Lecture 7:

Convolutional Neural Networks
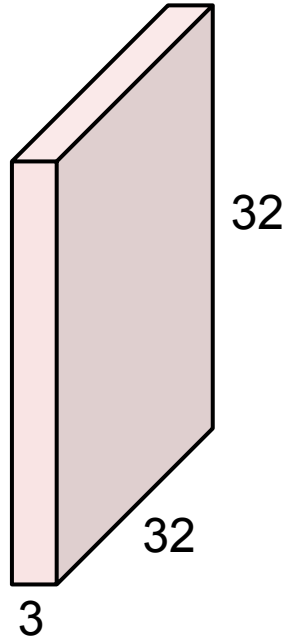
Convolution Layer

32x32x3 image

32 height

32 width

3 depth

Convolution Layer

32x32x3 image

32

32

3
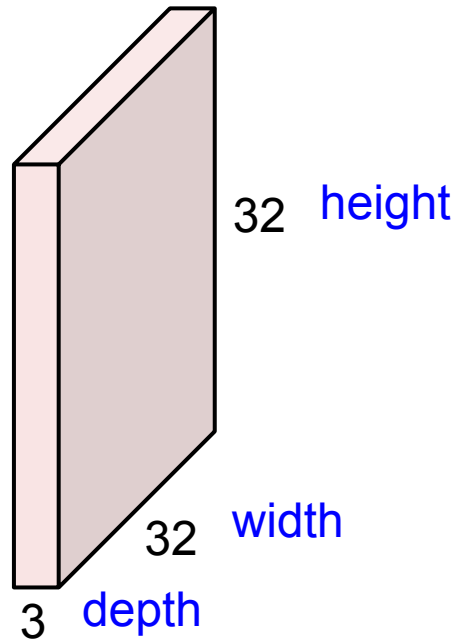
5x5x3 filter

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

Convolution Layer

32x32x3 image

Filters always extend the full depth of the input volume
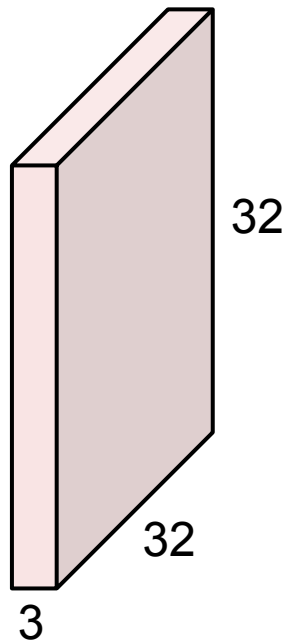
5x5x3 filter

32

32

3

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

Convolution Layer

32x32x3 image
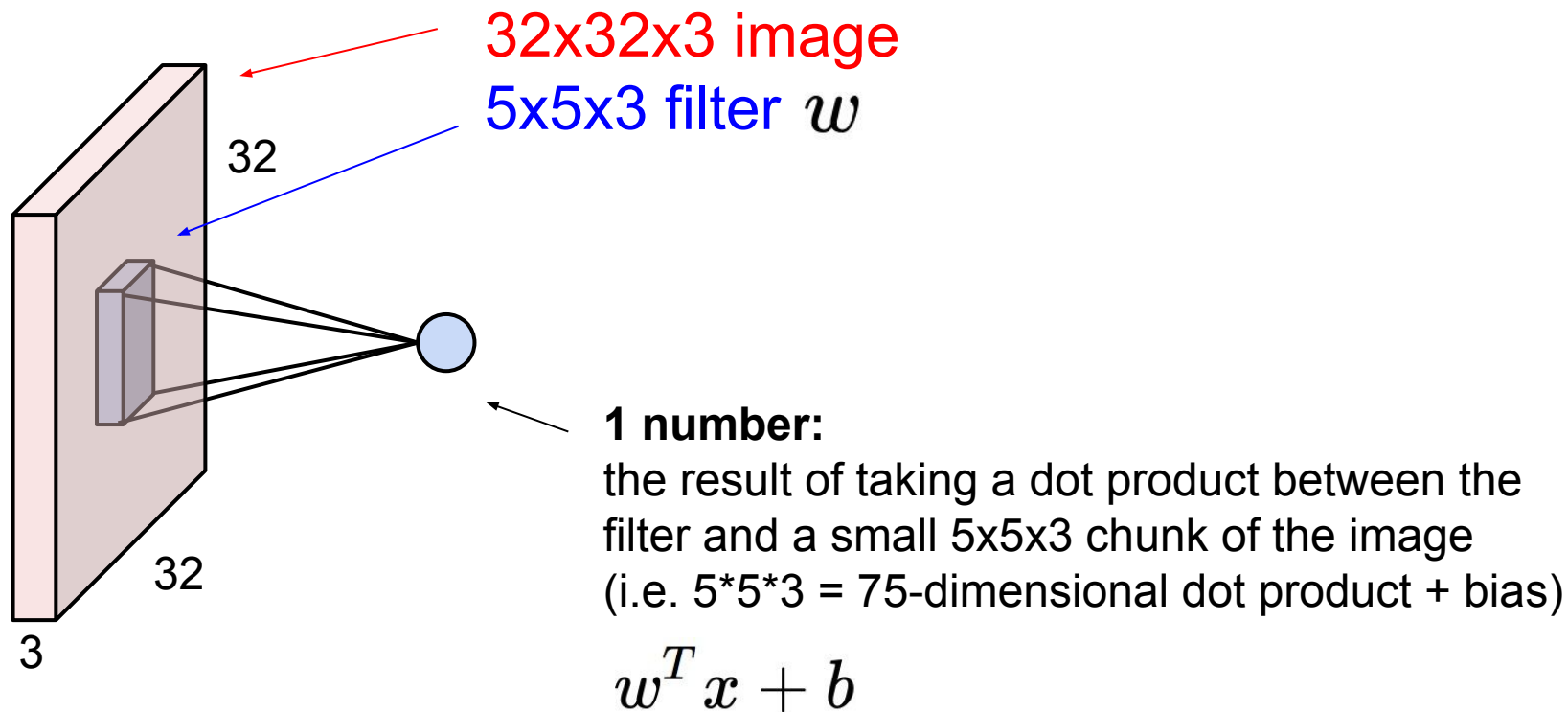5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolution Layer

