CENG 506 Deep Learning

Lecture 6
CNN Architectures

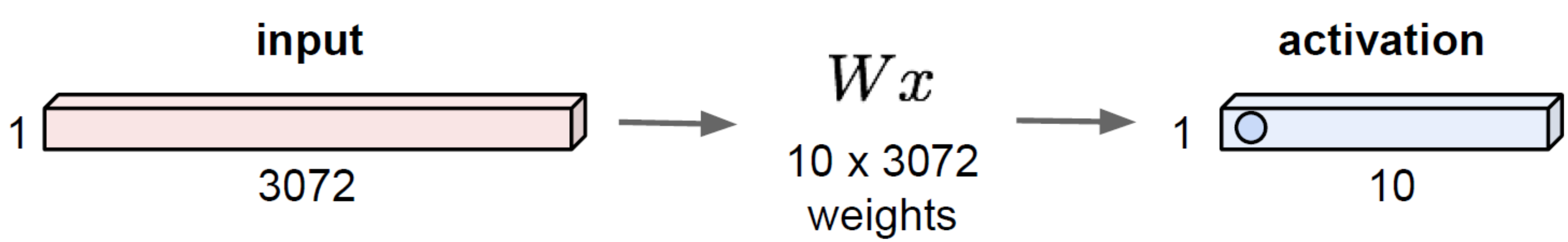
Slides were prepared using the course material of
Stanford's CNN Course (CS231n, Fei-Fei,Johnson,Yeung) and

Why not regular NN for images?

- Regular Neural Nets don’t scale well to full images.
- CIFAR-10 image size is 32x32x3, 3072 weights for a single neuron in the first hidden layer.
  (If you think about 200x200 images, it is 120,000 weights.)

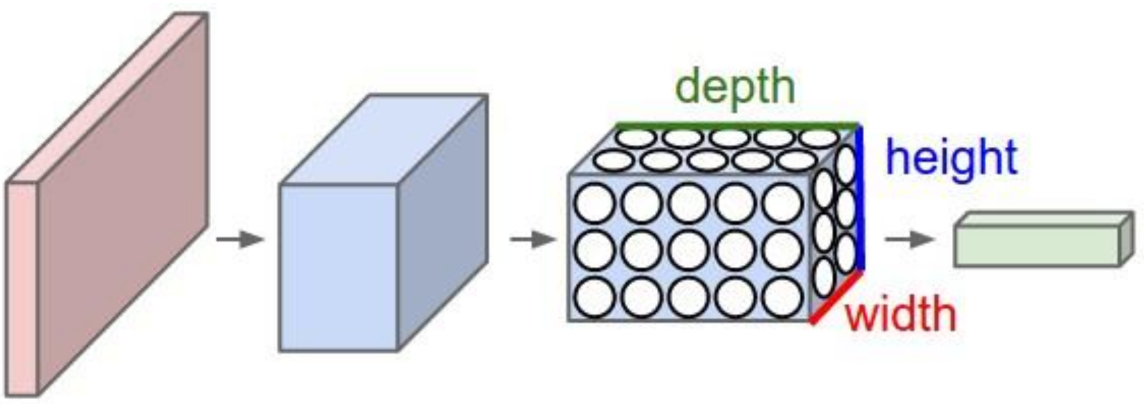
A fully-connected layer:

32x32x3 image -> stretch to 3072 x 1



Why not regular NN for images?

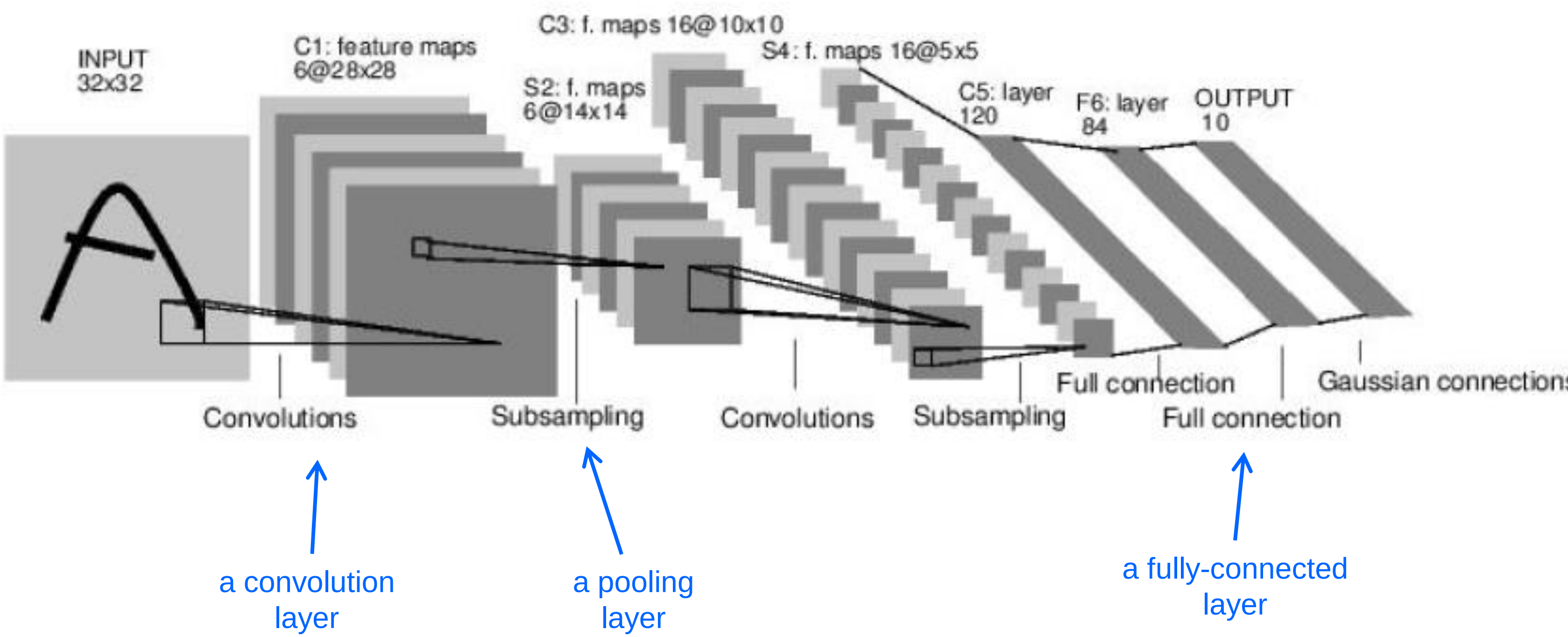
- Moreover, we would like to preserve spatial structure.
- Convolutional NN have neurons arranged in three dimensions: width, height, depth.



Input:
32x32x3

Output:
1x1x10

A CNN Architecture

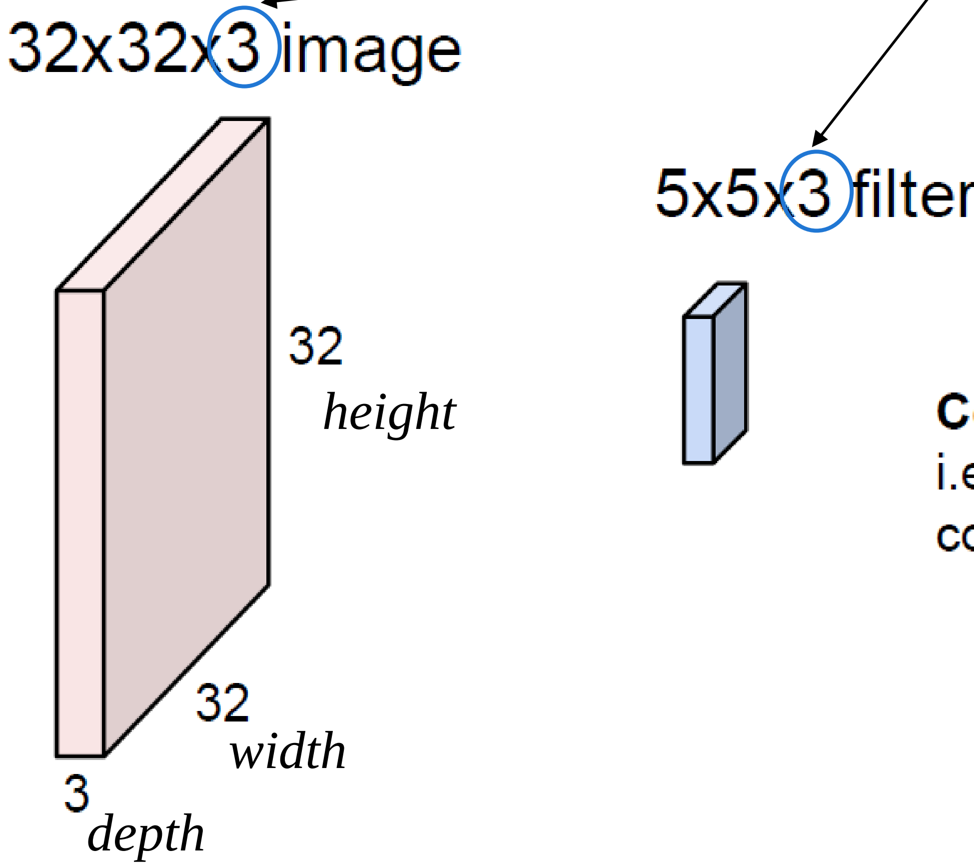


LeNet-5, LeCun 1998

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

Filters always extend the full depth of the input volume



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

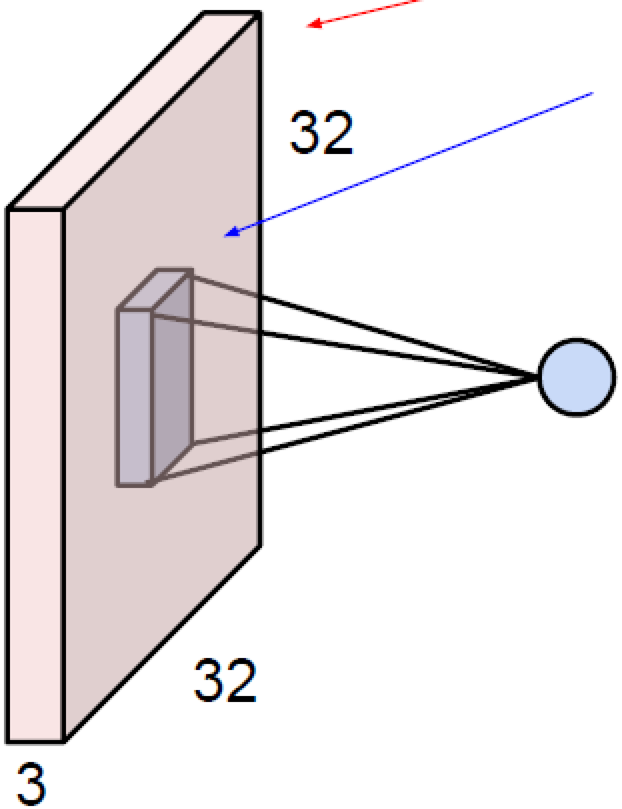
Convolution in image processing
image g, filter f

$$\boxed{\sum_{s=-a}^{a}\sum_{t=-b}^{b} f(s,t)\,g(x+s,y+t)}$$

Convolution Layer

32x32x3 image

5x5x3 filter w



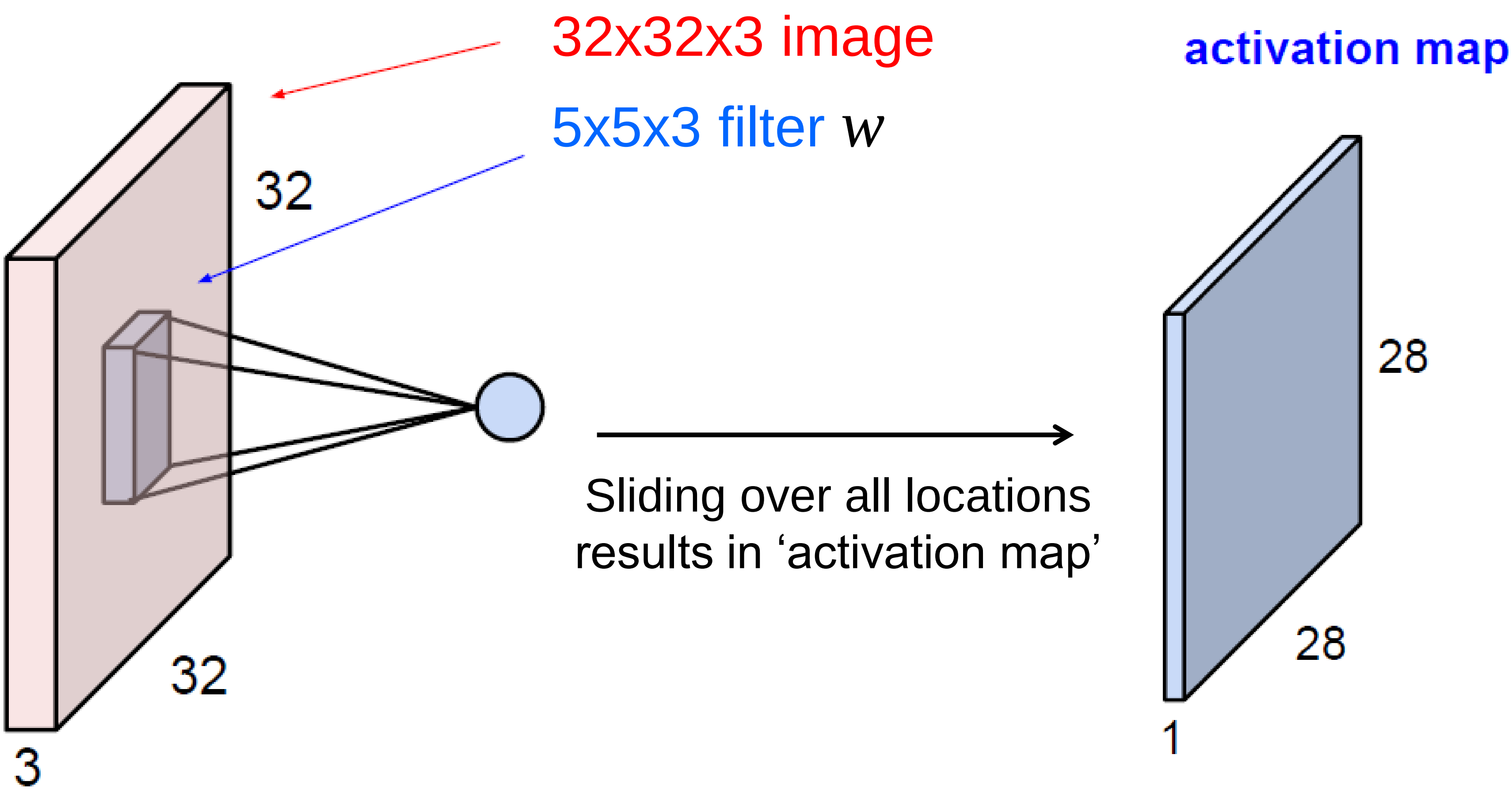
results in one number (scalar)

the dot product between the filter and a 5x5x3 chunk of the image

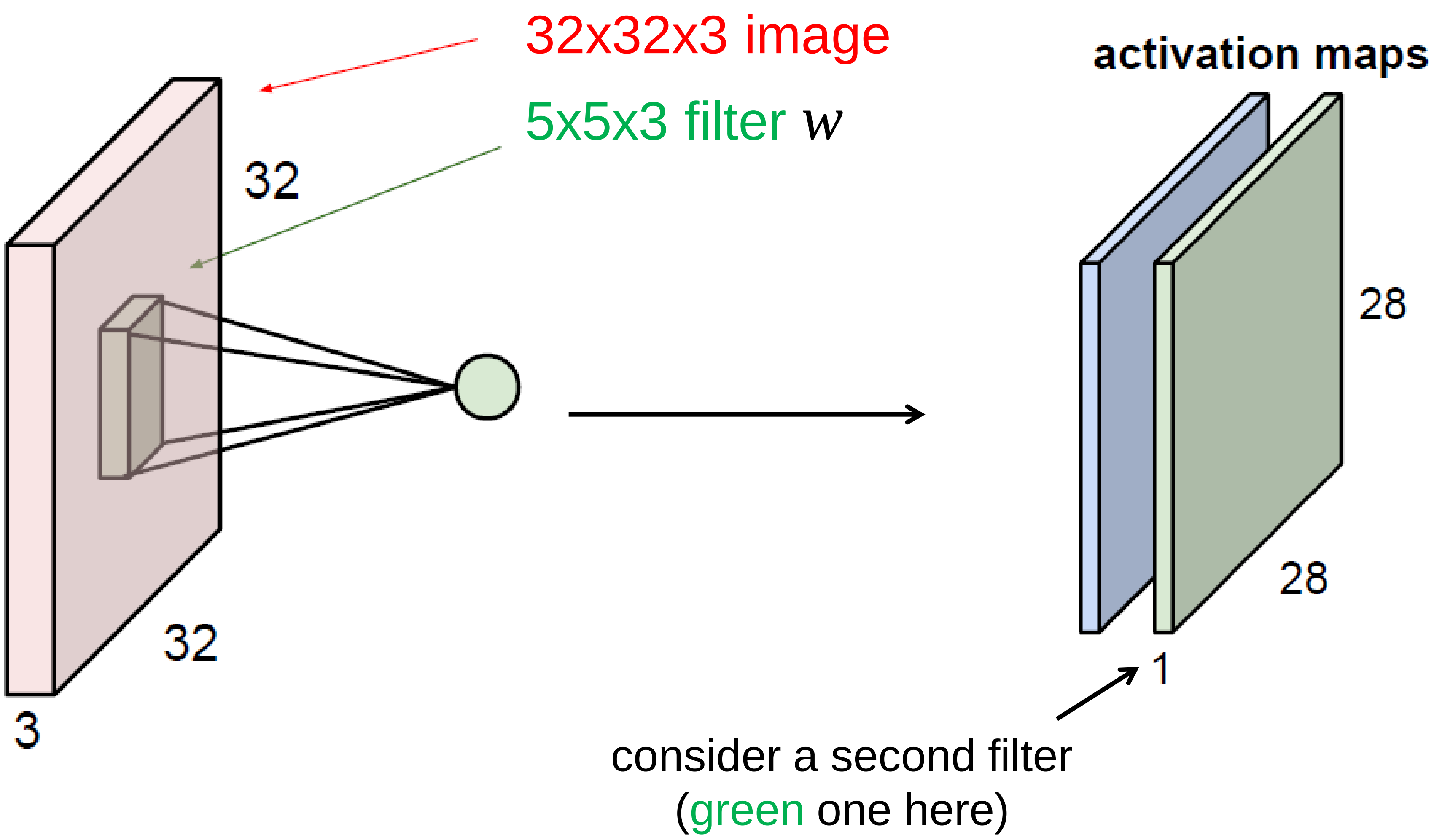
(5*5*3 = 75-dimensional dot product + bias)

