

CENG 506 Deep Learning

Lecture 5 Convolutional Neural Networks

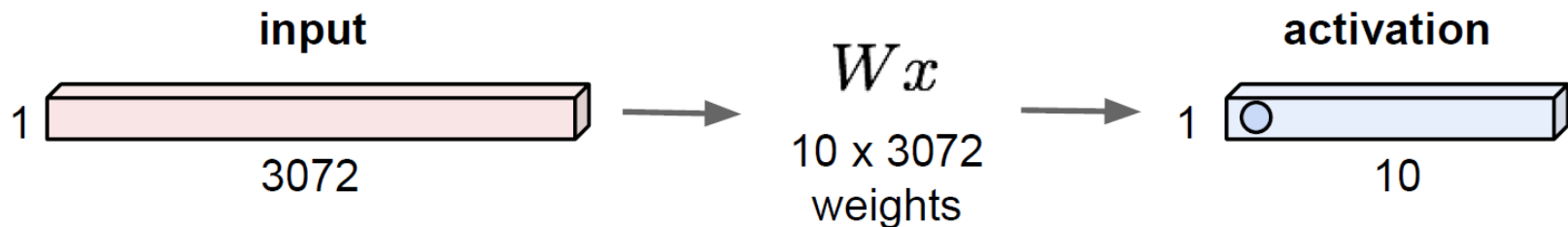
Slides were prepared using the course material of
Stanford's CNN Course (CS231n by Fei-Fei, Johnson, Yeung)

Why not regular NN for images?

- Regular Neural Nets don't scale well to full images.
- CIFAR-10 image size is $32 \times 32 \times 3$, 3072 weights for a single neuron in the first hidden layer.
(If you think about 200×200 images, it is 120,000 weights.)

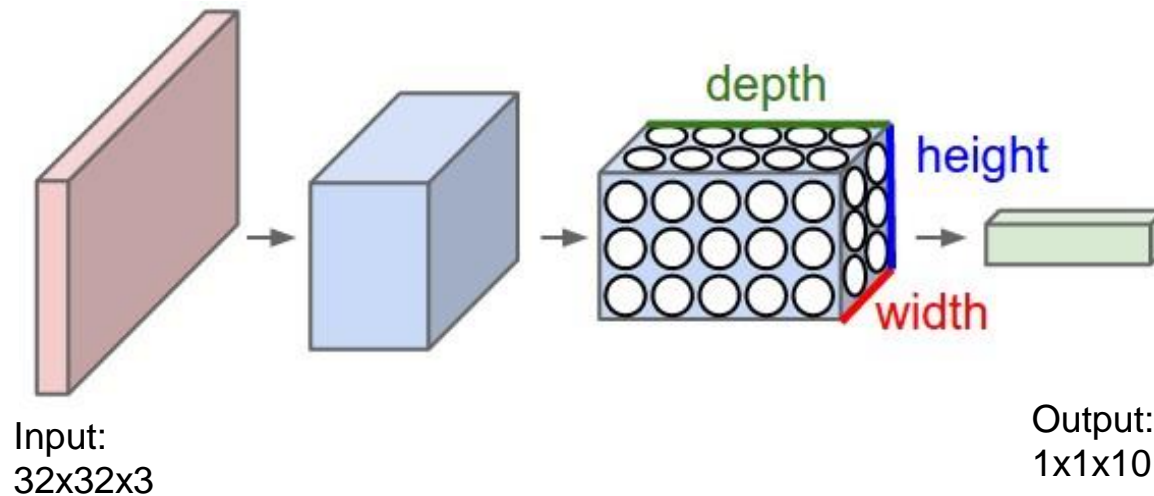
A fully-connected layer:

$32 \times 32 \times 3$ image \rightarrow stretch to 3072×1

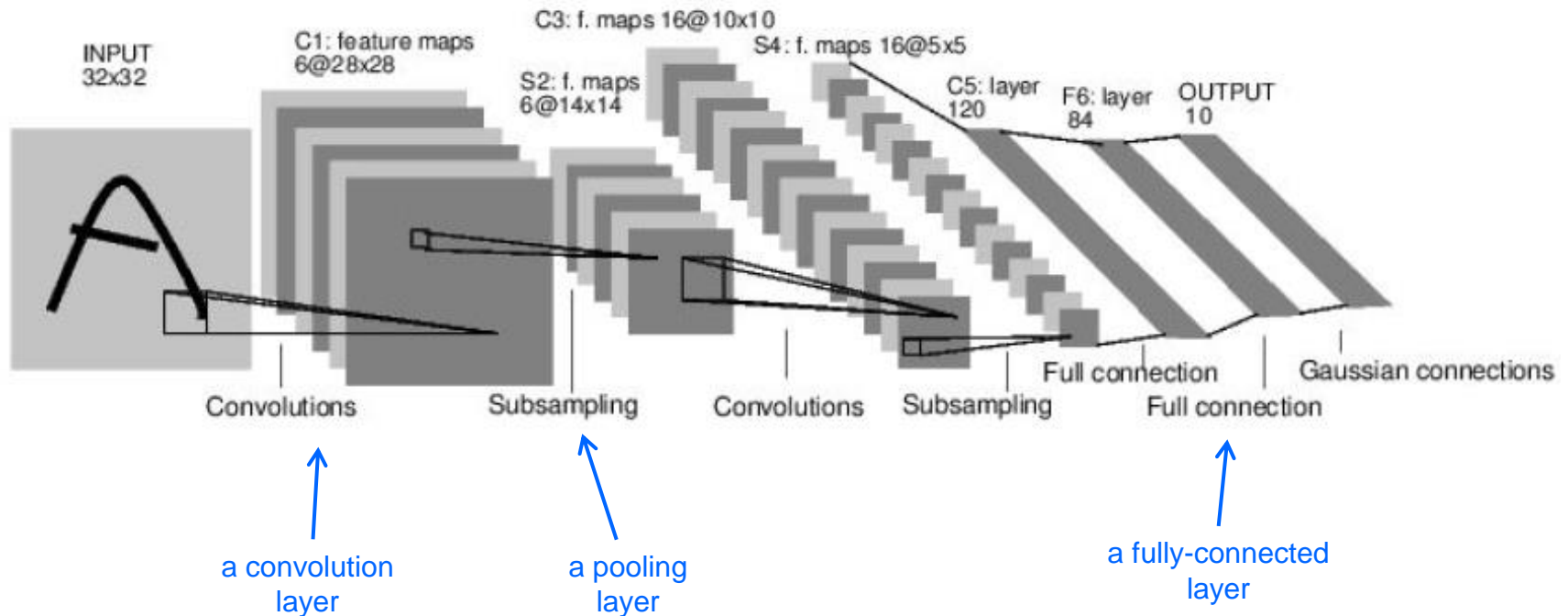


Why not regular NN for images?

- Moreover, we would like to preserve spatial structure.
- Convolutional NN have neurons arranged in three dimensions: width, height, depth.



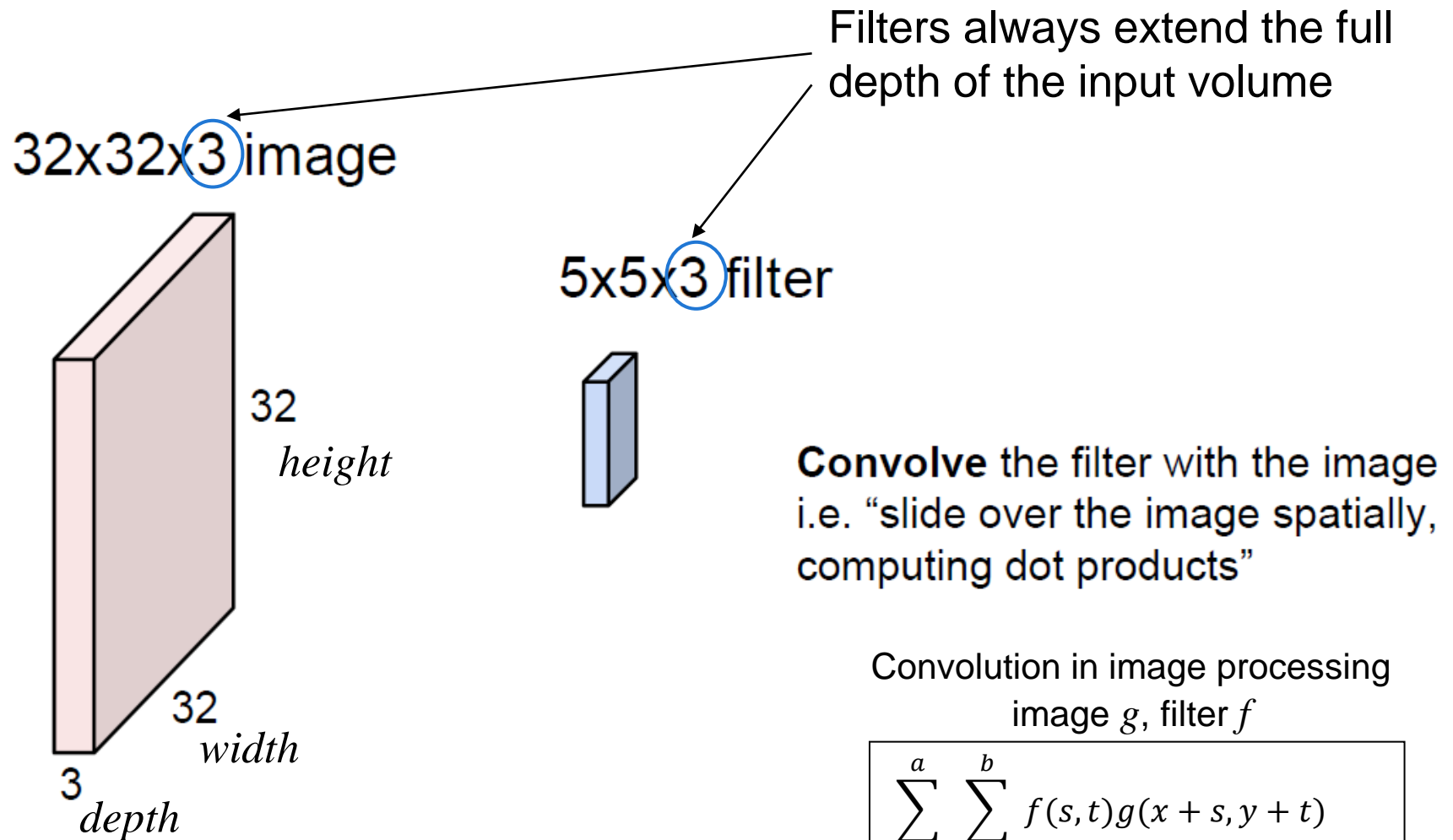
A CNN Architecture



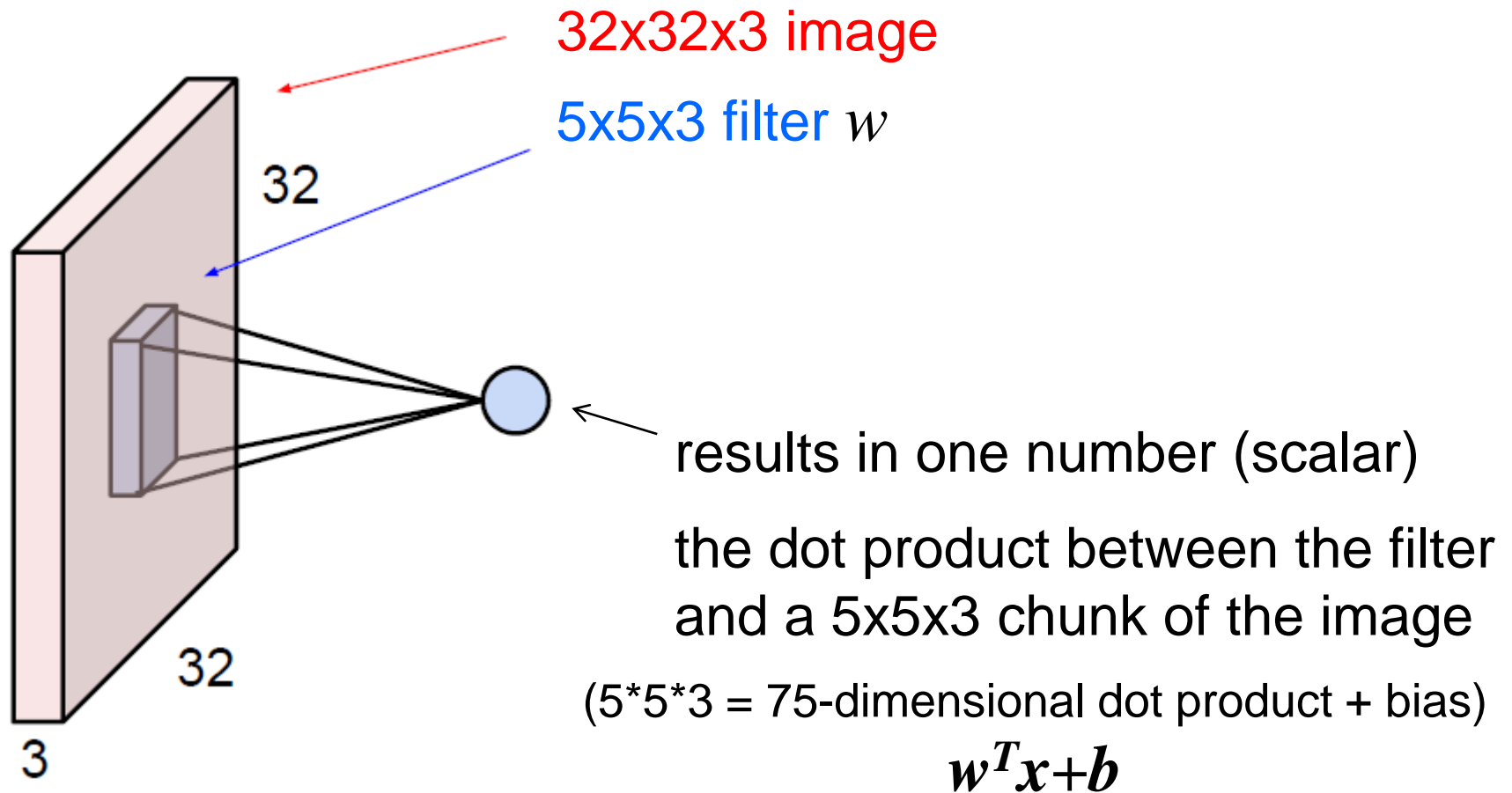
LeNet-5, LeCun 1998

Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: **Gradient-Based Learning Applied to Document Recognition**, *Proceedings of the IEEE*, 86(11):2278-2324, November 1998

