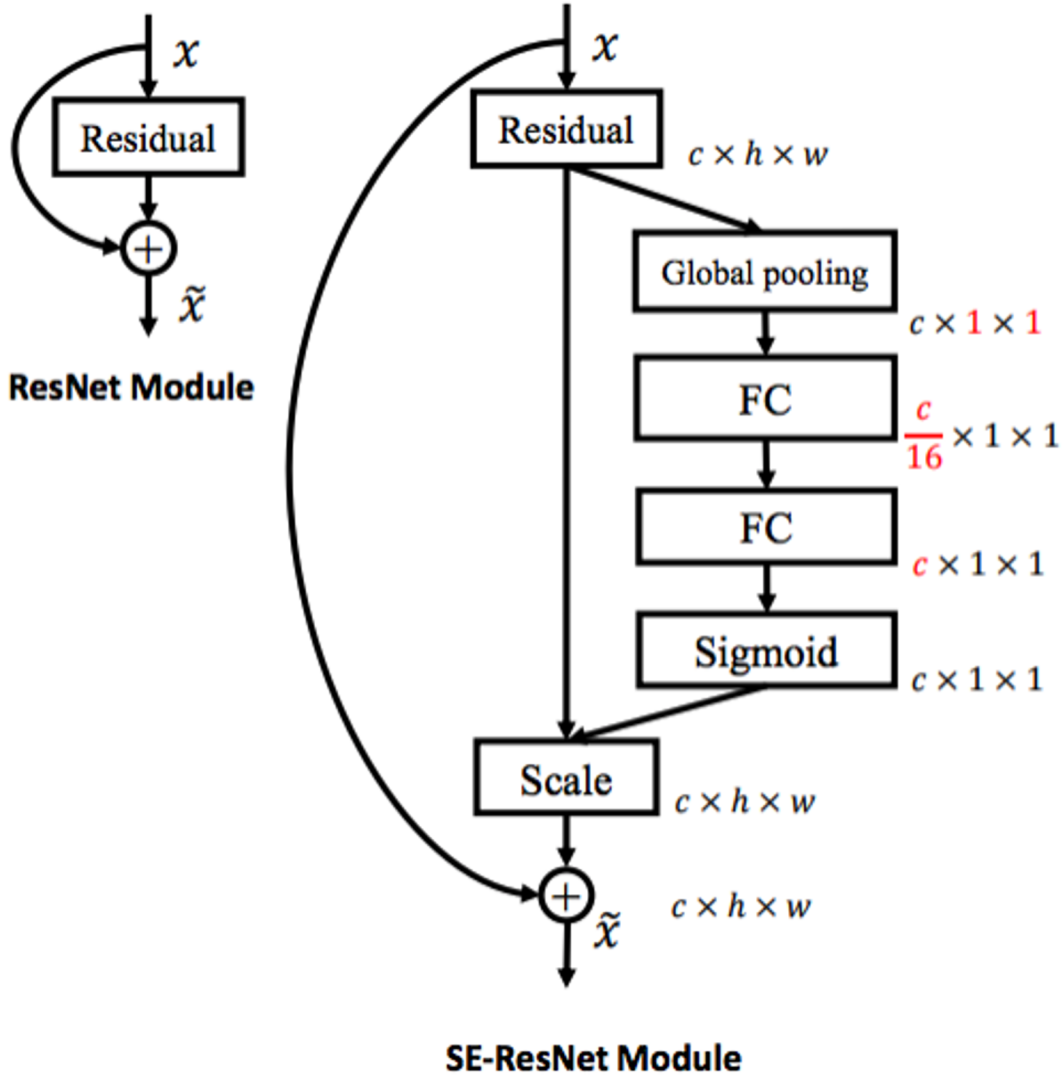


Squeeze-and-Excitation Networks

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

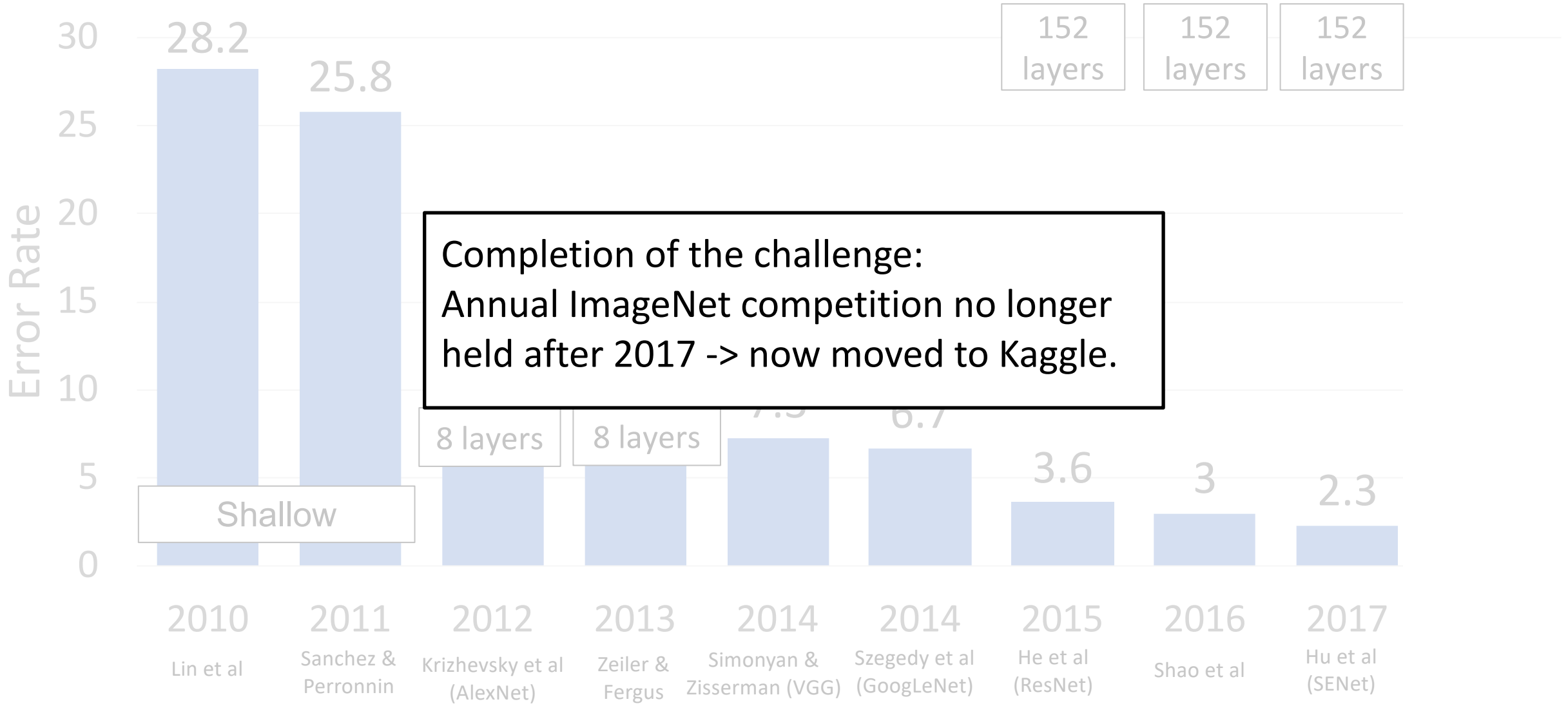
Adds **global context** to each residual block!

Won ILSVRC 2017 with ResNeXt-152-SE



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

ImageNet Classification Challenge

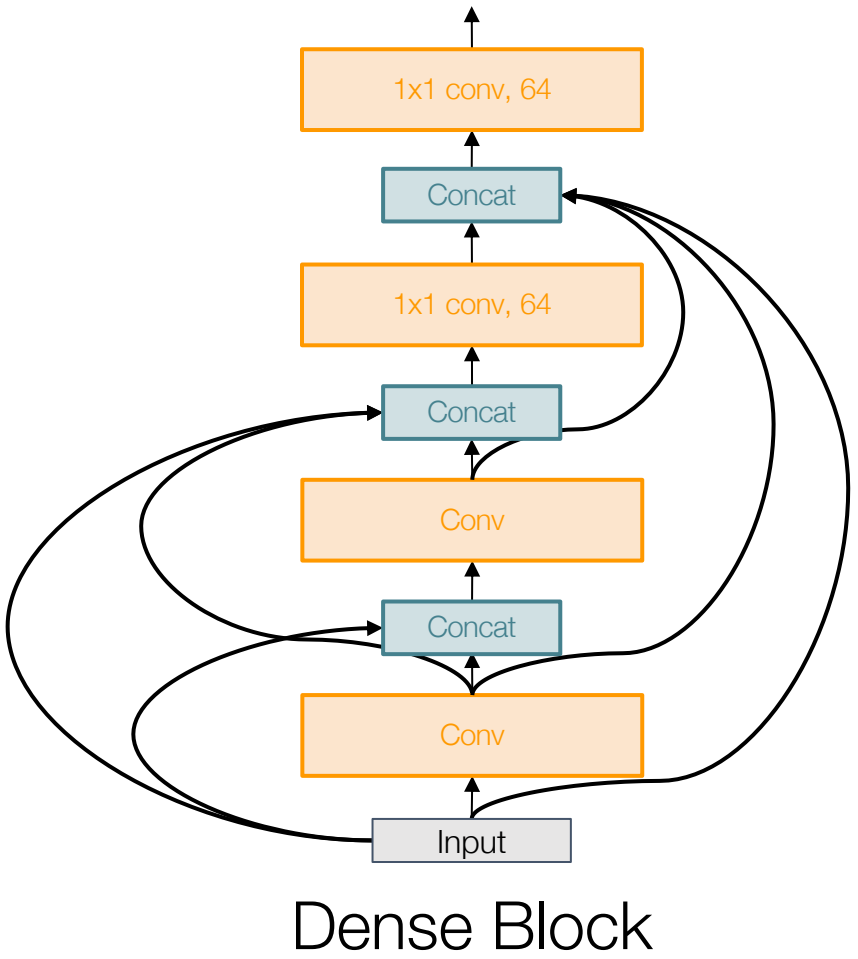


Completion of the challenge:
Annual ImageNet competition no longer held after 2017 -> now moved to Kaggle.

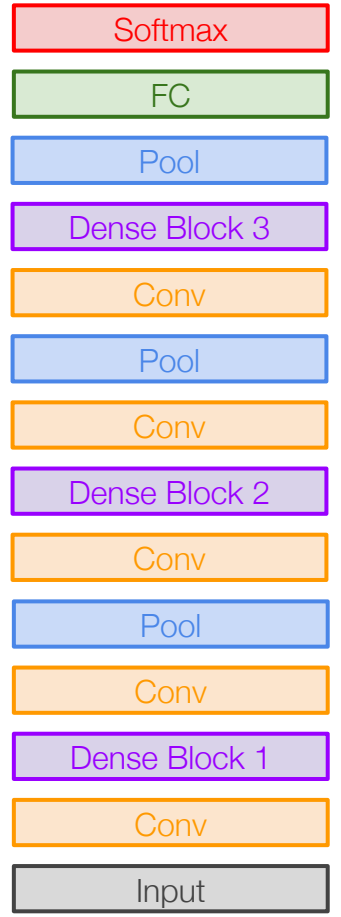
Densely Connected Neural Networks

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

Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse



Dense Block

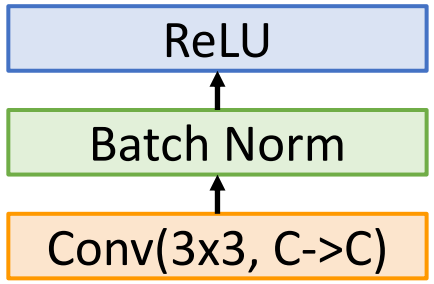


Huang et al, "Densely connected neural networks", CVPR 2017

MobileNets: Tiny Networks (For Mobile Devices)

Standard Convolution Block

Total cost: $9C^2HW$



$9C^2HW$

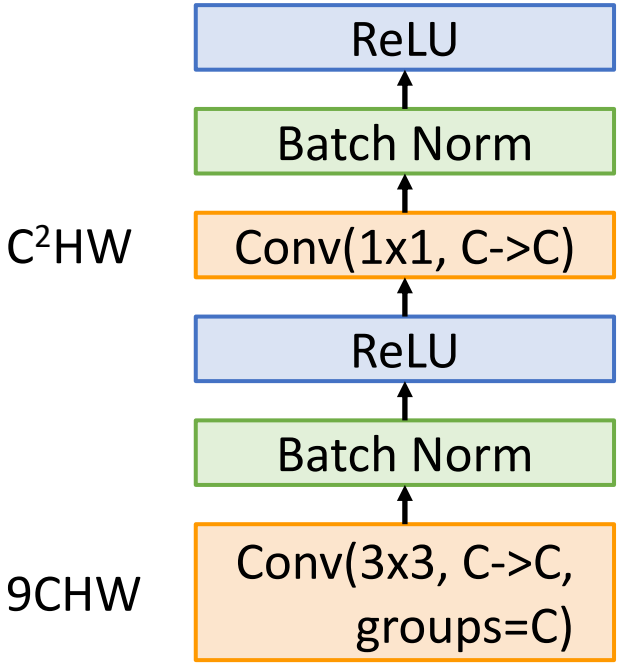
Speedup = $9C^2/(9C+C^2)$

= $9C/(9+C)$

=> 9 (as C->inf)

Depthwise Separable Convolution

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



C^2HW "Pointwise Convolution"

9CHW "Depthwise Convolution"

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

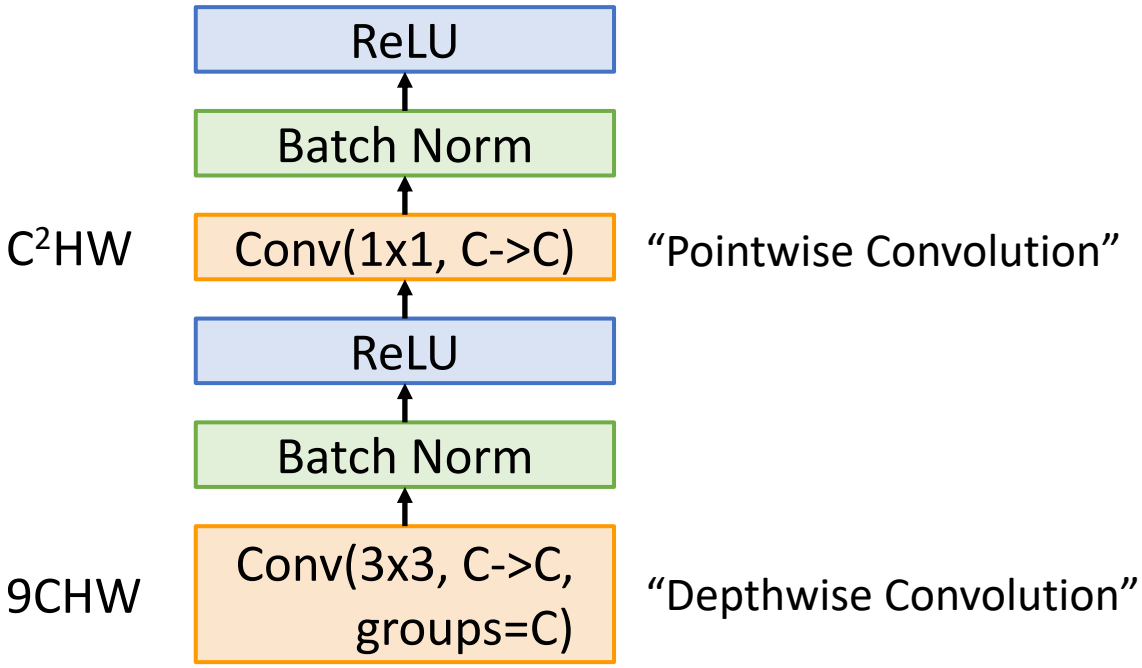
MobileNets: Tiny Networks (For Mobile Devices)

Also related:

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

Depthwise Separable Convolution

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

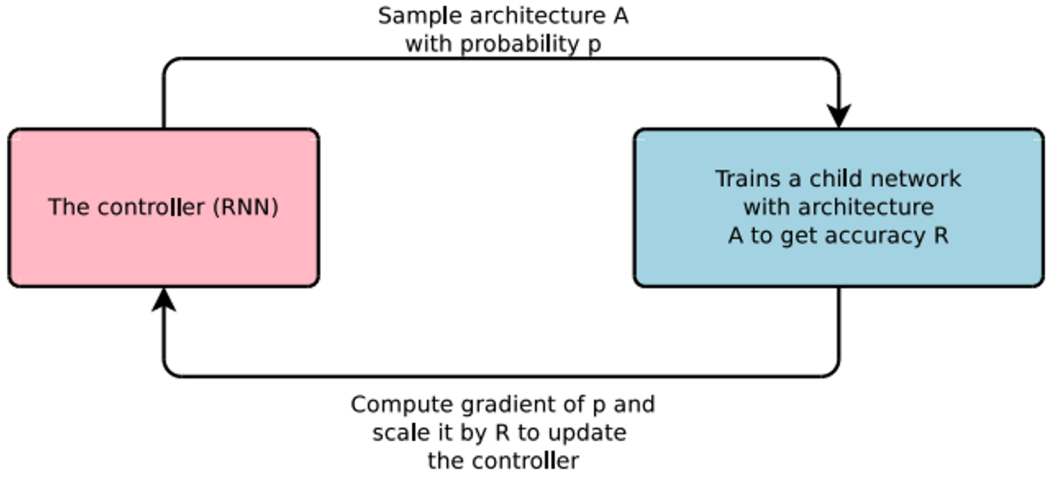


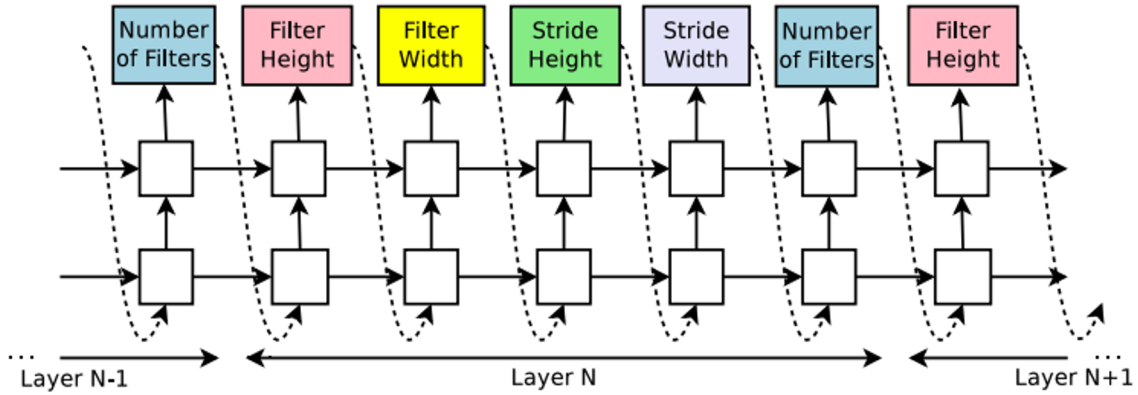
Howard et al, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", 2017