32x32x3 image

5x5x3 filter

activation map

32

32

3

convolve (slide) over all spatial locations

28

28

1
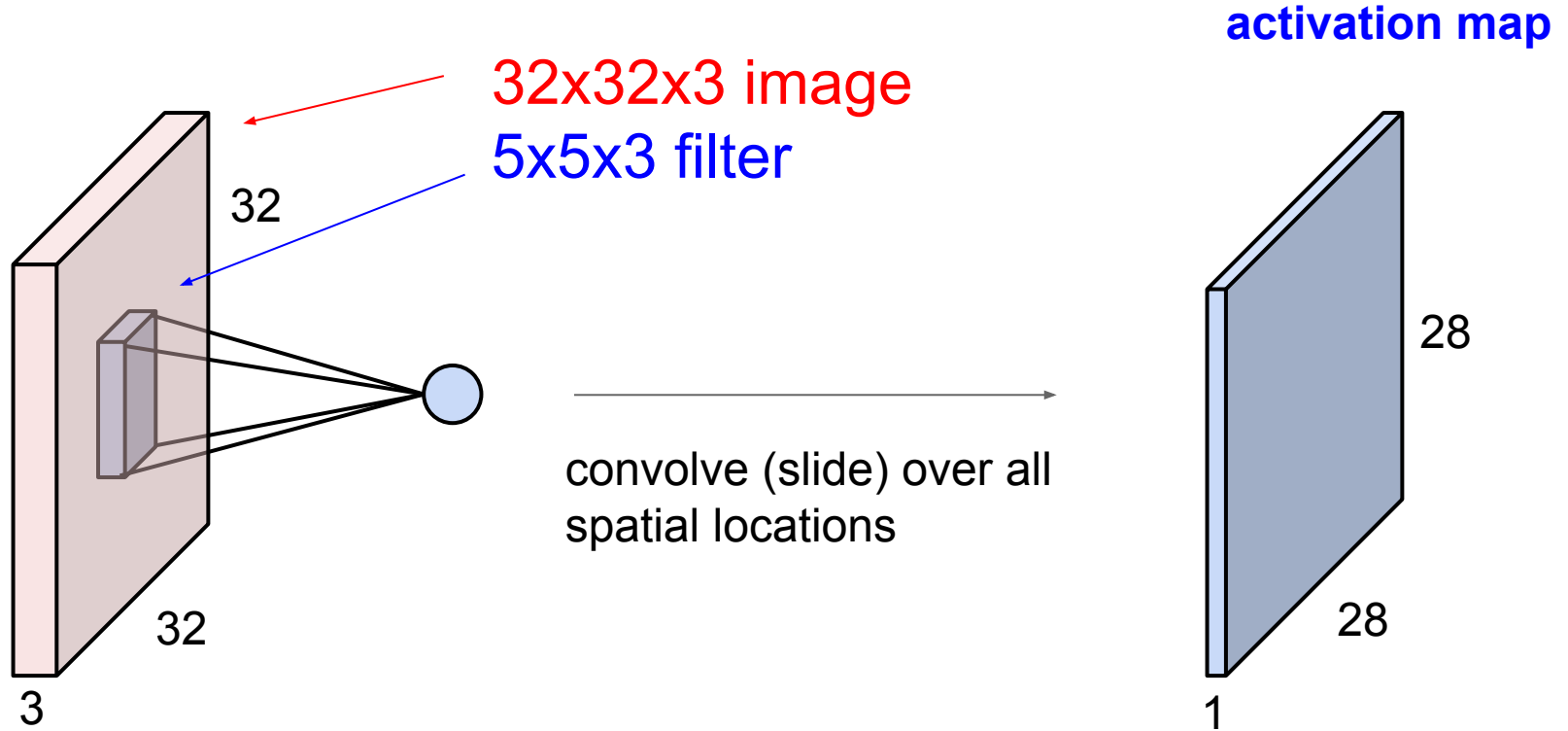
Convolution Layer

consider a second, green filter

32x32x3 image
5x5x3 filter

32

32

3

convolve (slide) over all
spatial locations

activation maps

28

28

1
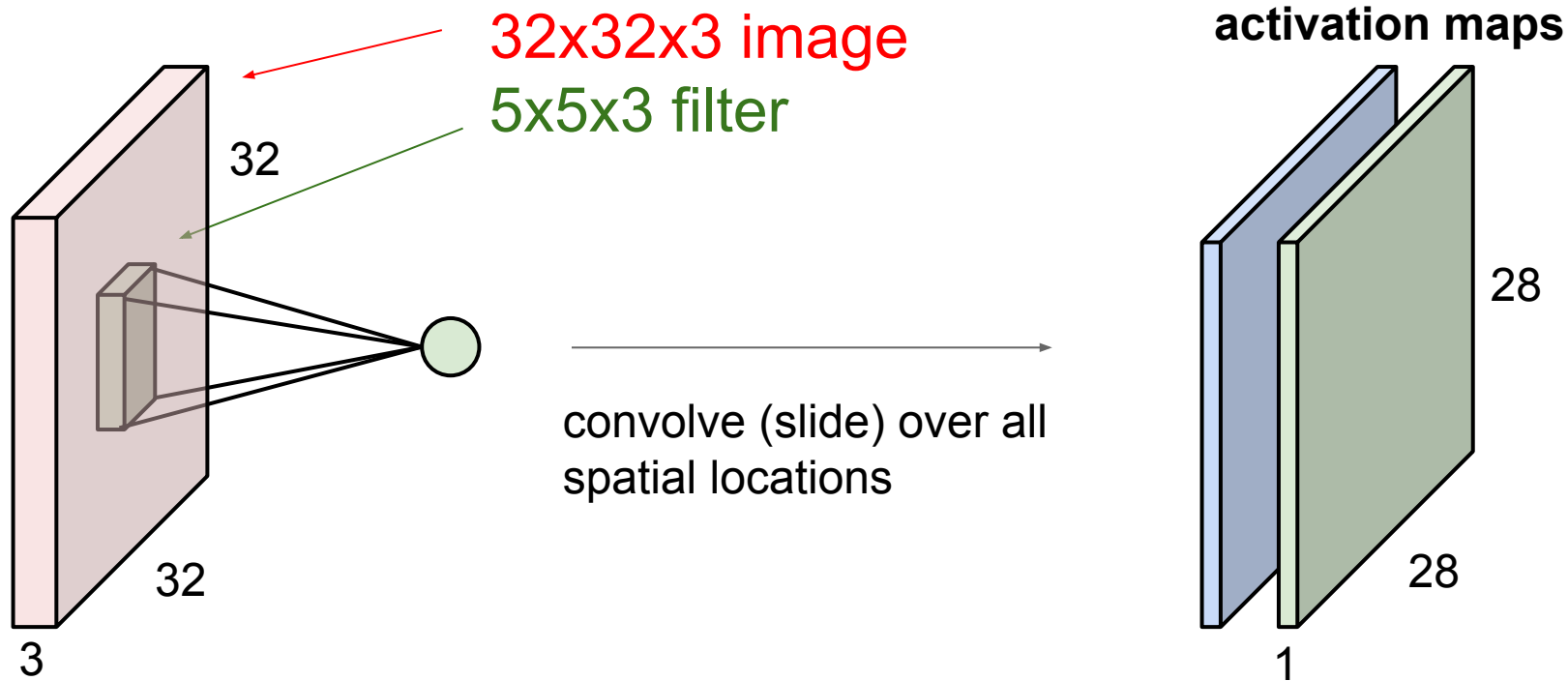
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

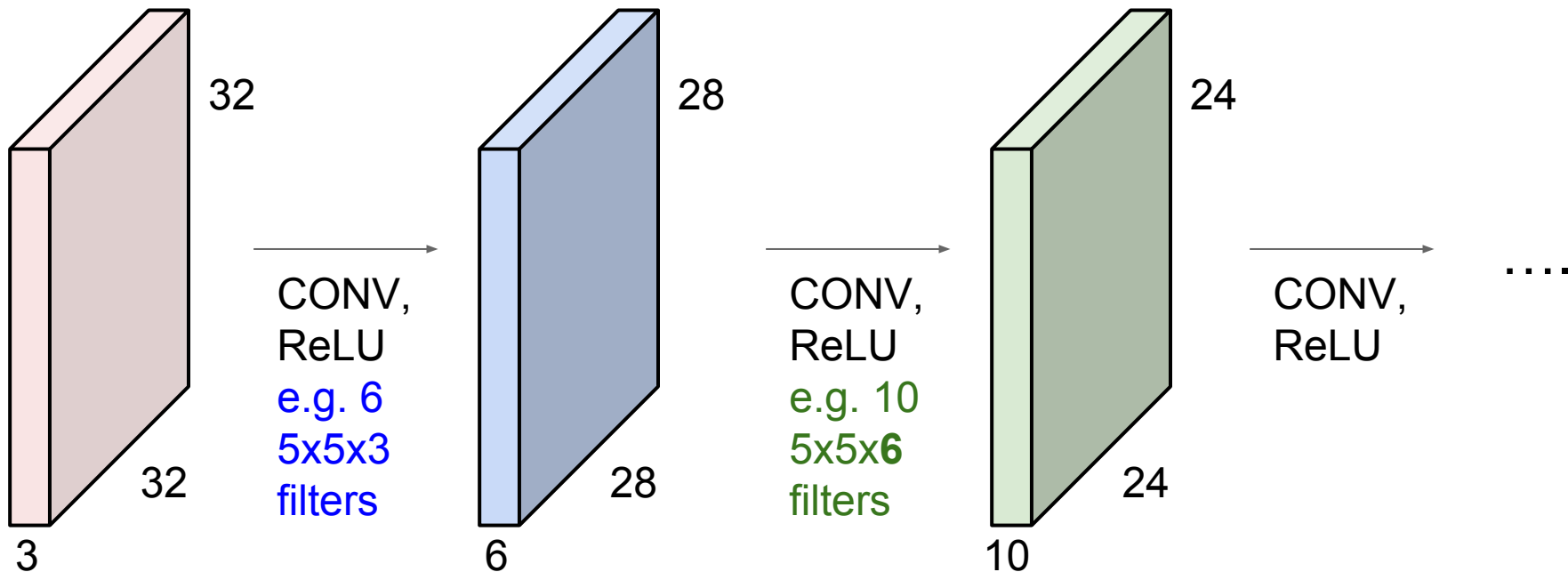**activation maps**

32

32

3

Convolution Layer

28

28

6
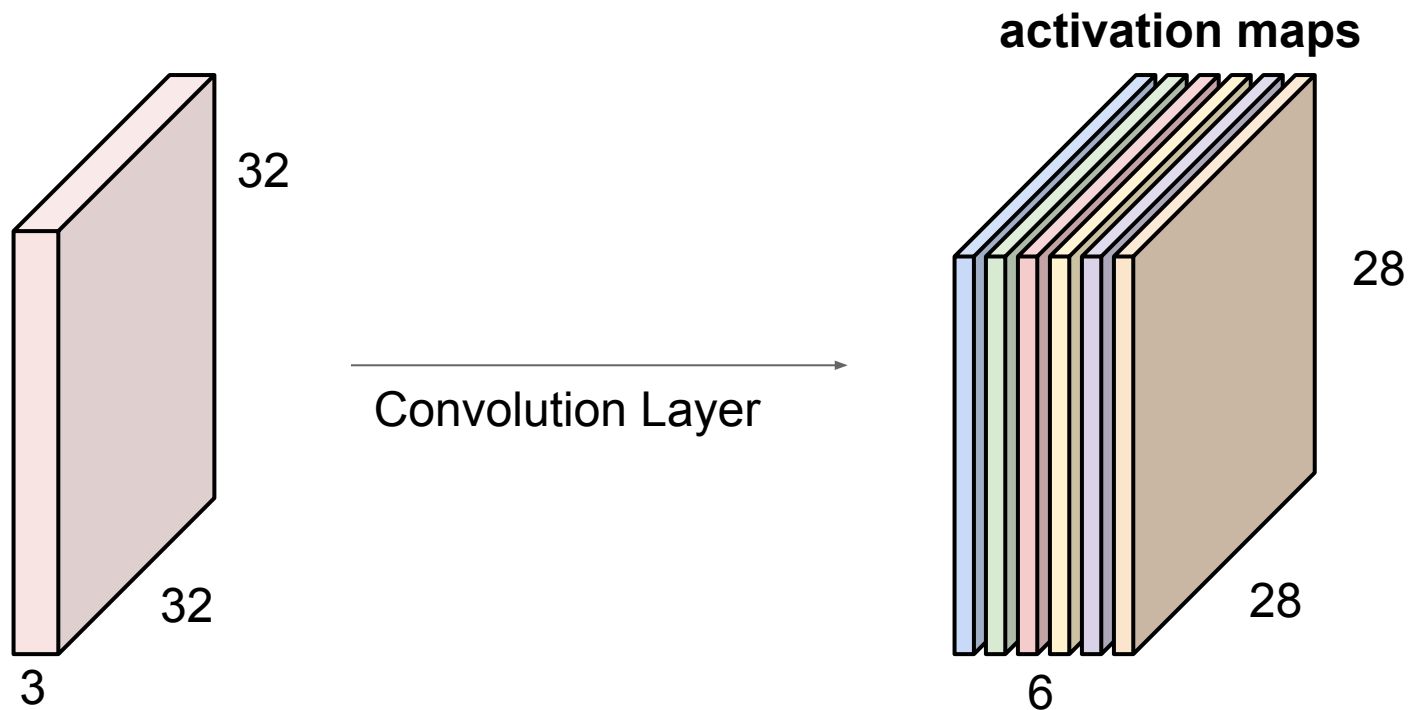
We stack these up to get a "new image" of size 28x28x6!