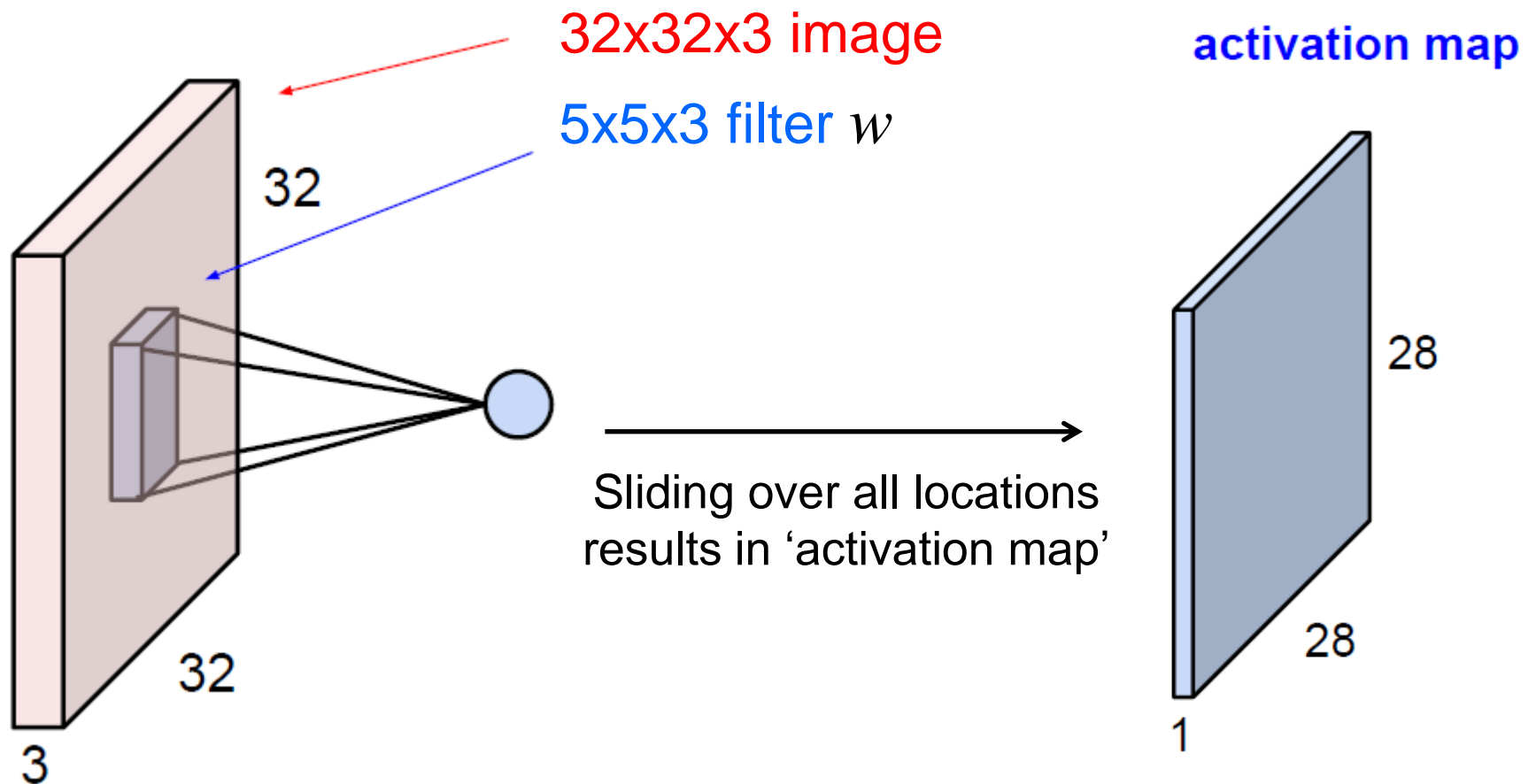
Convolution Layer



Convolution Layer

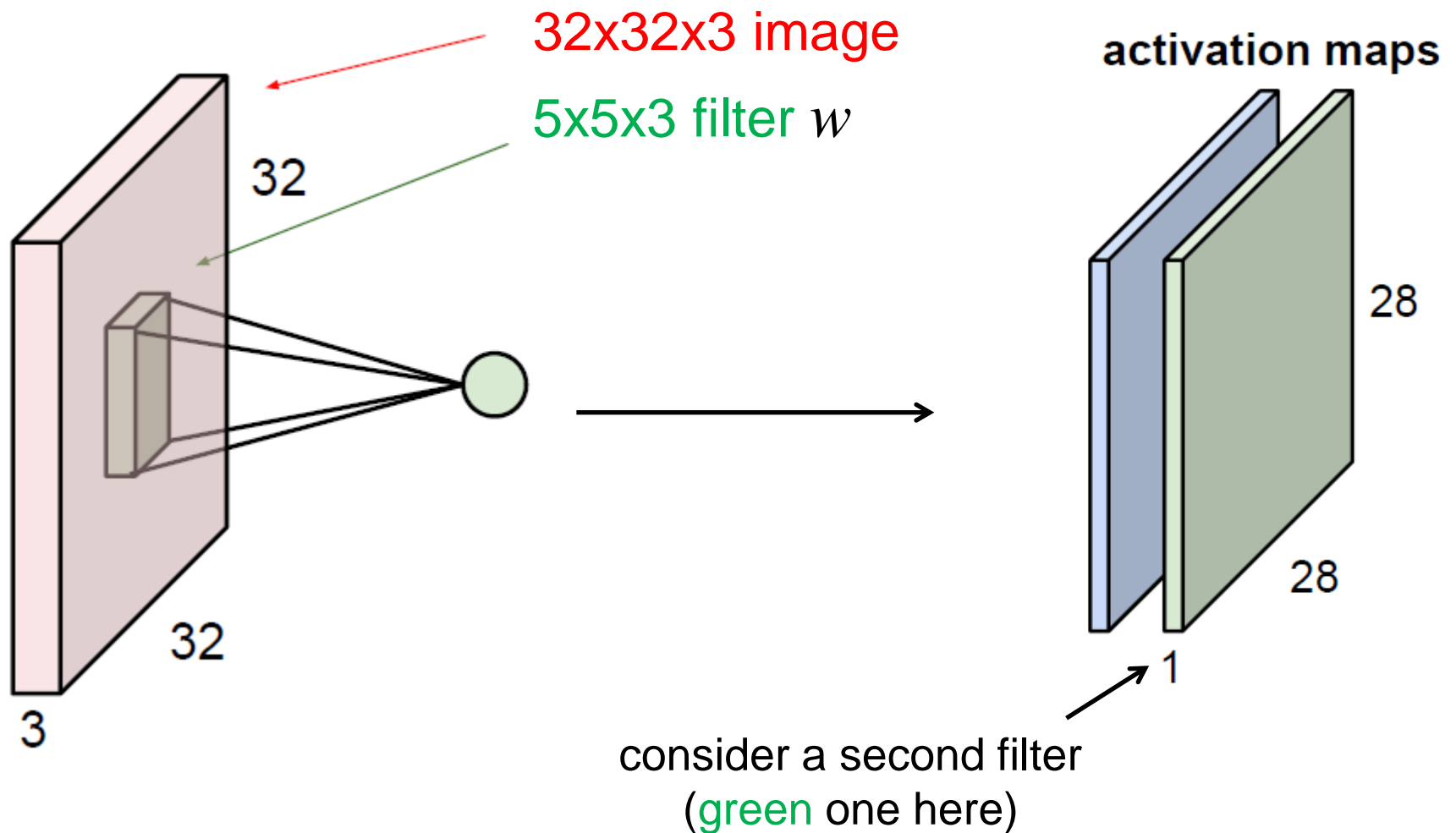


Convolution Layer



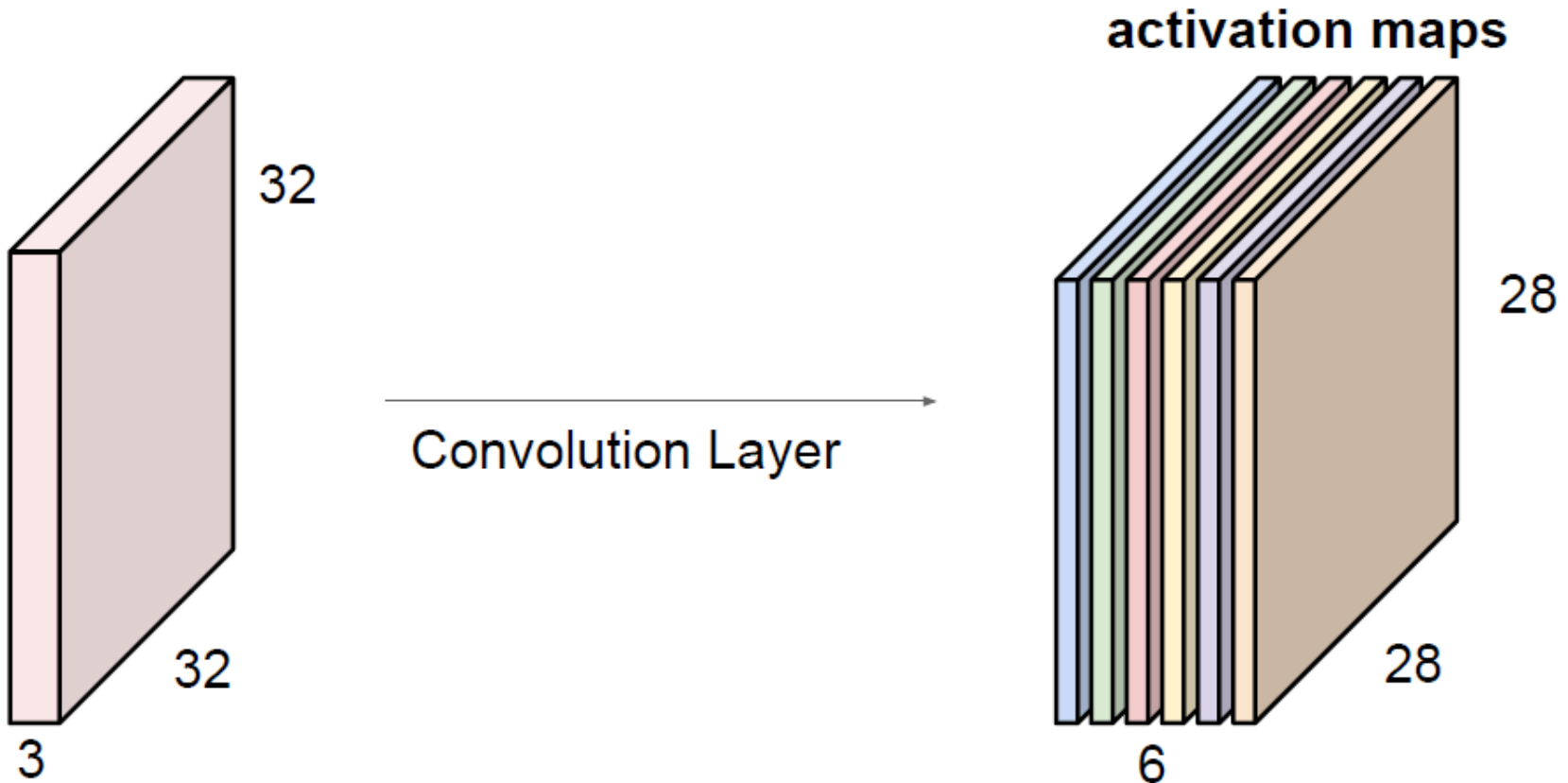
We use 75 weights here
(compare with 30k weights in fully connected case).

Convolution Layer

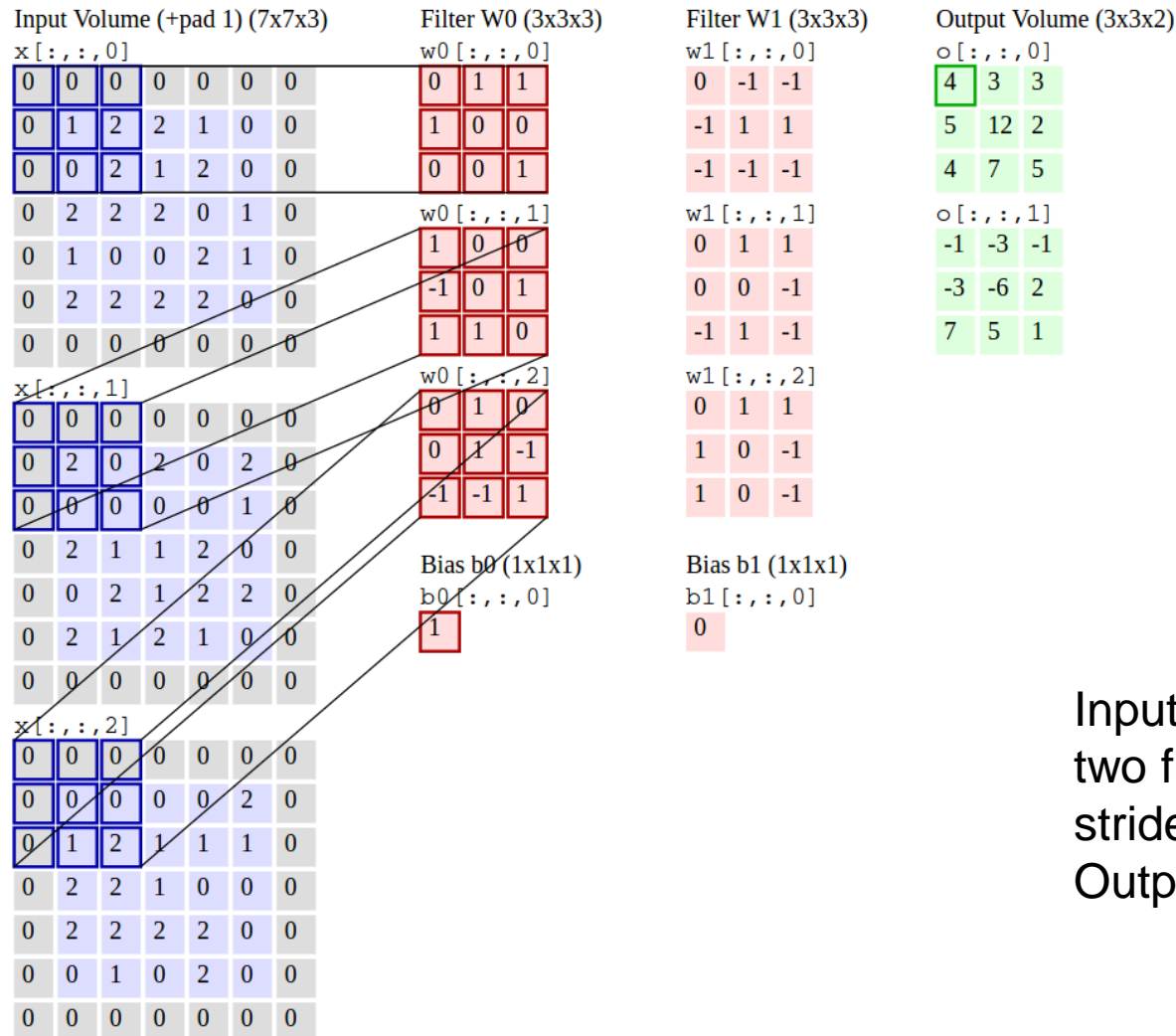


Convolution Layer

If we have 6 5x5 filters, we get 6 separate activation maps. We stack these up to get a “new image” of size 28x28x6!

