

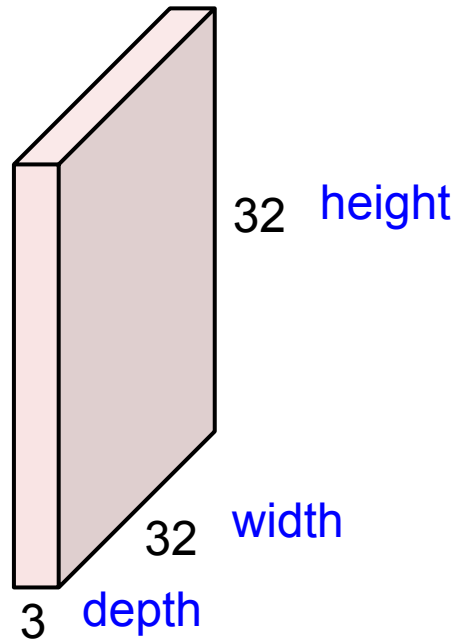
# Lecture 4:

# Convolutional Neural Networks

slides credited to Fei-Fei Li @stanford Univesity

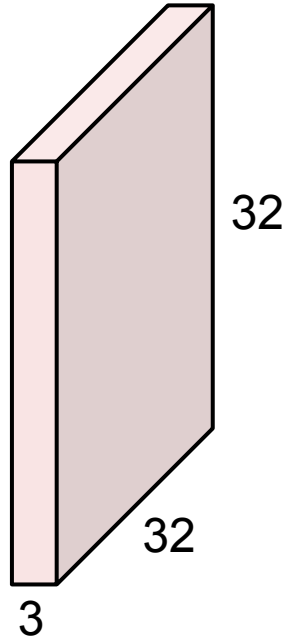
# Convolution Layer

32x32x3 image



# Convolution Layer

32x32x3 image



