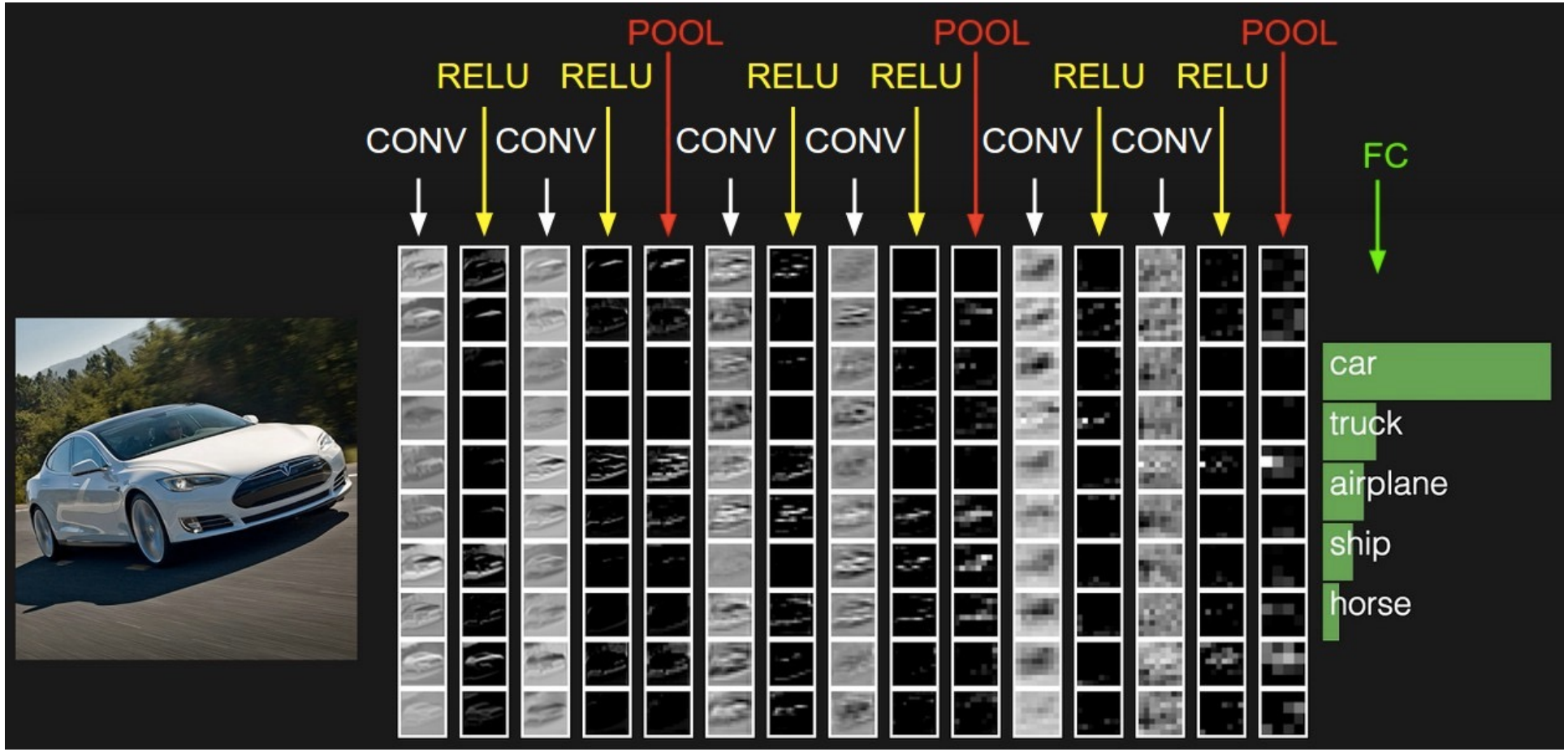
Convolutional neural networks

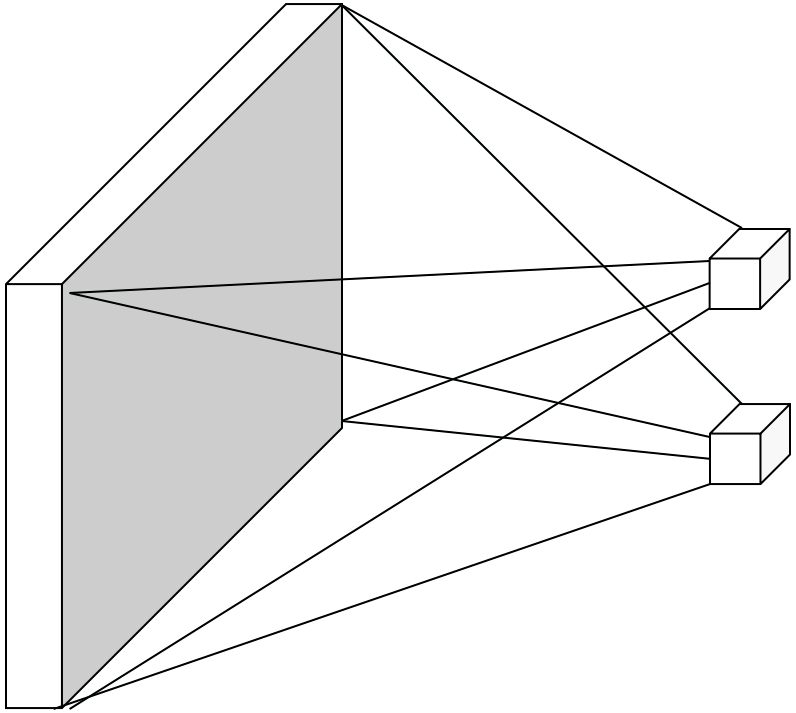


Many slides from Rob Fergus, Andrej Karpathy

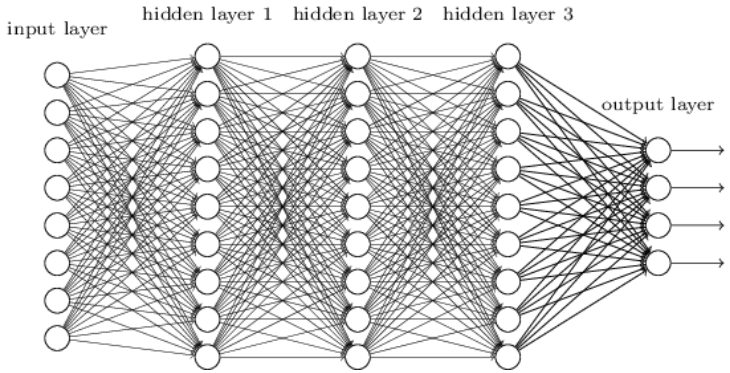
Outline

- Building blocks
  - Convolutional layers
  - Pooling layers and nonlinearities
- Architectures:
  - 1st generation (2012-2013): AlexNet
  - 2nd generation (2014): VGGNet, GoogLeNet
  - 3rd generation (2015): ResNet
  - 4th generation (2016): Wide ResNet, ResNeXt, DenseNet

Let’s design a neural network for images



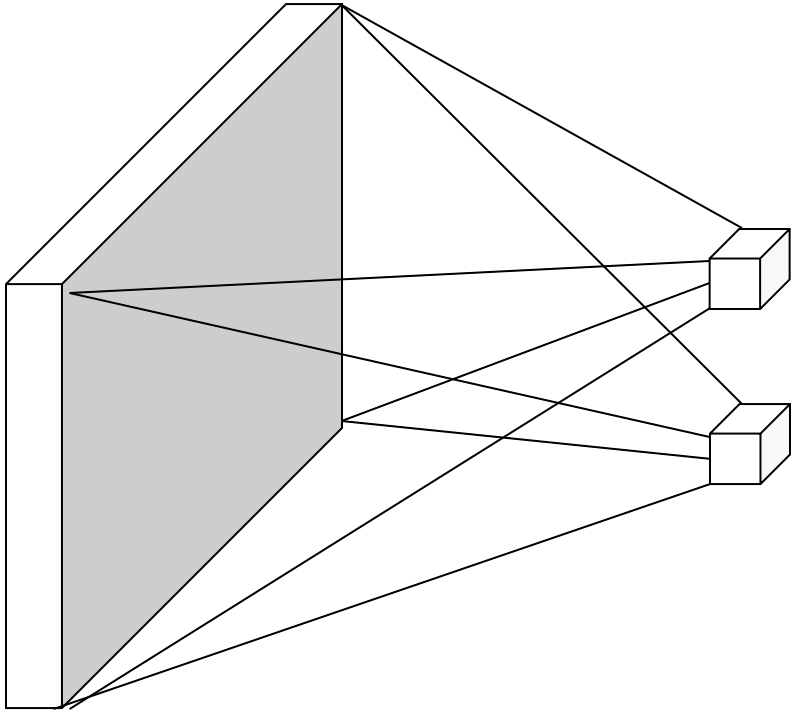
image



- This kind of design is known as *multi-layer perceptron* (MLP)

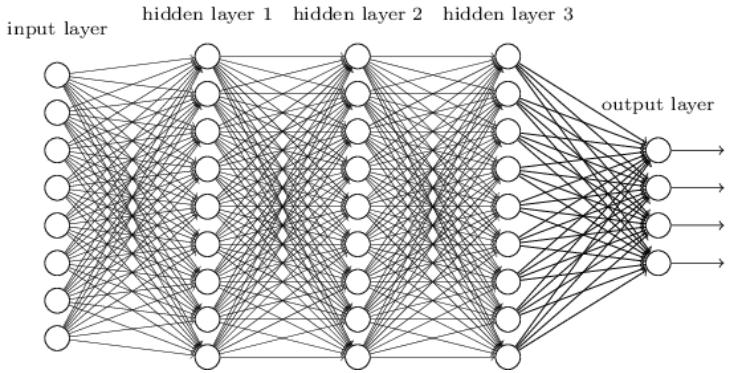
Fully connected layer

Let's design a neural network for images



image

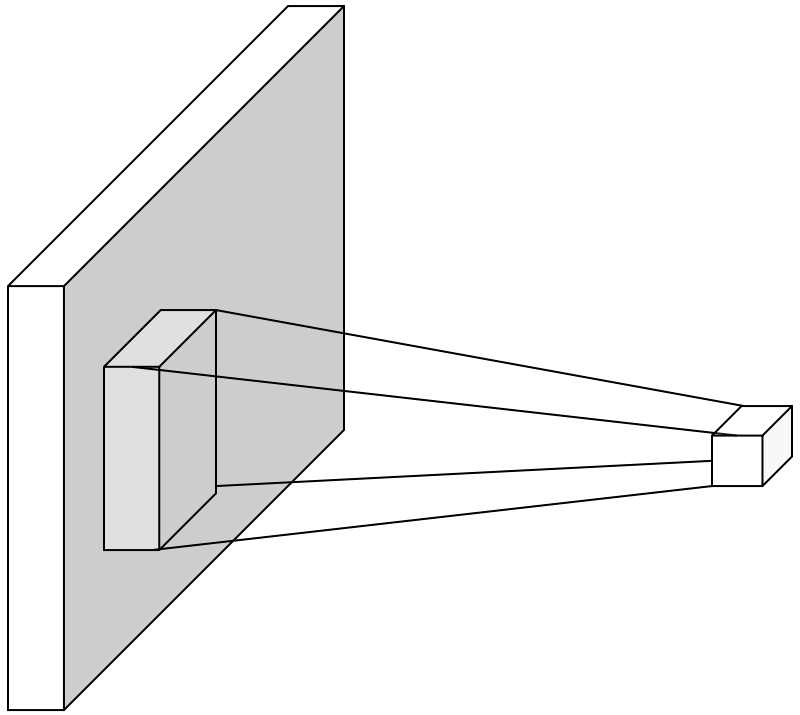
Fully connected layer



- This kind of design is known as *multi-layer perceptron* (MLP)
- What is wrong with this?

Recall: MLP as bank of whole-image templates

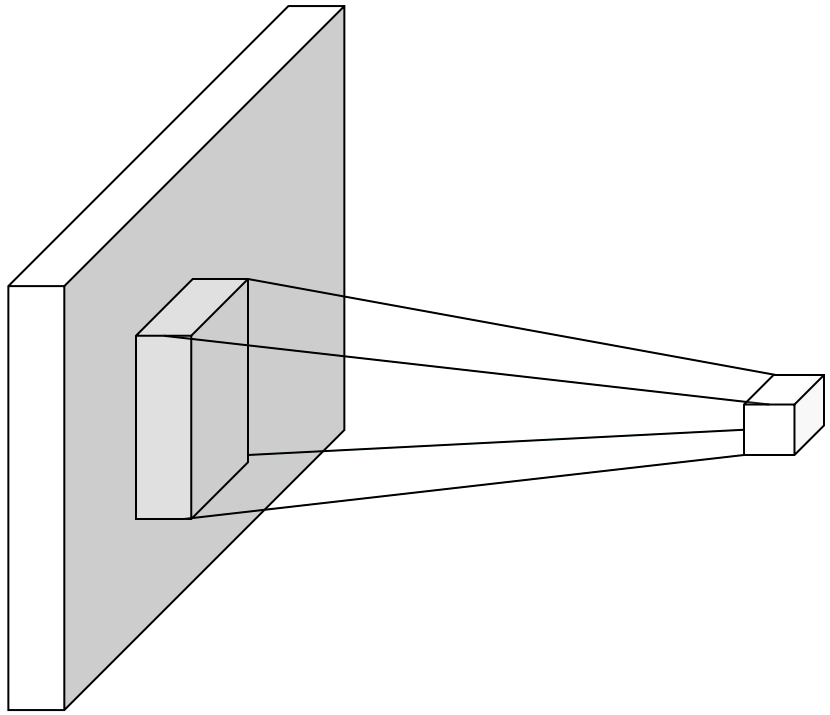
Convolutional architecture



image

- Let's limit the *receptive fields* of units, tile them over the input image, and share their weights

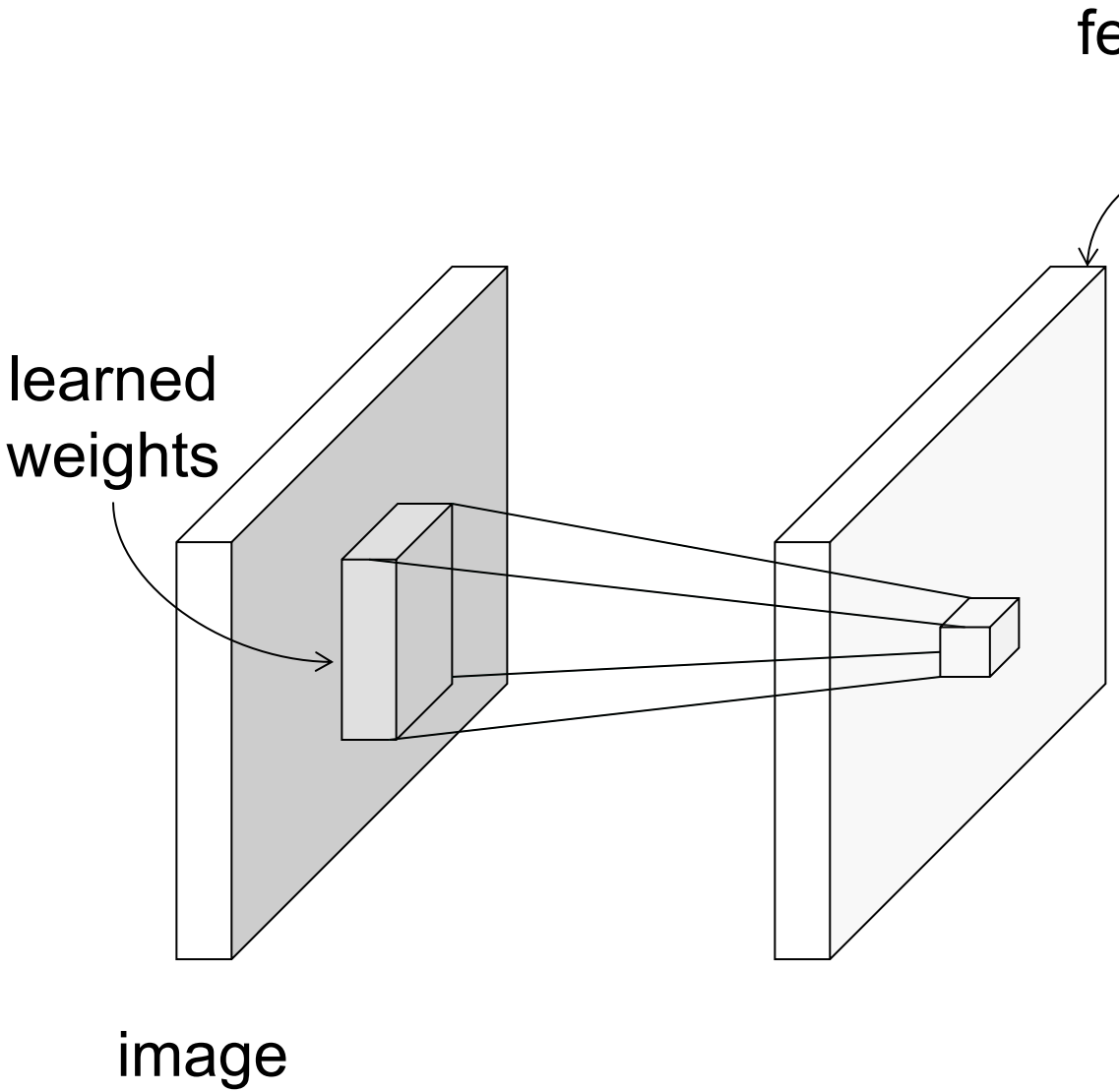
Convolutional architecture



image

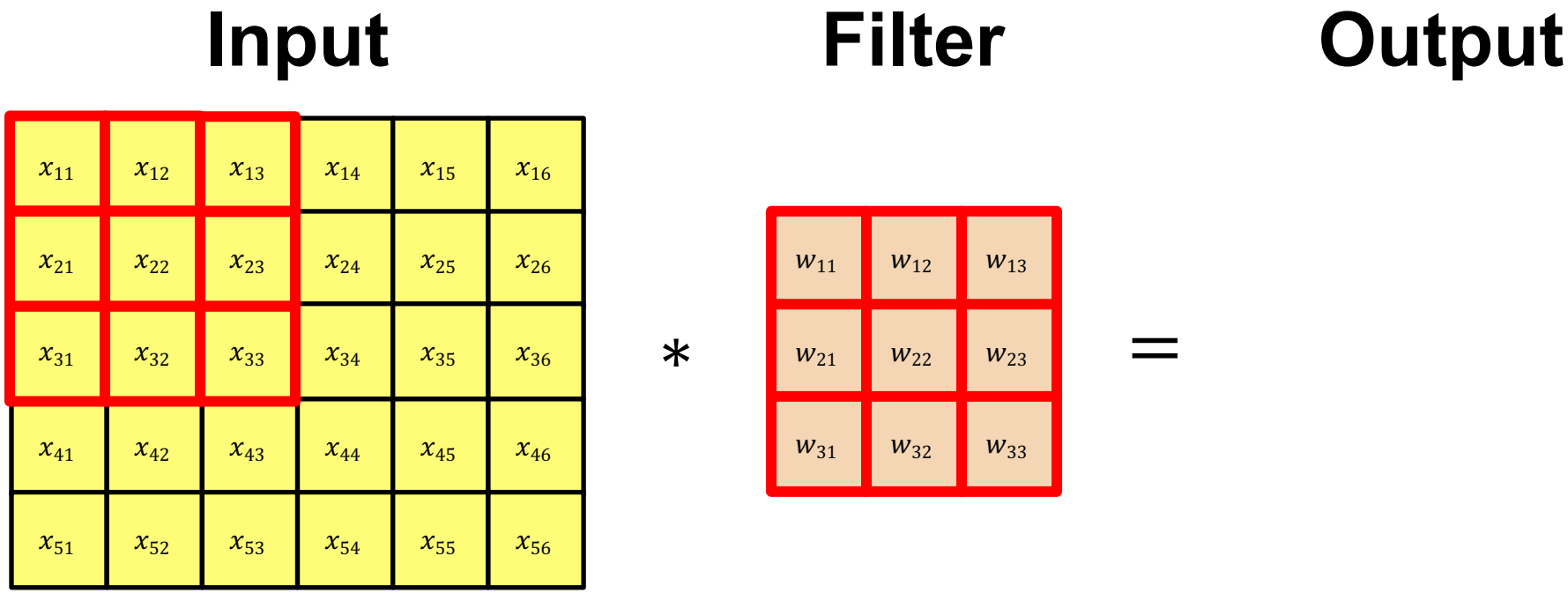
- Let's limit the *receptive fields* of units, tile them over the input image, and share their weights

Convolutional architecture



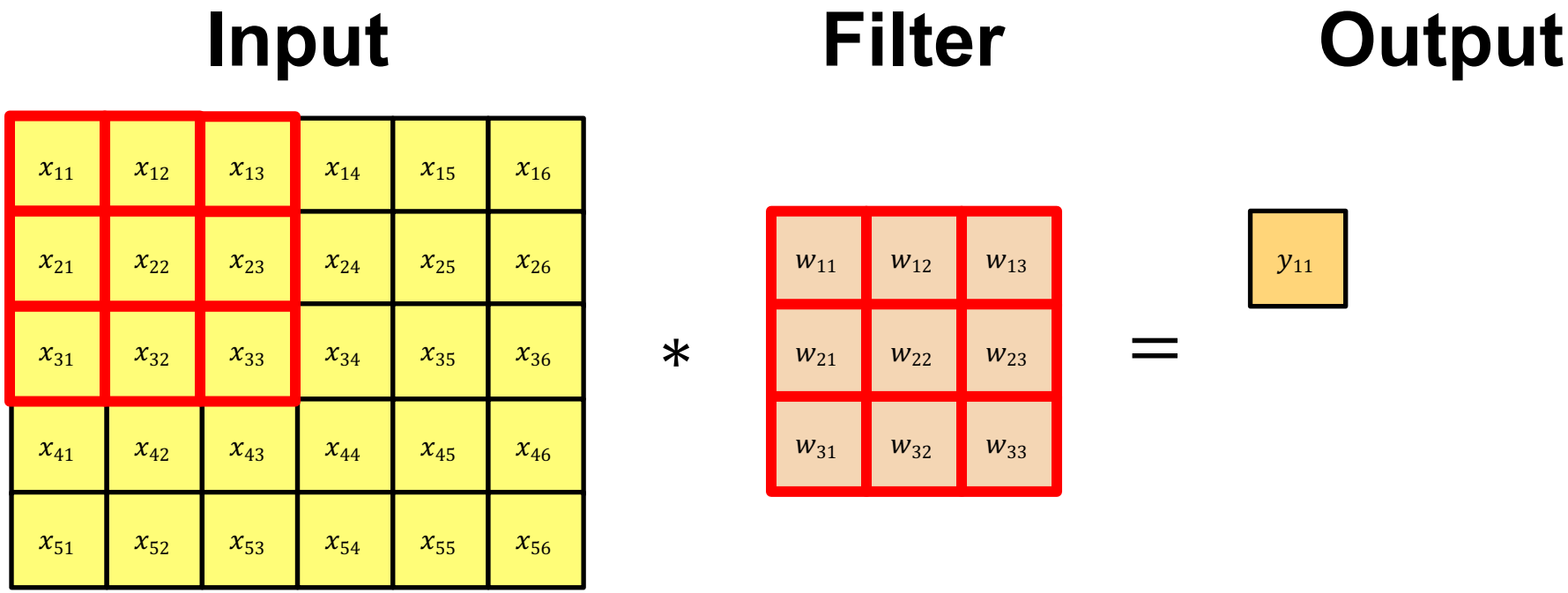
- Let's limit the *receptive fields* of units, tile them over the input image, and share their weights
- This is equivalent to sliding the learned filter over the image, computing dot products at every location

Convolution example



Adapted from D. Fouhey and J. Johnson

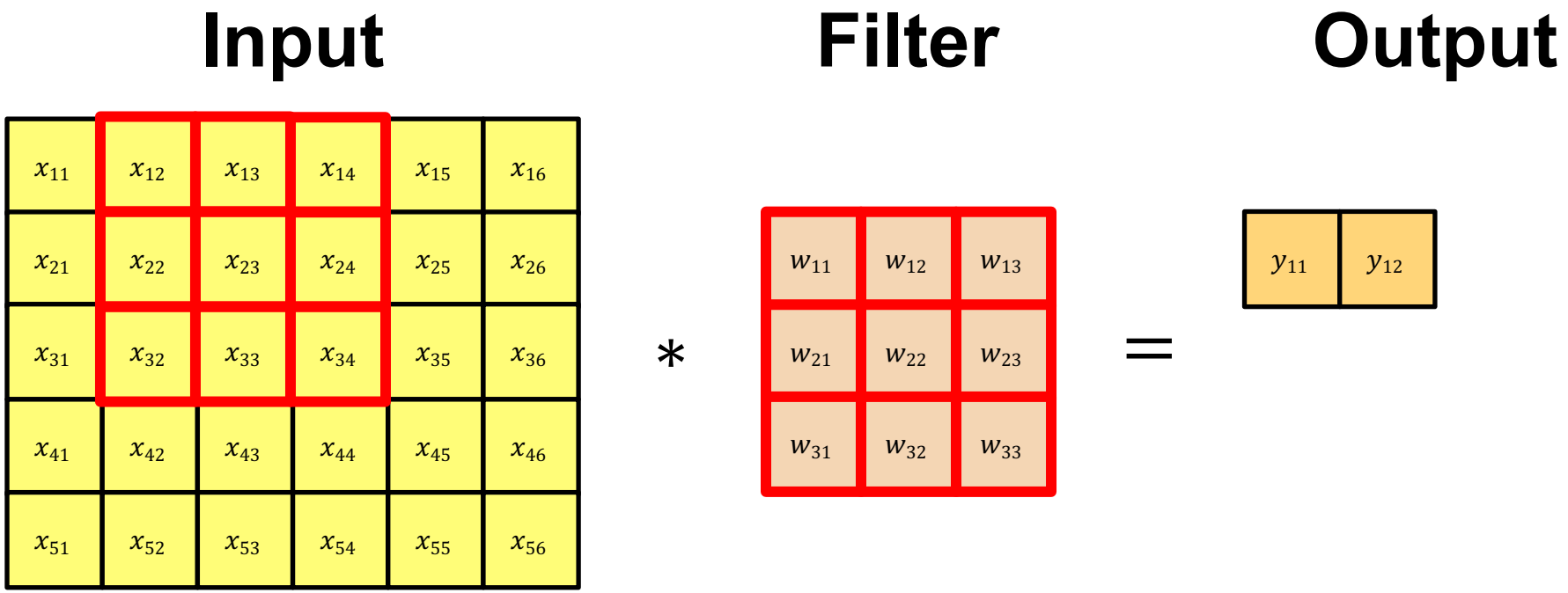
Convolution example



$$y_{11} = x_{11} \cdot w_{11} + x_{12} \cdot w_{12} + x_{13} \cdot w_{13} + \ \ldots + x_{33} \cdot w_{33}$$

Adapted from D. Fouhey and J. Johnson

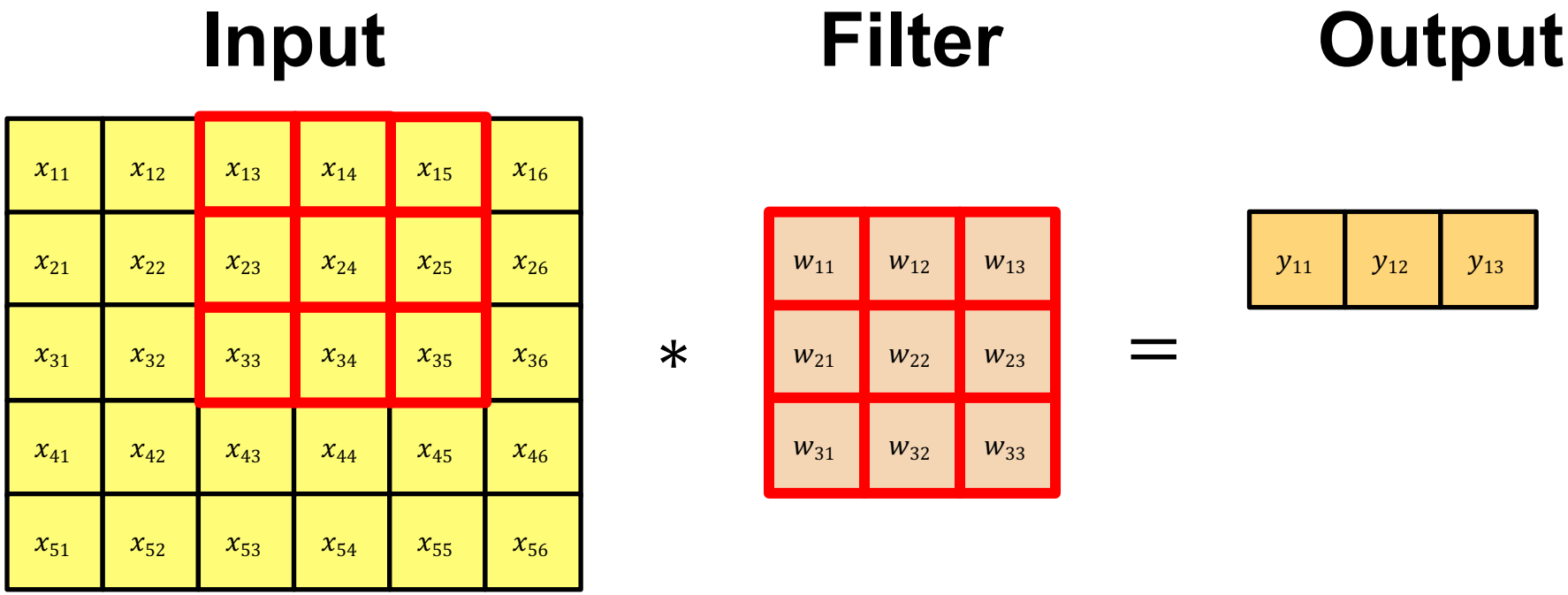
Convolution example



$$y_{12} = x_{12} \cdot w_{11} + x_{13} \cdot w_{12} + x_{14} \cdot w_{13} + \ \ldots + x_{34} \cdot w_{33}$$