Neural Architecture Search

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

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



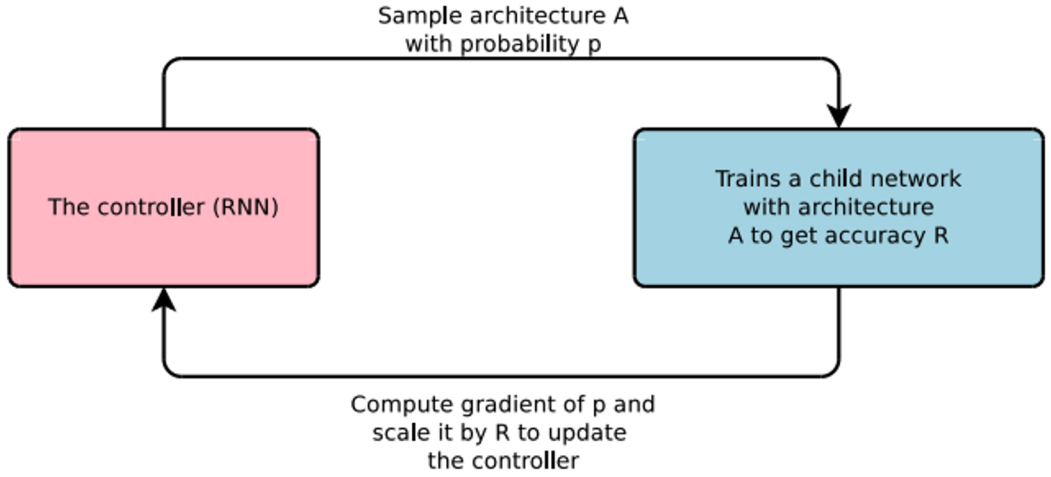


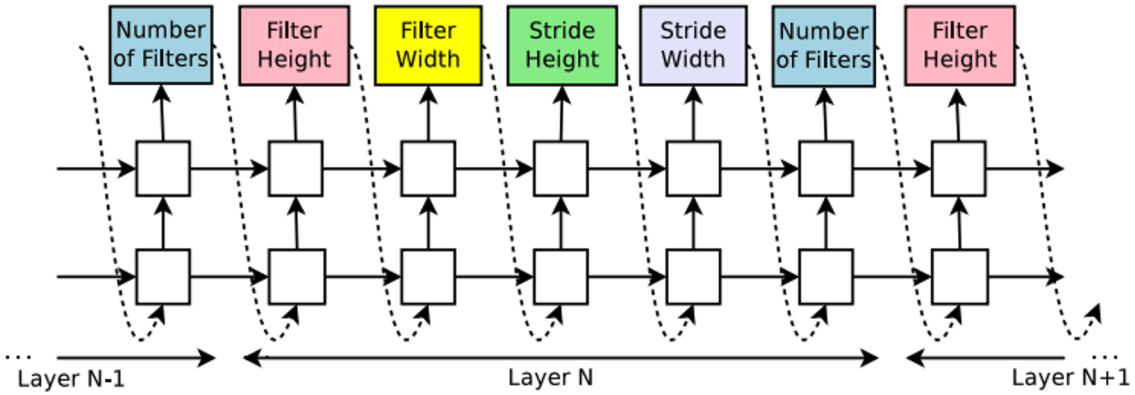
Zoph and Le, "Neural Architecture Search with Reinforcement Learning", ICLR 2017

Neural Architecture Search

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

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

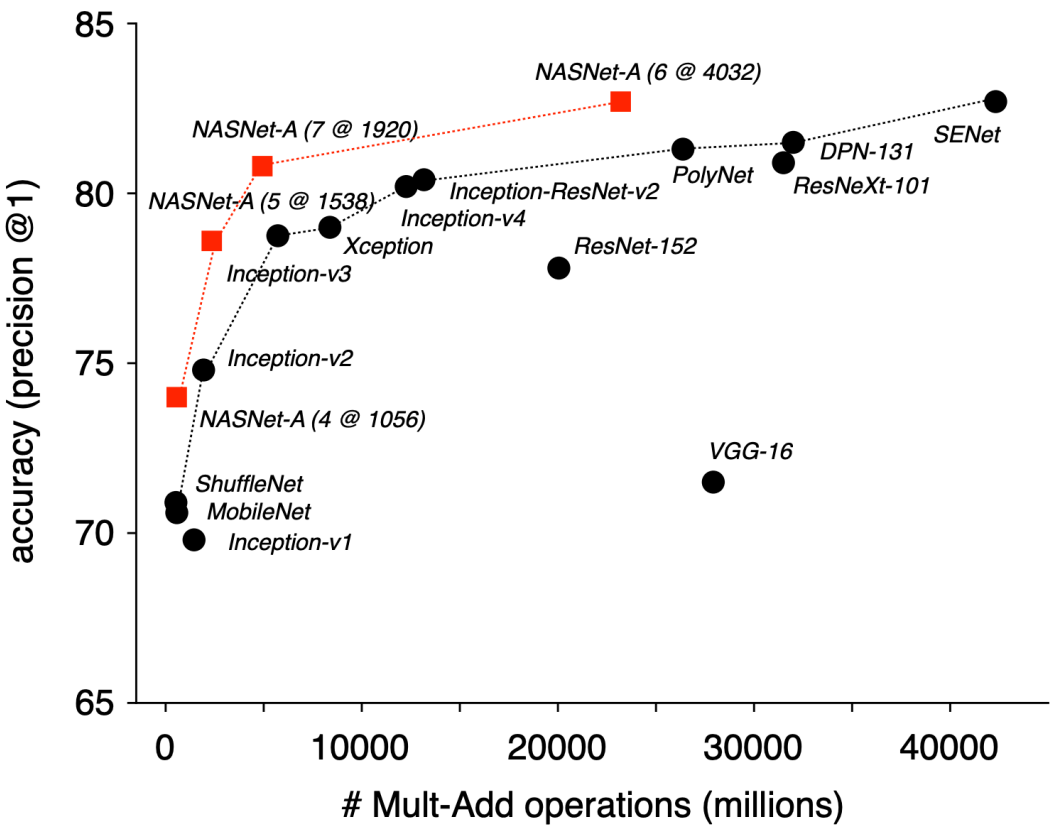


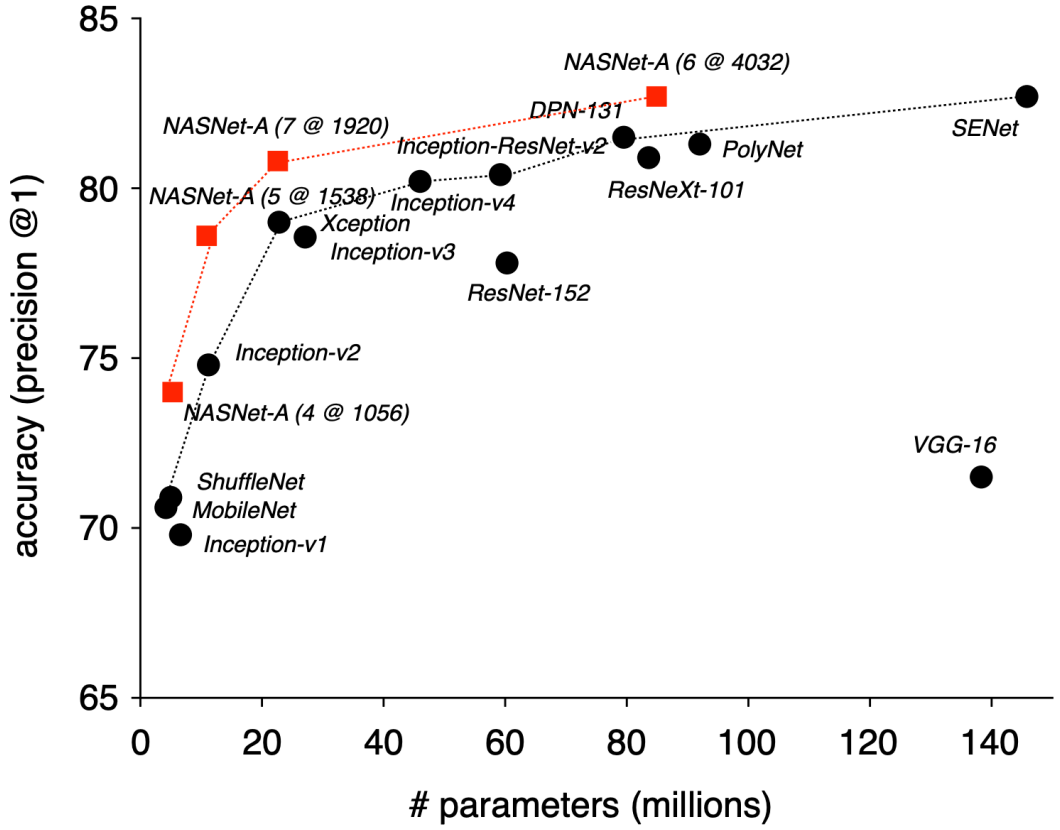


Zoph and Le, "Neural Architecture Search with Reinforcement Learning", ICLR 2017

Neural Architecture Search

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





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

CNN Architectures Summary

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

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

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

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

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

Neural Architecture Search promises to automate architecture design

Which Architecture should I use?

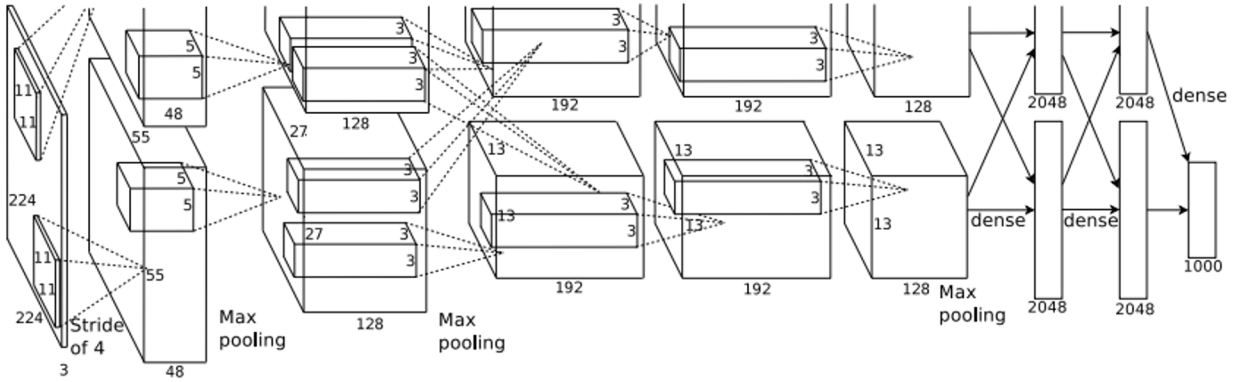
Don’t be a hero. For most problems you should use an off-the-shelf architecture; don’t try to design your own!

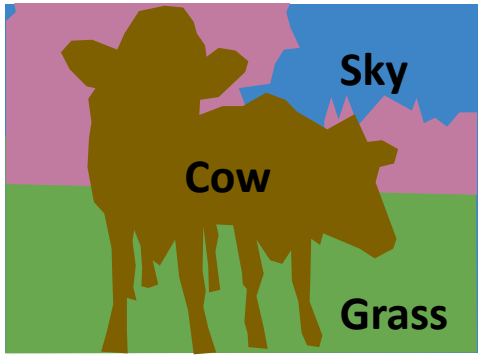
If you just care about accuracy, **ResNet-50** or **ResNet-101** are great choices

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

Network architectures

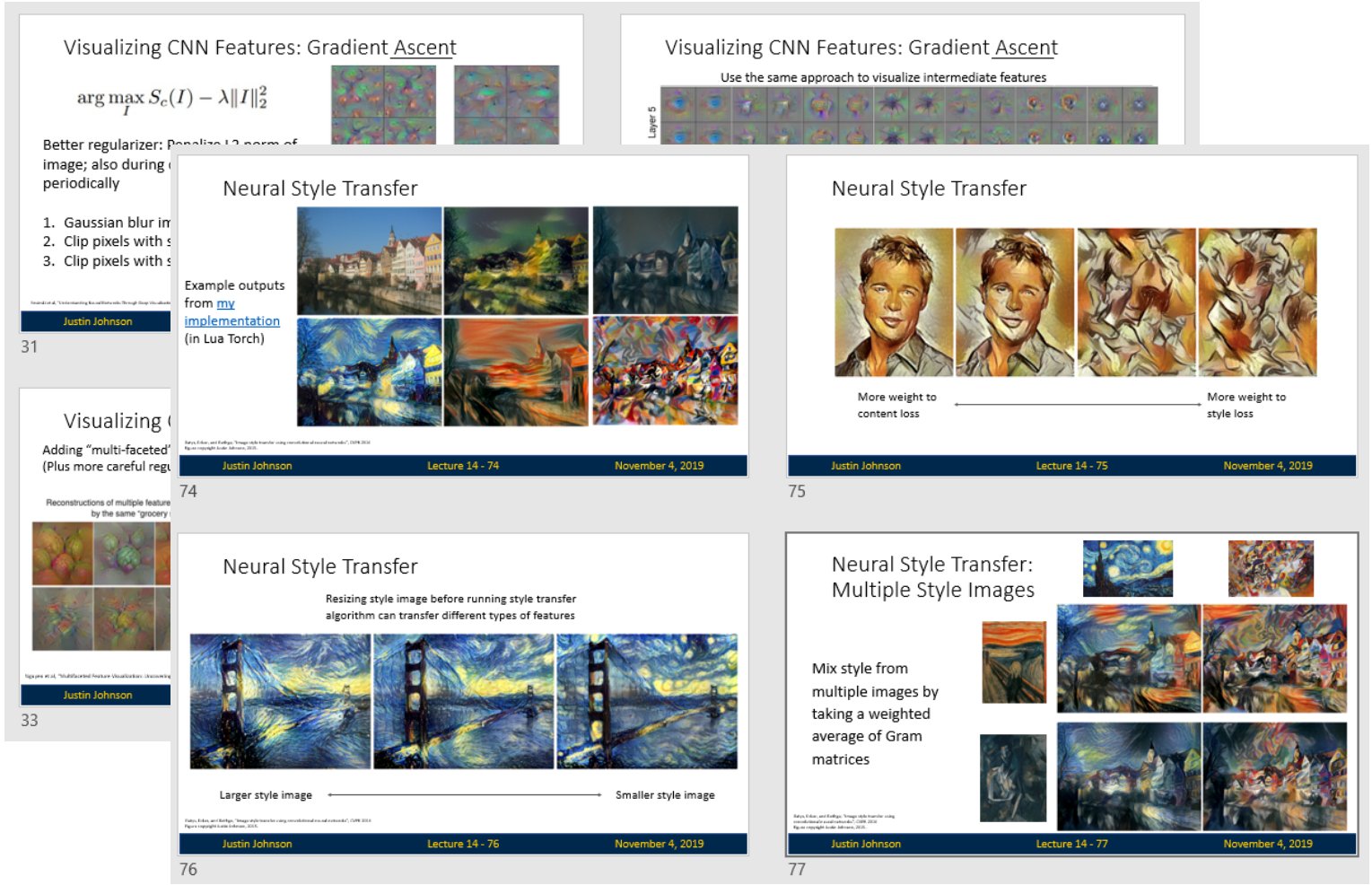
- Batch normalization
- CNN architectures
- **Visualization**
- Adversarial examples
- Deconvolution and U-Nets
- ~~Deep learning software~~