$$\boldsymbol{w^T x + b}$$

Convolution Layer

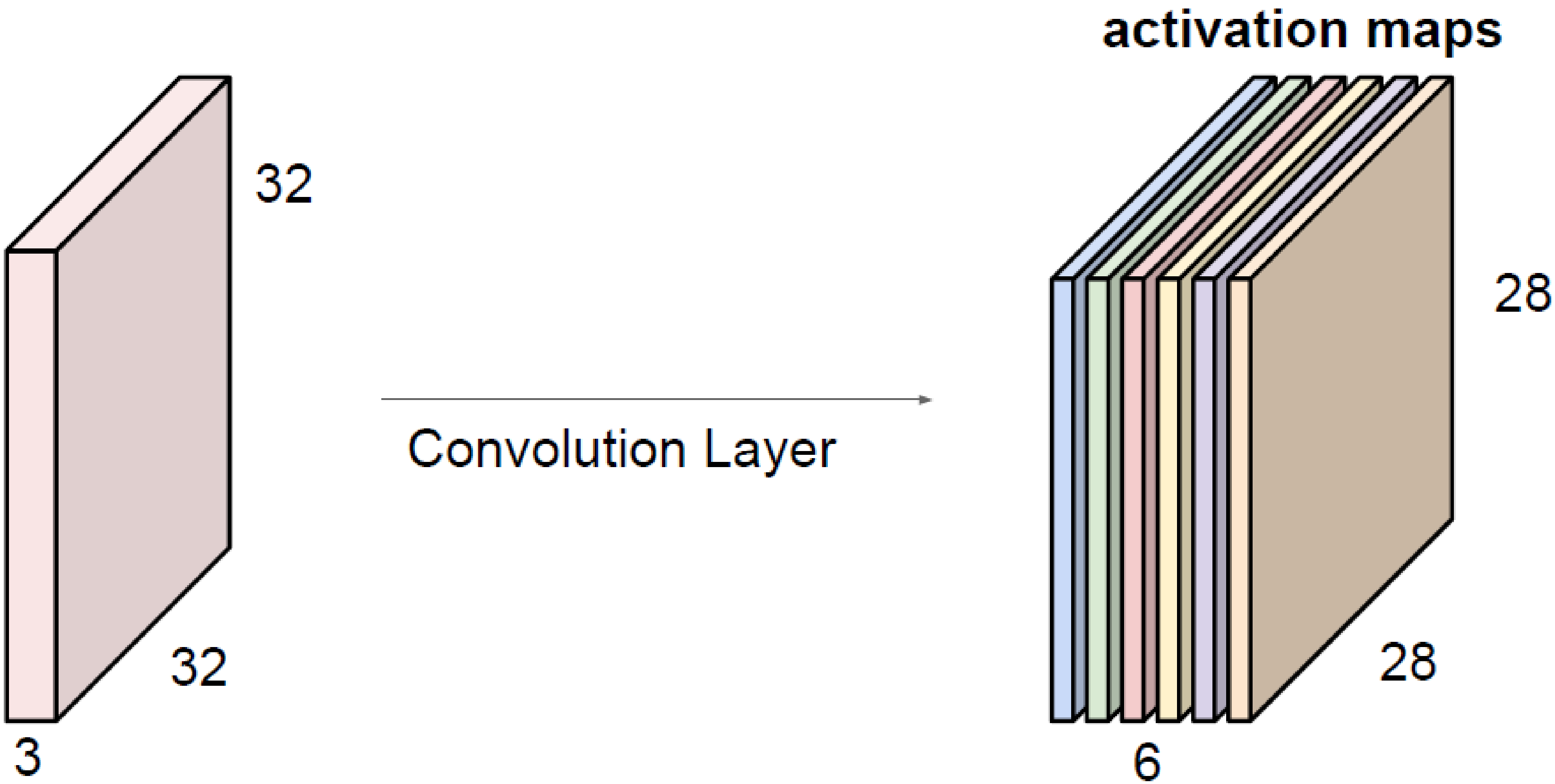


Convolution Layer

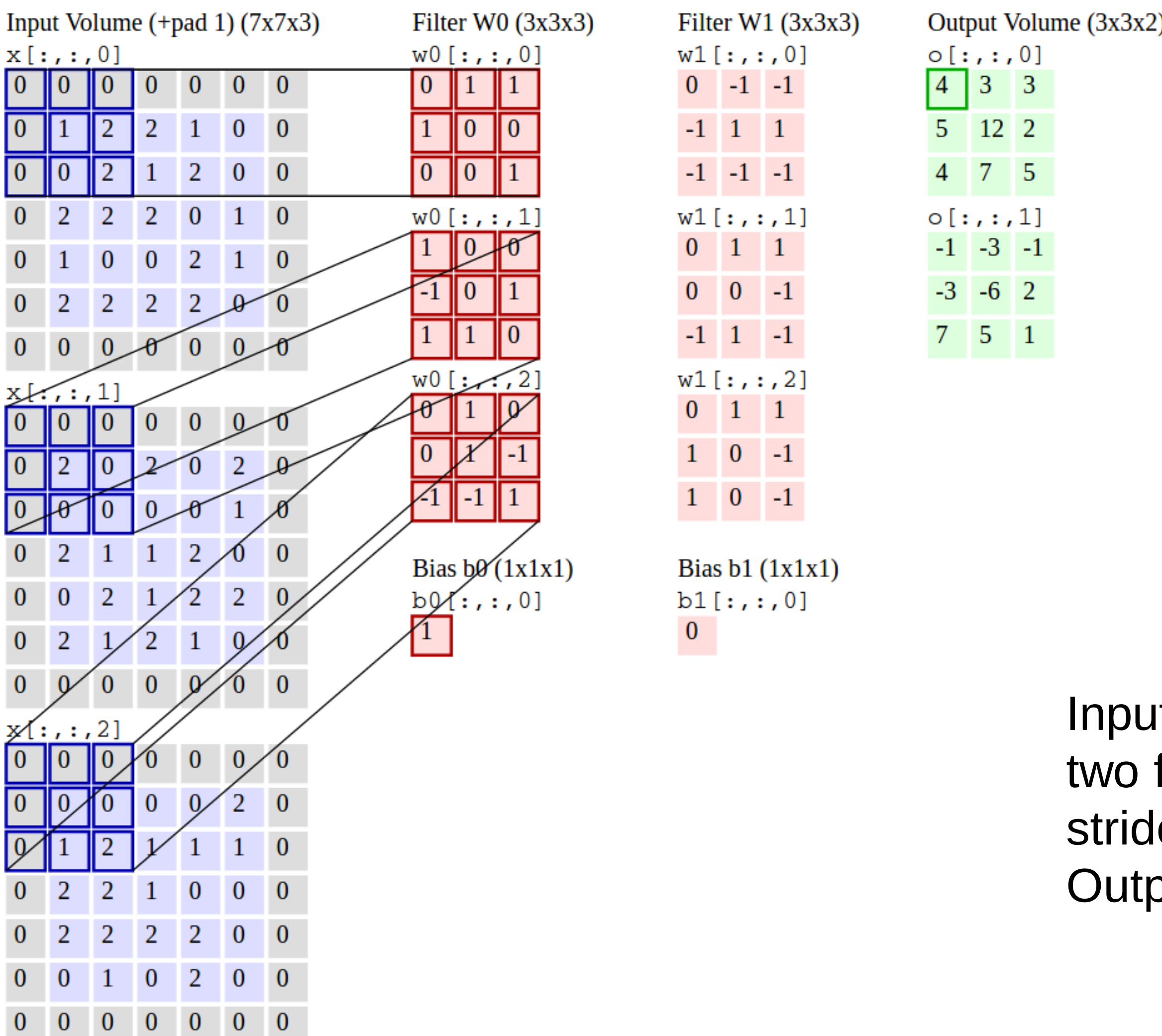


Convolution Layer

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 3x3,
stride of 2, zero padding of 1.
Output = (5-3+2)/2 + 1 = 3.

Convolution Demo

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	1	2	2	1	0	0
0	0	2	1	2	0	0
0	2	2	2	0	1	0
0	1	0	0	2	1	0
0	2	2	2	2	0	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	0	2	0	2	0
0	0	0	0	0	1	0
0	2	1	1	2	0	0
0	0	2	1	2	2	0
0	2	1	2	1	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	0	0	0	0	2	0
0	1	2	1	1	1	0
0	2	2	1	0	0	0
0	2	2	2	2	0	0
0	0	1	0	2	0	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

0	1	1
1	0	0
0	0	1

w0[:,:,1]

1	0	0
-1	0	1
1	1	0

w0[:,:,2]

0	1	0
0	1	-1
-1	-1	1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	-1	-1
-1	1	1
-1	-1	-1

w1[:,:,1]

0	1	1
0	0	-1
-1	1	-1

w1[:,:,2]

0	1	1
1	0	-1
1	0	-1

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

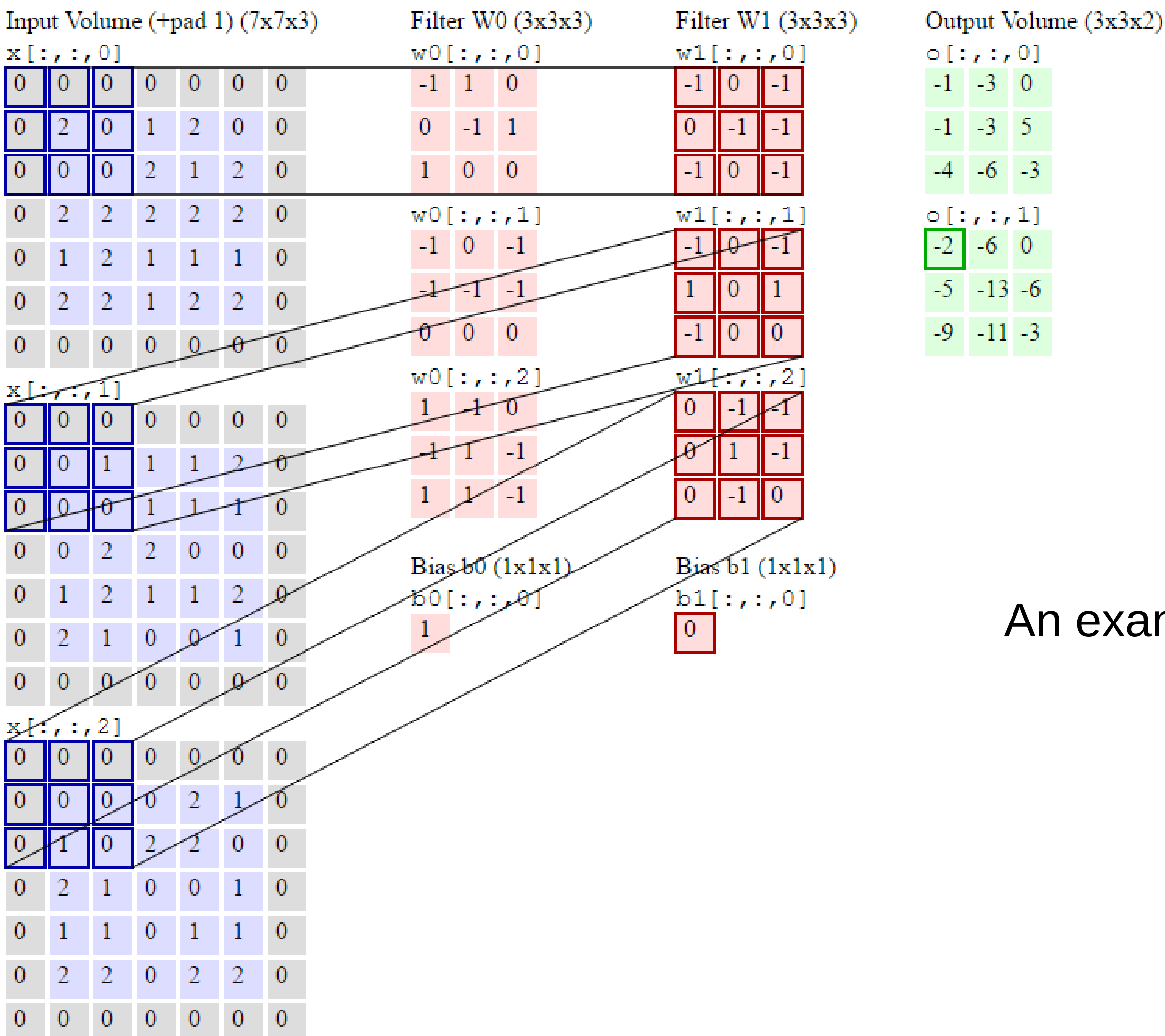
o[:,:,0]

4	3	3
5	12	2
4	7	5

o[:,:,1]

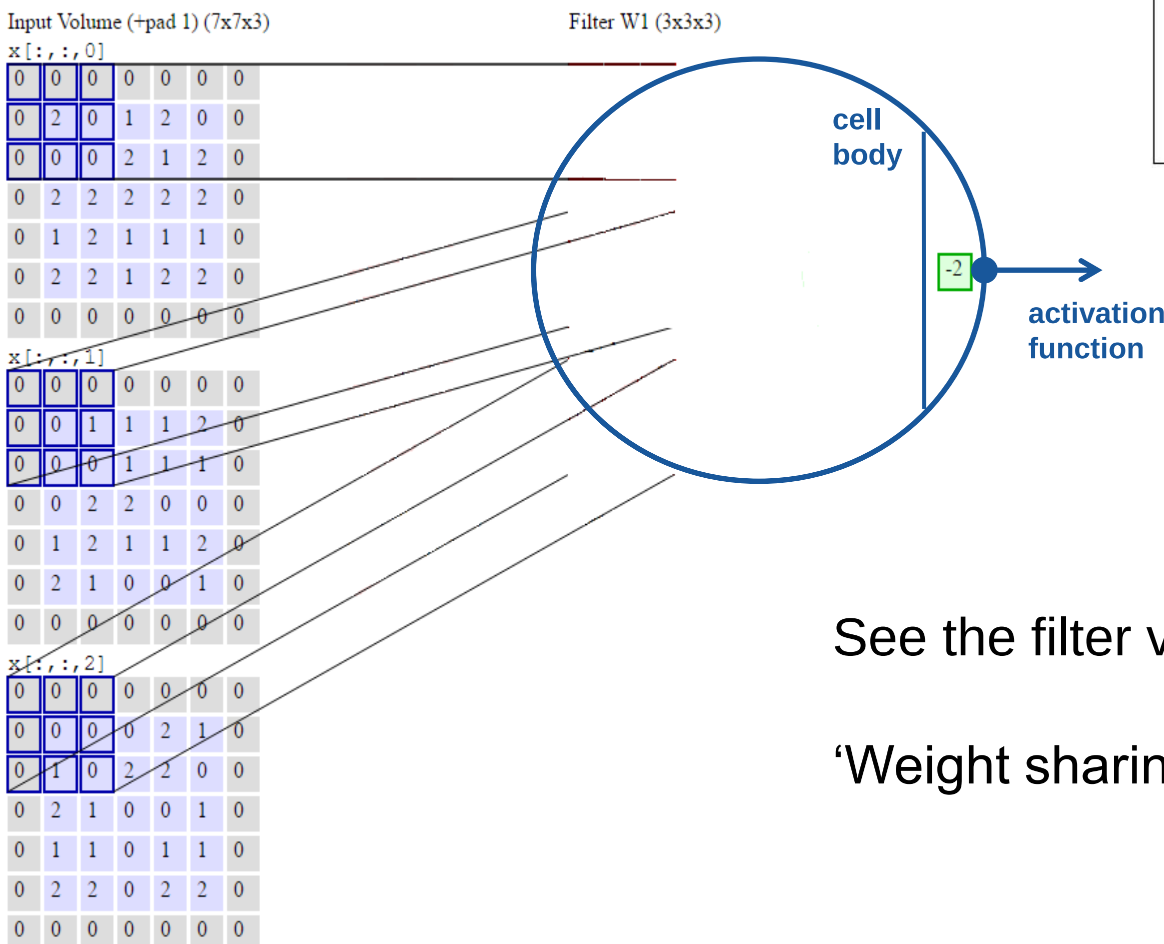
-1	-3	-1
-3	-6	2
7	5	1

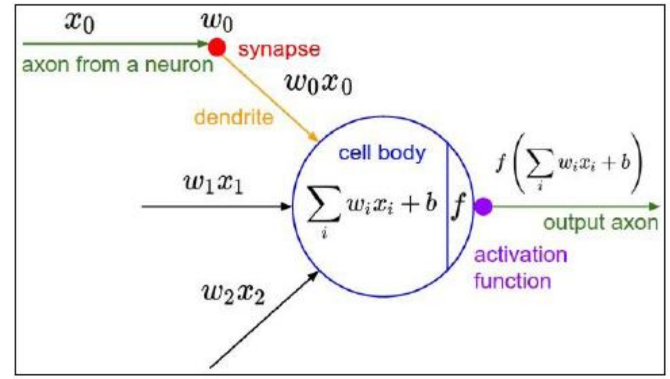
Convolution Demo



An example for second filter.

