Convolutional Neural Networks IV

CS7150, Spring 2025

Prof. Huaizu Jiang

Northeastern University

Recap

Convolution Summary

Input: C_{in} x H x W
Hyperparameters:
- **Kernel size**: K_H x K_W
- **Number filters**: C_{out}
- **Padding**: P
- **Stride**: S

Weight matrix: C_{out} x C_{in} x K_H x K_W
giving C_{out} filters of size C_{in} x K_H x K_W

Bias vector: C_{out}

Output size: C_{out} x H' x W' where:
- H' = Ceil((H – K + 2P + 1) / S)
- W' = Ceil((W – K + 2P + 1) / S)

Common settings:
$K_H = K_W$ (Small square filters)
P = (K – 1) / 2 ("Same" padding)
C_{in}, C_{out} = 32, 64, 128, 256 (powers of 2)
K = 3, P = 1, S = 1 (3x3 conv)
K = 5, P = 2, S = 1 (5x5 conv)
K = 1, P = 0, S = 1 (1x1 conv)
K = 3, P = 1, S = 2 (Downsample by 2)

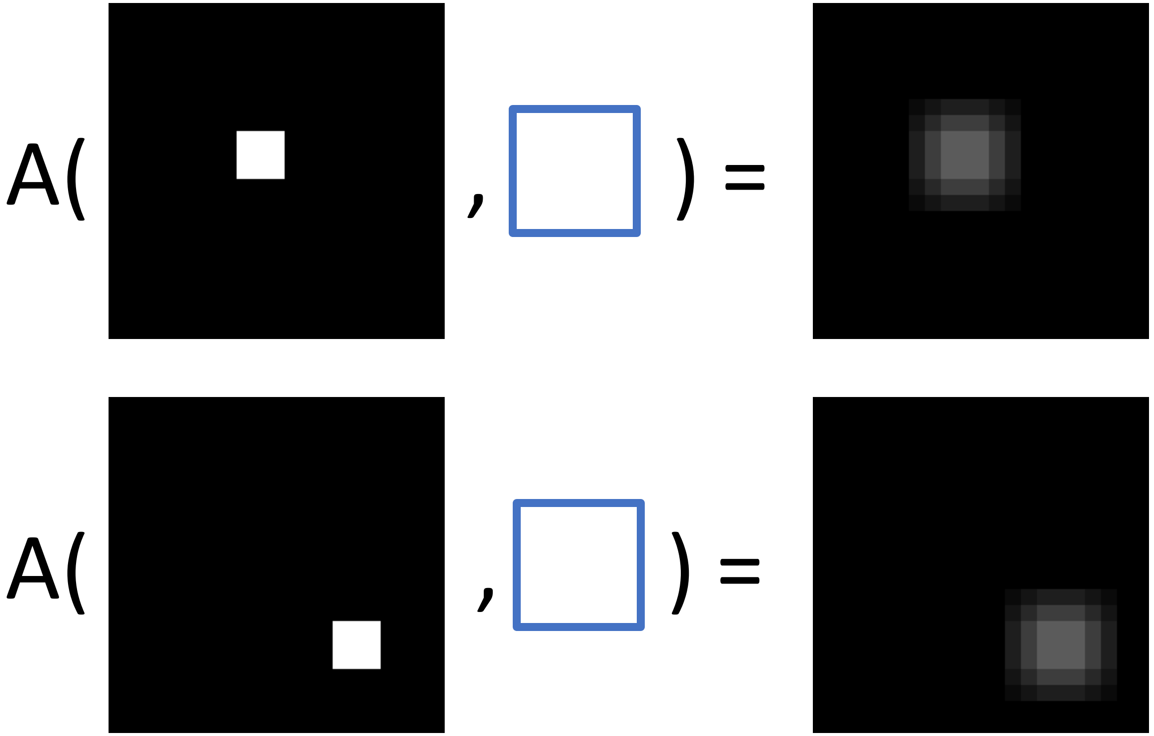
Slide credit: D Fouhey & J Johnson

Properties: Shift-Invariant

Assume: I image, f filter

Shift-invariant: shift(apply(I,f)) = apply(shift(I,f))

Intuitively: only depends on filter neighborhood



Slide credit: J Johnson

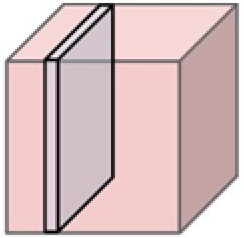
Max Pooling

Single depth slice

x

2	3	2	4
5	**6**	6	**8**
3	2	1	0
1	2	3	**4**

y



Max pooling with 2x2
kernel size and stride 2

6	8
3	4

Introduces **invariance** to small spatial shifts

No learnable parameters!

Global Average Pooling

x

2	3	2	4
5	6	6	8
3	2	1	0
1	2	3	4

y

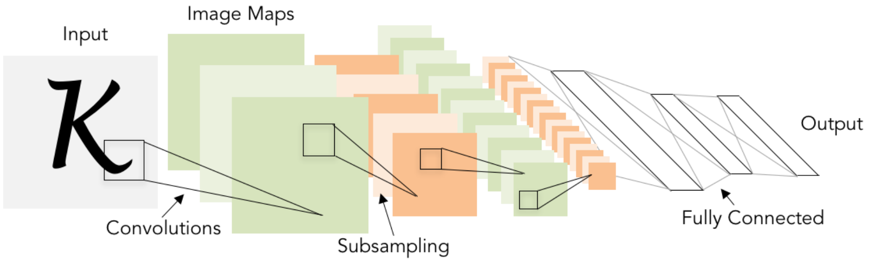
3.25

Average per channel (1 channel here).

Gradients?

Example: Lenet-5

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



As we go through the network:

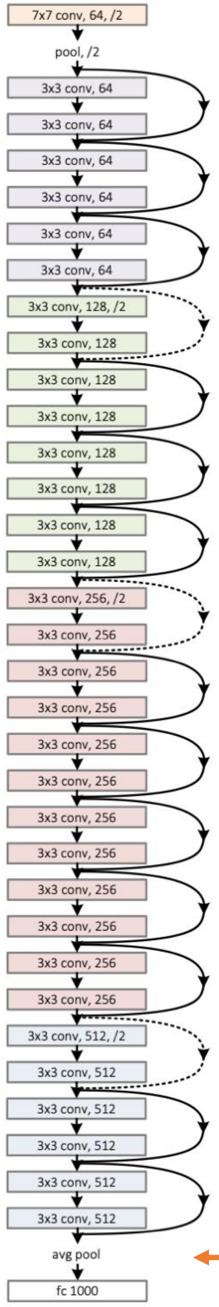
Spatial size **decreases**
(using pooling or strided conv)

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

Lecun et al, "Gradient-based learning applied to document recognition", 1998

Slide credit: D Fouhey & J Johnson

ResNet



Why is it a better idea than flattening the feature map before the classifier (a fully-connect layer)?

Number of parameters.

Spatial dimension of the input.

global average pooling

Batch Normalization

Idea: "Normalize" the outputs of each layer so they have zero mean and unit variance

Why? Helps reduce "internal covariate shift", improves optimization (hypothesis)

We can normalize a batch of activations like this:

$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}}$$

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

Batch Normalization

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

Input: $x \in \mathbb{R}^{N \times D}$

Learnable scale and shift parameters:

$$\gamma, \beta \in \mathbb{R}^{D}$$

Learning $\gamma = \sigma, \beta = \mu$ will recover the identity function (in expectation)

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

Per-channel mean, shape is D

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

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

Slide credit: D Fouhey & J Johnson

Batch Normalization: Test-Time

Input: $x \in \mathbb{R}^{N \times D}$

Learnable scale and shift parameters:

$\gamma, \beta \in \mathbb{R}^D$

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

$\mu_j =$ (Running) average of values seen during training — Per-channel mean, shape is D

$\sigma_j^2 =$ (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

Slide credit: D Fouhey & J Johnson

Today’s Class

- More about batch normalization layer
- Modern deep convolutional neural networks
- Training deep convolutional neural networks

Batch Normalization for ConvNets

Batch Normalization for **fully-connected** networks

$$x : N \times D$$

Normalize

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

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

$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

Batch Normalization for **convolutional** networks (Spatial Batchnorm, BatchNorm2D)

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

Normalize

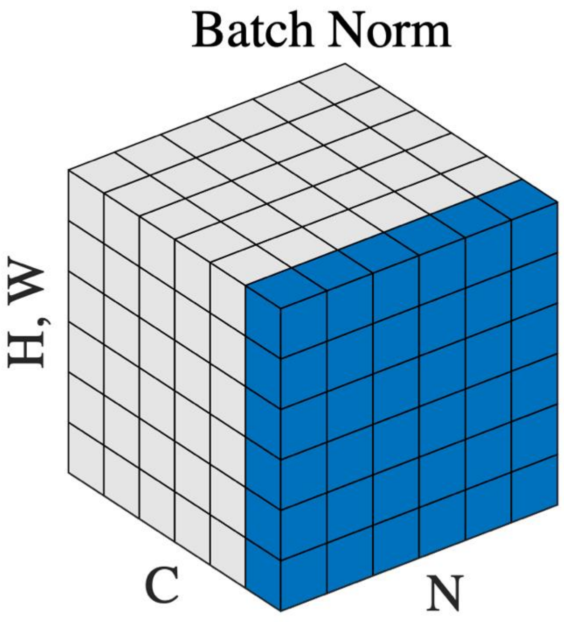
$$\mu, \sigma : 1 \times C \times 1 \times 1$$

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

$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

Slide credit: D Fouhey & J Johnson

Batch Normalization for ConvNets



Batch Normalization for **convolutional** networks (Spatial Batchnorm, BatchNorm2D)

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

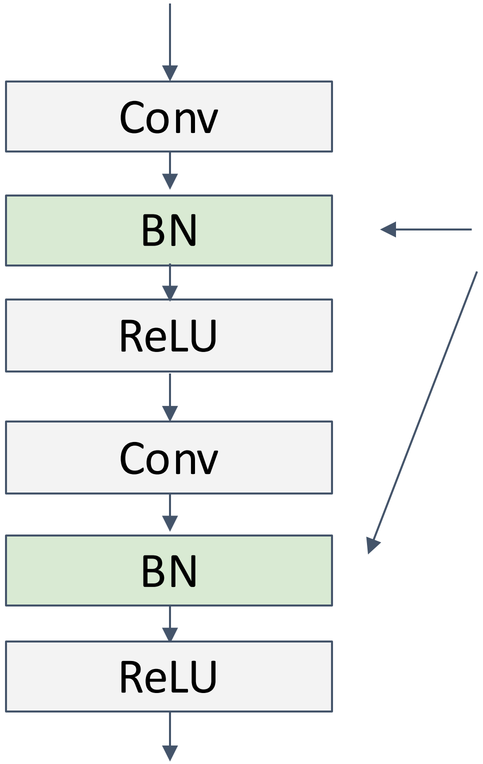
Normalize

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

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

$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

Batch Normalization

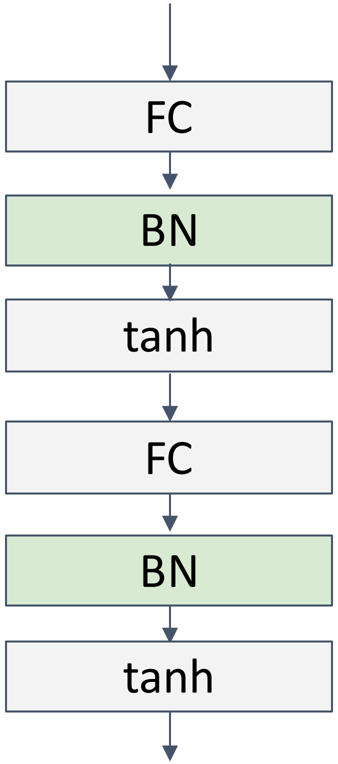


Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

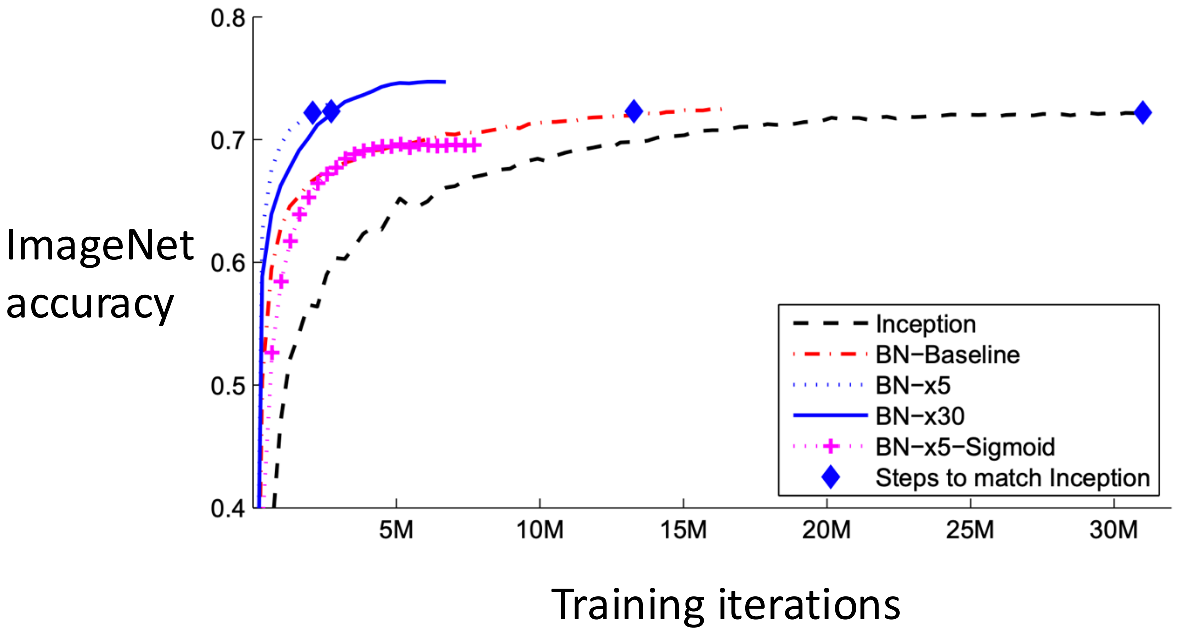
$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}}$$

Slide credit: D Fouhey & J Johnson

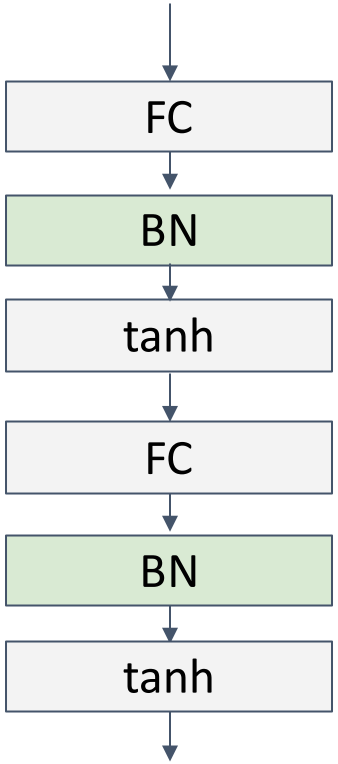
Batch Normalization



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



Batch Normalization

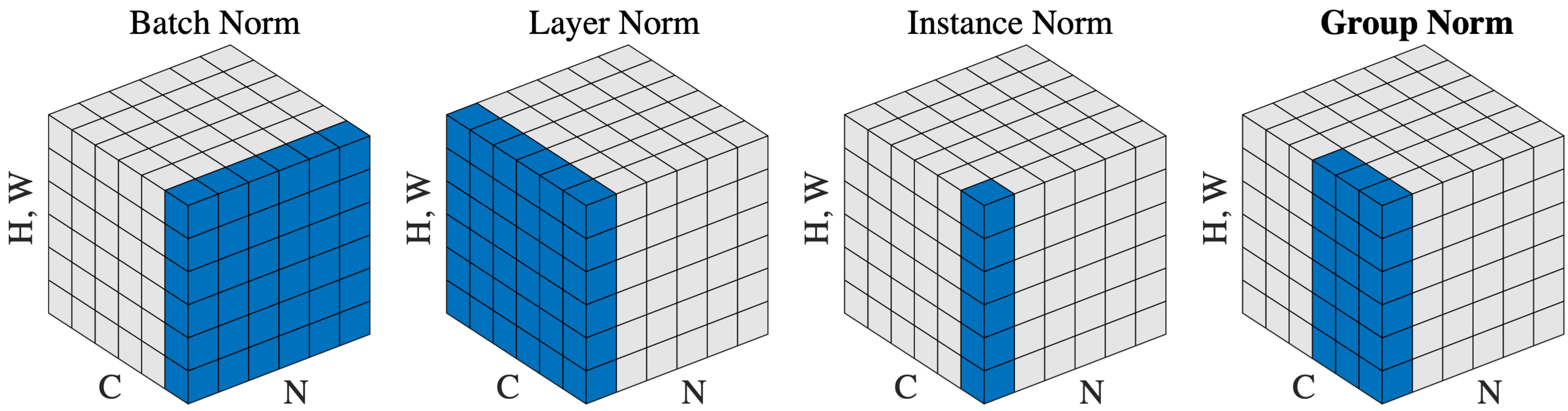


Ioffe and Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift”, ICML 2015

- Makes deep networks **much** easier to train!
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Free 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!

Slide credit: D Fouhey & J Johnson

Different Normalization Layers



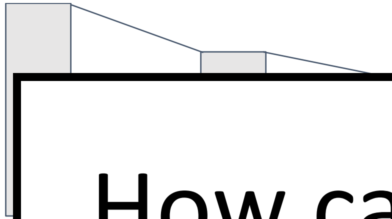
[Wu and He. Group Normalization. ECCV 2018. Best paper honorable mention.]

Devils in the details (mainly for PyTorch)

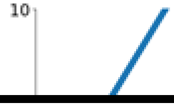
1. To reliably estimate BN statistics (running mean and average), you need at least 8 samples on each GPU
2. If you don't have enough samples on each GPU
   1. Synchronized BN layers
   2. Group normalization layers
3. Instance normalization layers are useful for some applications: such as style transfer, dense correspondence.

Convolutional Networks

Fully-Connected Layers

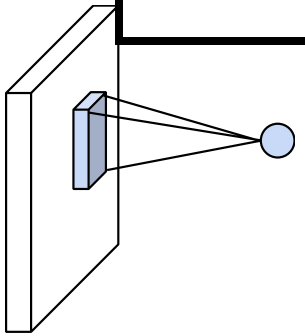


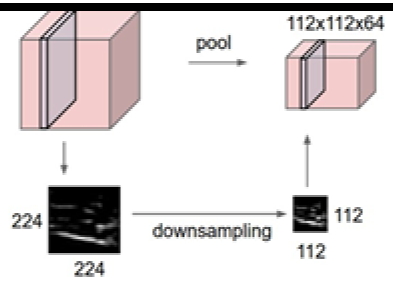
Activation Function



How can we combine these components into full architectures?