Adapted from D. Fouhey and J. Johnson

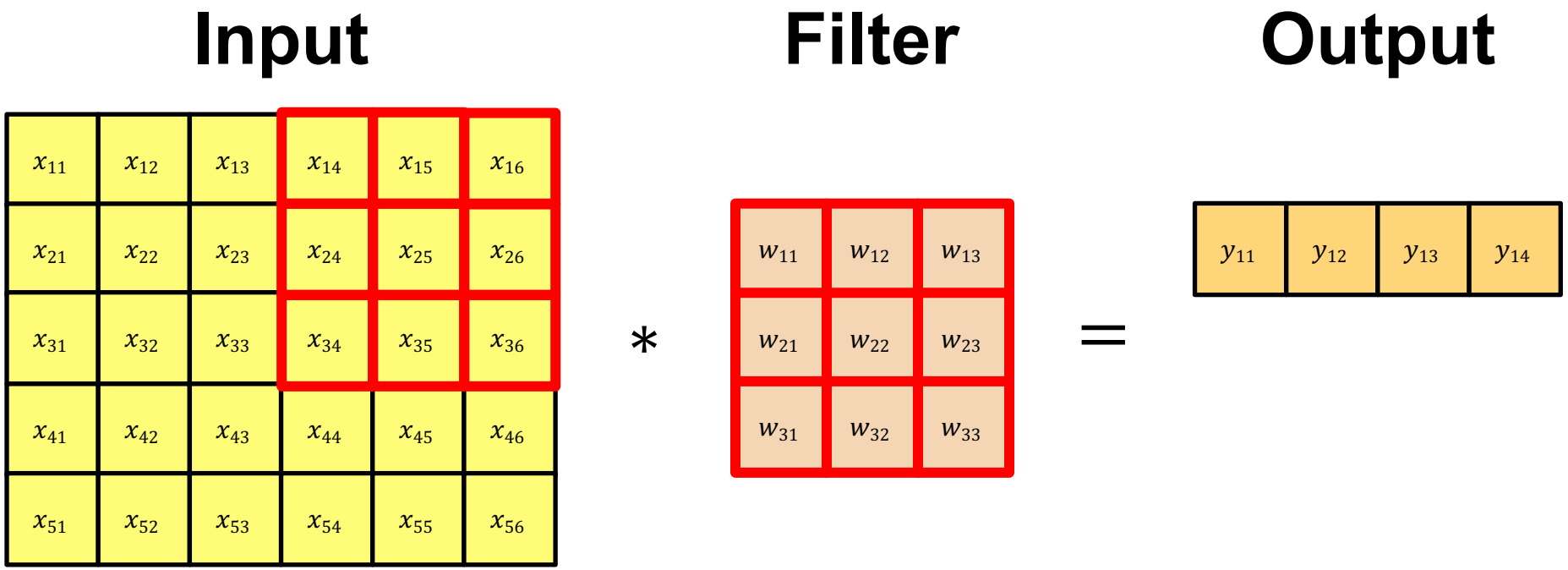
Convolution example



$$y_{13} = x_{13} \cdot w_{11} + x_{14} \cdot w_{12} + x_{15} \cdot w_{13} + \ldots + x_{35} \cdot w_{33}$$

Adapted from D. Fouhey and J. Johnson

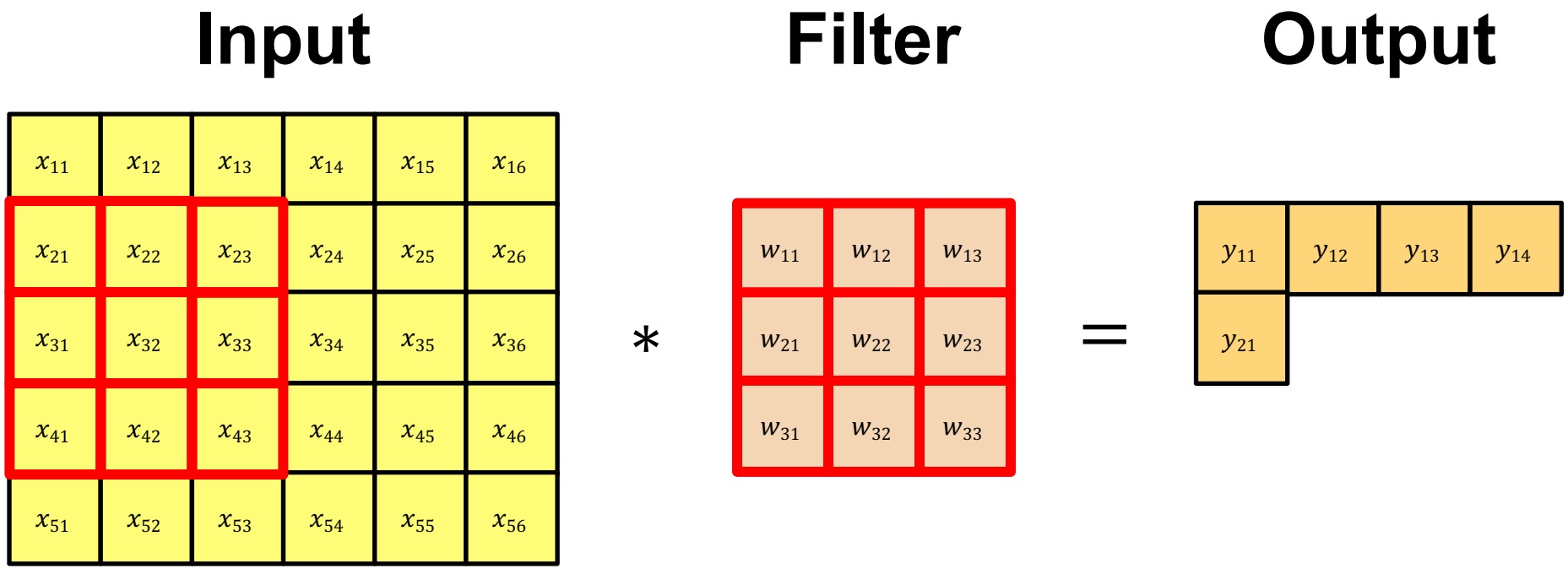
Convolution example



$$y_{14} = x_{14} \cdot w_{11} + x_{15} \cdot w_{12} + x_{16} \cdot w_{13} + \ \ldots + x_{36} \cdot w_{33}$$

Adapted from D. Fouhey and J. Johnson

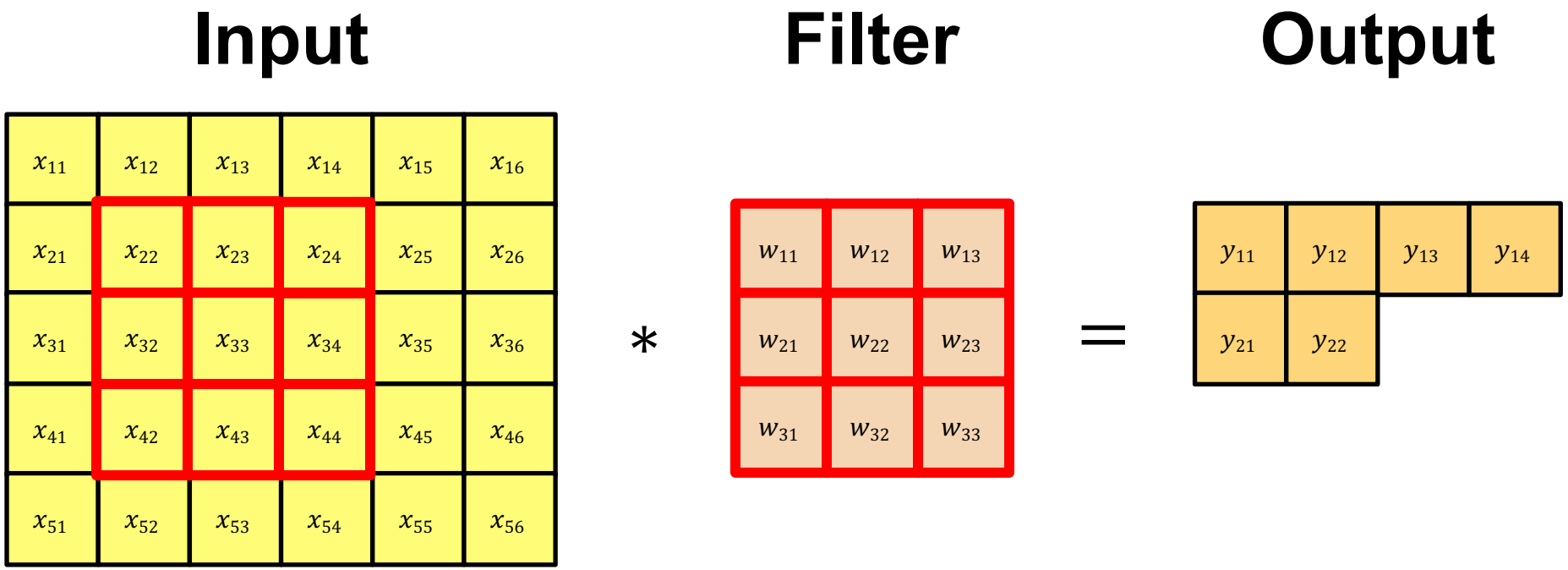
Convolution example



$$y_{21} = x_{21} \cdot w_{11} + x_{22} \cdot w_{12} + x_{23} \cdot w_{13} + \ \ldots + x_{43} \cdot w_{33}$$

Adapted from D. Fouhey and J. Johnson

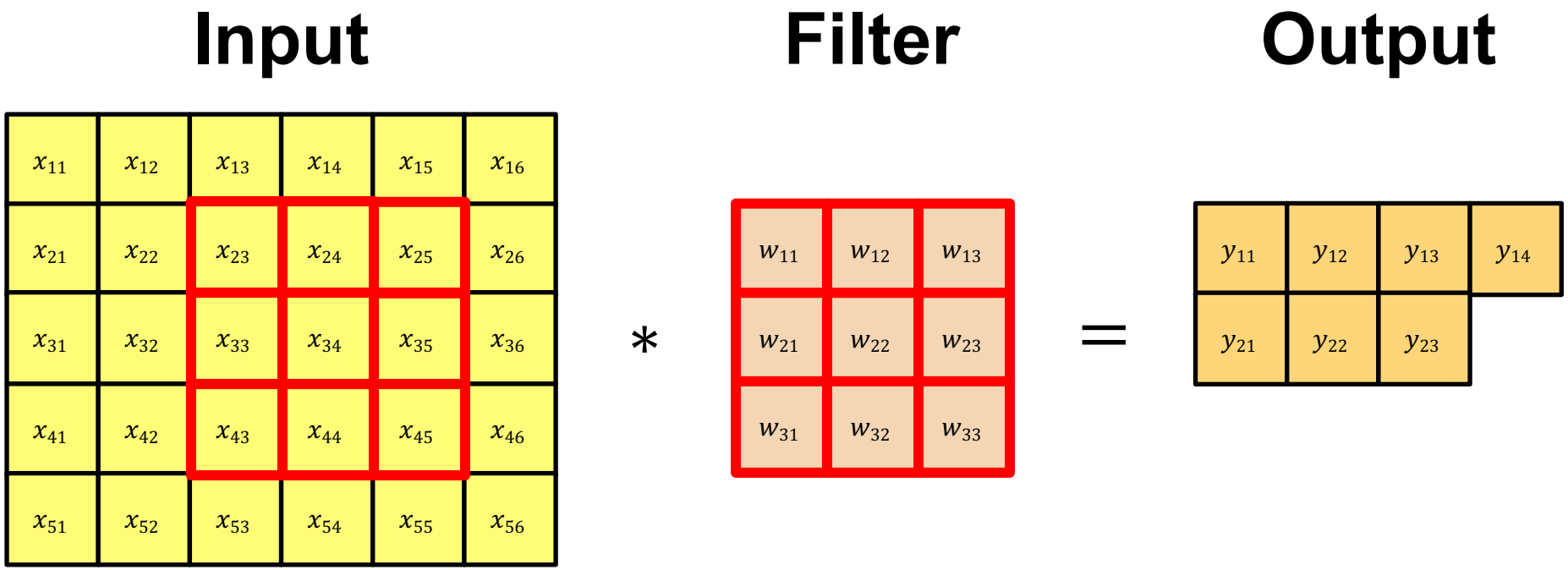
Convolution example



$$y_{22} = x_{22} \cdot w_{11} + x_{23} \cdot w_{12} + x_{24} \cdot w_{13} + \ldots + x_{44} \cdot w_{33}$$

Adapted from D. Fouhey and J. Johnson

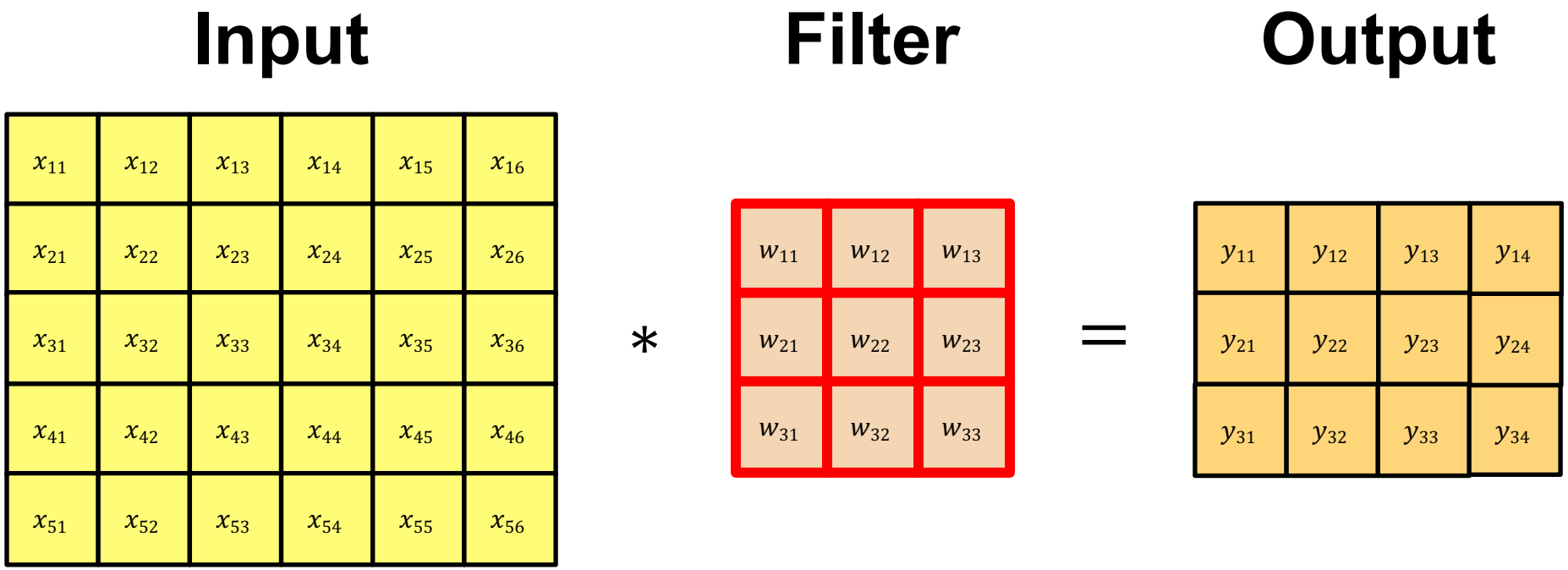
Convolution example



$$y_{23} = x_{23} \cdot w_{11} + x_{24} \cdot w_{12} + x_{25} \cdot w_{13} + \ldots + x_{45} \cdot w_{33}$$

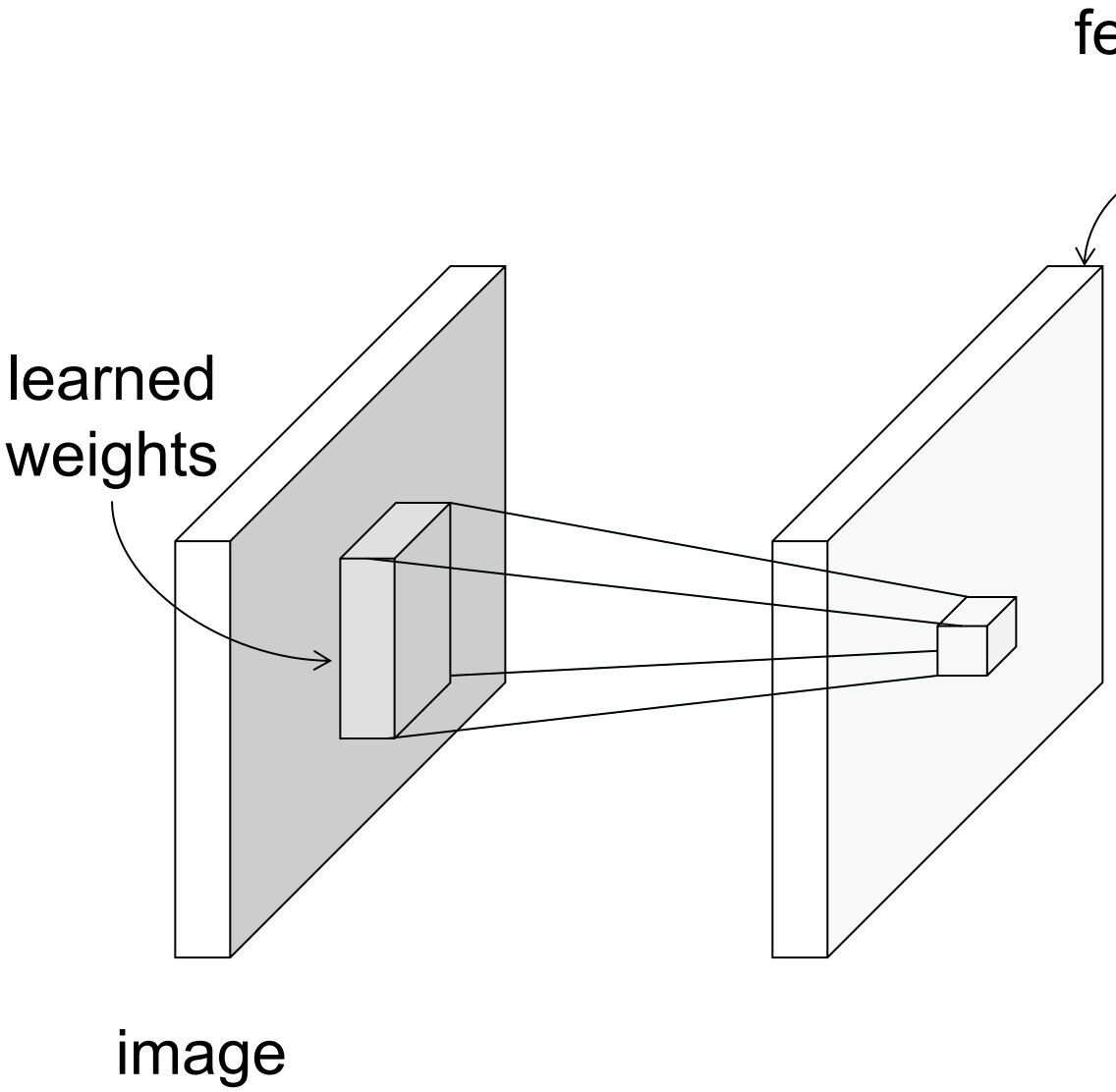
Adapted from D. Fouhey and J. Johnson

Convolution example

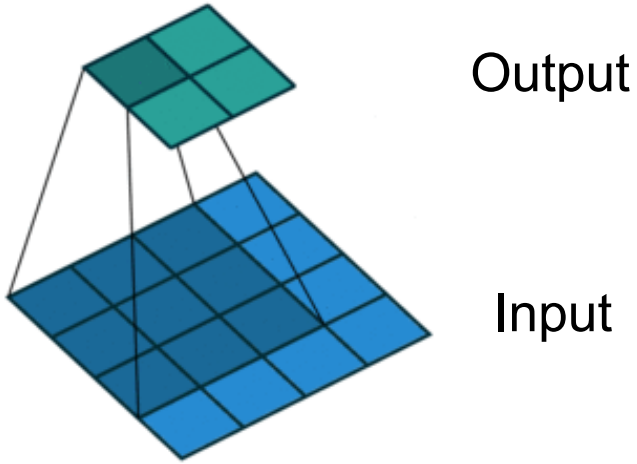


Adapted from D. Fouhey and J. Johnson

Convolutional architecture



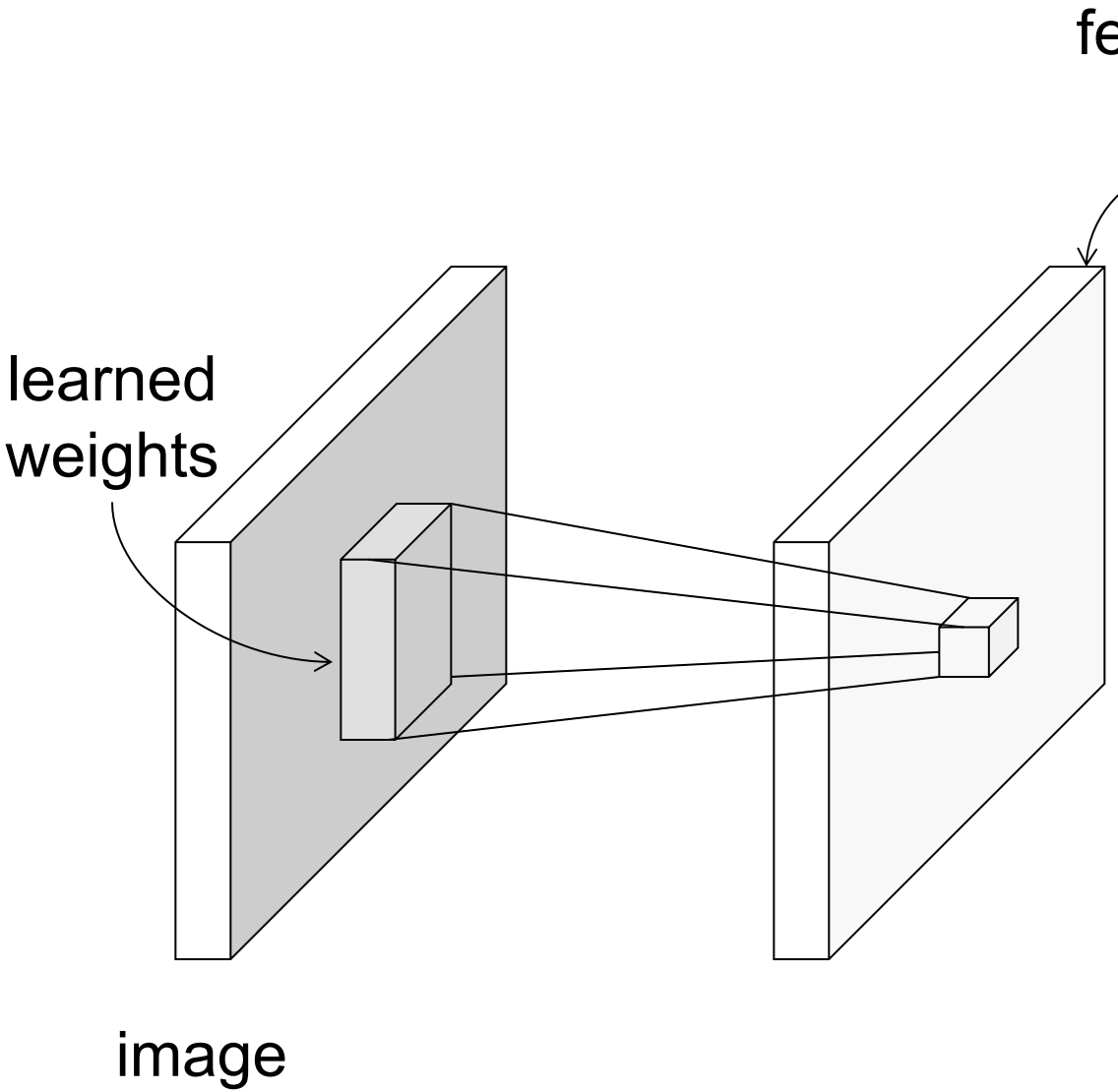
Output feature map resolution depends on *padding* and *stride*



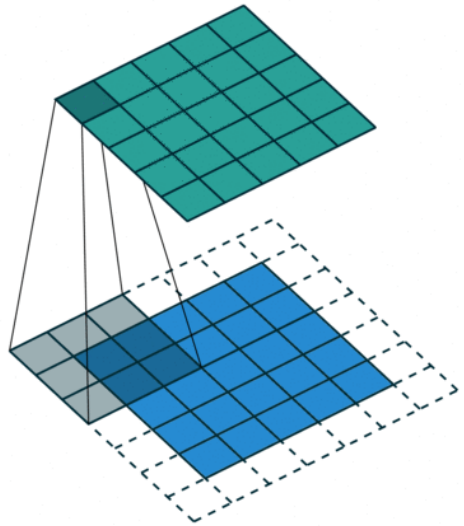
No padding, stride 1

Animation source

Convolutional architecture



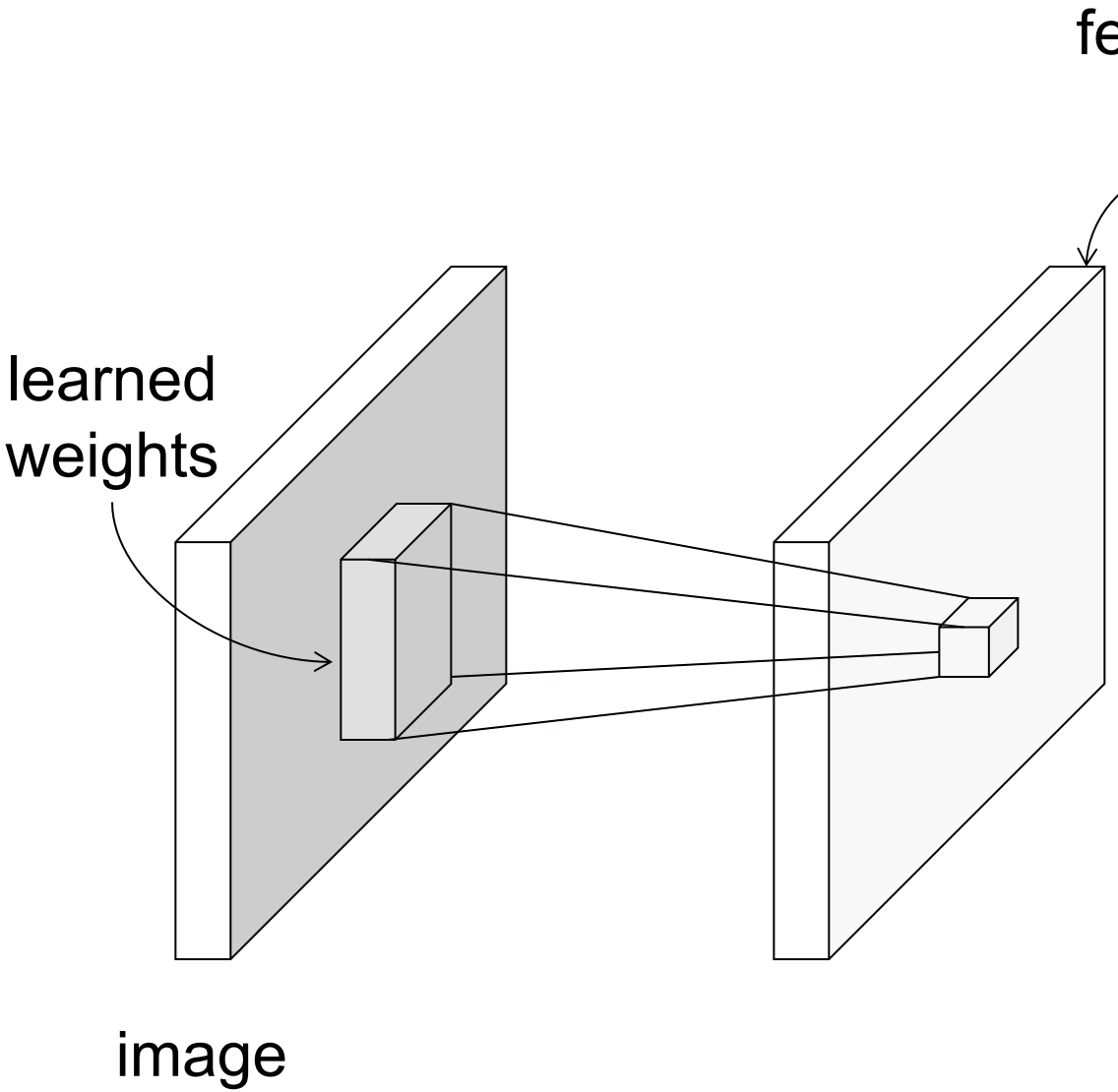
Output feature map resolution depends on *padding* and *stride*