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

32

28

CONV,
ReLU
e.g. 6
5x5x3
filters

32

28

3

6
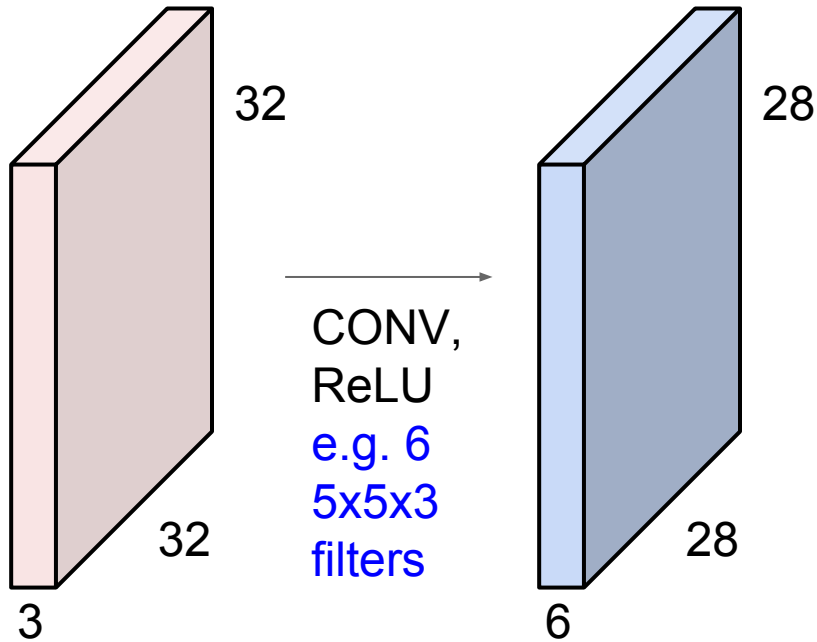
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions

Low-Level Feature → Mid-Level Feature → High-Level Feature → Trainable Classifier
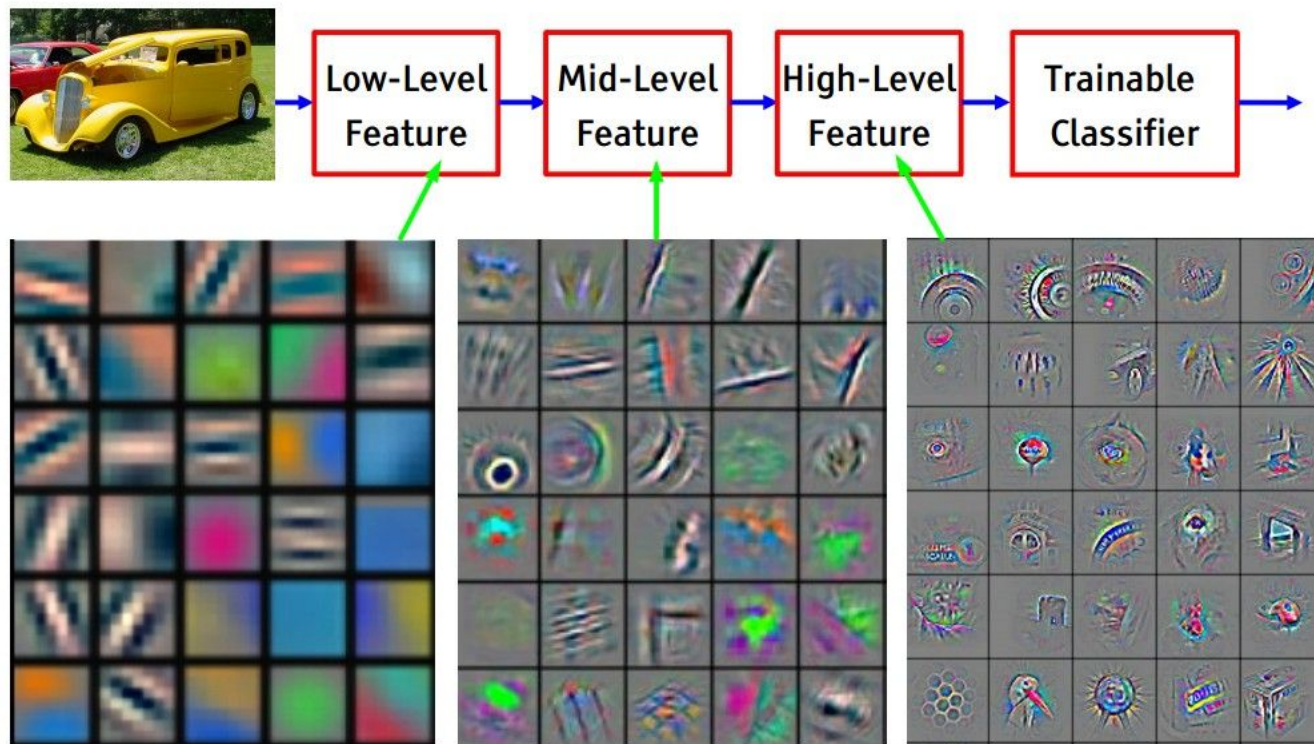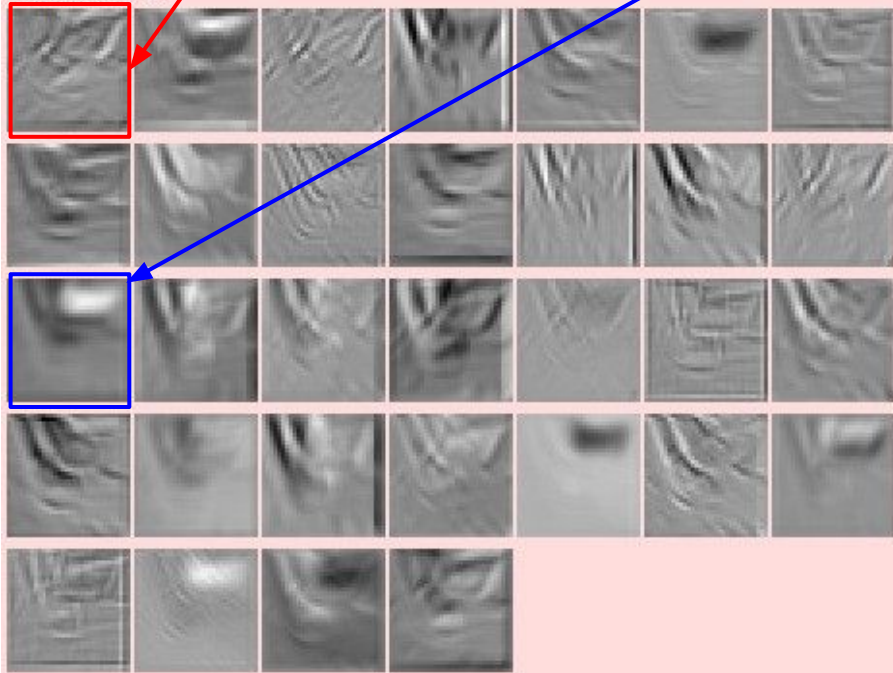
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

one filter =>
one activation map

example 5x5 filters
(32 total)

Activations:

We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] \;=\; \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

preview:
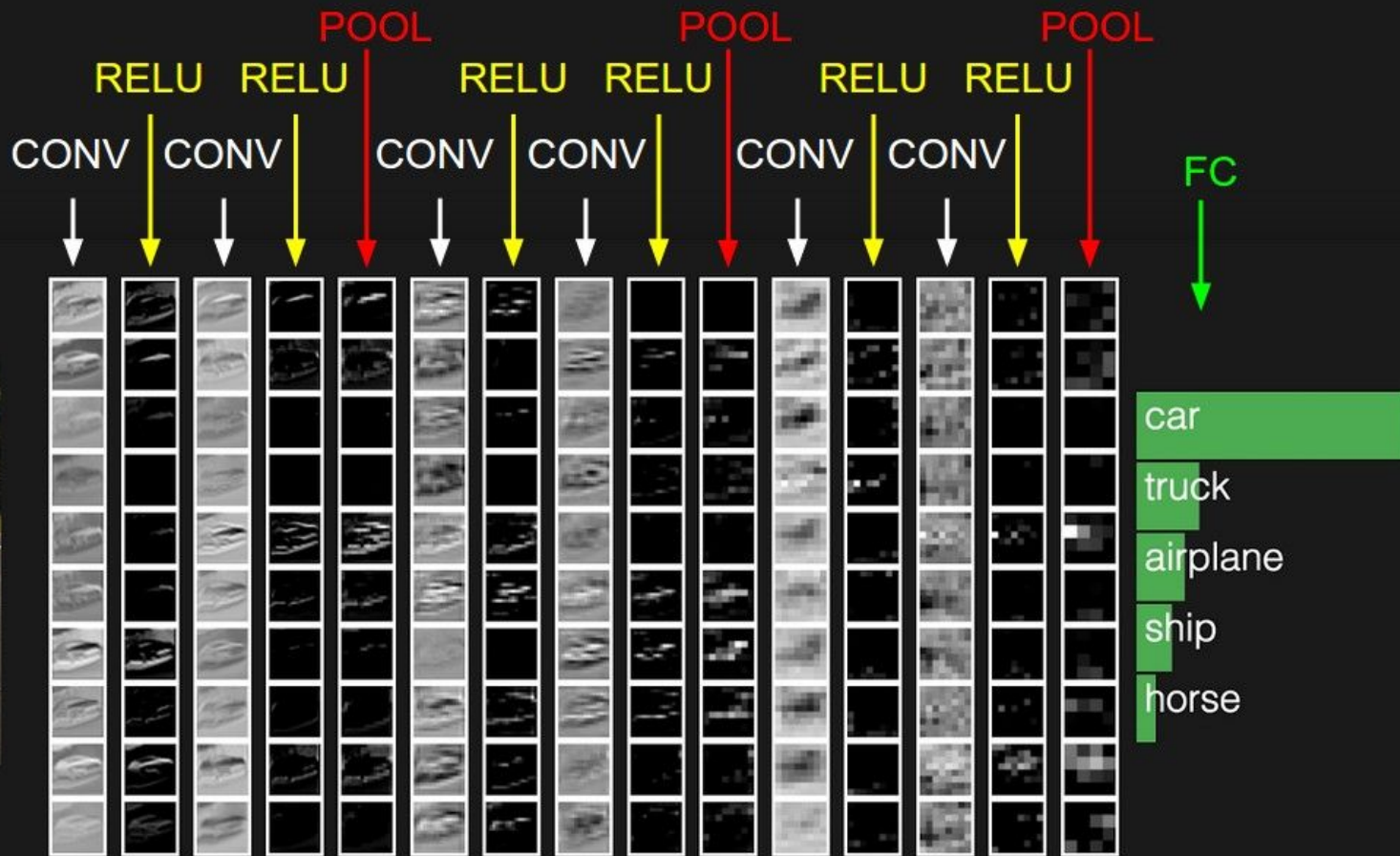
A closer look at spatial dimensions:
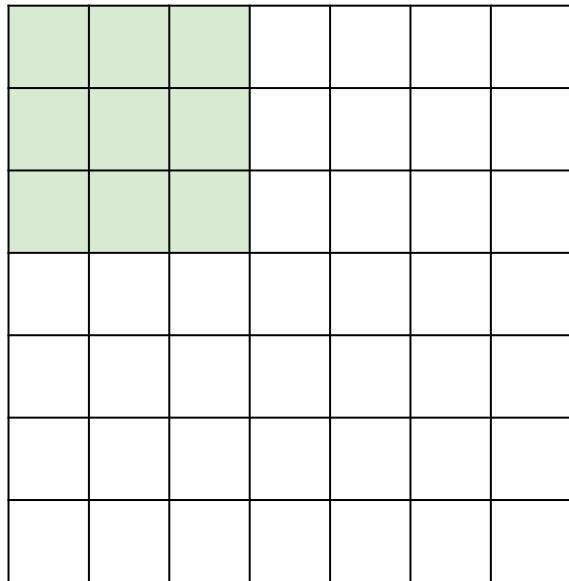
7