Convoluti



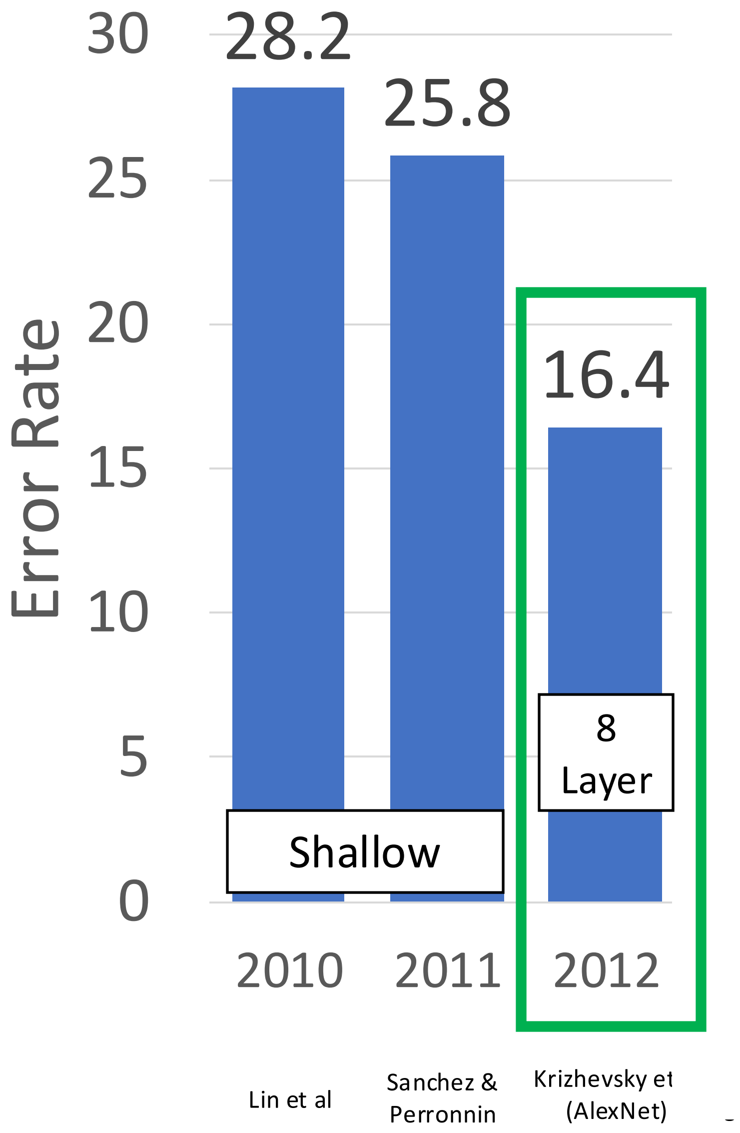


tion

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

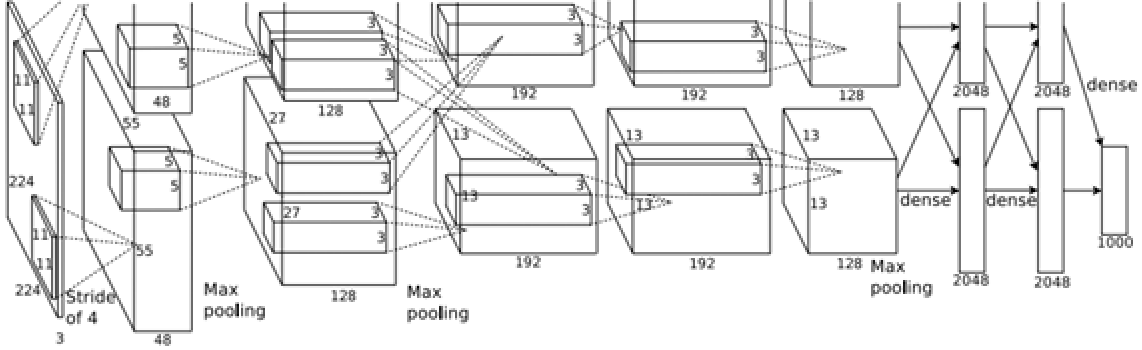
Slide credit: D Fouhey & J Johnson

ImageNet Classification Challenge



Slide credit: D Fouhey & J Johnson

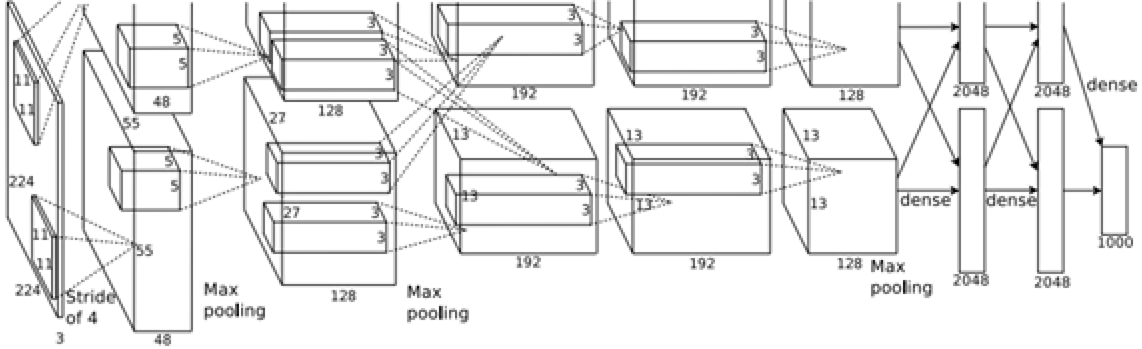
AlexNet



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

Slide credit: D Fouhey & J Johnson

AlexNet



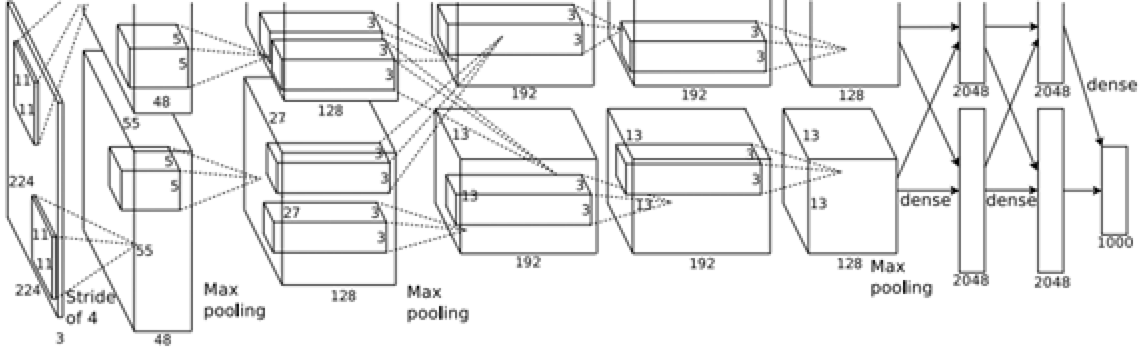
227 x 227 inputs
5 Convolutional layers
Max pooling
3 fully-connected layers
ReLU nonlinearities

Used “Local response normalization”;
Not used anymore

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

Slide credit: D Fouhey & J Johnson

AlexNet



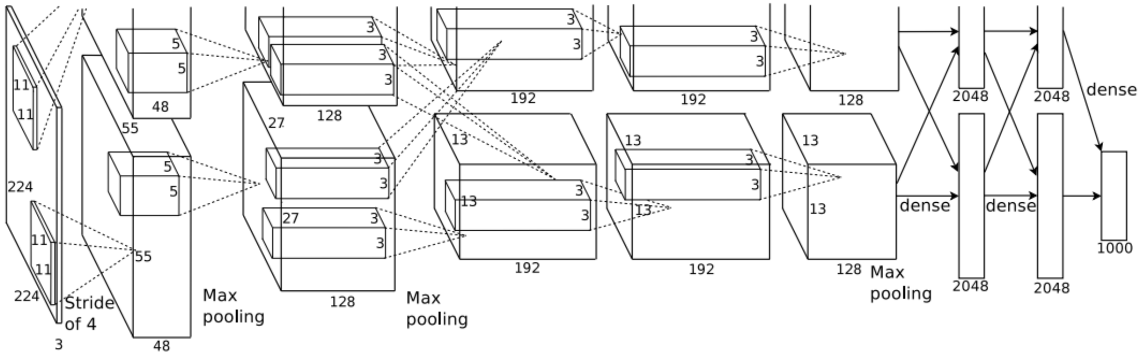
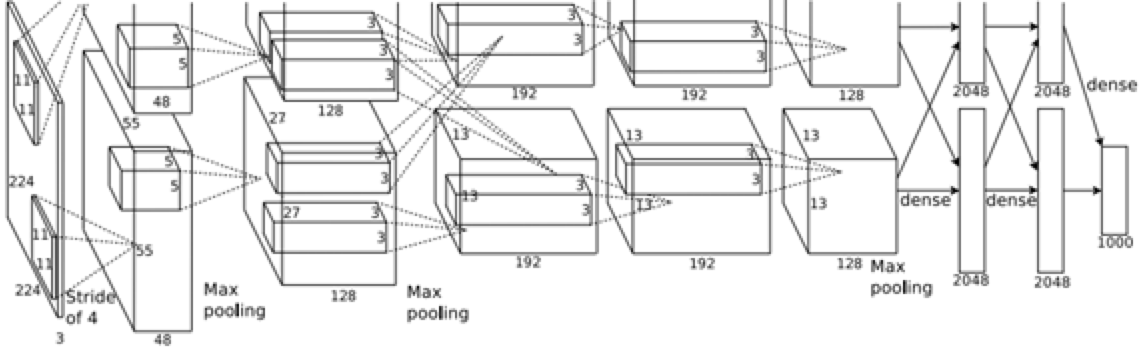


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input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

Slide credit: D Fouhey & J Johnson

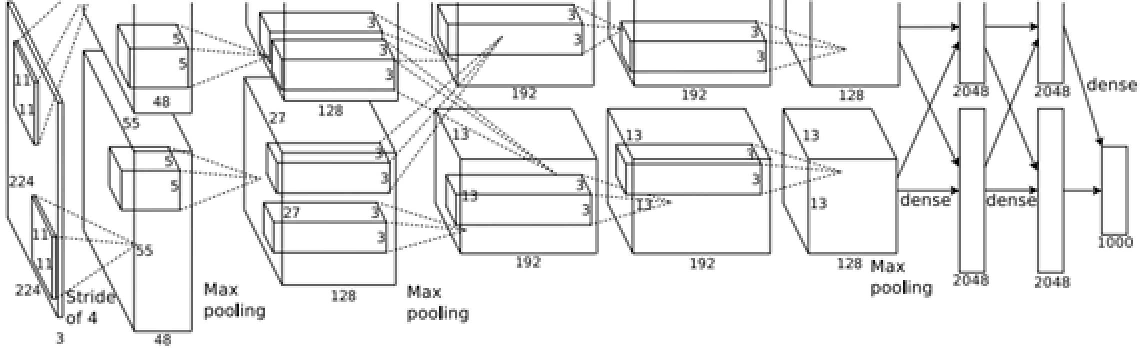
AlexNet



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

Slide credit: D Fouhey & J Johnson

AlexNet

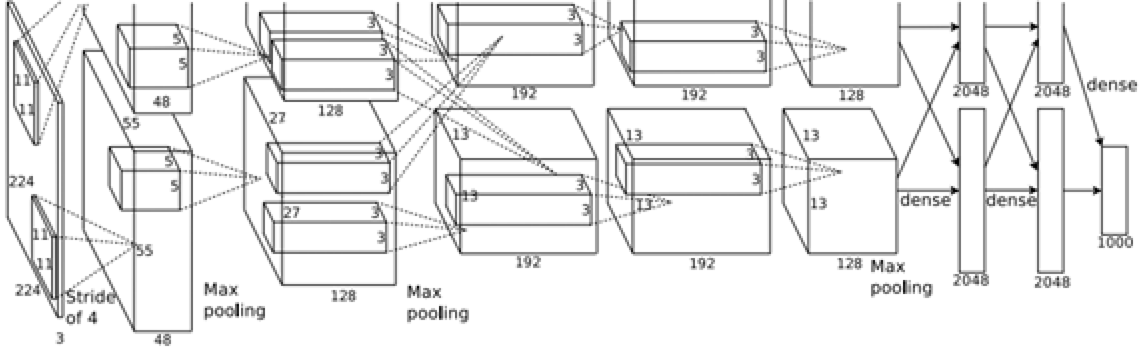


	Input size		Layer				Output size	
Layer	C	H / W	filters	kernel	stride	pad	C	H / W
conv1	3	227	64	11	4	2	64	?

Recall: Output channels = number of filters

Slide credit: D Fouhey & J Johnson

AlexNet

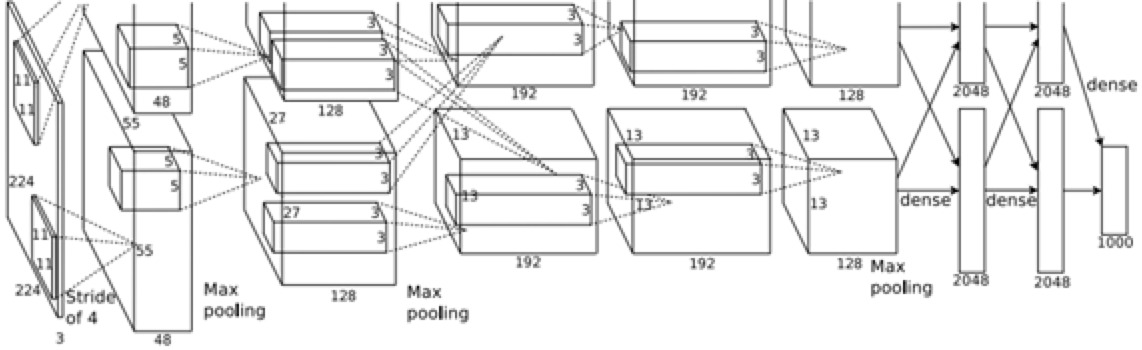


	Input size		Layer				Output size	
Layer	C	H / W	filters	kernel	stride	pad	C	H / W
conv1	3	227	64	11	4	2	64	56

$$\begin{aligned} \text{Recall: } W' &= \text{ceil}((W - K + 1 + 2P) / S) \\ &= \text{ceil}((227 - 11 + 1 + 2*2) / 4) \\ &= \text{ceil}(221/4) = 56 \end{aligned}$$

Slide credit: D Fouhey & J Johnson

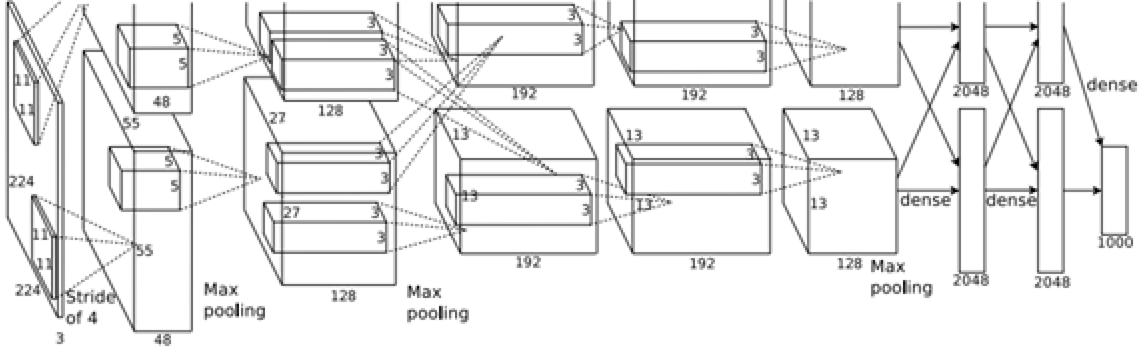
AlexNet



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

Slide credit: D Fouhey & J Johnson

AlexNet



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

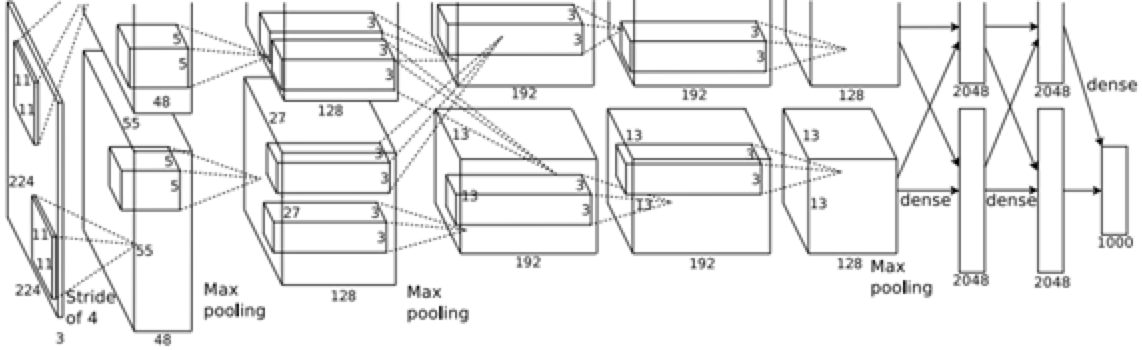
Number of output elements = C * H’ * W’
= 64*56*56 = 200,704