Convolution Demo



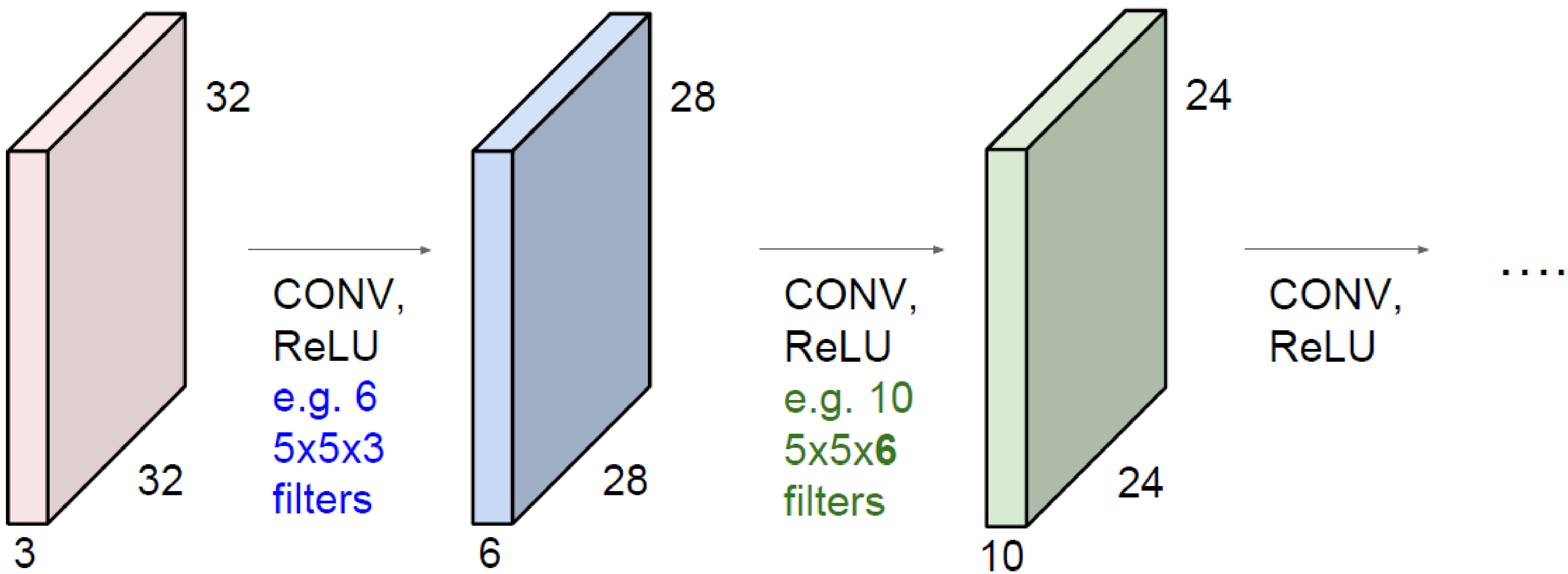


See the filter values as weights.

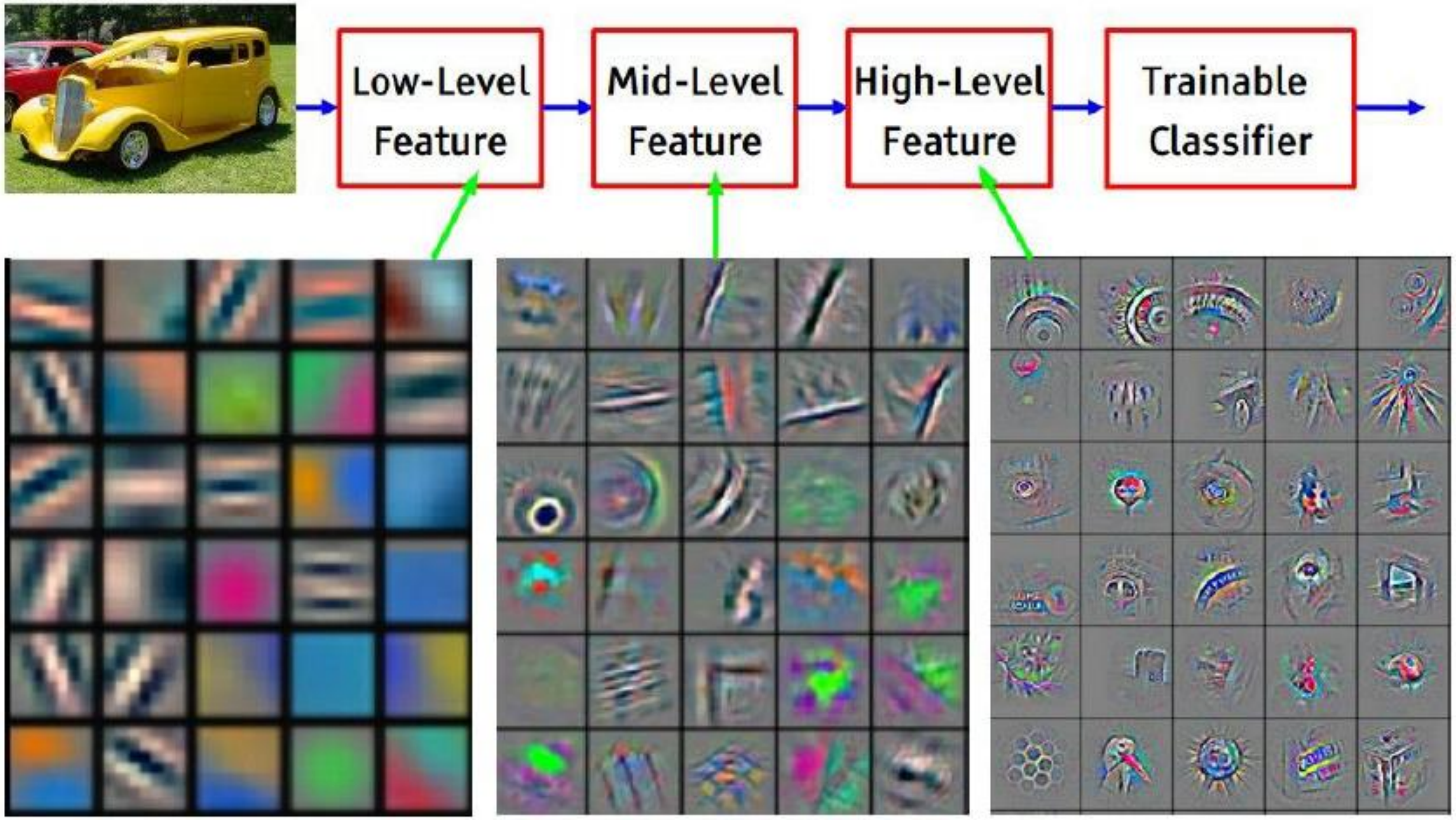
‘Weight sharing’ for all spatial locations.

ConvNet

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



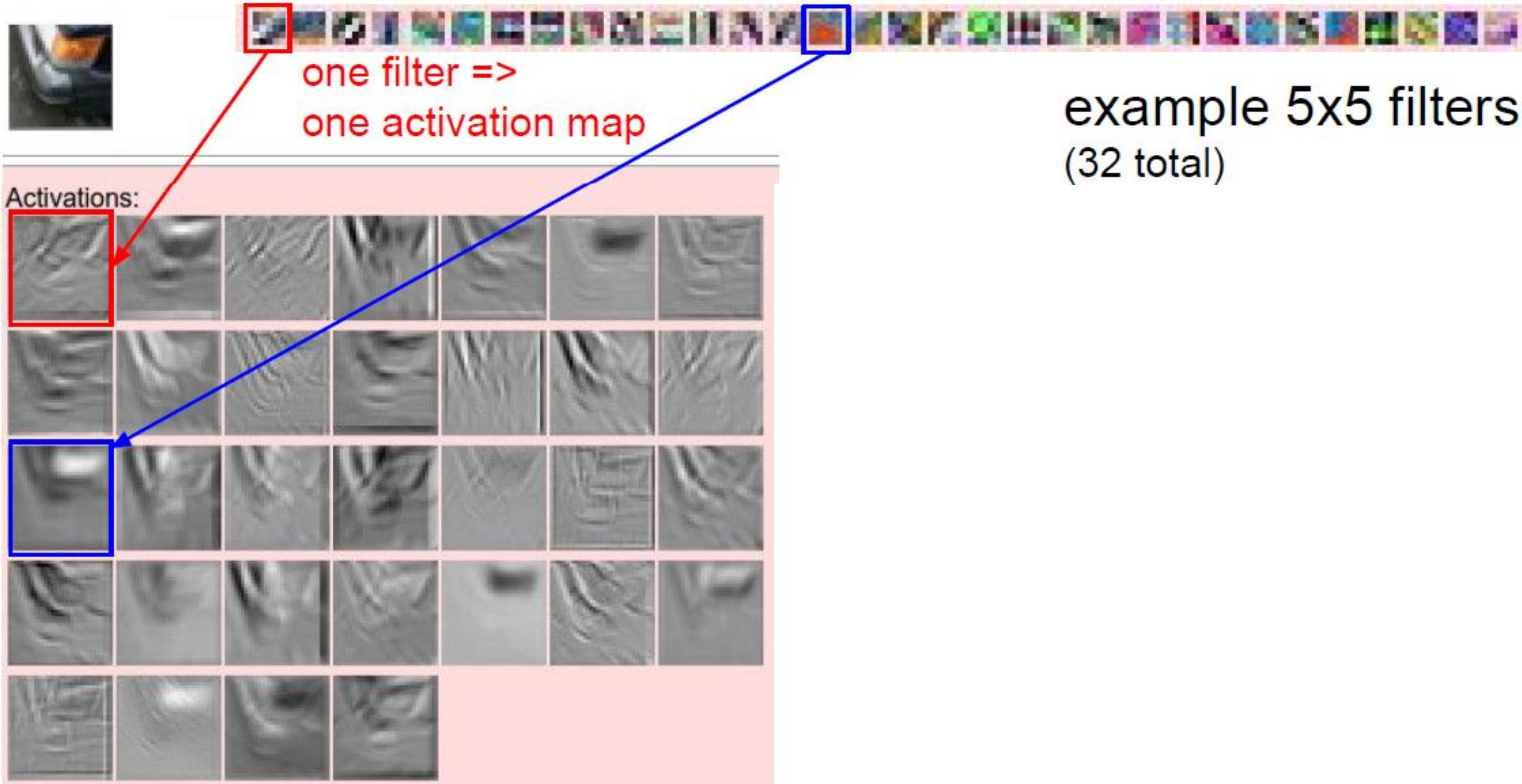
Learned filters



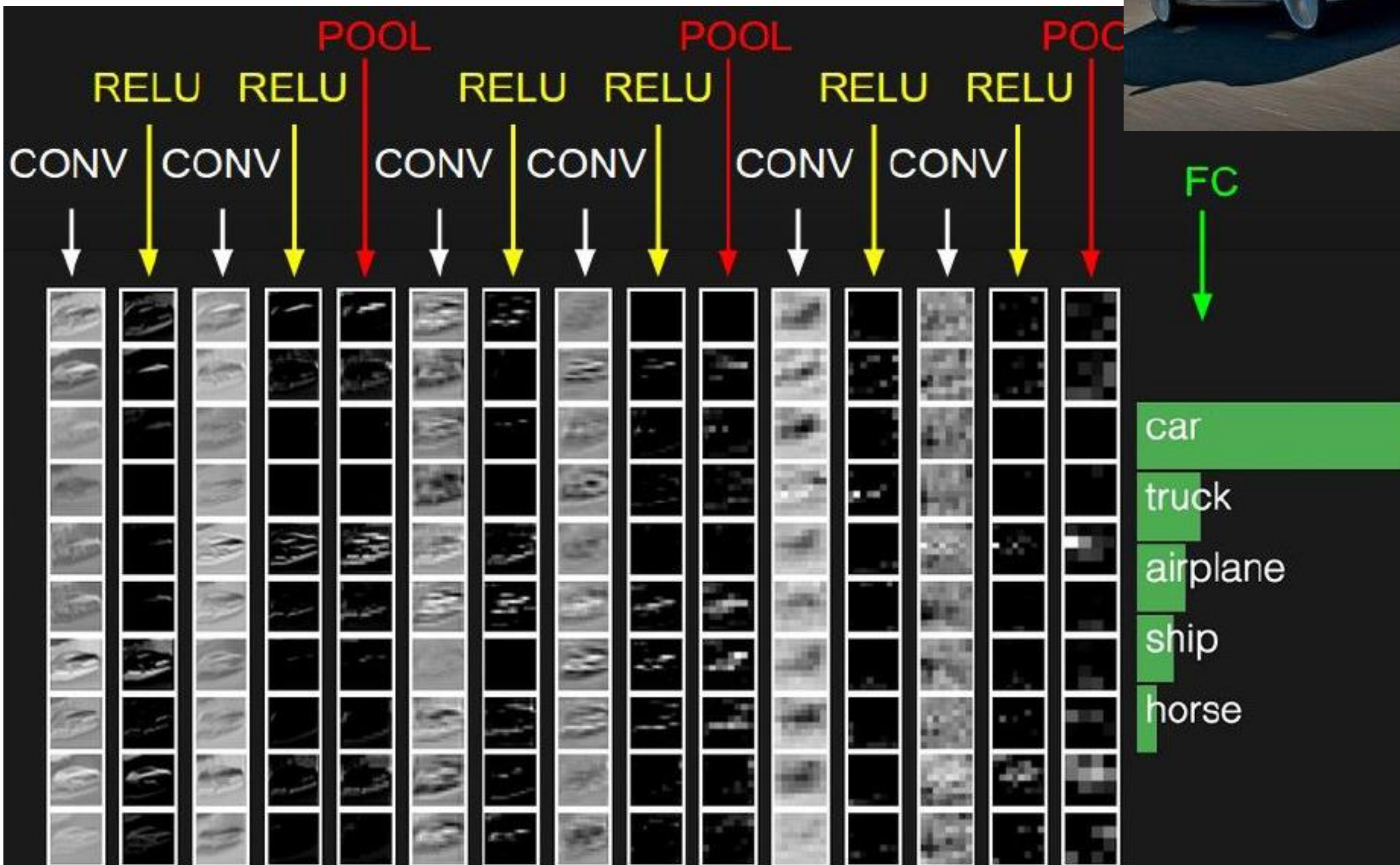
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