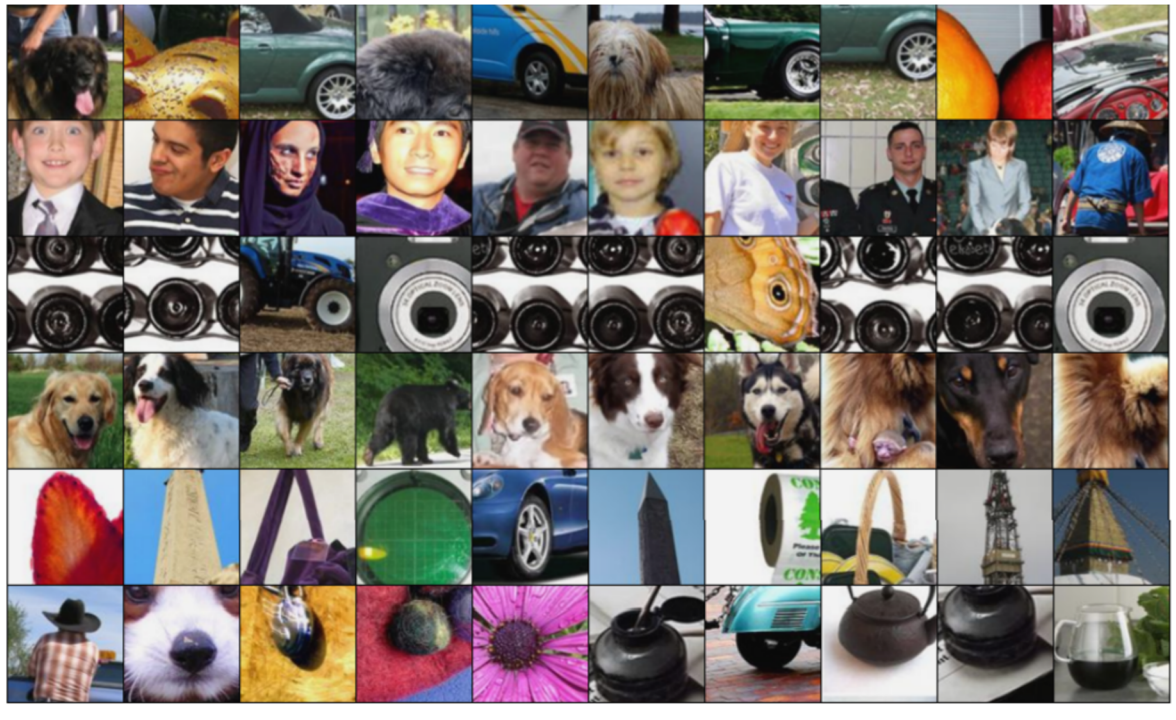


Lecture 14: Visualizing CNNs

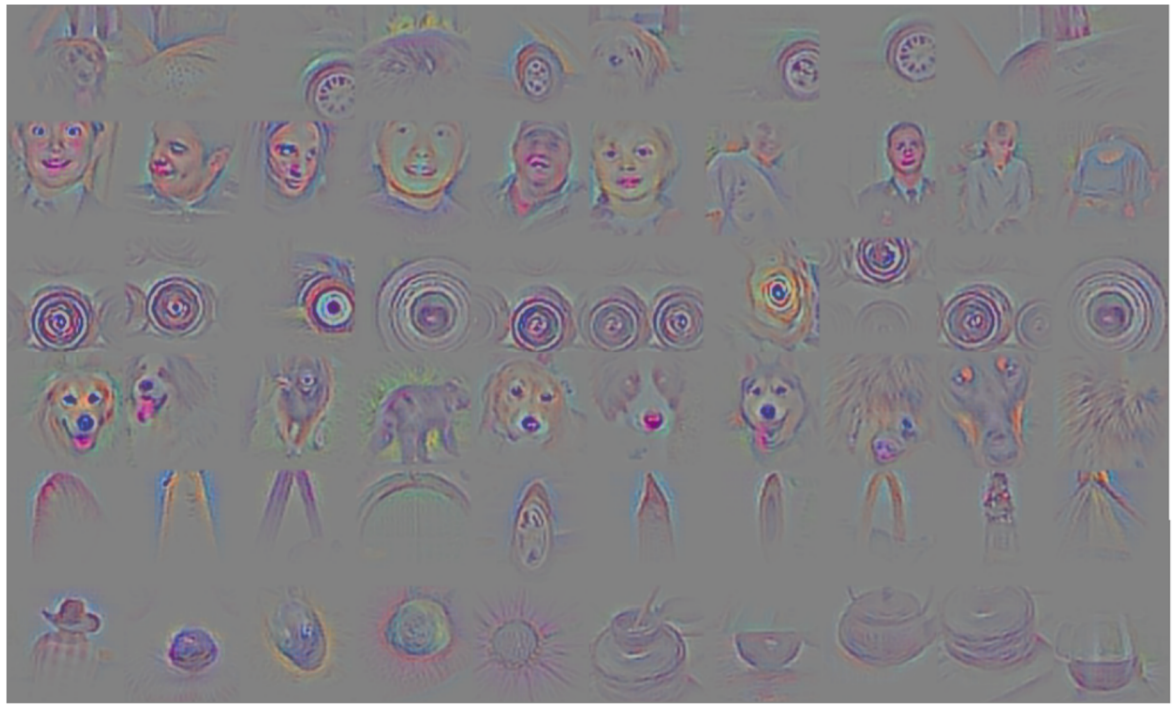
(see slides on Web site)



Intermediate Features via (guided) backprop



Maximally activating patches
(Each row is a different neuron)



Guided Backprop

Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014
Springenberg et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015

Visualizing CNN Features: Gradient Ascent

(Guided) backprop:
Find the part of an image that a neuron responds to

Gradient ascent:
Generate a synthetic image that maximally activates a neuron

$$I^* = \arg\max_I f(I) + R(I)$$

Neuron value

Natural image regularizer

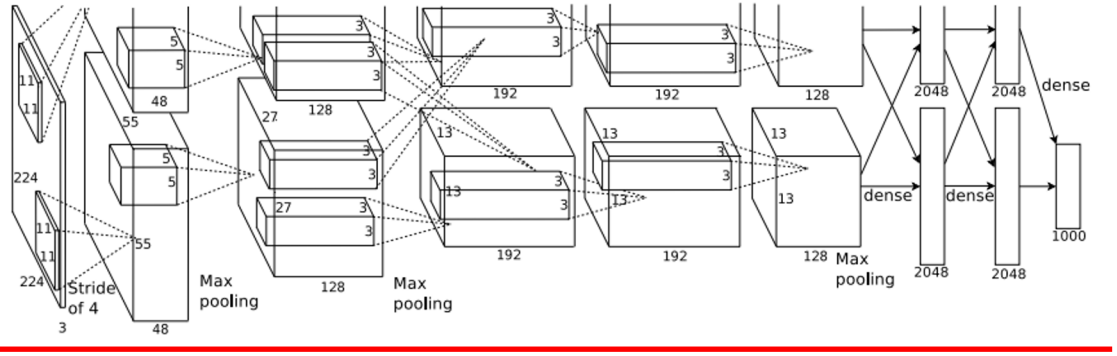
Visualizing CNN Features: Gradient Ascent

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

score for class c (before Softmax)

1. Initialize image to zeros

zero image



Repeat:

2. Forward image to compute current scores
3. Backprop to get gradient of neuron value with respect to image pixels
4. Make a small update to the image

Visualizing CNN Features: Gradient Ascent

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

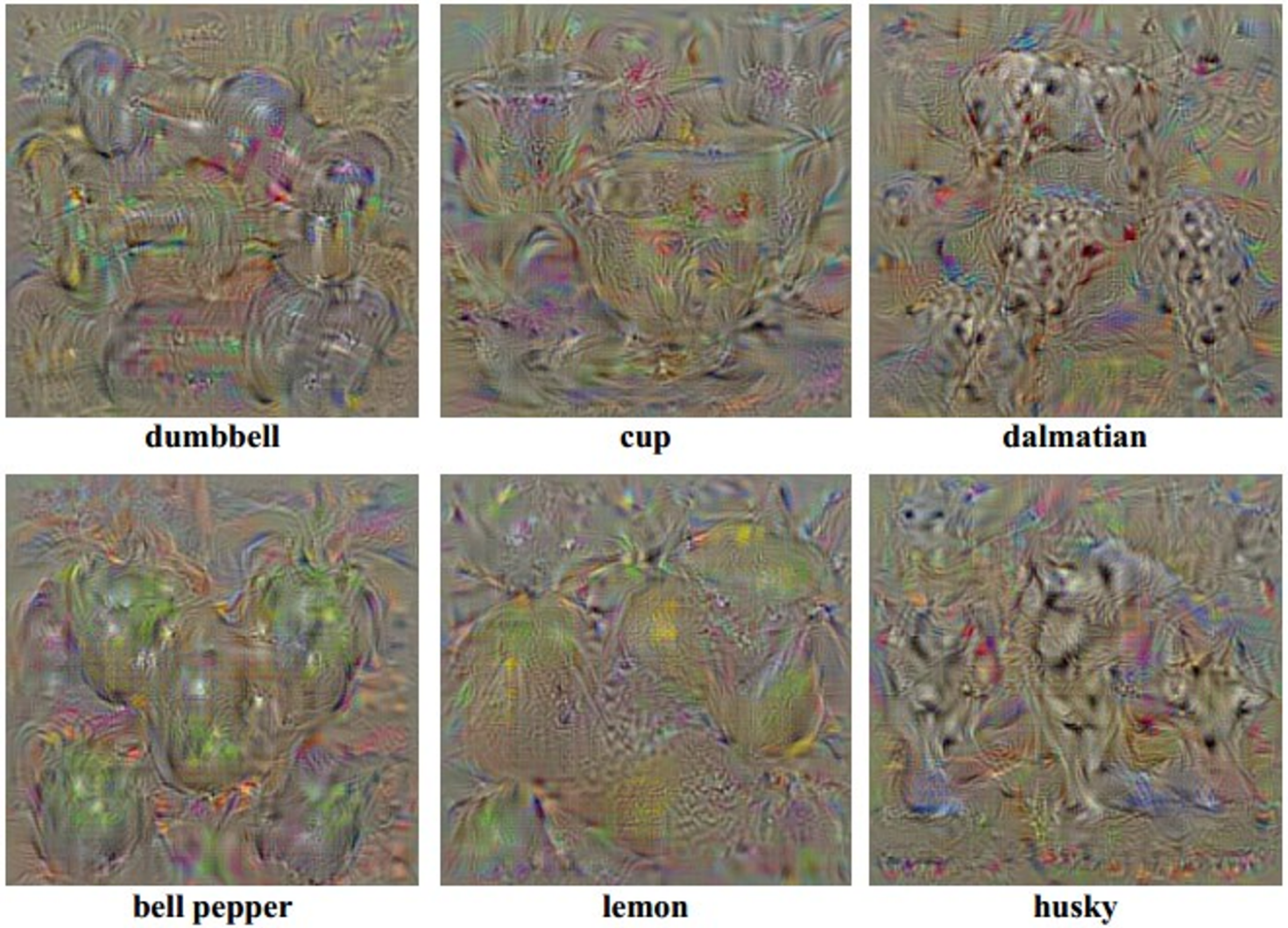
Simple regularizer: Penalize
L2 norm of generated image

Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.

Visualizing CNN Features: Gradient Ascent

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

Simple regularizer: Penalize
L2 norm of generated image

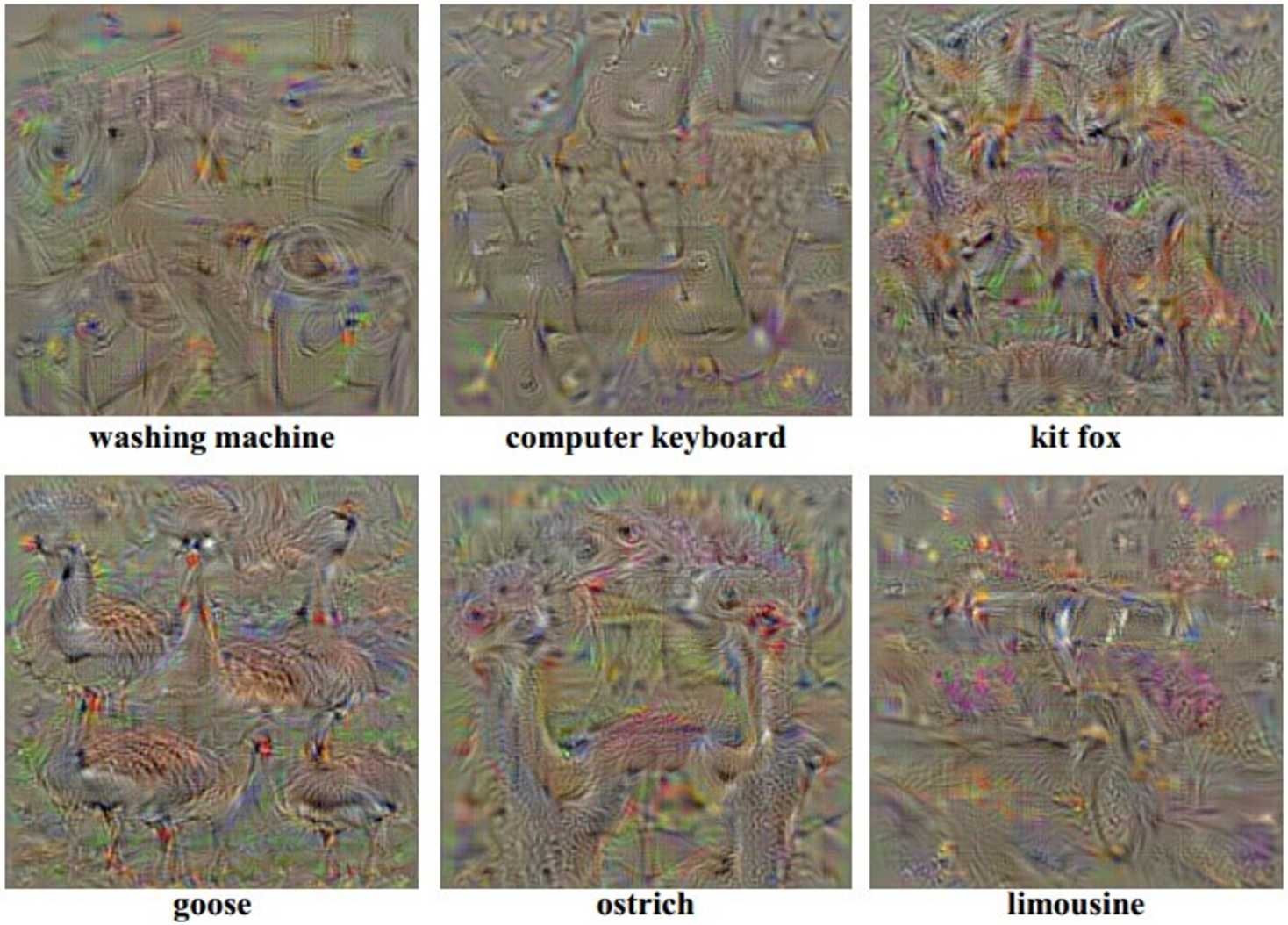


Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.
Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.

Visualizing CNN Features: Gradient Ascent

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

Simple regularizer: Penalize
L2 norm of generated image



Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.
Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.

Visualizing CNN Features: Gradient Ascent

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

Better regularizer: Penalize L2 norm of image; also during optimization periodically

1. Gaussian blur image
2. Clip pixels with small values to 0
3. Clip pixels with small gradients to 0

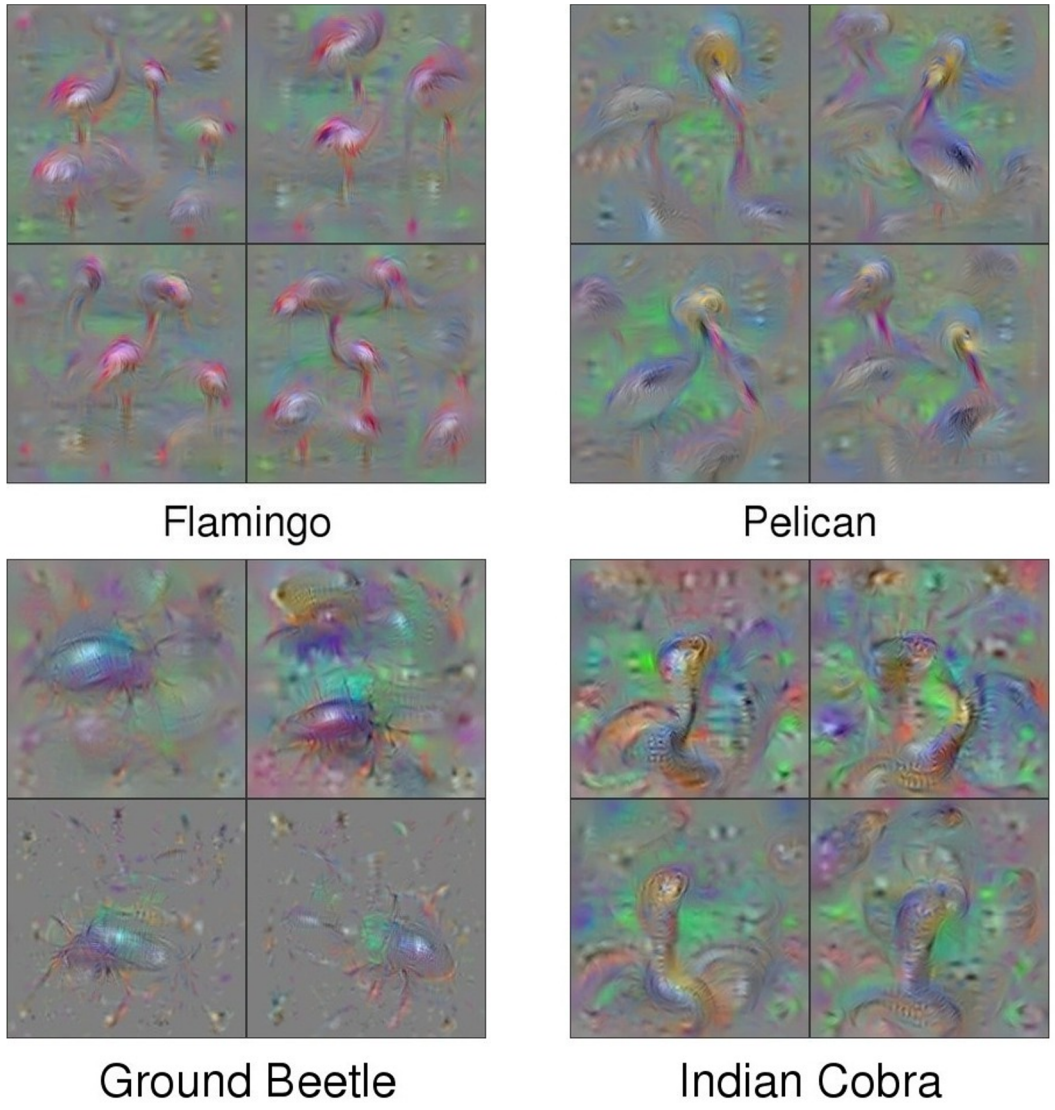
Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.