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

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

Slide credit: D Fouhey & J Johnson

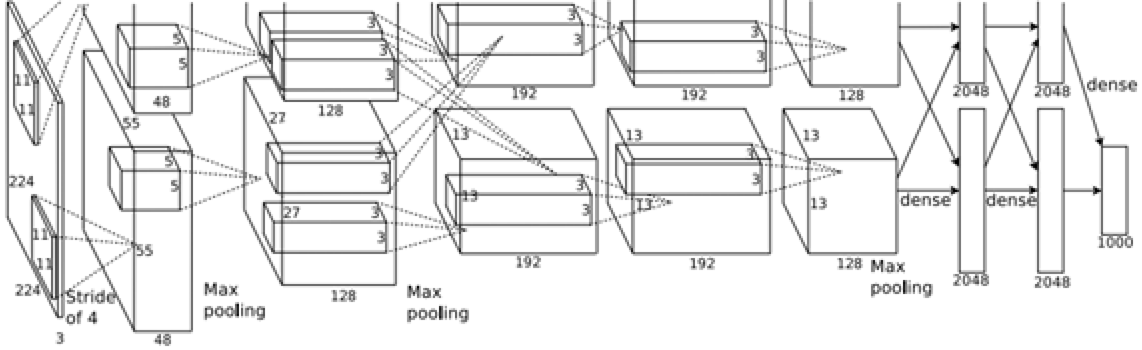
AlexNet



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

Slide credit: D Fouhey & J Johnson

AlexNet



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

Weight shape = C_{out} x C_{in} x K x K

= 64 x 3 x 11 x 11

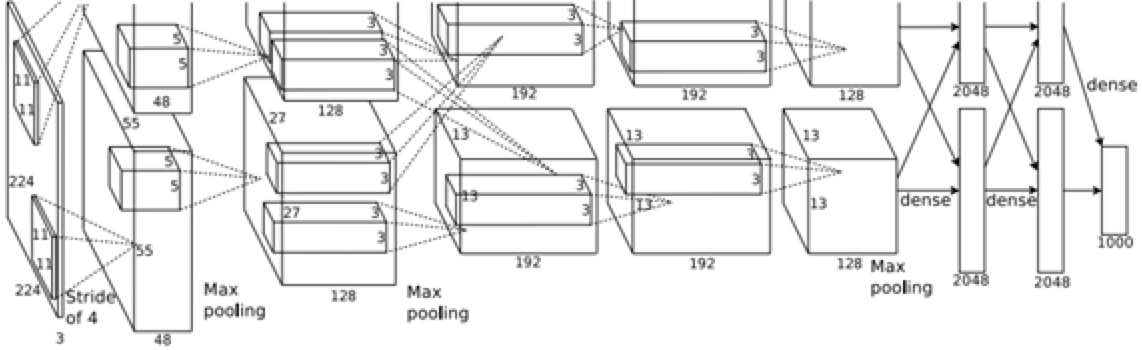
Bias shape = C_{out} = 64

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

= **23,296**

Slide credit: D Fouhey & J Johnson

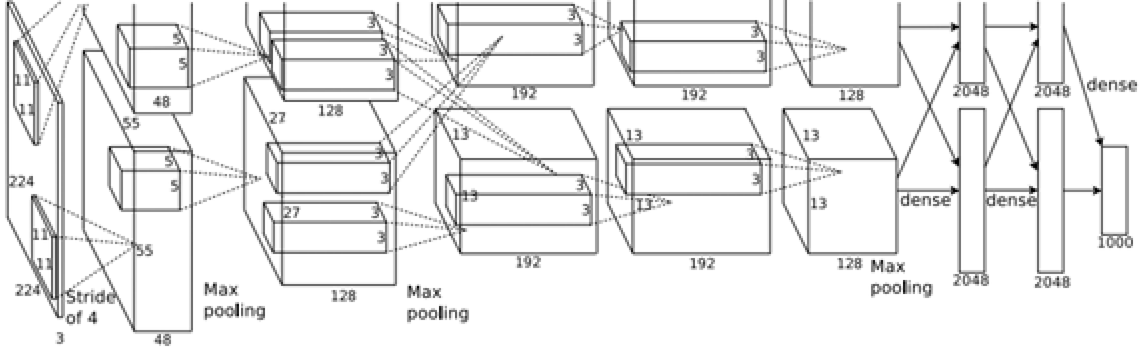
AlexNet



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

Slide credit: D Fouhey & J Johnson

AlexNet



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

Number of floating point operations (multiply+add)

= (number of output elements) * (ops per output elem)

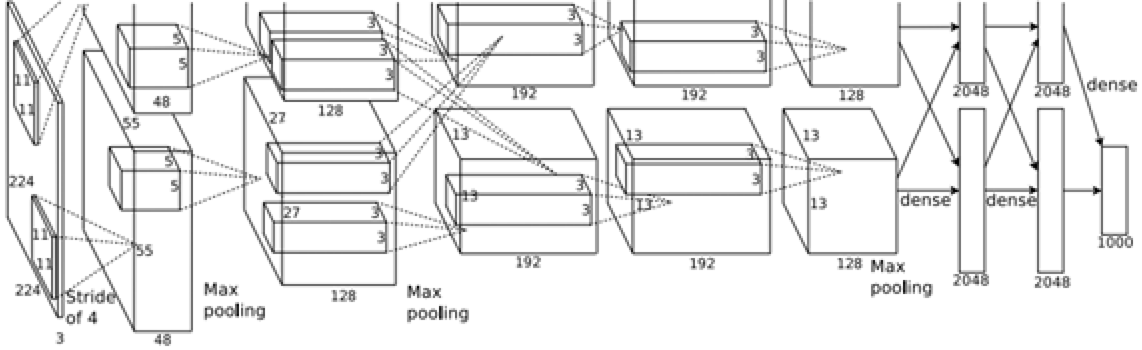
= $(C_{out} \times H' \times W') * (C_{in} \times K \times K)$

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

= 200,704 * 363

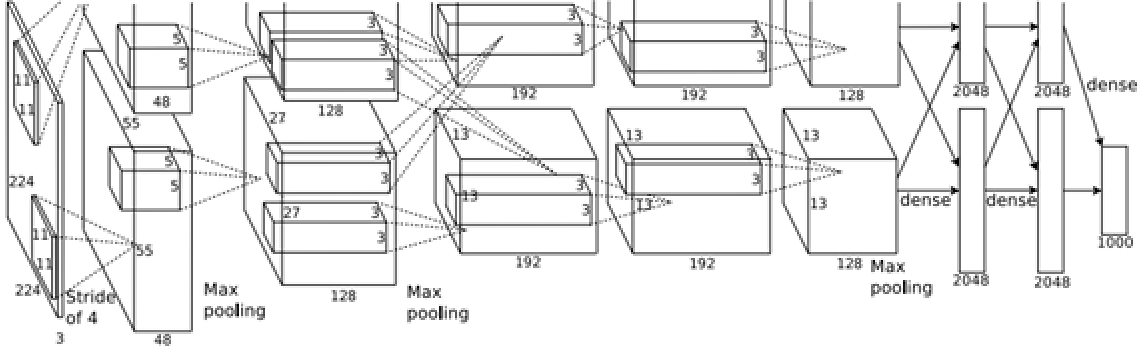
= **72,855,552**

AlexNet



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

AlexNet



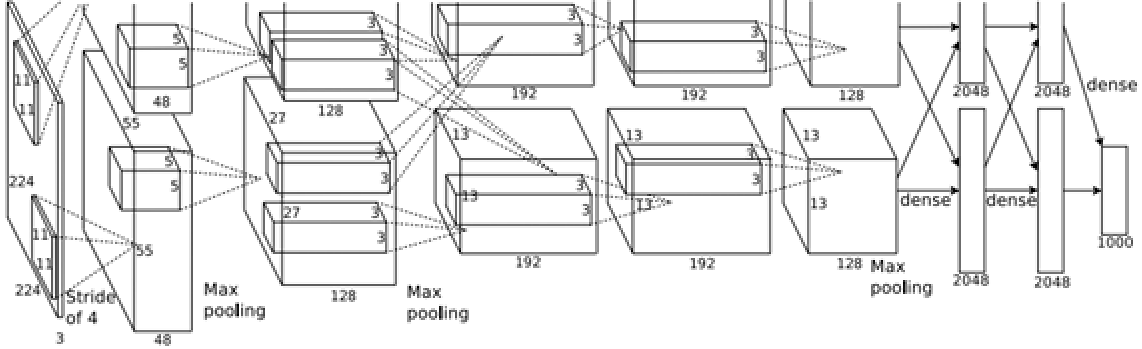
	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27			

For pooling layer:

#output channels = #input channels = 64

W' = ceil((W – K + 1) / S)
= ceil(54 / 2) = ceil(27) = **27**

AlexNet



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

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

Bytes per elem = 4

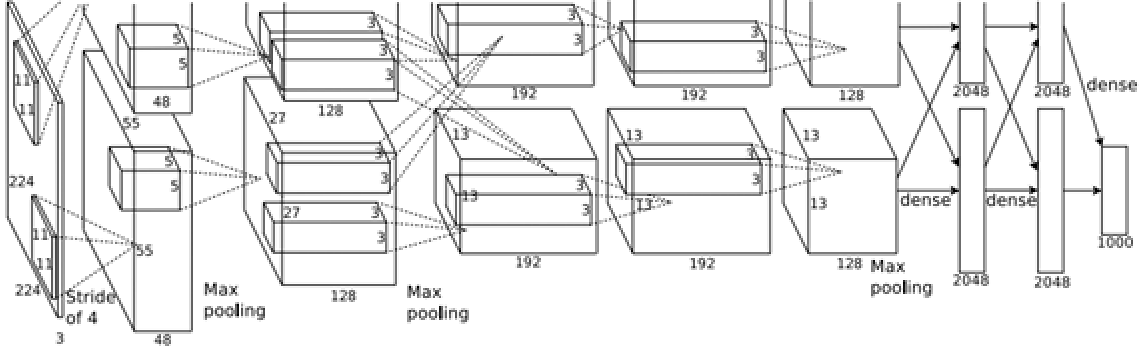
KB = C_{out} * H' * W' * 4 / 1024

= 64 * 27 * 27 * 4 / 1024

= **182.25**

Slide credit: D Fouhey & J Johnson

AlexNet

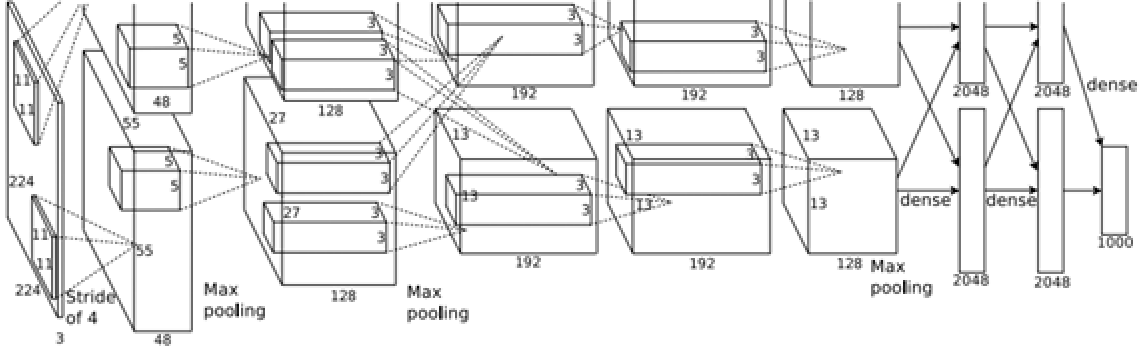


	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	?

Pooling layers have no learnable parameters!

Slide credit: D Fouhey & J Johnson

AlexNet



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

Floating-point ops for pooling layer

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

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

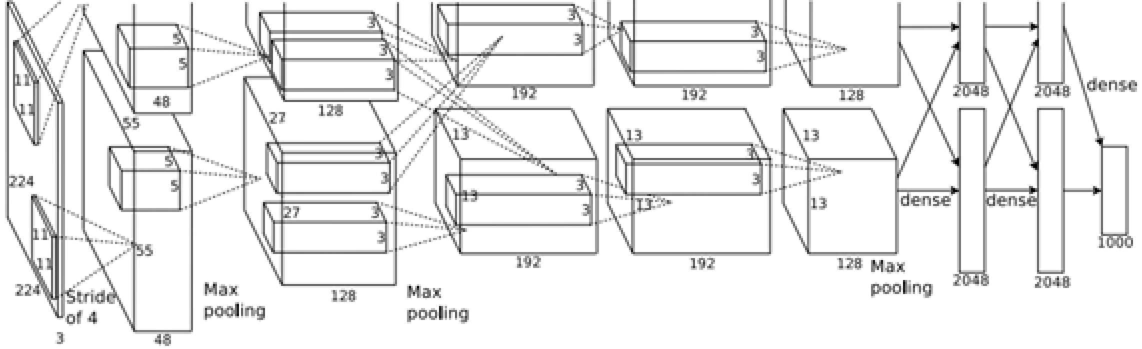
= (64 * 27 * 27) * (3 * 3)

= 419,904

= **0.4 MFLOP**

Slide credit: D Fouhey & J Johnson

AlexNet



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

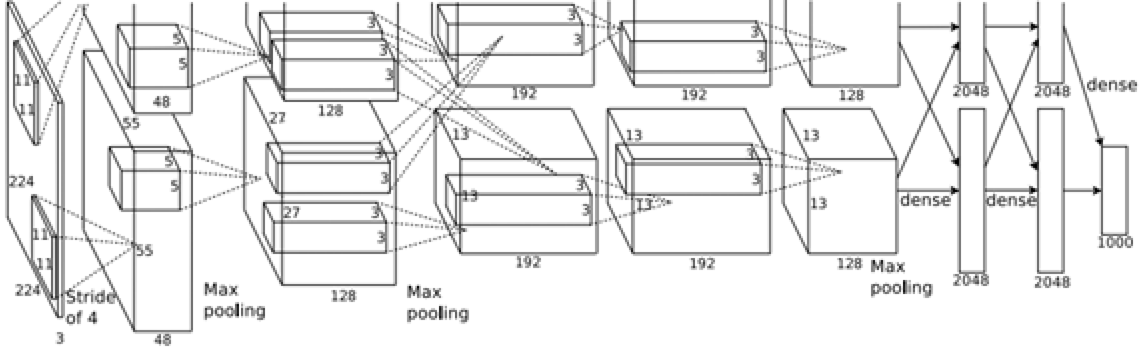
Flatten output size = C_{in} x H x W

= 256 * 6 * 6

= **9216**

Slide credit: D Fouhey & J Johnson

AlexNet



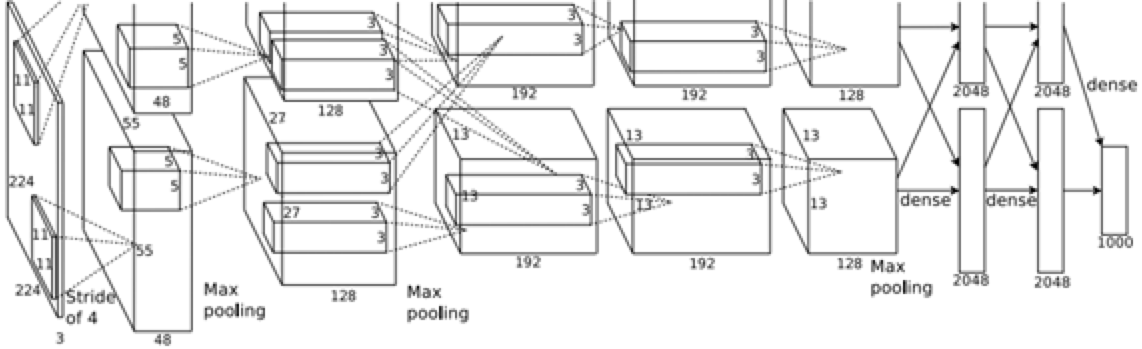
	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H / W	memory (KB)	params (k)	flop (M)
conv1	3	227	64	11	4	2	64	56	784	23	73
pool1	64	56		3	2	0	64	27	182	0	0
conv2	64	27	192	5	1	2	192	27	547	307	224
pool2	192	27		3	2	0	192	13	127	0	0
conv3	192	13	384	3	1	1	384	13	254	664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

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

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

Slide credit: D Fouhey & J Johnson

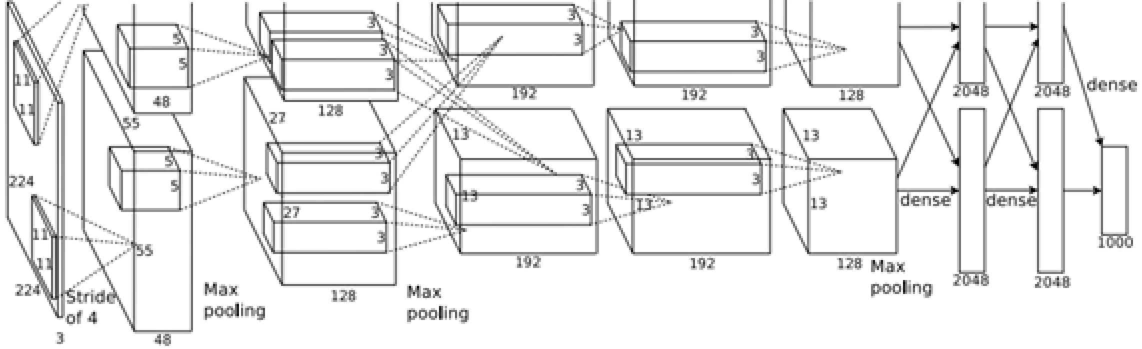
AlexNet



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

Slide credit: D Fouhey & J Johnson

AlexNet



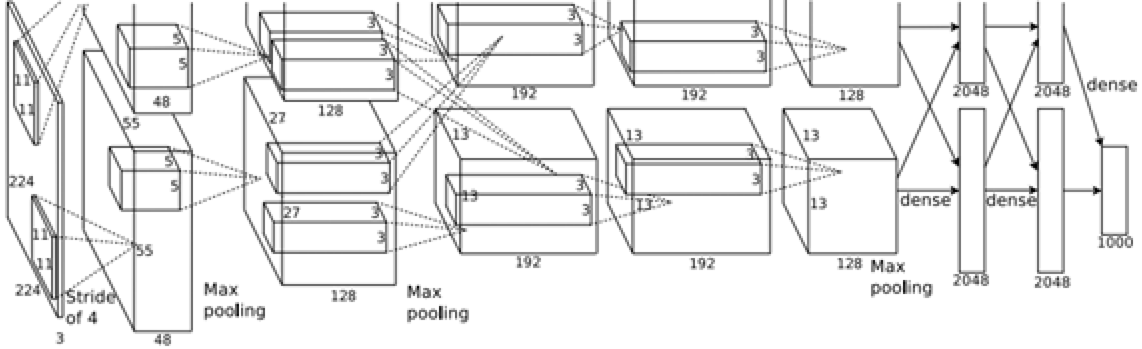
How to choose this?

Trial and error =(

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

Slide credit: D Fouhey & J Johnson

AlexNet

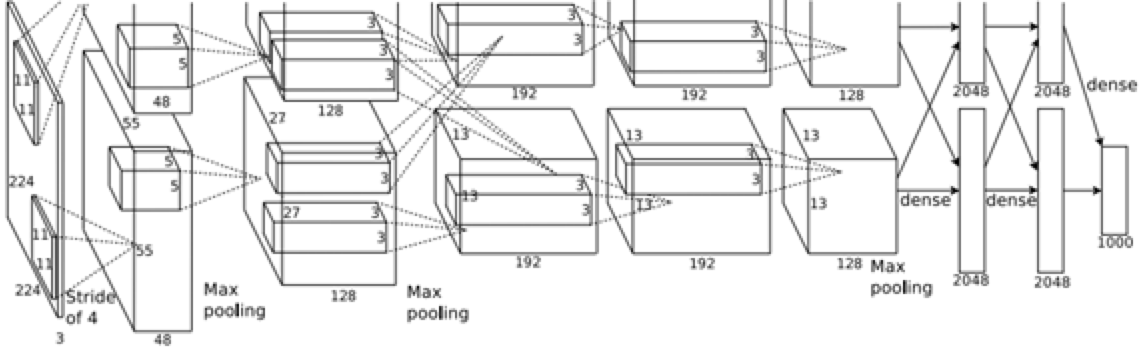


Interesting trends here!