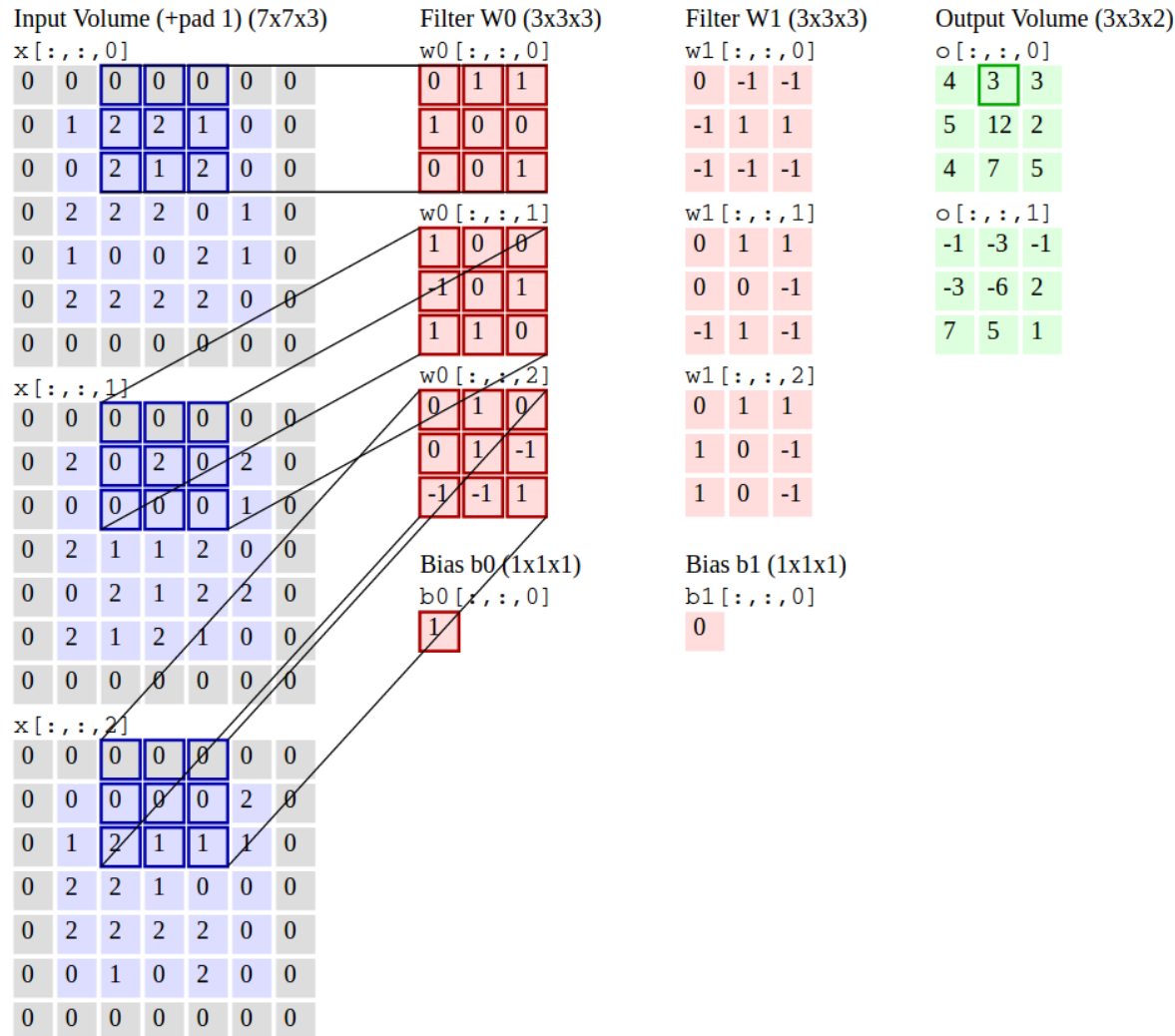


Convolution Demo

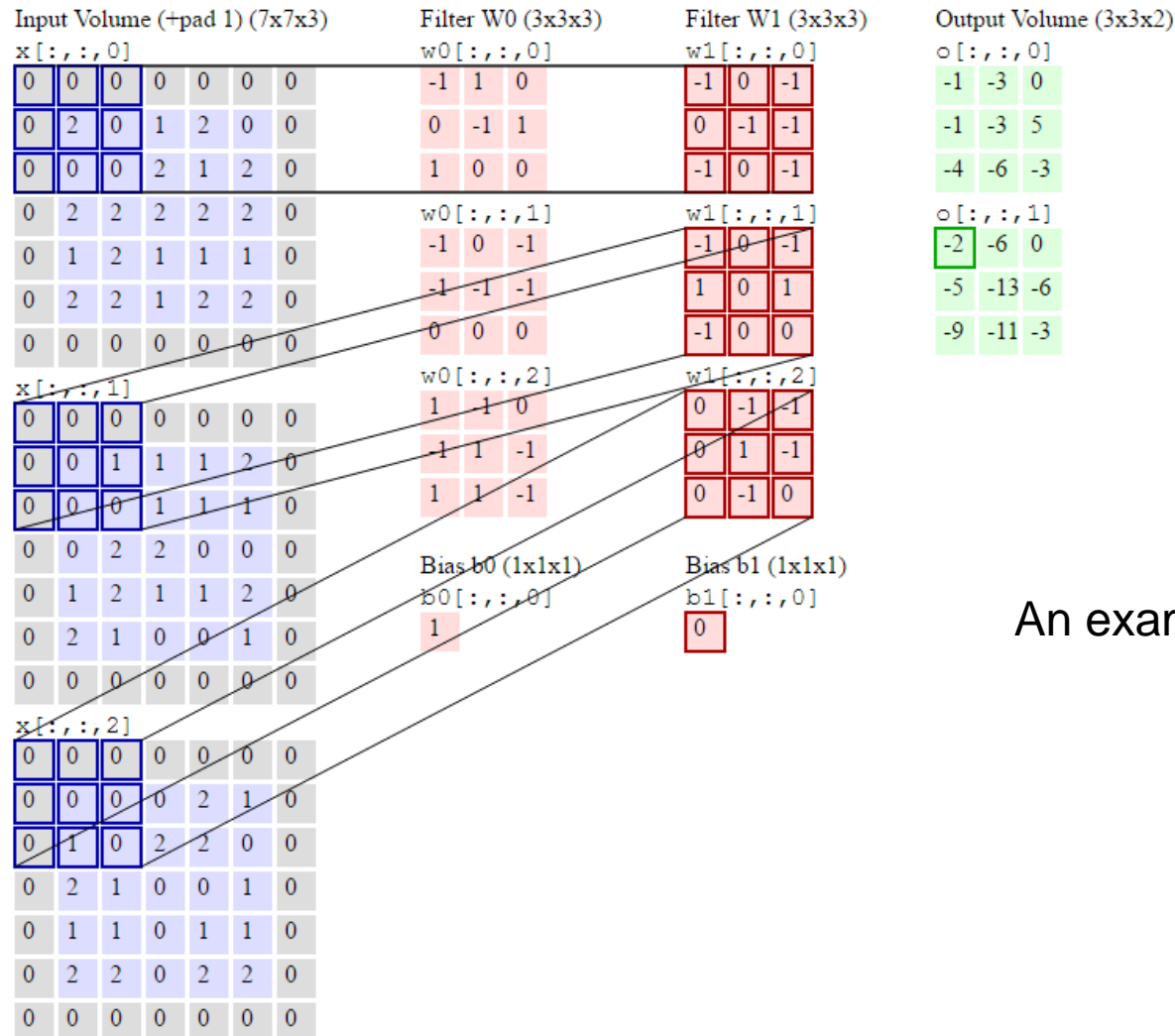


Input volume $w=5$, $h=5$, $d=3$.
 two filters of size $3 \times 3 \times 3$,
 stride of 2, zero padding of 1.
 Output = $(5-3+2)/2 + 1 = 3$.

Convolution Demo

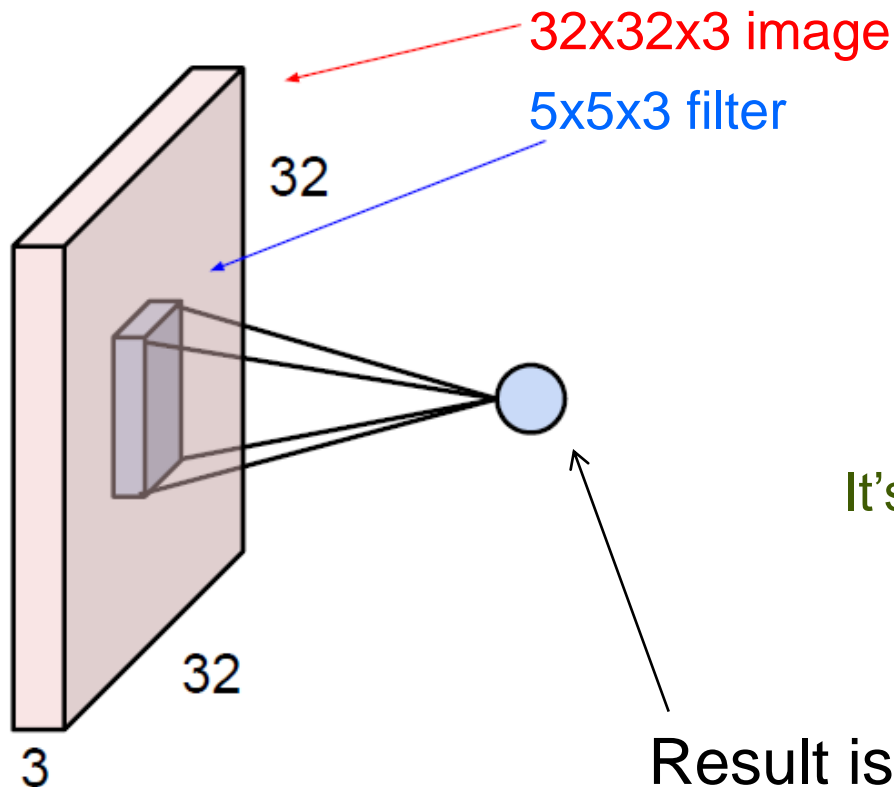


Convolution Demo

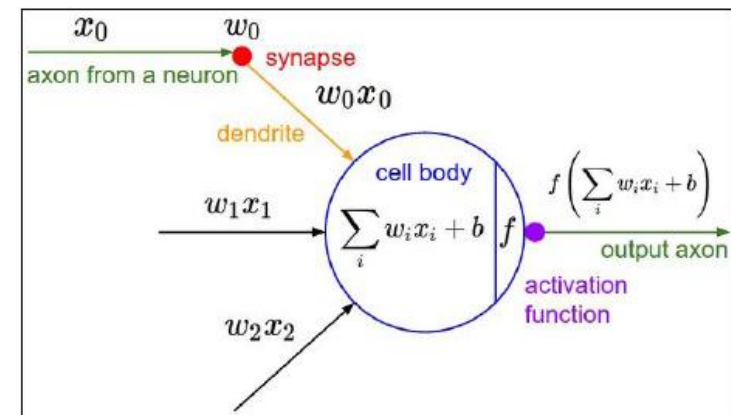


An example for second filter.

The brain/neuron view of CONV Layer

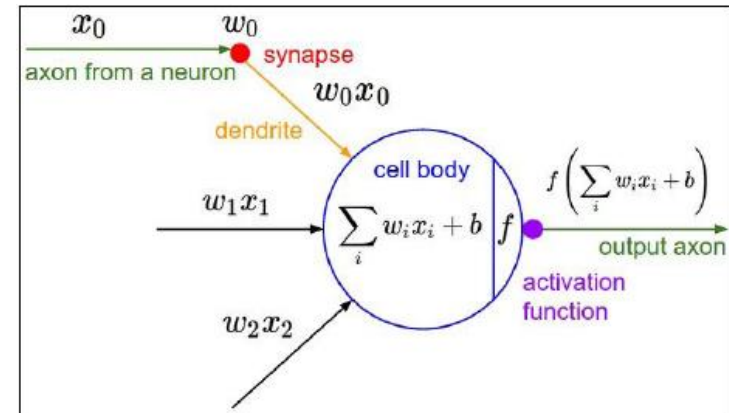
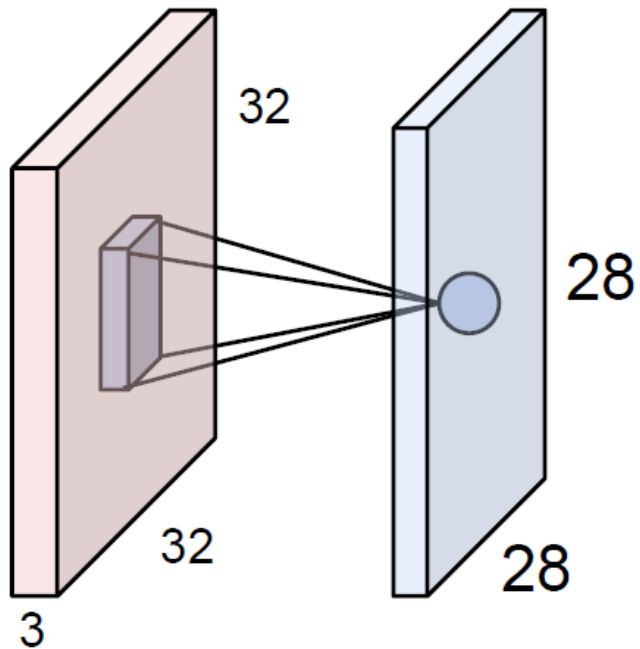


Result is a number
($5 \times 5 \times 3 = 75$ -dimensional dot product)



It's just a neuron with local connectivity!