Visualizing CNN Features: Gradient Ascent

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

Better regularizer: Penalize L2 norm of image; also during optimization periodically

1. Gaussian blur image
2. Clip pixels with small values to 0
3. Clip pixels with small gradients to 0



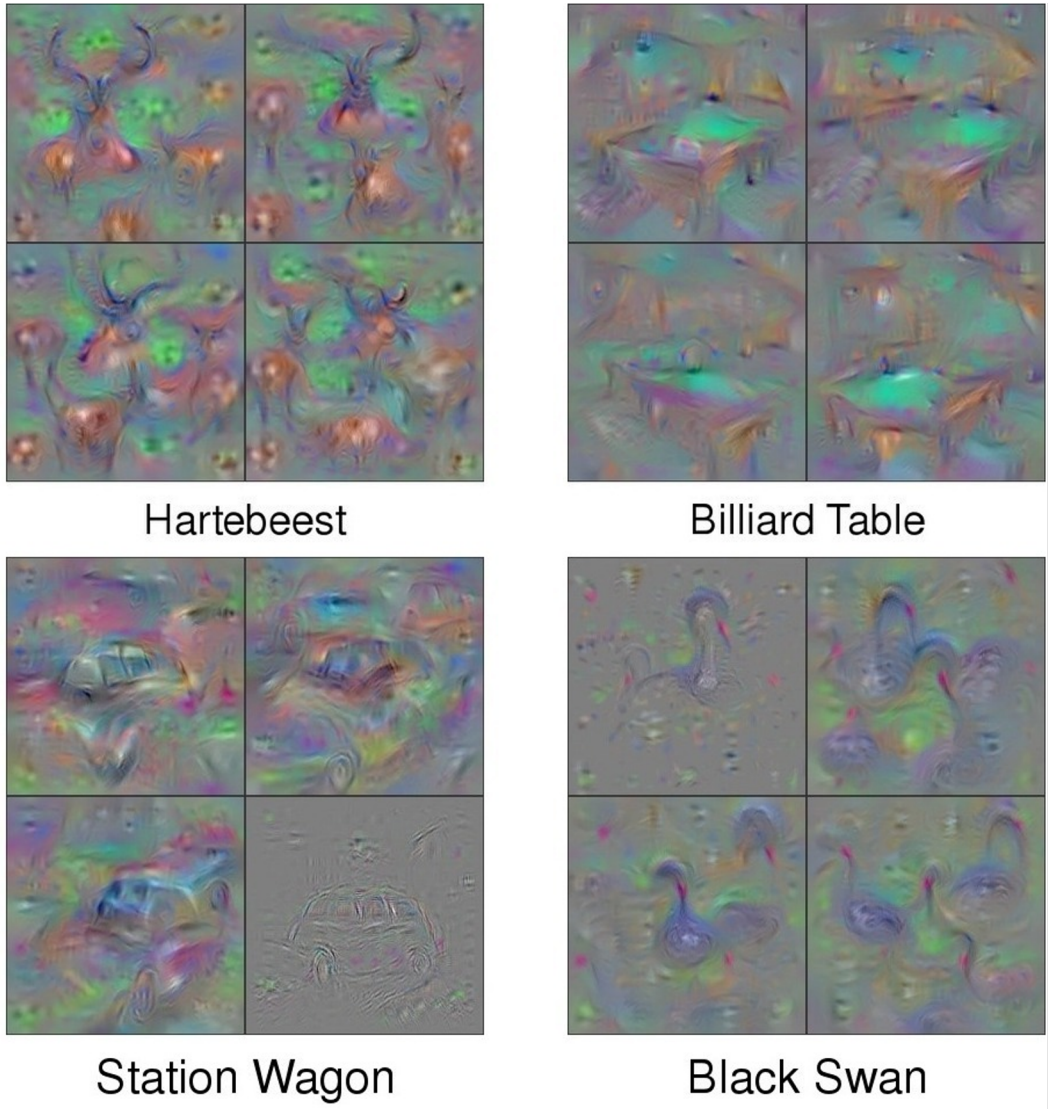
Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.

Visualizing CNN Features: Gradient Ascent

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

Better regularizer: Penalize L2 norm of image; also during optimization periodically

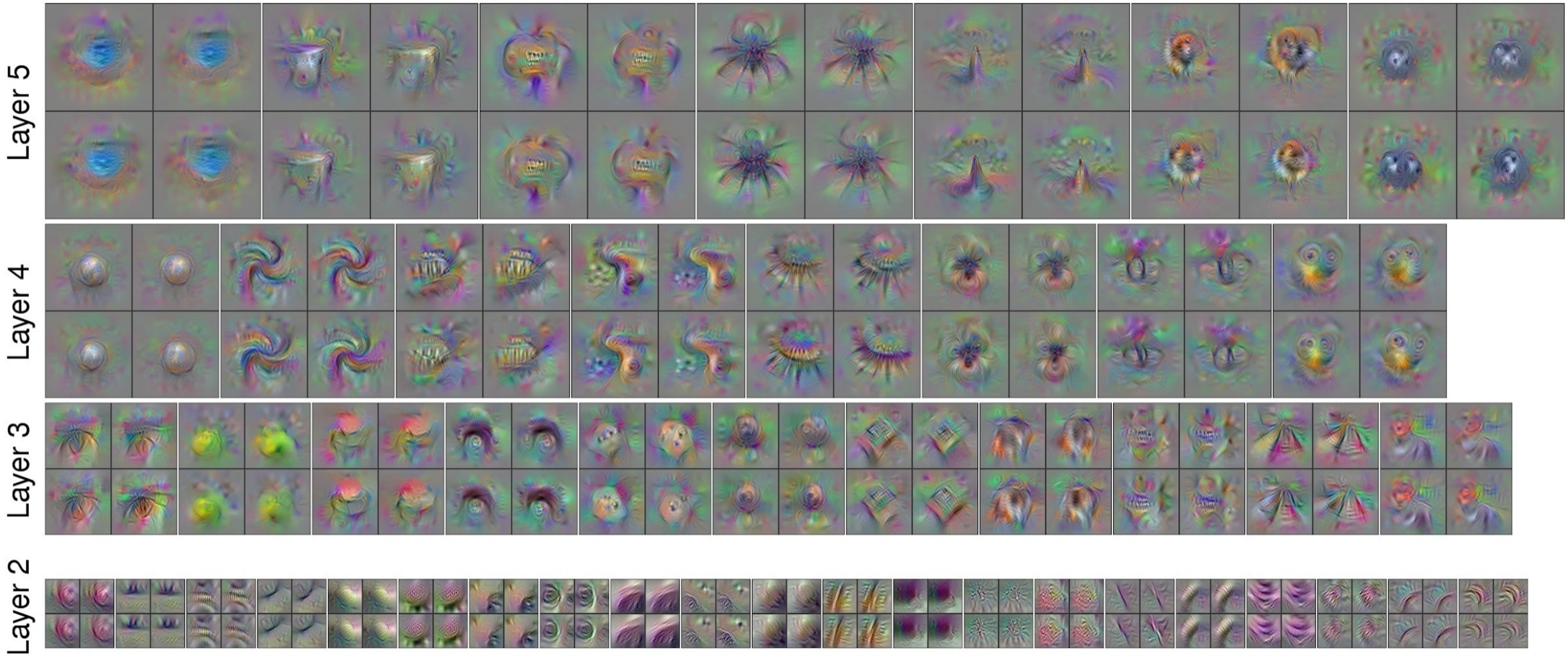
1. Gaussian blur image
2. Clip pixels with small values to 0
3. Clip pixels with small gradients to 0



Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.

Visualizing CNN Features: Gradient Ascent

Use the same approach to visualize intermediate features



Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.

Visualizing CNN Features: Gradient Ascent

Adding “multi-faceted” visualization gives even nicer results:
(Plus more careful regularization, center-bias)

Reconstructions of multiple feature types (facets) recognized by the same “grocery store” neuron

Corresponding example training set images recognized by the same neuron as in the "grocery store" class

Nguyen et al, “Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks”, ICML Visualization for Deep Learning Workshop 2016.

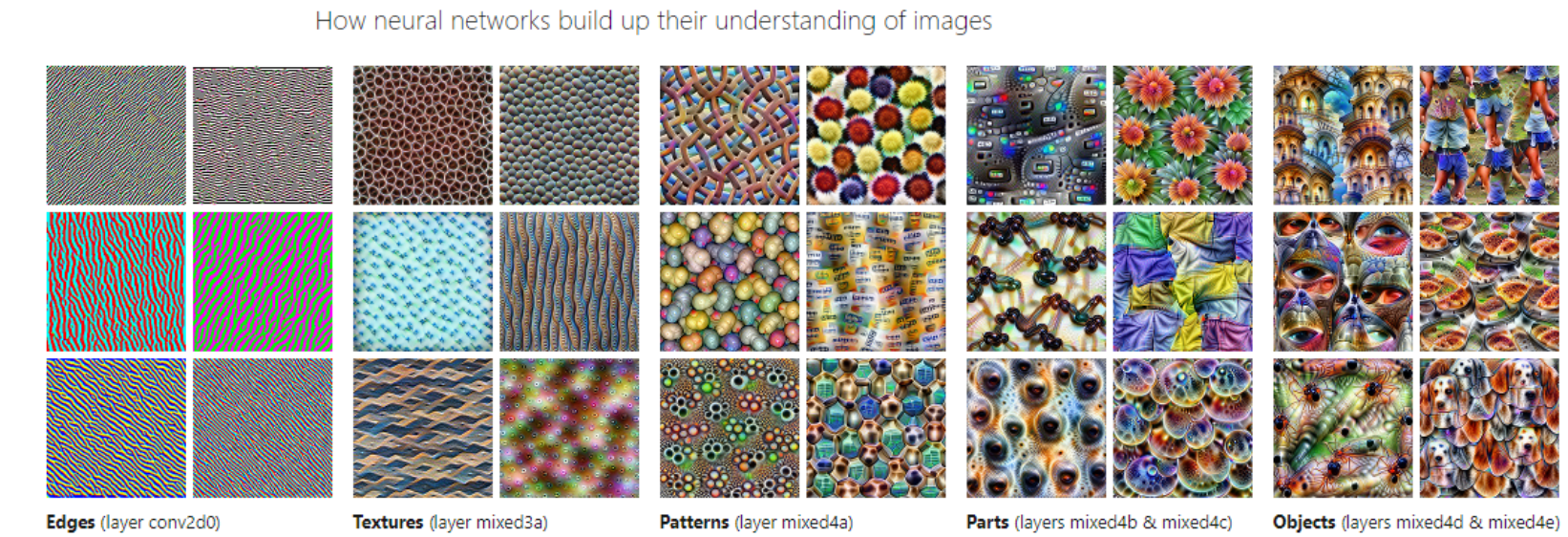
Visualizing CNN Features: Gradient Ascent

Nguyen et al, "Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks", ICML Visualization for Deep Learning Workshop 2016.

Visualizing CNN Features: Gradient Ascent

Nguyen et al, "Synthesizing the preferred inputs for neurons in neural networks via deep generator networks," NIPS 2016

Feature visualization by optimization



https://distill.pub/2017/feature-visualization/

Adversarial Examples

1. Start from an arbitrary image
2. Pick an arbitrary category
3. Modify the image (via gradient ascent) to maximize the class score
4. Stop when the network is fooled

Szegedy et al, “Intriguing properties of neural networks”, 2013

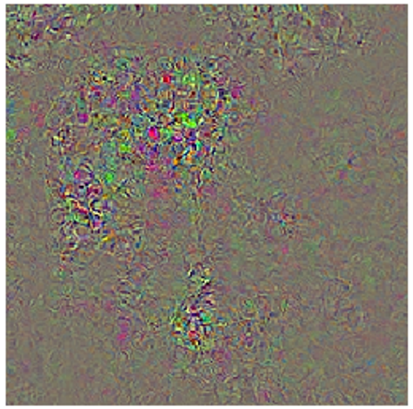
Adversarial Examples

African elephant

koala

Difference

10x Difference

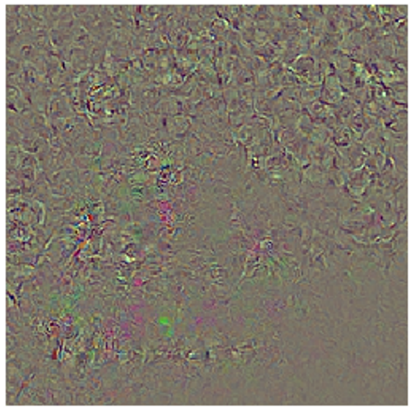


schooner

iPod

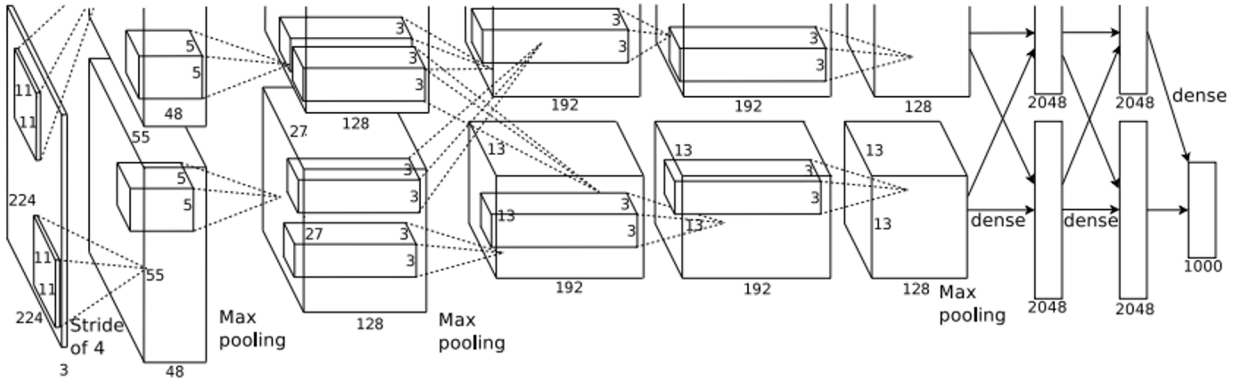
Difference

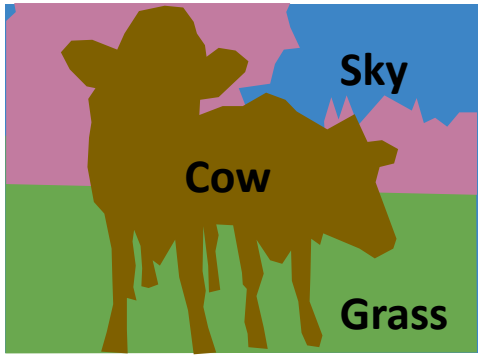
10x Difference



Network architectures

- Batch normalization
- CNN architectures
- Visualization
- Adversarial examples
- **Deconvolution and U-Nets**
- ~~Deep learning software~~





Lecture 16: Semantic Segmentation

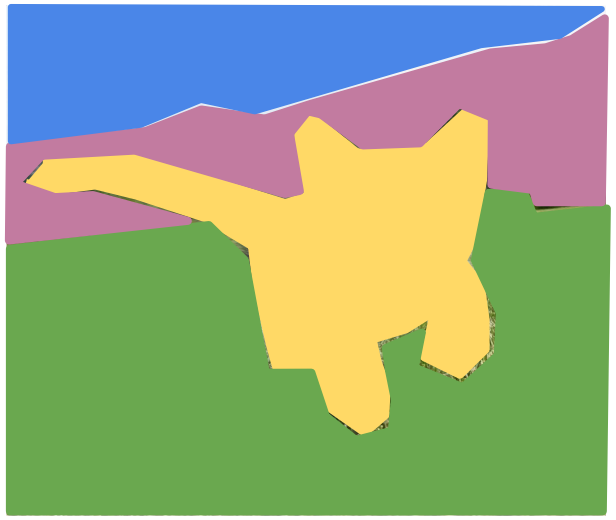
Computer Vision Tasks: Semantic Segmentation

Classification

CAT

No spatial extent

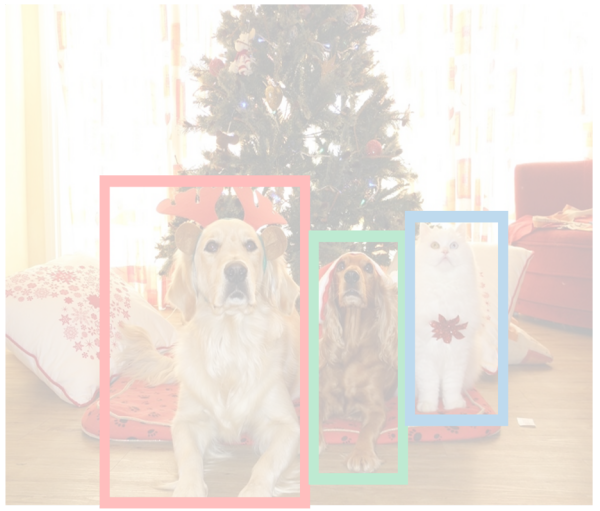
Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