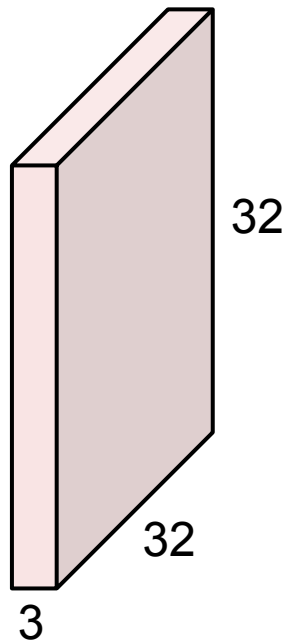
5x5x3 filter



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

# Convolution Layer

32x32x3 image



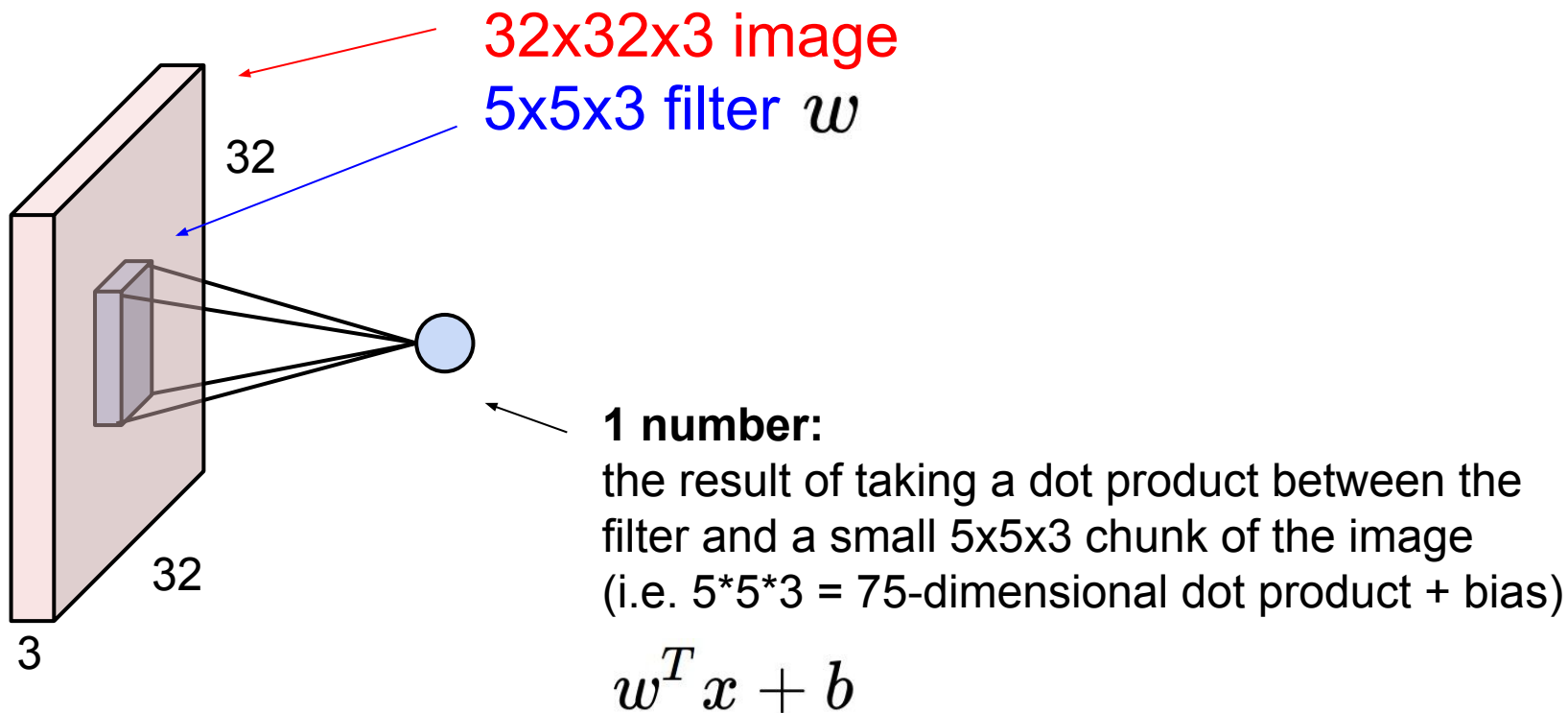
Filters always extend the full depth of the input volume