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

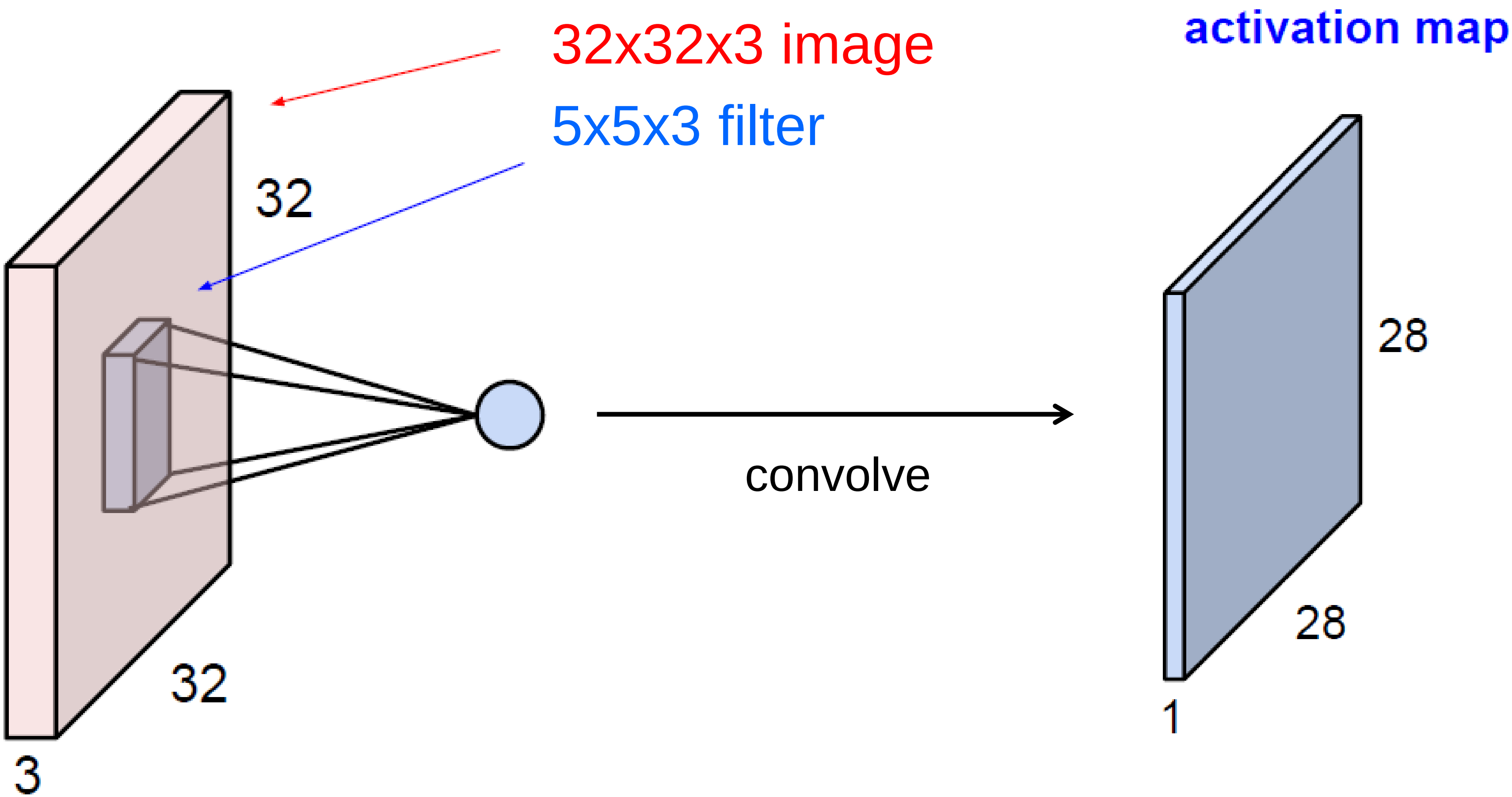


ConvNet: preview

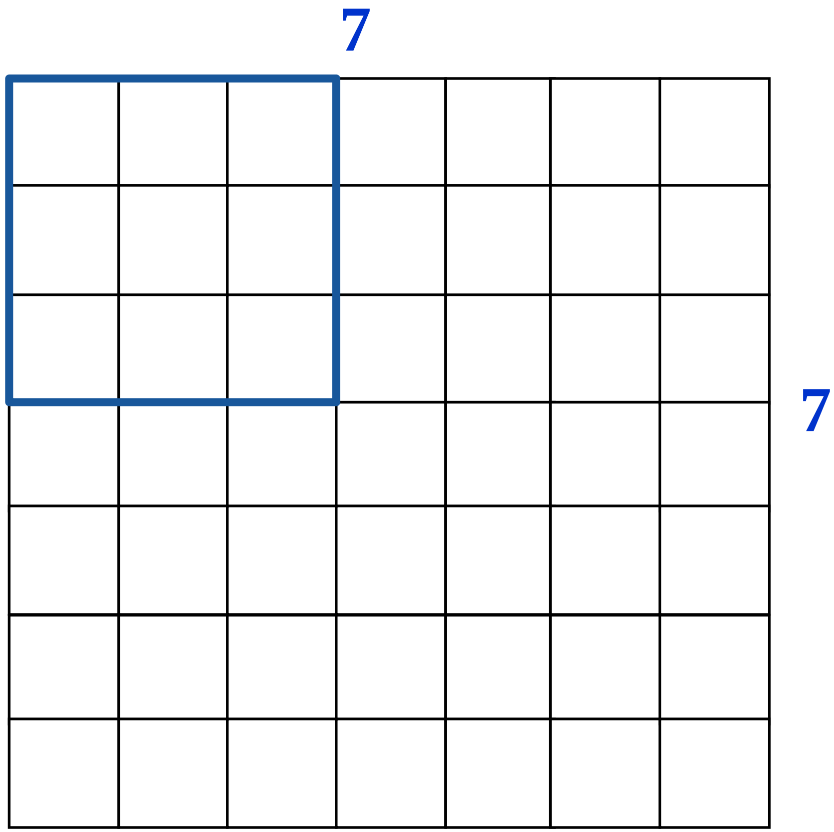


Convolution Layer

A closer look at spatial dimensions

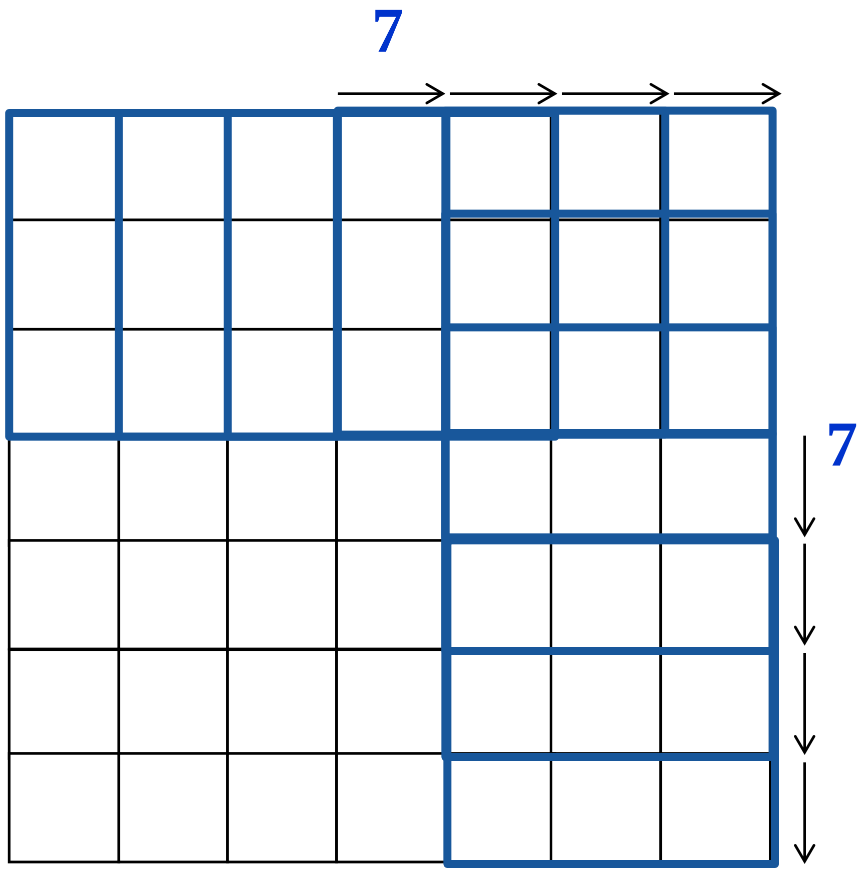


Convolution Layer



Assume,
input image is 7x7
filter is 3x3

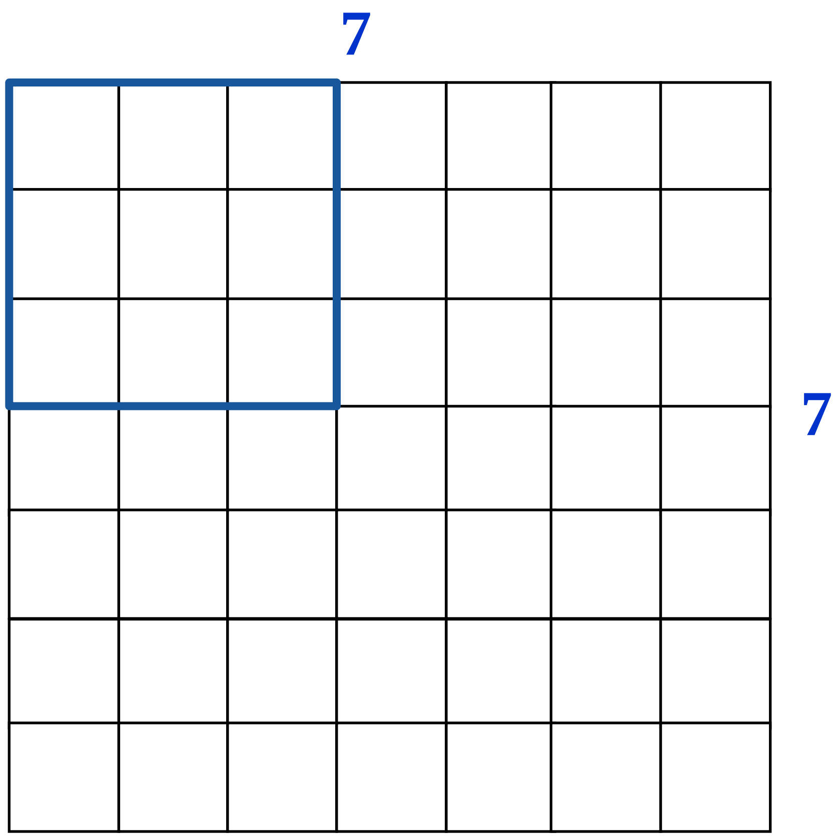
Convolution Layer



Assume,
input image is 7x7
filter is 3x3

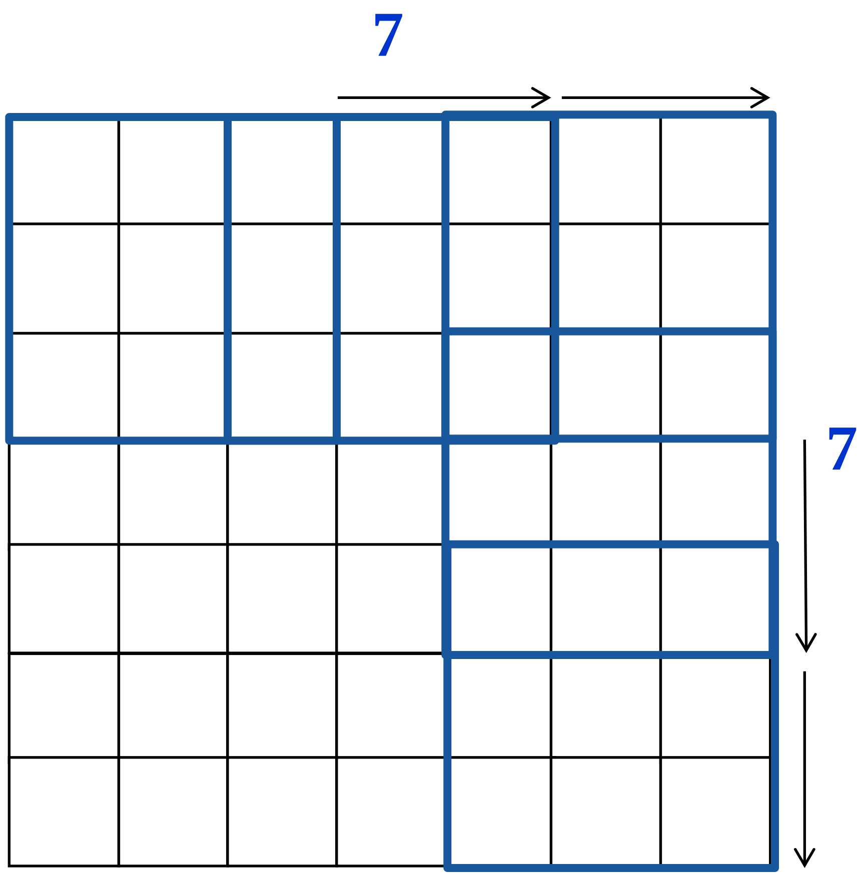
output is 5x5

Convolution Layer



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

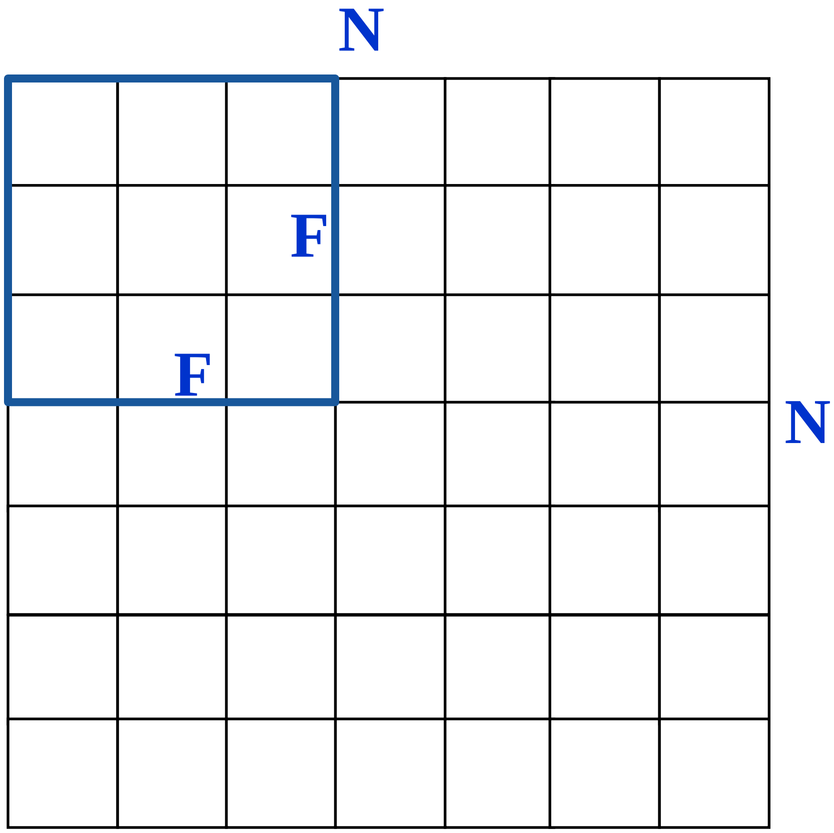
Convolution Layer



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

output is 3x3

Convolution Layer



Output size:

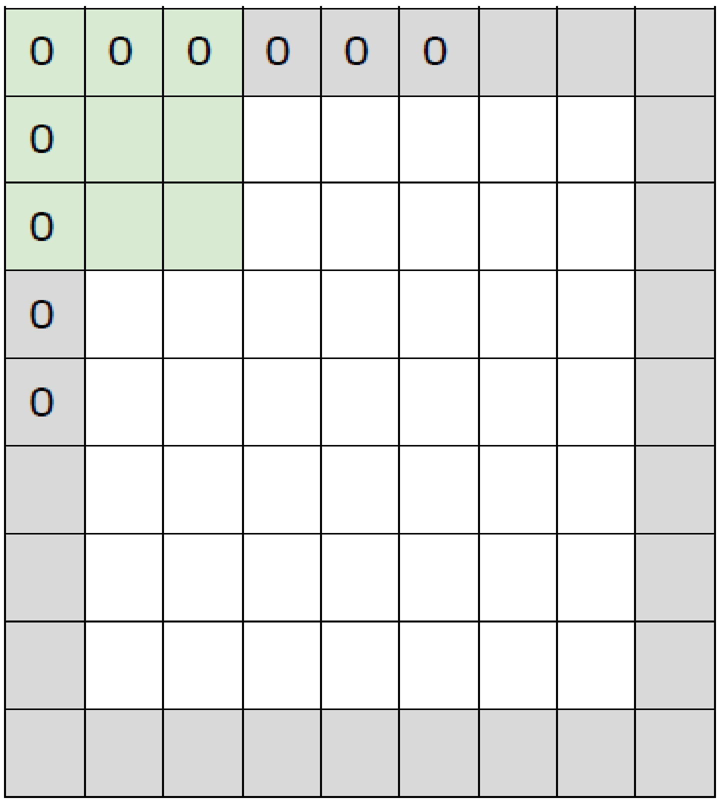
(N-F) / stride +1

Example N = 7, F = 3:

- stride 1=> (7 - 3)/1 + 1 = 5
- stride 2=> (7 - 3)/2 + 1 = 3
- stride 3=> (7 - 3)/3 + 1 = 2.33

:-(stride 3 is not proper for N=7 and F=3

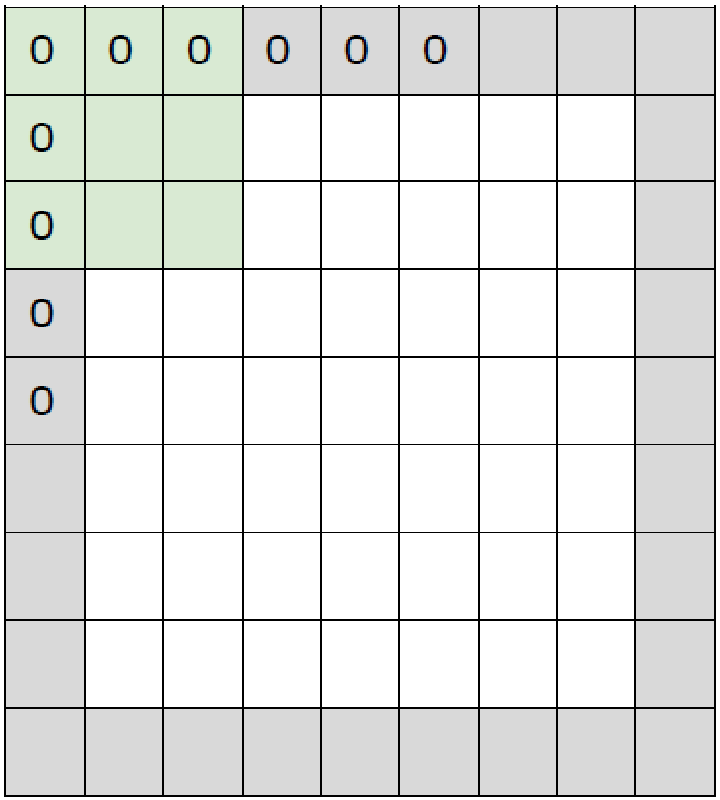
Convolution in practice:
Zero pad the border



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

Output=(N - F) / stride + 1

Convolution in practice: Zero pad the border



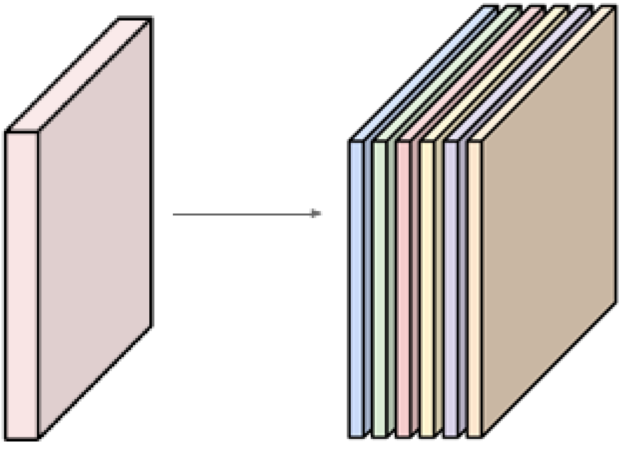
e.g. input 7x7, filter 3x3,
applied with **stride 1, pad** with **1 pixel**
=> what is the output?