Object Detection



DOG, DOG, CAT

Instance Segmentation

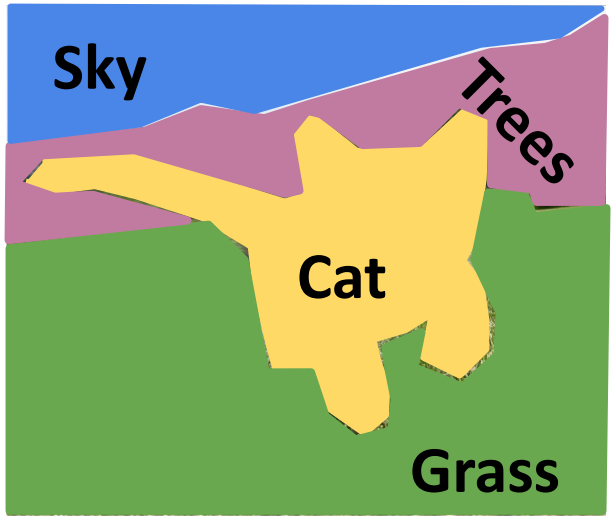
DOG, DOG, CAT

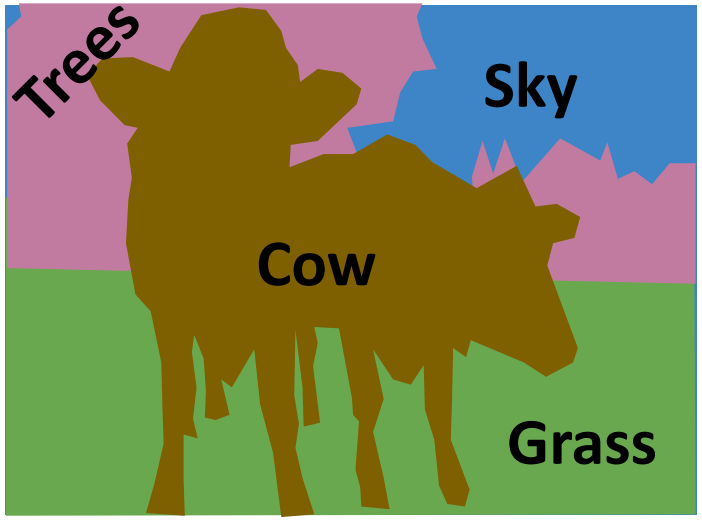
Multiple Objects

Semantic Segmentation

Label each pixel in the image with a category label

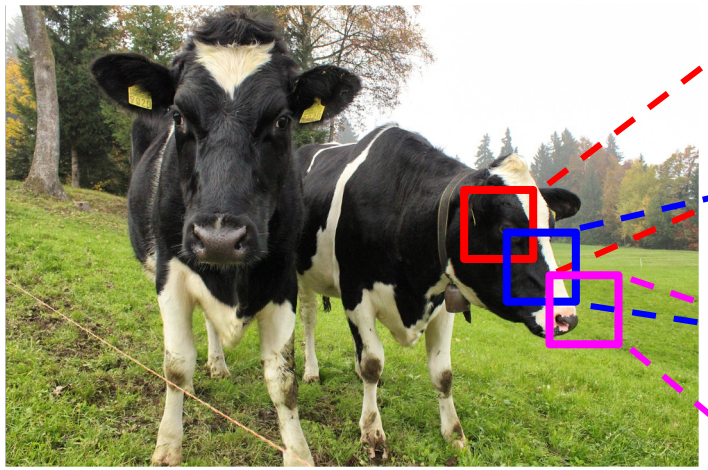
Don’t differentiate instances, only care about pixels



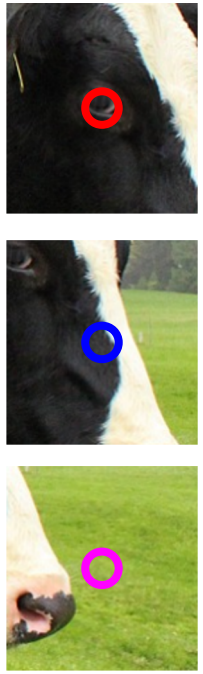


Semantic Segmentation Idea: Sliding Window

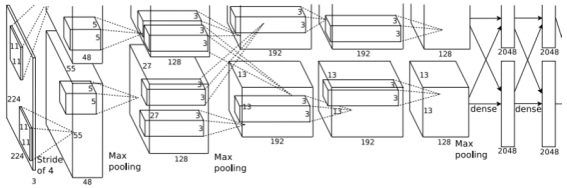
Full image



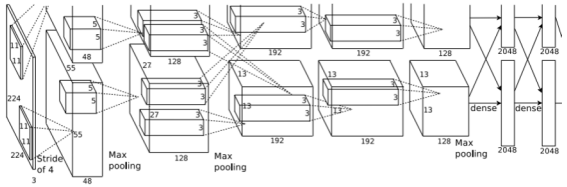
Extract patch



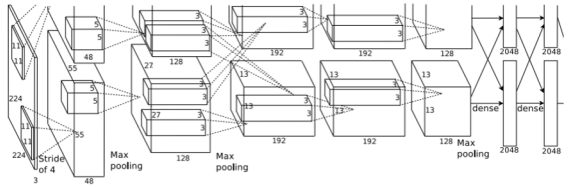
Classify center pixel with CNN



Cow



Cow



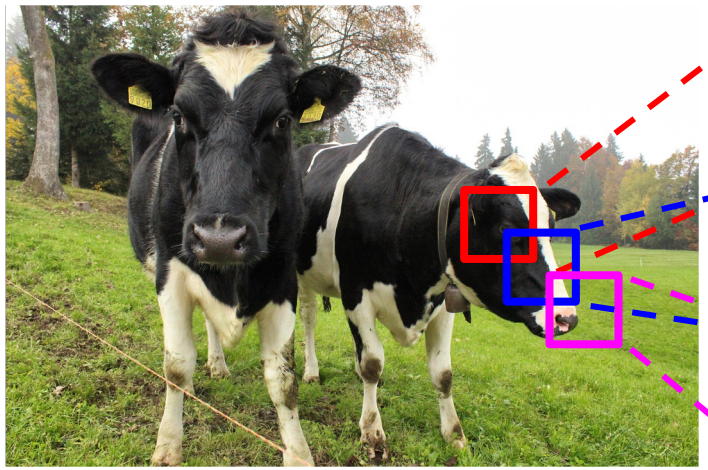
Grass

Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013

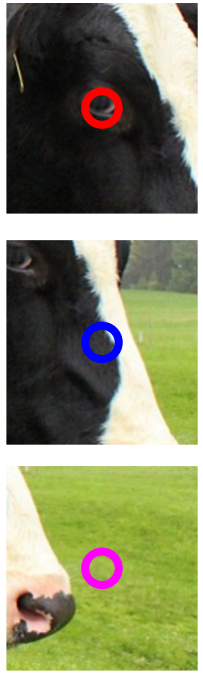
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

Semantic Segmentation Idea: Sliding Window

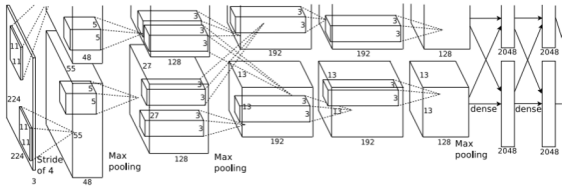
Full image



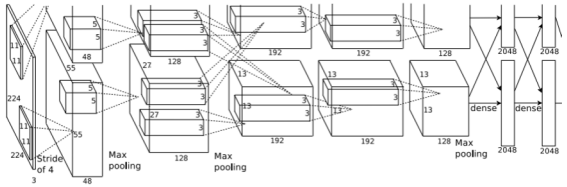
Extract patch



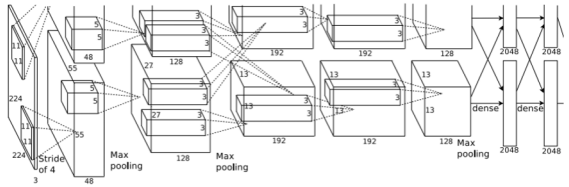
Classify center pixel with CNN



Cow



Cow



Grass

Problem: Very inefficient! Not reusing shared features between overlapping patches

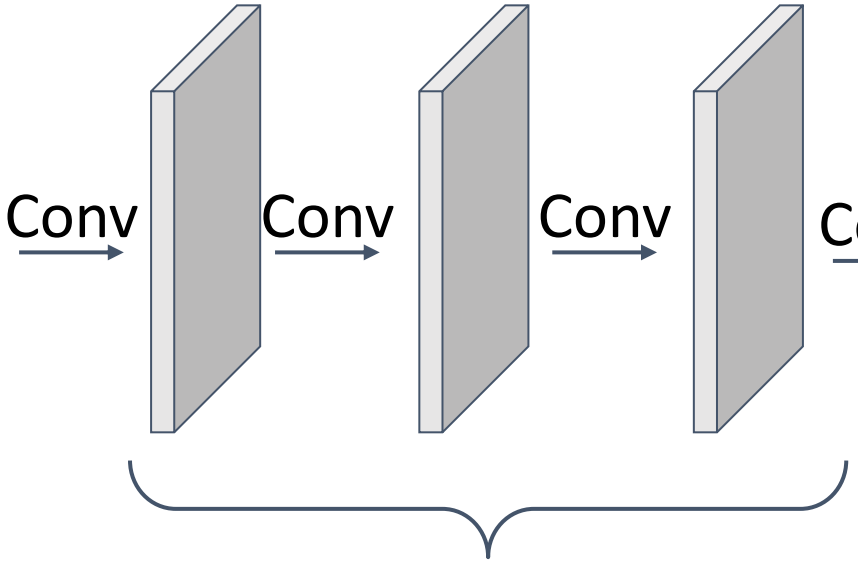
Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013

Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

Semantic Segmentation: Fully Convolutional Network

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!

Input:
3 x H x W



Convolutions:
D x H x W

Scores:
C x H x W

argmax

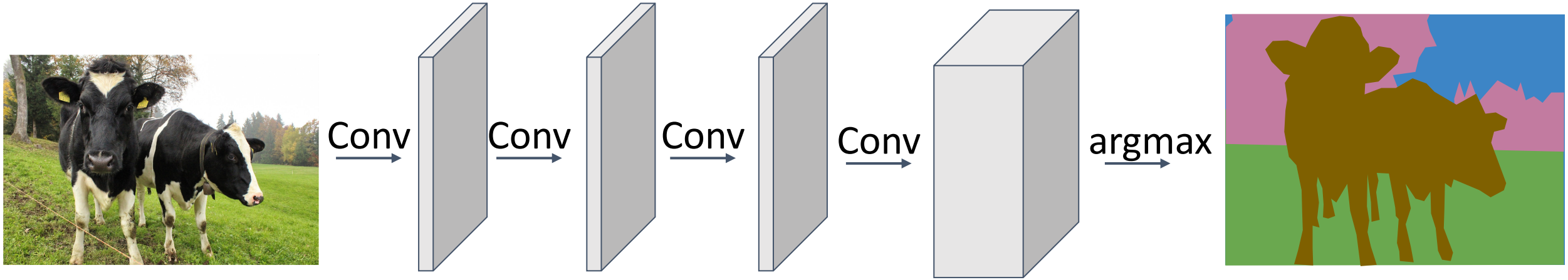
Predictions:
H x W

Loss function: Per-Pixel cross-entropy

Long et al, "Fully convolutional networks for semantic segmentation", CVPR 2015

Semantic Segmentation: Fully Convolutional Network

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!



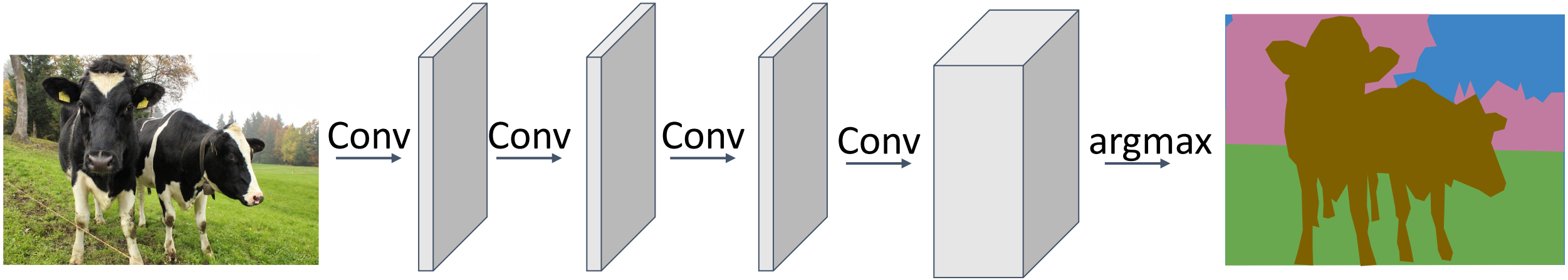
Input:
3 x H x W

Problem #1: Effective receptive field size is linear in number of conv layers: With L 3x3 conv layers, receptive field is 1+2L

Long et al, "Fully convolutional networks for semantic segmentation", CVPR 2015

Semantic Segmentation: Fully Convolutional Network

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!



Input:
3 x H x W

Problem #1: Effective receptive field size is linear in number of conv layers: With L 3x3 conv layers, receptive field is 1+2L

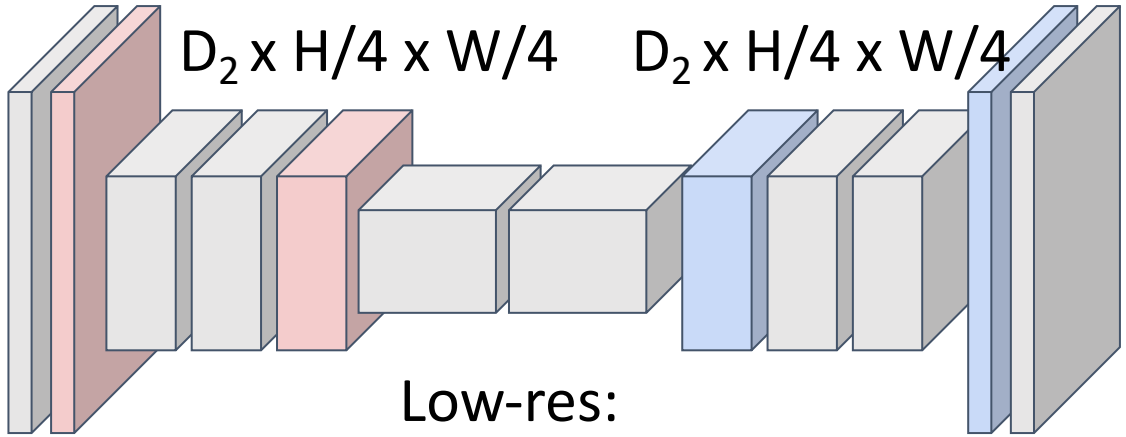
Problem #2: Convolution on high res images is expensive! Recall ResNet stem aggressively downsamples

Long et al, "Fully convolutional networks for semantic segmentation", CVPR 2015

Semantic Segmentation: Fully Convolutional Network

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Input:
3 x H x W



Predictions:
H x W

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

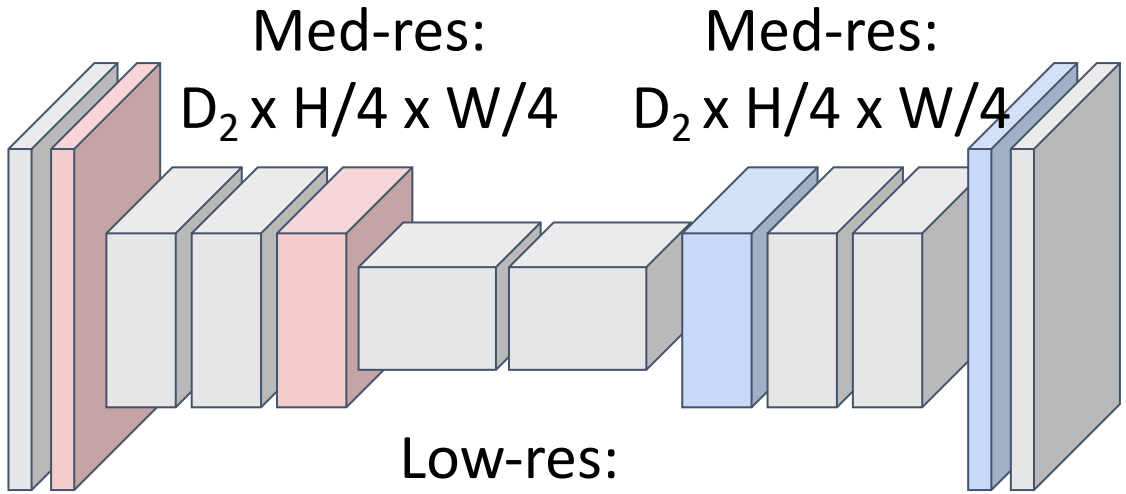
Semantic Segmentation: Fully Convolutional Network

Downsampling: Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling: ???