	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

Slide credit: D Fouhey & J Johnson

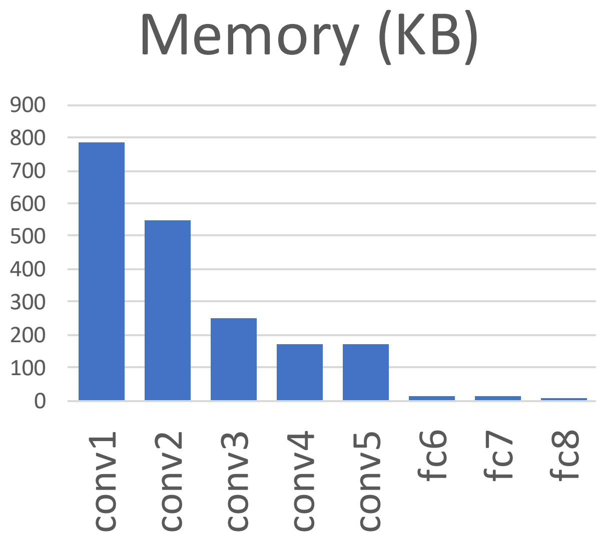
AlexNet

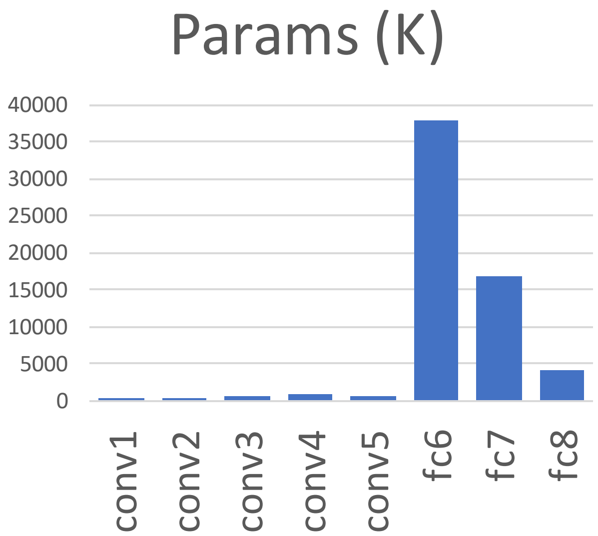


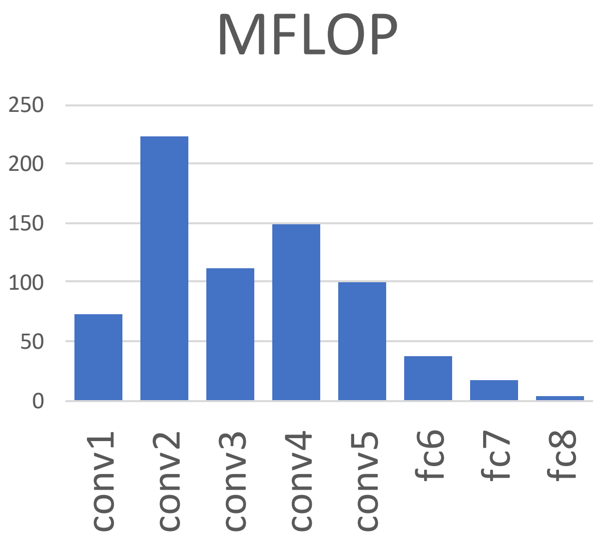
Most of the **memory usage** is in the early convolution layers

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

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

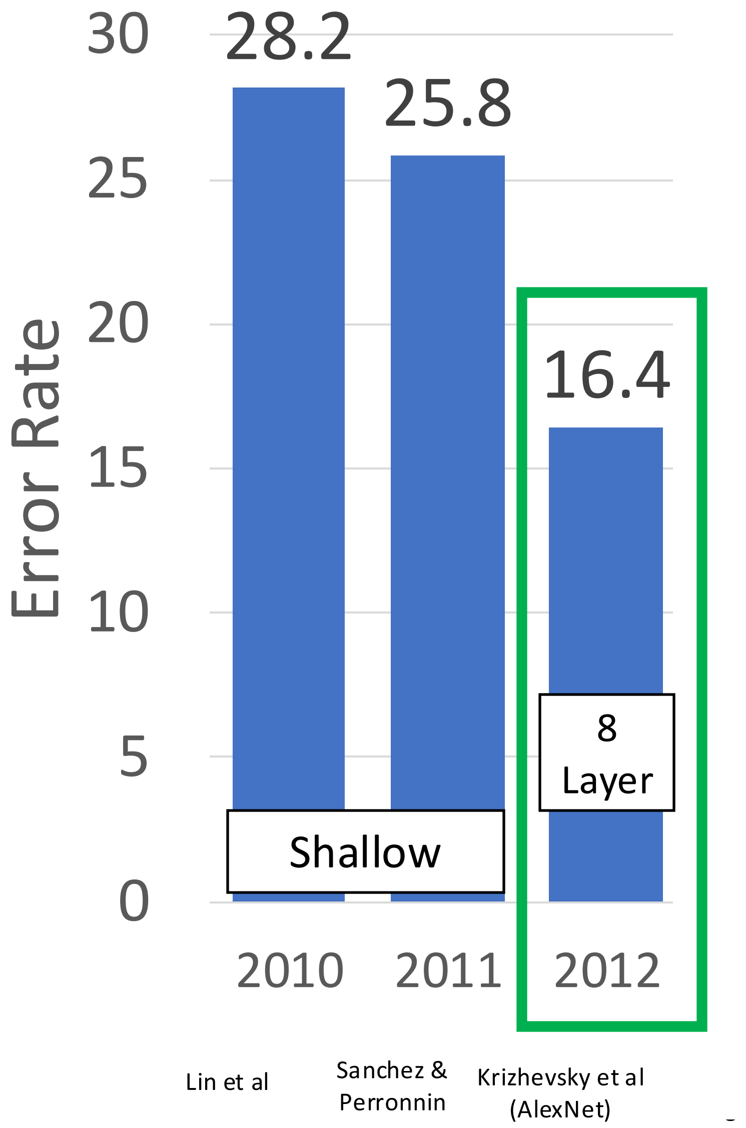






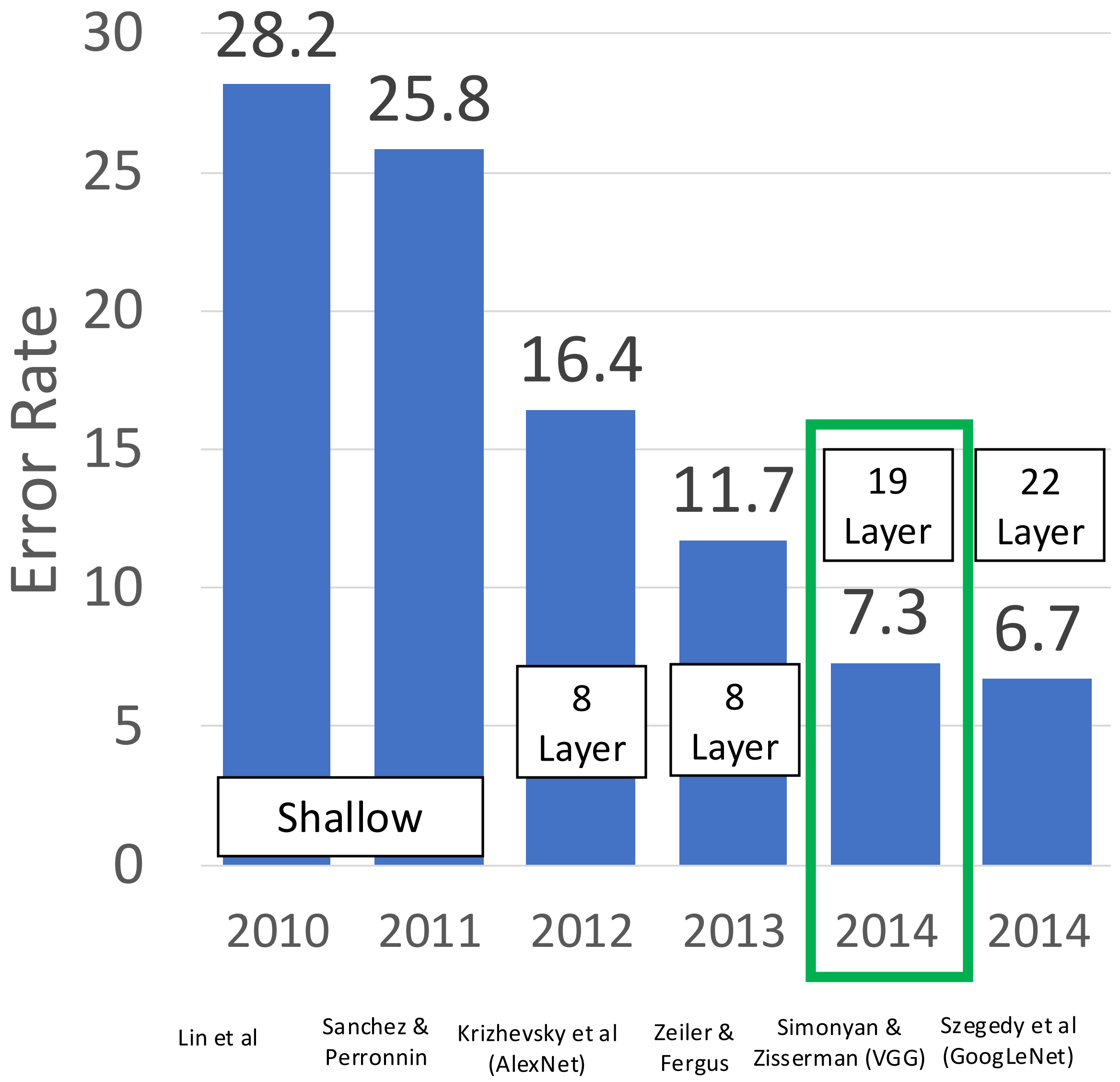
Slide credit: D Fouhey & J Johnson

ImageNet Classification Challenge



Slide credit: D Fouhey & J Johnson

ImageNet Classification Challenge



Slide credit: D Fouhey & J Johnson

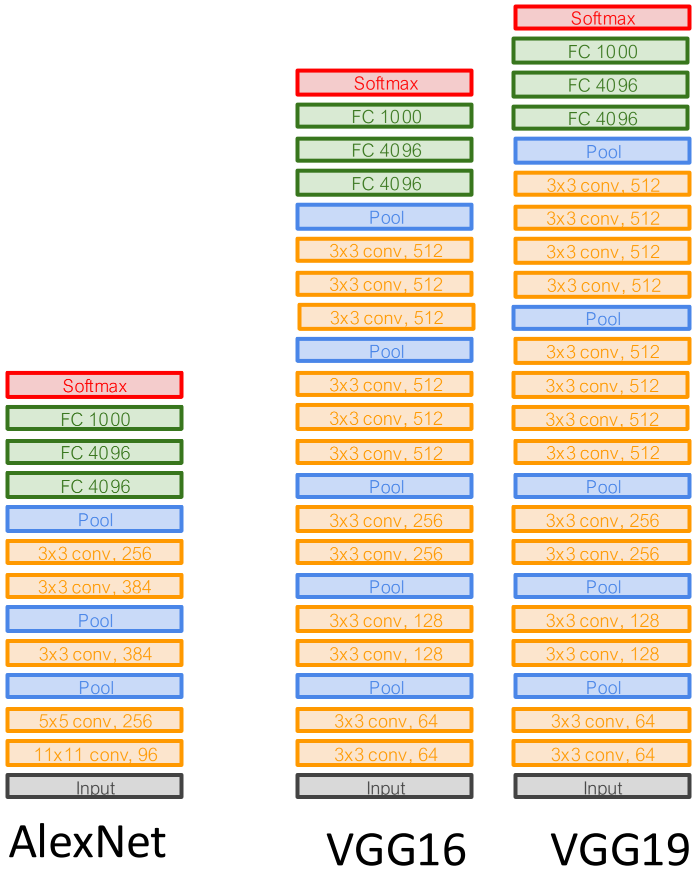
VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels



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

Slide credit: D Fouhey & J Johnson

VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

Network has 5 convolutional **stages**:

Stage 1: conv-conv-pool

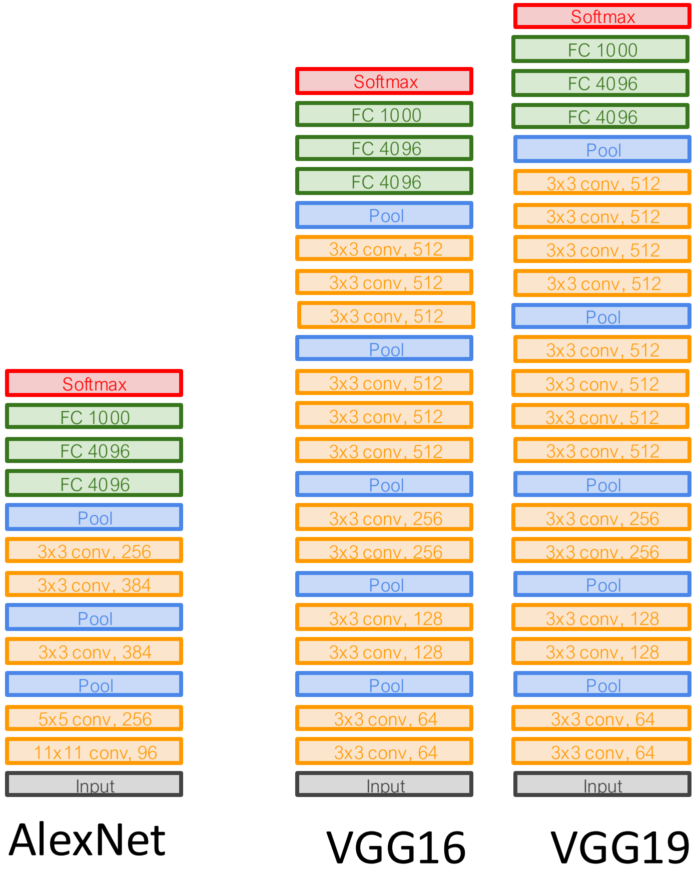
Stage 2: conv-conv-pool

Stage 3: conv-conv-pool

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

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

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



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

Slide credit: D Fouhey & J Johnson

VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

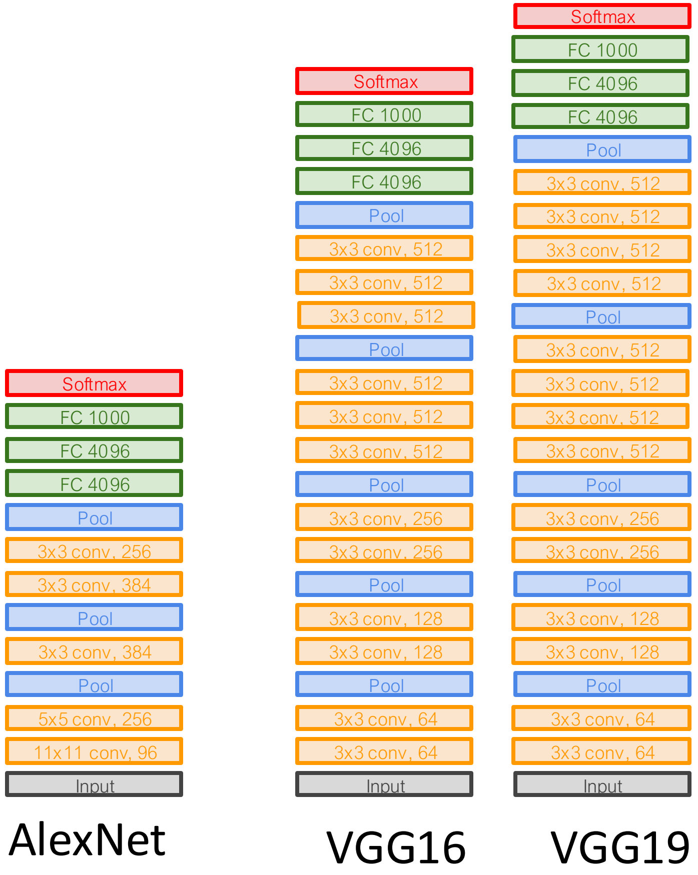
After pool, double #channels

<u>Option 1:</u>

Conv(5x5, C -> C)

Params: $25C^2$

FLOPs: $25C^2HW$



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

Slide credit: D Fouhey & J Johnson

VGG: Deeper Networks, Regular Design

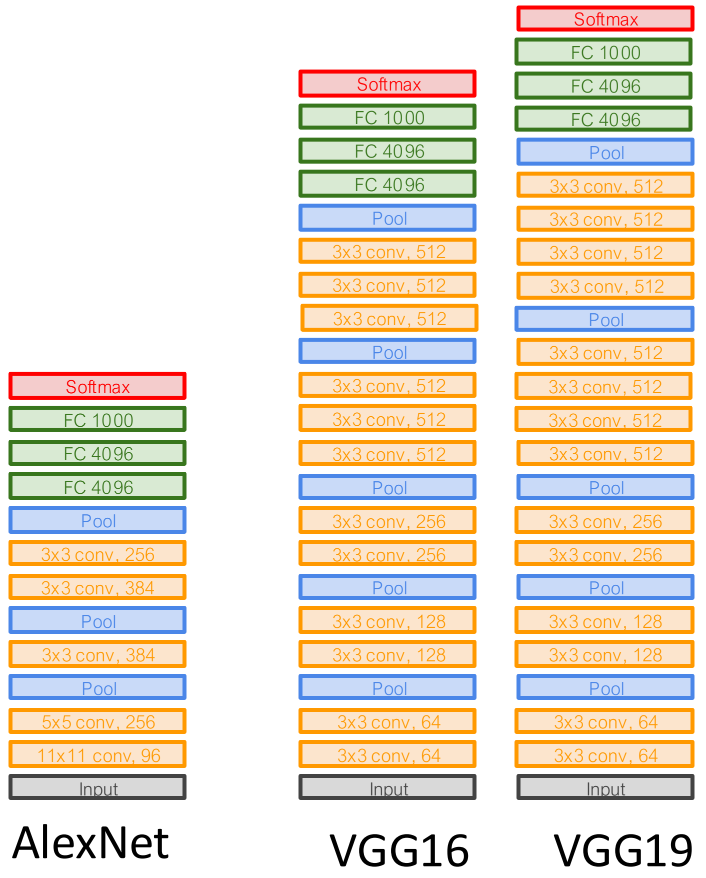
<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