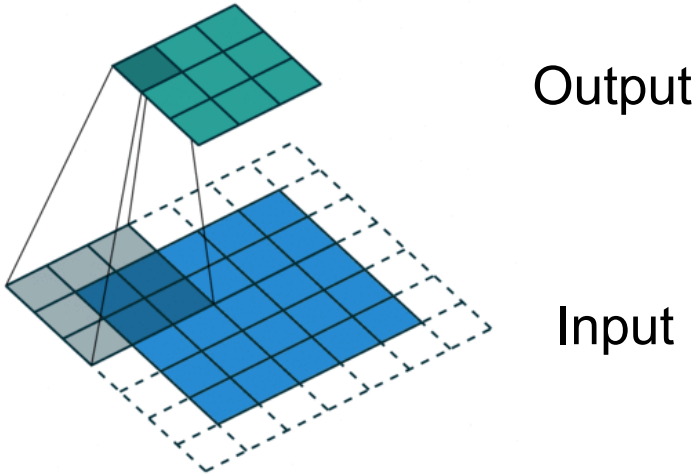
With padding, stride 1

Animation source

Convolutional architecture

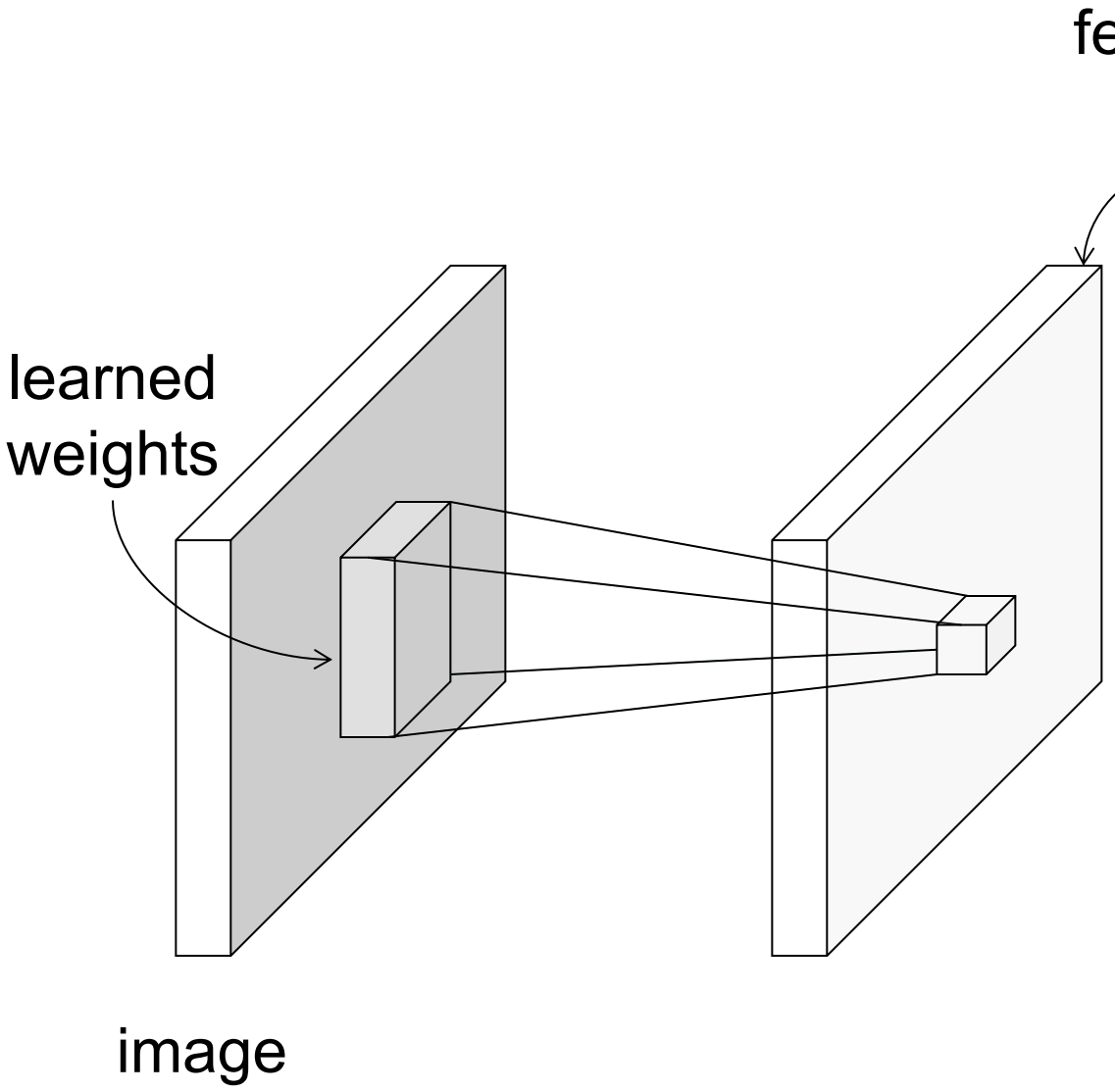


Output feature map resolution depends on *padding* and *stride*



With padding, stride 2

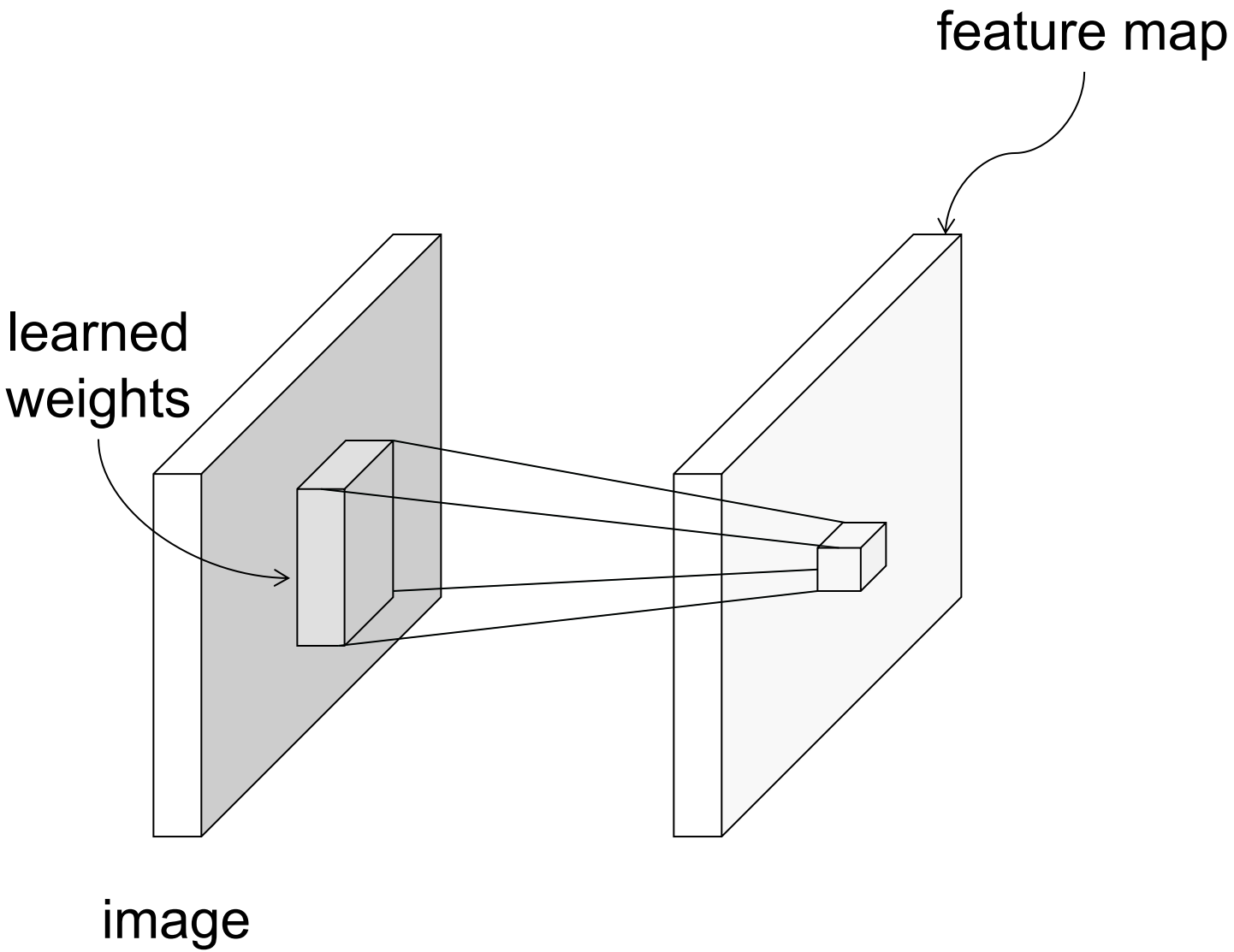
Convolutional architecture



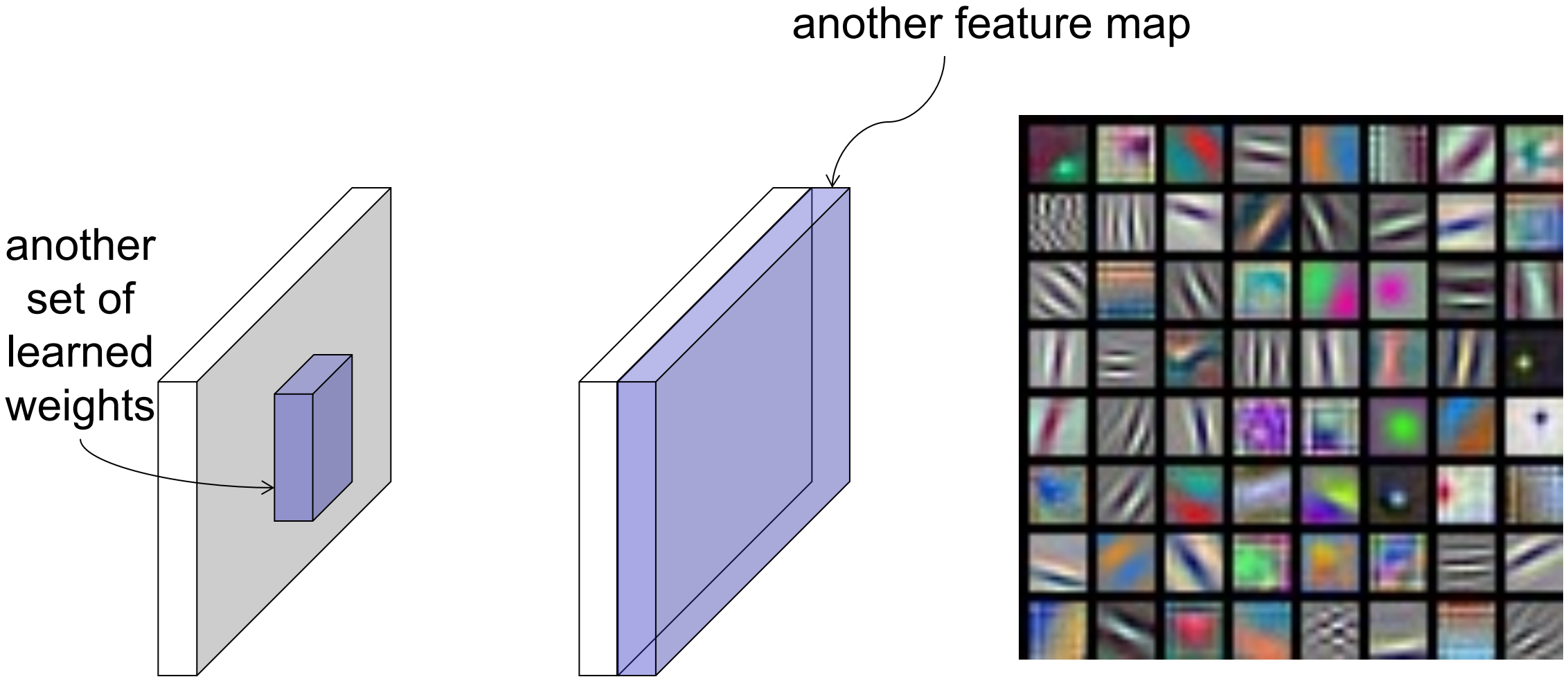
Output feature map resolution depends on *padding* and *stride*

With padding, spatial resolution remains the same if stride of 1 is used, is reduced by factor of $1/S$ if stride of S is used

Convolutional architecture



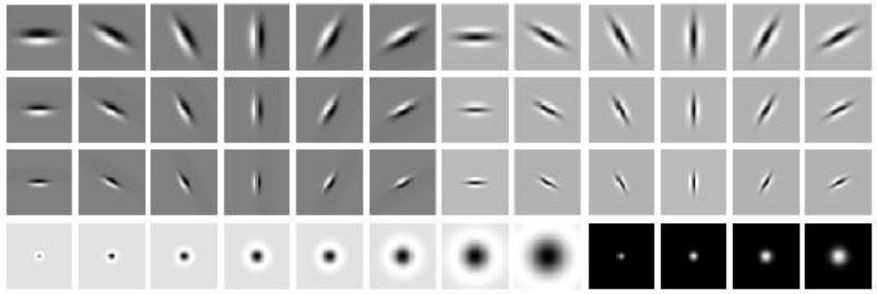
Convolutional architecture



Learned weights can be thought of as *local* templates

Convolution and traditional feature extraction

bank of K filters

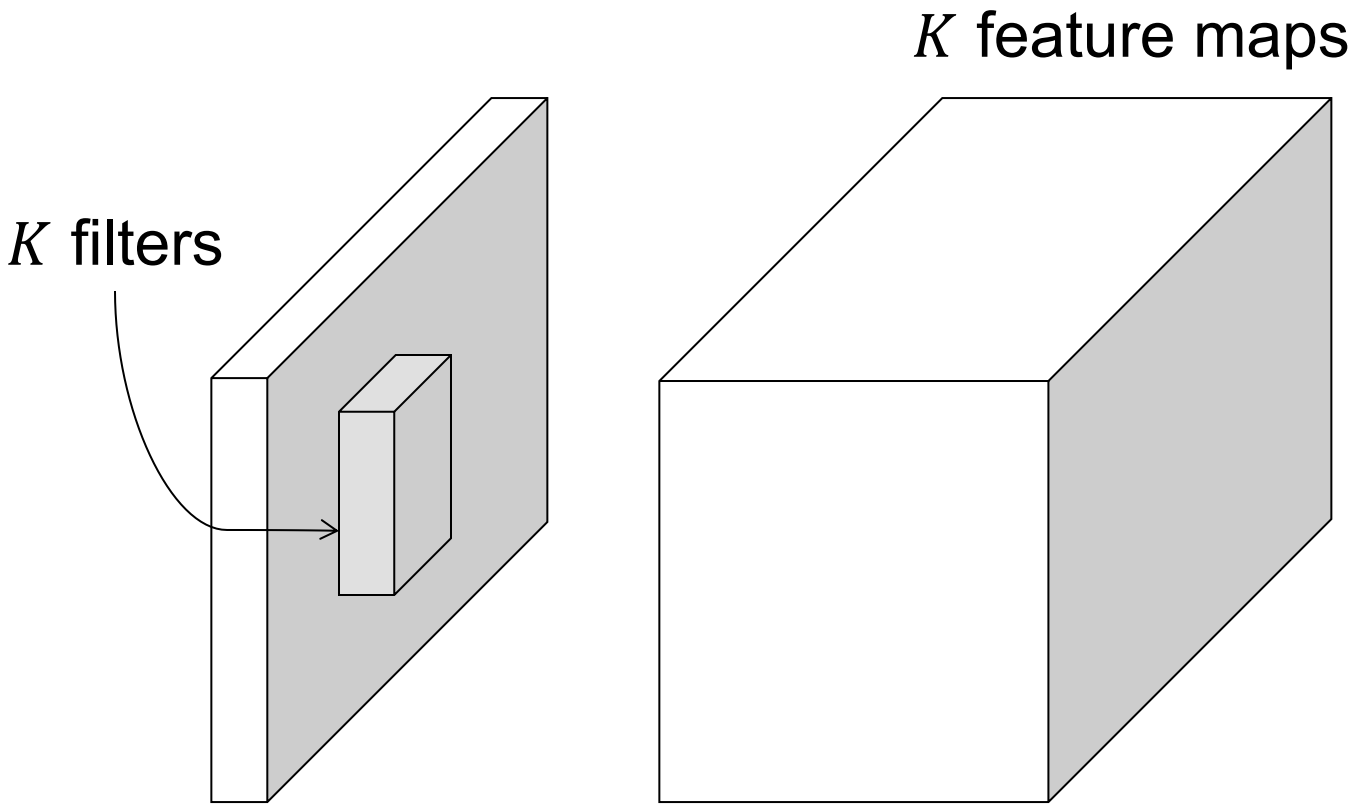


K feature maps

image

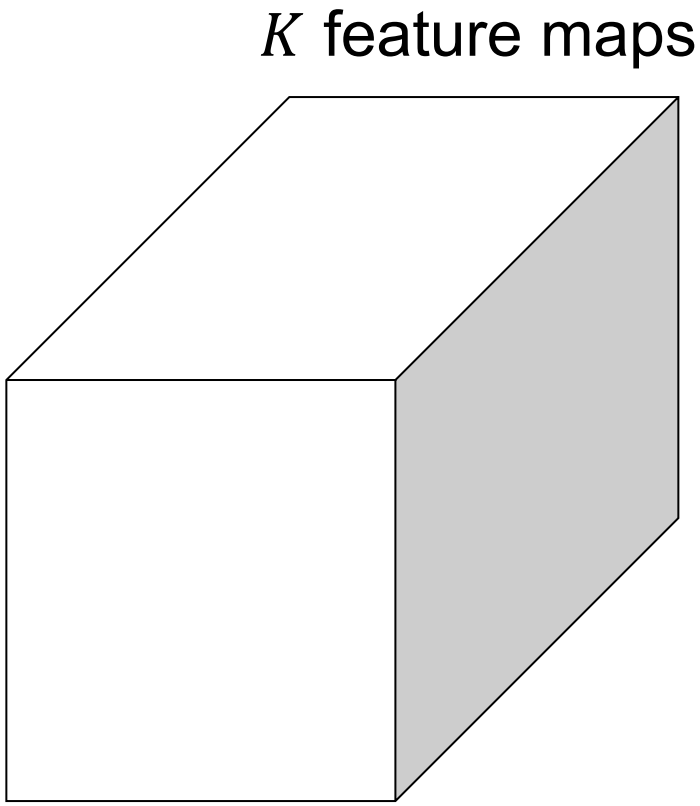
feature map

Convolutional layer anatomy



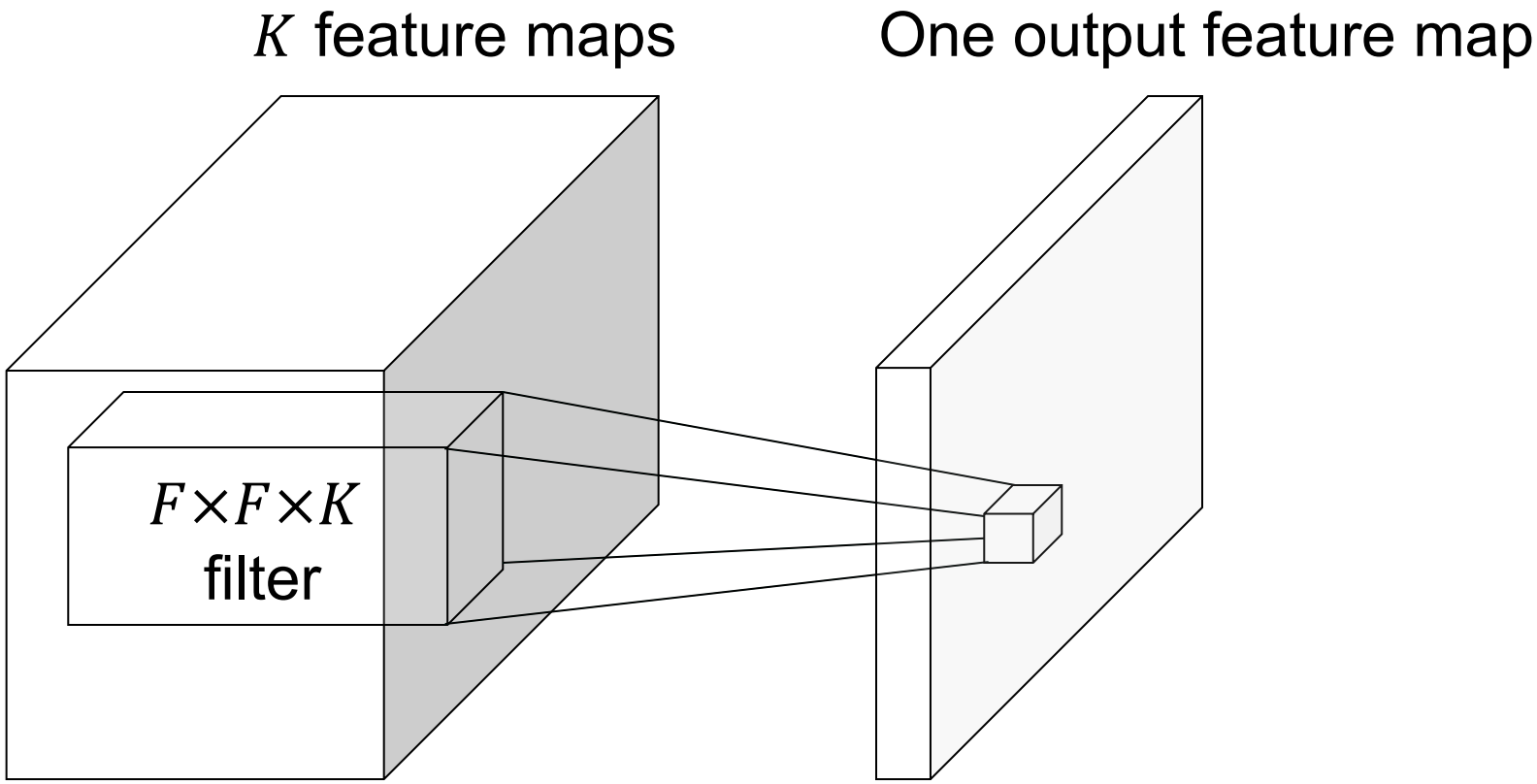
Convolutional layers are almost always directly followed by a ReLU

Convolutional layer anatomy

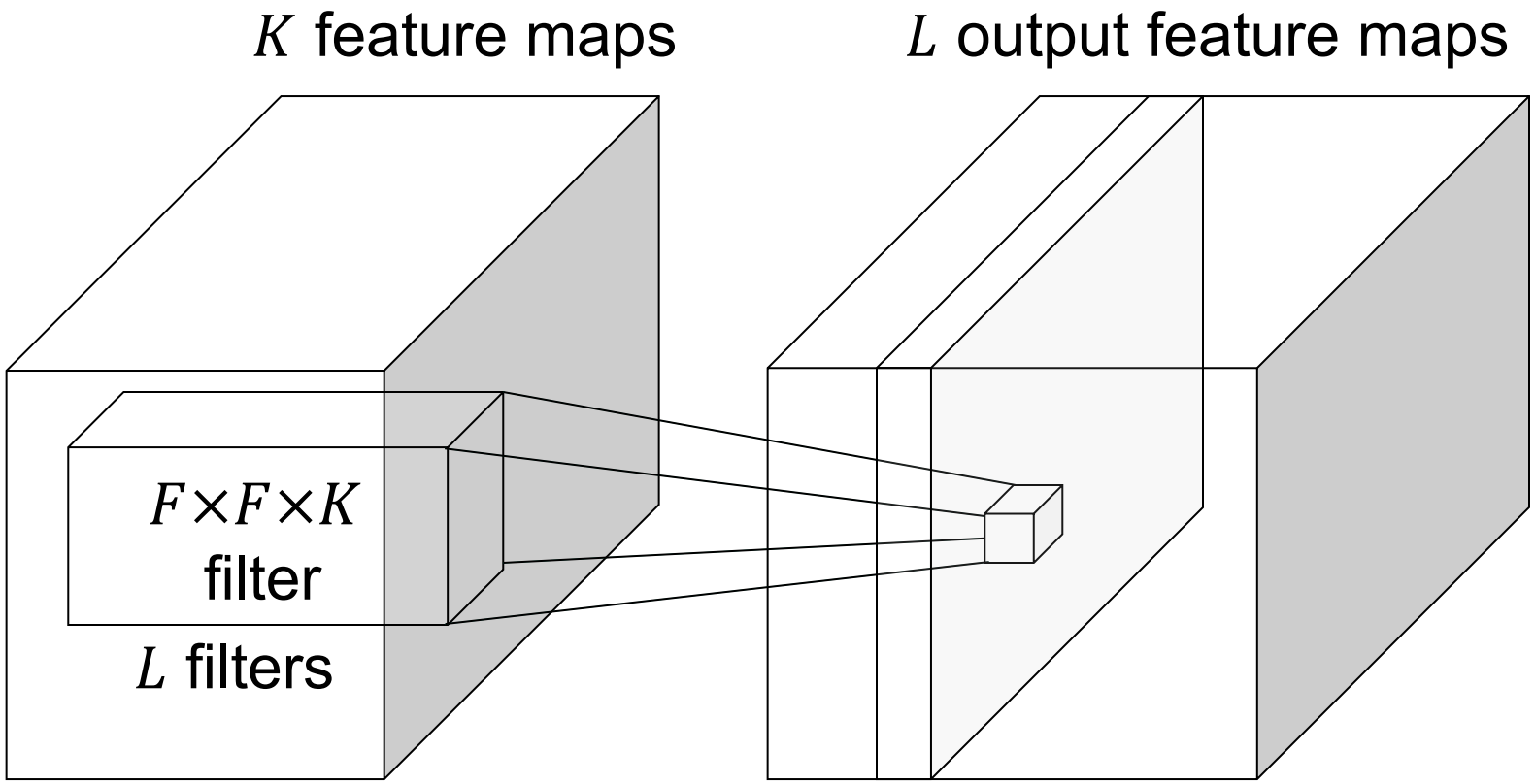


How do we put another convolutional layer on top of a stack of K feature maps?

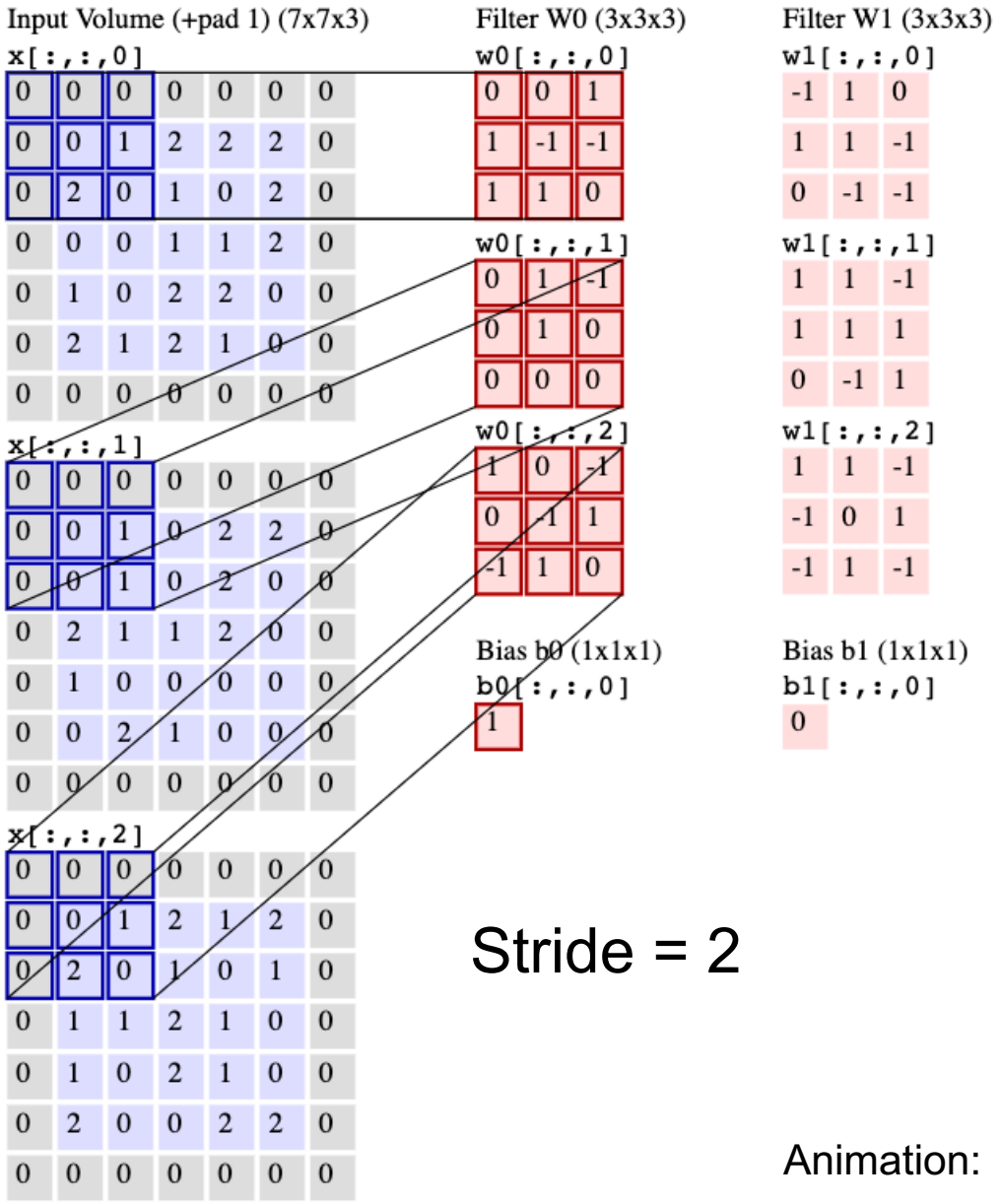
Convolutional layer anatomy



Convolutional layer anatomy



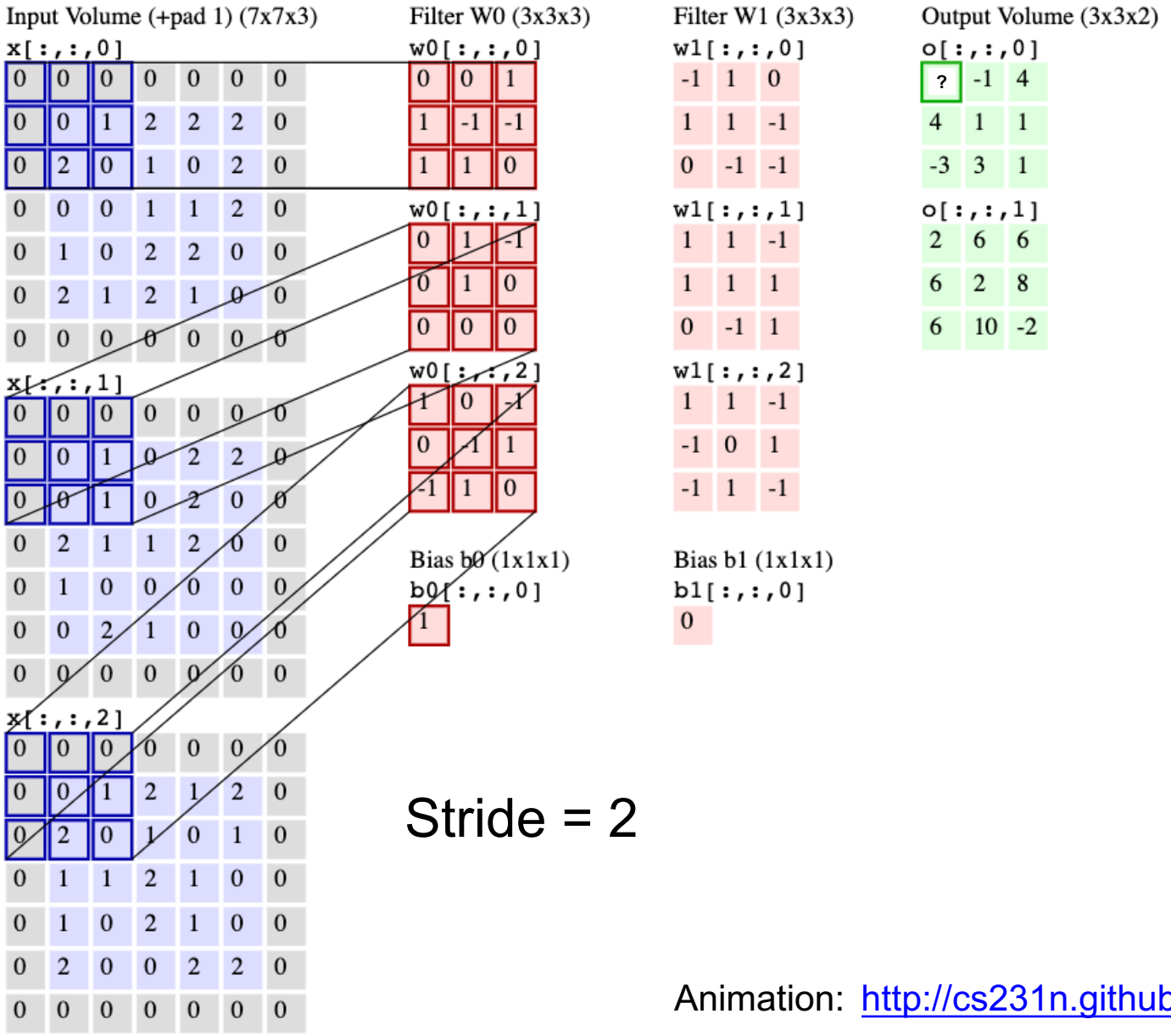
Convolutional layer example



Stride = 2

Animation: http://cs231n.github.io/convolutional-networks/#conv

Convolutional layer example



Stride = 2

Animation: http://cs231n.github.io/convolutional-networks/#conv

Convolutional layer: Details

- Efficient implementation: reshape all image neighborhoods into columns (im2col operation), do matrix-vector multiplication