7

7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

7

7

7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

7

7
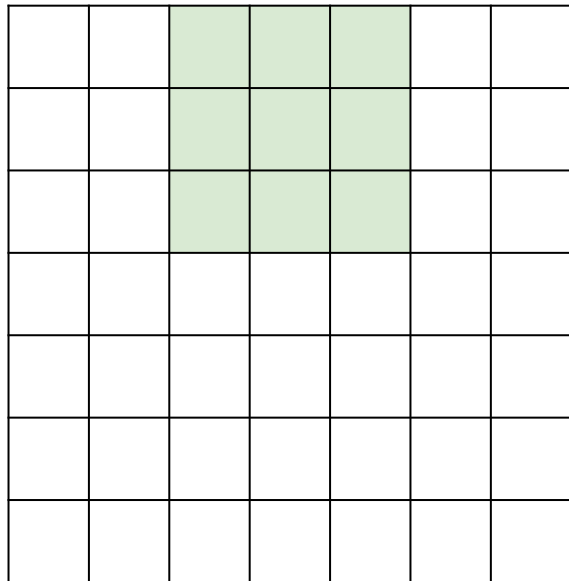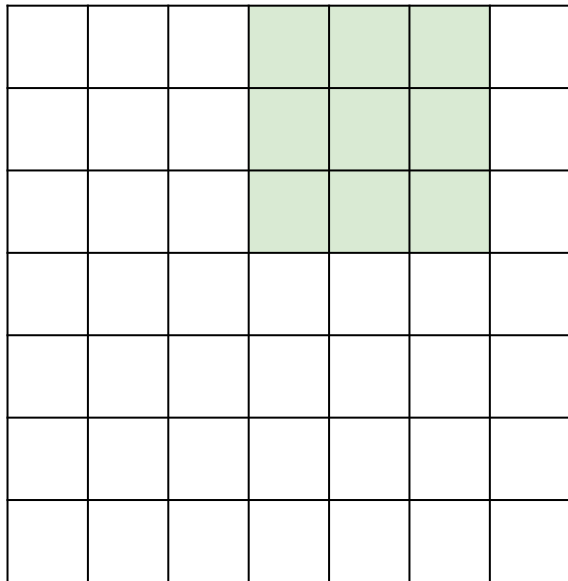
7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

A closer look at spatial dimensions:
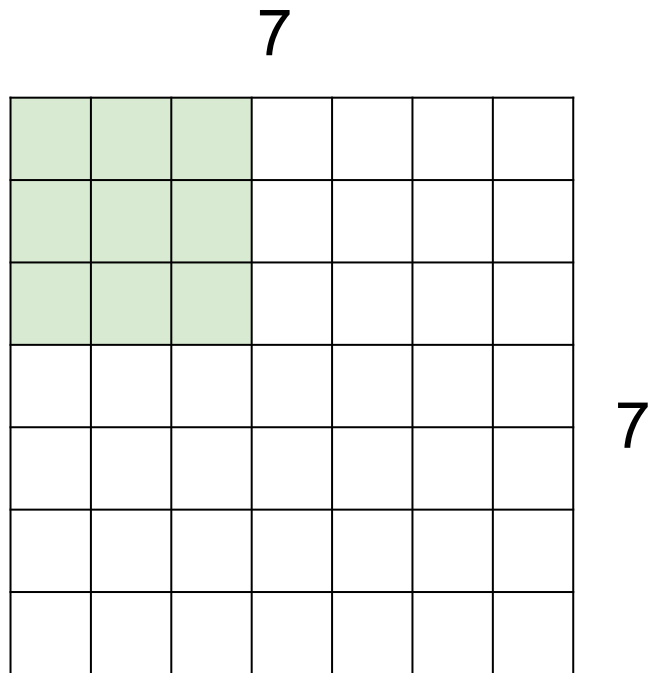
7

7

7x7 input (spatially)
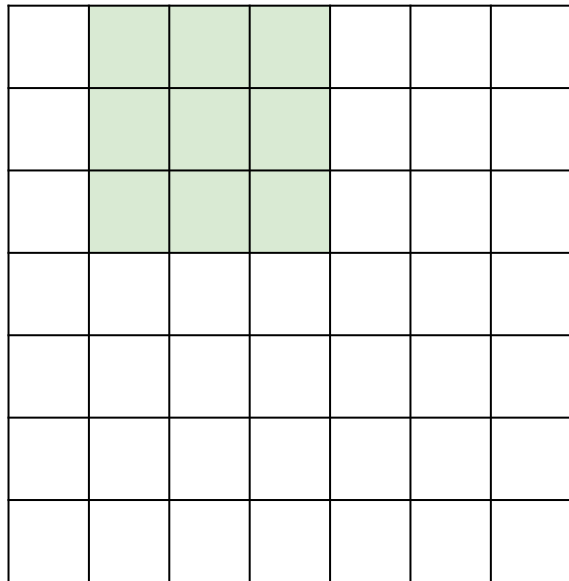assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:

7

7
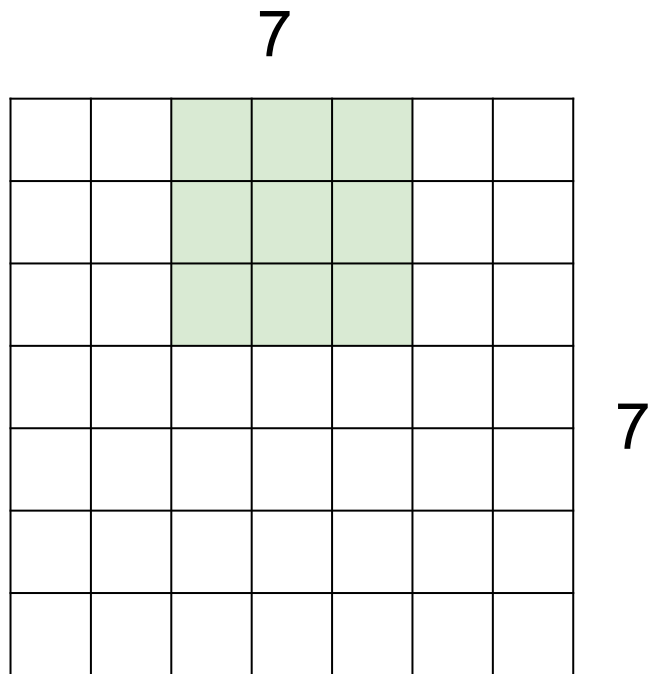
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:
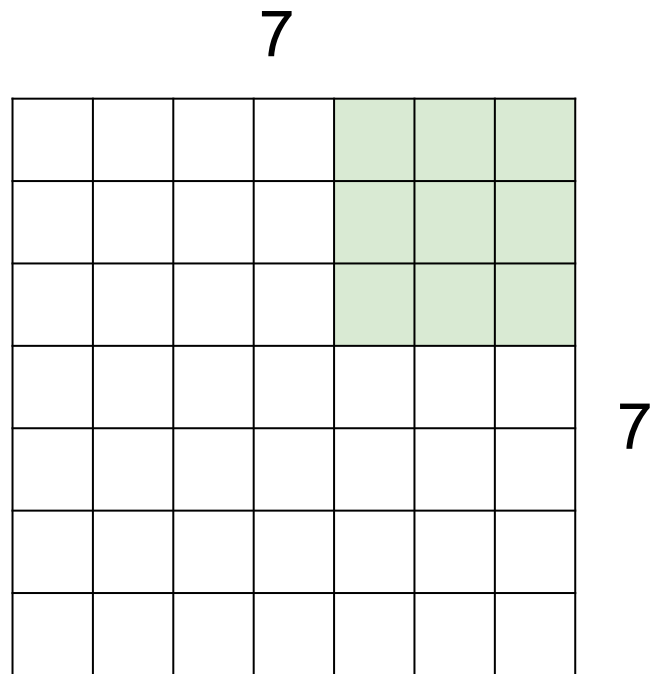
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:

7

7
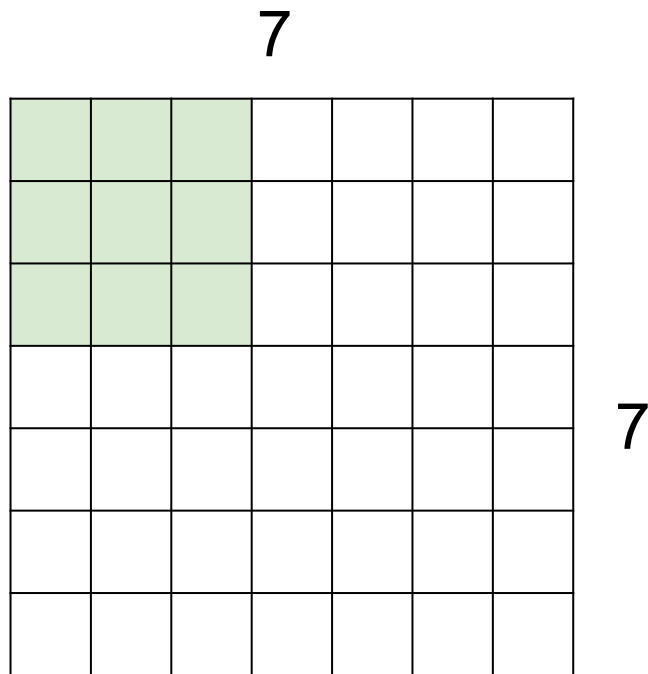
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:

7

7
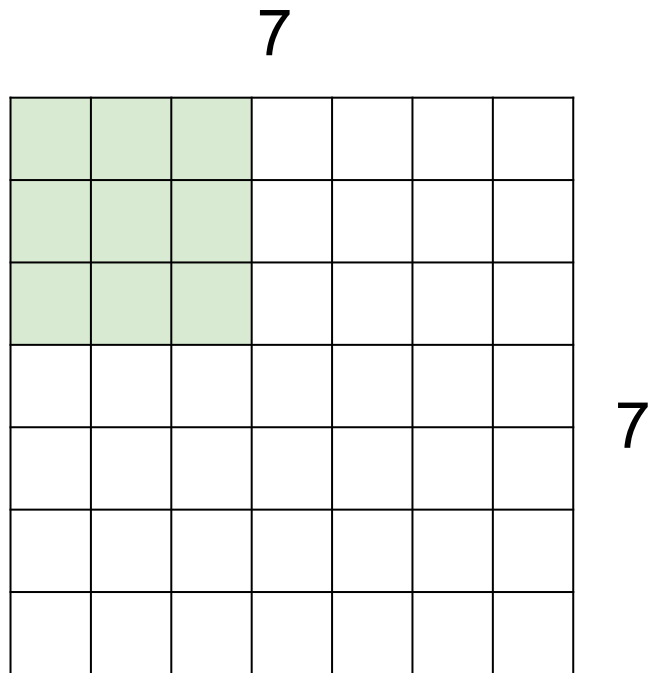
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

N

F

F

N
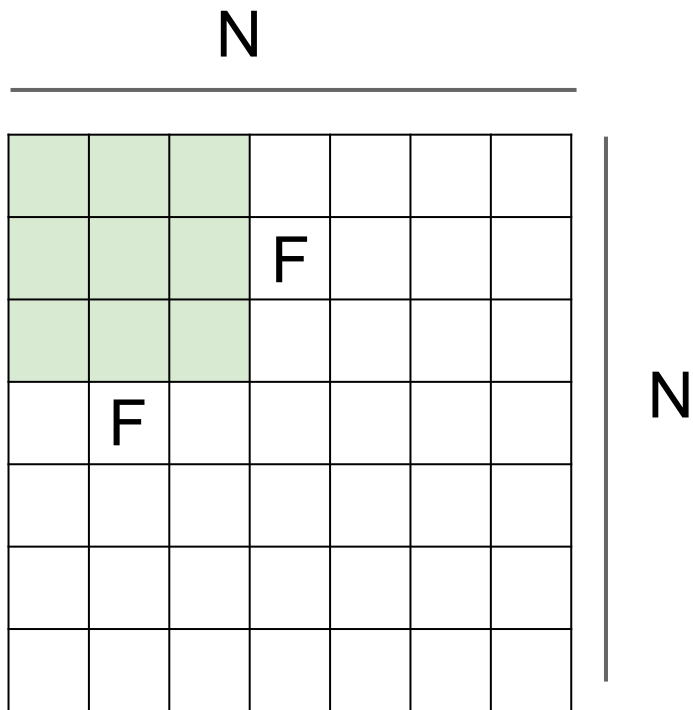
Output size:
(N - F) / stride + 1

e.g. 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 :\

In practice: Common to zero pad the border

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

(recall:)
(N - F) / stride + 1

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

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

7x7 output!

In practice: Common to zero pad the border

e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => 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

Remember back to…
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!
(32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.

32

28

24

32

28

24

3

6

10

CONV,
ReLU
e.g. 6
5x5x3
filters

CONV,
ReLU
e.g. 10
5x5x**6**
filters

CONV,
ReLU

….
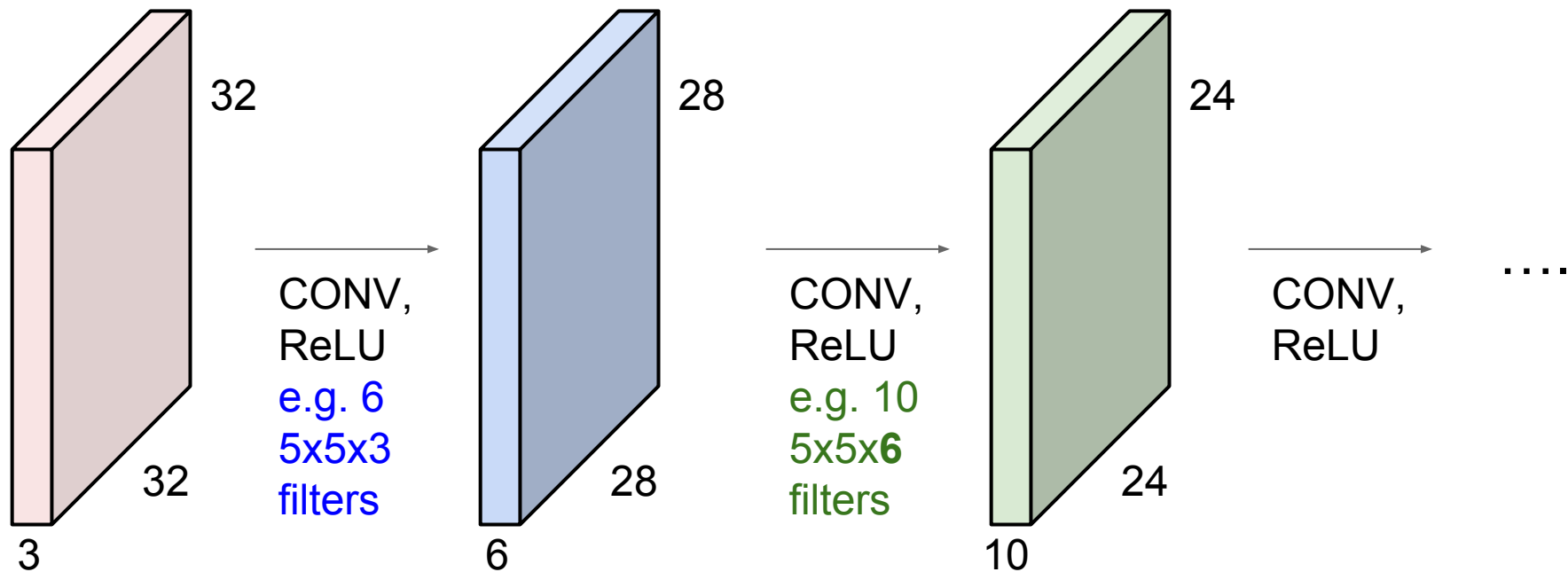
Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2
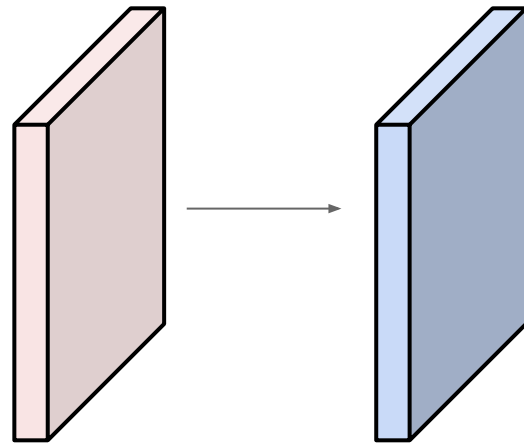
Output volume size: ?

Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10
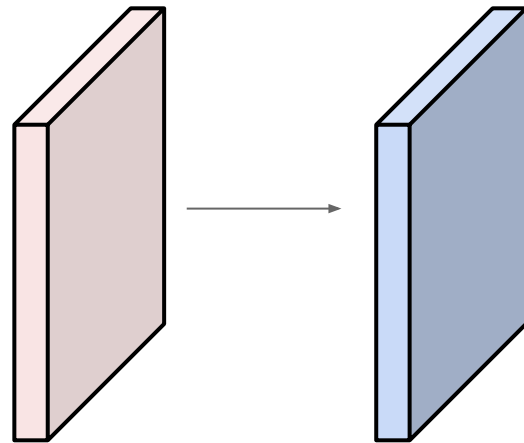
Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2
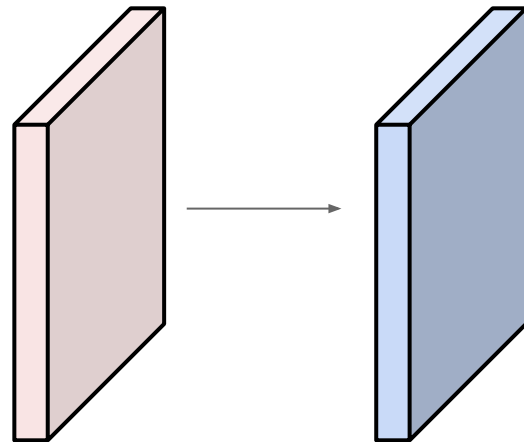
Number of parameters in this layer?

Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2
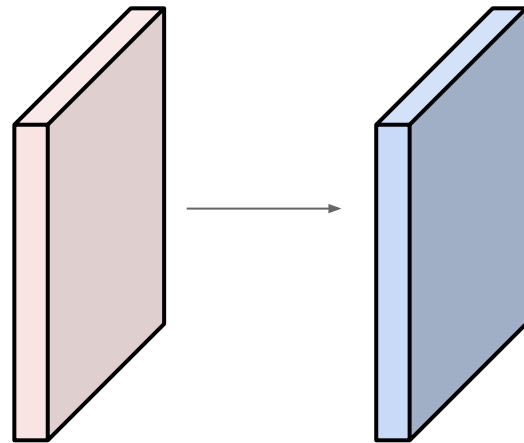
Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params   (+1 for bias)
=> 76*10 = **760**

Summary. To summarize, the Conv 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.

(btw, 1x1 convolution layers make perfect sense)

1x1 CONV
with 32 filters

(each filter has size
1x1x64, and performs a
64-dimensional dot
product)

56

56

64

56

56

32
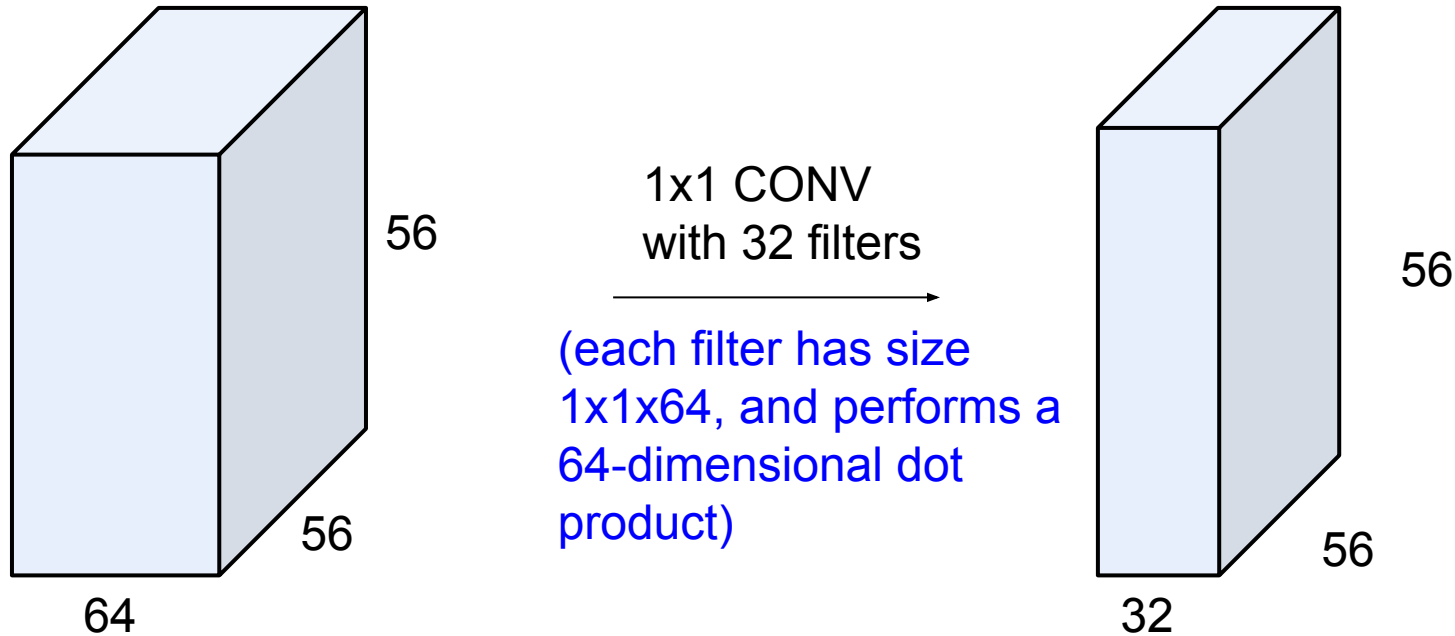
Example: CONV layer in Caffe

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  # learning rate and decay multipliers for the filters
  param { lr_mult: 1 decay_mult: 1 }
  # learning rate and decay multipliers for the biases
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 96     # learn 96 filters
    kernel_size: 11    # each filter is 11x11
    stride: 4          # step 4 pixels between each filter application
    weight_filler {
      type: "gaussian" # initialize the filters from a Gaussian
      std: 0.01        # distribution with stdev 0.01 (default mean: 0)
    }
    bias_filler {
      type: "constant" # initialize the biases to zero (0)
      value: 0
    }
  }
}
```

Summary. To summarize, the Conv 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.

The brain/neuron view of CONV Layer
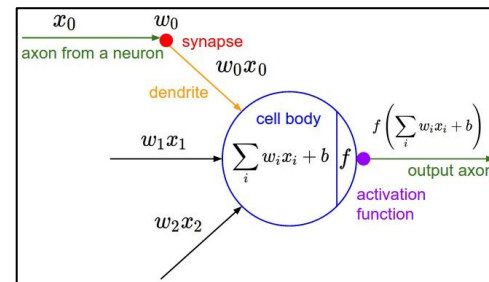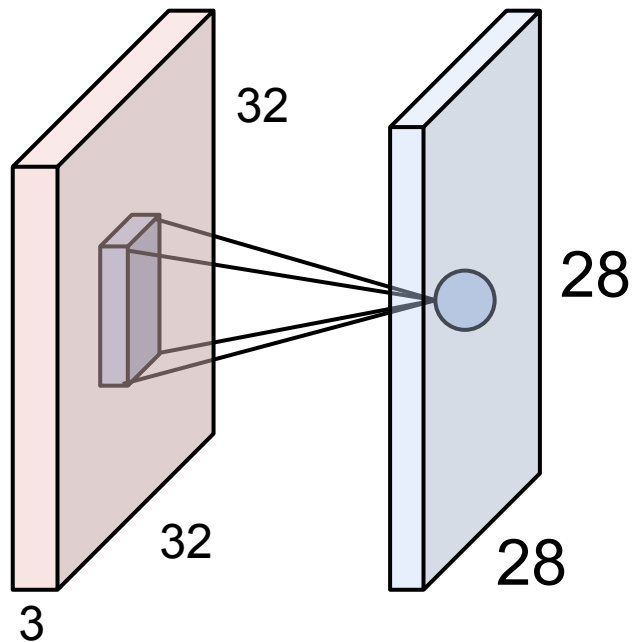
An activation map is a 28x28 sheet of neuron outputs:
1. Each is connected to a small region in the input
2. All of them share parameters

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

The brain/neuron view of CONV Layer
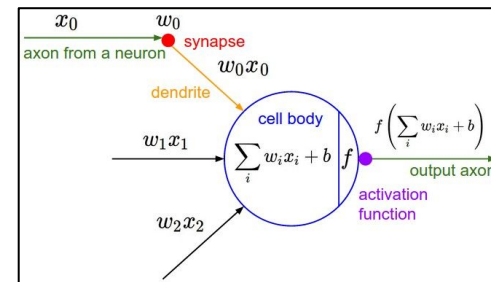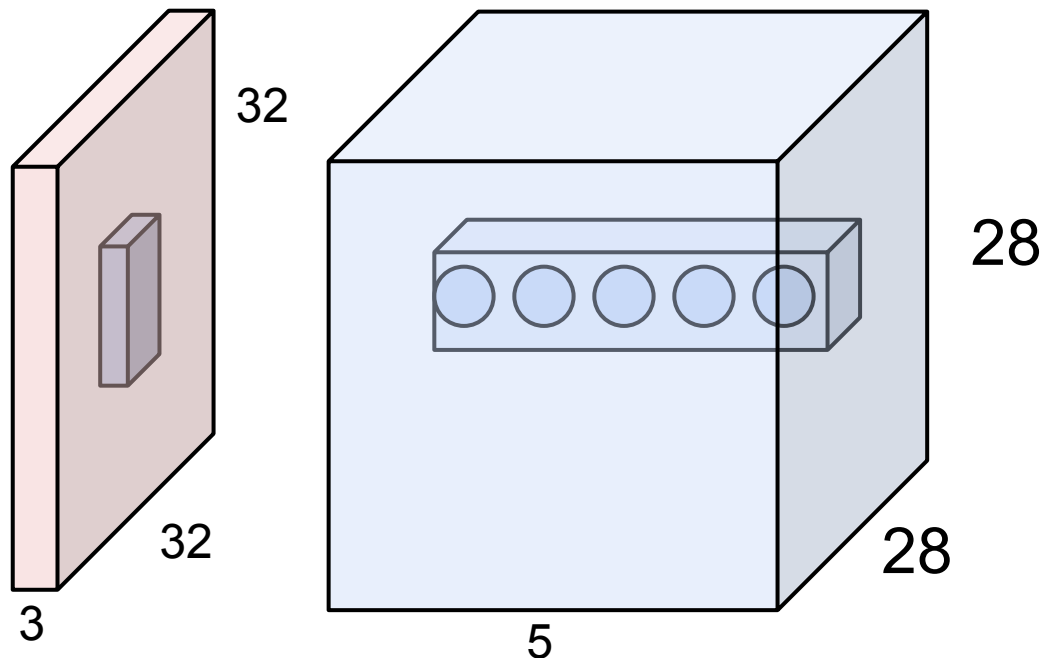
E.g. with 5 filters,
CONV layer consists of
neurons arranged in a 3D grid
(28x28x5)

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

two more layers to go: POOL/FC

Pooling layer

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

MAX POOLING

Single depth slice

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

x

y

max pool with 2x2 filters
and stride 2

→