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



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

Slide credit: D Fouhey & J Johnson

VGG: Deeper Networks, Regular Design

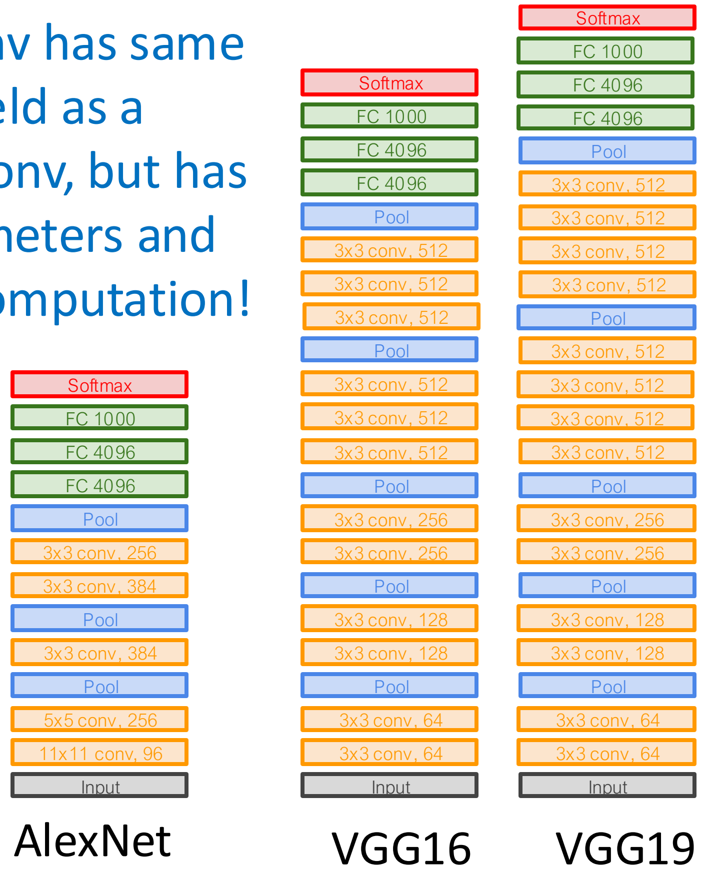
<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

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



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

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

Slide credit: D Fouhey & J Johnson

VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

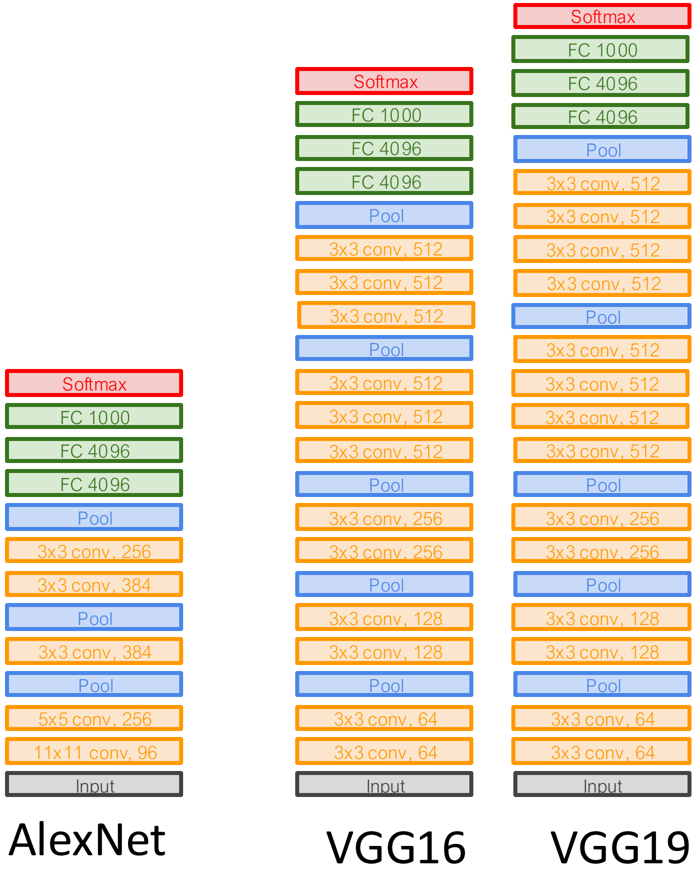
Input: C x 2H x 2W

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

Memory: 4HWC

Params: $9C^2$

FLOPs: $36HWC^2$



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

Slide credit: D Fouhey & J Johnson

VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

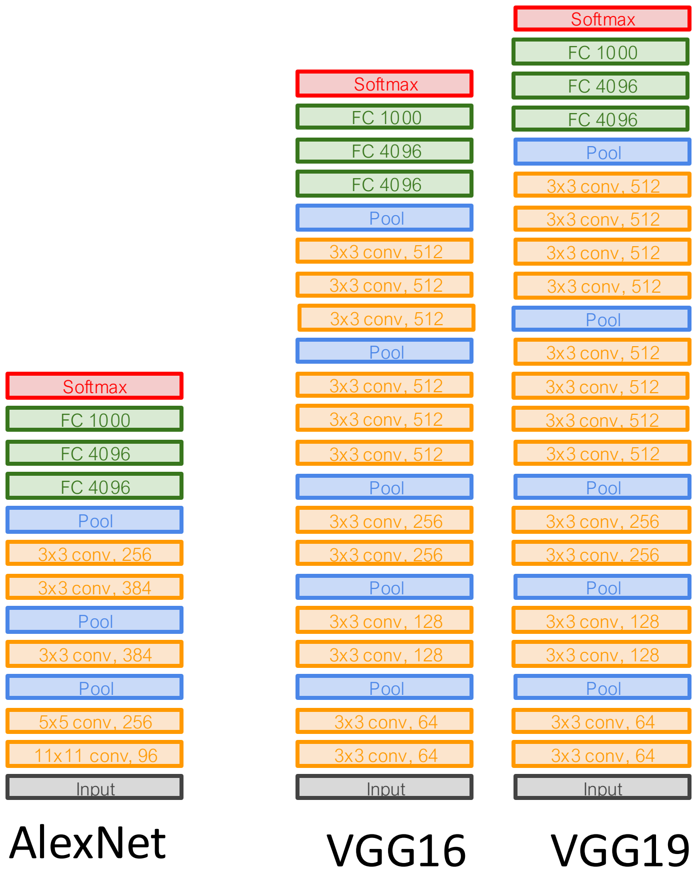
After pool, double #channels

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

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

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

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



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

Slide credit: D Fouhey & J Johnson

VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

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

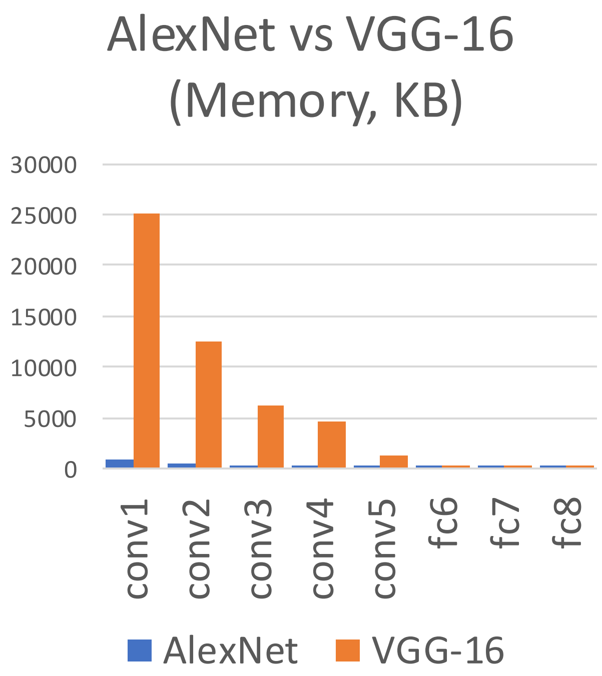
Input: C x 2H x 2W	Input: 2C x H x W
Layer: Conv(3x3, C->C)	Conv(3x3, 2C -> 2C)
Memory: 4HWC	Memory: 2HWC
Params: $9C^2$	Params: $36C^2$
FLOPs: $36HWC^2$	FLOPs: $36HWC^2$

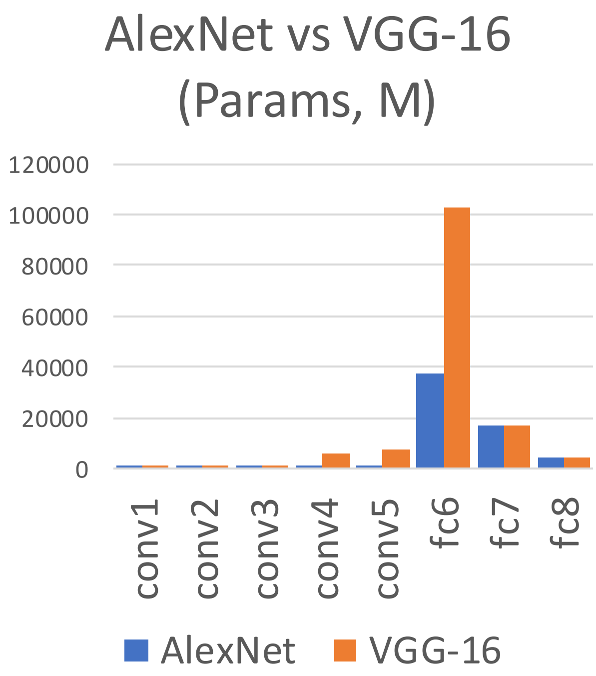
AlexNet VGG16 VGG19

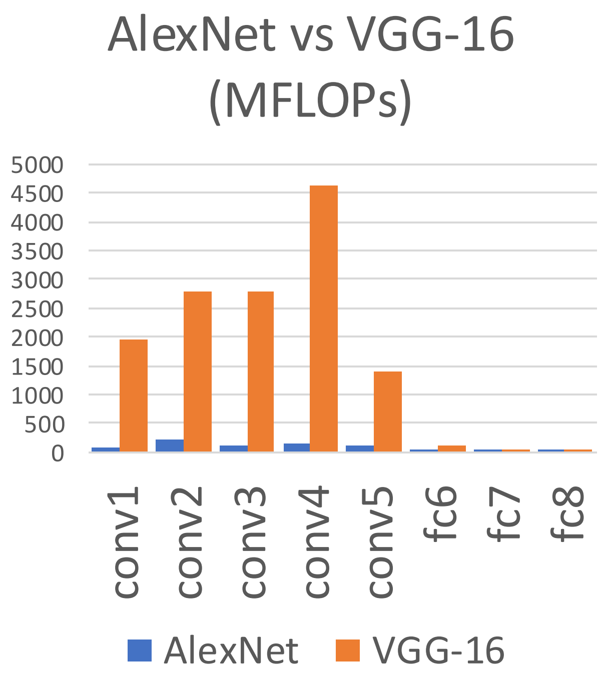
Simonyan and Zissermann, "Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR 2015

Slide credit: D Fouhey & J Johnson

AlexNet vs VGG-16: Much Bigger!







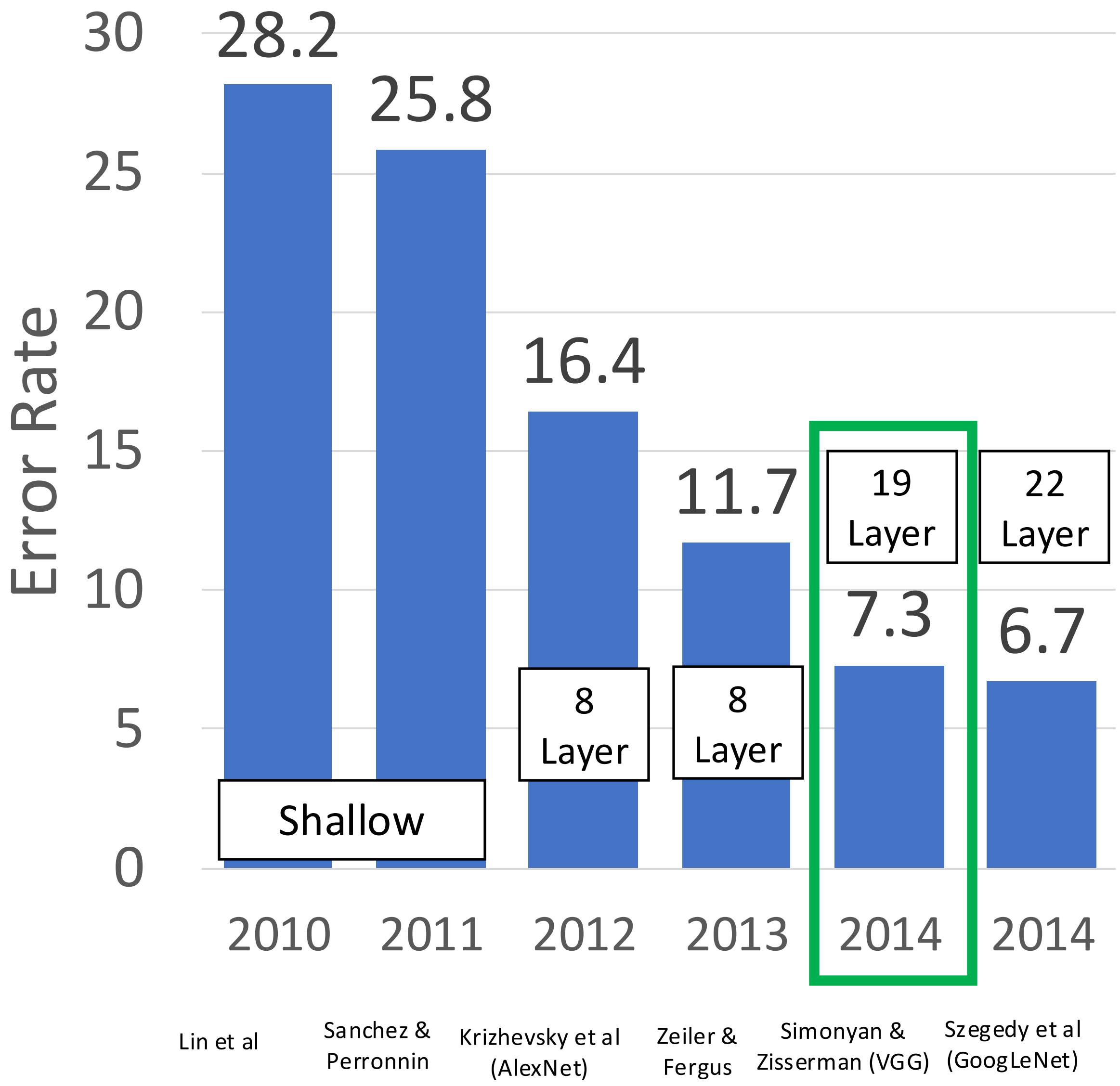
AlexNet total: 1.9 MB
VGG-16 total: 48.6 MB (25x)

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

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

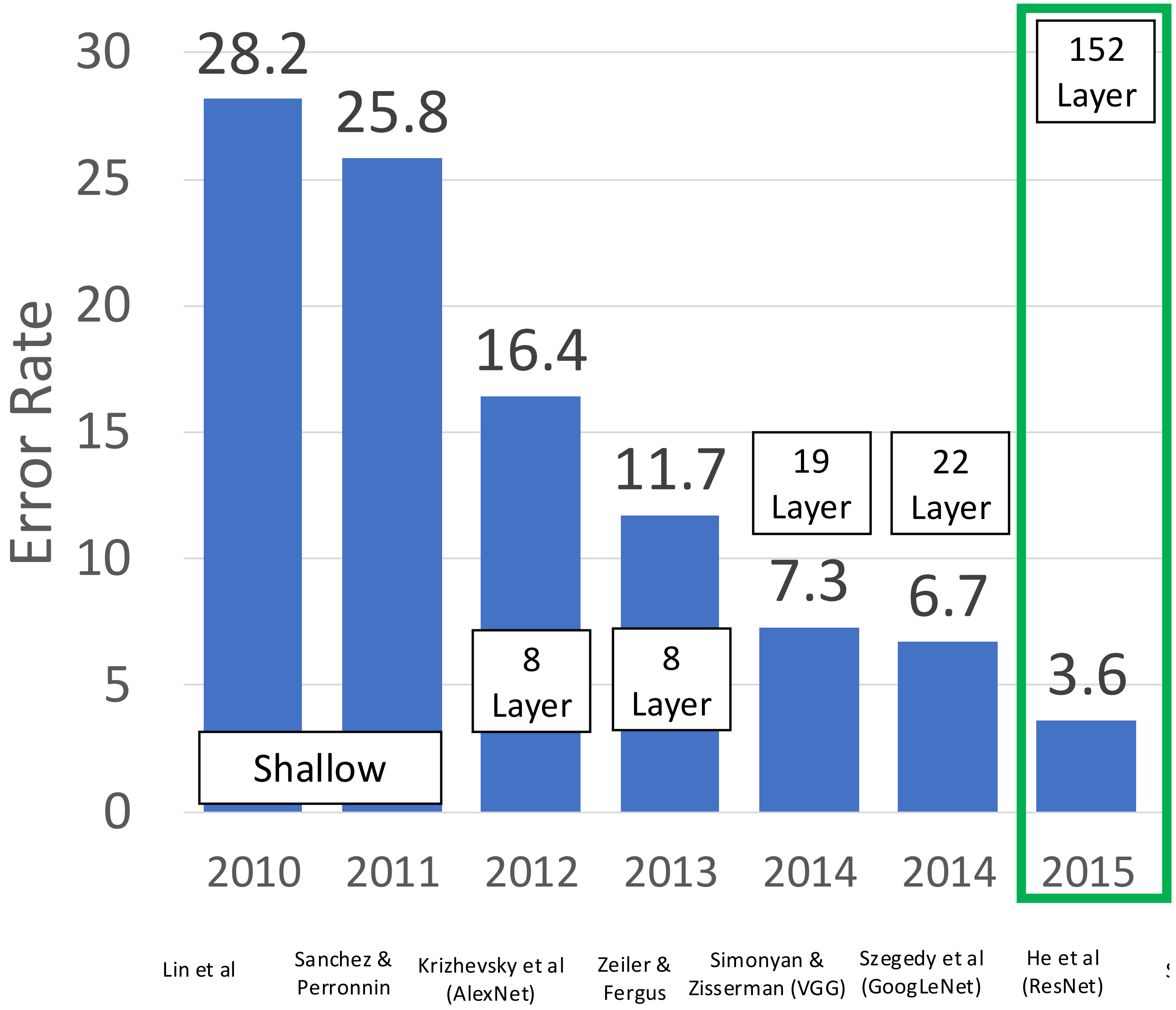
Slide credit: D Fouhey & J Johnson

ImageNet Classification Challenge



Slide credit: D Fouhey & J Johnson

ImageNet Classification Challenge



Slide credit: D Fouhey & J Johnson

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

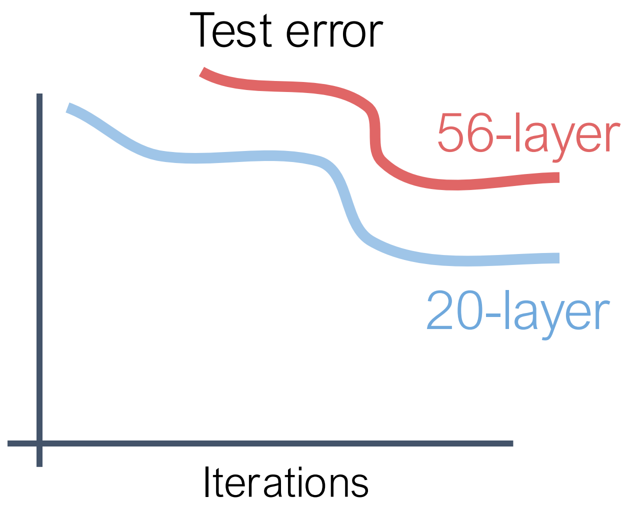
Slide credit: D Fouhey & J Johnson

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

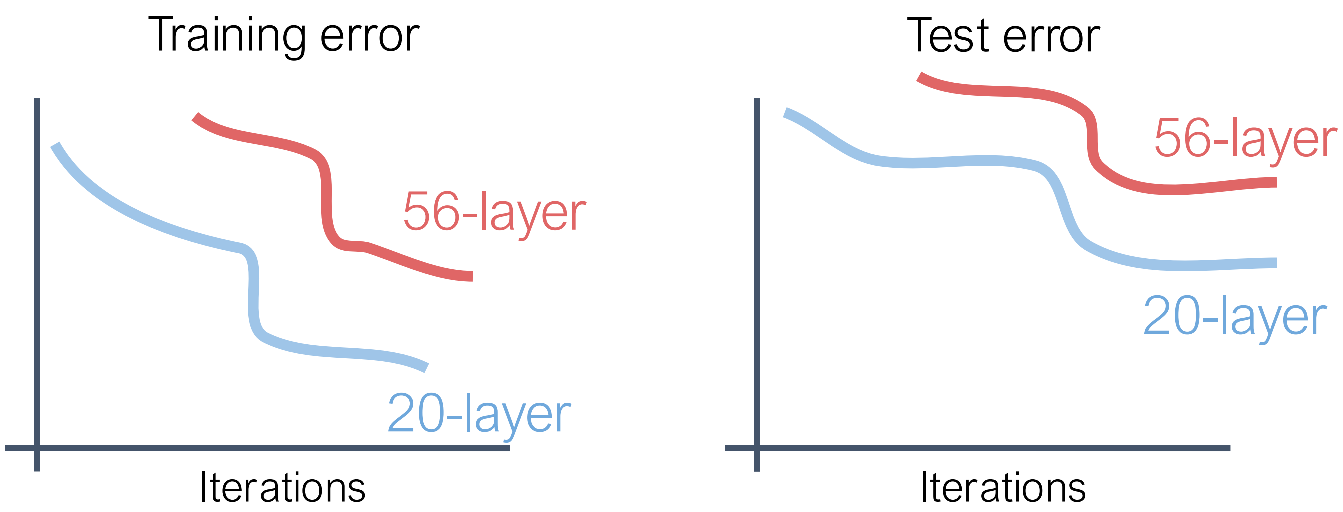


He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Slide credit: D Fouhey & J Johnson