5x5x3 filter

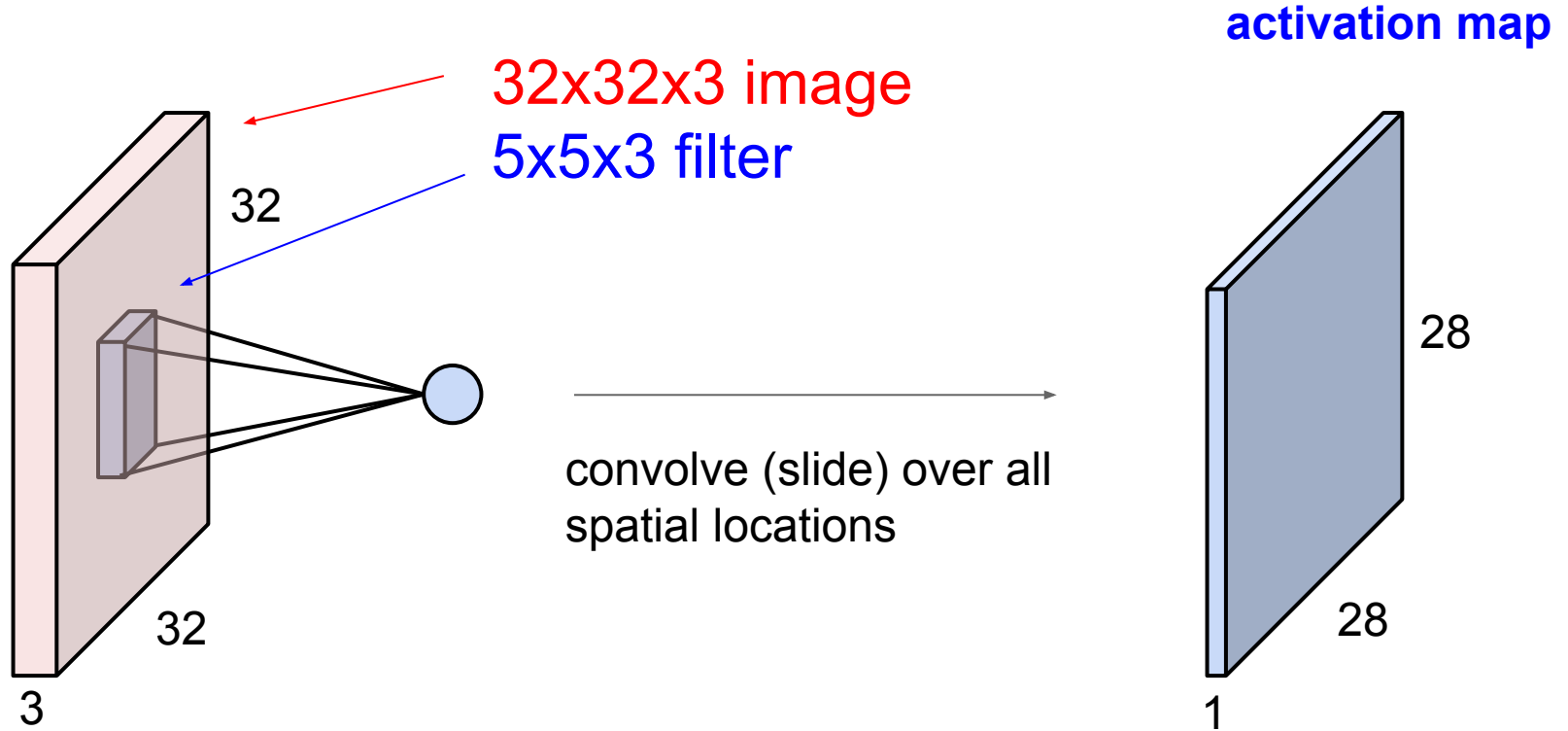


**Convolve** the filter with the image  
i.e. “slide over the image spatially,  