The brain/neuron view of CONV Layer

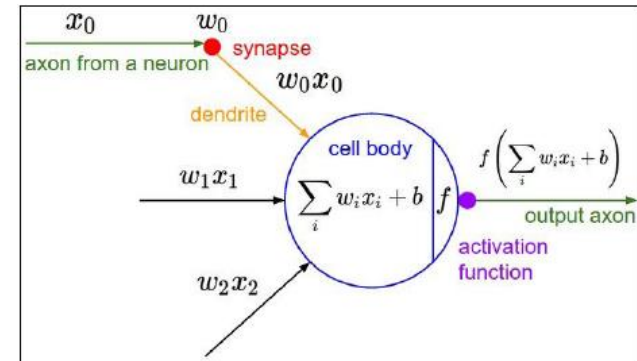
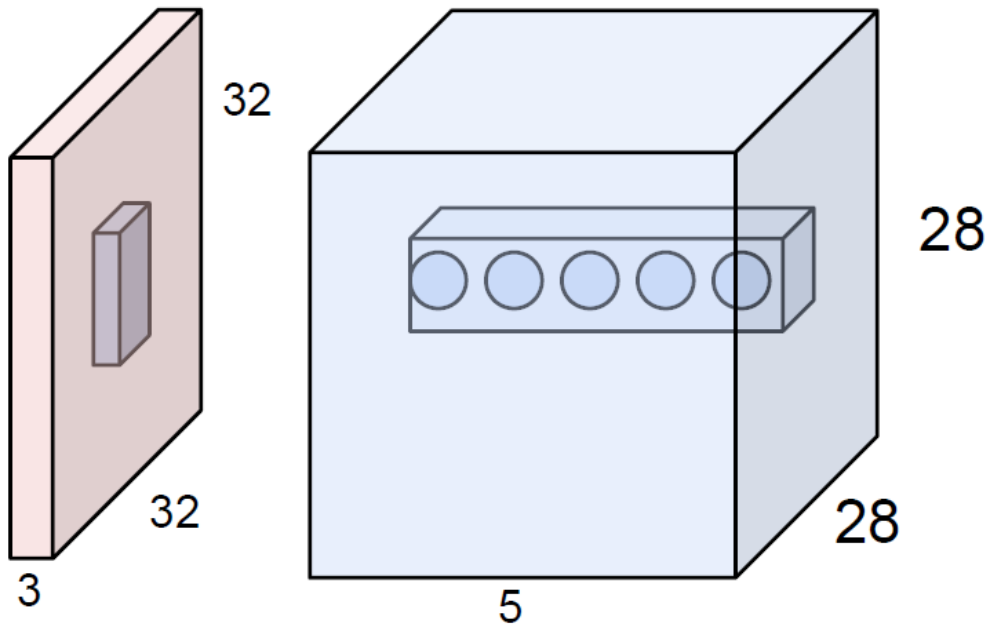


An activation map is a 28x28 sheet of neuron outputs:

1. Each is connected to a small region in the input
2. All of them share parameters

5x5 filter -> 5x5 receptive field for each neuron

The brain/neuron view of CONV Layer

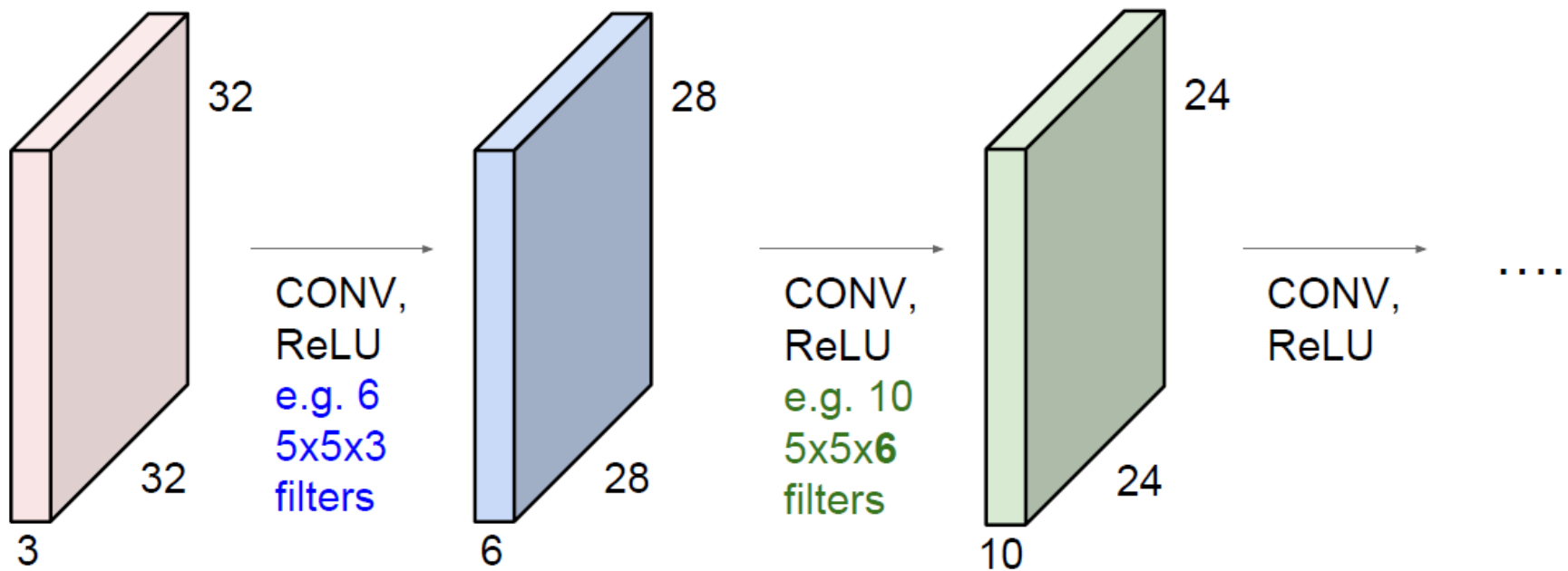


E.g. with 5 filters,
CONV layer consists of
neurons arranged in a 3D
grid (28x28x5).

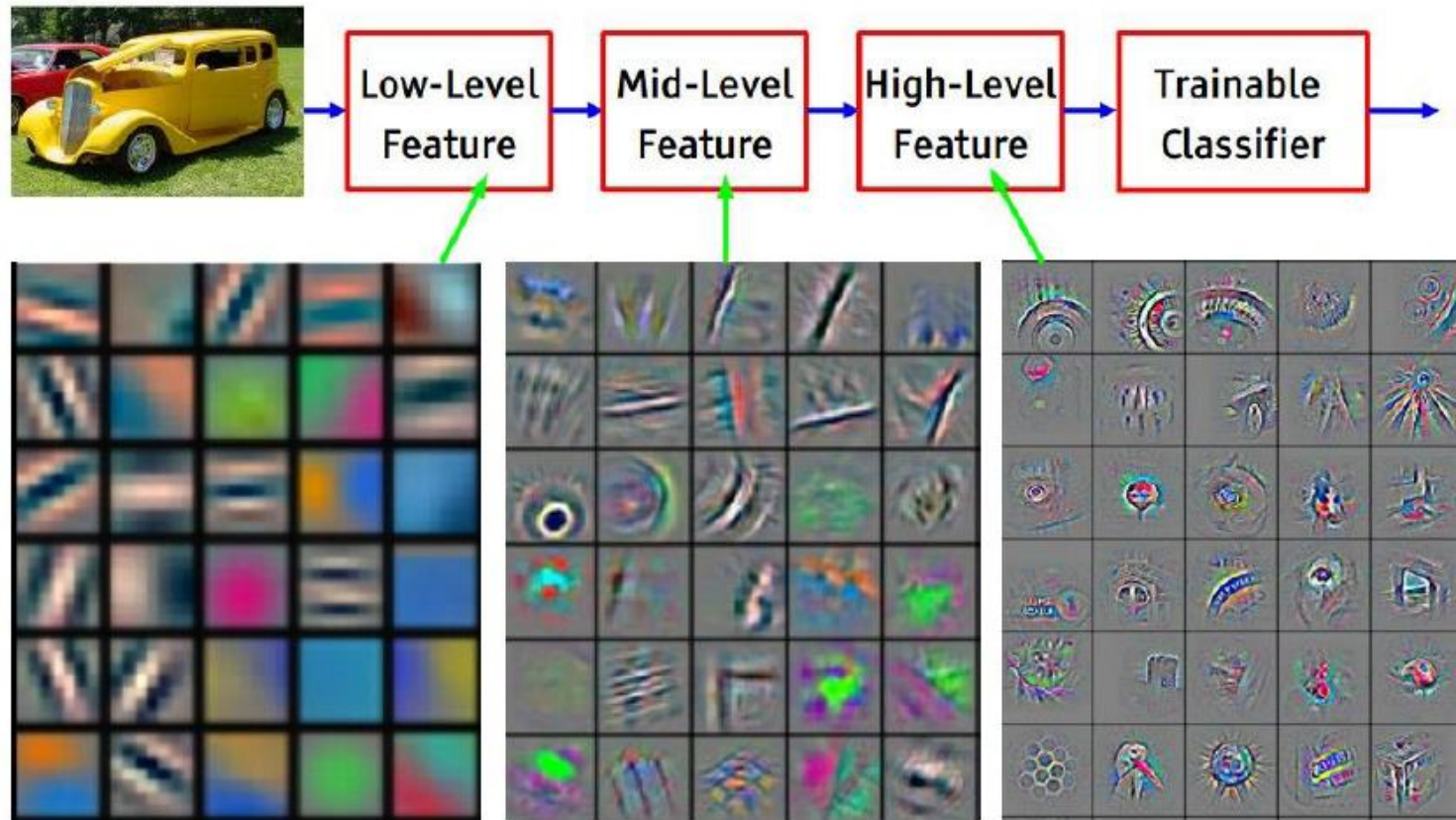
There will be 5 different neurons all looking
at the same region in the input volume.

ConvNet

In the simplest sense, ConvNet is a sequence of convolution layers, interspersed with activation functions.



Learned filters



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Filters activate when they see some type of visual feature such as an edge or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers.

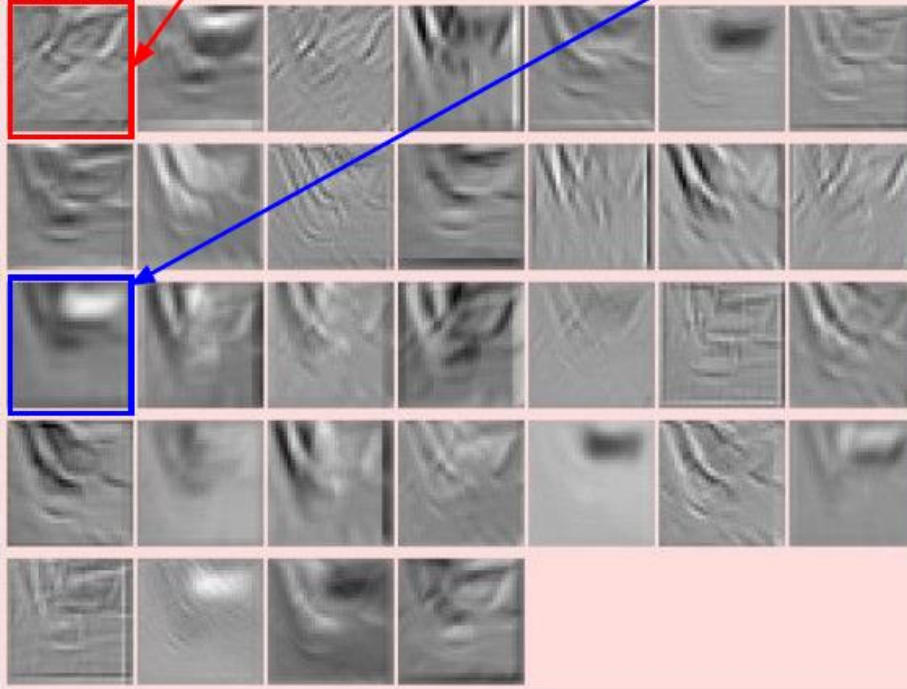
Learned filters



one filter =>
one activation map



Activations:



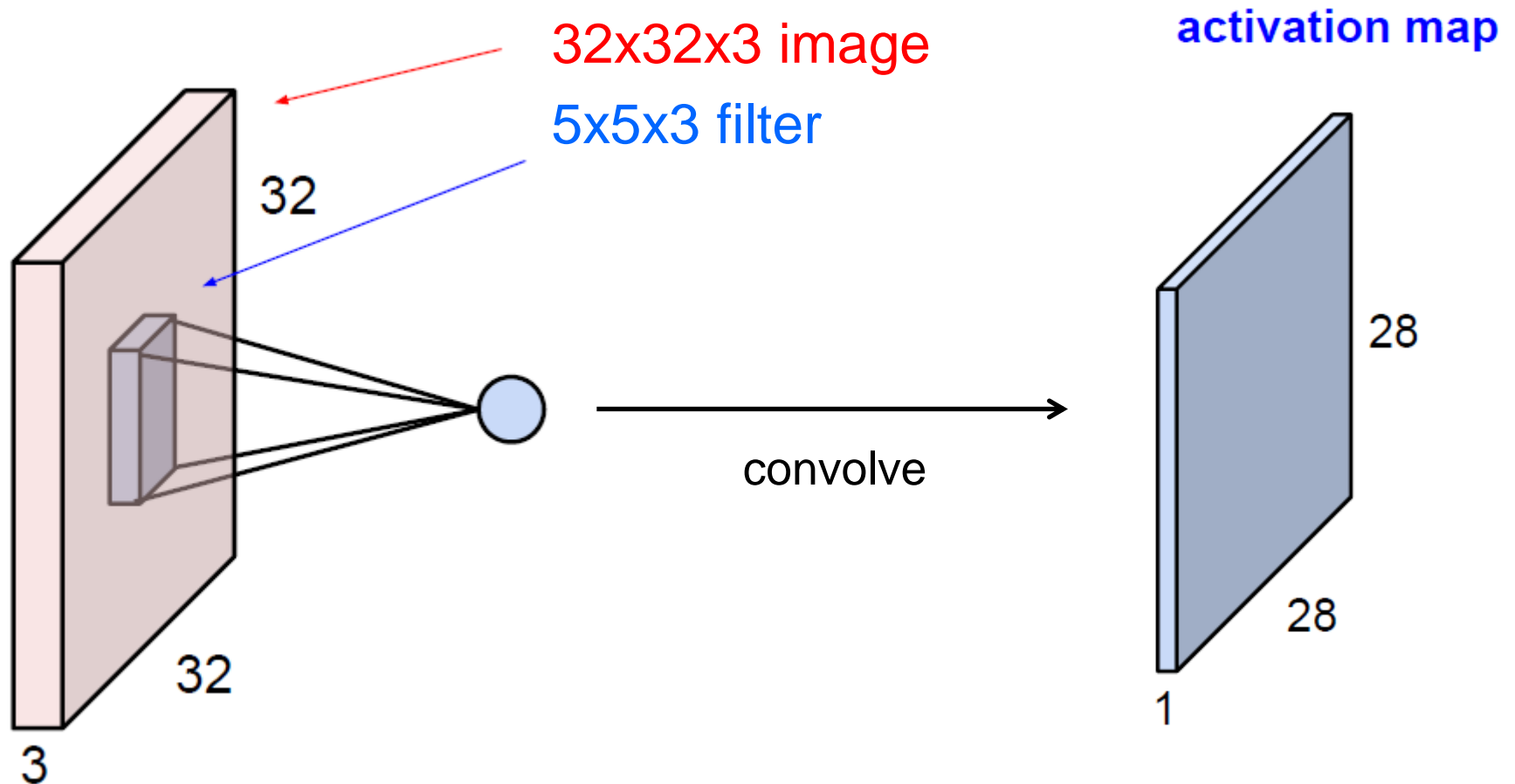
example 5x5 filters
(32 total)