Input:
3 x H x W

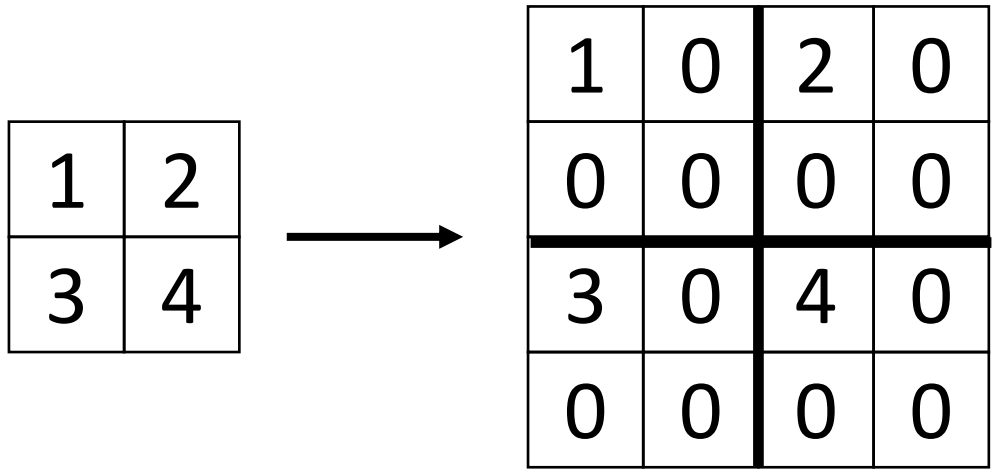


Predictions:
H x W

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

In-Network Upsampling: “Unpooling”

Bed of Nails

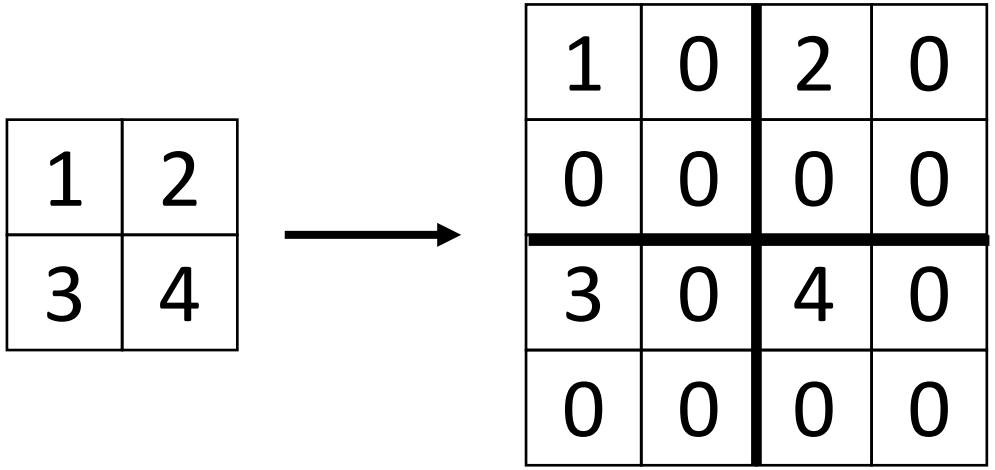


Input
C x 2 x 2

Output
C x 4 x 4

In-Network Upsampling: “Unpooling”

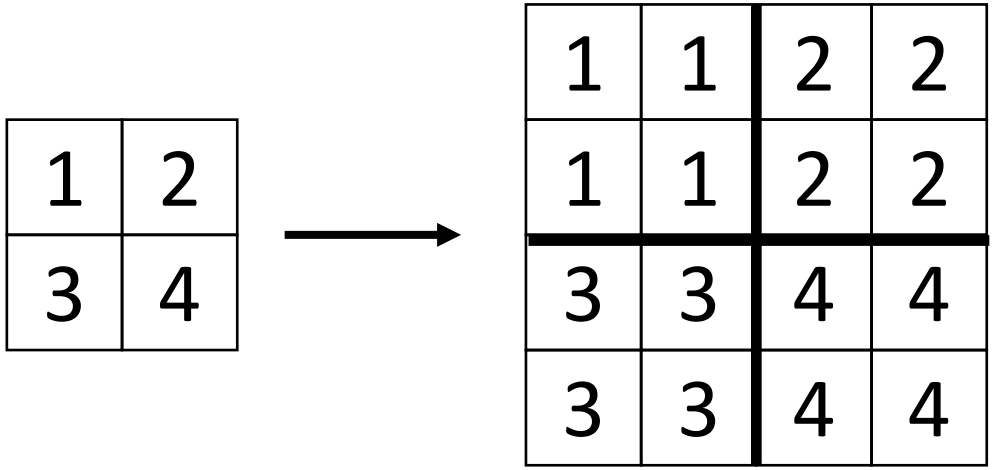
Bed of Nails



Input
C x 2 x 2

Output
C x 4 x 4

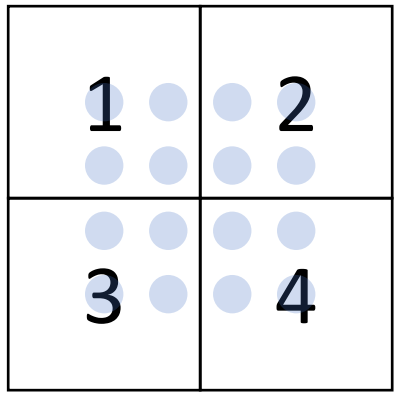
Nearest Neighbor



Input
C x 2 x 2

Output
C x 4 x 4

In-Network Upsampling: Bilinear Interpolation



1.00	1.25	1.75	2.00
1.50	1.75	2.25	2.50
2.50	2.75	3.25	3.50
3.00	3.25	3.75	4.00

Input: C x 2 x 2

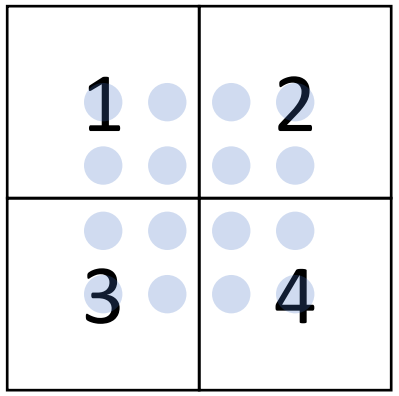
Output: C x 4 x 4

$$f_{x,y} = \sum_{i,j} f_{i,j} \max(0, 1 - |x - i|) \max(0, 1 - |y - j|)$$

$$i \in \{\lfloor x \rfloor - 1, \ldots, \lceil x \rceil + 1\}$$
$$j \in \{\lfloor y \rfloor - 1, \ldots, \lceil y \rceil + 1\}$$

Use two closest neighbors in x and y to construct linear approximations

In-Network Upsampling: Bicubic Interpolation



0.68	1.02	1.56	1.89
1.35	1.68	2.23	2.56
2.44	2.77	3.32	3.65
3.11	3.44	3.98	4.32

Input: C x 2 x 2

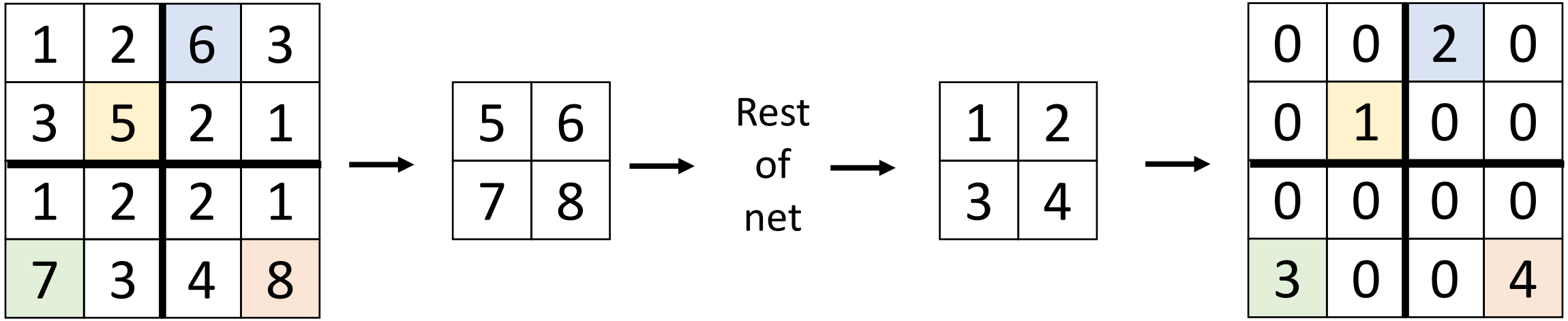
Output: C x 4 x 4

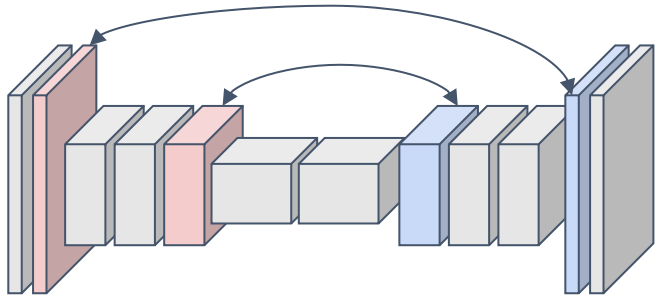
Use **three** closest neighbors in x and y to construct **cubic** approximations
(This is how we normally resize images!)

In-Network Upsampling: "Max Unpooling"

Max Pooling: Remember which position had the max

Max Unpooling: Place into remembered positions



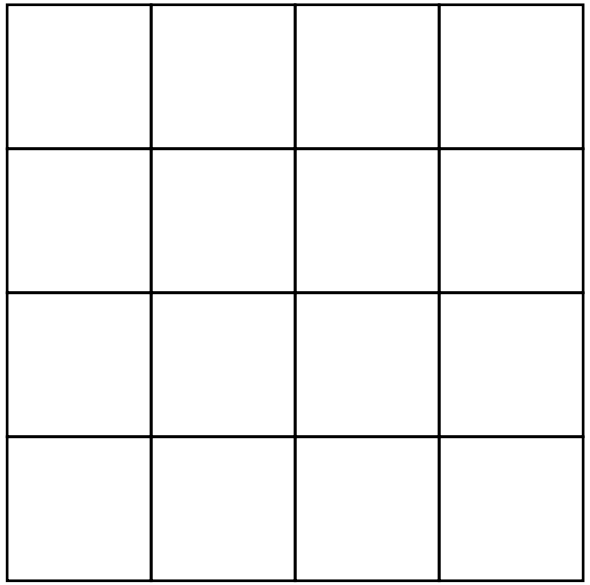


Pair each downsampling layer with an upsampling layer

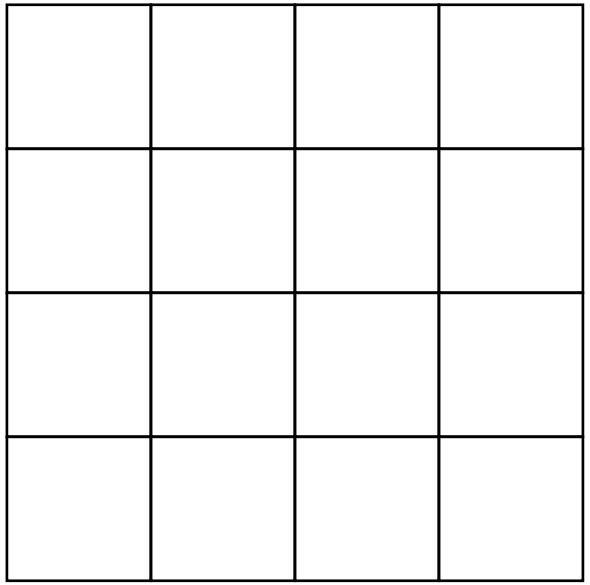
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Learnable Upsampling: Transposed Convolution

Recall: Normal 3 x 3 convolution, stride 1, pad 1



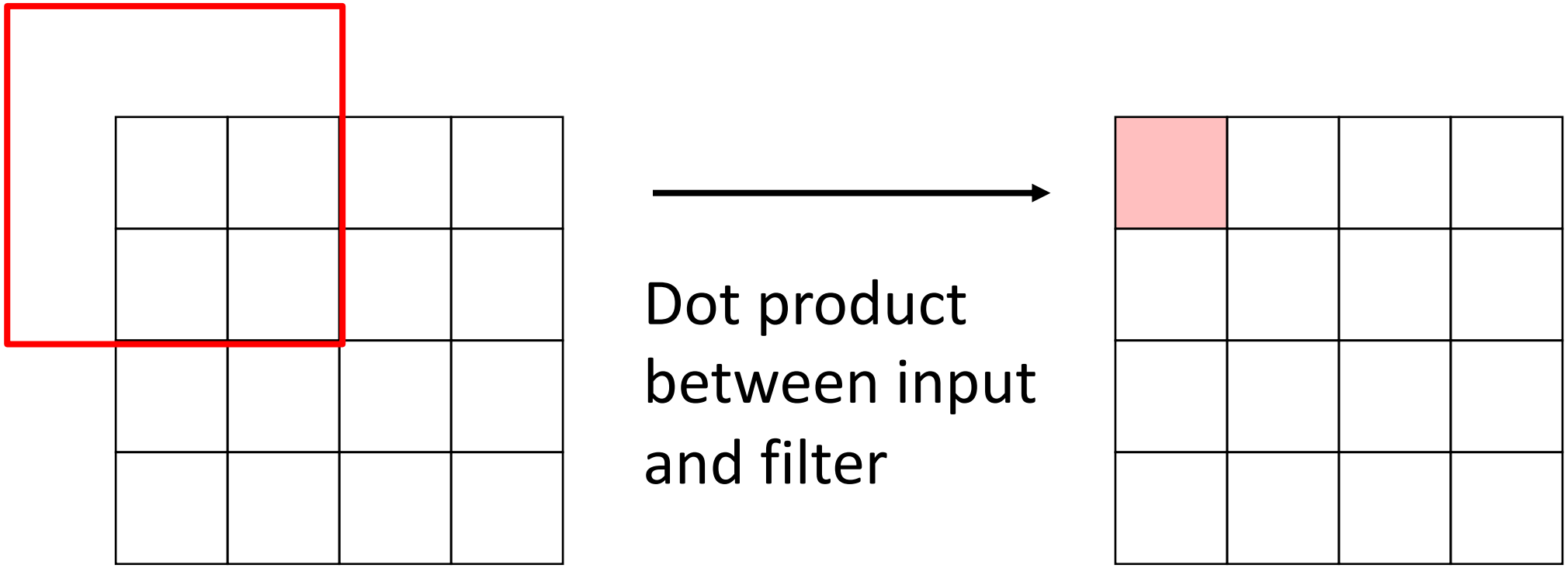
Input: 4 x 4



Output: 4 x 4

Learnable Upsampling: Transposed Convolution

Recall: Normal 3 x 3 convolution, stride 1, pad 1

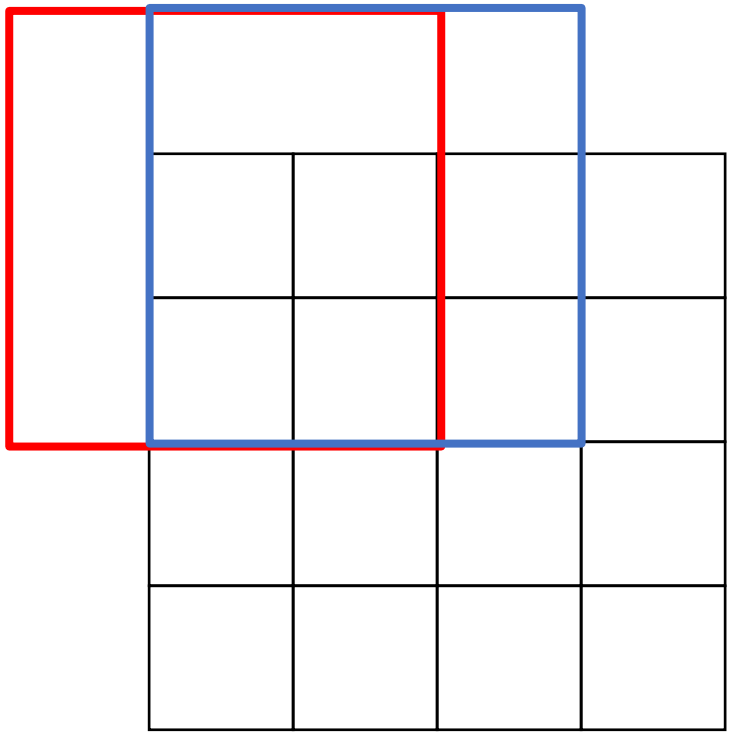


Input: 4 x 4

Output: 4 x 4

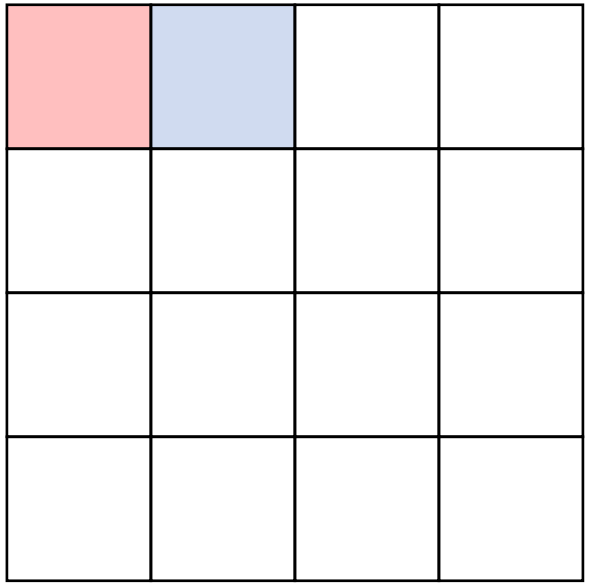
Learnable Upsampling: Transposed Convolution