6	8
3	4
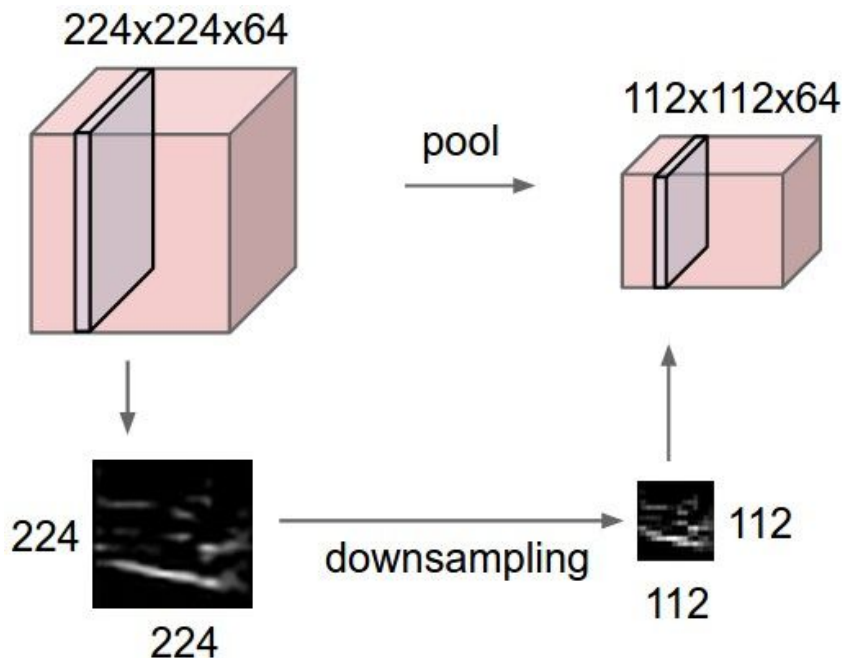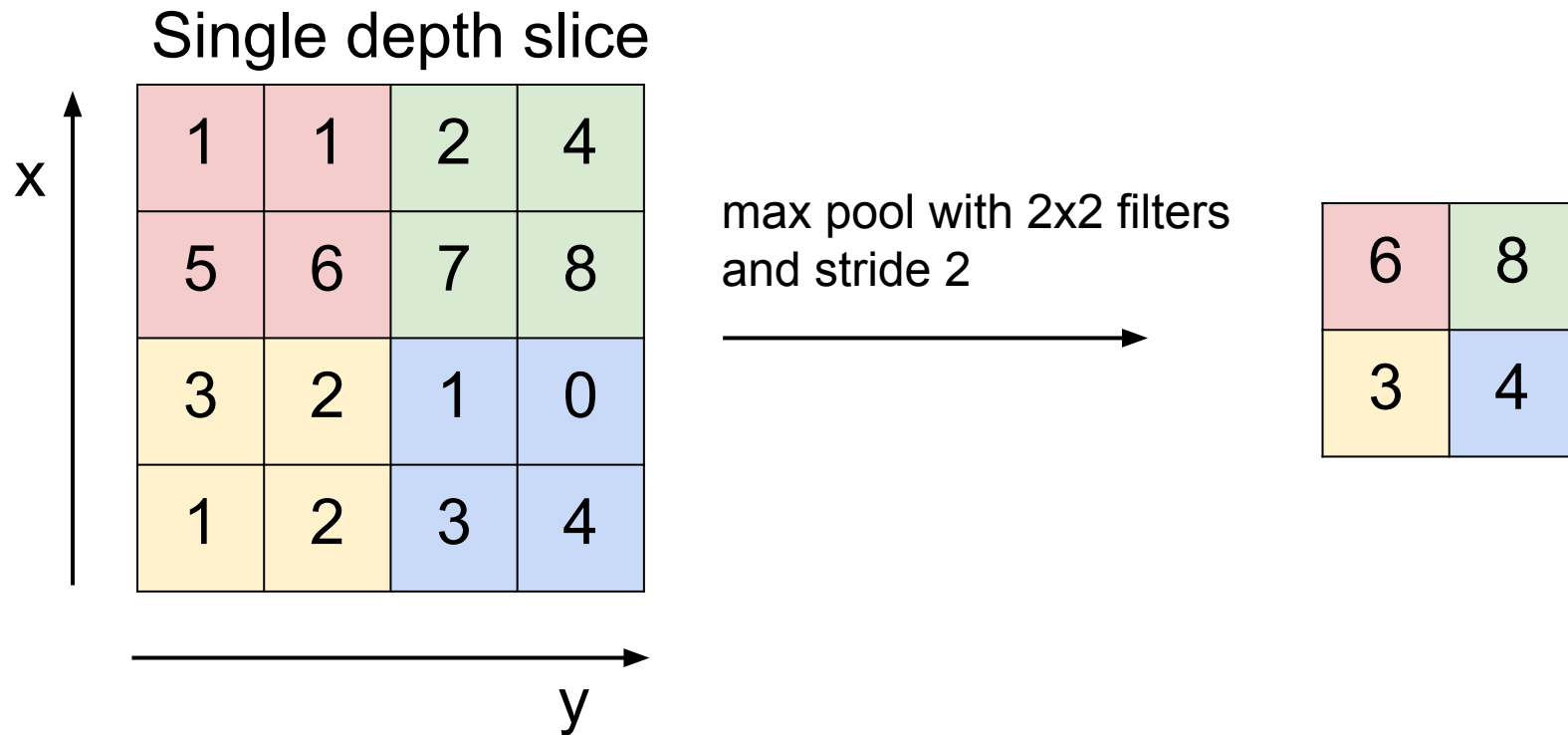
Common settings:

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

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three 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

Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks
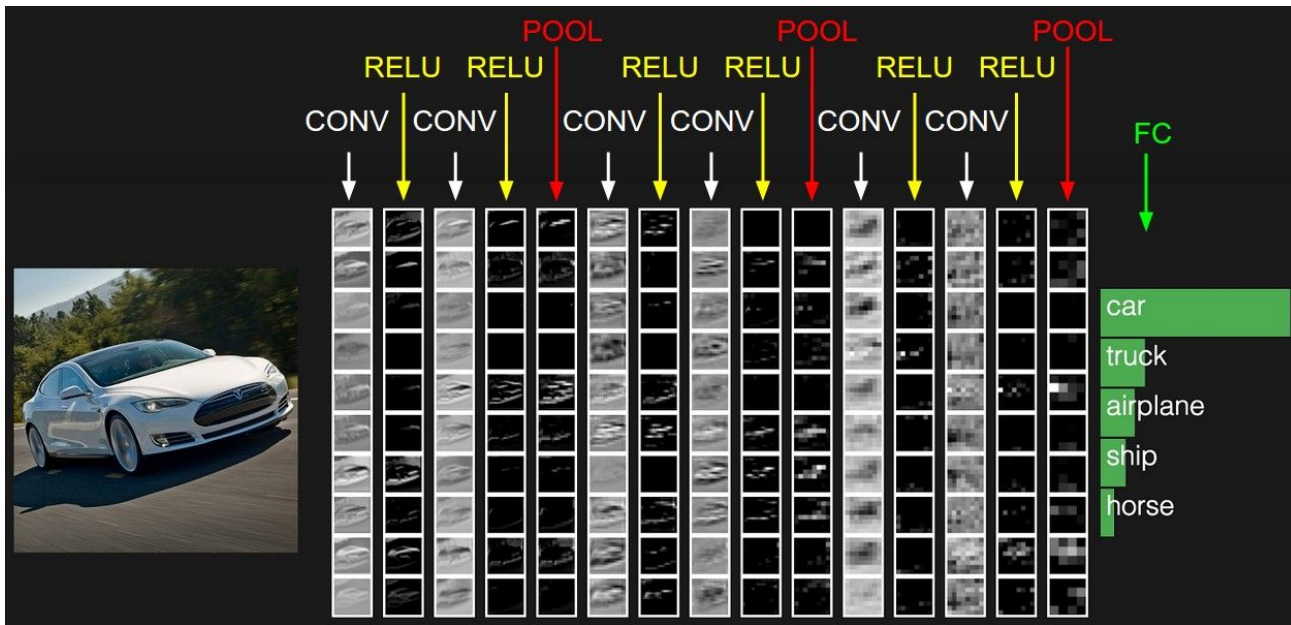
[ConvNetJS demo: training on CIFAR-10]

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

Case Study: AlexNet

[Krizhevsky et al. 2012]
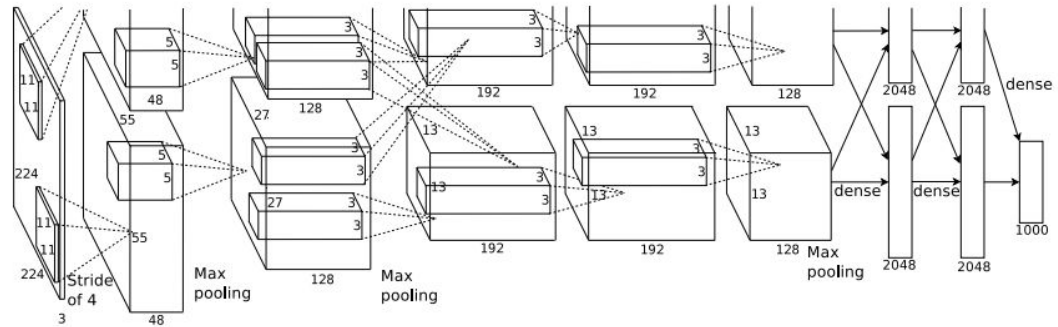
Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Q: what is the output volume size? Hint: (227-11)/4+1 = 55

Case Study: AlexNet
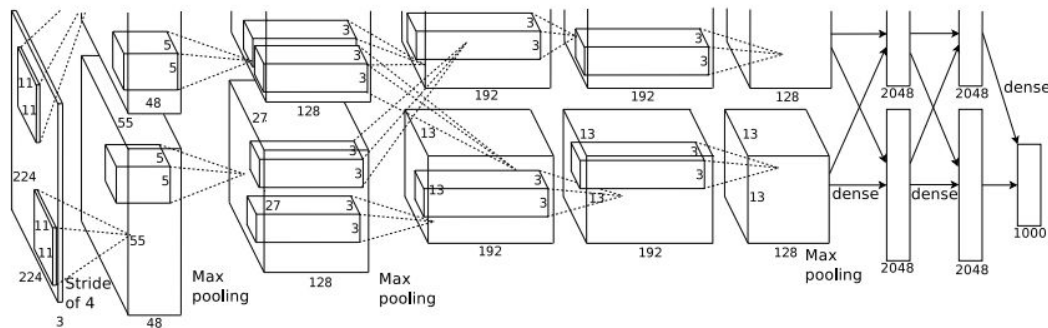
[Krizhevsky et al. 2012]

Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?
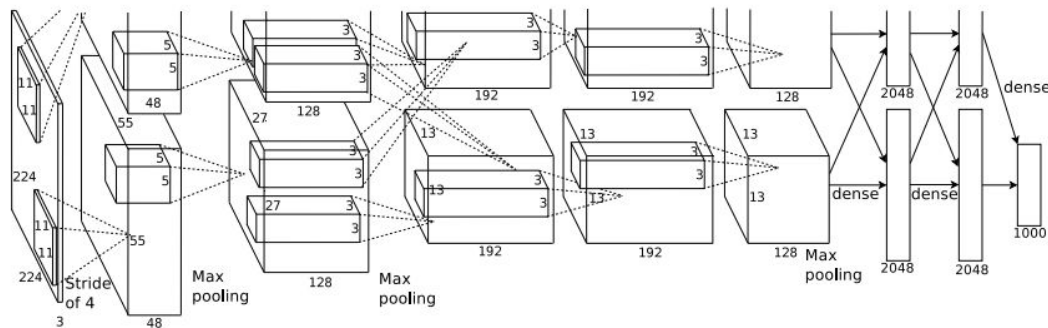
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**

Case Study: AlexNet
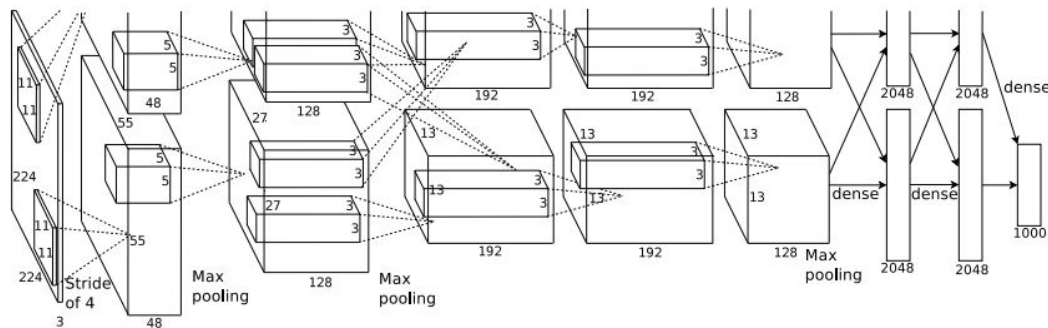
[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96

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

Q: what is the output volume size? Hint: (55-3)/2+1 = 27

Case Study: AlexNet
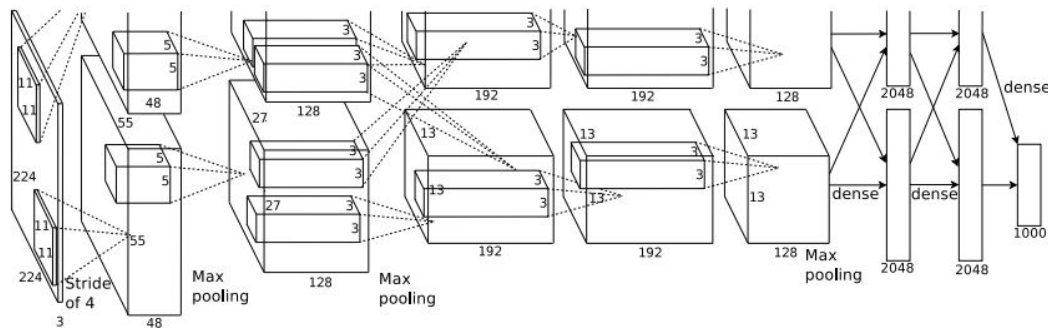
[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

Q: what is the number of parameters in this layer?
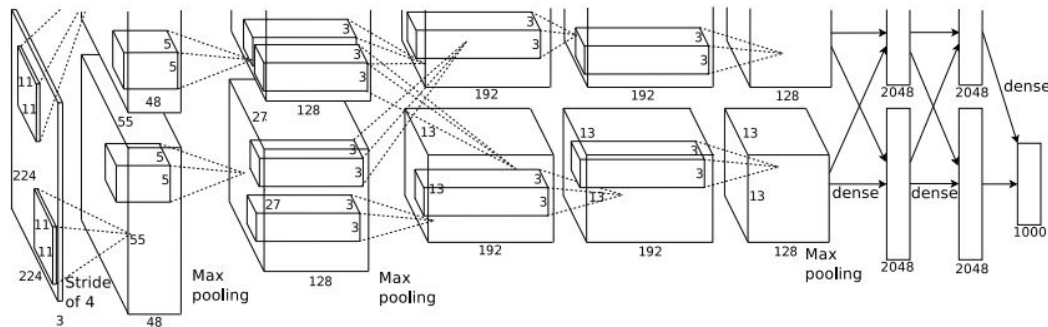
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

Case Study: AlexNet

[Krizhevsky et al. 2012]
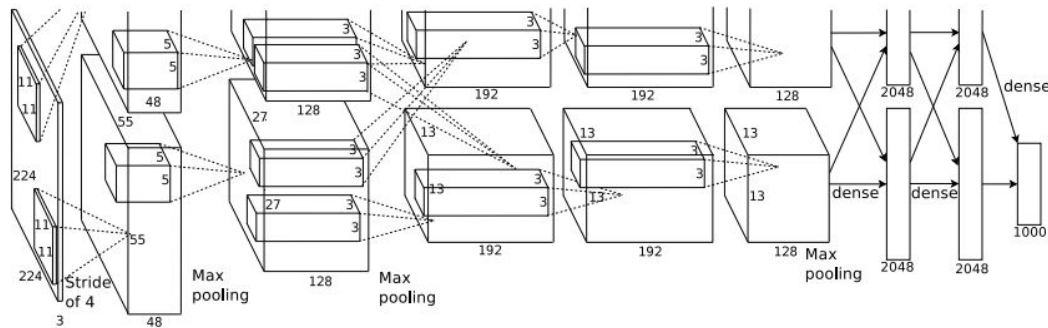
Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...

Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) 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: VGGNet

[Simonyan and Zisserman, 2014]

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

best model

11.2% top 5 error in ILSVRC 2013
->
7.3% top 5 error

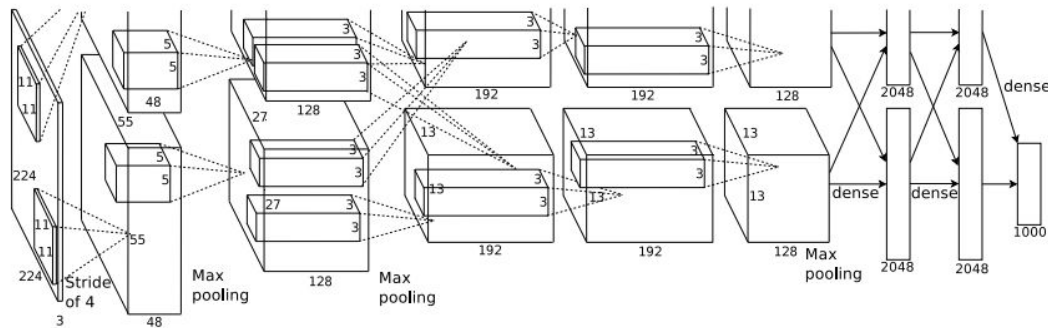| ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64	conv3-64	conv3-64	conv3-64	conv3-64
	LRN	**conv3-64**	conv3-64	conv3-64	conv3-64
maxpool					
conv3-128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