ConvNet: preview

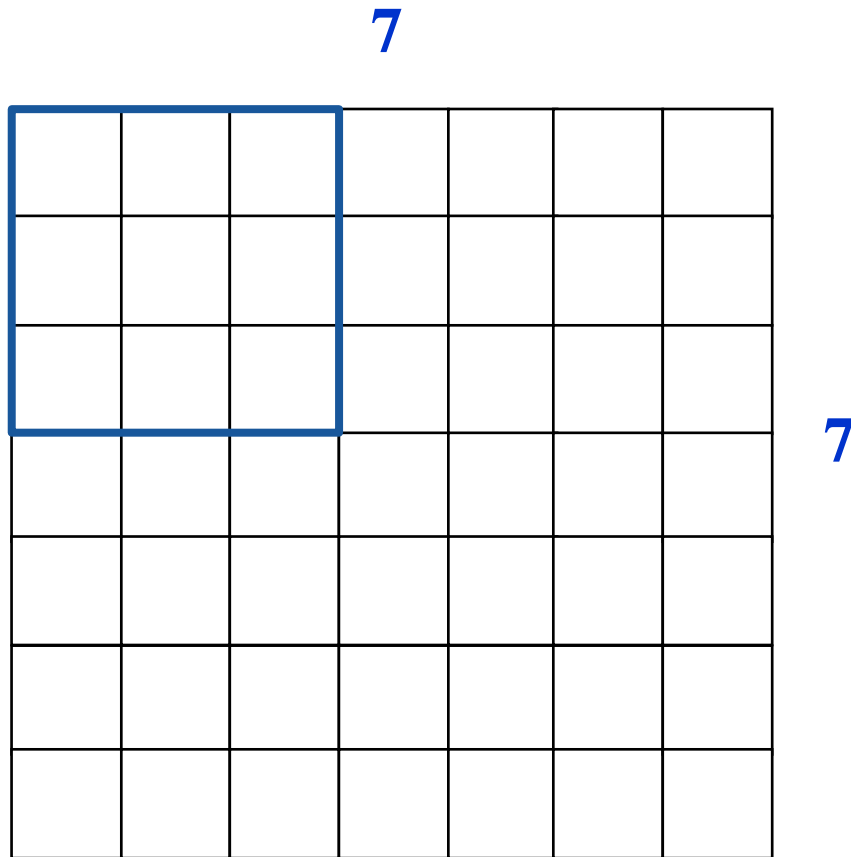


Convolution Layer

A closer look at spatial dimensions

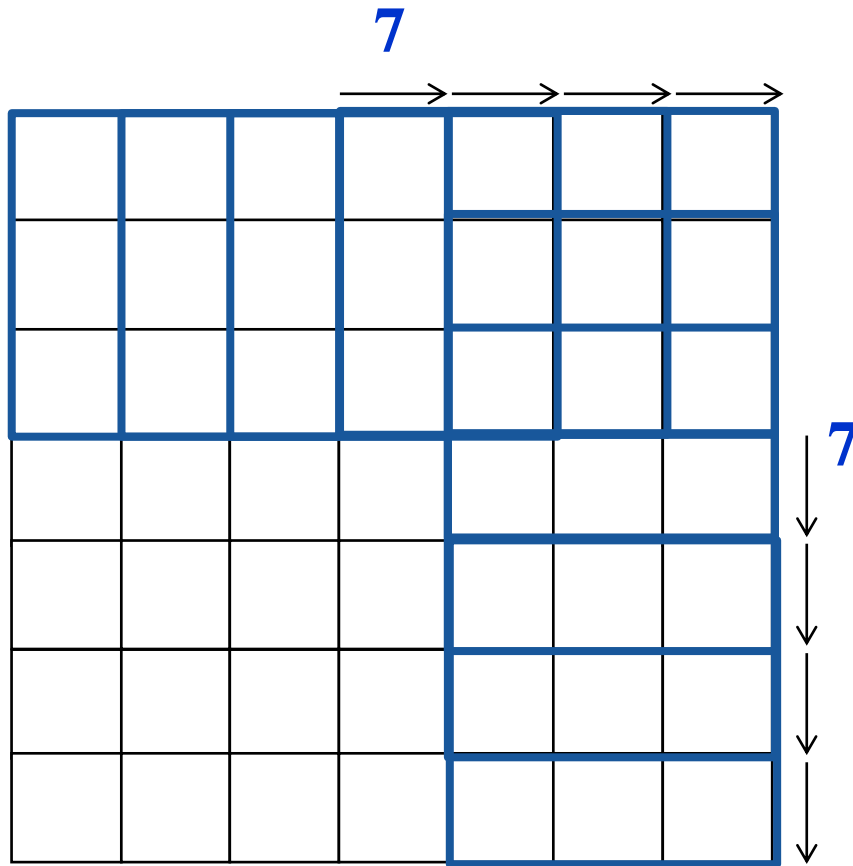


Convolution Layer



Assume,
input image is 7x7
filter is 3x3

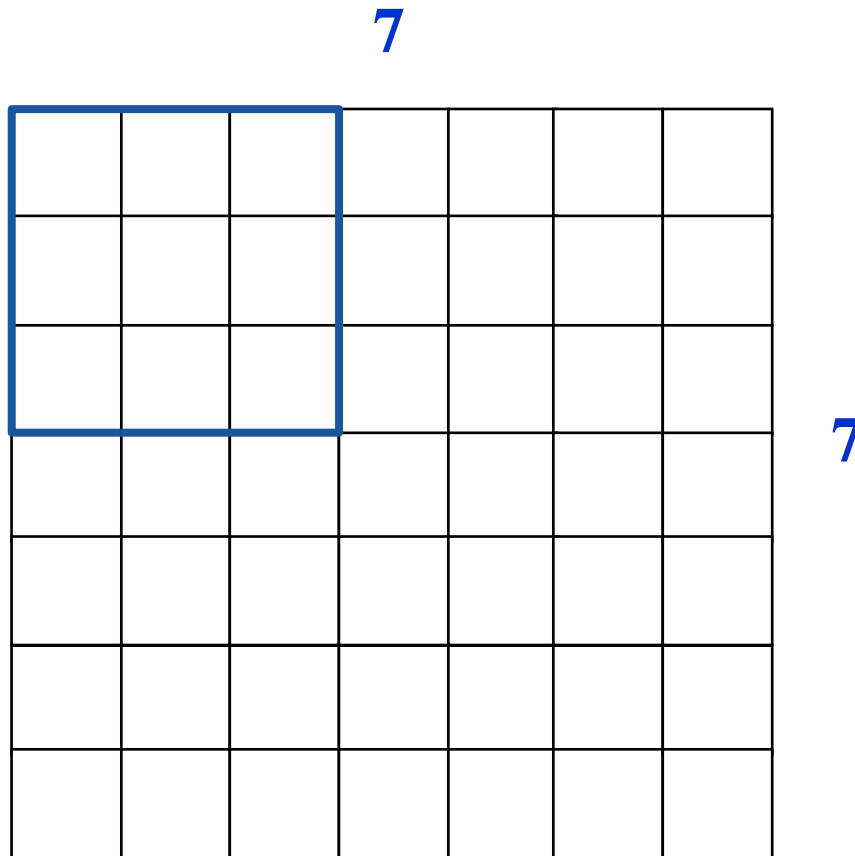
Convolution Layer



Assume,
input image is 7x7
filter is 3x3

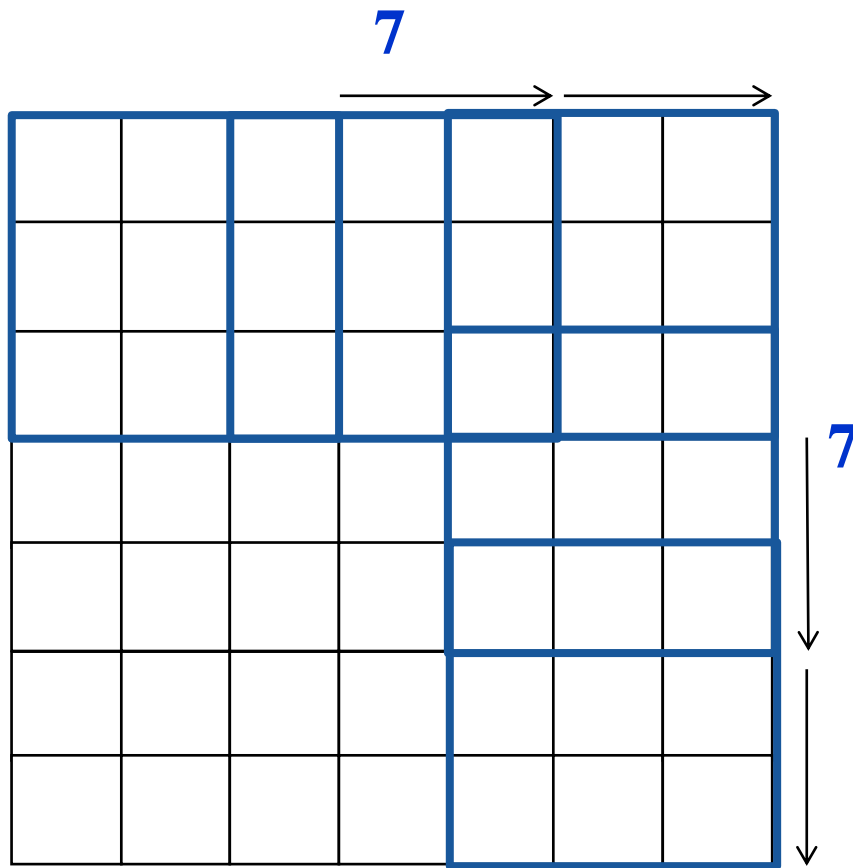
output is 5x5

Convolution Layer



Assume,
input image is 7x7
filter is 3x3
stride is 2

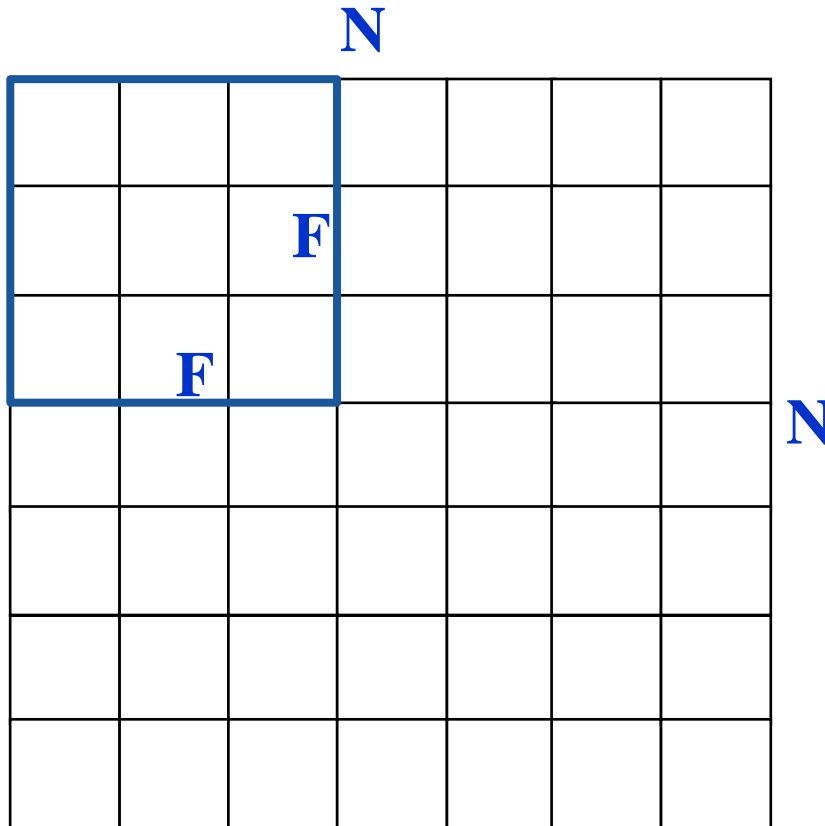
Convolution Layer



Assume,
input image is 7x7
filter is 3x3
stride is 2

output is 3x3

Convolution Layer



Output size:

$$(N-F) / \text{stride} + 1$$

Example $N = 7, F = 3$:

- stride 1 $\Rightarrow (7 - 3)/1 + 1 = 5$
- stride 2 $\Rightarrow (7 - 3)/2 + 1 = 3$
- stride 3 $\Rightarrow (7 - 3)/3 + 1 = 2.33$

\therefore (stride 3 is not proper
for $N=7$ and $F=3$

Convolution in practice: Zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7, filter 3x3,
applied with **stride 1**, **pad** with **1 pixel**
=> what is the output?

$$\text{Output} = (N - F) / \text{stride} + 1$$

Convolution in practice: Zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

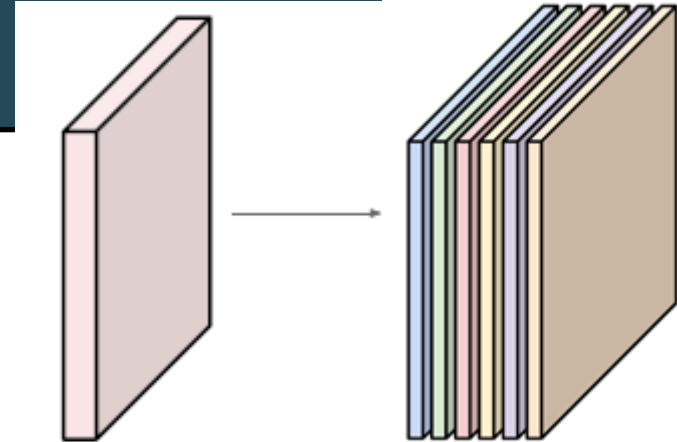
e.g. input 7x7, filter 3x3,
applied with **stride 1**, **pad** with **1 pixel**
=> what is the output?

7x7 output!

In general, common to see CONV
layers with stride 1, filters of size $F \times F$,
and zero-padding with $(F-1)/2$
(will preserve size spatially).