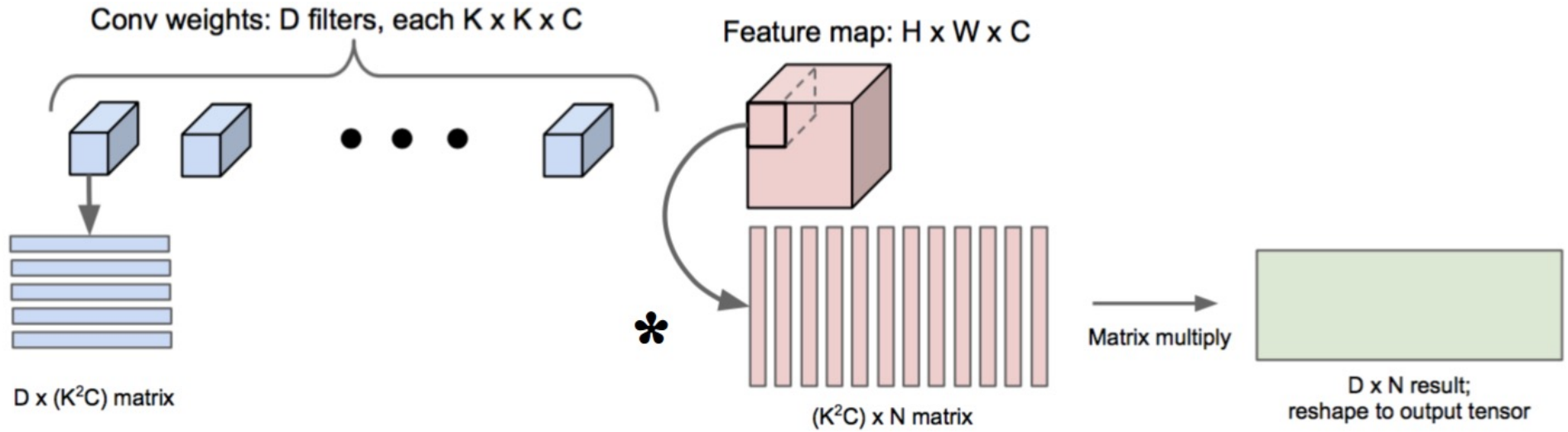
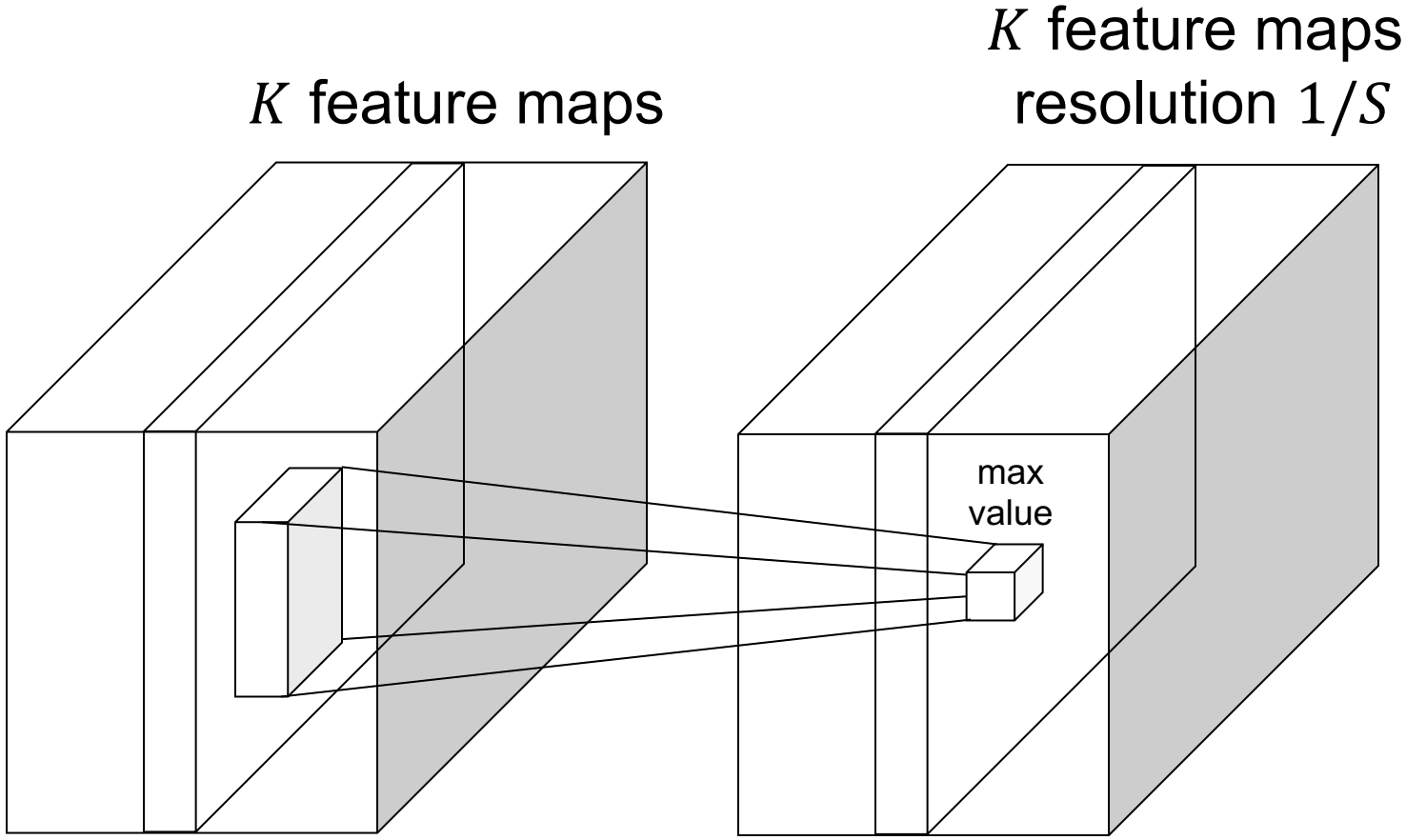


Image source

Convolutional layer: Details

- Efficient implementation: reshape all image neighborhoods into columns (im2col operation), do matrix-vector multiplication
- Backward pass: special case of linear layer, operations also turn out to be convolutions
  - Downstream gradient (of error w.r.t. input) is a *transposed convolution*, or convolution of output with filter flipped both horizontally and vertically

Max pooling layer



$F \times F$ pooling window, stride S

Usually: $F = 2$ or 3, $S = 2$

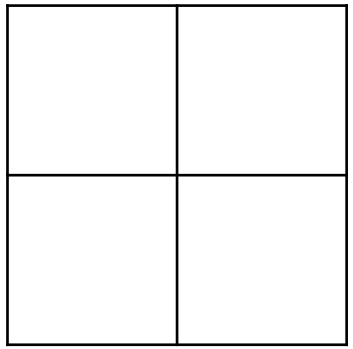
Backward pass: upstream gradient is passed back only to the unit with max value

Max pooling: Example

Single channel

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

Max pooling with 2×2 kernel size and stride 2



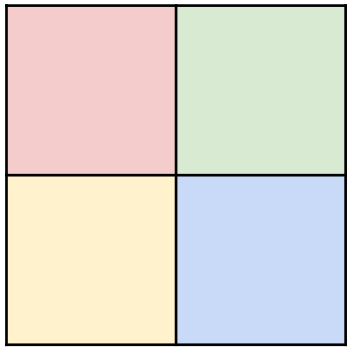
Source: J. Johnson

Max pooling: Example

Single channel

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

Max pooling with 2×2 kernel size and stride 2



Source: J. Johnson

Max pooling: Example

Single channel

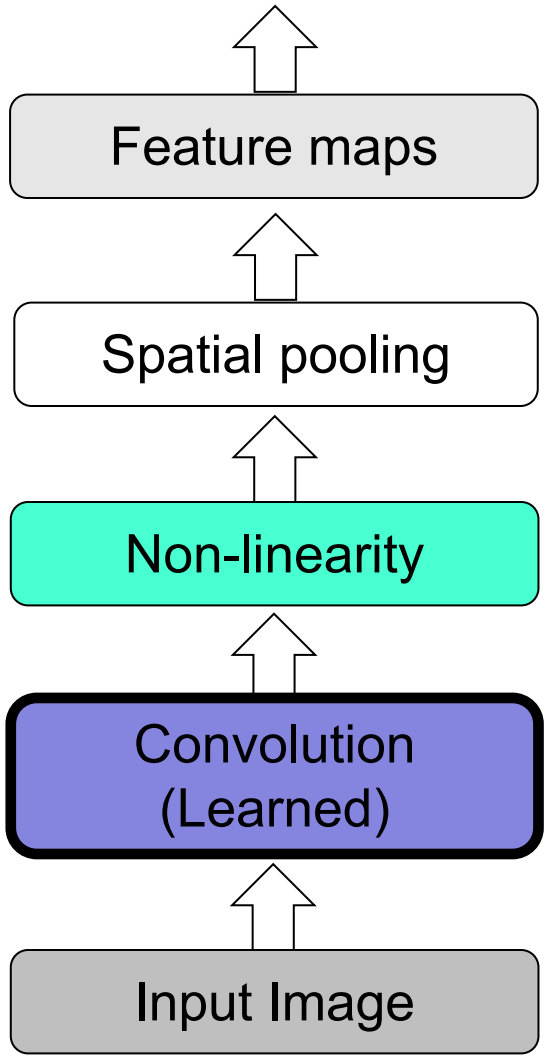
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

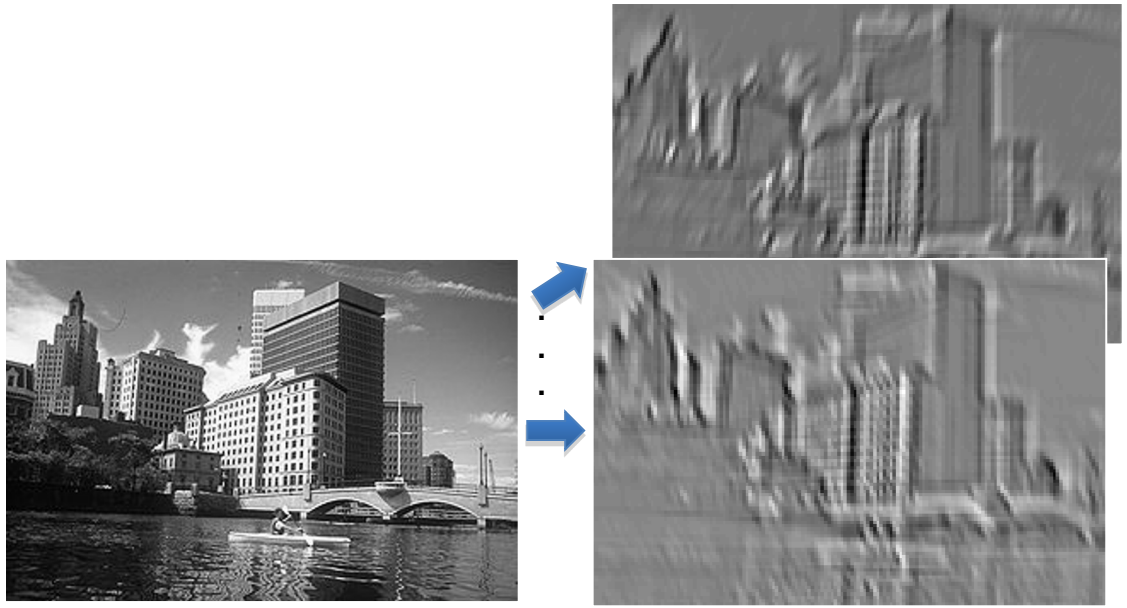
Max pooling with 2×2 kernel size and stride 2

6	8
3	4

Source: J. Johnson

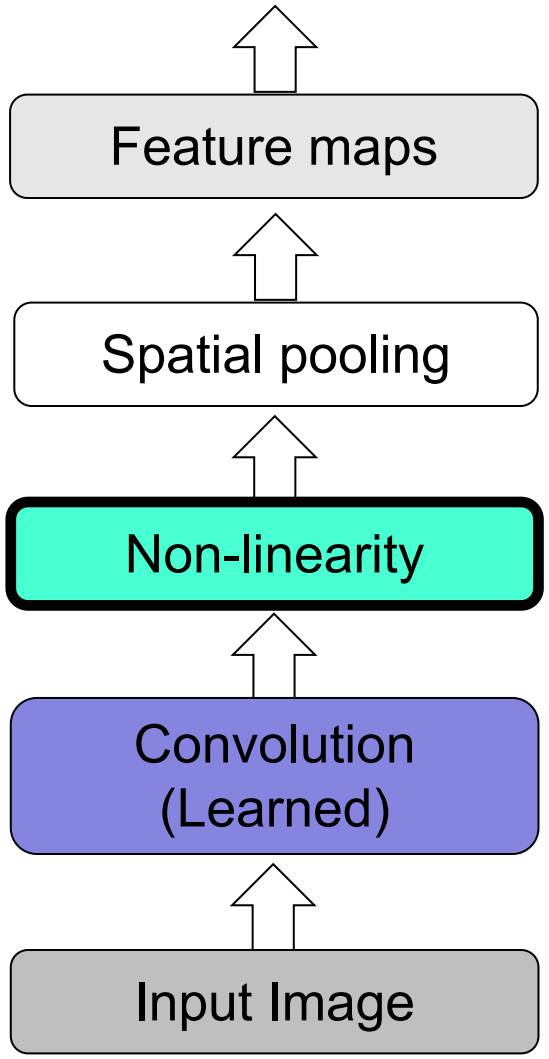
Summary: CNN pipeline





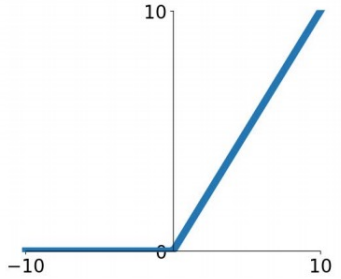
Source: R. Fergus, Y. LeCun

Summary: CNN pipeline



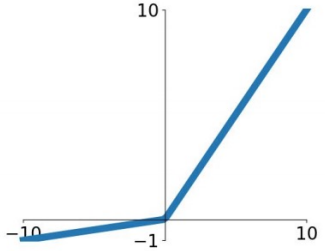
ReLU

$$\max(0, x)$$



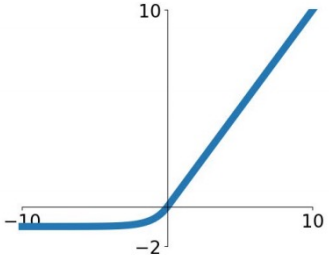
Leaky ReLU

$$\max(0.1x, x)$$



ELU

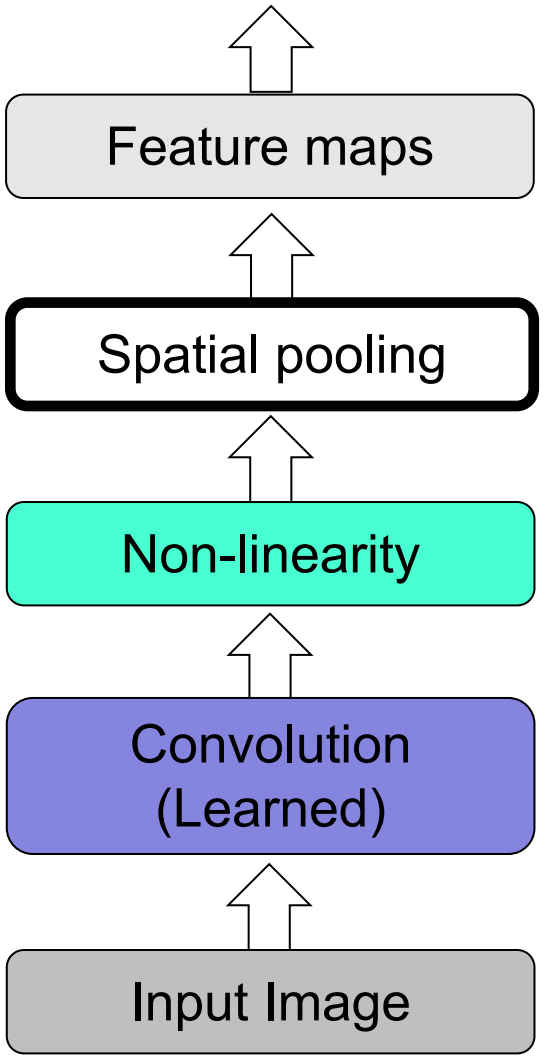
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

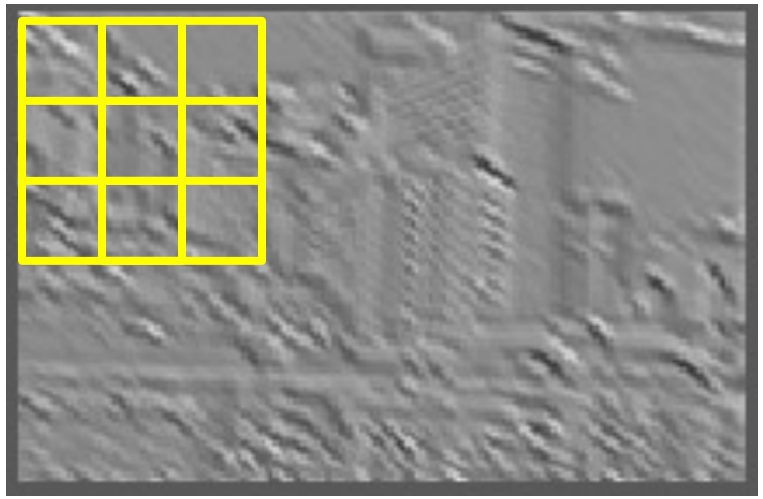


Source: R. Fergus, Y. LeCun

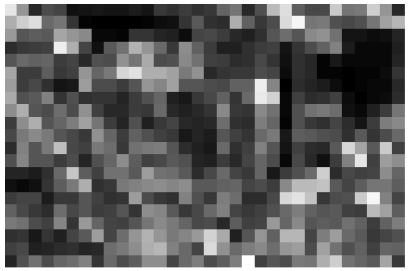
Source: Stanford 231n

Summary: CNN pipeline



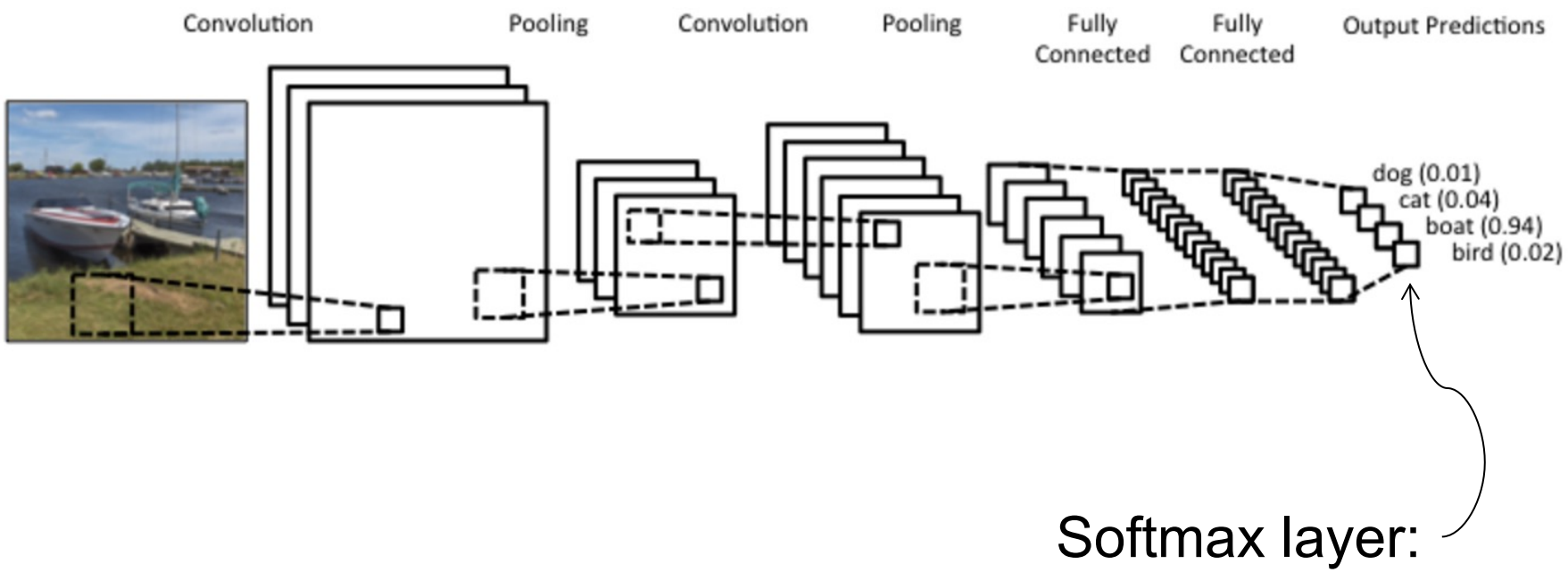


Max
(or Avg)



Source: R. Fergus, Y. LeCun

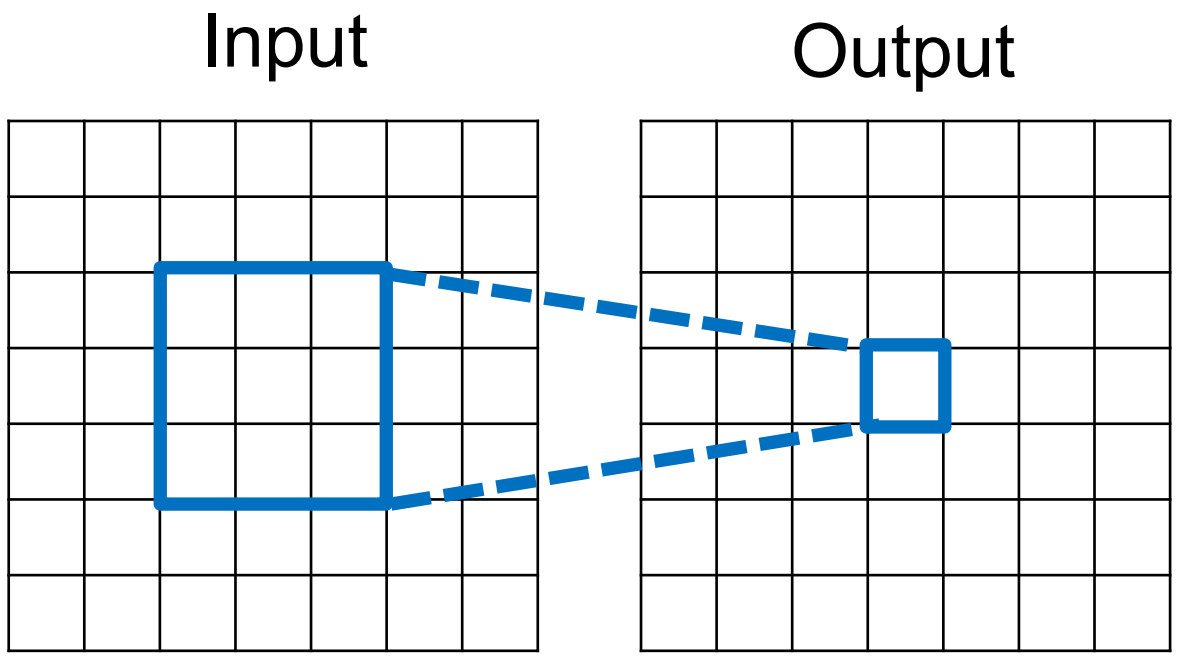
Summary: CNN pipeline



Receptive field

The *receptive field* of a unit is the region of the input feature map whose values contribute to the response of that unit (either in the previous layer or in the initial image)