computing dot products”

# Convolution Layer

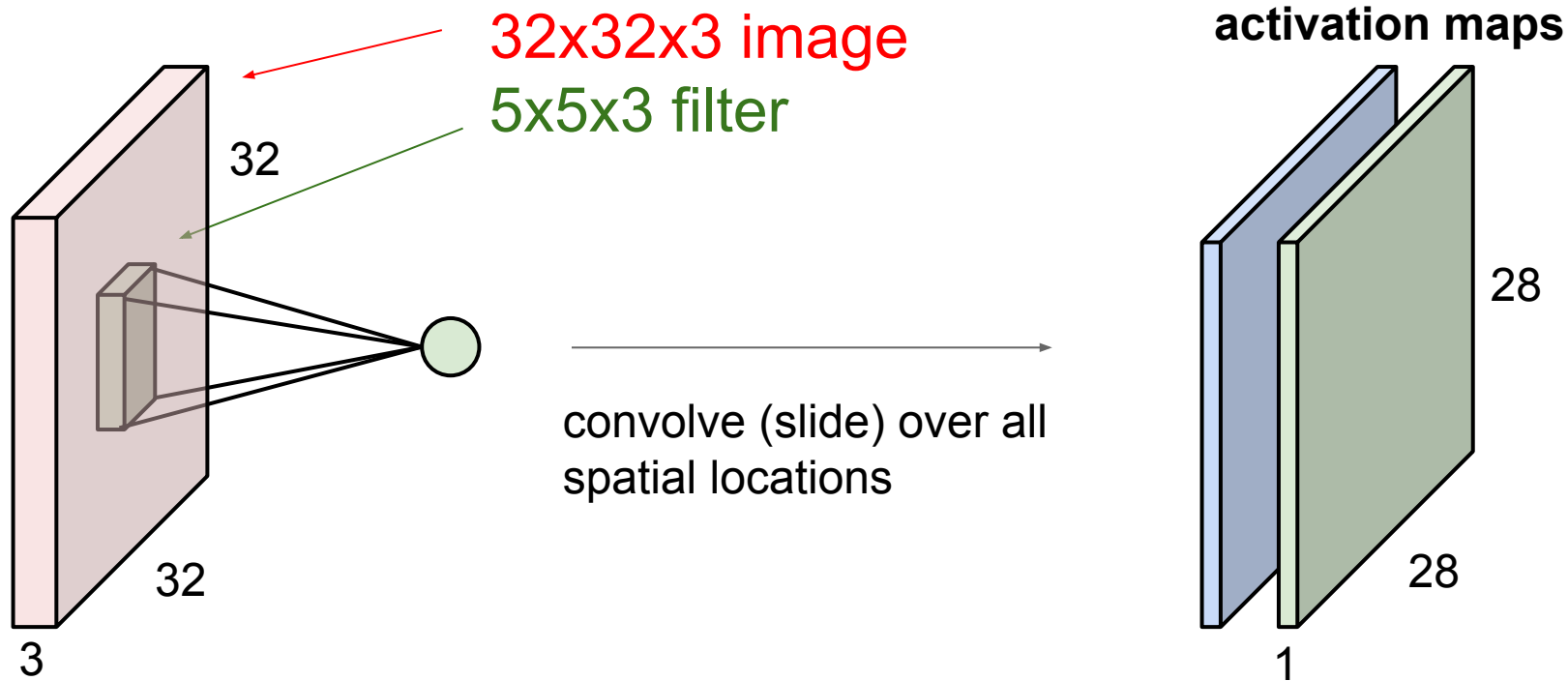


# Convolution Layer

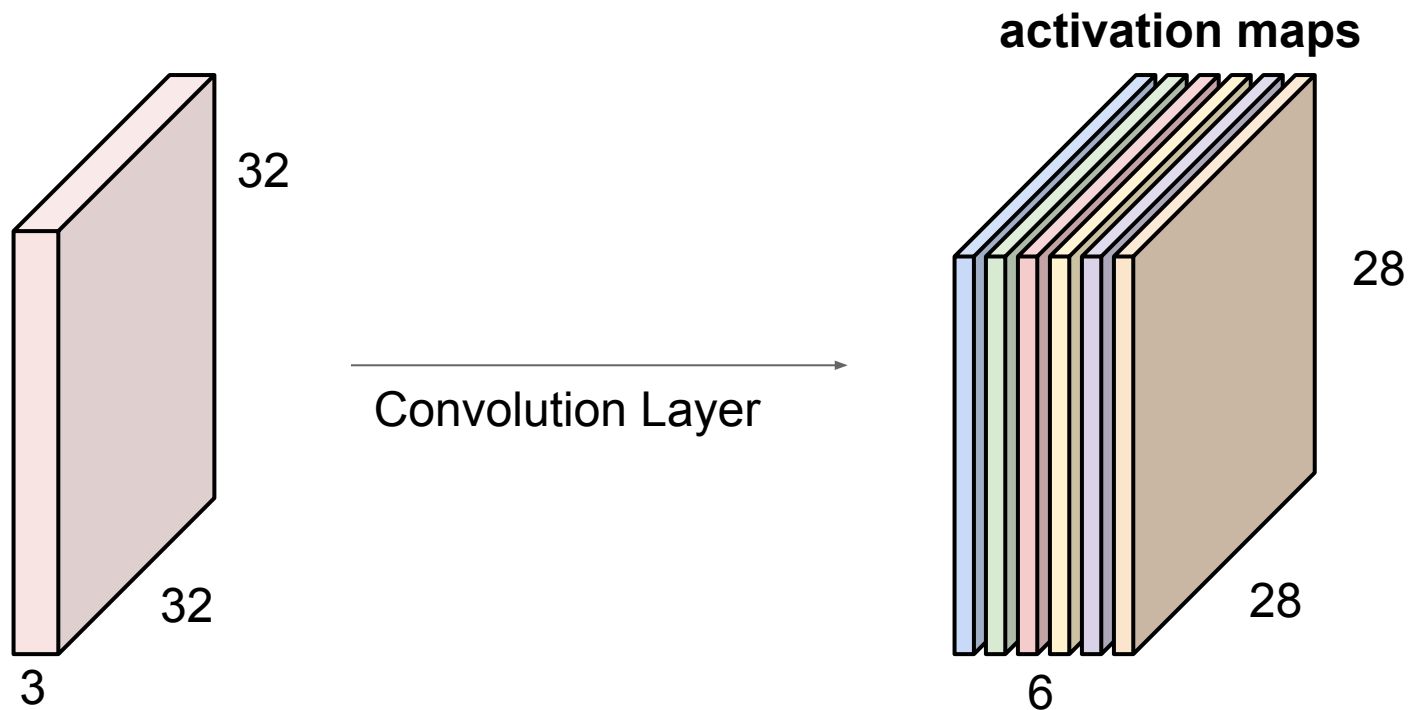