Recall: Normal 3 x 3 convolution, stride 1, pad 1



Input: 4 x 4

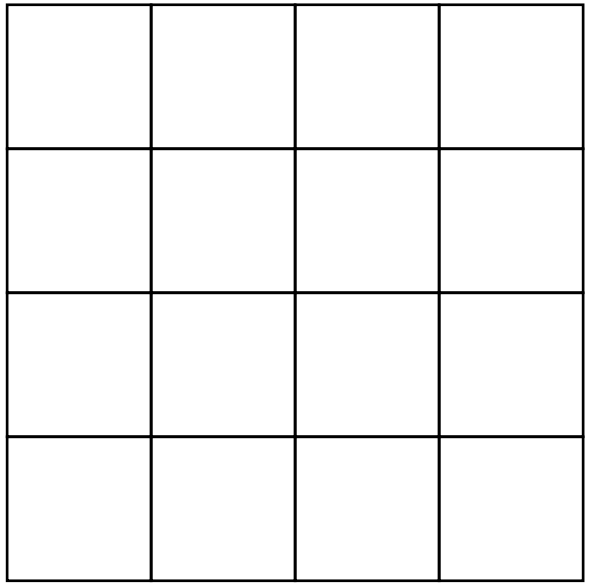
Dot product between input and filter



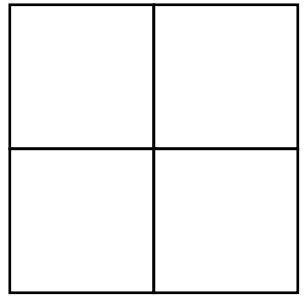
Output: 4 x 4

Learnable Upsampling: Transposed Convolution

Recall: Normal 3 x 3 convolution, stride 2, pad 1



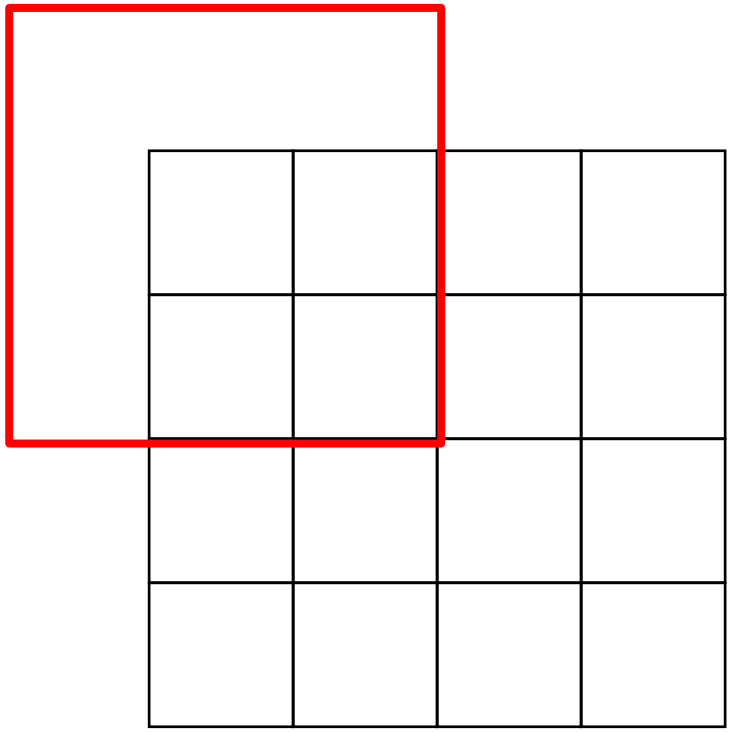
Input: 4 x 4



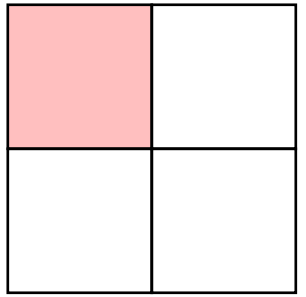
Output: 2 x 2

Learnable Upsampling: Transposed Convolution

Recall: Normal 3 x 3 convolution, stride 2, pad 1



Dot product between input and filter

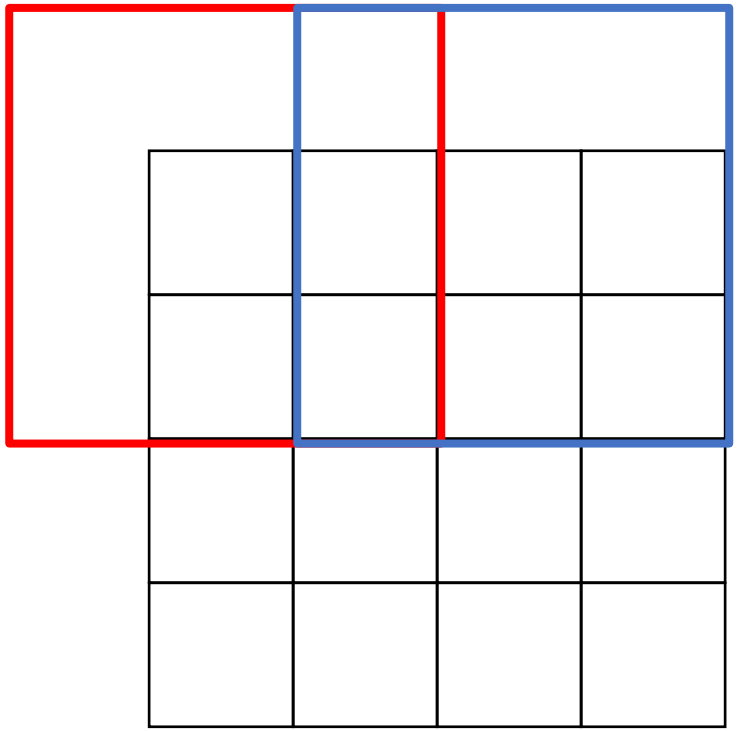


Input: 4 x 4

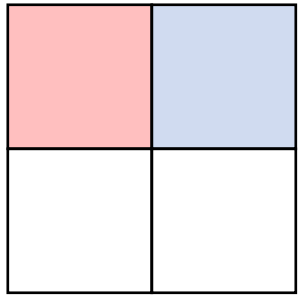
Output: 2 x 2

Learnable Upsampling: Transposed Convolution

Recall: Normal 3 x 3 convolution, stride 2, pad 1



Dot product between input and filter

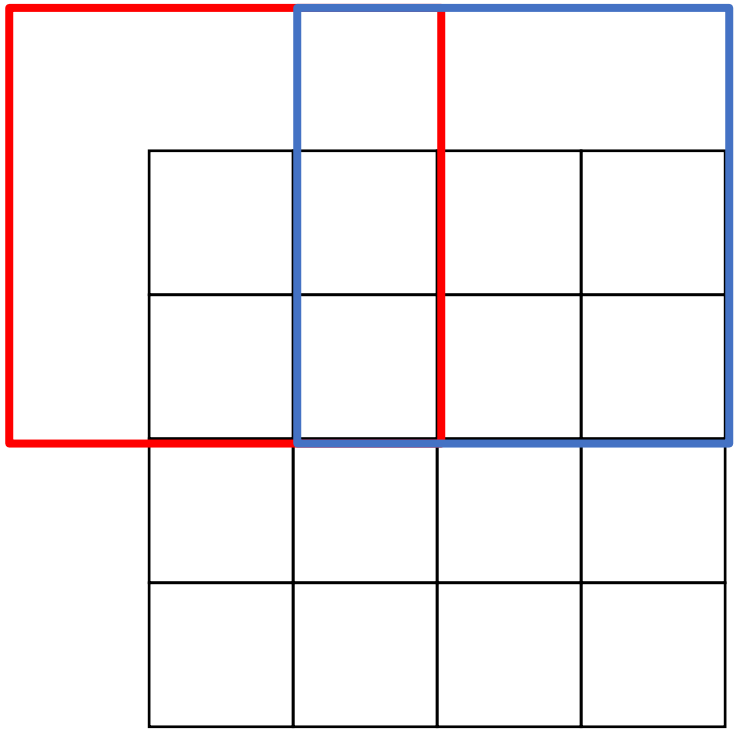


Input: 4 x 4

Output: 2 x 2

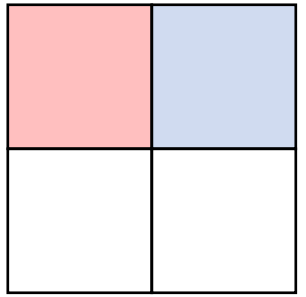
Learnable Upsampling: Transposed Convolution

Recall: Normal 3 x 3 convolution, stride 2, pad 1



Convolution with stride > 1 is “Learnable Downsampling”
Can we use stride < 1 for “Learnable Upsampling”?

Dot product between input and filter

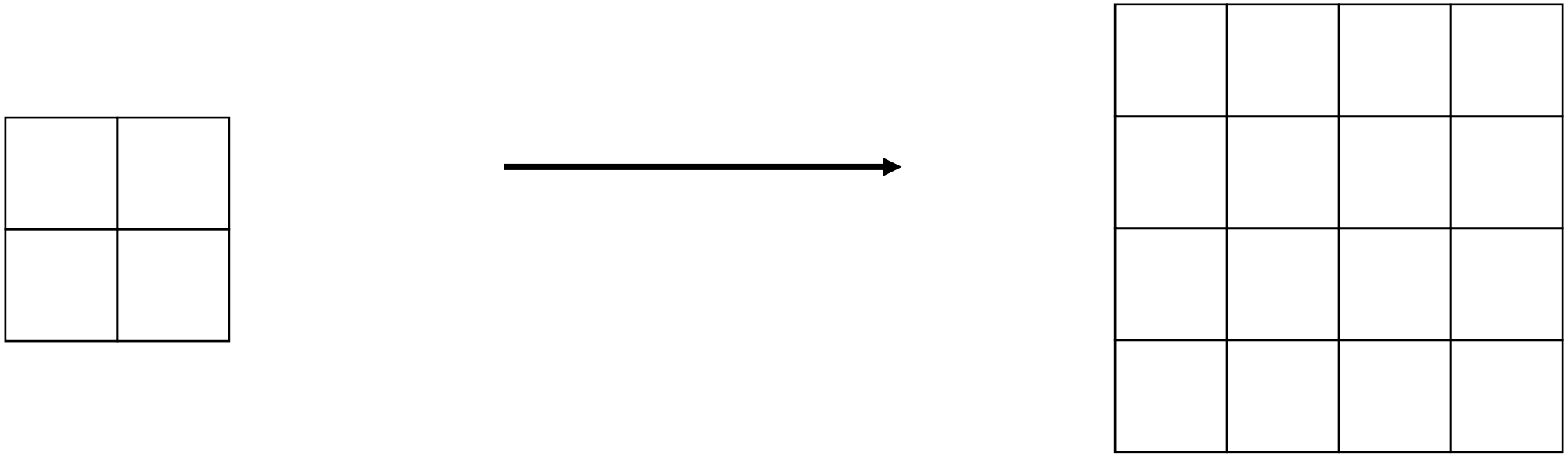


Input: 4 x 4

Output: 2 x 2

Learnable Upsampling: Transposed Convolution

3 x 3 **convolution transpose**, stride 2

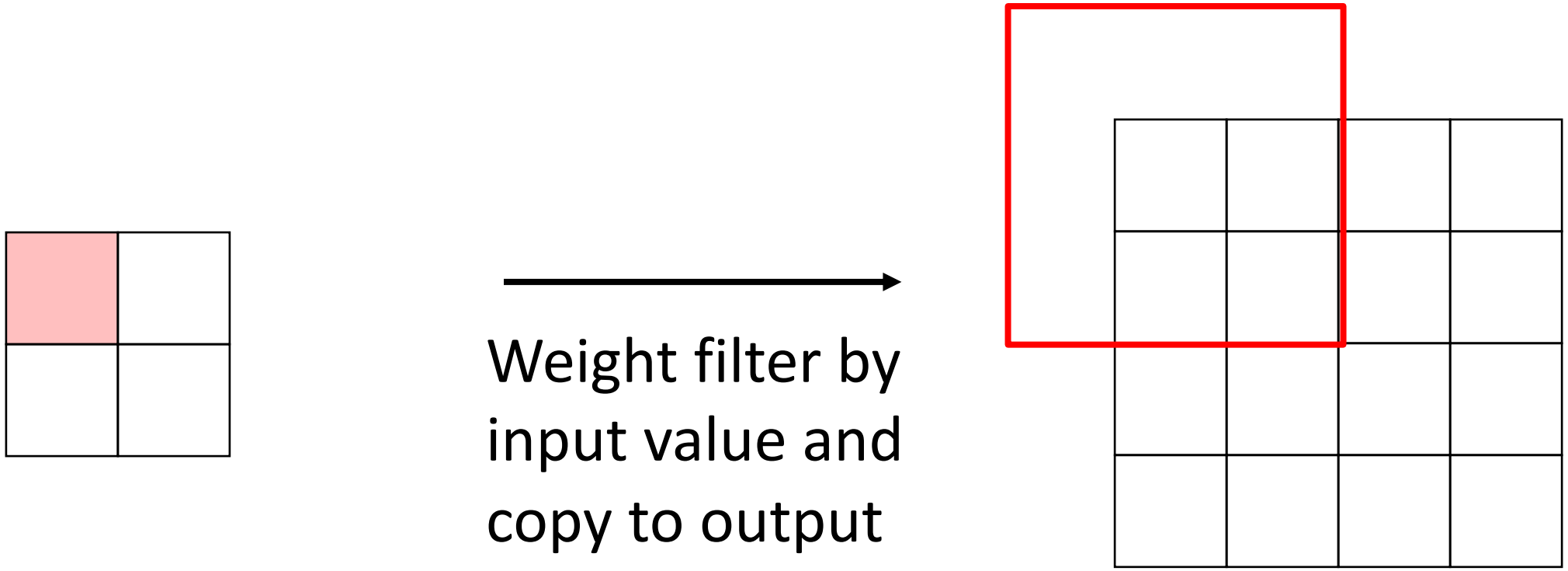


Input: 2 x 2

Output: 4 x 4

Learnable Upsampling: Transposed Convolution

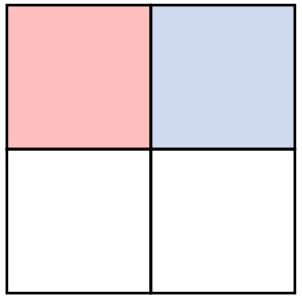
3 x 3 **convolution transpose**, stride 2



Learnable Upsampling: Transposed Convolution

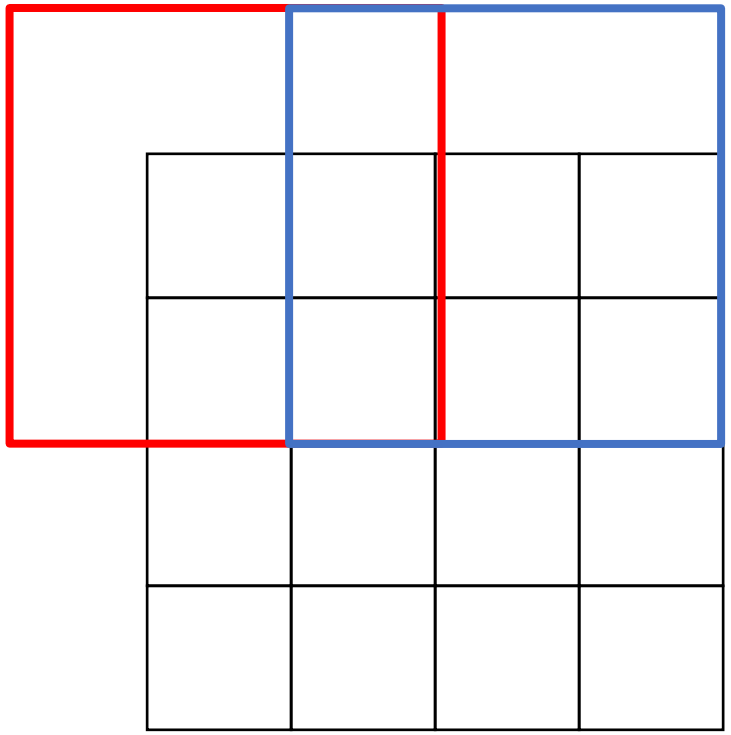
3 x 3 **convolution transpose**, stride 2

Filter moves 2 pixels in output
for every 1 pixel in input



Input: 2 x 2

Weight filter by
input value and
copy to output

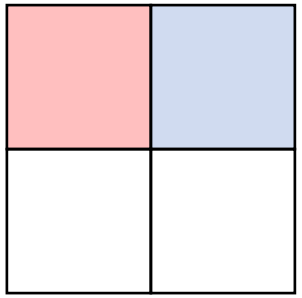


Output: 4 x 4

Learnable Upsampling: Transposed Convolution

3 x 3 **convolution transpose**, stride 2

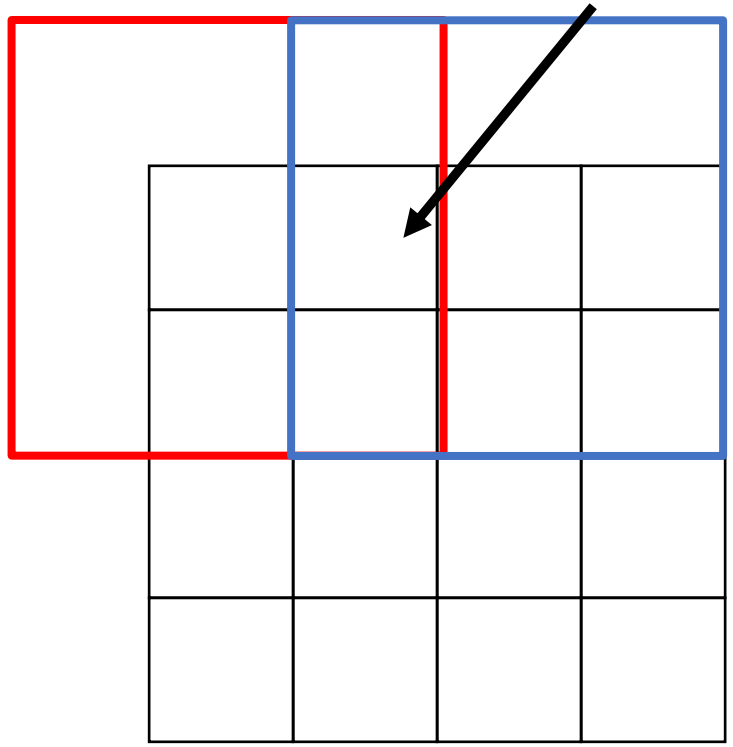
Filter moves 2 pixels in output
for every 1 pixel in input