INPUT: [224x224x3]   memory: 224*224*3=150K   params: 0   (not counting biases)
CONV3-64: [224x224x64]   memory: 224*224*64=3.2M   params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64]   memory: 224*224*64=3.2M   params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K   params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M   params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M   params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K   params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K   params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K   params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K   params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K   params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K   params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K   params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K   params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K   params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K   params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K   params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K   params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K   params: 0
FC: [1x1x4096] memory: 4096   params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096   params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000   params: 4096*1000 = 4,096,000

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

INPUT: [224x224x3]   memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

(not counting biases)
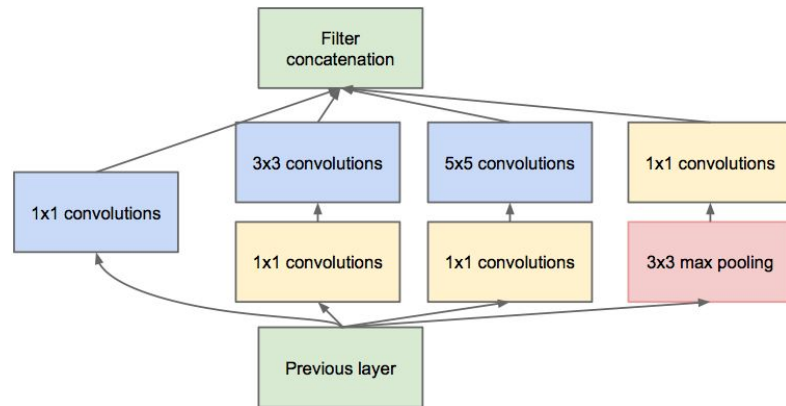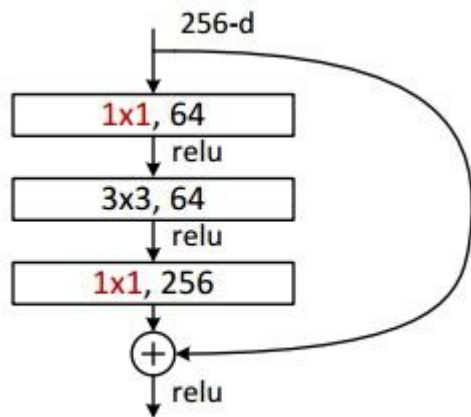
Note:

Most memory is in early CONV

Most params are in late FC

Case Study: ResNet

[He et al., 2015]

- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

Case Study: ResNet

[He et al., 2015]

(this trick is also used in GoogLeNet)

Summary

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- 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.
    - but recent advances such as ResNet/GoogLeNet
      challenge this paradigm