E.g. $F = 3 \Rightarrow$ zero pad with 1
 $F = 5 \Rightarrow$ zero pad with 2
 $F = 7 \Rightarrow$ zero pad with 3

No. of parameters



Input volume: **32x32x3**

10 different 5x5 filters with stride 1, pad 2

Output volume size:

$(32+2*2-5)/1+1 = 32$ spatially,

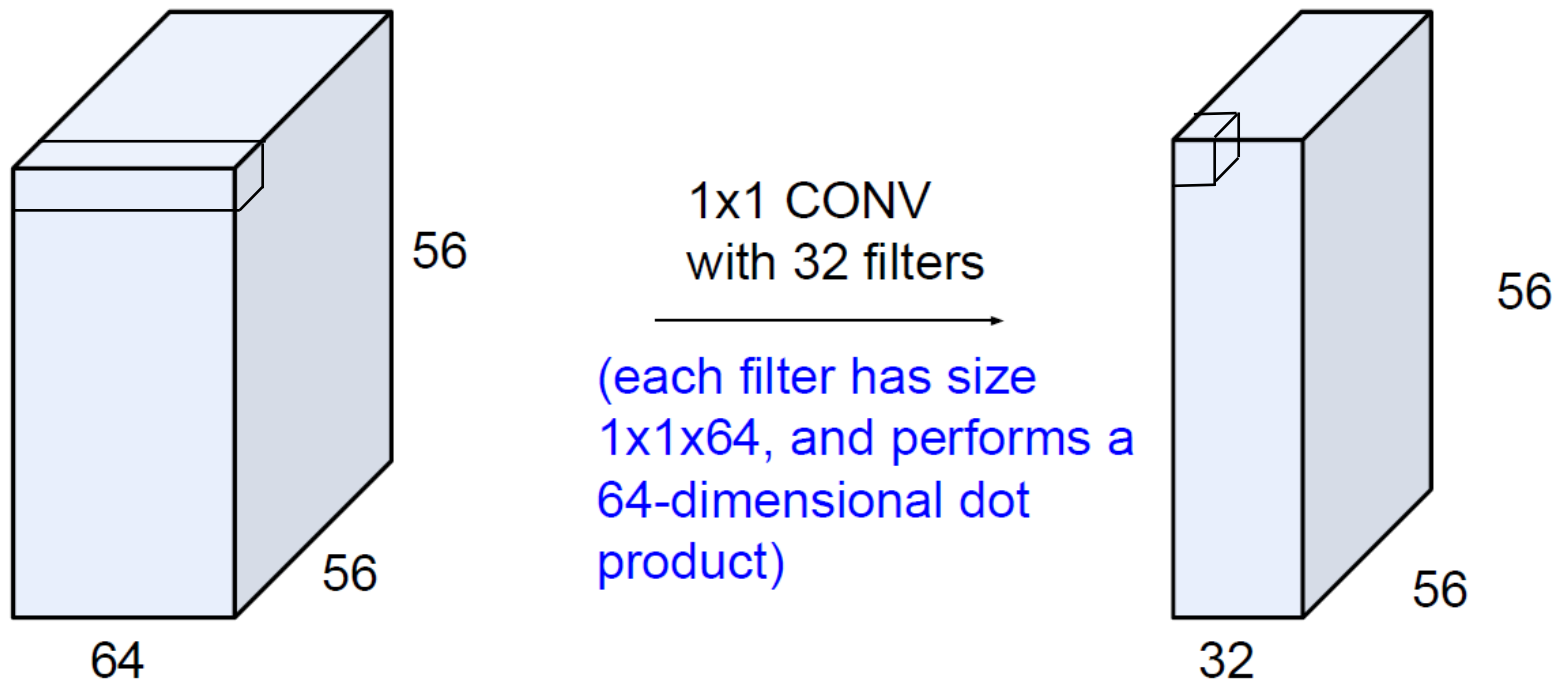
so **32x32x10**

Number of parameters in this layer:

each filter has $5*5*3 + 1 = 76$ params (+1 for bias)

so **76*10 = 760**

1x1 convolution layers make sense



Examples:

CONV layer in Tensorflow-Keras

- Number of filters K ,
- their spatial extent F ,
- the stride S ,
- the amount of zero padding P .



```
tf.keras.layers.Conv2D(32, kernel_size=5, strides=2, padding='same', activation='relu')
```

CONV layer in PyTorch

```
from torch import nn
nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=3, padding=1)
nn.ReLU()
```

CONV layer in MATLAB

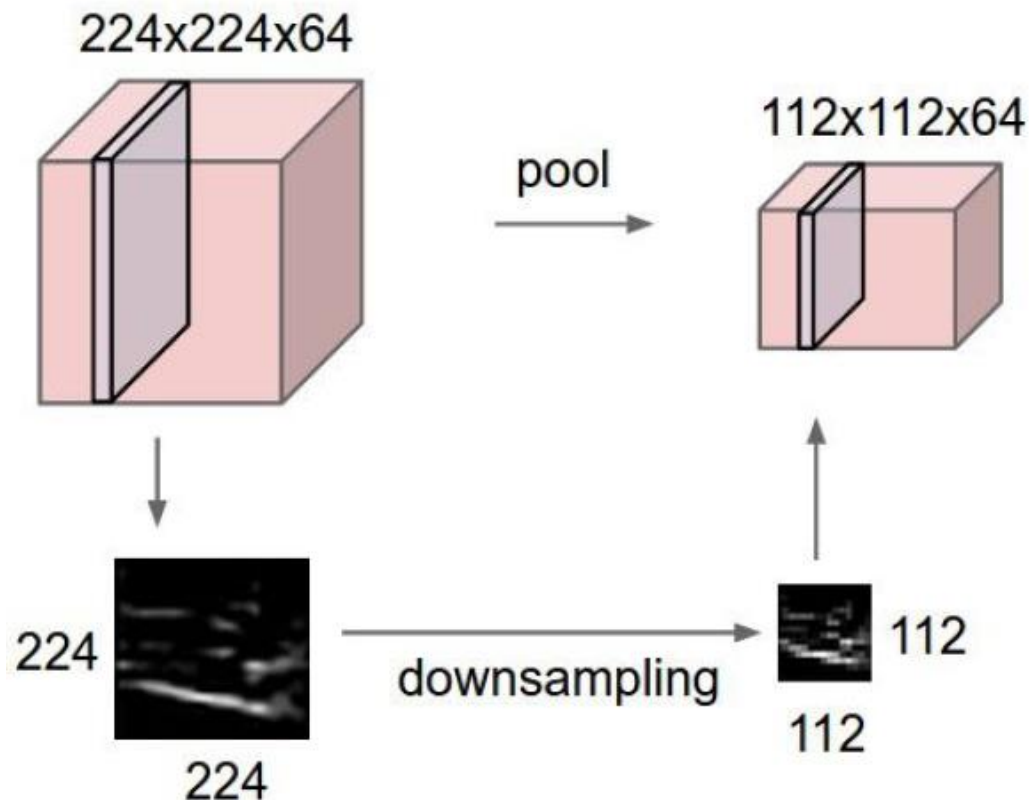
```
convolution2dLayer(11,96,'Stride',[4 4],'Padding',[2 2 2 2])
%11 indicates that the filter size is 11-by-11
%The second argument is the number of filters
```

Two more layers to go: POOL and FC

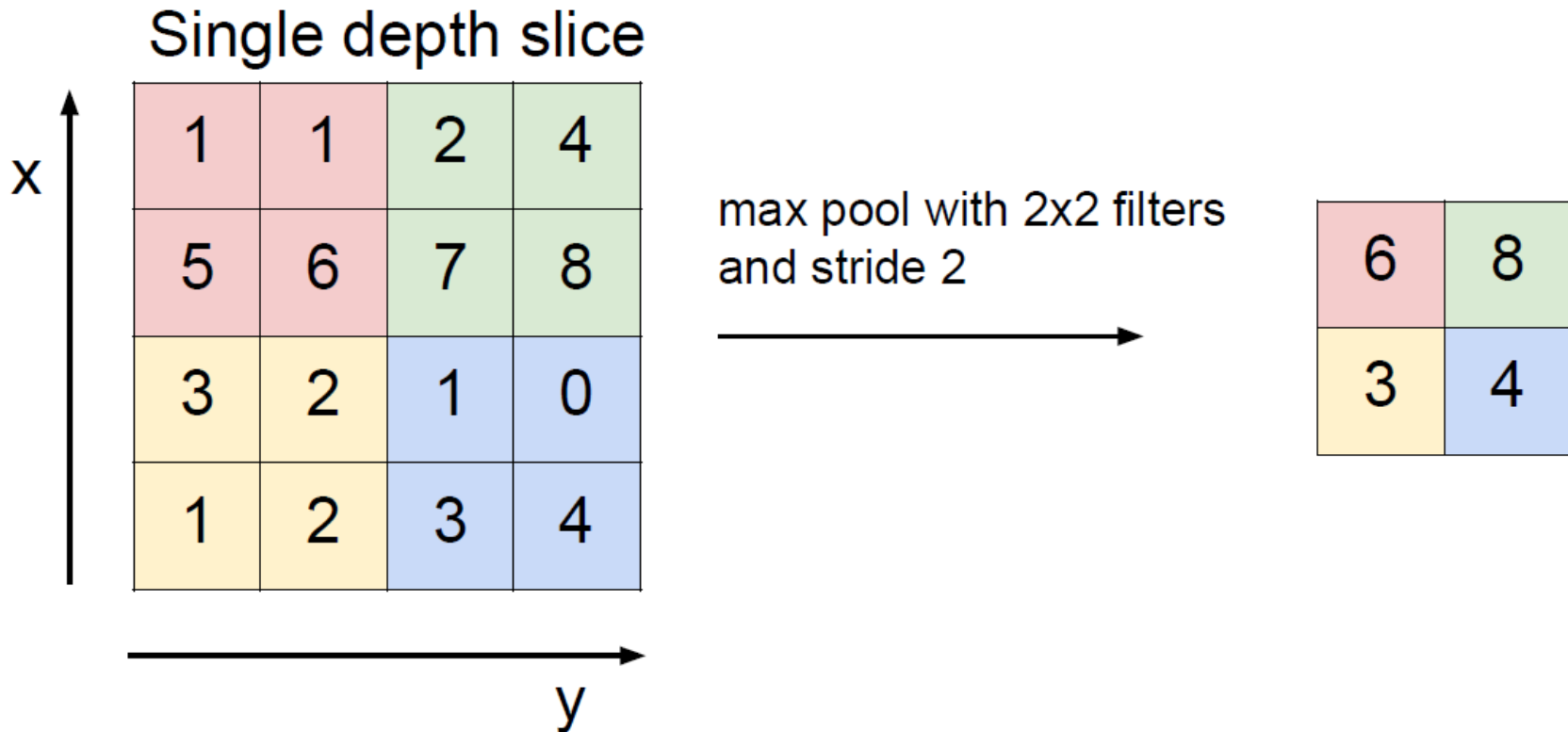


Pooling Layer

- makes the representations smaller and more manageable
- brings partial 'translation invariance'
- operates over each activation map independently:

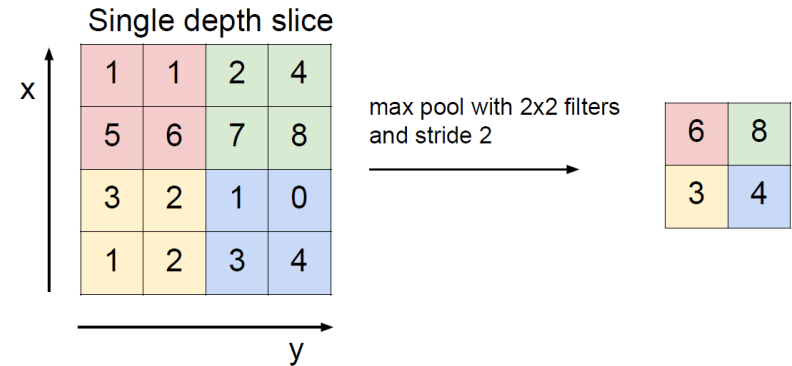


Max Pooling



Max Pooling

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
 - their spatial extent F ,
 - the stride S ,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F)/S + 1$
 - $H_2 = (H_1 - F)/S + 1$
 - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers



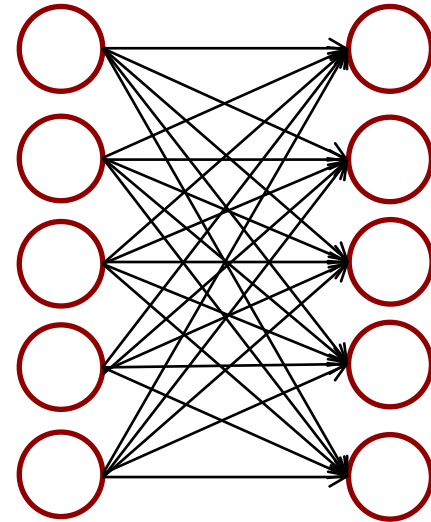
Common settings:

$$F = 2, S = 2$$

$$F = 3, S = 2$$

Fully Connected Layer

Contains neurons that connect to the entire input volume, as in ordinary neural networks.