Input: 2 x 2

Weight filter by input value and copy to output

Sum where output overlaps



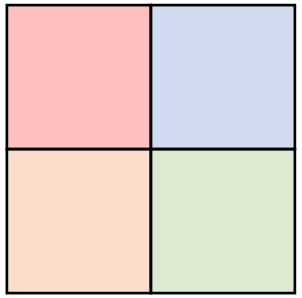
Output: 4 x 4

Learnable Upsampling: Transposed Convolution

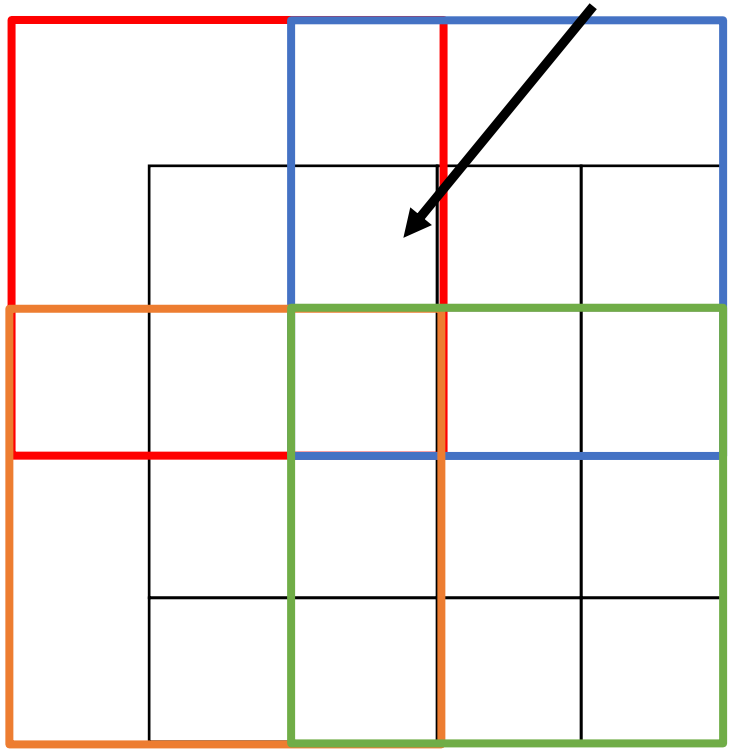
3 x 3 **convolution transpose**, stride 2

Sum where output overlaps

This gives 5x5 output – need to trim one pixel from top and left to give 4x4 output



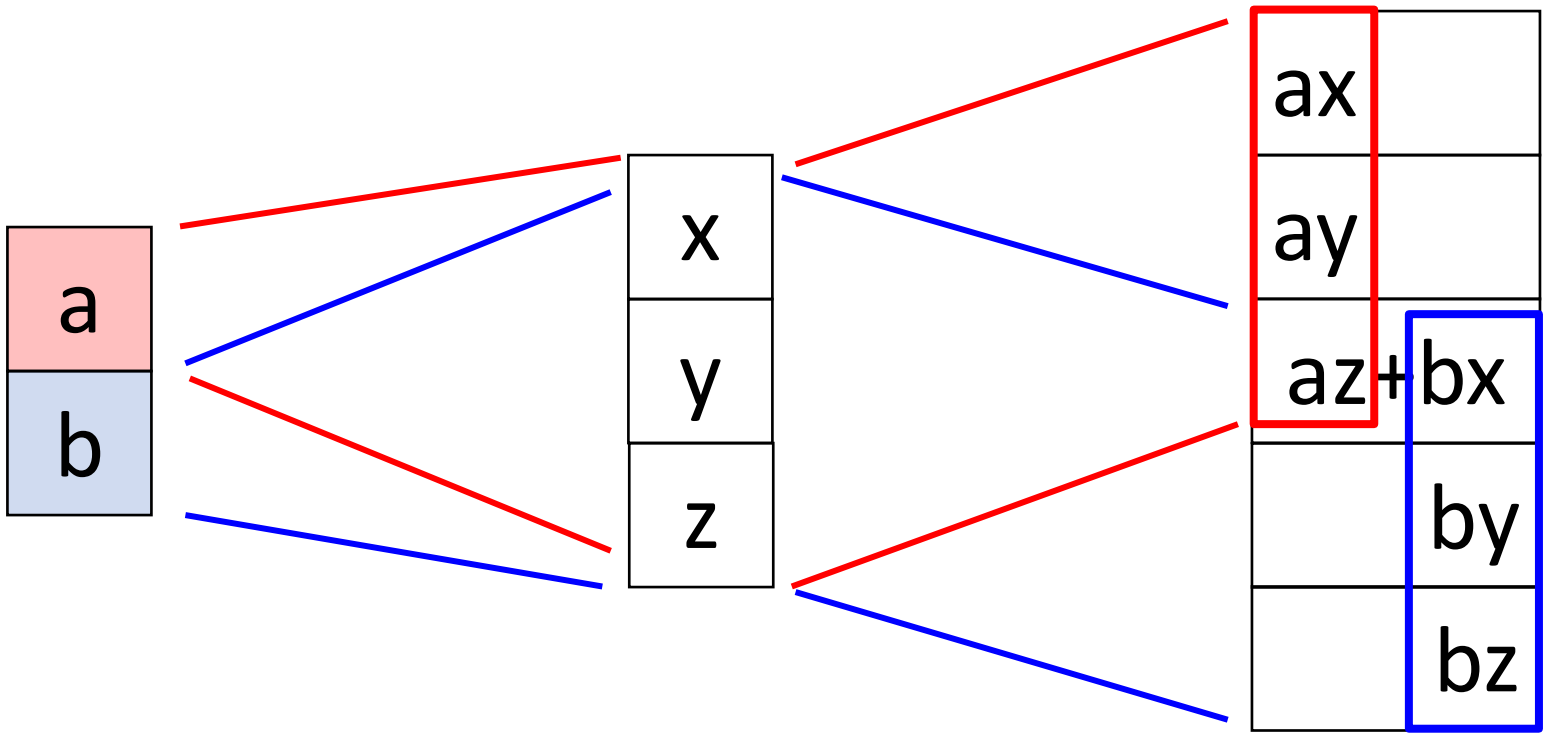
Weight filter by input value and copy to output



Input: 2 x 2

Output: 4 x 4

Transposed Convolution: 1D example

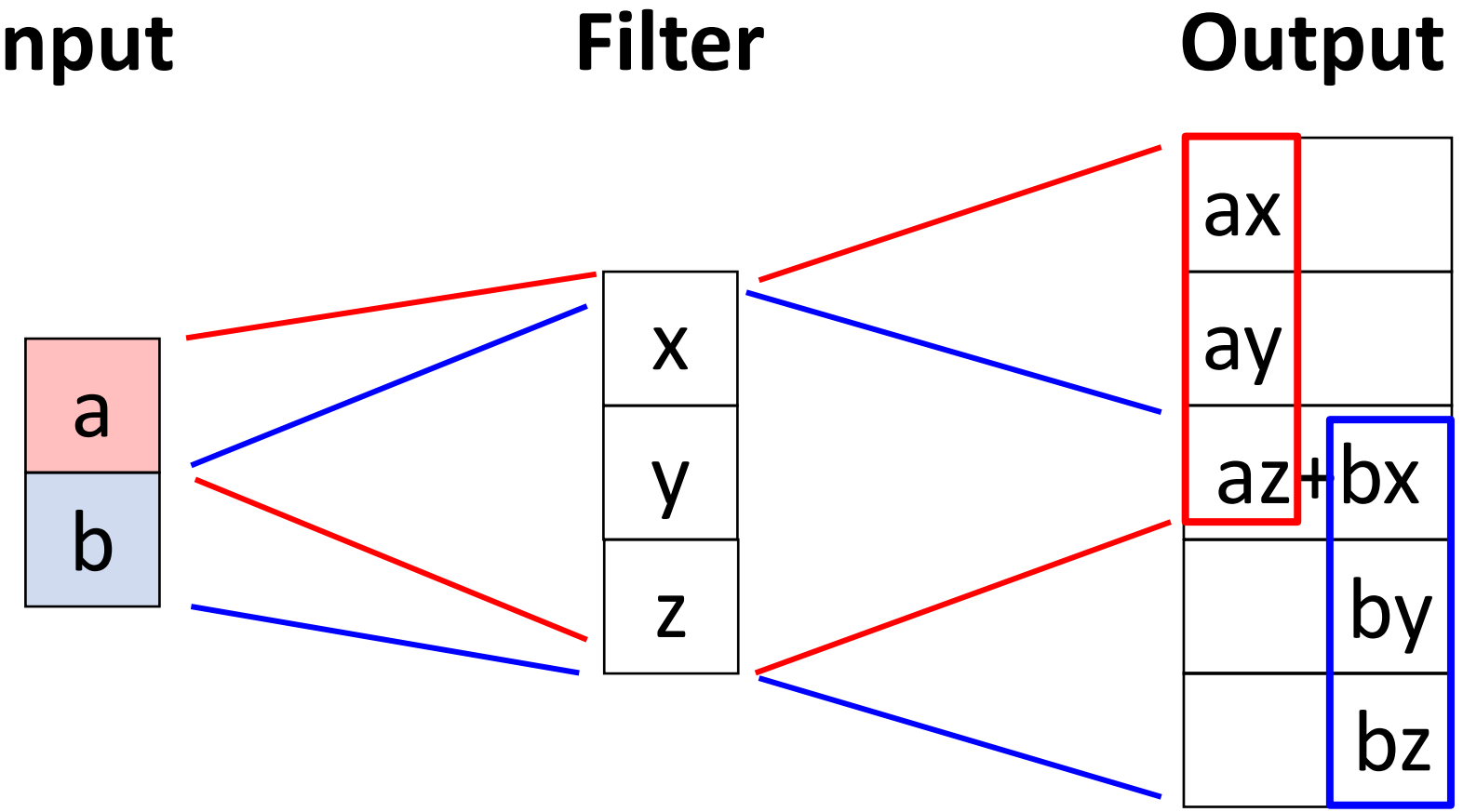


Output has copies of filter weighted by input

Stride 2: Move 2 pixels output for each pixel in input

Sum at overlaps

Transposed Convolution: 1D example



This has many names:

- Deconvolution (bad)!
- Upconvolution
- Fractionally strided convolution
- Backward strided convolution
- Transposed Convolution (best name)

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & x & 0 & 0 & 0 \\ 0 & x & y & x & 0 & 0 \\ 0 & 0 & x & y & x & 0 \\ 0 & 0 & 0 & x & y & x \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & x & 0 & 0 & 0 \\ 0 & x & y & x & 0 & 0 \\ 0 & 0 & x & y & x & 0 \\ 0 & 0 & 0 & x & y & x \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

Transposed convolution multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T \vec{a}$$

$$\begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix}$$

When stride=1, transposed conv is just a regular conv (with different padding rules)

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & x & 0 & 0 & 0 \\ 0 & 0 & x & y & x & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=2, padding=1

Transposed convolution multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T \vec{a}$$

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & x & 0 & 0 & 0 \\ 0 & 0 & x & y & x & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=2, padding=1

Transposed convolution multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T \vec{a}$$

$$\begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

When stride>1, transposed convolution cannot be expressed as normal conv

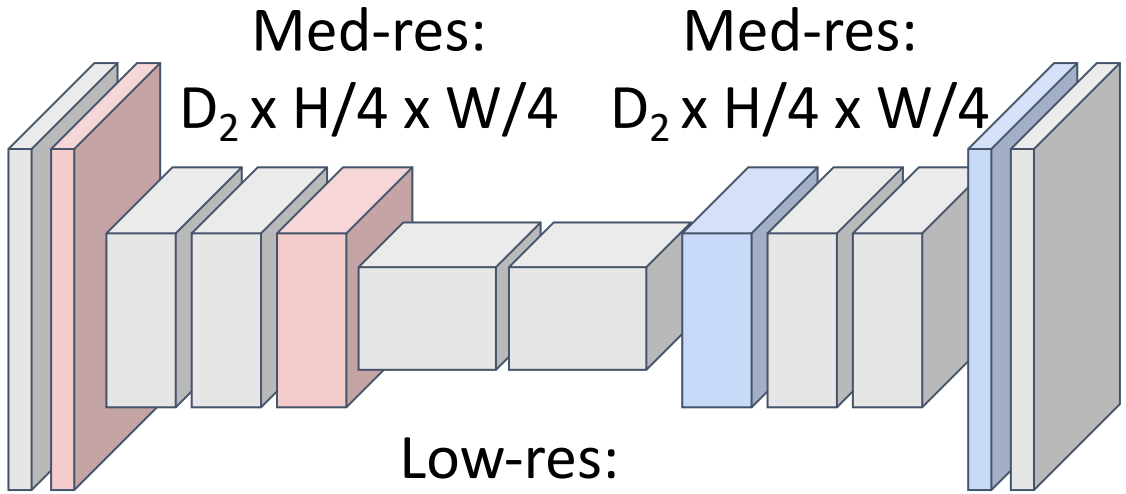
Semantic Segmentation: Fully Convolutional Network

Downsampling: Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling: linterpolation, transposed conv

Input:
3 x H x W



High-res:
D1 x H/2 x W/2

High-res:
D1 x H/2 x W/2

Predictions:
H x W

Loss function: Per-Pixel cross-entropy

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

U-Nets

- Skip connections between encoding and decoding stages
- Widely used in image denoising, inpainting, flow, stereo, …

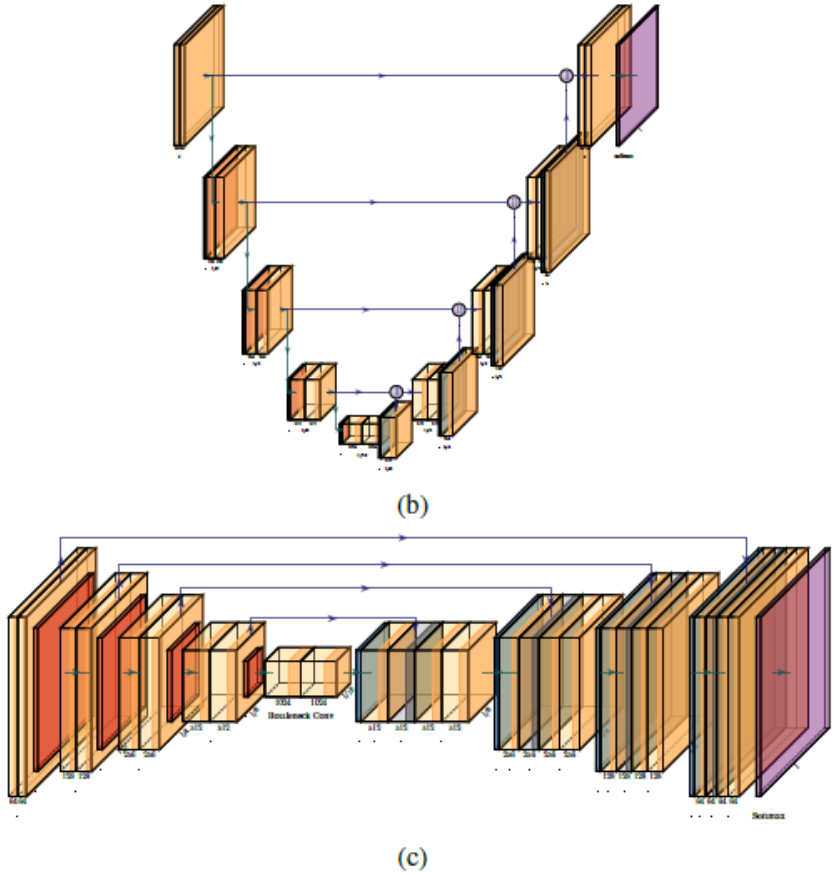
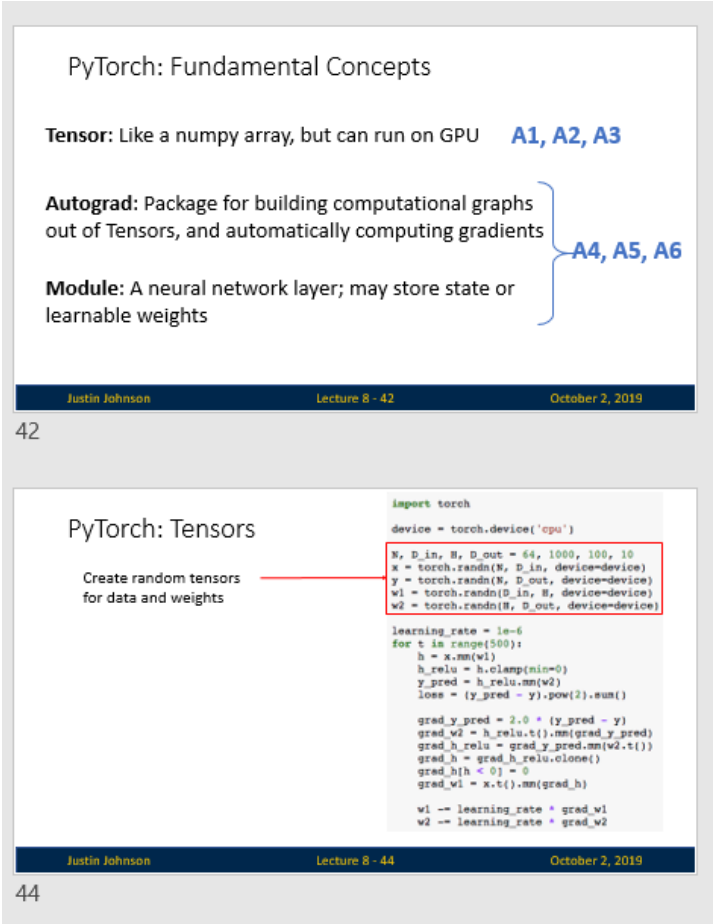


Figure 5.41 *(a) The deconvolution network of Noh, Hong, and Han (2015) © 2015 IEEE and (b–c) the U-Net of Ronneberger, Fischer, and Brox (2015), redrawn using the PlotNeuralNet LaTeX package by Matt Dietke. In addition to the fine-to-coarse-to-fine bottleneck used in (a), the U-Net also has skip connections between encoding and decoding layers at the same resolution.*

~~Deep Learning~~ Software

Will cover this May 5 (CSE 576 Lecture 11)

JIT for HW4



Summary: Network architectures

- Batch normalization
- CNN architectures
- Visualization
- Adversarial examples
- Deconvolution and U-Nets
- ~~Deep learning software~~