# Convolution Layer

consider a second, **green** filter



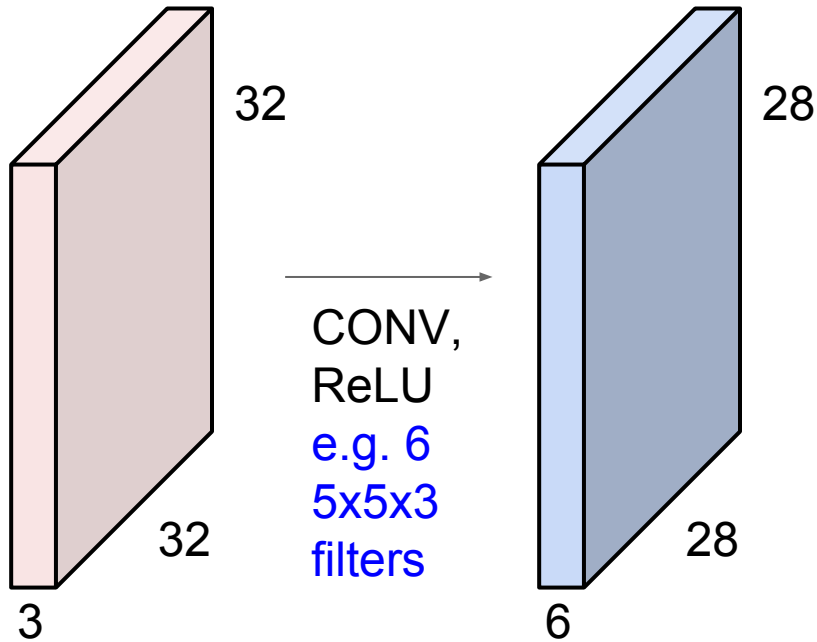
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