7x7 output!

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

E.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Example



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**

Summary

A convolution layer,

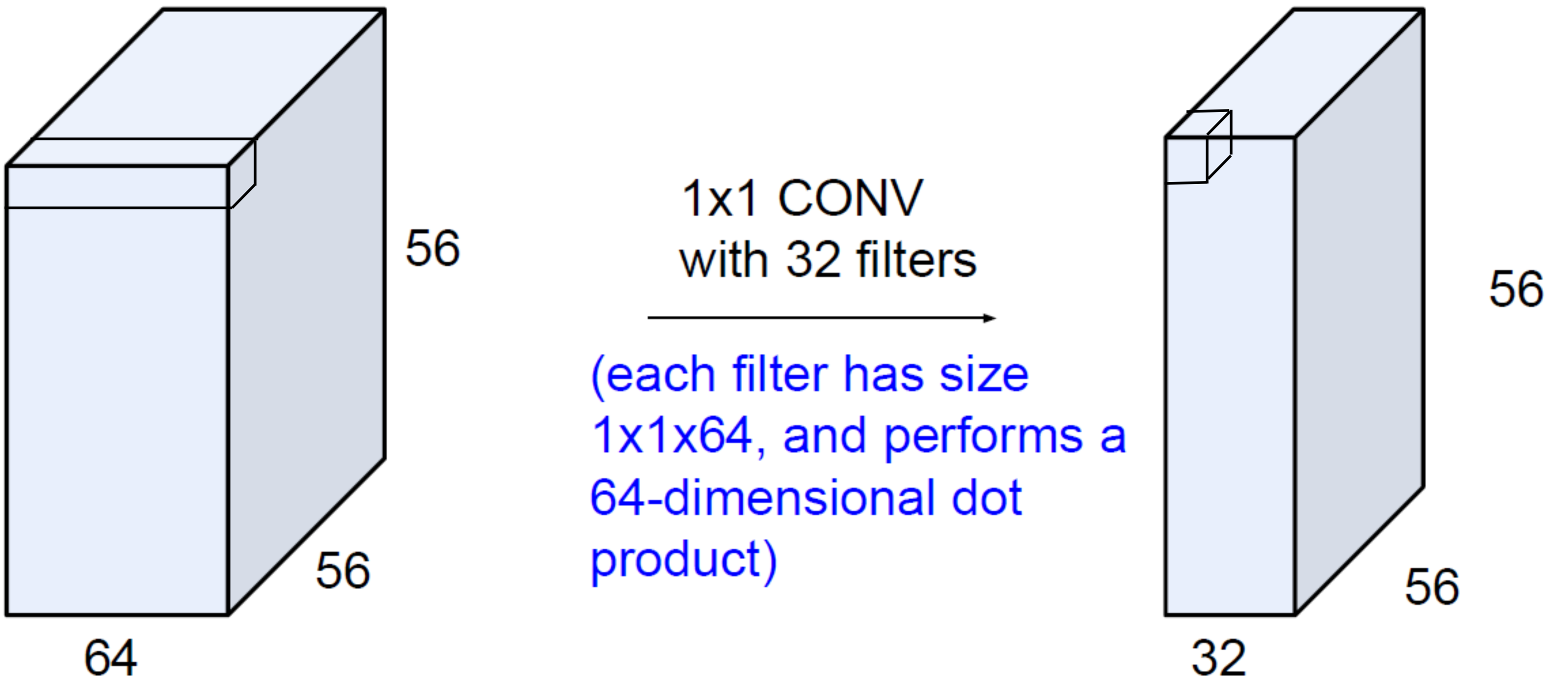
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)

- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

1x1 convolution layers make sense



Example: CONV layer in Torch

SpatialConvolution

```
module = nn.SpatialConvolution(nInputPlane, nOutputPlane, kW, kH, [dW], [dH], [padW], [padH])
```

Applies a 2D convolution over an input image composed of several input planes. The `input` tensor in `forward(input)` is expected to be a 3D tensor (`nInputPlane x height x width`).

The parameters are the following:

- `nInputPlane` : The number of expected input planes in the image given into `forward()` .
- `nOutputPlane` : The number of output planes the convolution layer will produce.
- `kW` : The kernel width of the convolution
- `kH` : The kernel height of the convolution
- `dW` : The step of the convolution in the width dimension. Default is `1` .
- `dH` : The step of the convolution in the height dimension. Default is `1` .
- `padW` : The additional zeros added per width to the input planes. Default is `0` , a good number is `(kW-1)/2` .
- `padH` : The additional zeros added per height to the input planes. Default is `padW` , a good number is `(kH-1)/2` .

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

Note that depending of the size of your kernel, several (of the last) columns or rows of the input image might be lost. It is up to the user to add proper padding in images.

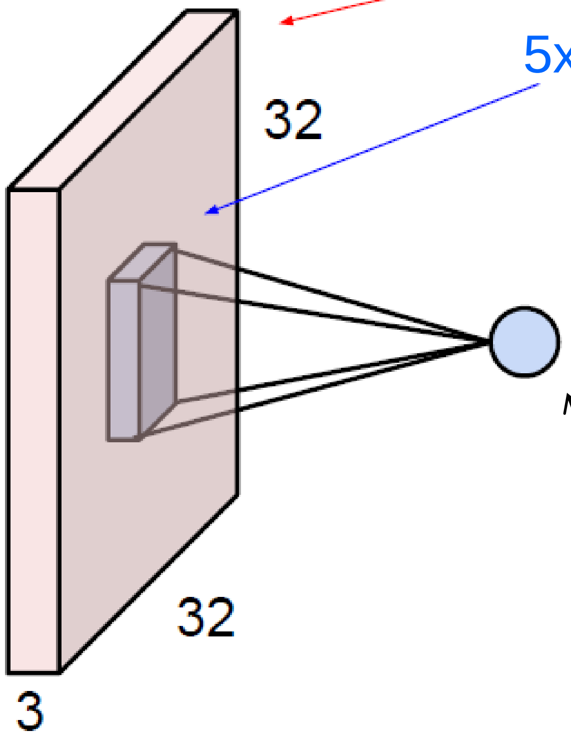
If the input image is a 3D tensor `nInputPlane x height x width` , the output image size will be `nOutputPlane x oheight x owidth` where

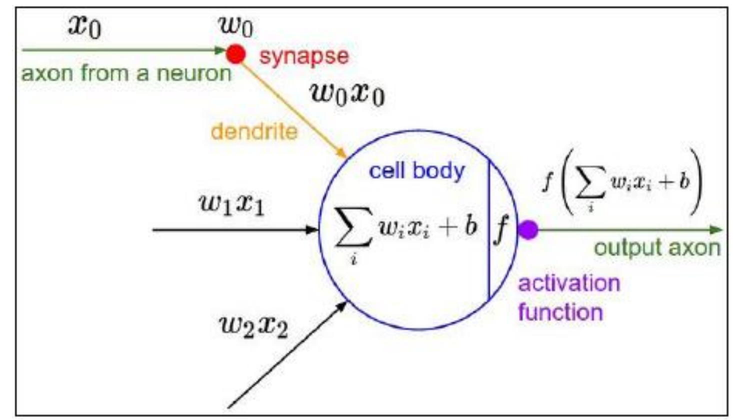
```
owidth  = floor((width  + 2*padW - kW) / dW + 1)
oheight = floor((height + 2*padH - kH) / dH + 1)
```

The brain/neuron view of CONV Layer

32x32x3 image

5x5x3 filter



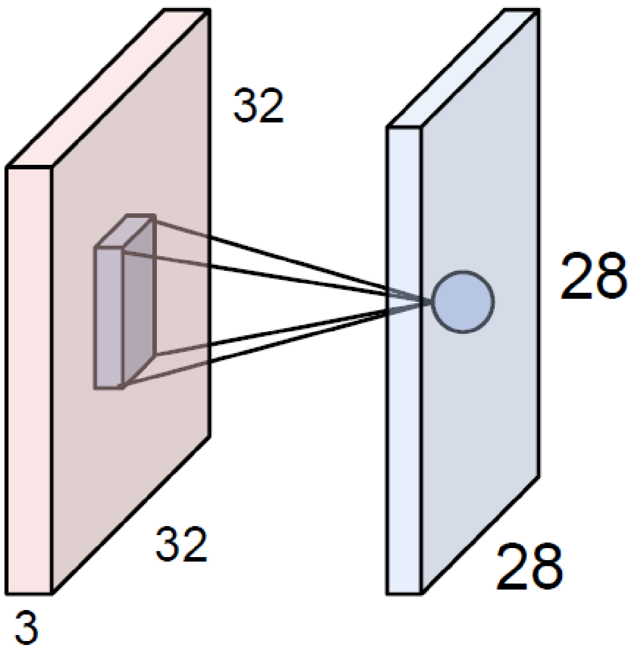


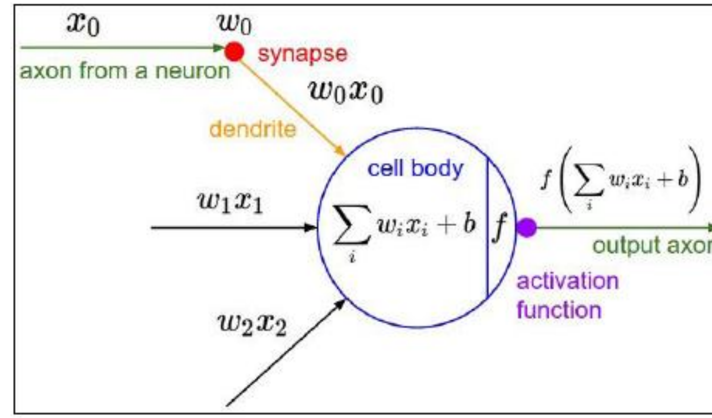
It's just a neuron with local connectivity!

Result is a number

(5*5*3 = 75-dimensional dot product)

The brain/neuron view of CONV Layer



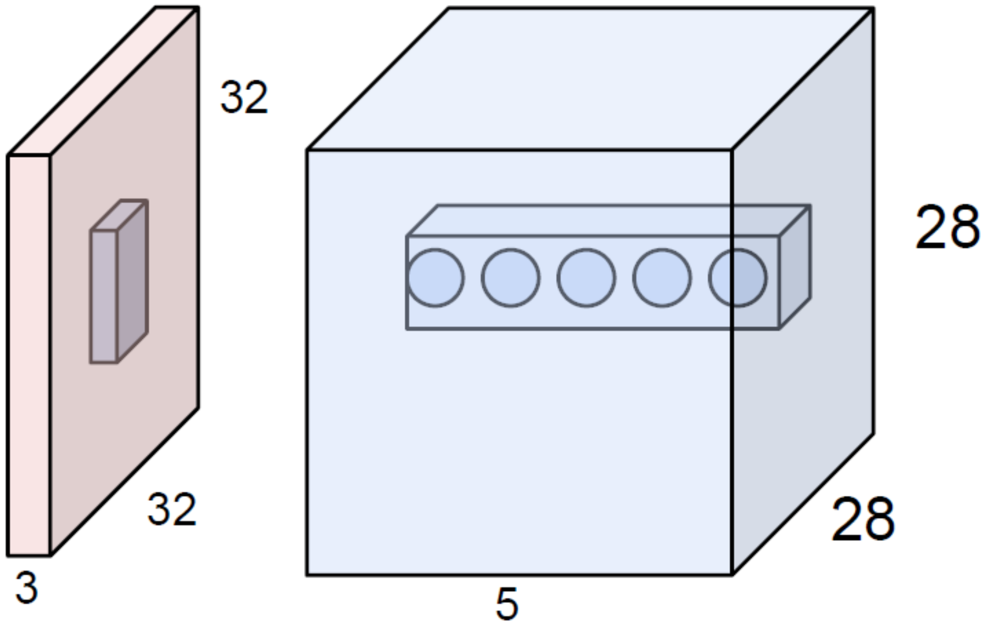


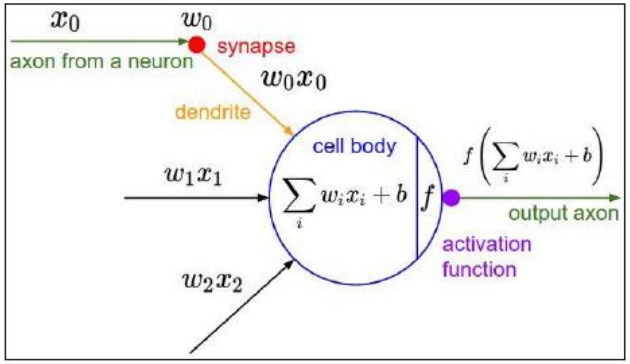
An activation map is a 28x28 sheet of neuron outputs:

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



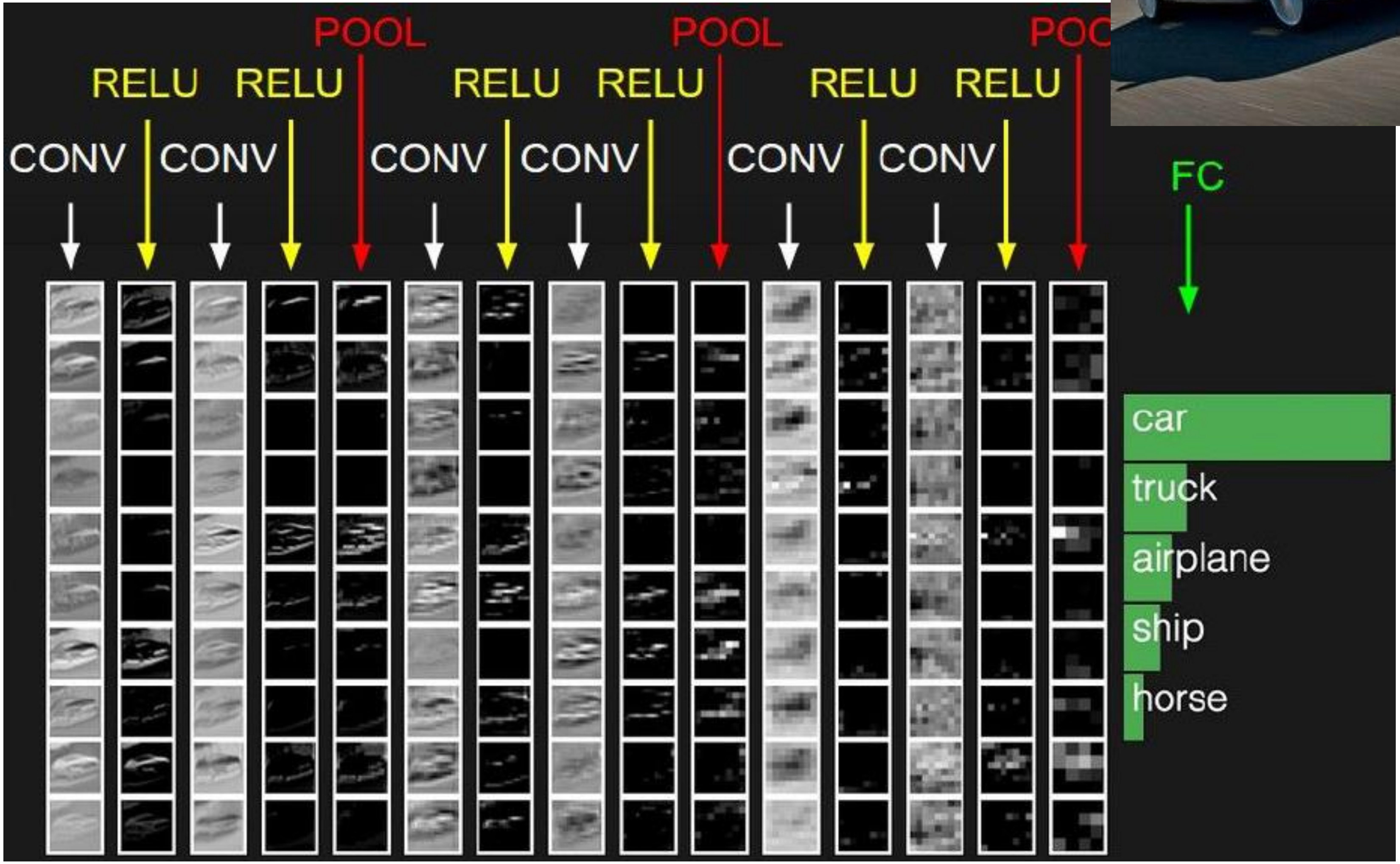


E.g. with 5 filters,

CONV layer consists of neurons arranged in a 3D grid (28x28x5).