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

Slide credit: D Fouhey & J Johnson

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

Slide credit: D Fouhey & J Johnson

Residual Networks

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

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

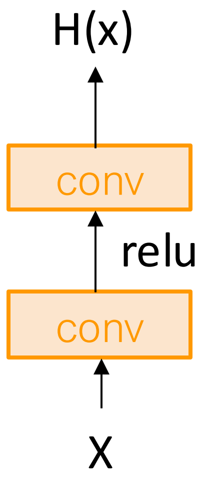
Hypothesis: This is an <u>optimization</u> problem. Deeper models are harder to optimize, and in particular don't learn identity functions to emulate shallow models

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

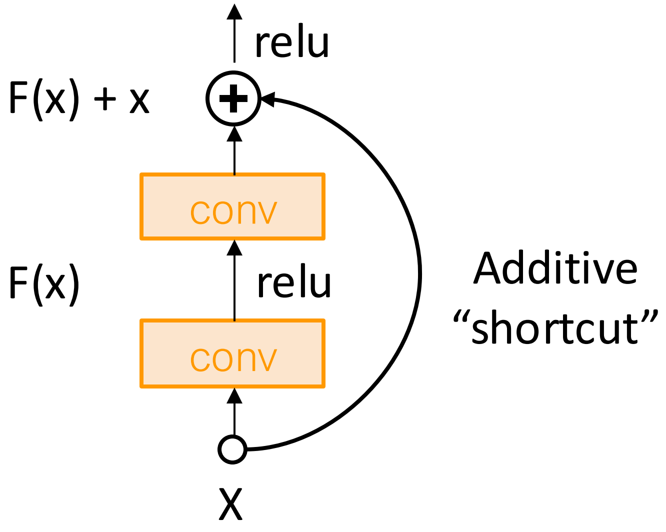
He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Residual Networks

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



"Plain" block



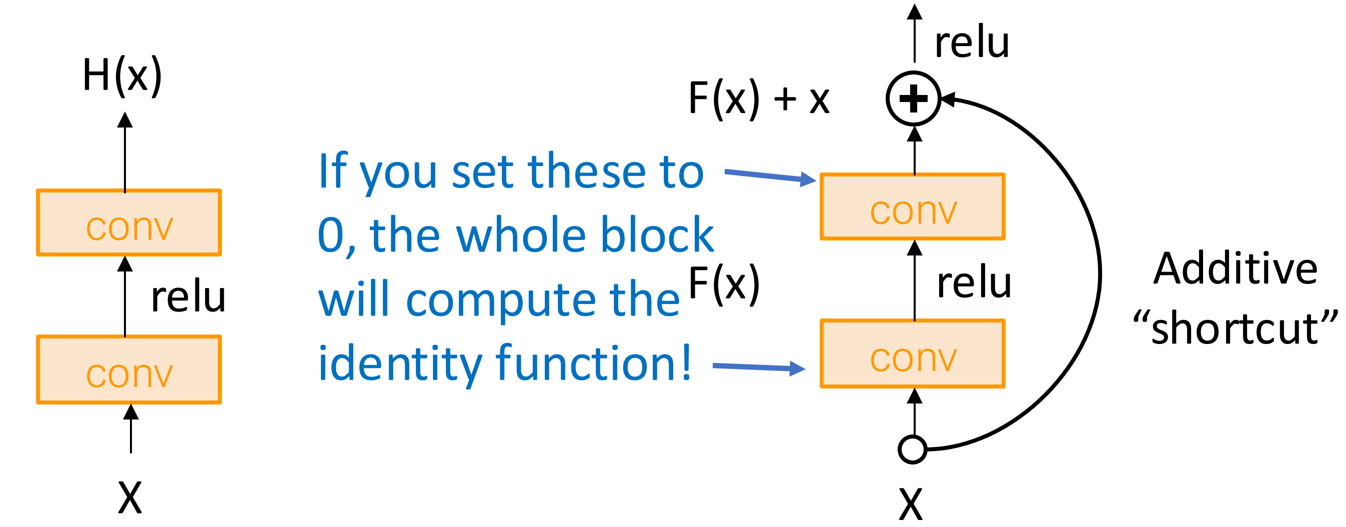
Residual Block

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

Slide credit: D Fouhey & J Johnson

Residual Networks

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



“Plain” block

Residual Block

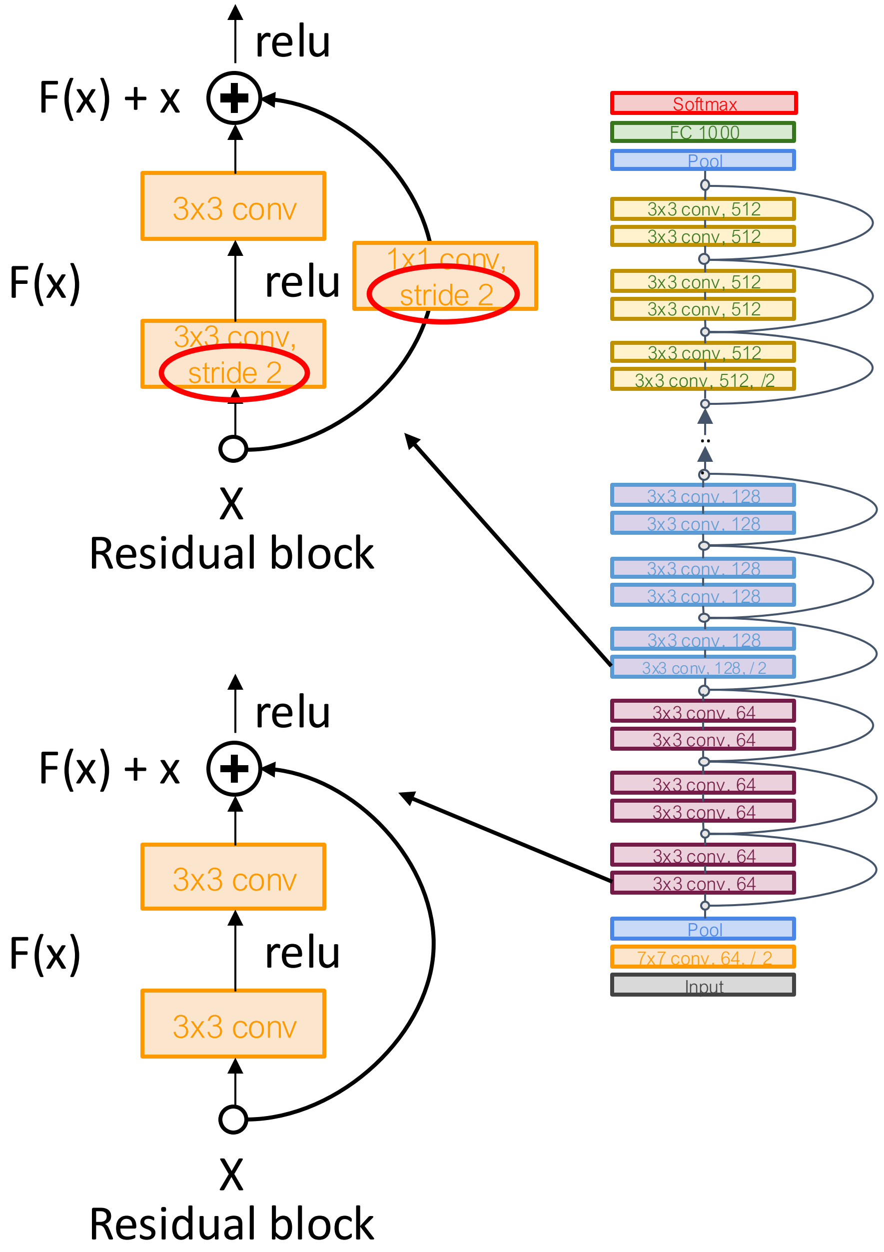
He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016

Residual Networks

A residual network is a stack of many residual blocks

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

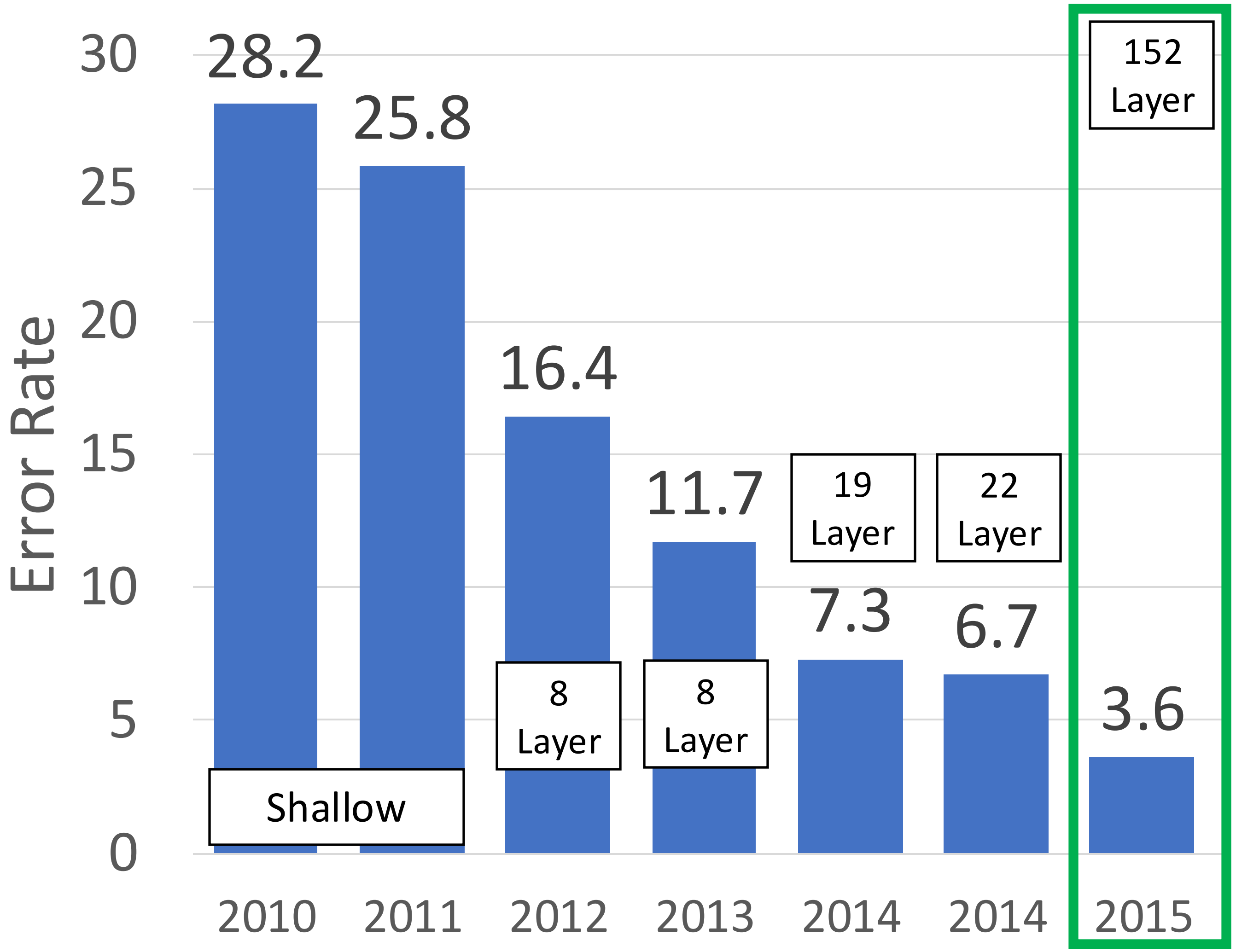
Network is divided into **stages**: the first block of each stage halves the resolution (with stride-2 conv) and doubles the number of channels



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

Adapted from: D Fouhey & J Johnson

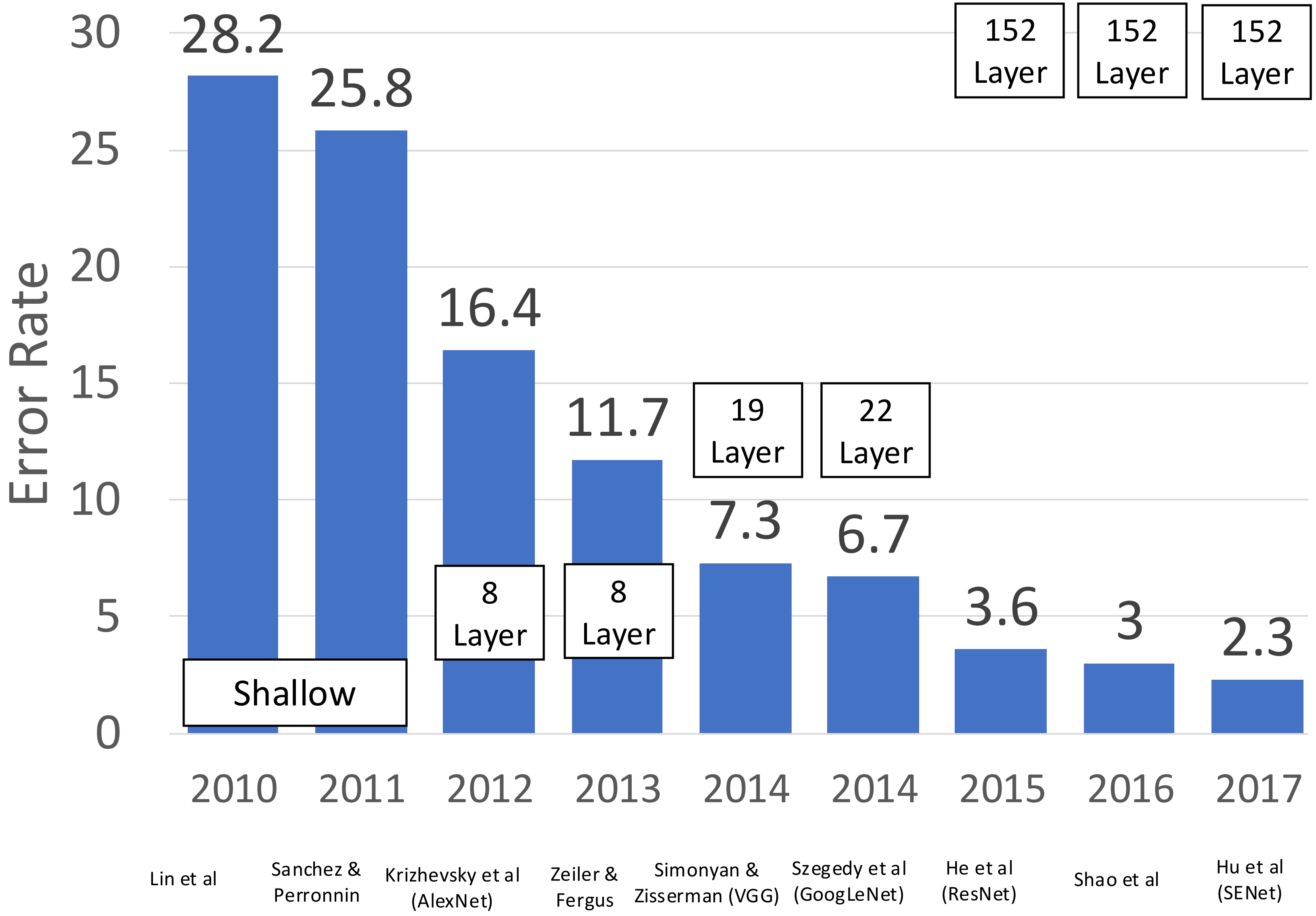
ImageNet Classification Challenge



Lin et al | Sanchez & Perronnin | Krizhevsky et al (AlexNet) | Zeiler & Fergus | Simonyan & Zisserman (VGG) | Szegedy et al (GoogLeNet) | He et al (ResNet)