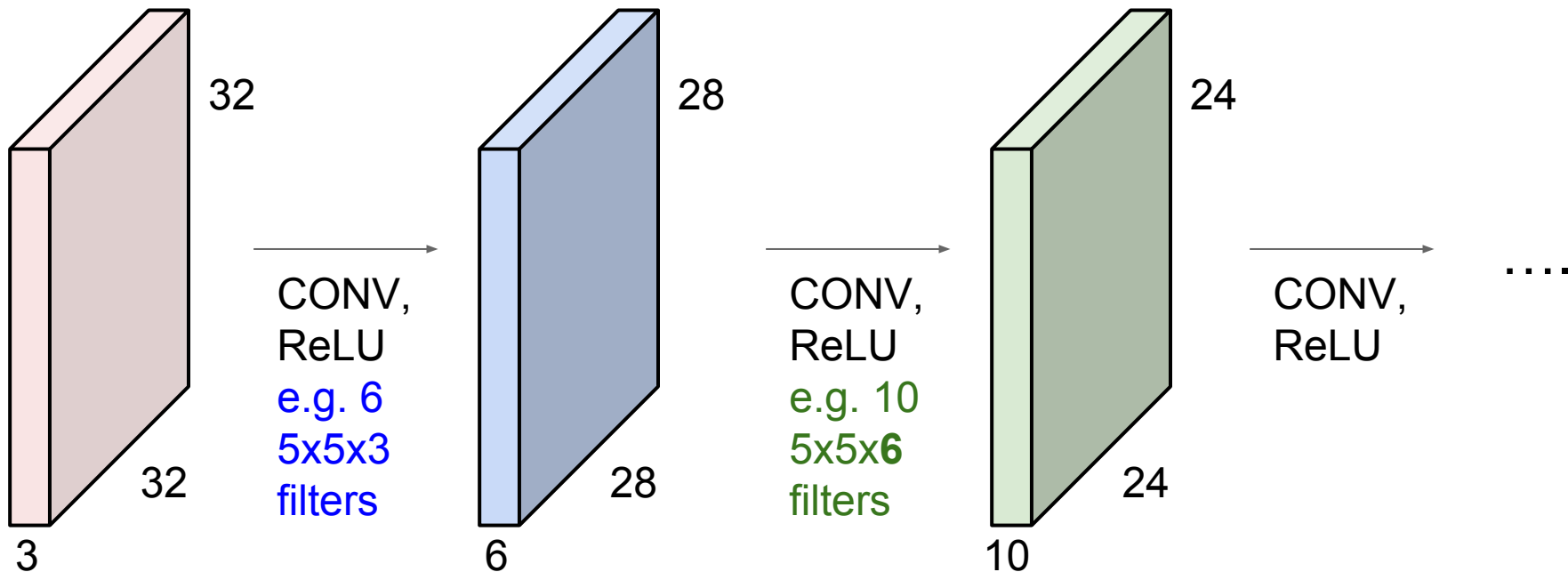
We stack these up to get a “new image” of size 28x28x6!