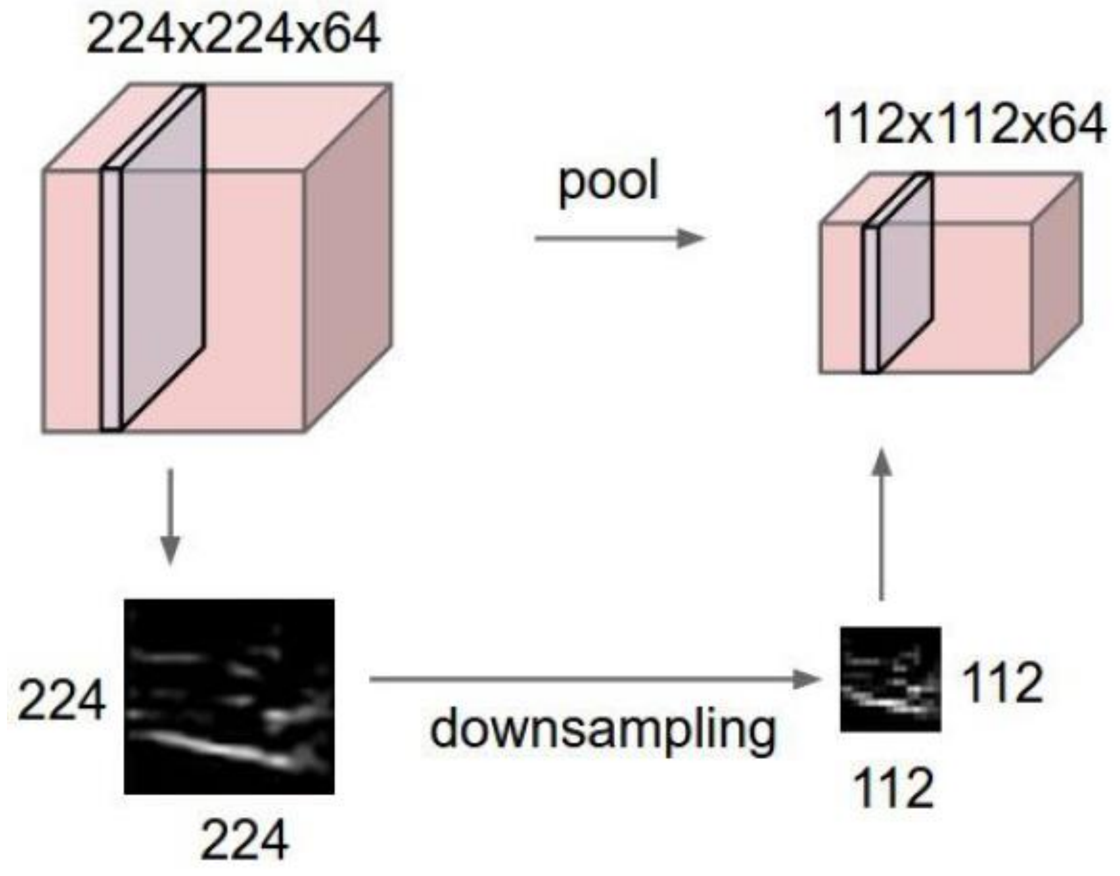
There will be 5 different neurons all looking at the same region in the input volume.

Two more layers to go: POOL and FC

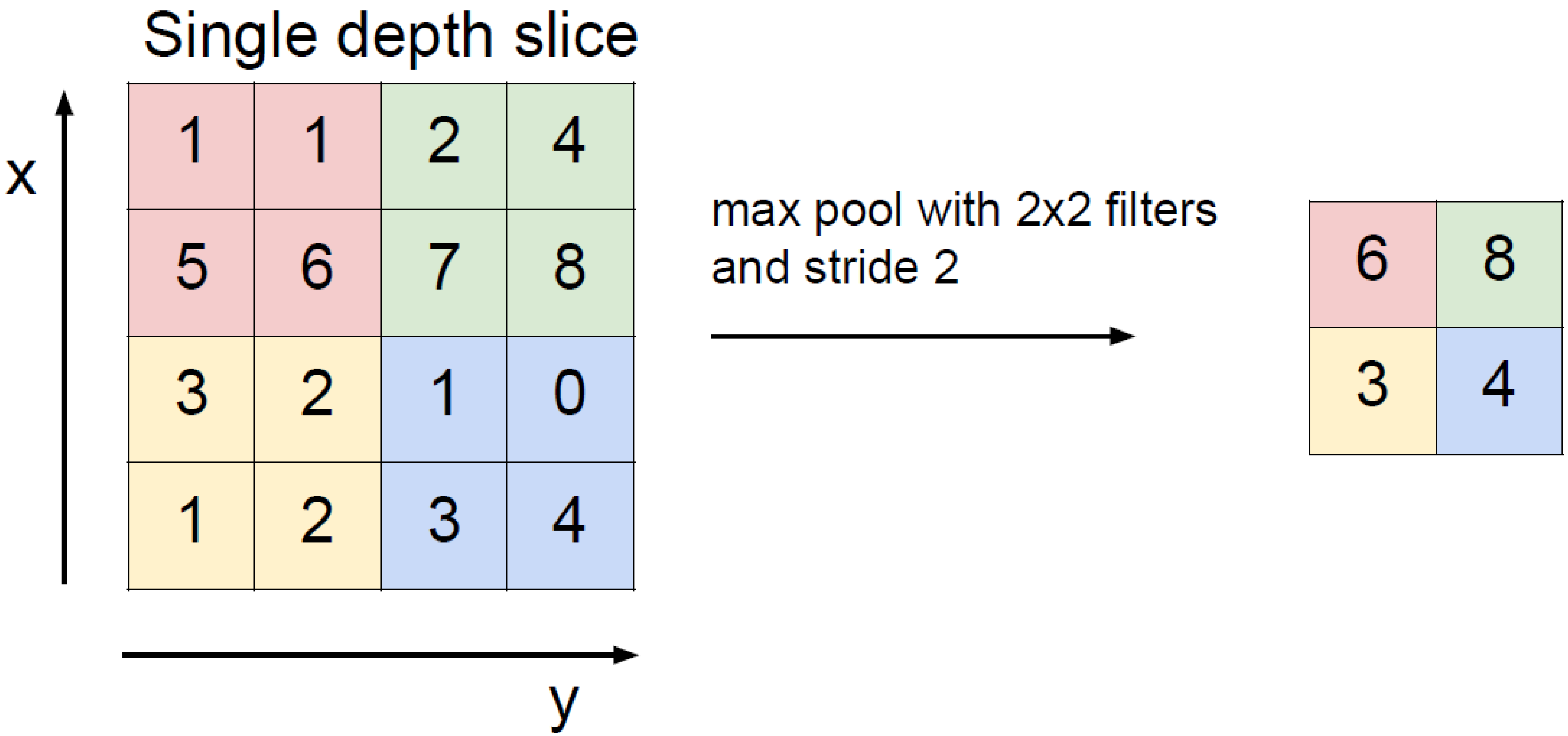


Pooling Layer

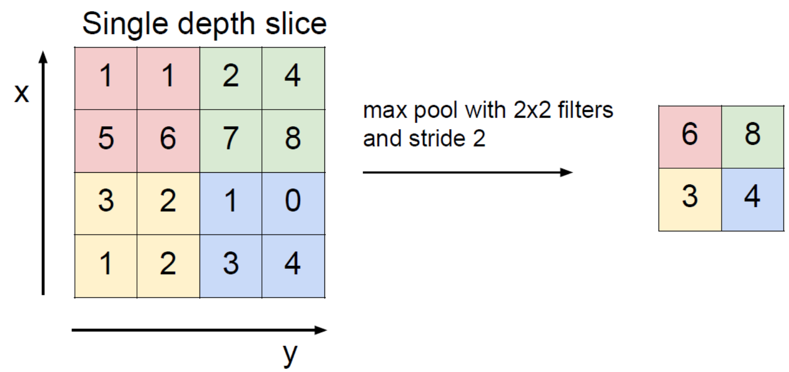
- makes the representations smaller and more manageable
- operates over each activation map independently:



Max Pooling


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Max Pooling



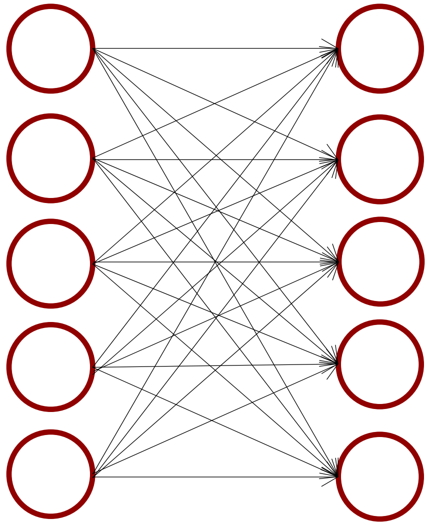
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
    - their spatial extent F,
    - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
    - $W_2 = (W_1 - F)/S + 1$
    - $H_2 = (H_1 - F)/S + 1$
    - $D_2 = D_1$
- Introduces zero parameters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

Common settings:

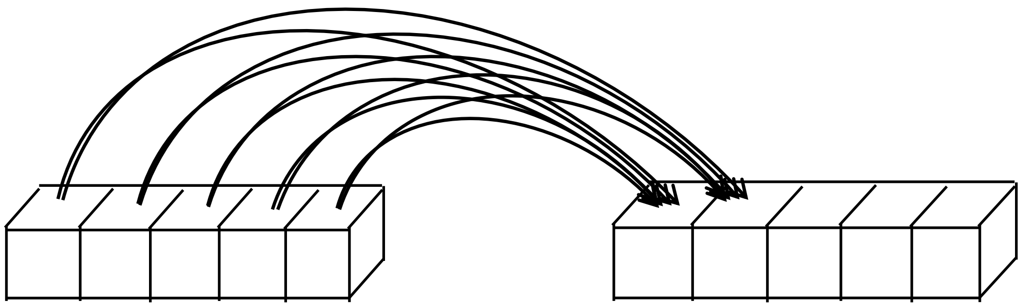
F = 2, S = 2
F = 3, S = 2

Fully Connected Layer

Contains neurons that connect to the entire input volume, as in ordinary Neural Networks.



FC layers still compute dot products. It's possible to convert between FC and CONV layers.



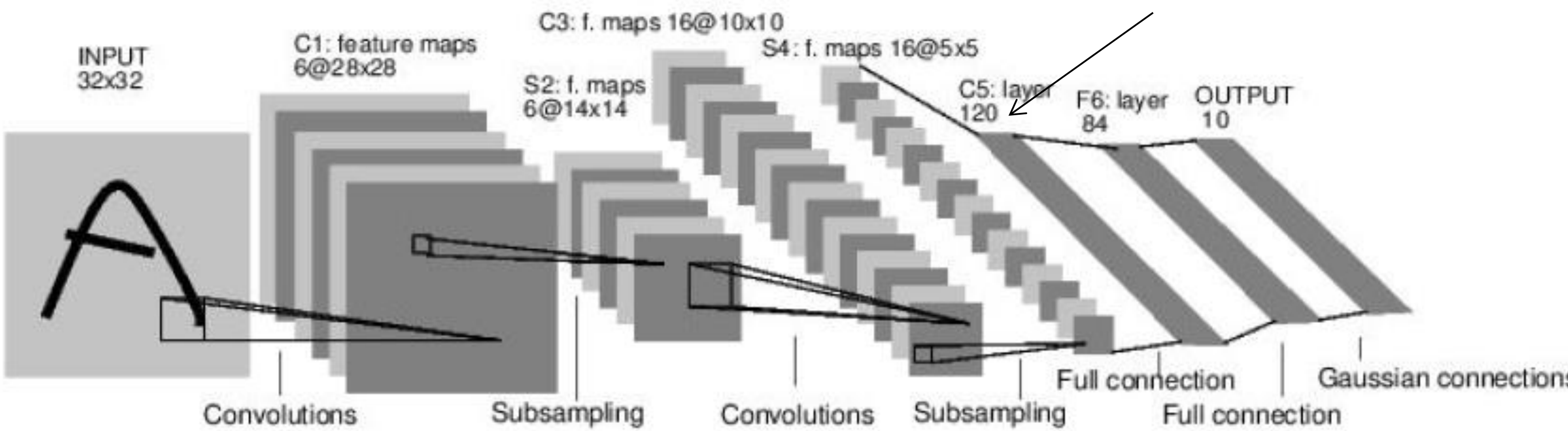
Case Study: LeNet-5 [LeCun et al. 1998]

Conv filters were 5x5, applied at stride 1

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

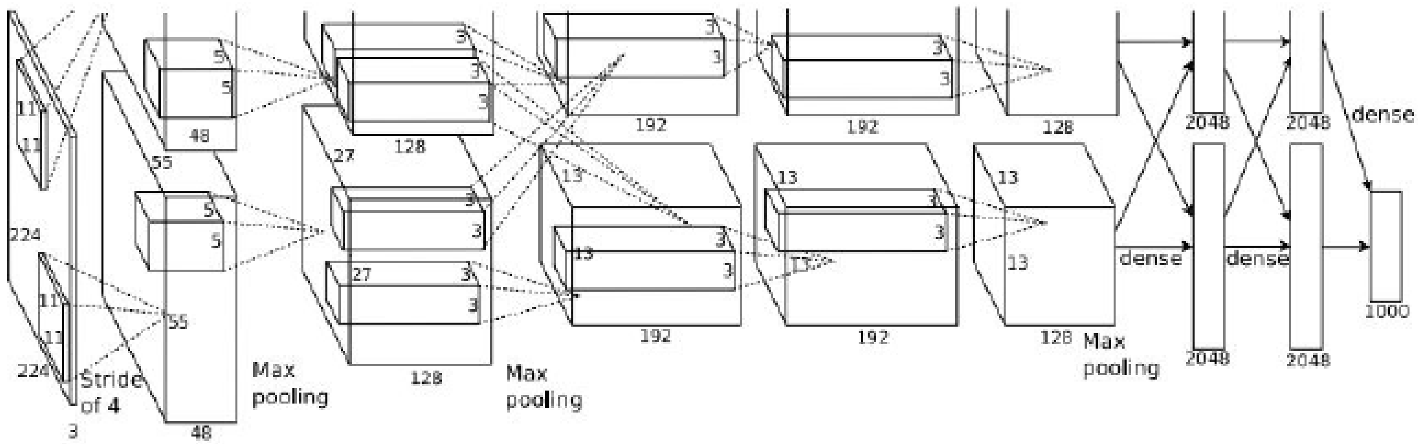
Architecture is [CONV-POOL-CONV-POOL-CONV-FC]

You can see this layer as 120@1x1
obtained with 120 5x5x16 filters



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

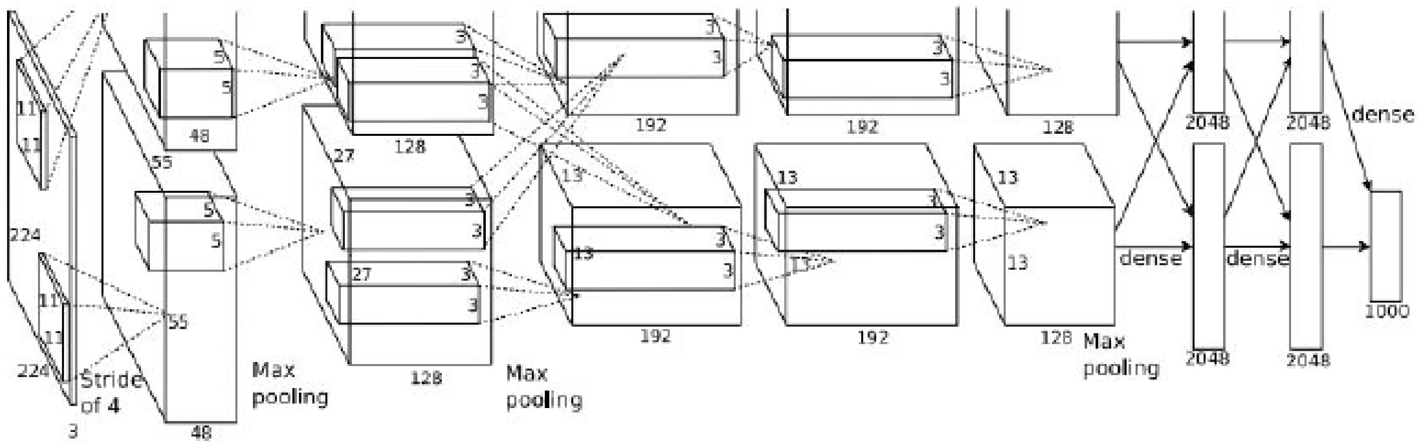
Output = 55 = (227-11)/4+1 = (N-F)/stride+1