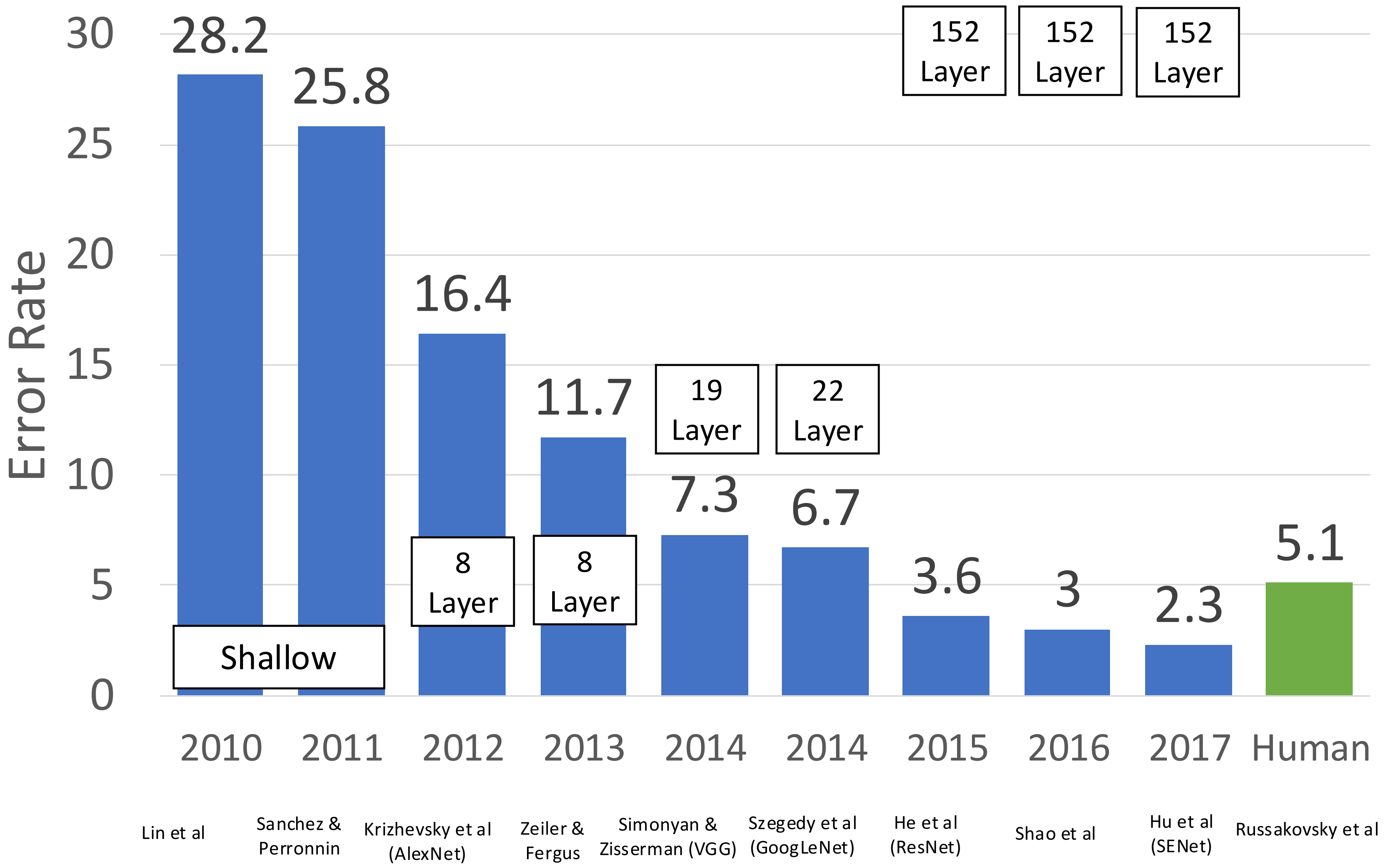
Slide credit: D Fouhey & J Johnson

ImageNet Classification Challenge



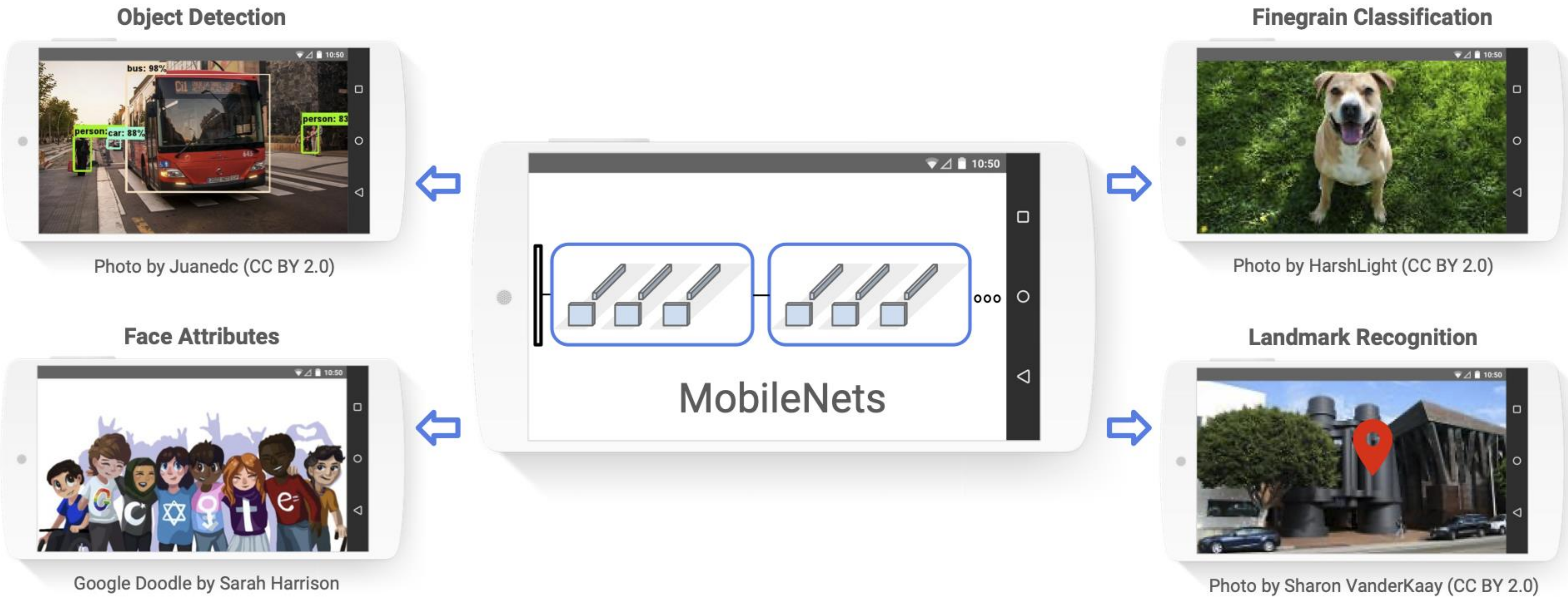
Slide credit: D Fouhey & J Johnson

ImageNet Classification Challenge



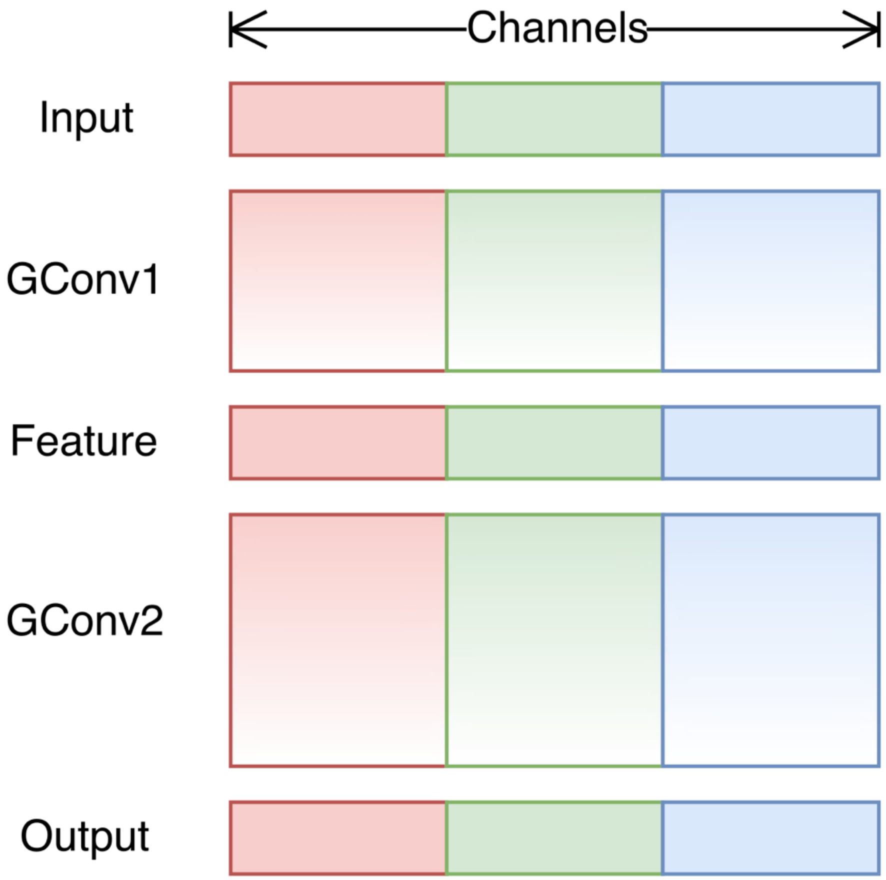
Slide credit: D Fouhey & J Johnson

Tiny Networks for Mobile Devices



[Howard et al., MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv 2017]

Group-based Convolution



Input: C_{in} x H x W

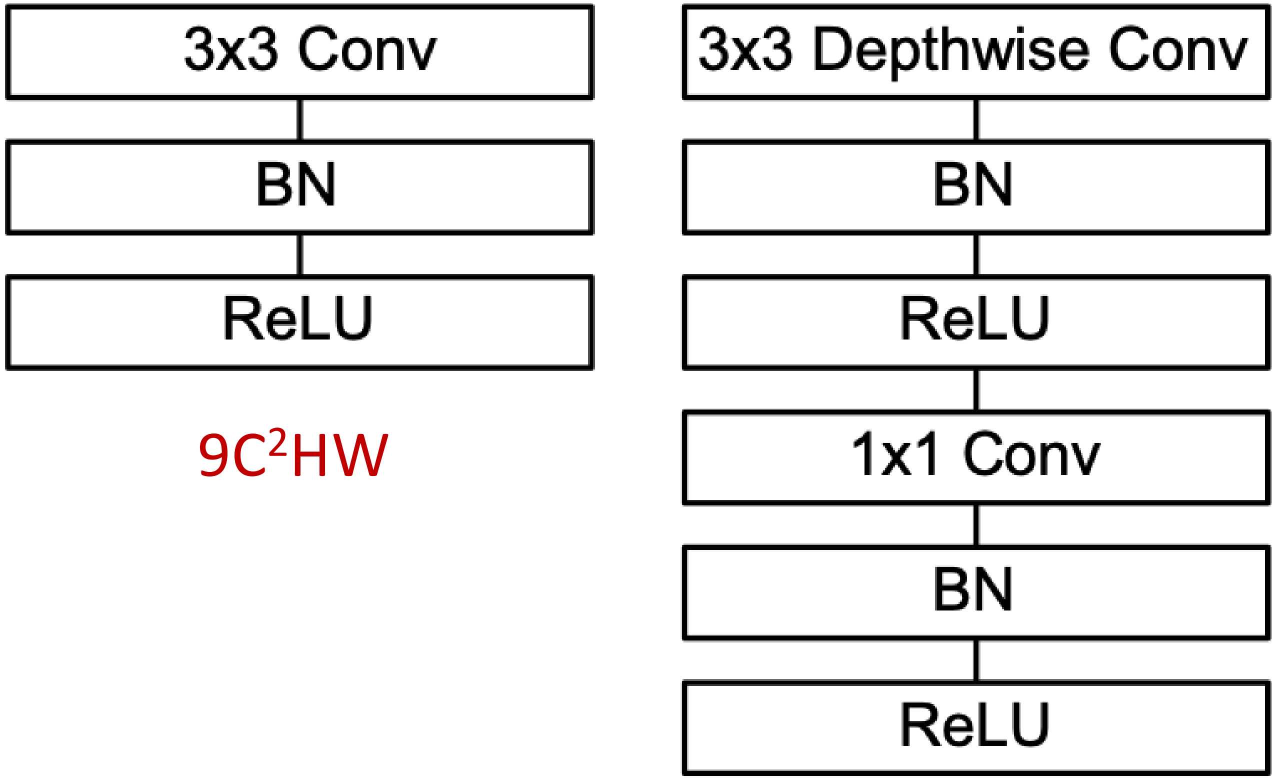
Hyperparameters:
- **Kernel size**: K_H x K_W
- **Number filters**: C_{out}
- **Padding**: P
- **Stride**: S
- **Groups:** G

Weight matrix: C_{out} /G x C_{in} /G x K_H x K_W x G

Bias vector: C_{out} /G

FLOPS: C_{out} /G x C_{in} /G x K_H x K_W x G x H x W

MobileNet



Channel: C, Groups: C

9CHW

$9C^2HW$

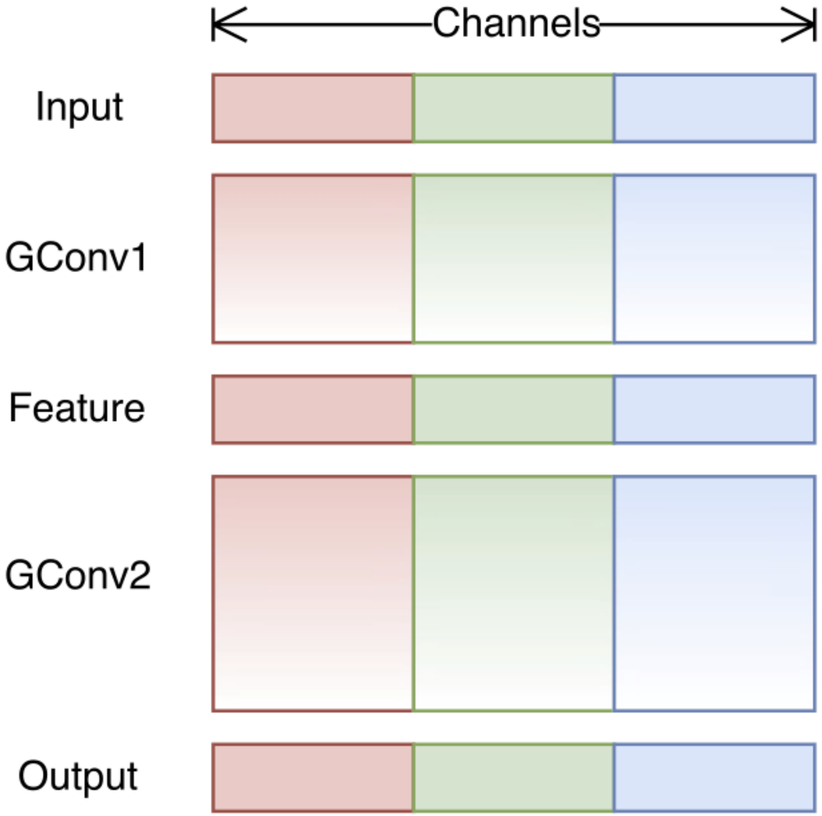
Pointwise Conv

C^2HW

Computation reduction: $9C^2HW/(9CHW + C^2HW) = 9C/(9+C)$

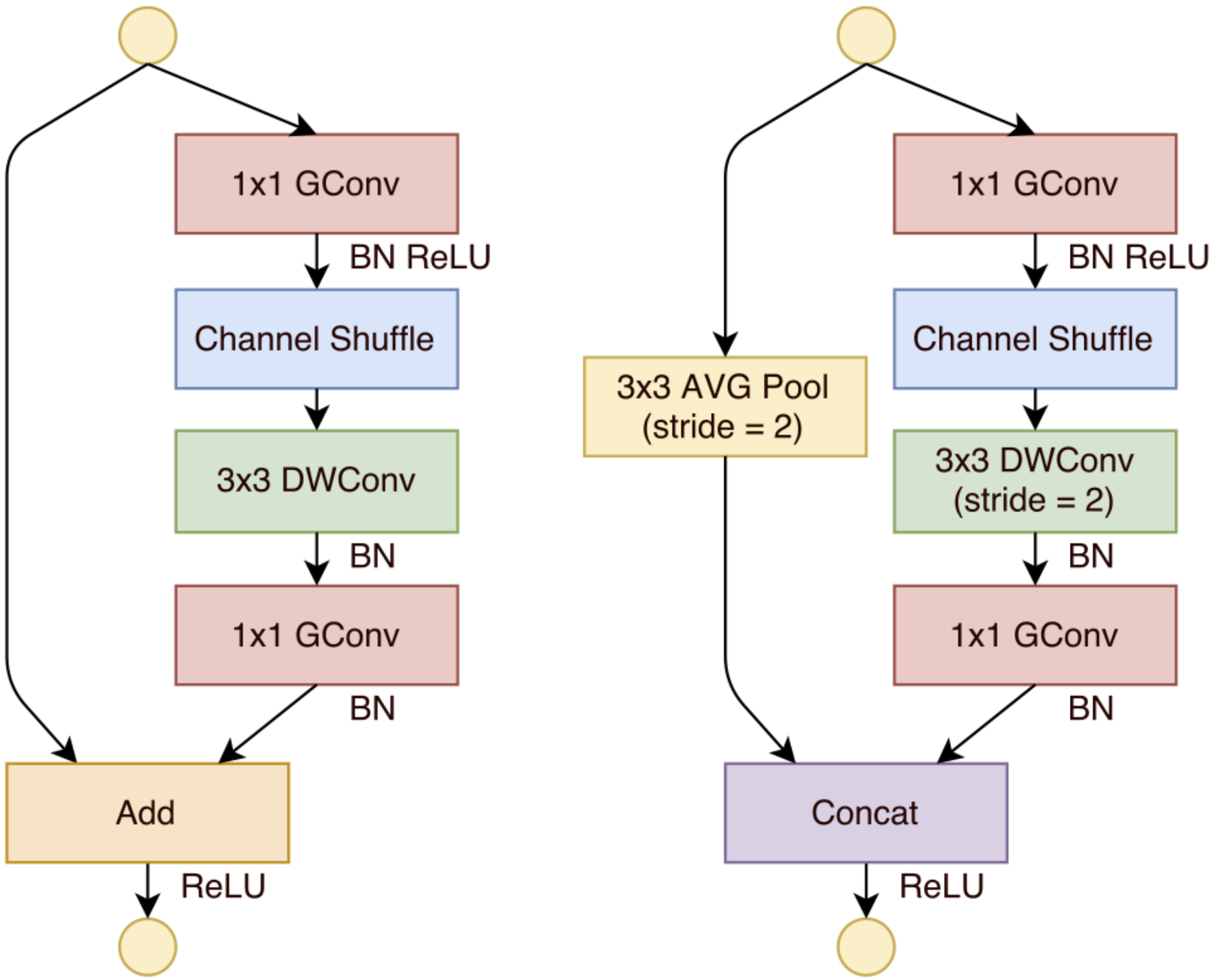
[Howard et al., MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv 2017]

ShuffleNet



[Zhang et al., ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. CVPR 2018]

ShuffleNet Units



[Zhang et al., ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. CVPR 2018]

Training Convolutional Networks

1. Download big datasets
2. Design CNN architecture
3. Initialize Weights
4. For t = 1 to T:
    1. Form minibatch
    2. Compute loss + gradient
    3. Update Weights
5. Apply trained model to task

Slide credit: D Fouhey & J Johnson

Training Convolutional Networks

1. Download big datasets
2. Design CNN architecture
3. Initialize Weights
4. For t = 1 to T:
    1. Form minibatch
    2. Compute loss + gradient
    3. Update Weights
5. Apply trained model to task

Slide credit: D Fouhey & J Johnson

Training Convolutional Networks

1. Download big datasets
2. Design CNN architecture
3. Initialize Weights
4. For t = 1 to T:
    1. Form minibatch
    2. Compute loss + gradient
    3. Update Weights
5. Apply trained model to task

If the model is big, won't we overfit?