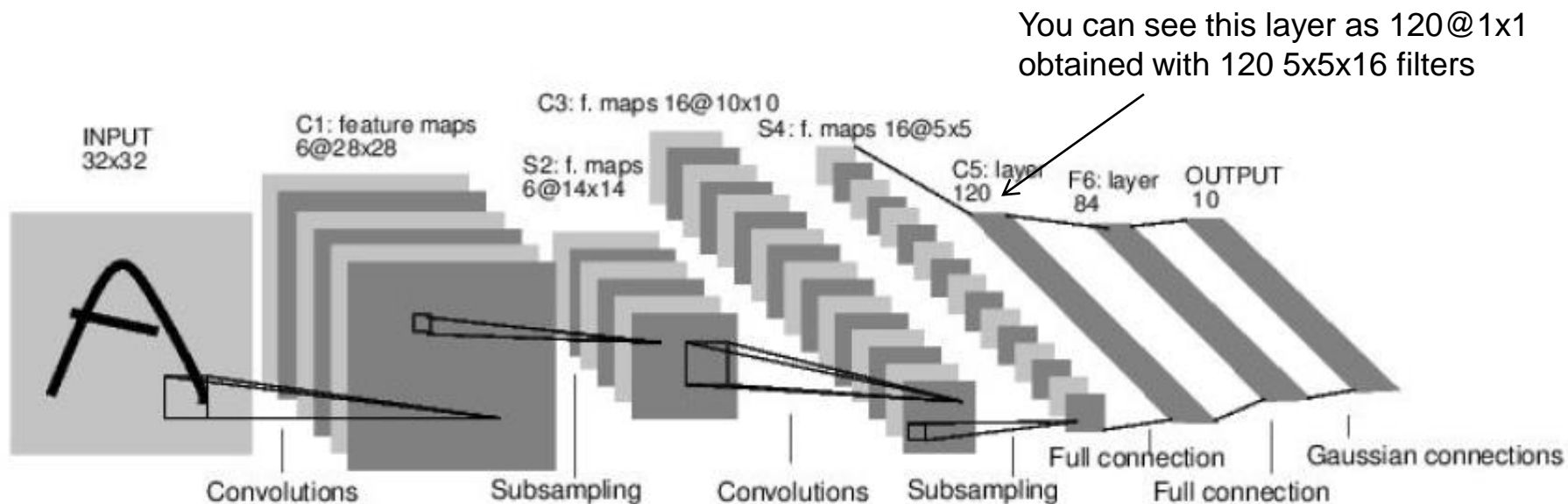


Case Study: LeNet-5 [LeCun et al. 1998]

Conv filters were 5x5, applied at stride 1

Subsampling (Pooling) layers were 2x2 applied at stride 2

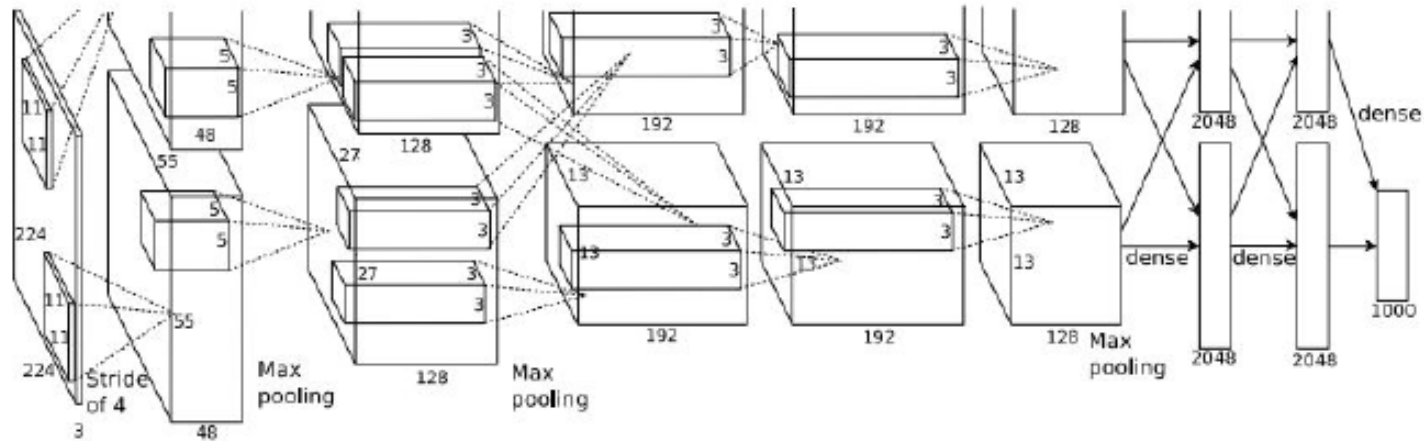
Architecture is [CONV-POOL-CONV-POOL-CONV-FC]



ConvNetJS demo: A 3-conv layer network training on CIFAR-10

<http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>

Case Study: AlexNet [Krizhevsky et al. 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11x3 filters applied at stride 4

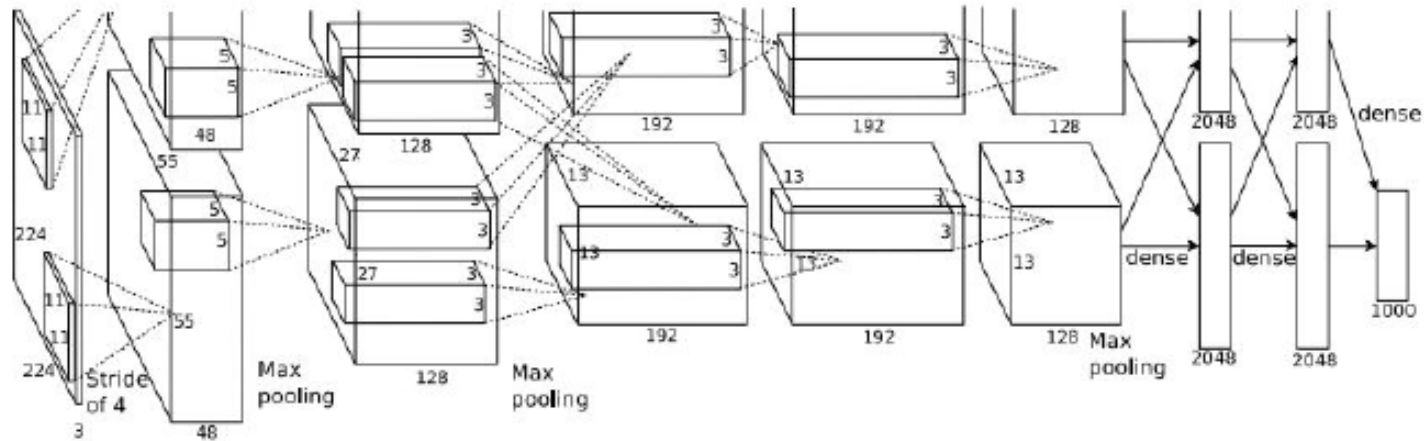
Output = $55 = (227-11)/4+1 = (N-F)/\text{stride}+1$

Output volume = **[55x55x96]**

Number of parameters in this layer = $(11*11*3+1)*96 = \mathbf{35K}$

without weight sharing it'd be 105M

Case Study: AlexNet [Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

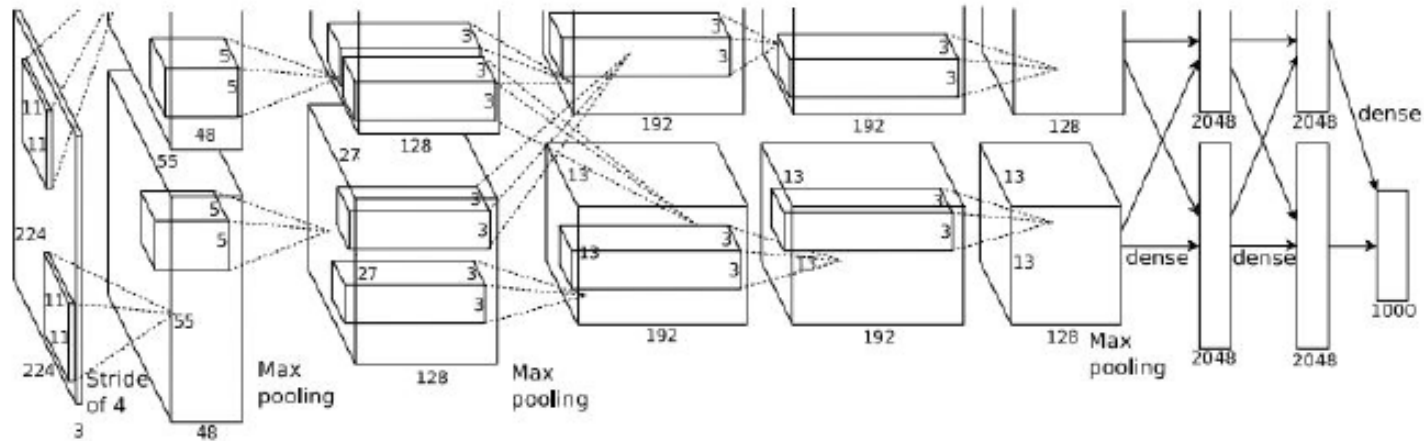
Second layer (POOL1): 3x3 filters applied at stride 2

Output = $27 = (55-3)/2+1 = (N-F)/stride+1$

Output volume = **[27x27x96]**

The number of parameters in this layer = 0

Case Study: AlexNet [Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

After POOL1: 27x27x96

Case Study: AlexNet [Krizhevsky et al. 2012]

Full AlexNet architecture:

[227x227x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL1: 3x3 filters at stride 2

[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

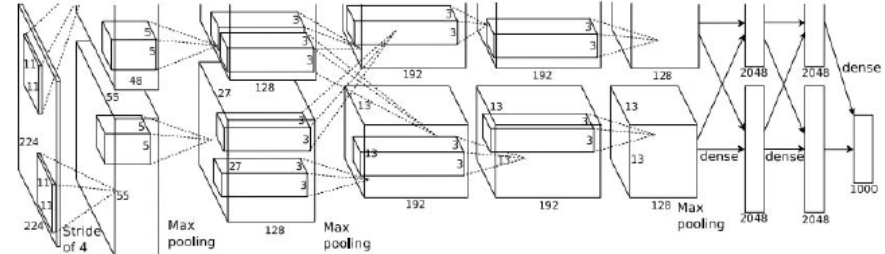
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

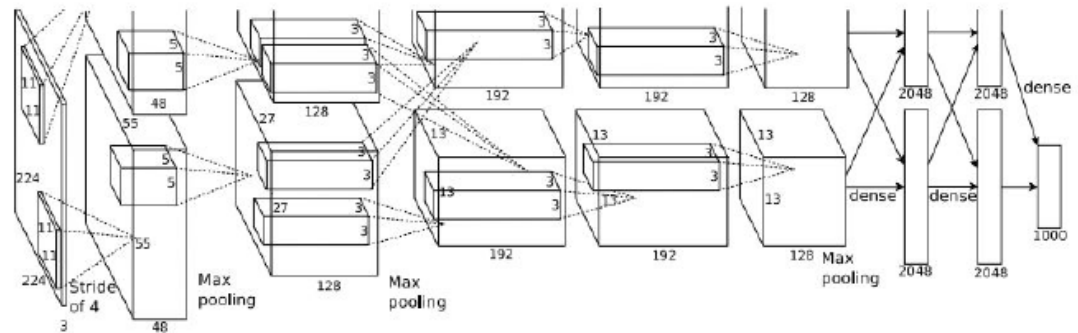
[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)



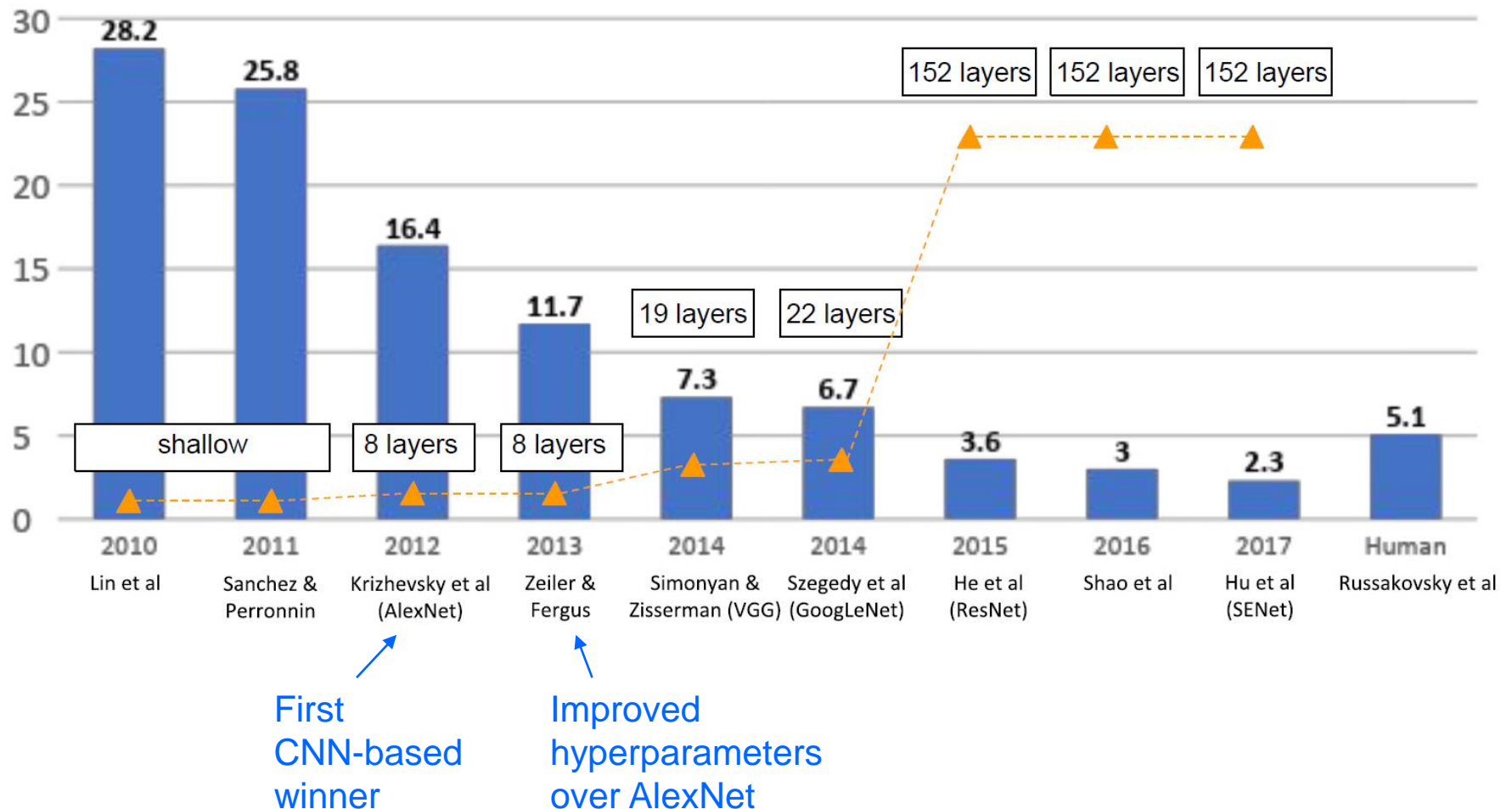
Case Study: AlexNet [Krizhevsky et al. 2012]



Details/Retrospectives:

- first use of ReLU
- dropout 0.5
- mini-batch size 128
- SGD Momentum 0.9
- learning rate: $1e-2$, learning decay by a factor of 10
- ImageNet top-5-error 16.4% (second best: 26%)

ILSVRC (ImageNet Challenge) winners



Case Study: VGGNet [Simonyan and Zisserman, 2014]

Small filters, deeper network

Only 3x3 CONV (stride 1, pad 1)

and 2x2 Max Pool (stride 2)

16 layers with weights

138 million parameters

(120 million from FC layers)

Best one among
tried alternatives

Decreased

11.2% top-5-error in 2013

to

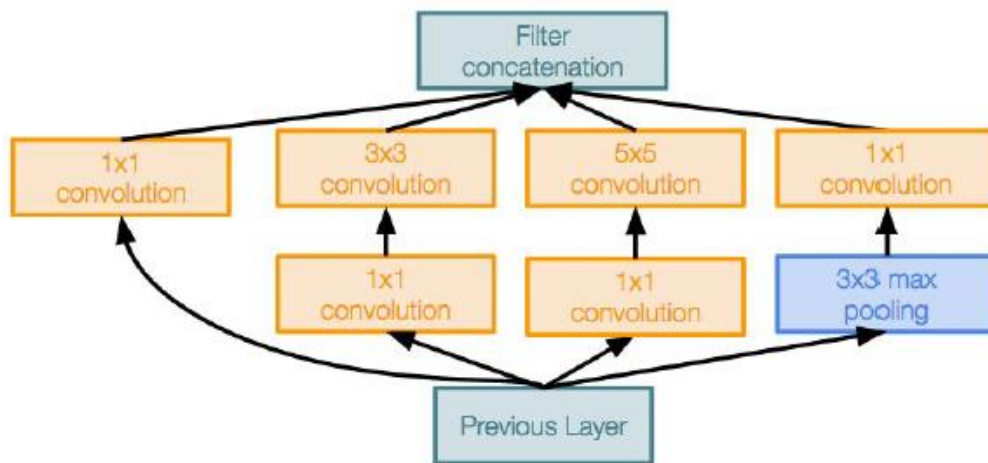
7.3% top-5-error (in 2014)