Output volume = **[55x55x96]**

Number of parameters in this layer = (11*11*3)*96 = **35K**

without weight sharing it'd be 105M

Case Study: AlexNet [Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

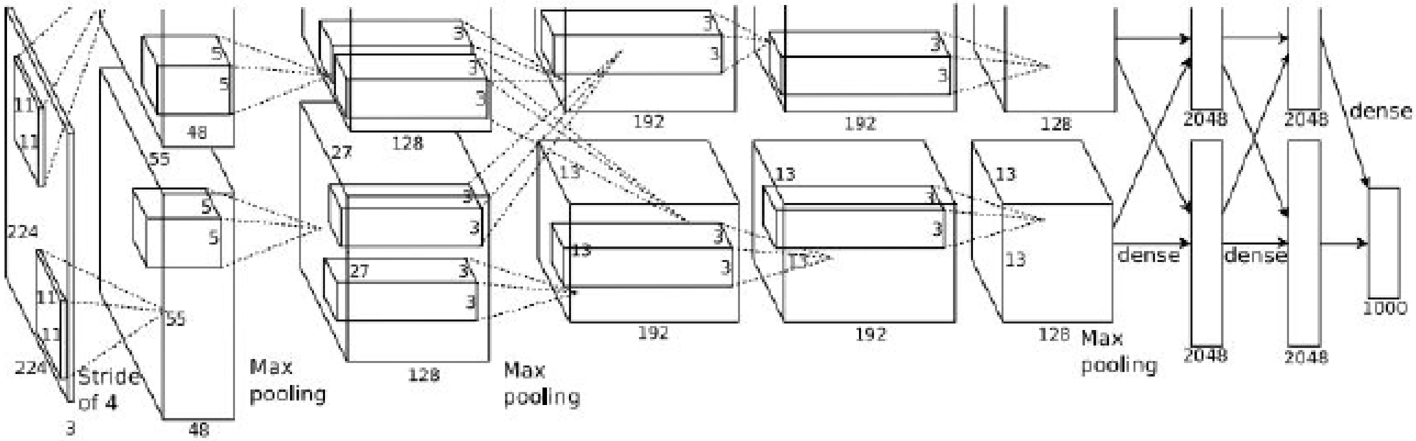
Second layer (POOL1): 3x3 filters applied at stride 2

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]

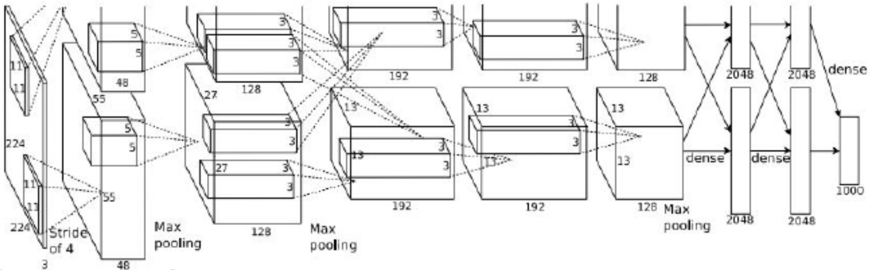


Input: 227x227x3 images

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

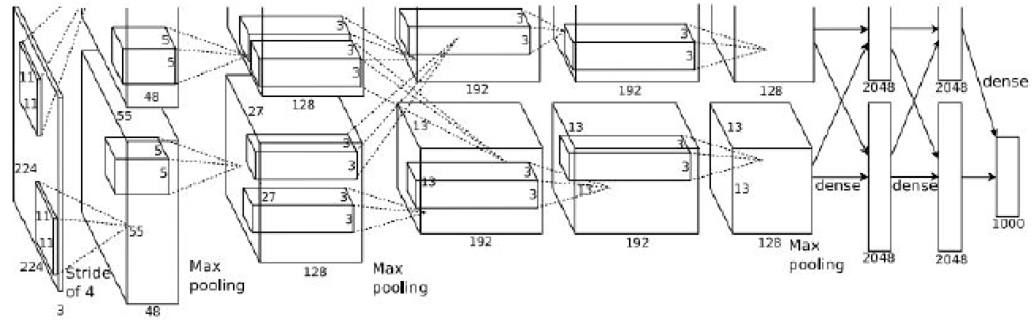
[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)

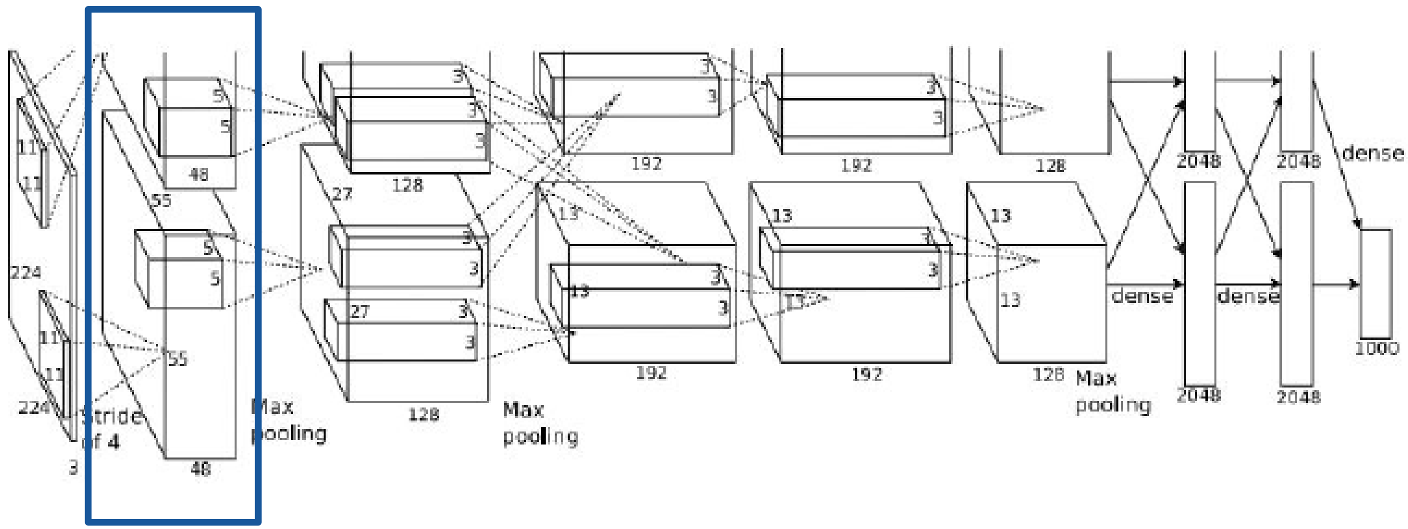
Case Study: AlexNet [Krizhevsky et al. 2012]



Details/Retrospectives:

- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- 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%)

Case Study: AlexNet [Krizhevsky et al. 2012]

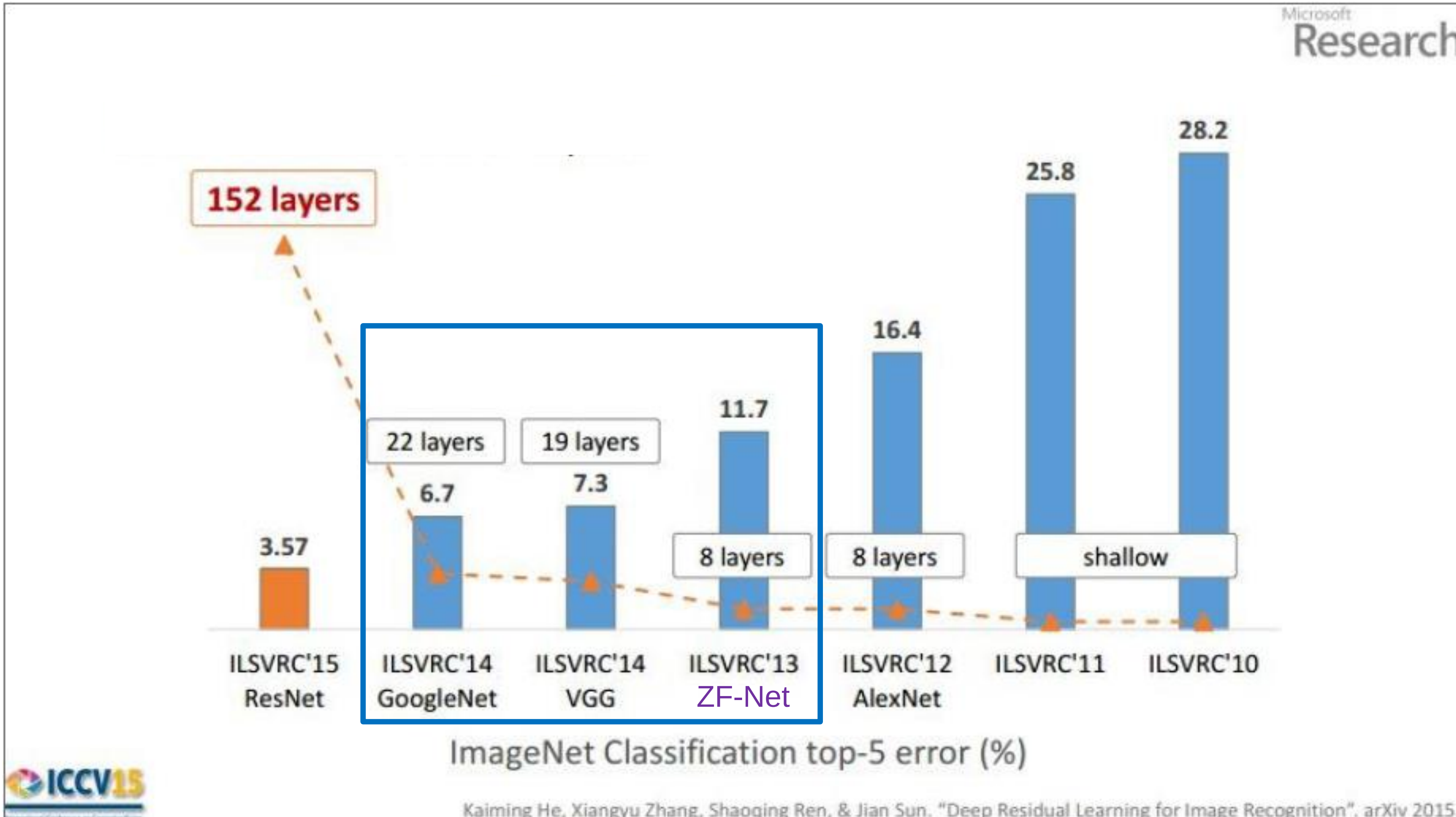


[55x55x48] x2

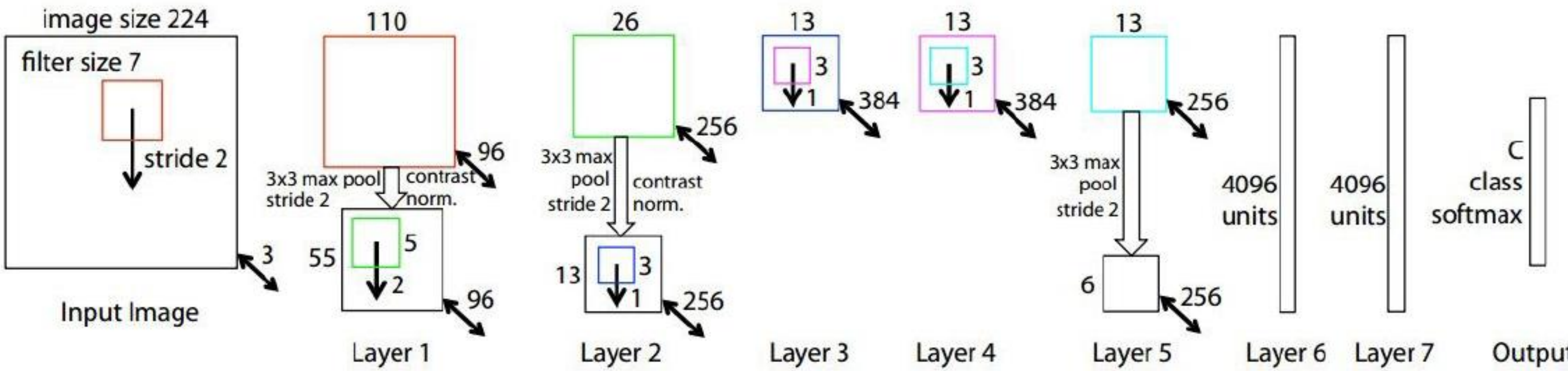
Historical Note:

- Trained on GTX 580 GPU with only 3 GB of memory.
- Network spread across 2 GPUs, half the neurons (feature maps) on each GPU.

ILSVRC (ImageNet Challenge) winners



Case Study: ZFNet [Zeiler and Fergus, 2013]



Similar to AlexNet, but:

- CONV1: change from (11x11 stride 4) to (7x7 stride 2)
- CONV3,4,5: instead of 384,384,256 use 512,1024,512
- ImageNet top-5-error 13.5%

Case Study: VGGNet [Simonyan and Zisserman, 2014]

Small filters, deeper network

Only 3x3 CONV (stride 1, pad 1)

and 2x2 Max Pool (stride2)

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: VGGNet [Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

A: Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

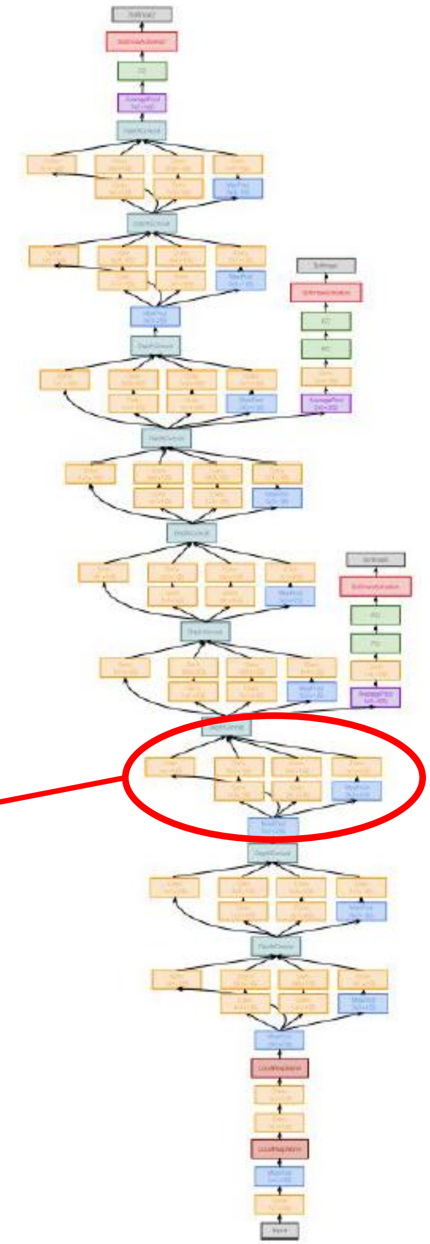
But deeper (more non-linearities) and
has fewer parameters:
$3*(3^2C^2)$ vs. 7^2C^2
(C channels per layer)

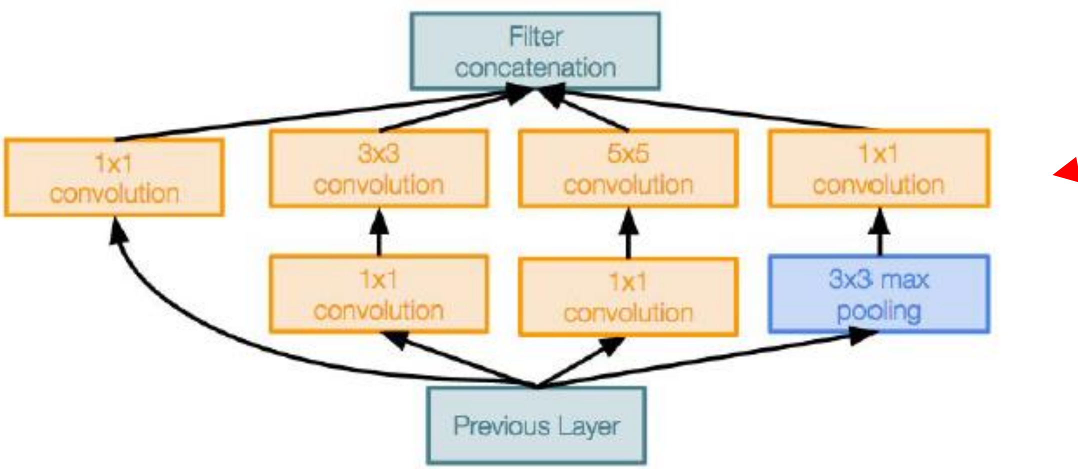
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	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)