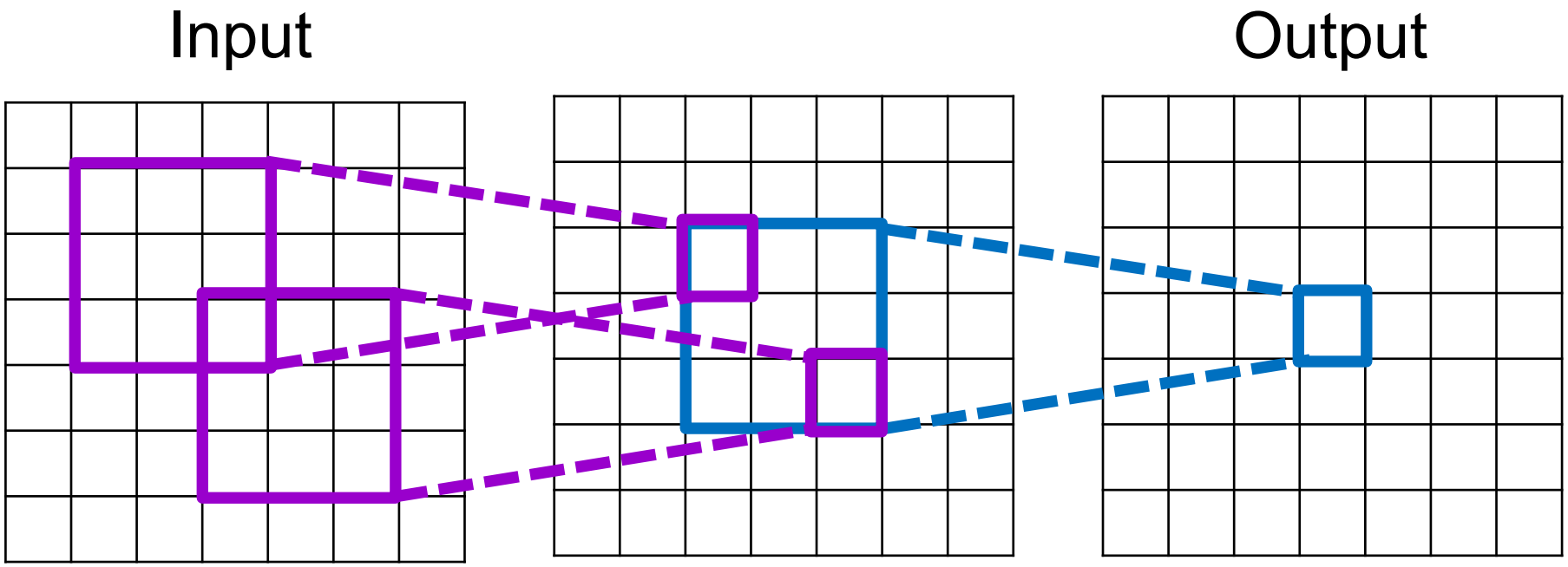
3x3 convolutions, stride 1



Receptive field size: 3

Receptive field

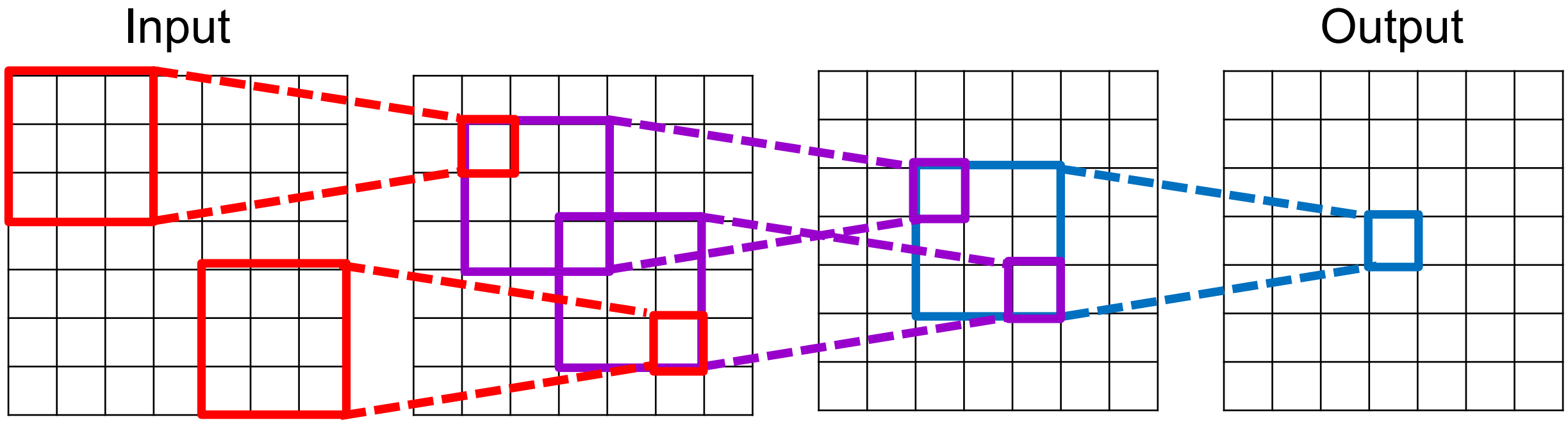
3x3 convolutions, stride 1



Receptive field size: 5

Receptive field

3x3 convolutions, stride 1



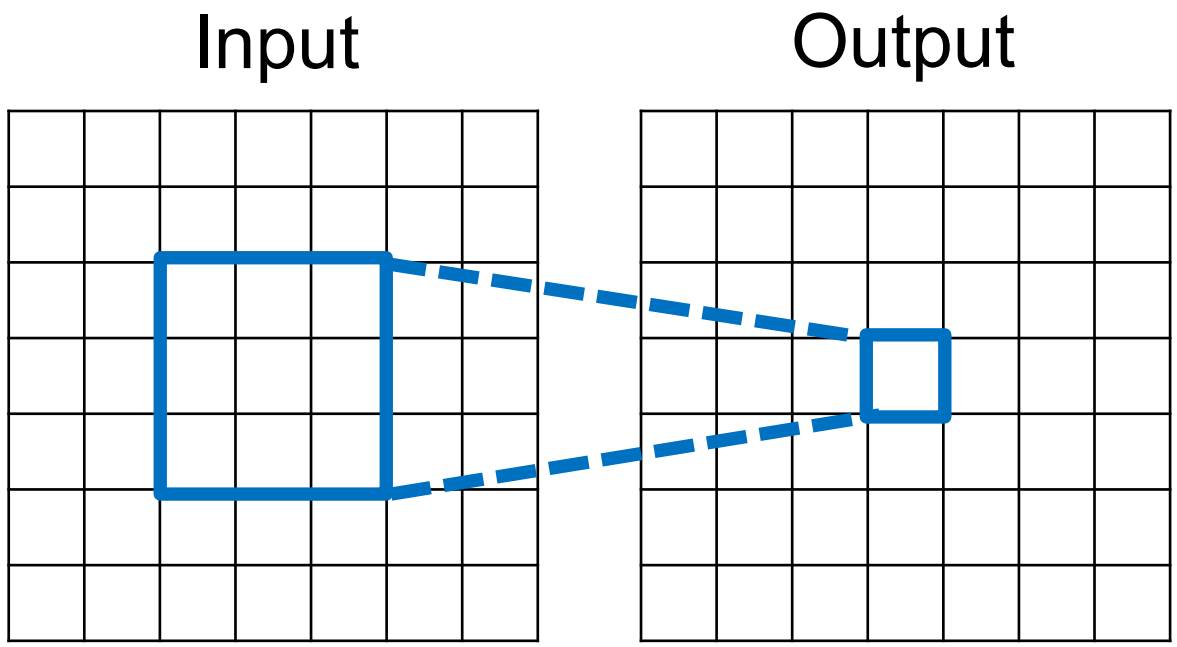
Receptive field size: 7

Each successive convolution adds $F - 1$ to the receptive field size

With L layers the receptive field size is $1 + L * (F - 1)$

Receptive field

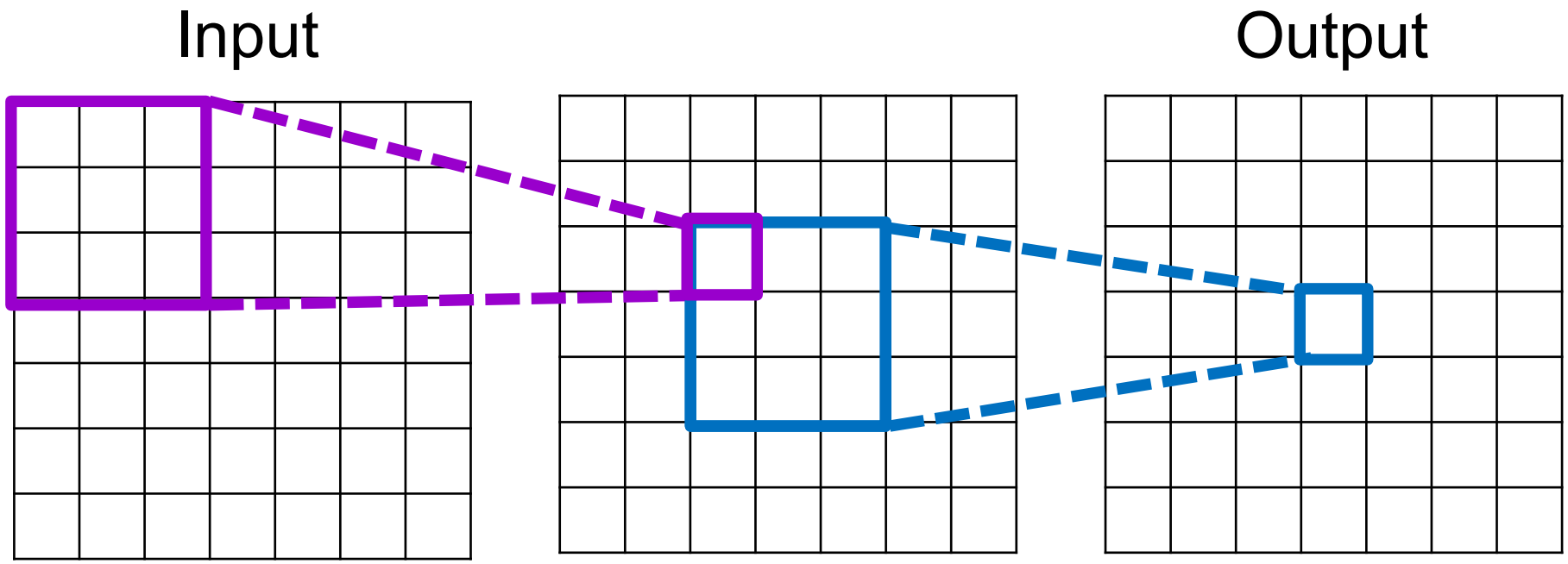
3x3 convolutions, stride 2



Receptive field size: 3

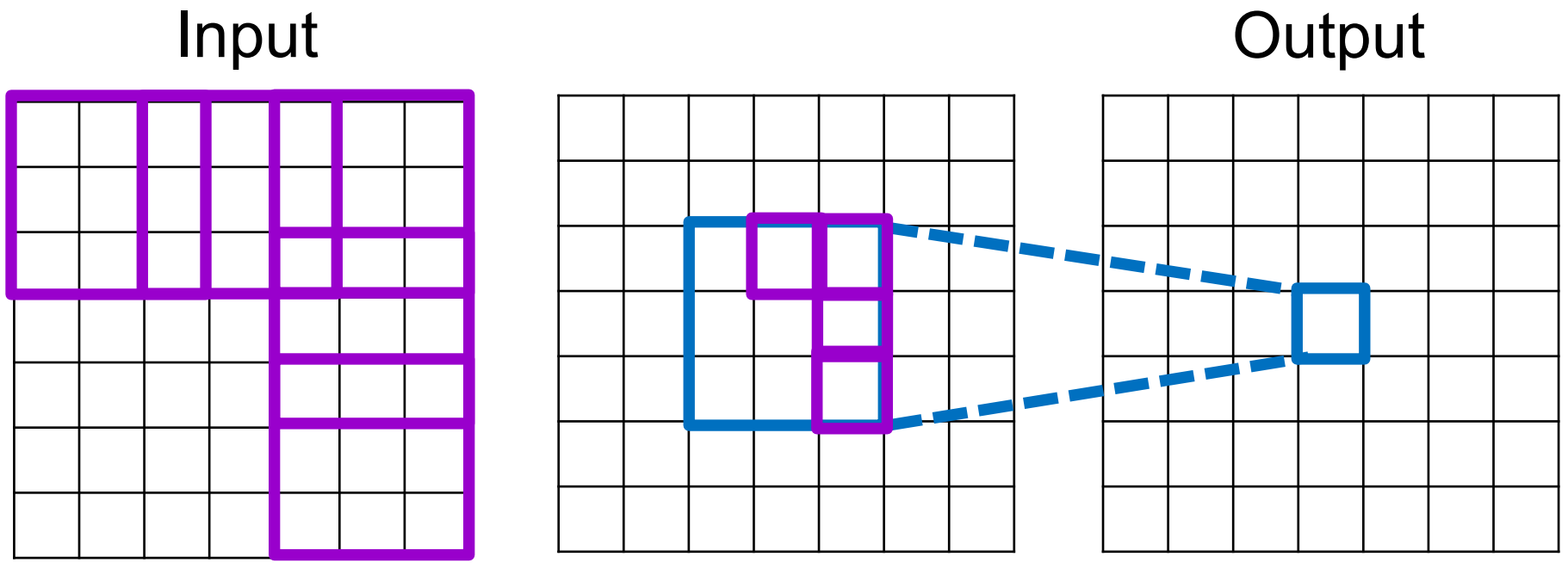
Receptive field

3x3 convolutions, stride 2



Receptive field

3x3 convolutions, stride 2

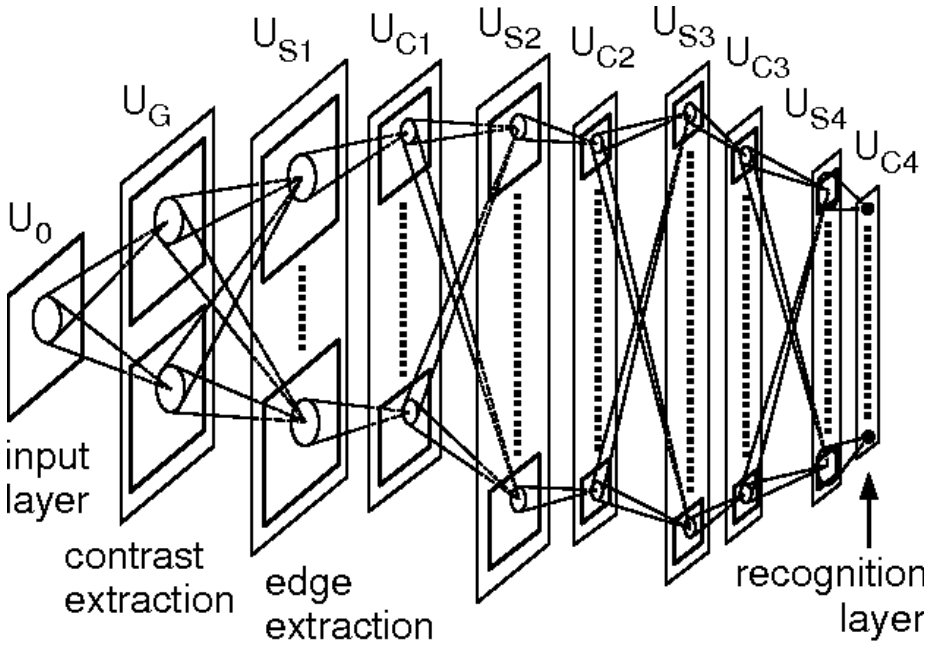


Receptive field size: 7

With a stride of 2, receptive field size is given by $2^{L+1} - 1$, i.e., it grows exponentially (though spatial resolution decreases exponentially)

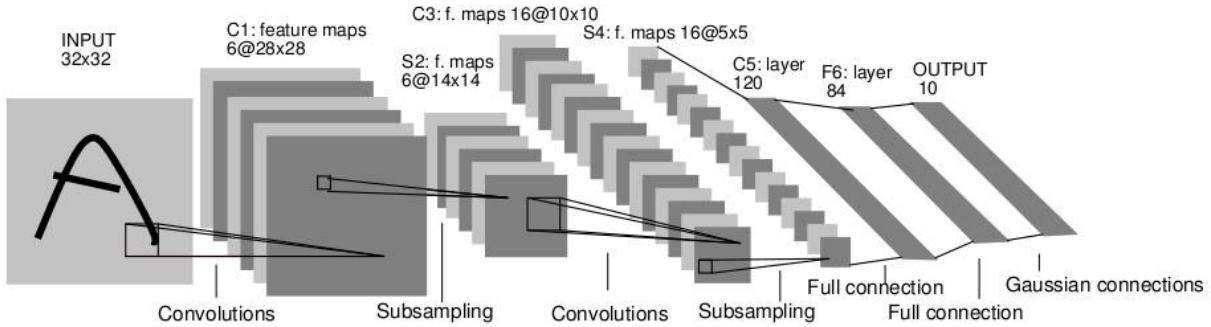
Backstory

Neocognitron



K. Fukishima. Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. Biological Cybernetics, 1980

LeNet-5



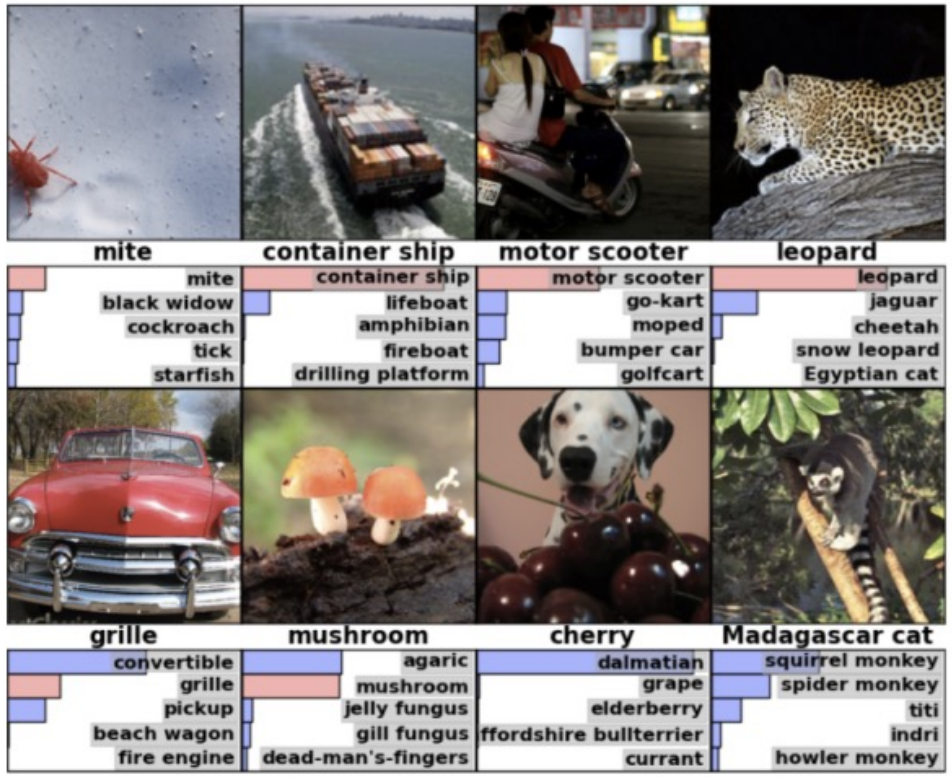
Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86(11): 2278–2324, 1998

Convolutional networks: Outline

- Building blocks
  - Convolutional layers
  - Pooling layers and nonlinearities
- Architectures:
  - 1st generation (2012-2013): AlexNet
  - 2nd generation (2014): VGGNet, GoogLeNet
  - 3rd generation (2015): ResNet
  - 4th generation (2016): WideResNet, ResNeXt, DenseNet

ImageNet Challenge

[ILSVRC]



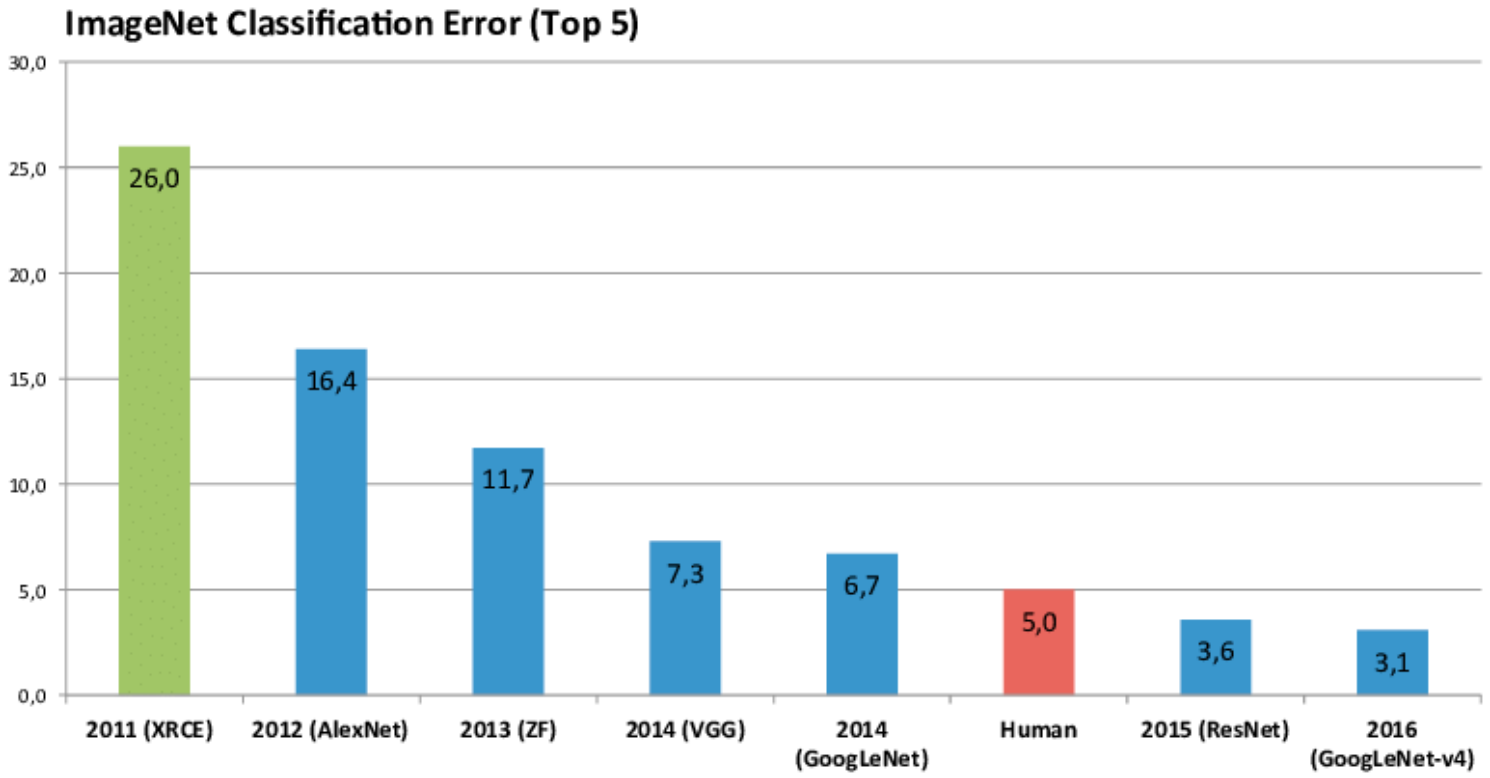
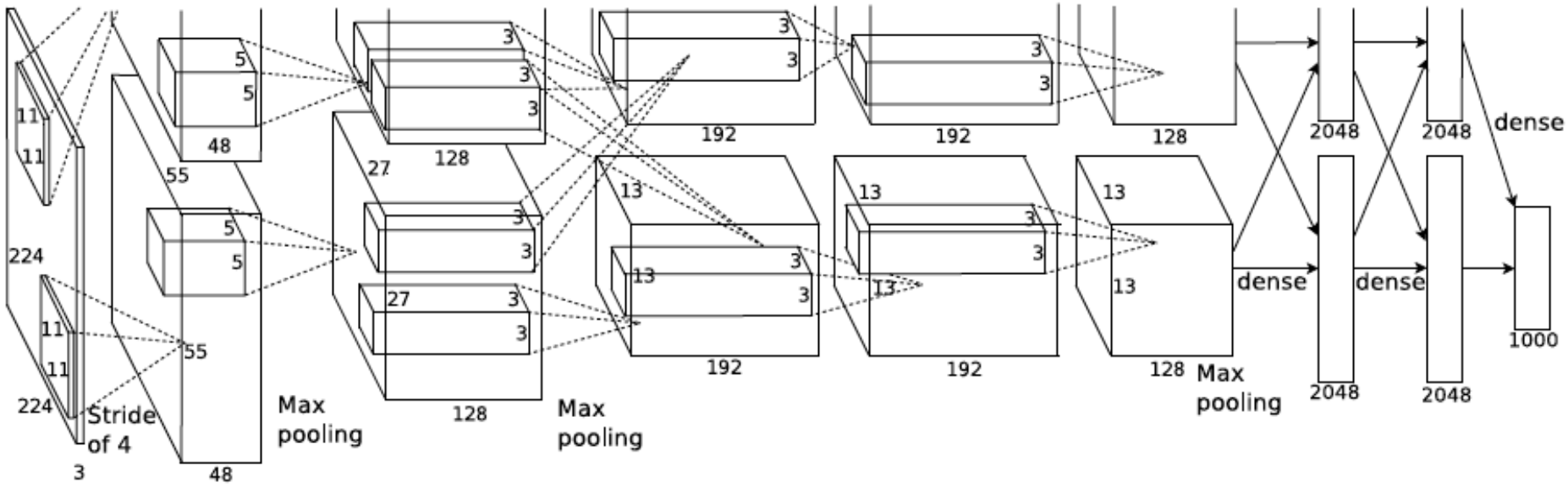


Figure source

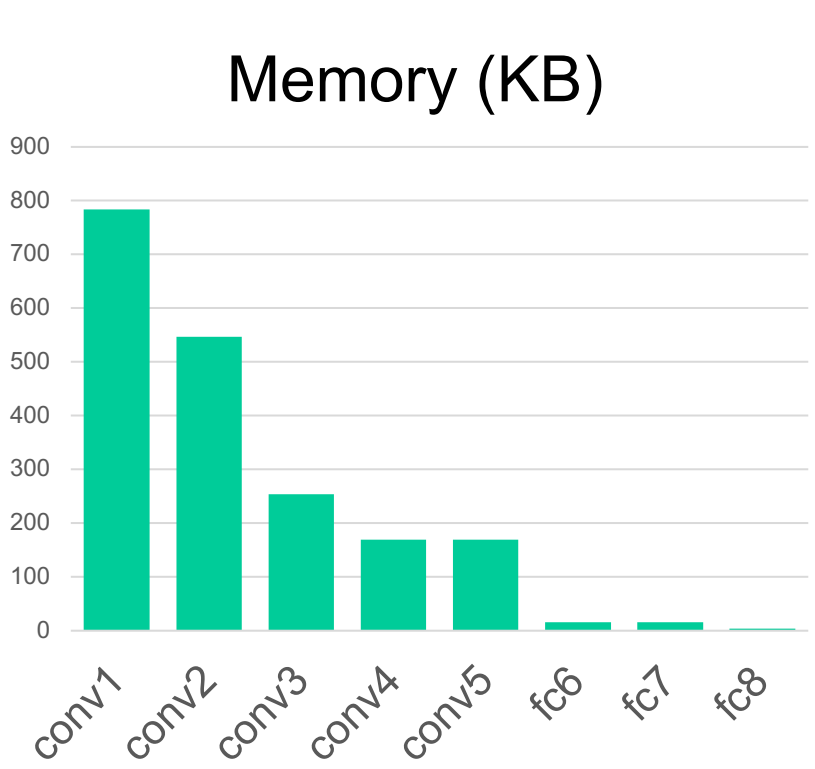
AlexNet: ILSVRC 2012 winner



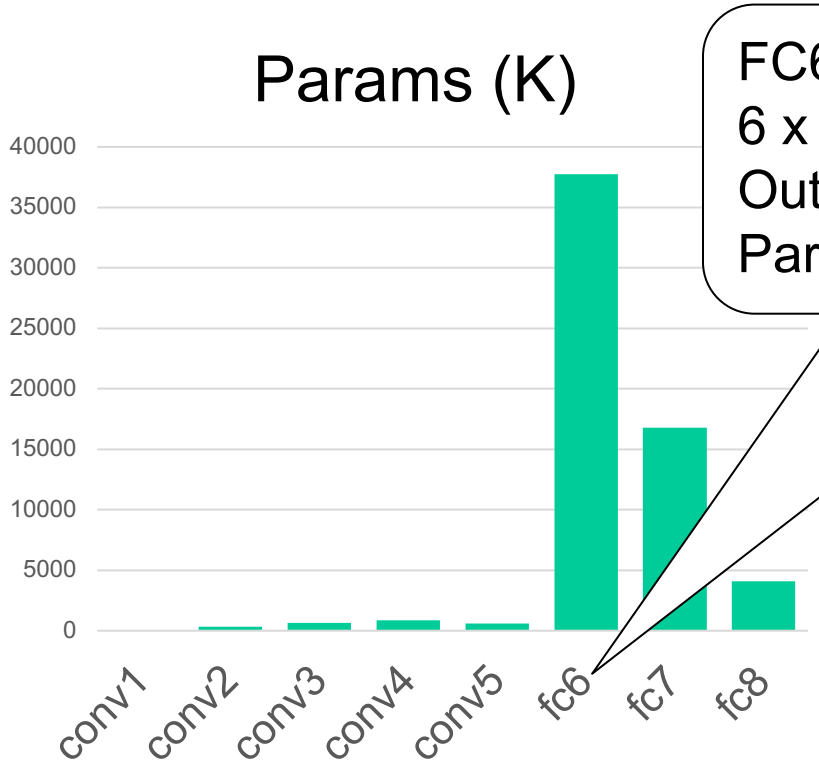
- Successor of LeNet-5 with a few crucial changes:
  - Max pooling, ReLU nonlinearity
  - Dropout regularization
  - More data and bigger model (7 hidden layers, 650K units, 60M params)
  - GPU implementation (50x speedup over CPU)
    - Trained on two GPUs for a week

A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

AlexNet by the numbers

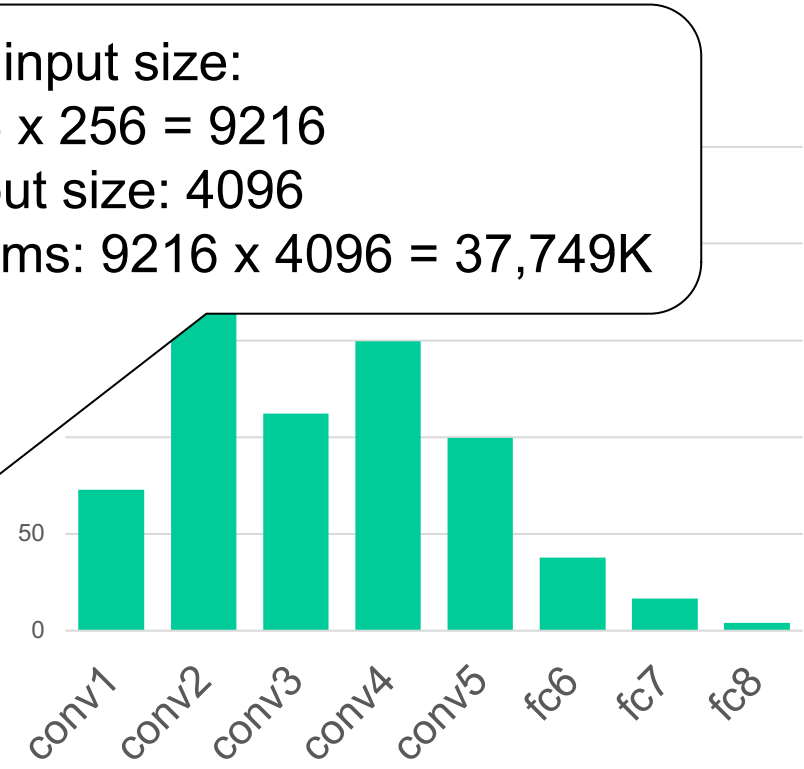


Most of the memory usage is in the early convolution layers



Nearly all parameters are in the fully-connected layers

FC6 input size:
6 x 6 x 256 = 9216
Output size: 4096
Params: 9216 x 4096 = 37,749K



Most floating-point ops occur in the convolution layers

Source: J. Johnson

Clarifai or ZFNet: ILSVRC 2013 winner

- Refinement of AlexNet

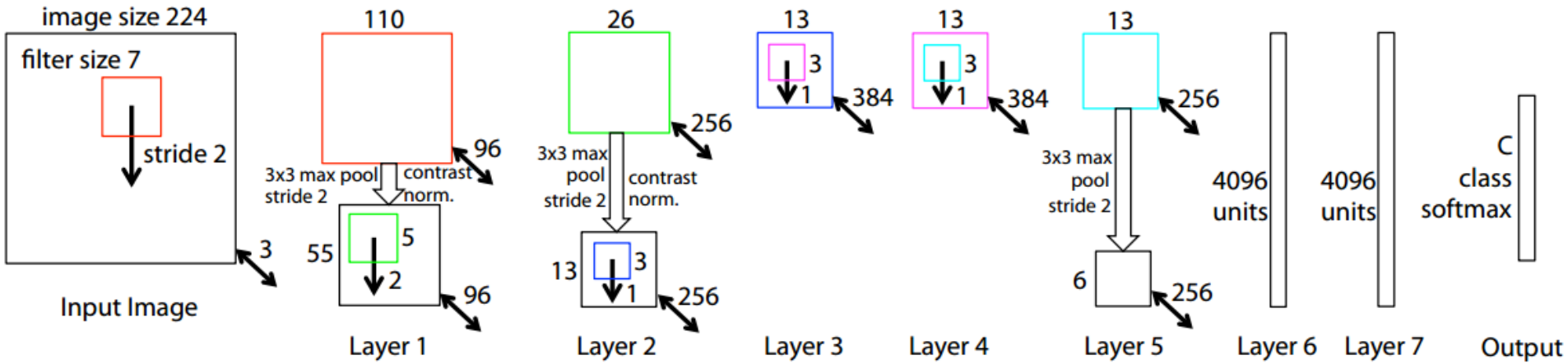


Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer filters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through a rectified linear function (not shown), (ii) pooled (max within 3x3 regions, using stride 2) and (iii) contrast normalized across feature maps to give 96 different 55 by 55 element feature maps. Similar operations are repeated in layers 2,3,4,5. The last two layers are fully connected, taking features from the top convolutional layer as input in vector form ($6 \cdot 6 \cdot 256 = 9216$ dimensions). The final layer is a C-way softmax function, C being the number of classes. All filters and feature maps are square in shape.