Slide credit: D Fouhey & J Johnson

Regularizing CNNs: Weight Decay

$$L_{reg} = \frac{1}{2}\sum_{\ell} \|W_\ell\|^2 \qquad \frac{\partial L_{reg}}{\partial W_\ell} = W_\ell$$

Add L2 regularization term L_{reg} to the loss penalizing large weight matrices

Usually don't regularize bias terms, or BatchNorm scale / shift params

*Technical note: Adding an explicit term to the loss is "L2 Regularization"; "Weight decay" adds a term to the gradient. Theyare equivalent for SGD, but not quite the same for other optimizers like Adam

Slide credit: D Fouhey & J Johnson

Regularizing CNNs: Data Augmentation

Hippo

Hippo?

Horizontal Flip

Hippo?

Color Jitter

Hippo?

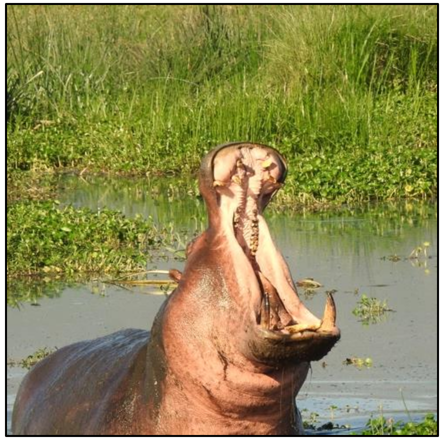
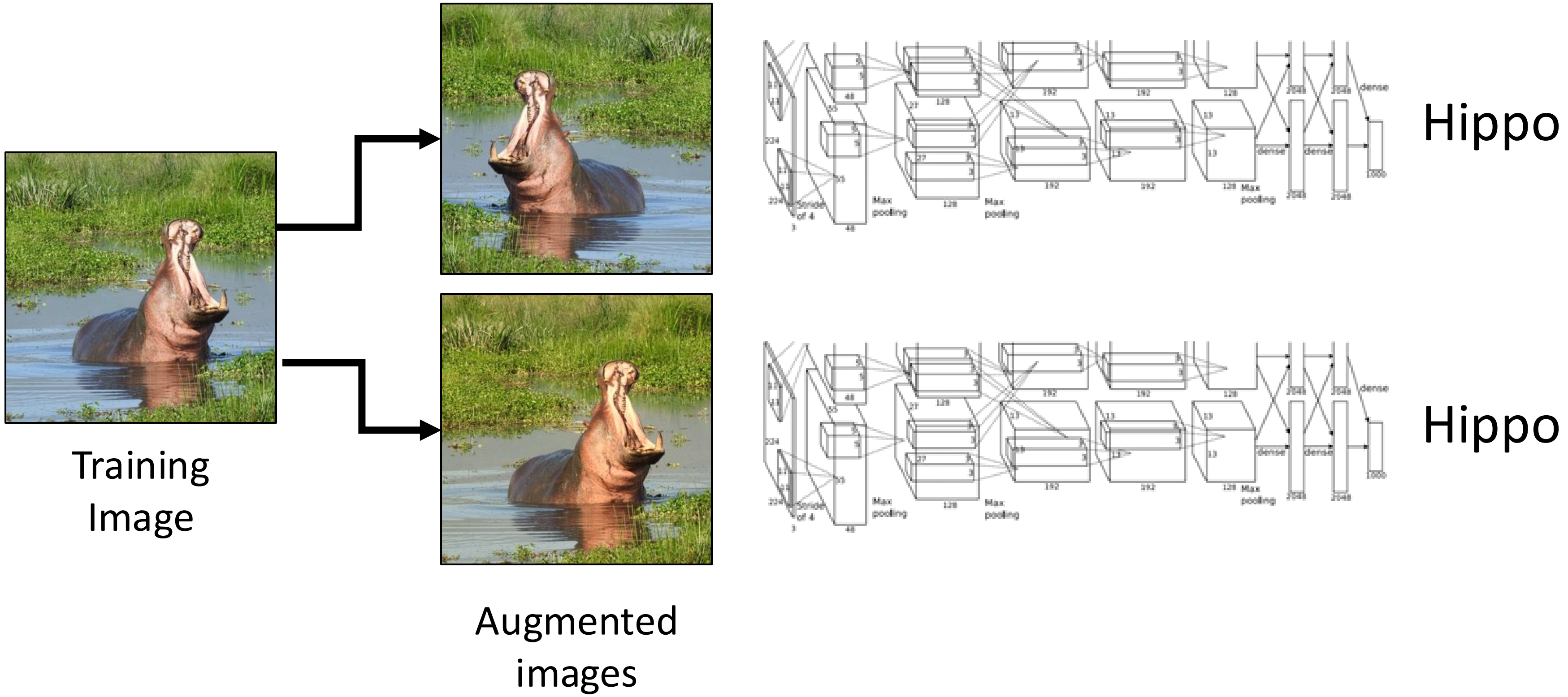


Image Cropping

Slide credit: D Fouhey & J Johnson

Regularizing CNNs: Data Augmentation

Apply random transformations to input images during training
Artificially “inflate” the size of your dataset



Training Convolutional Networks

1. Download big datasets
2. Design CNN architecture
3. Initialize Weights
4. For t = 1 to T:
    1. Form minibatch
    2. Compute loss + gradient
    3. Update Weights
5. Apply trained model to task

If the model is big, won't we overfit?

Slide credit: D Fouhey & J Johnson

Training Convolutional Networks

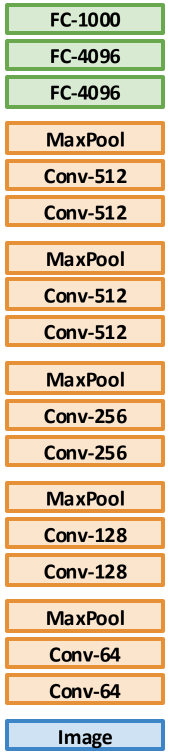
1. Download big datasets
2. Design CNN architecture
3. Initialize Weights
4. For t = 1 to T:
    1. Form minibatch
    2. Compute loss + gradient
    3. Update Weights
5. Apply trained model to task

What if we can't find one?

Slide credit: D Fouhey & J Johnson

Transfer Learning: Feature Extraction

1. Train on ImageNet

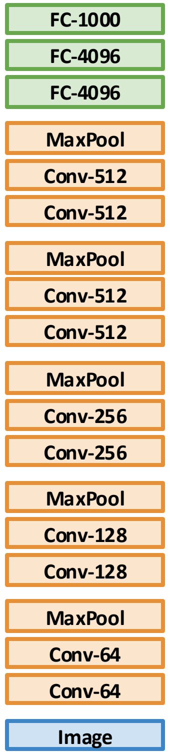


Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014

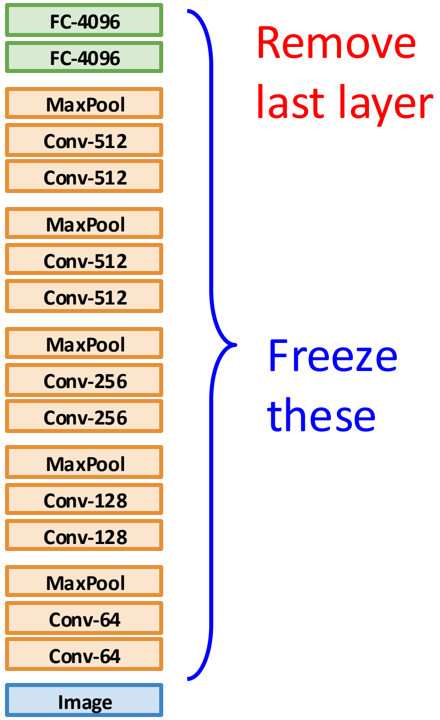
Slide credit: D Fouhey & J Johnson

Transfer Learning: Feature Extraction

1. Train on ImageNet



2. CNN as feature extractor



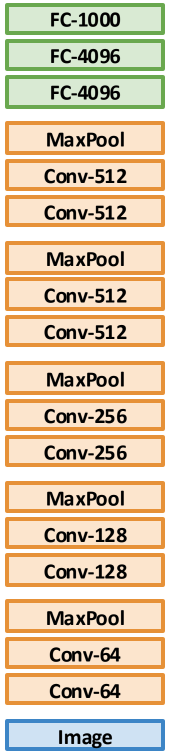
Use your small dataset to train a **linear classifier** on top of pretrained CNN features

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014

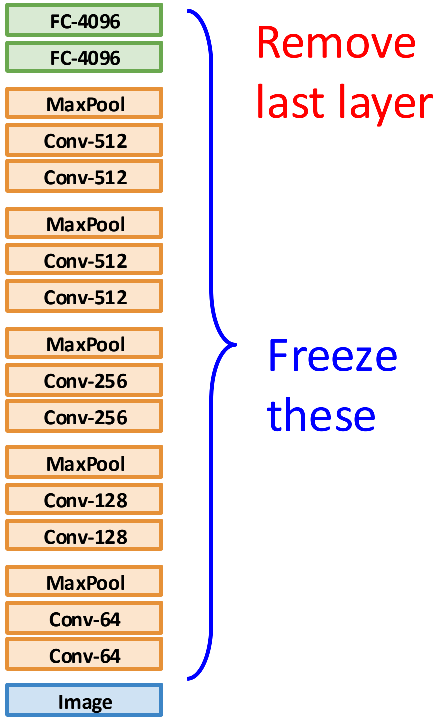
Slide credit: D Fouhey & J Johnson

Transfer Learning: Fine-Tuning

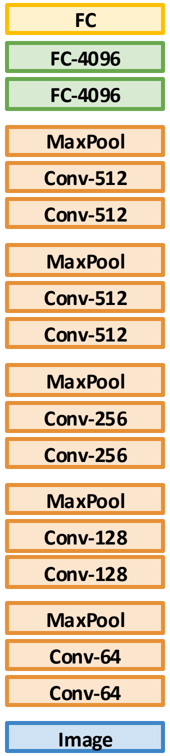
1. Train on ImageNet



2. CNN as feature extractor



3. Bigger dataset: **Fine-Tuning**

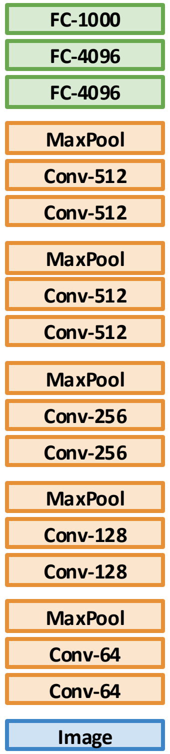


Reinitialize last layer and continue training whole network on your dataset

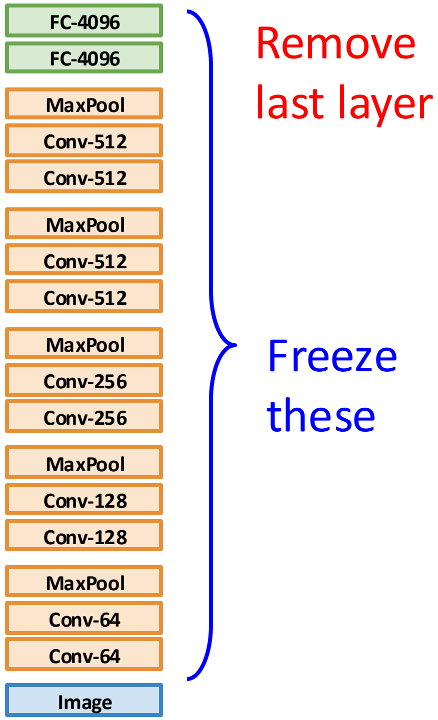
Slide credit: D Fouhey & J Johnson

Transfer Learning: Fine-Tuning

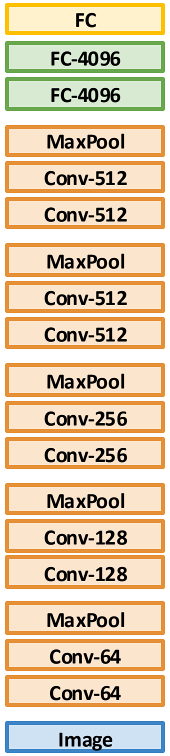
1. Train on ImageNet



2. CNN as feature extractor



3. Bigger dataset: **Fine-Tuning**



Reinitialize last layer and continue training whole network on your dataset

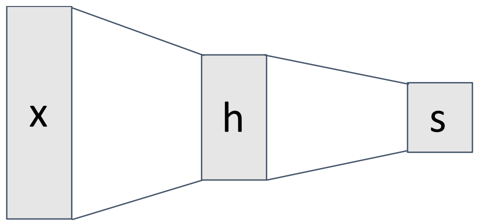
Some tricks:

- Train with feature extraction first before fine-tuning
- Lower the learning rate: use ~1/10 of LR used in original training
- Sometimes freeze lower layers to save computation

Slide credit: D Fouhey & J Johnson

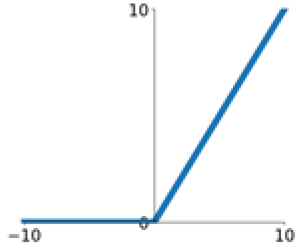
Recap: Convolutional Networks

Fully-Connected Layers



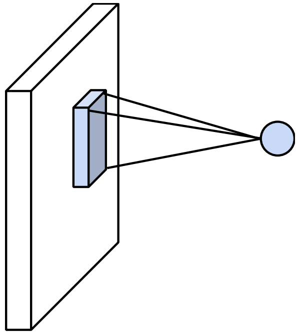
$$y = Wx + b$$

Activation Function

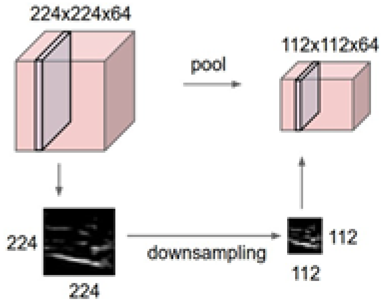


$$y = \max(0, x)$$

Convolution Layers



Pooling Layers

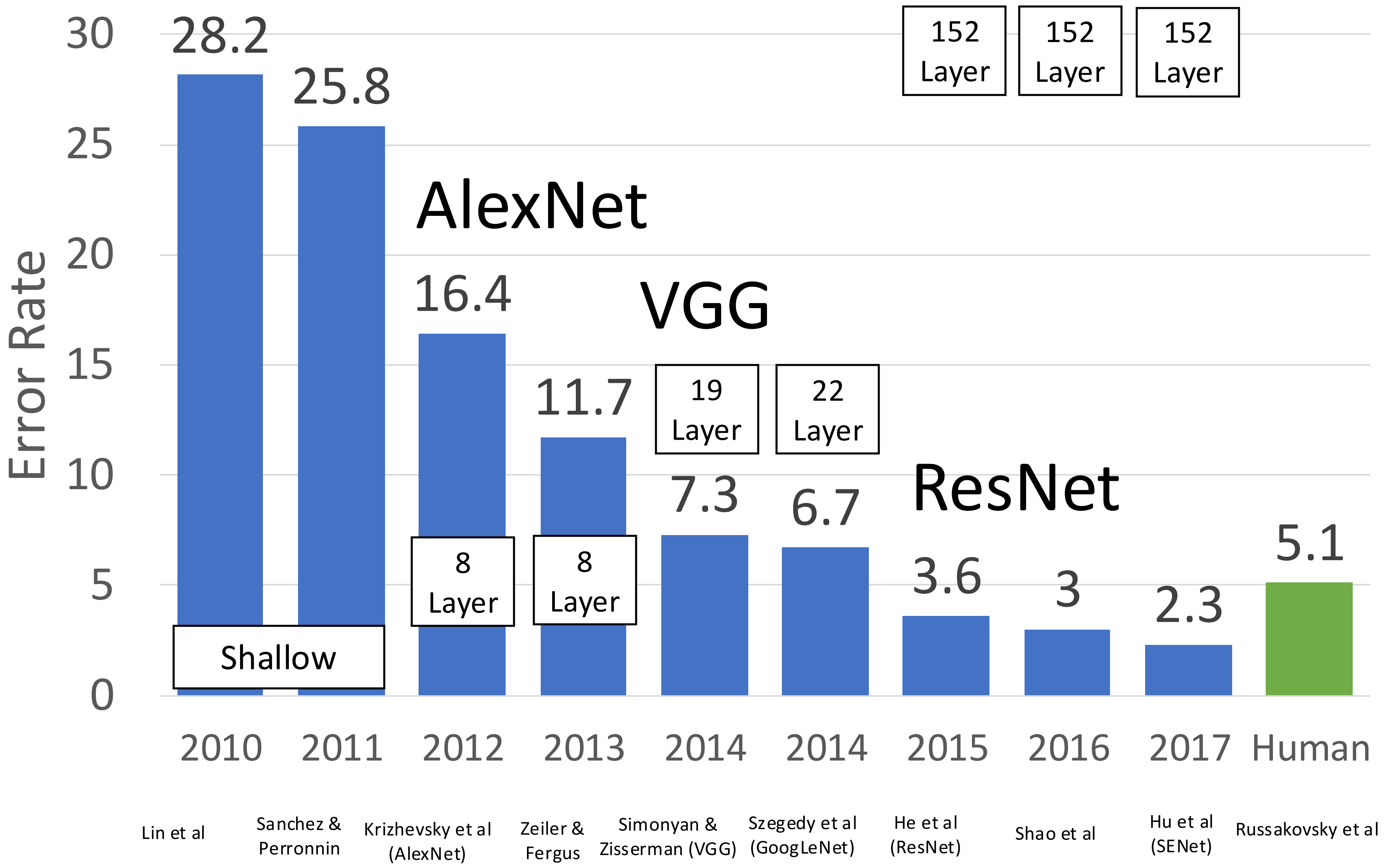


Normalization

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

Slide credit: D Fouhey & J Johnson

Recap: CNN Architectures



Slide credit: D Fouhey & J Johnson

Recap: Training CNNs

1. Download big datasets — Transfer Learning
2. Design CNN architecture
3. Initialize Weights — Xavier / MSRA Init
4. For t = 1 to T:
   1. Form minibatch
   2. Compute loss + gradient
   3. Update Weights
5. Apply trained model to task

Regularization + Data Augmentation

Slide credit: D Fouhey & J Johnson

Next Class

More about
Convolutional Neural Networks