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					

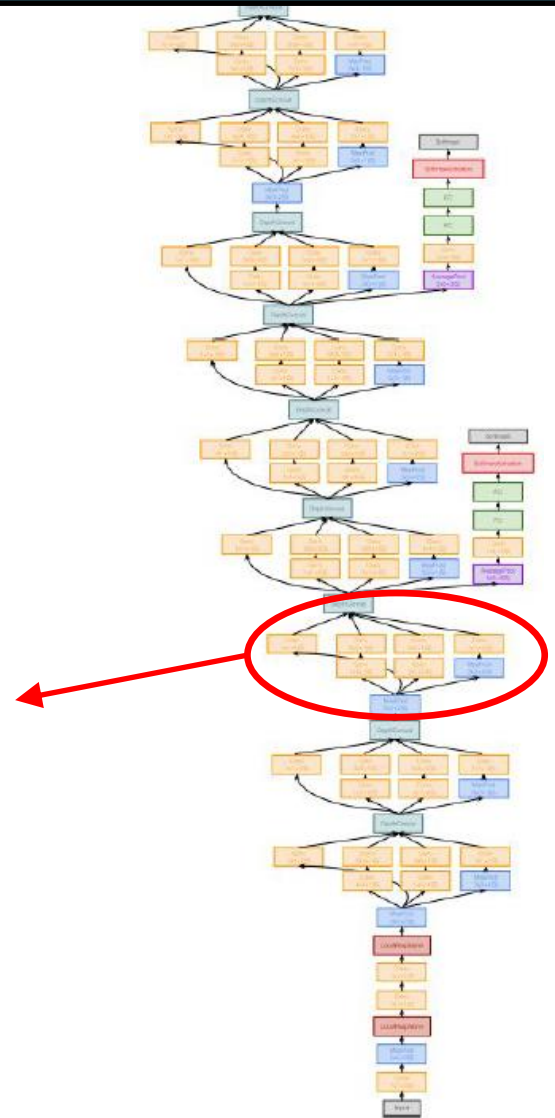
Case Study: GoogLeNet

[Szegedy et al., 2014]

- Efficient 'inception' modules
- Only 5 million parameters!
- No FC layers
- ILSVRC 2014 winner (6.7% top-5-error)



Inception module

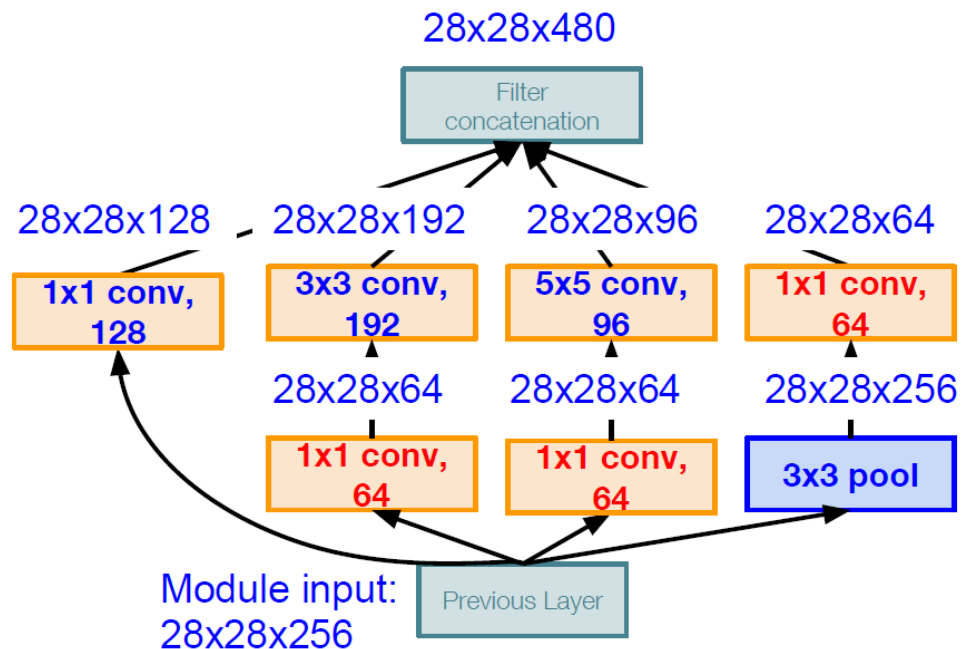


Case Study: GoogLeNet [Szegedy et al., 2014]

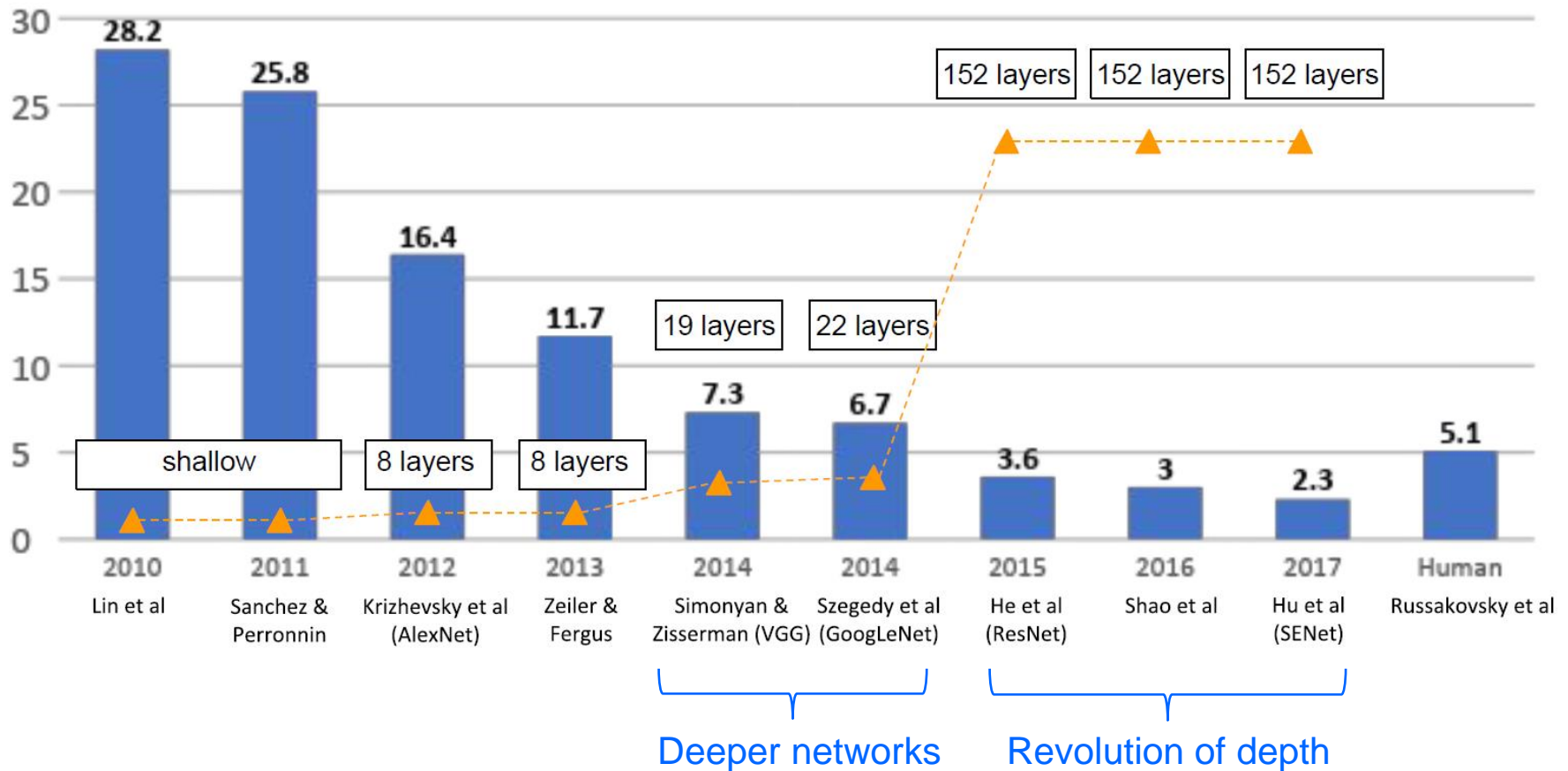
Inception module:

- Apply parallel filter operations on the input from previous layer
 - Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
 - Pooling operation (3x3)
- Concatenate all filter outputs together depth-wise

An example module with sizes:

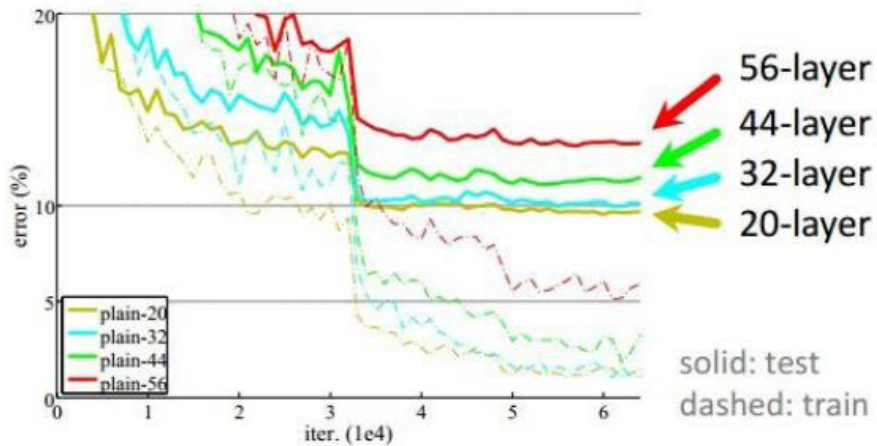


ILSVRC (ImageNet Challenge) winners

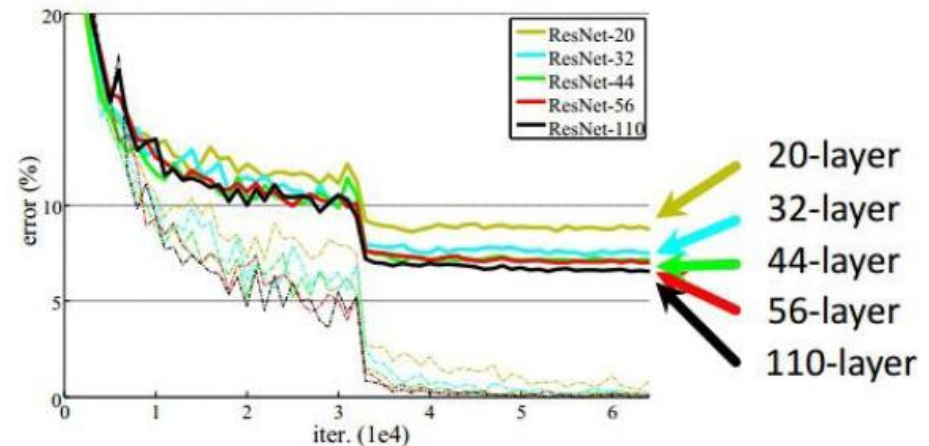


Case Study: ResNet [He et al., 2015]

CIFAR-10 plain nets



CIFAR-10 ResNets



Left: training and test errors on CIFAR-10 with 20, 32, 44, and 56-layer “plain” networks (AlexNet, VGG etc.)

The deeper network has higher train and test error.

Right: train/test errors with ResNets, the deeper the better.

Case Study: ResNet [He et al., 2015]

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize.

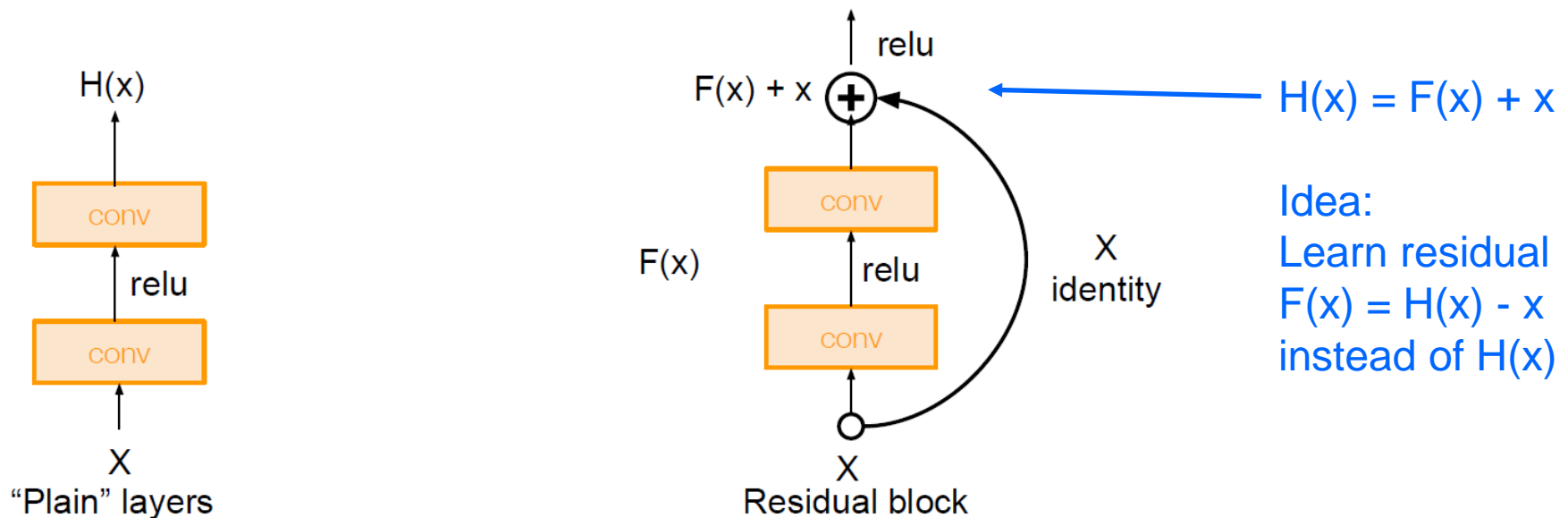
The deeper model should be able to perform at least as well as the shallower model.

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

Case Study: ResNet [He et al., 2015]

Solution: Use network layers to fit a residual mapping

Deep residual network consists of residual blocks $F(x)$ applied at regular intervals (e.g. every two layer).



Case Study: ResNet

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)