M. Zeiler and R. Fergus, Visualizing and Understanding Convolutional Networks, ECCV 2014 (Best Paper Award winner)

Outline

- Building blocks
  - Convolutional layers
  - Pooling layers and nonlinearities
- Architectures:
  - 1st generation (2012-2013): AlexNet
  - 2nd generation (2014): VGGNet, GoogLeNet
  - 3rd generation (2015): ResNet
  - 4th generation (2016): ResNeXt, DenseNet

VGGNet: ILSVRC 2014 2nd place

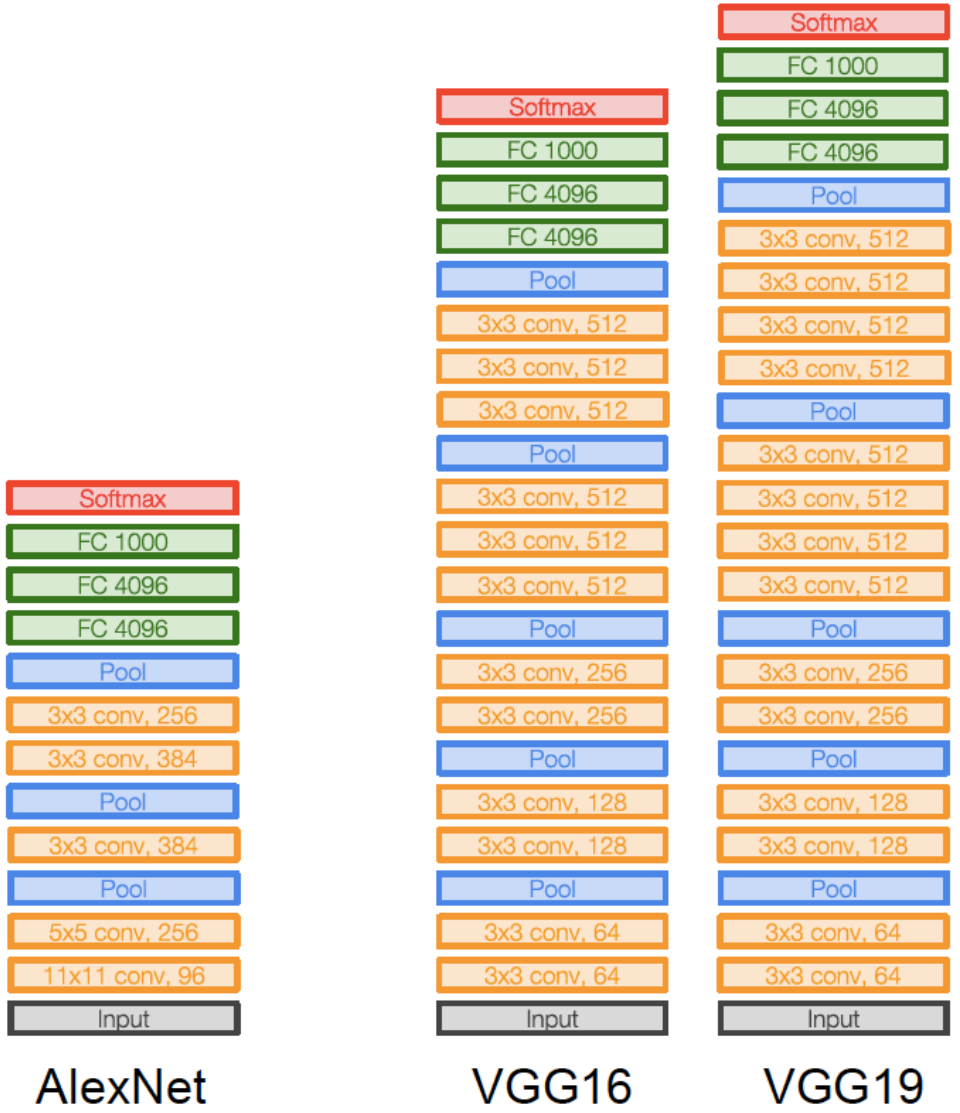


Image source

K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015

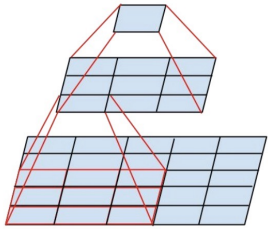
VGGNet: ILSVRC 2014 2nd place

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

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

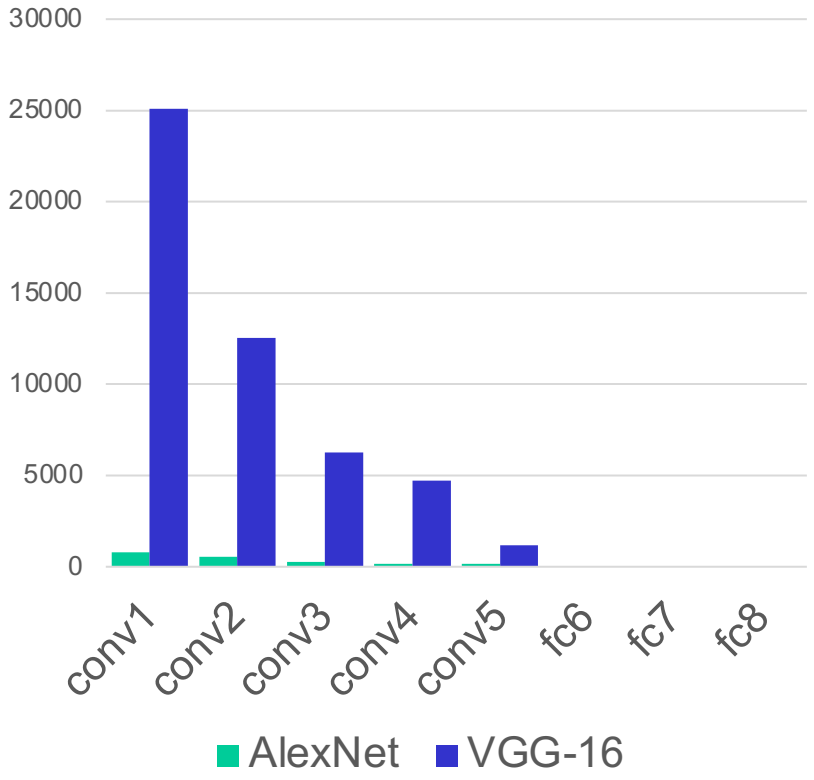
- Sequence of deeper networks trained progressively
- Large receptive fields replaced by successive layers of 3×3 convolutions (with ReLU in between)



- How many weights does one 7×7 conv layer with K input and output channels have?
  - $49K^2$
- What about three 3×3 conv layers with K input and output channels?
  - $27K^2$
- All maxpool layers are 2×2 stride 2, followed by doubling of the number of channels (keeping cost of convolutions the same)

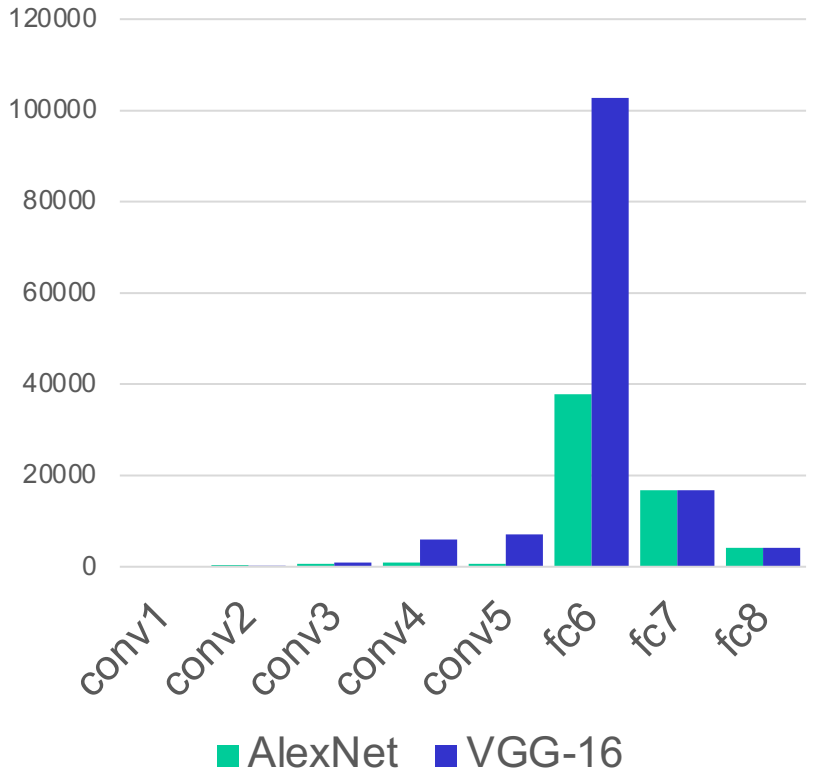
VGGNet vs. AlexNet

AlexNet vs VGG-16
(Memory, KB)



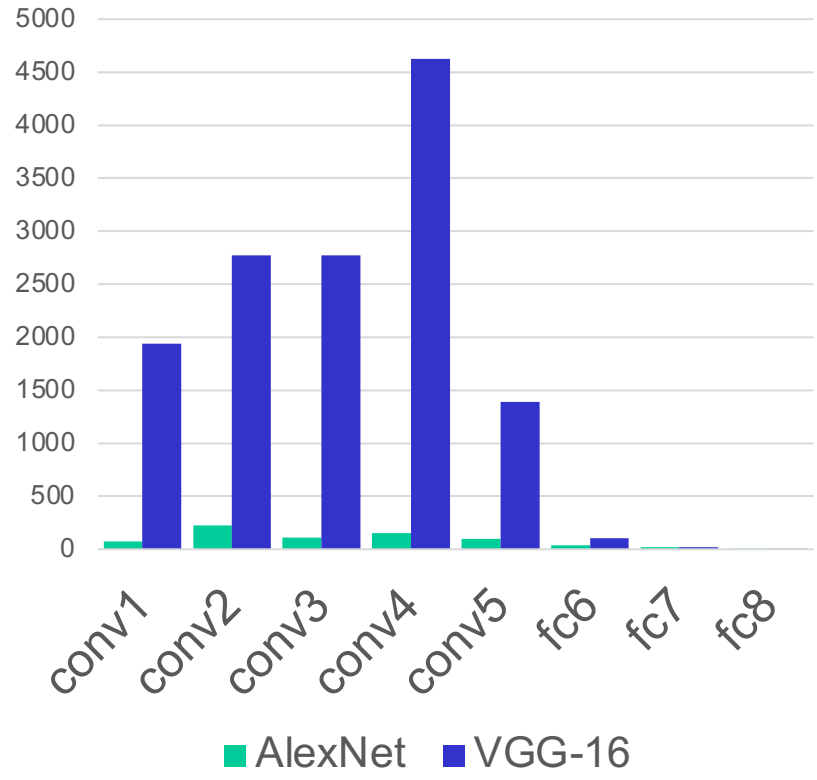
AlexNet total: 1.9 MB
VGG-16 total: 48.6 MB (25x)

AlexNet vs VGG-16
(Params, M)



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

AlexNet vs VGG-16
(MFLOPs)

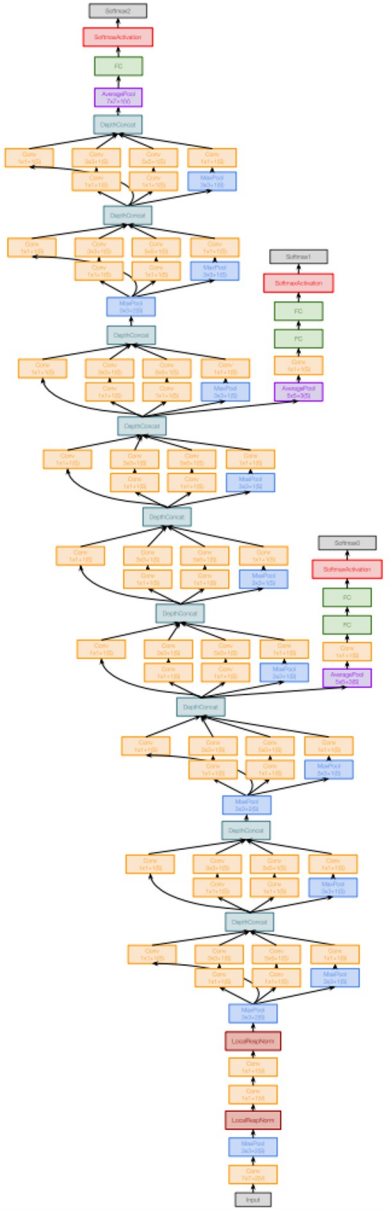


AlexNet total: 0.7 GFLOP
VGG-16 total: 13.6 GFLOP (19.4x)

Source: J. Johnson

GoogLeNet: ILSVRC 2014 winner

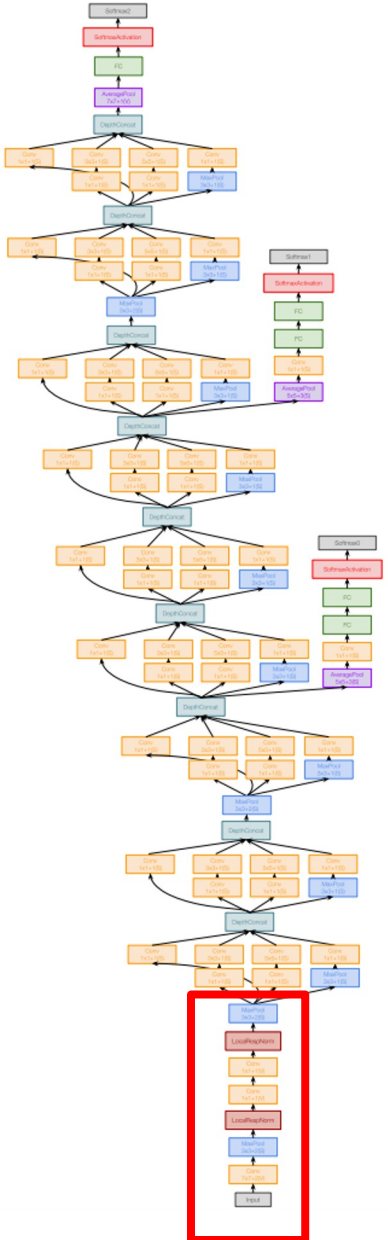
http://knowyourmeme.com/memes/we-need-to-go-deeper



C. Szegedy et al., Going deeper with convolutions, CVPR 2015

GoogLeNet: Aggressive stem

- **Stem network** at the start aggressively downsamples input

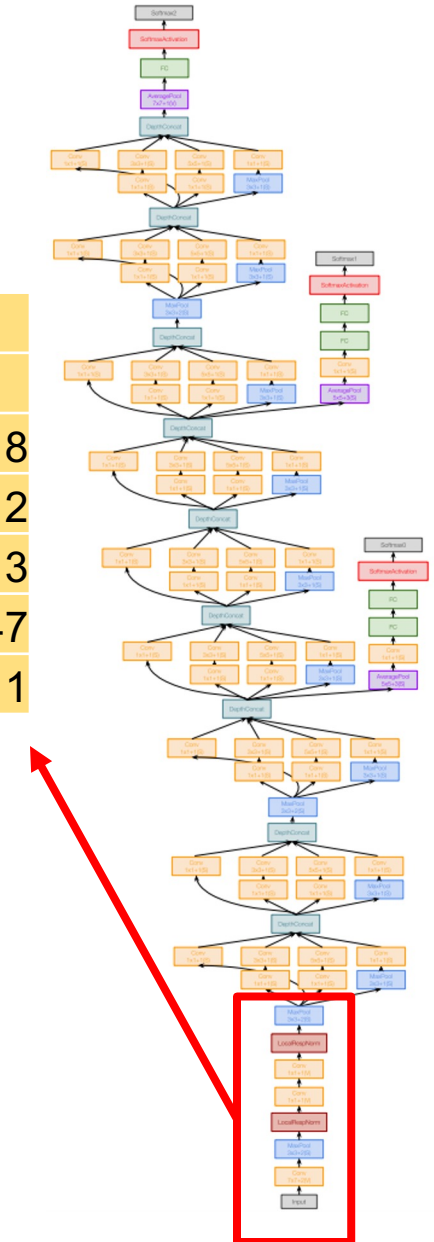


Source: J. Johnson

GoogLeNet: Aggressive stem

- **Stem network** at the start aggressively downsamples input

	Input size		Layer				Output size				
Layer	C	H / W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (K)	flop (M)
conv	3	224	64	7	2	3	64	112	3136	9	118
max-pool	64	112		3	2	1	64	56	784	0	2
conv	64	56	64	1	1	0	64	56	784	4	13
conv	64	56	192	3	1	1	192	56	2352	111	347
max-pool	192	56		3	2	1	192	28	588	0	1

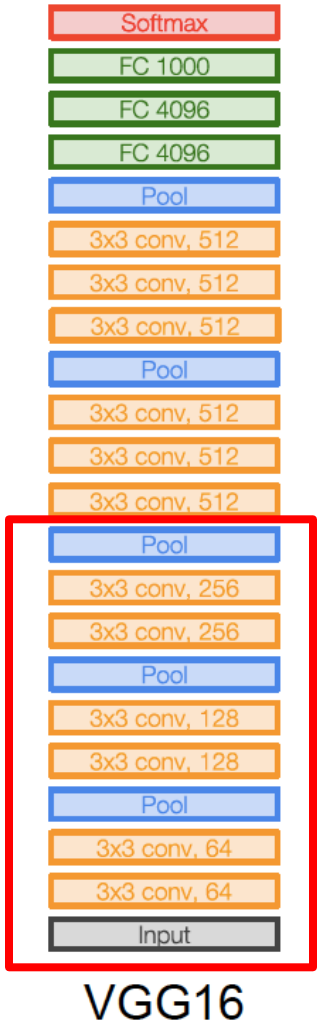


Total from 224 to 28 resolution:
Memory: 7.5 MB
Params: 124K
MFLOP: 418

Source: J. Johnson

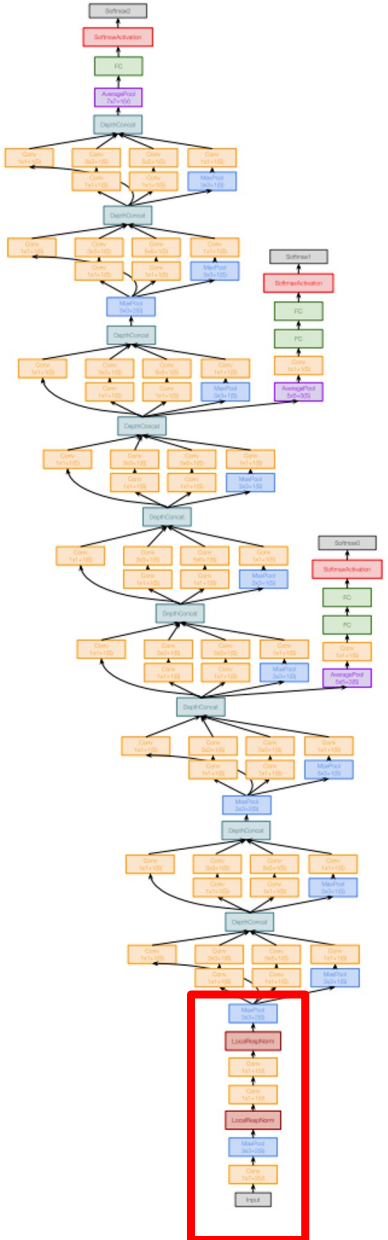
GoogLeNet: Aggressive stem

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



VGG16

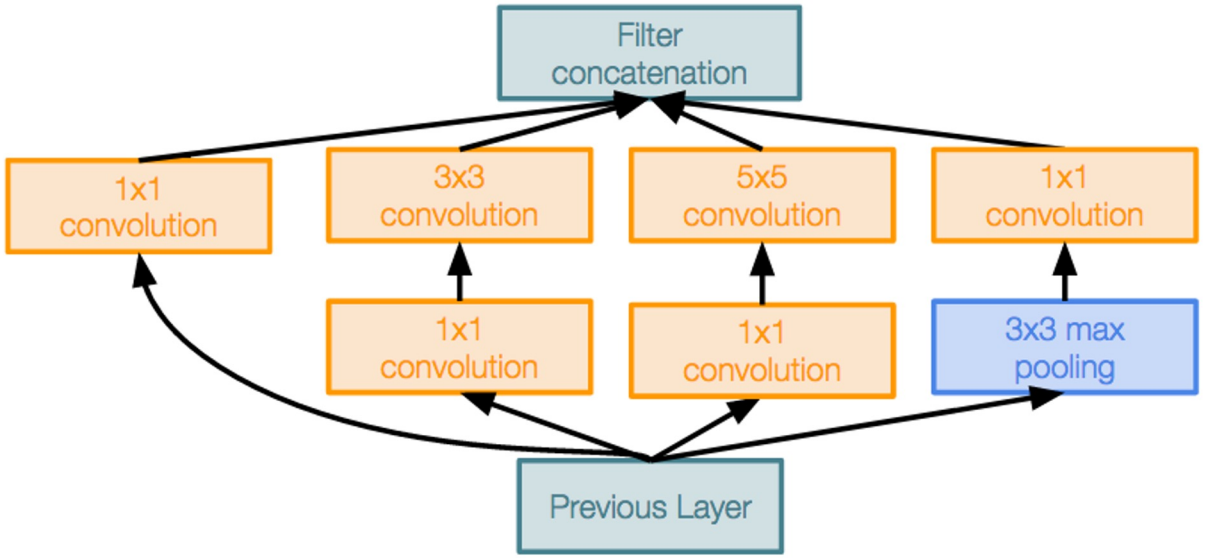
Total from 224 to 28 resolution:
Memory: 7.5 MB
Params: 124K
MFLOP: 418

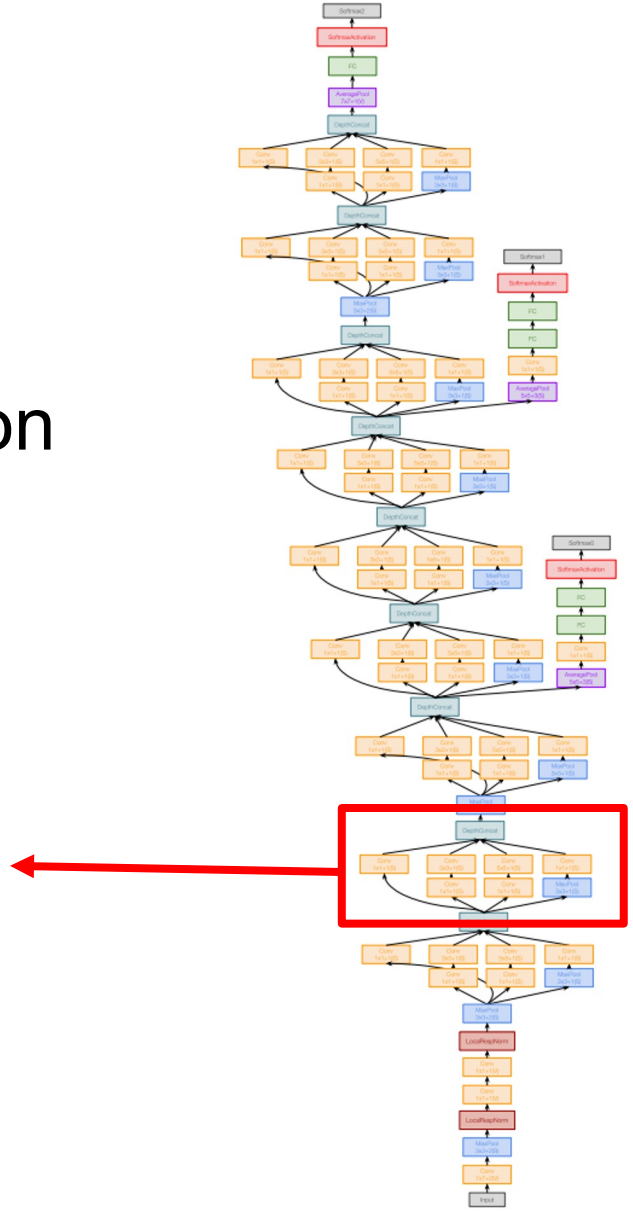


Source: J. Johnson

GoogLeNet: Inception module

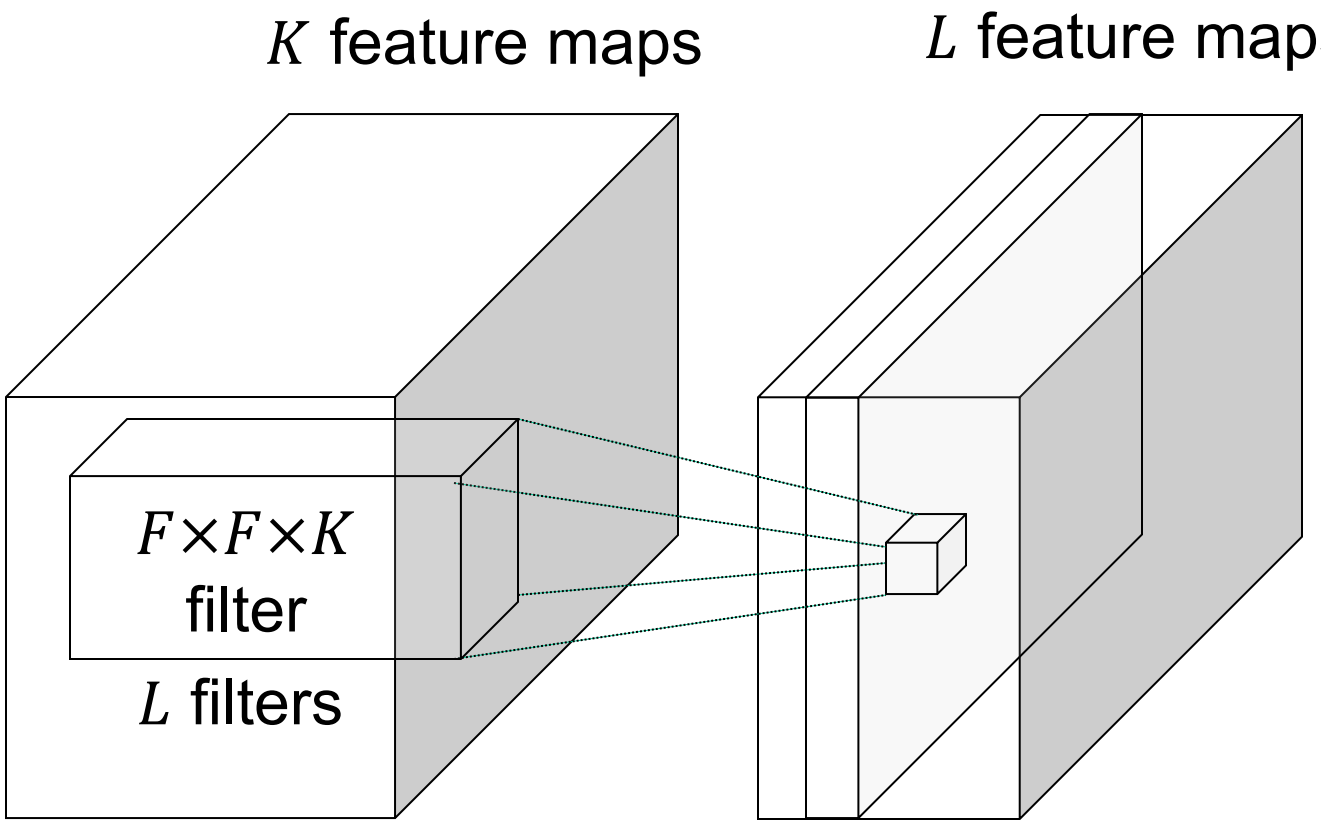
- Parallel paths with different receptive field sizes and operations are meant to capture sparse patterns of correlations in the stack of feature maps
- Use 1x1 convolutions for dimensionality reduction before expensive convolutions



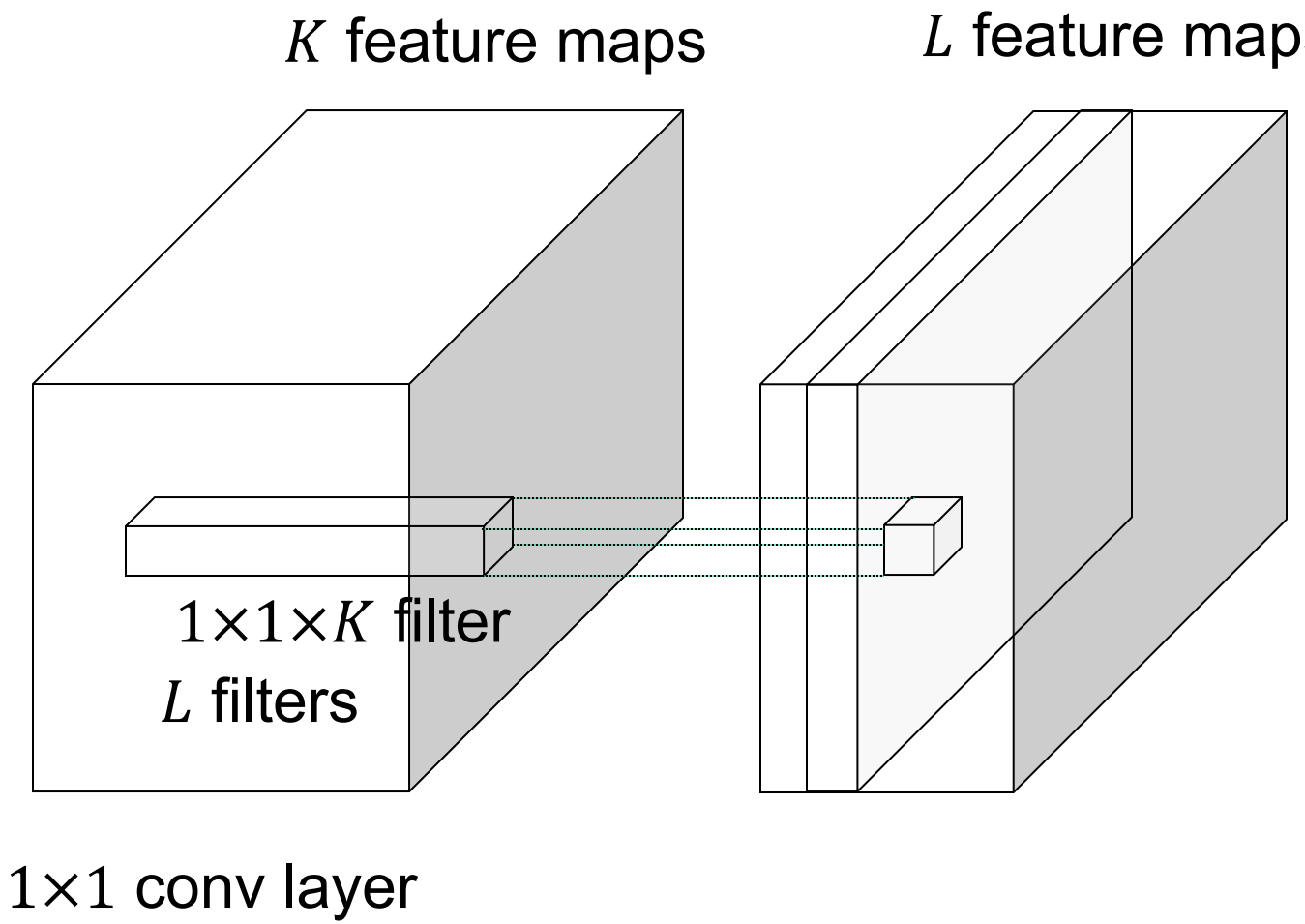


Source: J. Johnson

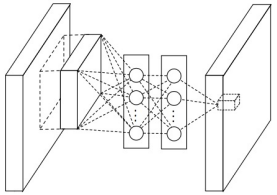
Recall: Regular conv layer



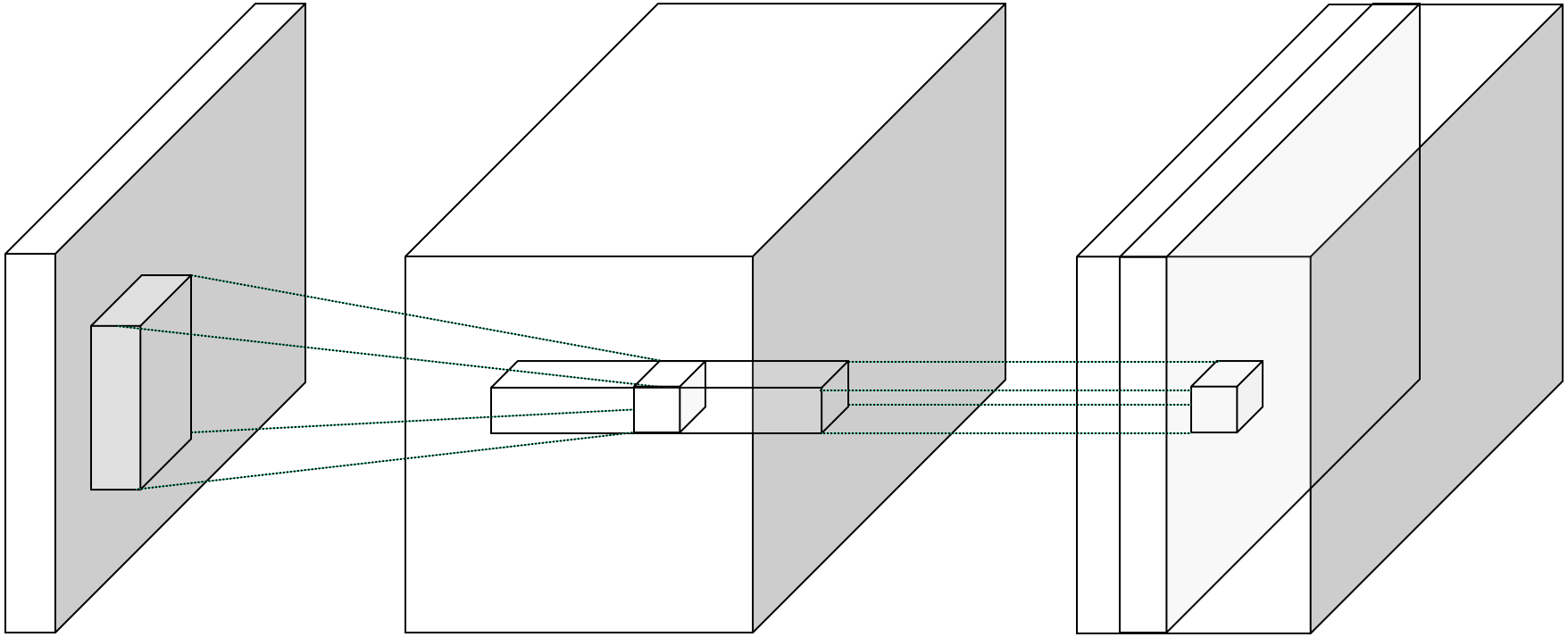
1x1 convolution



1x1 convolution



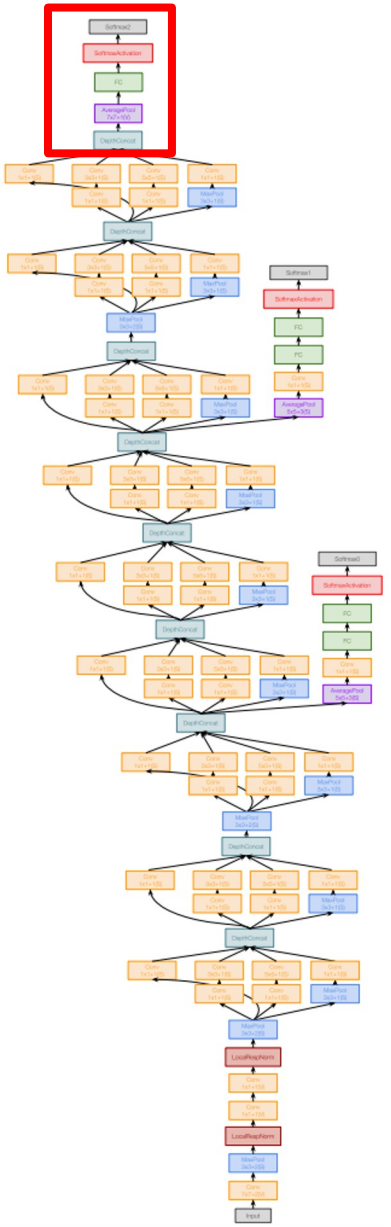
“network in network”



M. Lin, Q. Chen, and S. Yan, Network in network, ICLR 2014

GoogLeNet: Global average pooling

- Instead of large FC layers at the end, use *global average pooling* to collapse spatial dimensions, followed by linear layer to produce class scores
  - Recall VGG-16: Most parameters were in the FC layers!