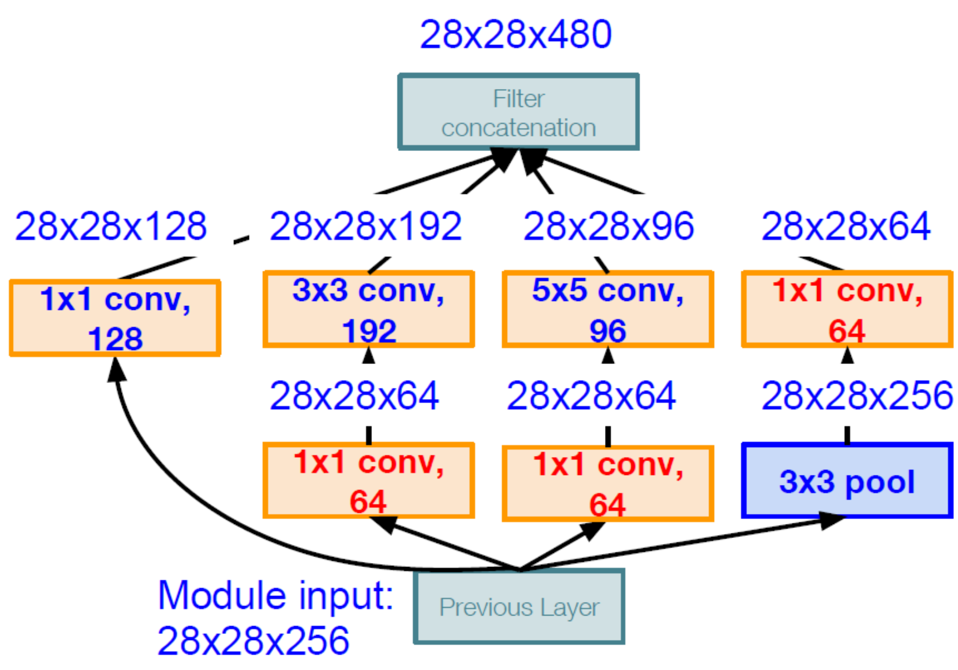
Inception module

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

Inception module:

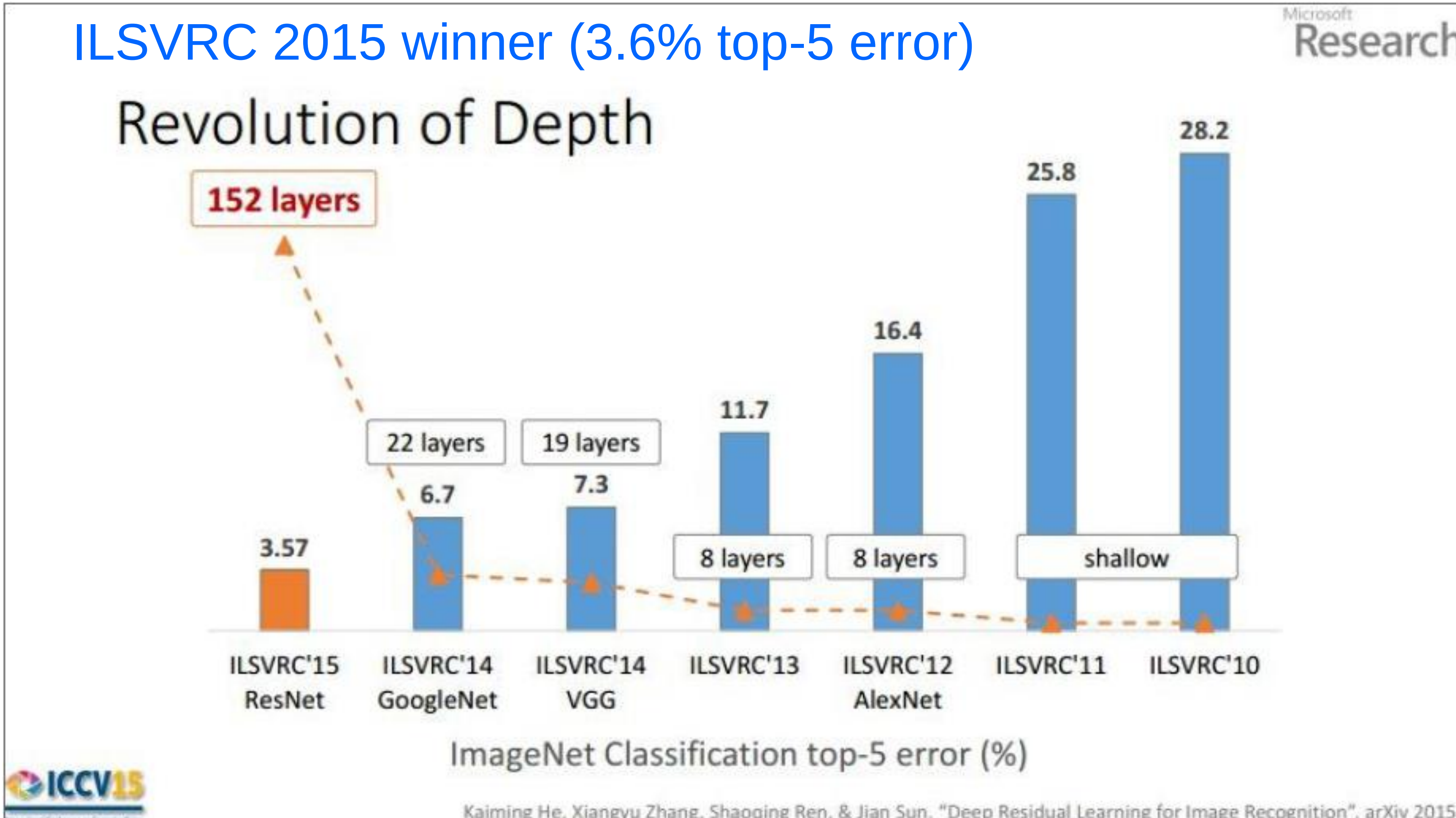
- Apply parallel filter operations on the input from previous layer
  - Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
  - Pooling operation (3x3)
- Concatenate all filter outputs together depth-wise

An example module with sizes:



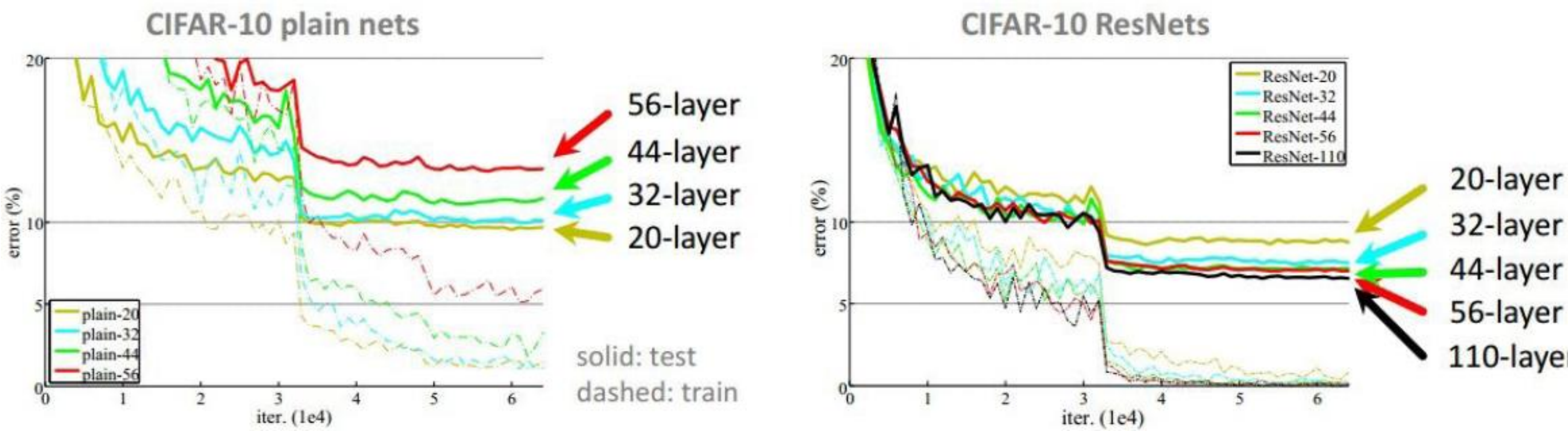
Case Study: ResNet [He et al., 2015]

ILSVRC 2015 winner (3.6% top-5 error)



(slide from Kaiming He's recent presentation)

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



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.

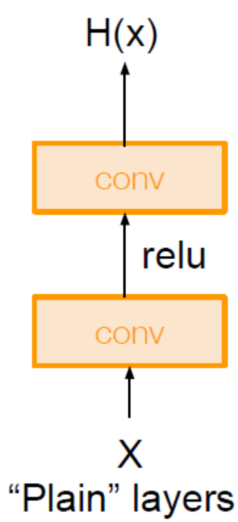
The deeper model should be able to perform at least as well as the shallower model.

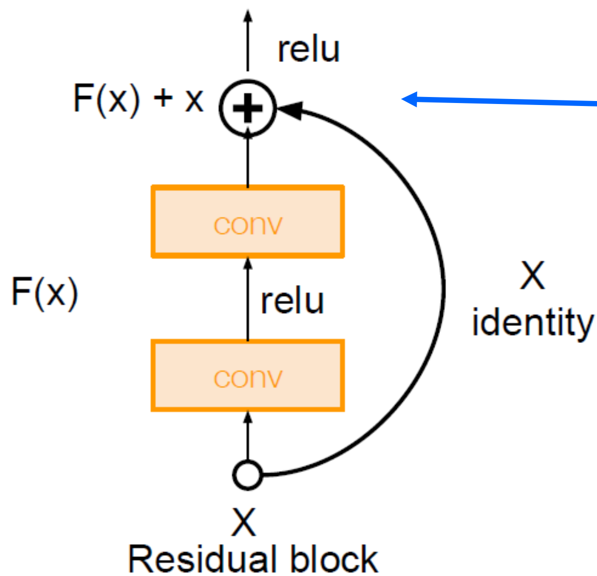
A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

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).





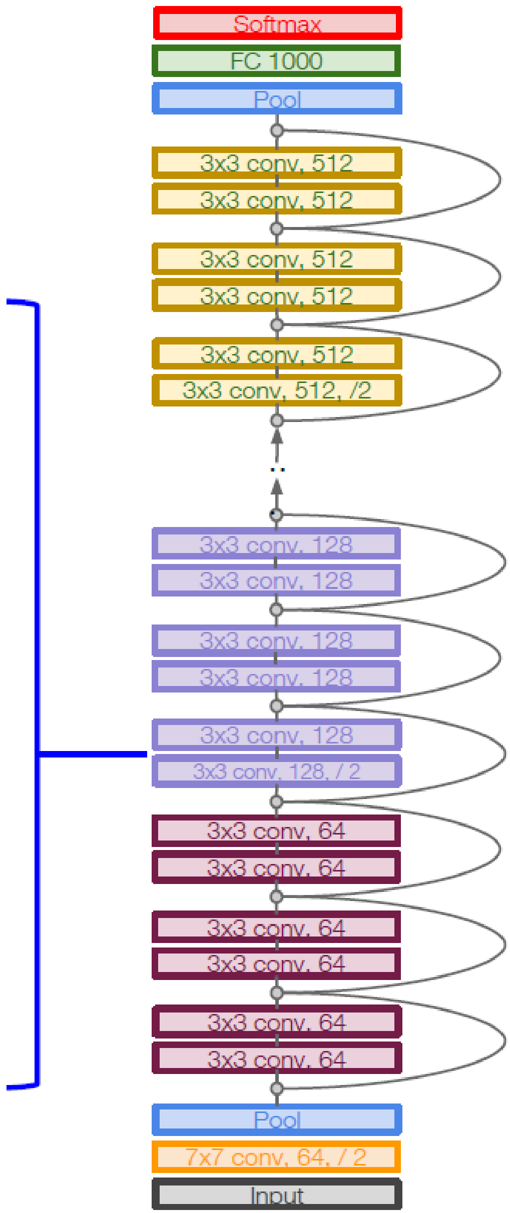
H(x) = F(x) + x

Idea:
Learn residual
F(x) = H(x) - x
instead of H(x)

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)



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

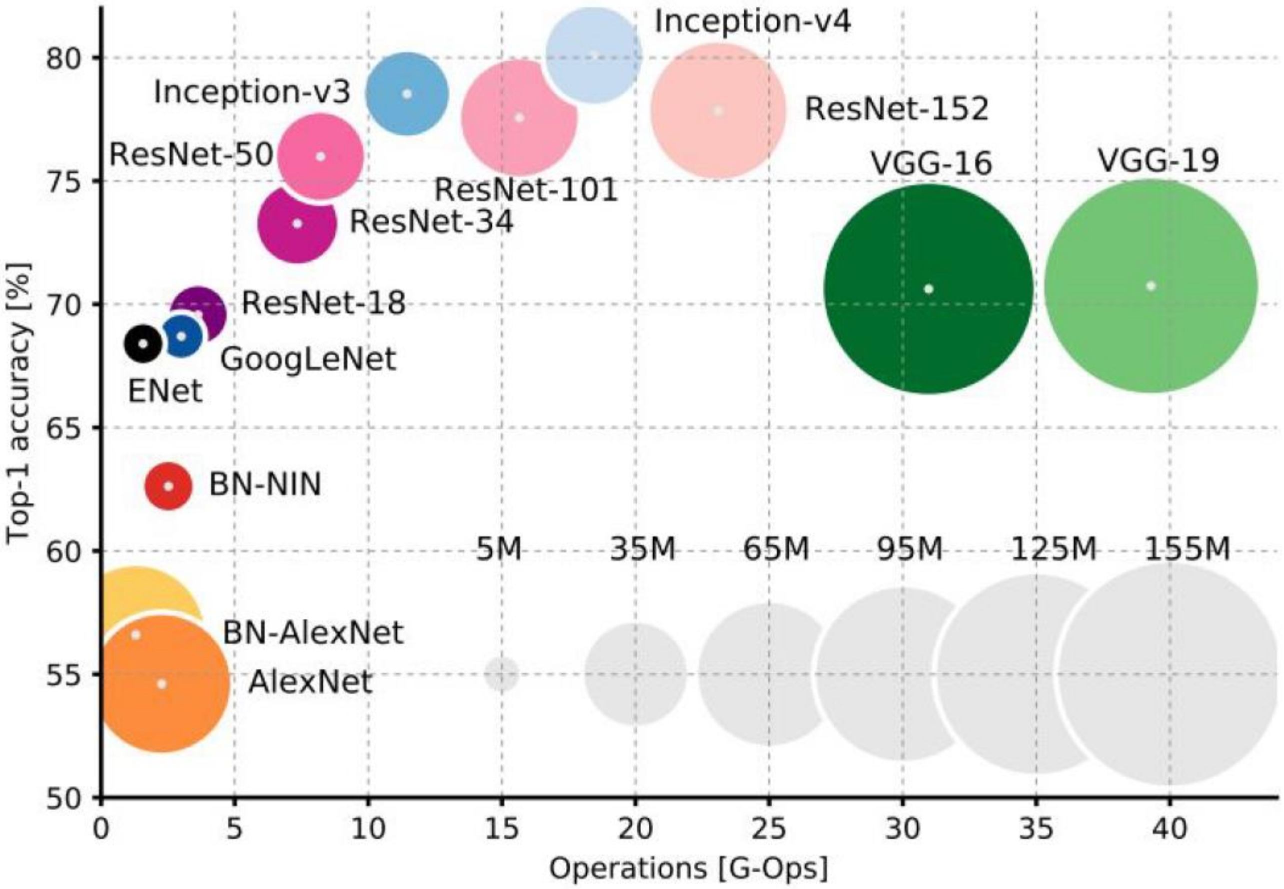
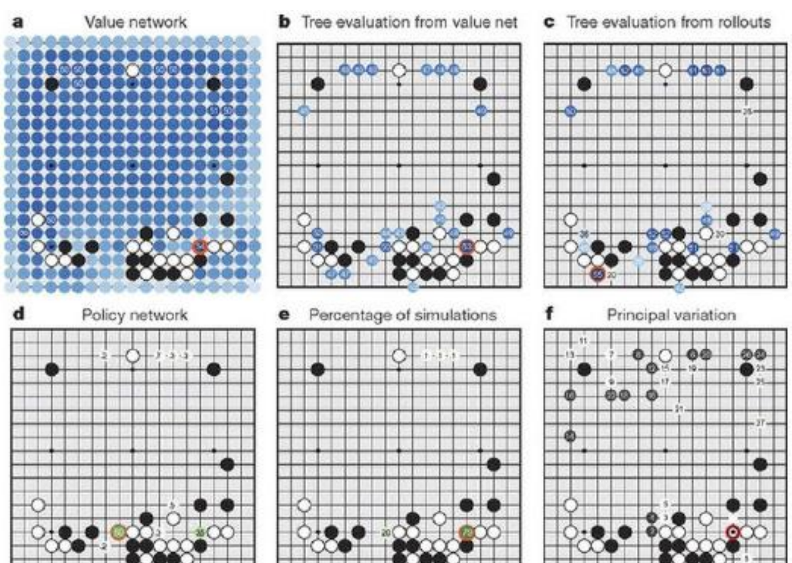


Figure copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017.

Case Study Bonus: DeepMind’s AlphaGo



Case Study Bonus: DeepMind's AlphaGo

The input to the policy network is a 19 × 19 × 48 image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23 × 23 image, then convolves *k* filters of kernel size 5 × 5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21 × 21 image, then convolves *k* filters of kernel size 3 × 3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1 × 1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used *k* = 192 filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with *k* = 128, 256 and 384 filters.

policy network:
[19x19x48] Input
CONV1: 192 5x5 filters , stride 1, pad 2 => [19x19x192]
CONV2..12: 192 3x3 filters, stride 1, pad 1 => [19x19x192]
CONV: 1 1x1 filter, stride 1, pad 0 => [19x19] *(probability map of promising moves)*

Summary

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Typical architectures look like

  [(CONV-RELU)*N-POOL]*M-(FC-RELU)*K,SOFTMAX

  where N is usually up to ~5, M is large, 0 <= K <= 2.
- Recent advances such as ResNet/GoogLeNet challenge this paradigm
- Trend towards decreasing or getting rid of POOL/FC layers (just CONV)
- Trend towards examining importance of depth vs. width and residual connections