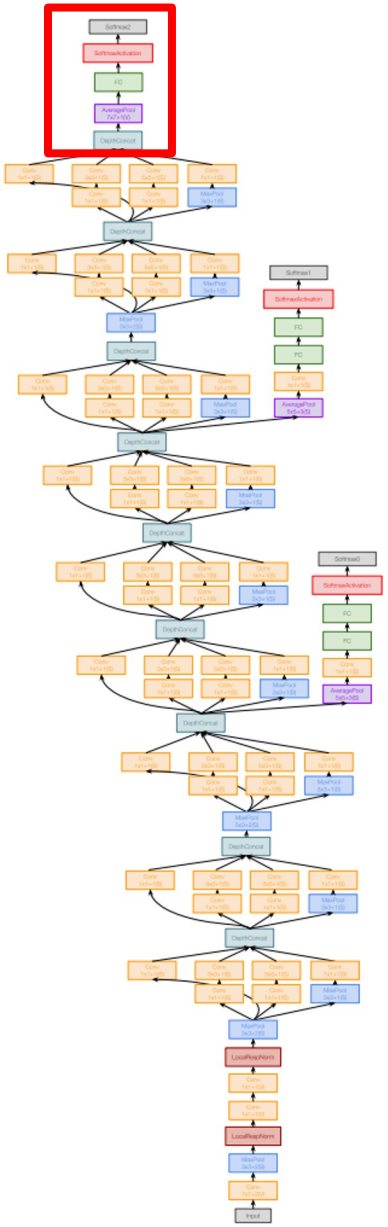


Source: J. Johnson

GoogLeNet: Global average pooling

GoogLeNet head:

	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



Source: J. Johnson

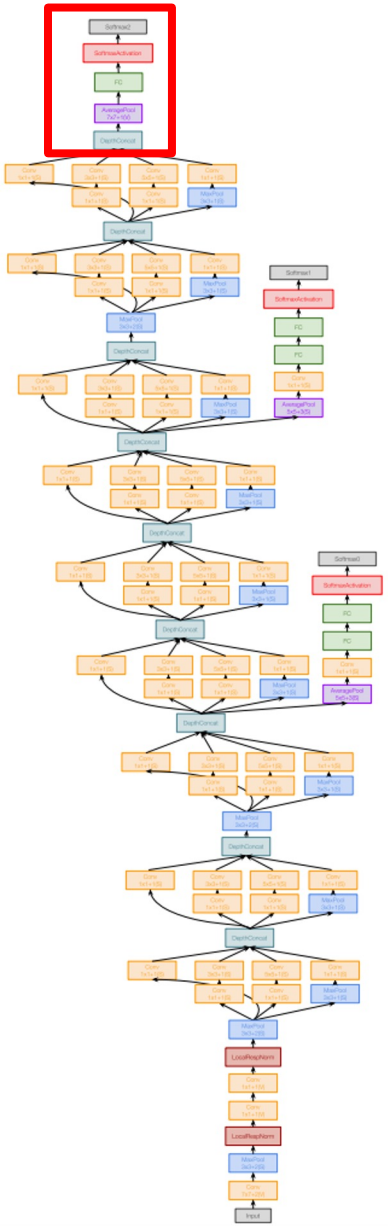
GoogLeNet: Global average pooling

GoogLeNet head:

	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:

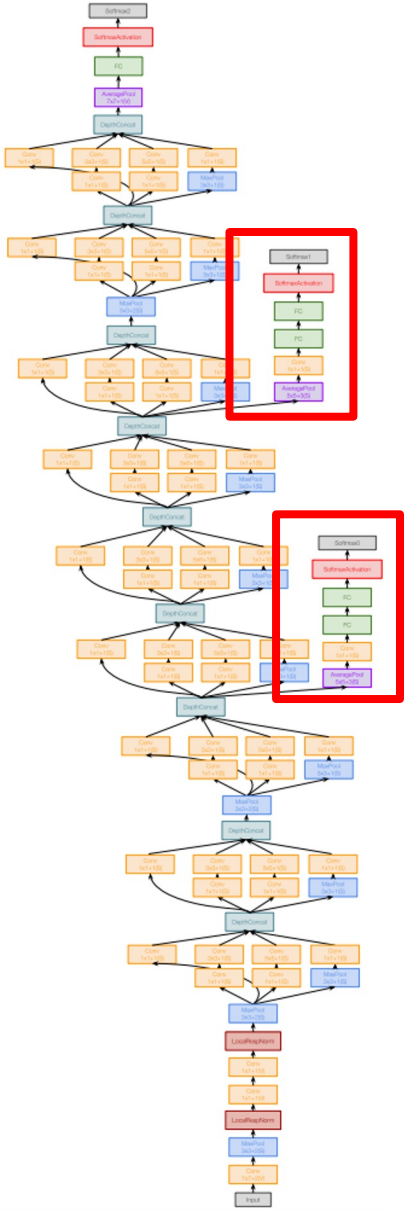
Layer	C	H/W	filters	kernel	stride	pad	C	H/W	memory (KB)	params (k)	flop (M)
flatten	512	7					25088		98		
fc6	25088			4096			4096		16	102760	103
fc7	4096			4096			4096		16	16777	17
fc8	4096			1000			1000		4	4096	4



Source: J. Johnson

GoogLeNet: Auxiliary classifiers

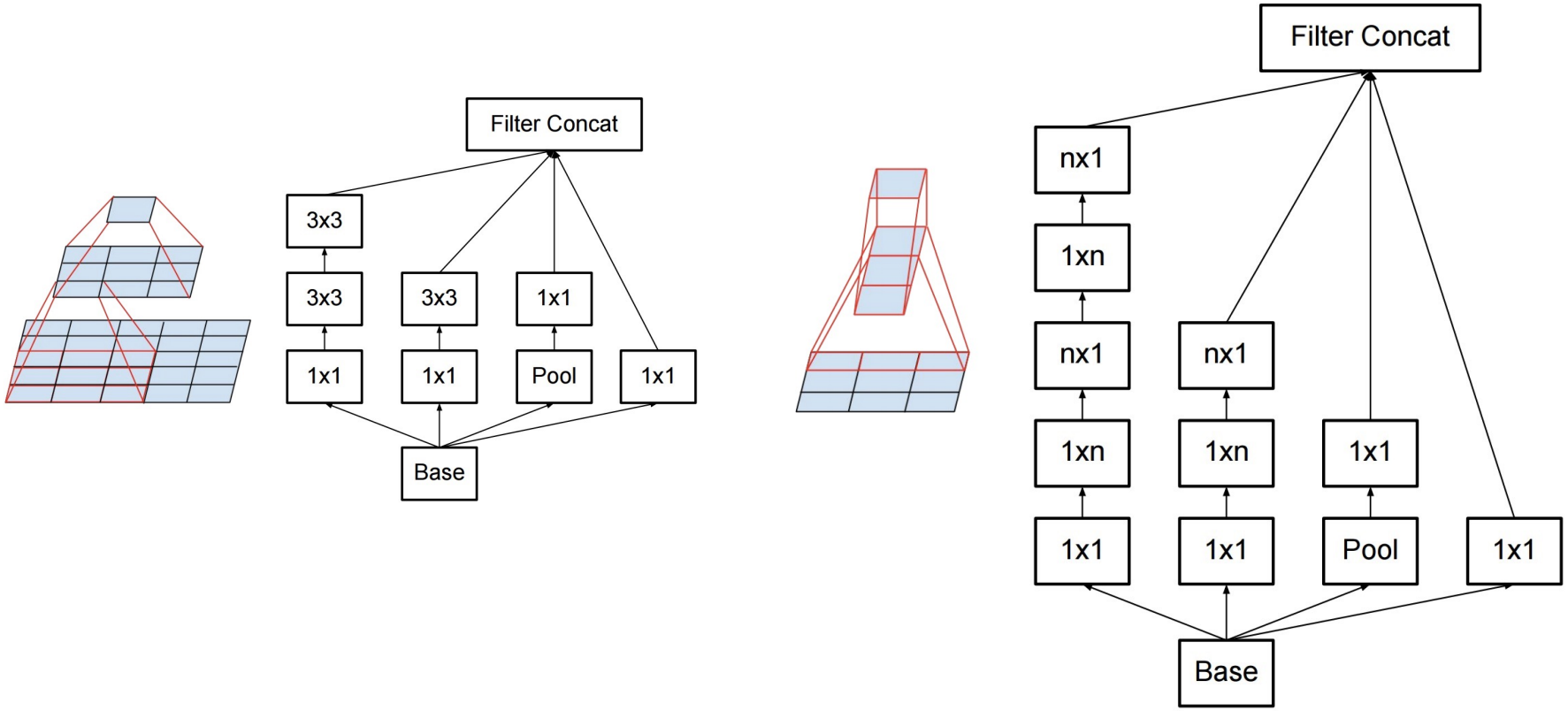
- Training using loss at the end of the network didn't work well: network is too deep, gradients don't propagate cleanly
- As a hack, attach "auxiliary classifiers" at several intermediate points in the network that also try to classify the image and receive loss
- GoogLeNet was before batch normalization! With BatchNorm no longer need to use this trick



Source: J. Johnson

Inception v2, v3

- Improve training with batch normalization, eliminating need for auxiliary classifiers
- More variants of inception modules with aggressive factorization of filters



C. Szegedy et al., Rethinking the inception architecture for computer vision, CVPR 2016

ImageNet Challenge 2012-2014

Team	Year	Place	Error (top-5)	External data
SuperVision – Toronto (7 layers)	2012	-	16.4%	no
SuperVision	2012	1st	15.3%	ImageNet 22k
Clarifai – NYU (7 layers)	2013	-	11.7%	no
Clarifai	2013	1st	11.2%	ImageNet 22k
VGG – Oxford (16 layers)	2014	2nd	7.32%	no
GoogLeNet (19 layers)	2014	1st	6.67%	no
Human expert*			5.1%	

http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/

Convolutional networks: Outline

- Building blocks
  - Convolutional layers
  - Pooling layers and nonlinearities
- Architectures:
  - 1st generation (2012-2013): AlexNet
  - 2nd generation (2014): VGGNet, GoogLeNet
  - 3rd generation (2015): ResNet
  - 4th generation (2016): ResNeXt, DenseNet

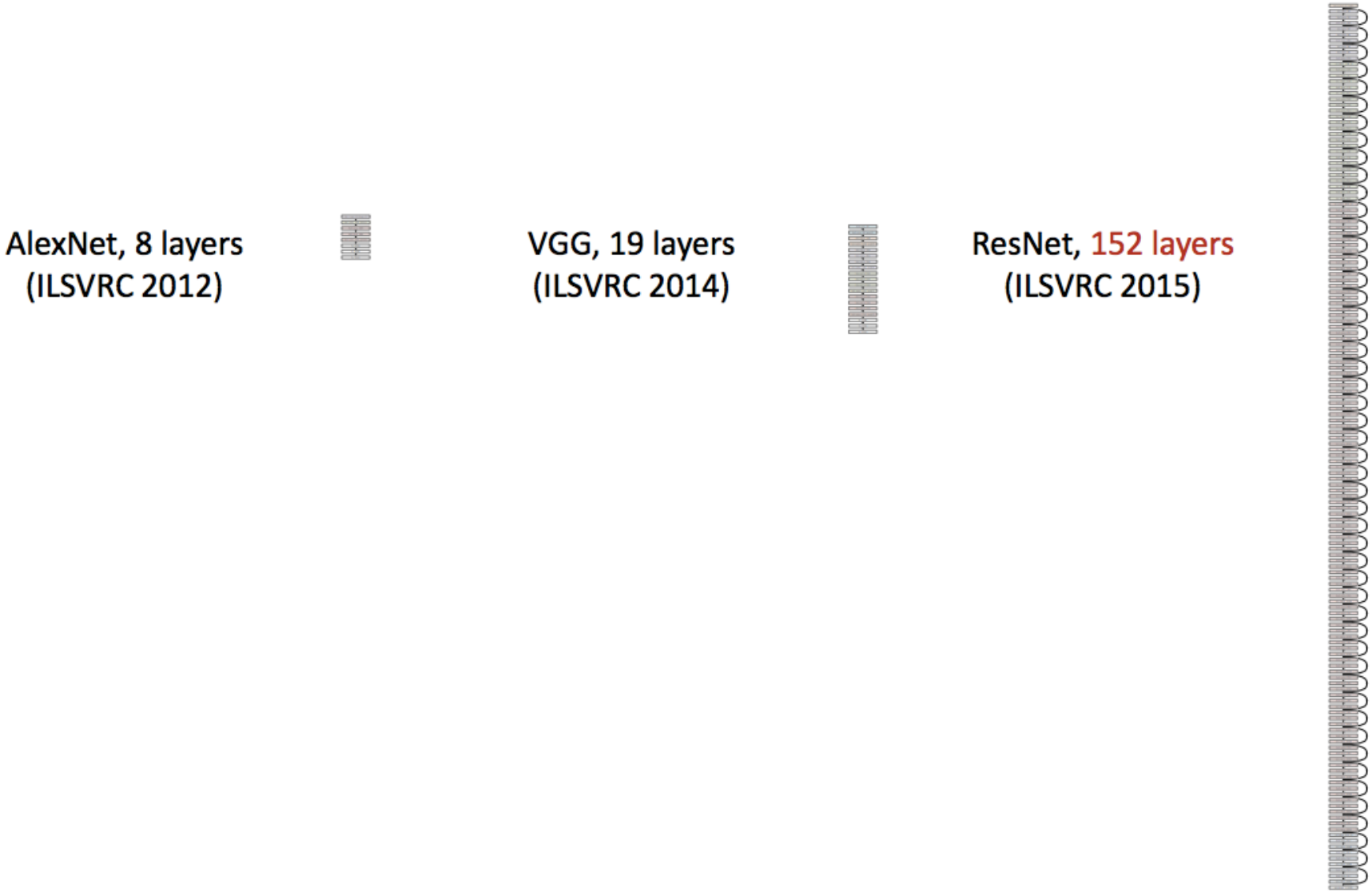
ResNet: ILSVRC 2015 winner

AlexNet, 8 layers
(ILSVRC 2012)

VGG, 19 layers
(ILSVRC 2014)

K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)

ResNet: ILSVRC 2015 winner



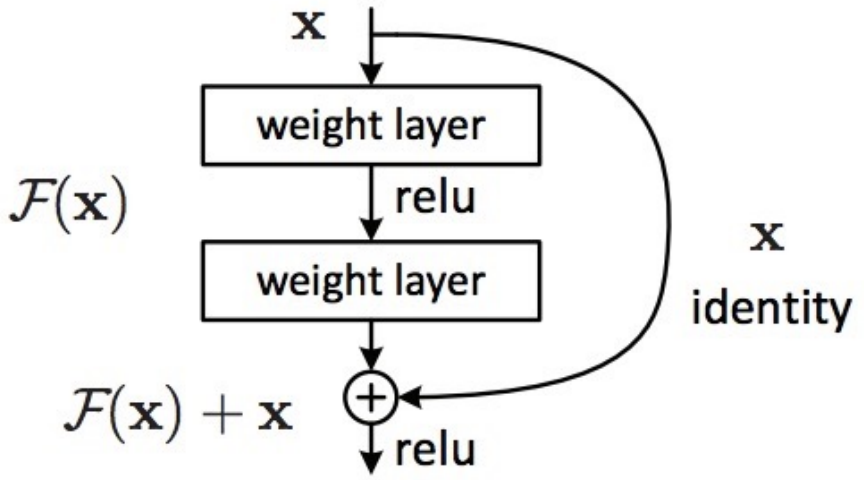
K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)


I WAS WINNING
IMAGENET
UNTIL A
DEEPER MODEL
CAME ALONG
imgflip.com

Source (?)

ResNet

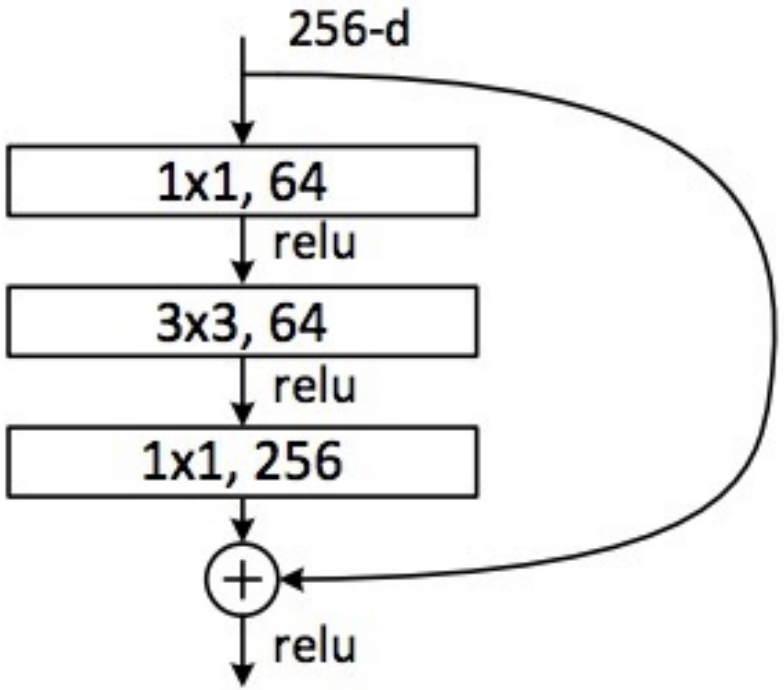
- The residual module
  - Introduce *skip* or *shortcut* connections (existing before in various forms in literature)
  - Make it easy for network layers to represent the identity mapping



K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)

ResNet

Deeper residual module (bottleneck)



- Directly performing 3x3 convolutions with 256 feature maps at input and output:
  256 x 256 x 3 x 3 ~ 600K operations
- Using 1x1 convolutions to reduce 256 to 64 feature maps, followed by 3x3 convolutions, followed by 1x1 convolutions to expand back to 256 maps:
  256 x 64 x 1 x 1 ~ 16K
  64 x 64 x 3 x 3 ~ 36K
  64 x 256 x 1 x 1 ~ 16K
  Total: ~70K

K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)

ResNet

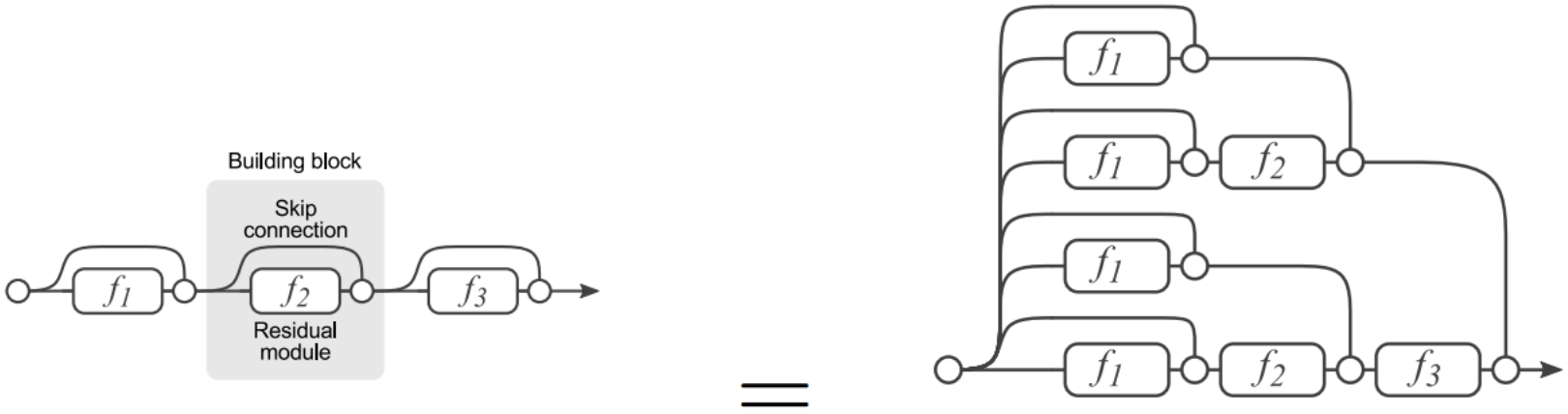
- Architectures for ImageNet:

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)

Why do ResNets work?

- ResNets are collections of many paths of different length, and shorter paths predominantly contribute to training



(a) Conventional 3-block residual network

(b) Unraveled view of (a)

Figure 1: Residual Networks are conventionally shown as (a), which is a natural representation of Equation (1). When we expand this formulation to Equation (6), we obtain an *unraveled view* of a 3-block residual network (b). Circular nodes represent additions. From this view, it is apparent that residual networks have $O(2^n)$ implicit paths connecting input and output and that adding a block doubles the number of paths.

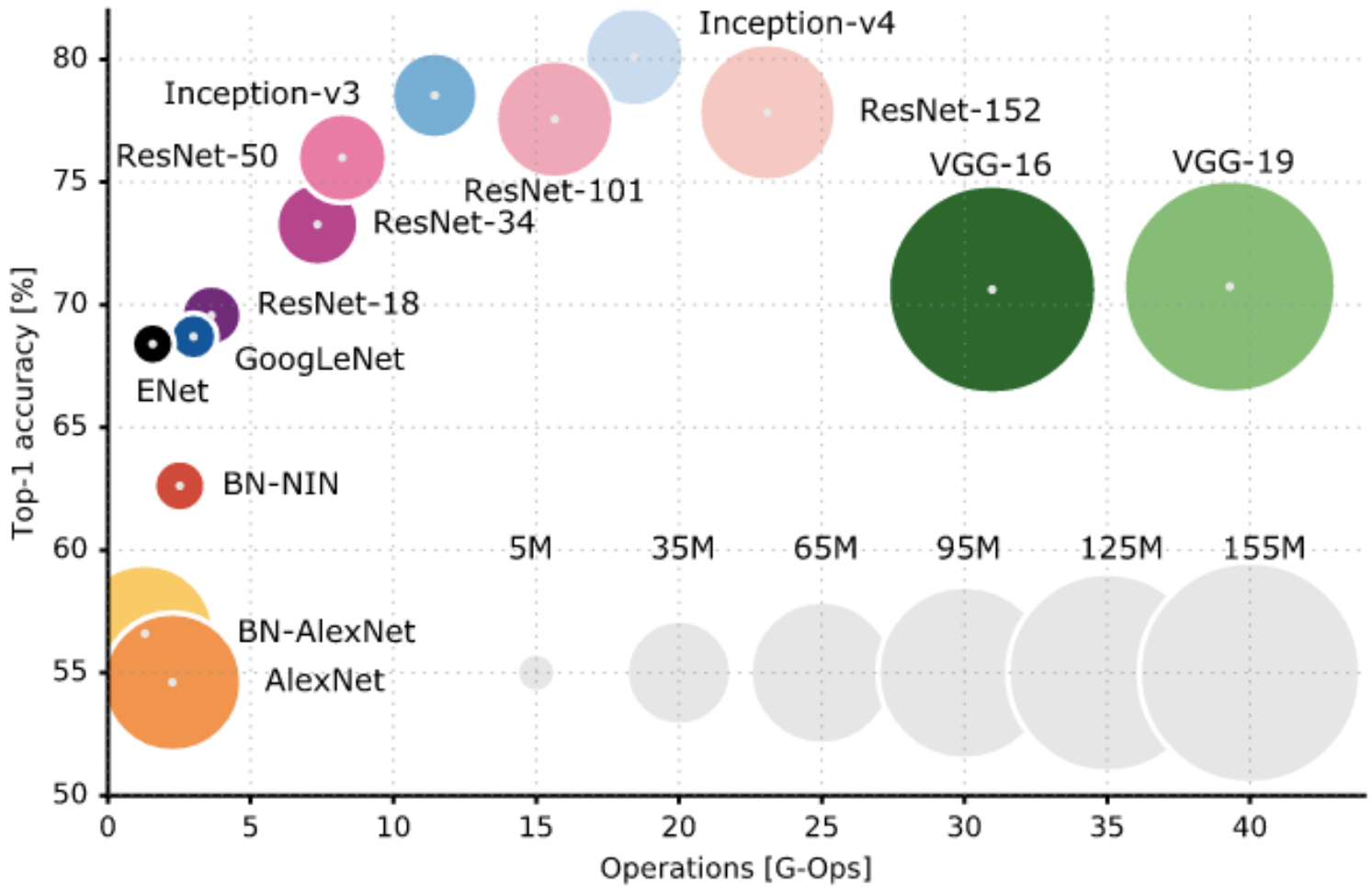
A. Veit, M. Wilber, S. Belongie, Residual Networks Behave Like Ensembles of Relatively Shallow Networks, NIPS 2016

Summary: ILSVRC 2012-2015

Team	Year	Place	Error (top-5)	External data
SuperVision – Toronto (AlexNet, 7 layers)	2012	-	16.4%	no
SuperVision	2012	1st	15.3%	ImageNet 22k
Clarifai – NYU (7 layers)	2013	-	11.7%	no
Clarifai	2013	1st	11.2%	ImageNet 22k
VGG – Oxford (16 layers)	2014	2nd	7.32%	no
GoogLeNet (19 layers)	2014	1st	6.67%	no
ResNet (152 layers)	2015	1st*	3.57%	

*Officially, there was no longer a classification competition and ResNet-based systems won in localization and detection

Comparing architectures



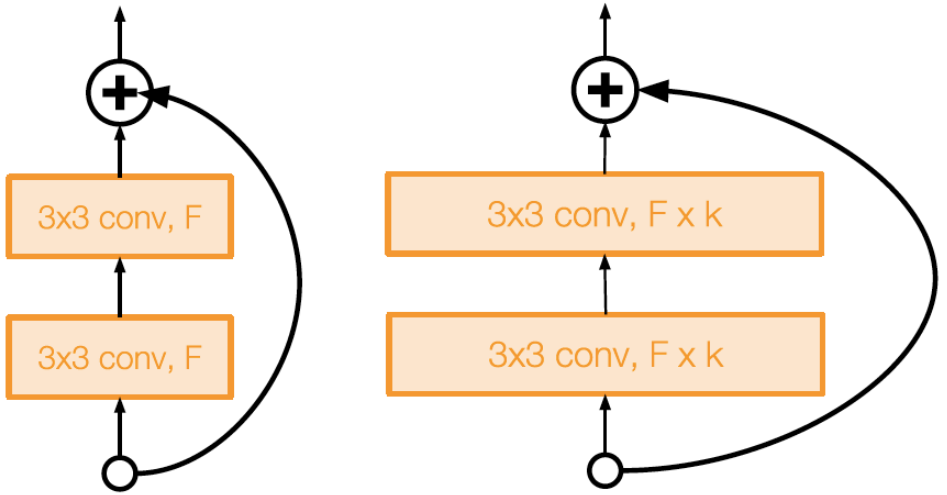
https://culurciello.github.io/tech/2016/06/04/nets.html

Outline

- Building blocks
  - Convolutional layers
  - Pooling layers and nonlinearities
- Architectures:
  - 1st generation (2012-2013): AlexNet
  - 2nd generation (2014): VGGNet, GoogLeNet
  - 3rd generation (2015): ResNet
  - 4th generation (2016): Wide ResNet, ResNeXt, DenseNet

Beyond ResNet: Wide ResNet

- Reduce number of residual blocks, but increase number of feature maps in each block
  - More parallelizable, better feature reuse
  - 16-layer WRN outperforms 1000-layer ResNets, though with much larger # of parameters



Basic residual block Wide residual block

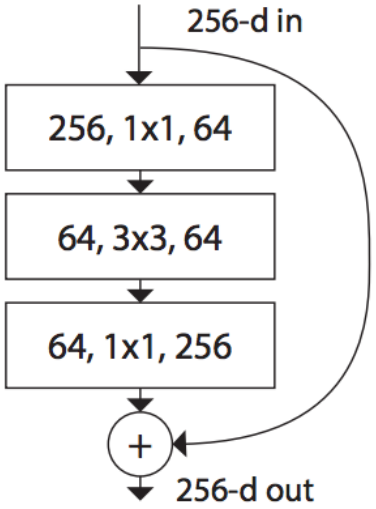
Image source

S. Zagoryuko and N. Komodakis, Wide Residual Networks, BMVC 2016

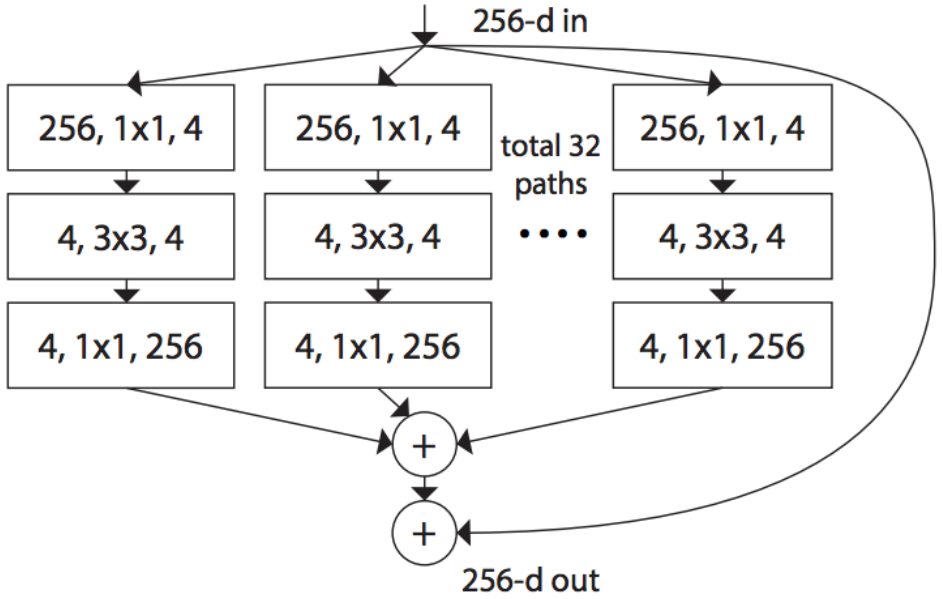
Beyond ResNet: ResNeXt

- Propose “cardinality” as a new factor in network design, apart from depth and width
- Claim that increasing cardinality is a better way to increase capacity than increasing depth or width

ResNet



ResNeXt, roughly same complexity



S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, Aggregated Residual Transformations for Deep Neural Networks, CVPR 2017

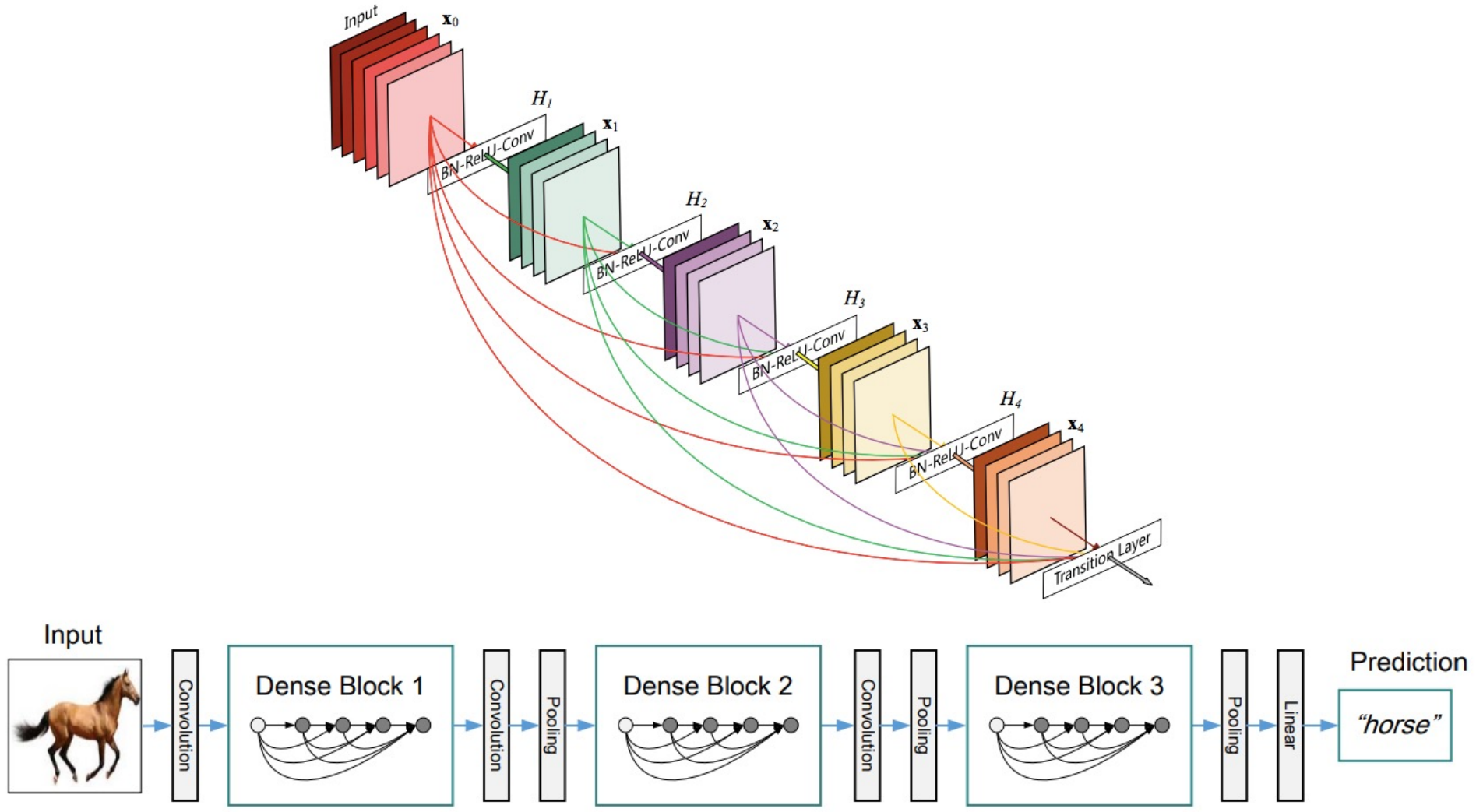
Beyond ResNet: ResNeXt

	setting	top-1 error (%)
ResNet-50	1 × 64d	23.9
ResNeXt-50	2 × 40d	23.0
ResNeXt-50	4 × 24d	22.6
ResNeXt-50	8 × 14d	22.3
ResNeXt-50	32 × 4d	**22.2**
ResNet-101	1 × 64d	22.0
ResNeXt-101	2 × 40d	21.7
ResNeXt-101	4 × 24d	21.4
ResNeXt-101	8 × 14d	21.3
ResNeXt-101	32 × 4d	**21.2**

Table 3. Ablation experiments on ImageNet-1K. (**Top**): ResNet-50 with preserved complexity (∼4.1 billion FLOPs); (**Bottom**): ResNet-101 with preserved complexity (∼7.8 billion FLOPs). The error rate is evaluated on the single crop of 224×224 pixels.

S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, Aggregated Residual Transformations for Deep Neural Networks, CVPR 2017

Beyond ResNet: DenseNets



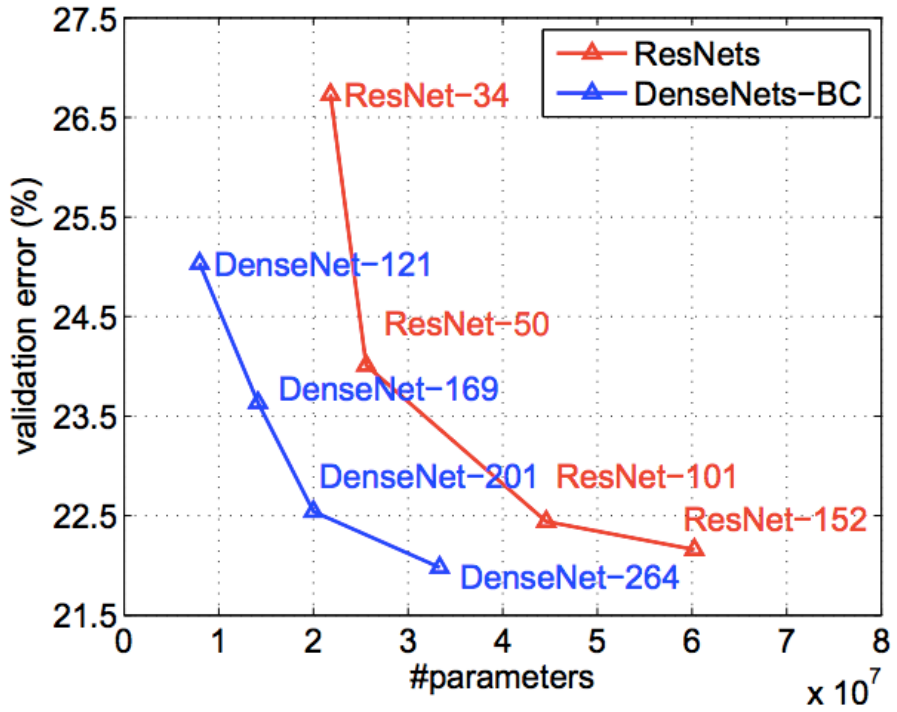
G. Huang, Z. Liu, and L. van der Maaten, Densely Connected Convolutional Networks, CVPR 2017 (Best Paper Award)

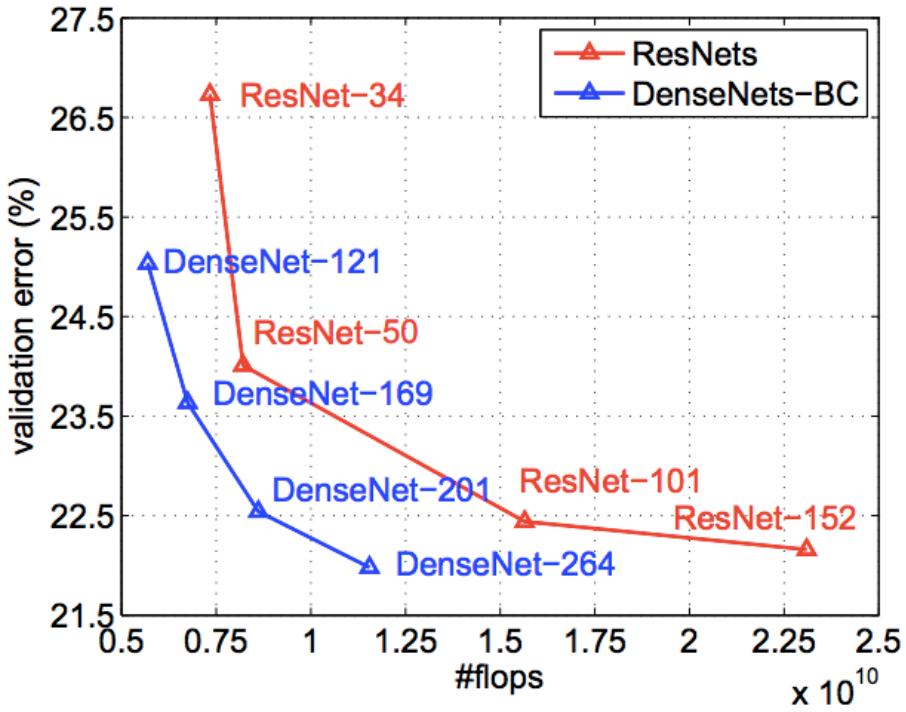
DenseNets

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

G. Huang, Z. Liu, and L. van der Maaten, Densely Connected Convolutional Networks, CVPR 2017 (Best Paper Award)

DenseNets





ImageNet validation error vs. number of parameters and test-time operations

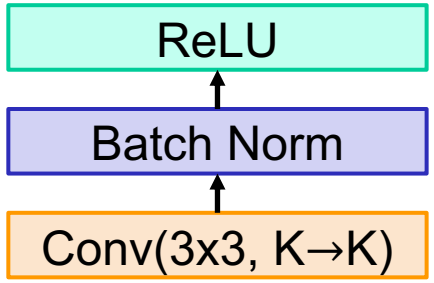
G. Huang, Z. Liu, and L. van der Maaten, Densely Connected Convolutional Networks, CVPR 2017 (Best Paper Award)

Outline

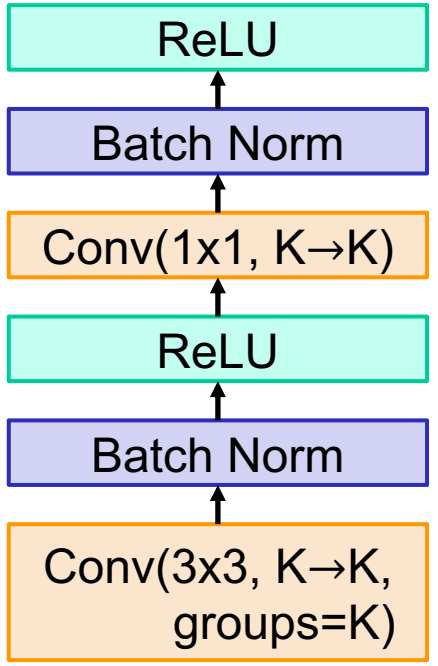
- Building blocks
  - Convolutional layers
  - Pooling layers and nonlinearities
- Architectures:
  - 1st generation (2012-2013): AlexNet
  - 2nd generation (2014): VGGNet, GoogLeNet
  - 3rd generation (2015): ResNet
  - 4th generation (2016): Wide ResNet, ResNeXt, DenseNet
  - MobileNets

MobileNets: Tiny networks (for mobile devices)

Standard Convolution Block



Depthwise Separable Convolution



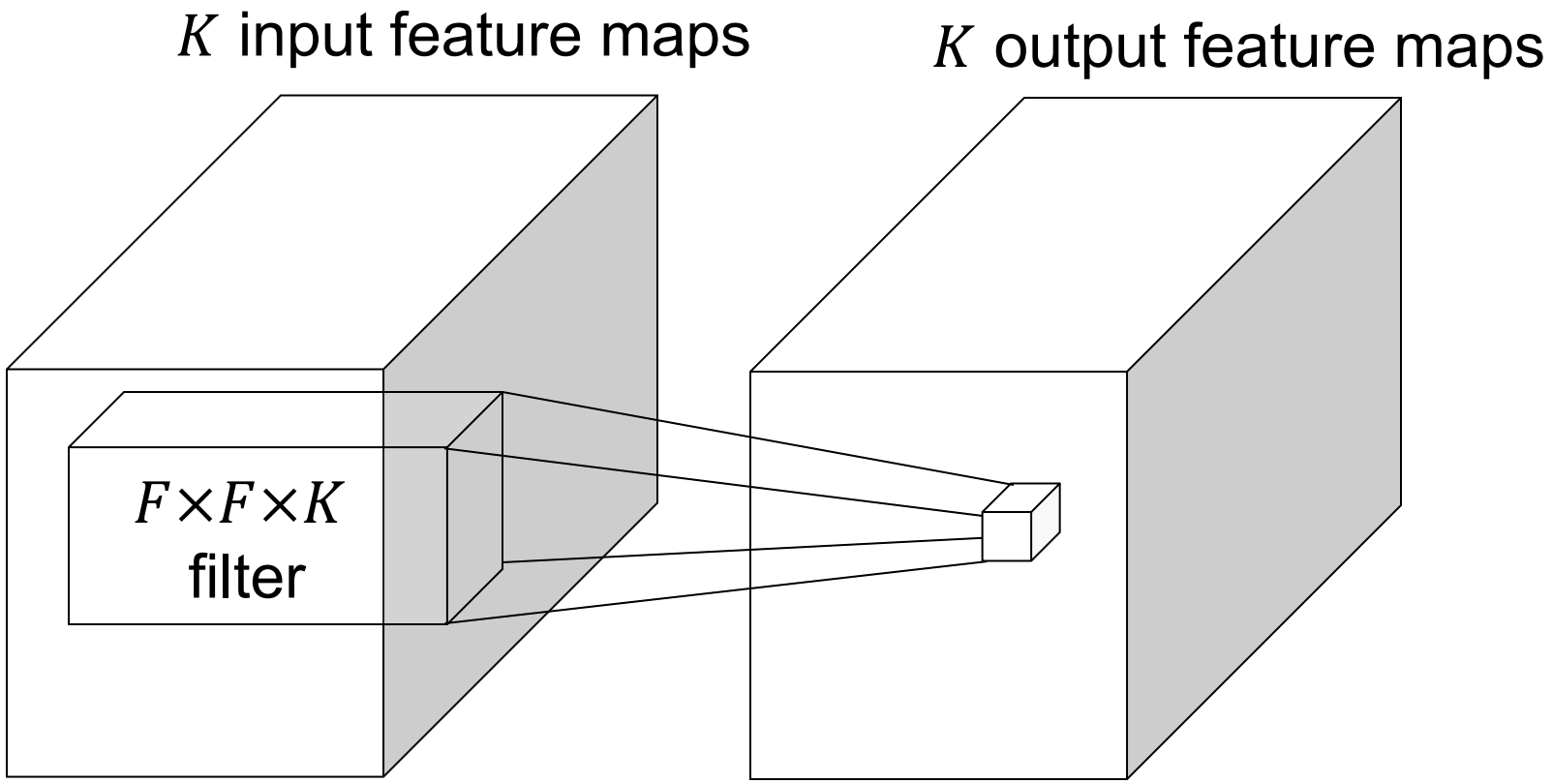
"Pointwise Convolution"

"Depthwise Convolution"

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

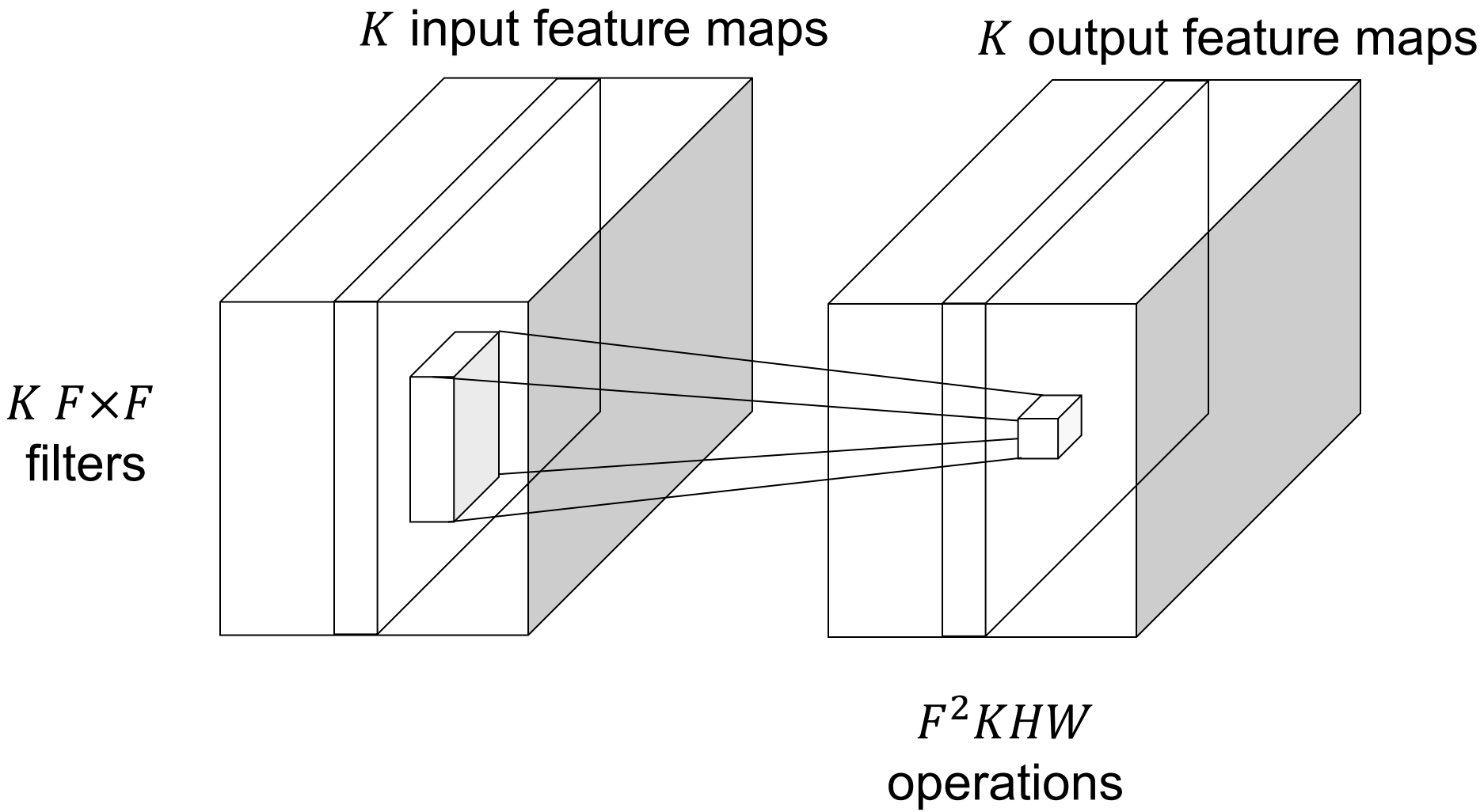
Source: J. Johnson

Recall: normal convolutional layer



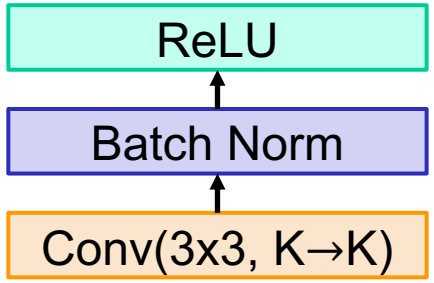
F^2K^2HW
operations

Depthwise convolution



MobileNets: Tiny networks (for mobile devices)

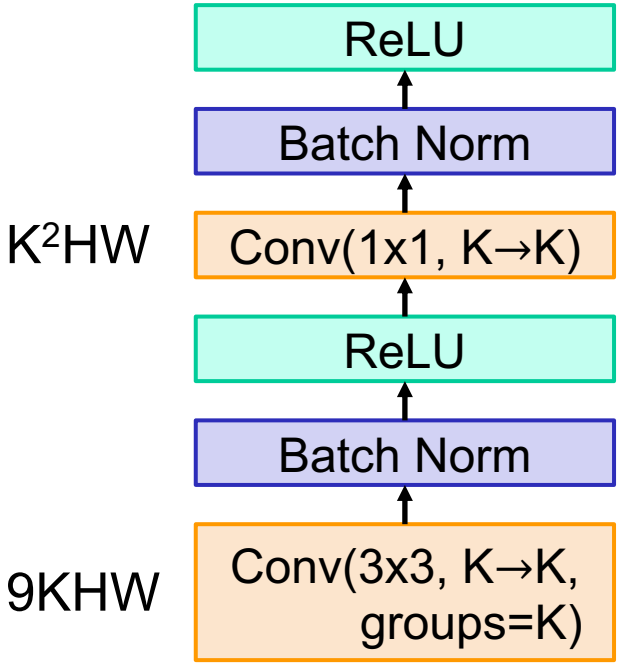
Standard Convolution Block
Total cost: $9K^2HW$



$9K^2HW$

Speedup = $9K^2/(9K+K^2)$
= $9K/(9+K)$
=> 9 (as K → inf)

Depthwise Separable Convolution
Total cost: $(9K + K^2)HW$



K^2HW "Pointwise Convolution"

$9KHW$ "Depthwise Convolution"

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

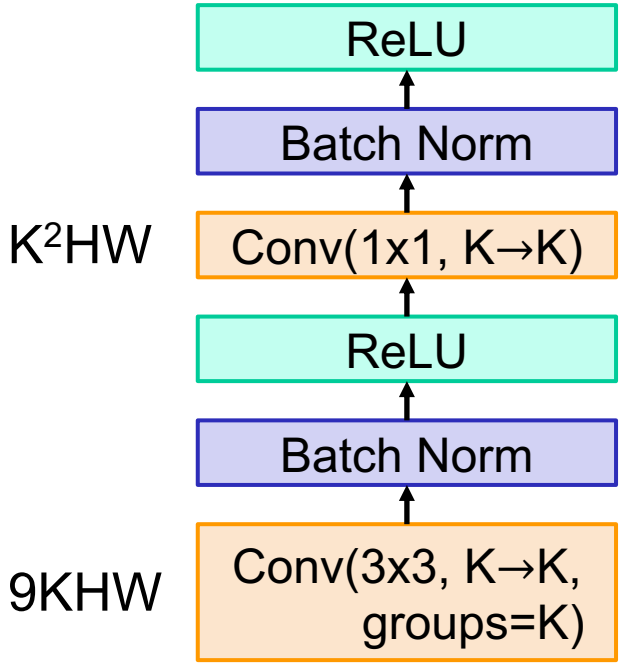
Source: J. Johnson

MobileNets: Tiny networks (for mobile devices)

See also:

SqueezeNet: Iandola et al., 2016
ShuffleNet: Zhang et al., CVPR 2018
MobileNetV2: Sandler et al., CVPR 2018
ShuffleNetV2: Ma et al., ECCV 2018
EfficientNet: Tan and Le, ICML 2019

Depthwise Separable Convolution
Total cost: $(9K + K^2)HW$



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

Source: J. Johnson

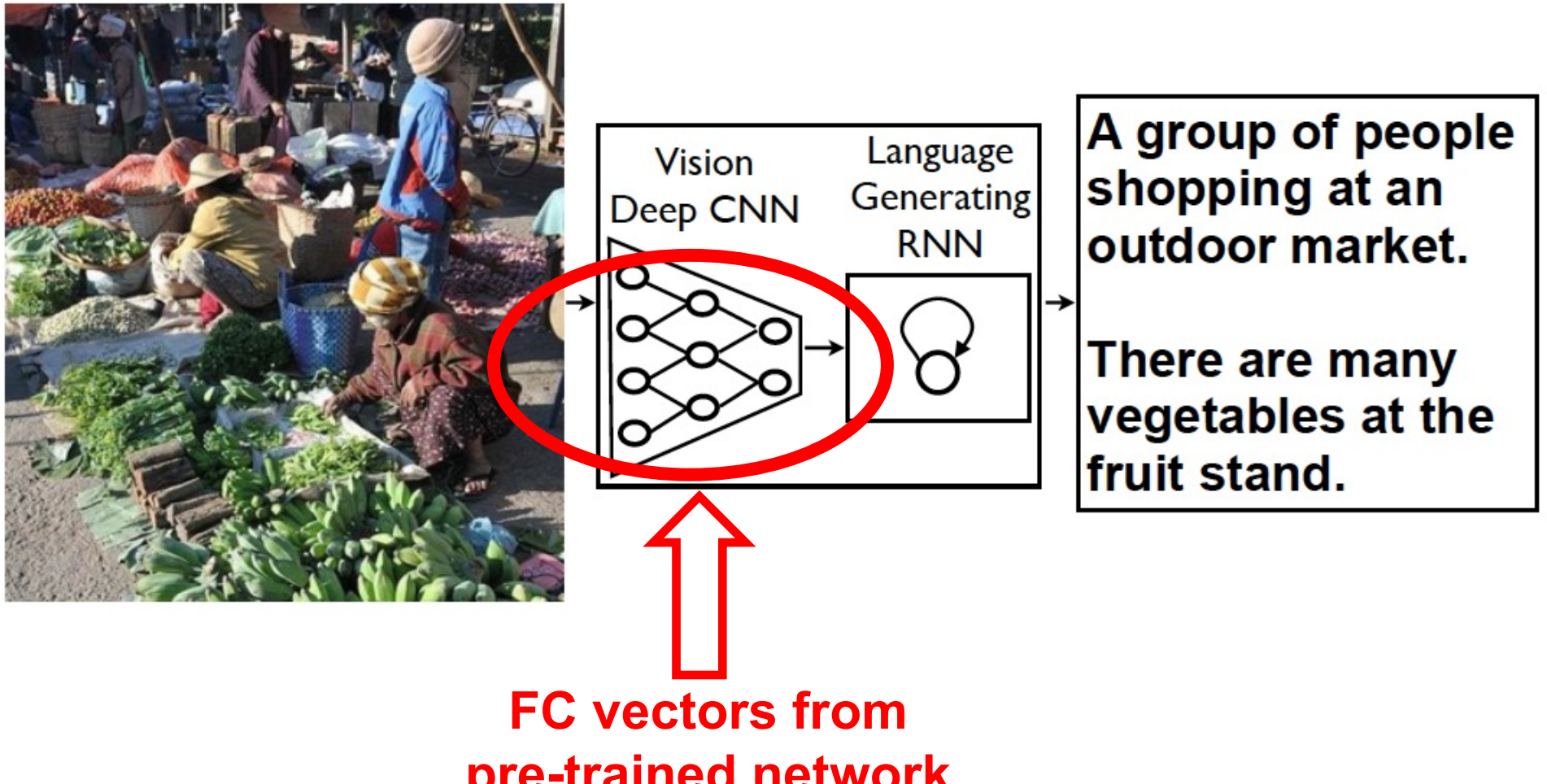
Design principles

- Make networks parameter-efficient
  - Reduce filter sizes, factorize filters
  - Use 1x1 convolutions to reduce number of feature maps before more expensive operations
  - Minimize reliance on FC layers
- Reduce spatial resolution gradually, within each level of resolution replicate a given “block” multiple times
- Use skip connections and/or create multiple redundant paths through the network
- Play around with depth vs. width vs. “cardinality” (or “groups”)

How to use a pre-trained network for a new task?

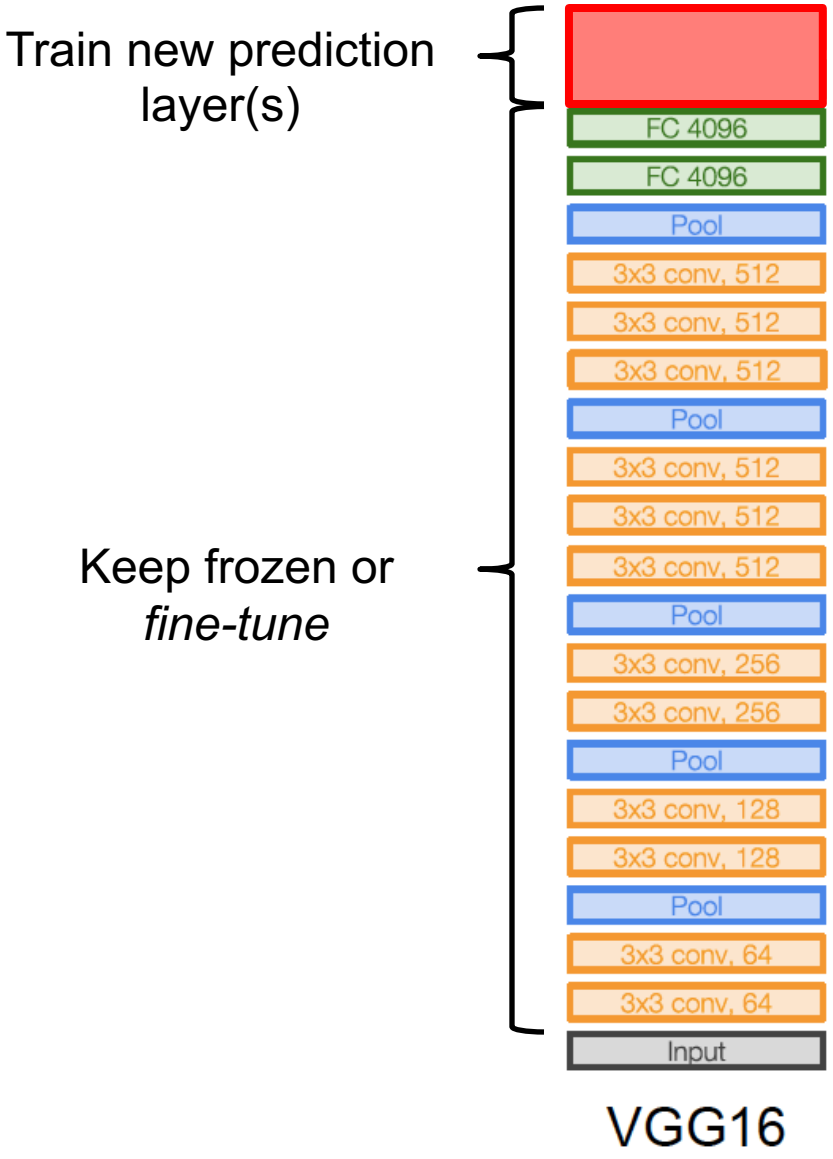
- Strategy 1: Use as feature extractor

Example: CNNs for image captioning



O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image captio n generator. CVPR 2015

How to use a pre-trained network for a new task?



- Strategy 2: Transfer learning