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

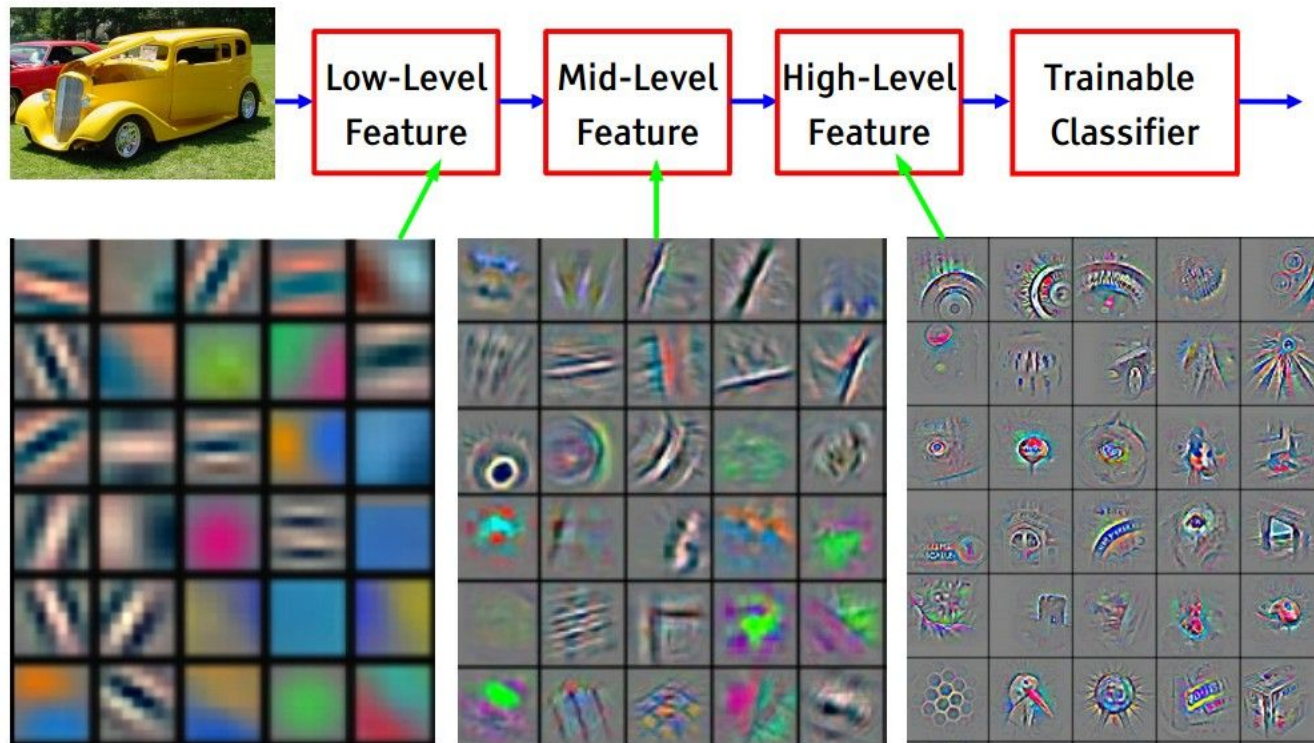


**Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



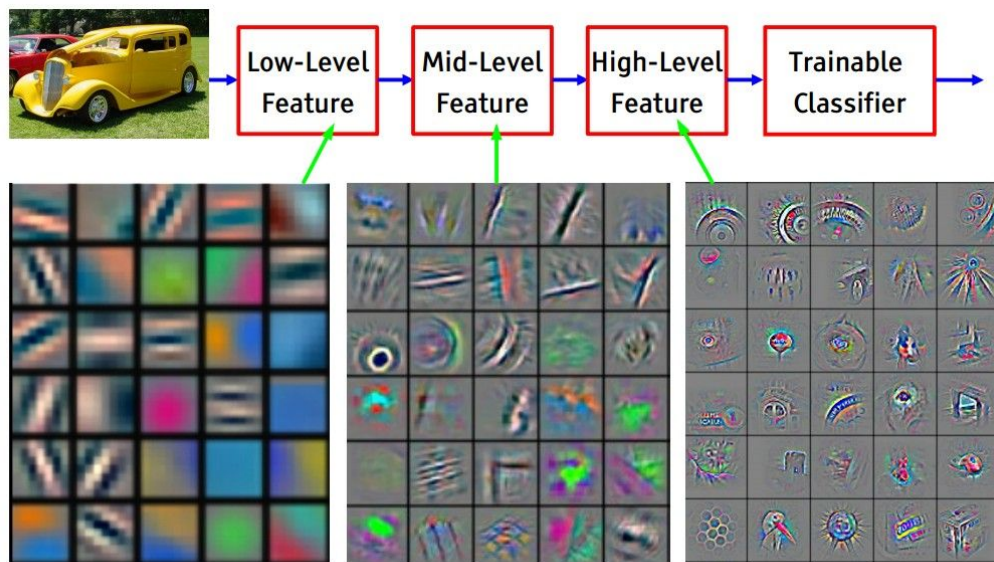
# Preview

[From recent Yann  
LeCun slides]



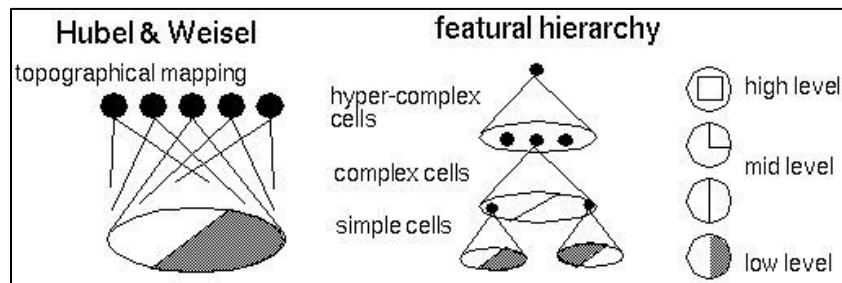
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

# Preview



[From recent Yann LeCun slides]

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

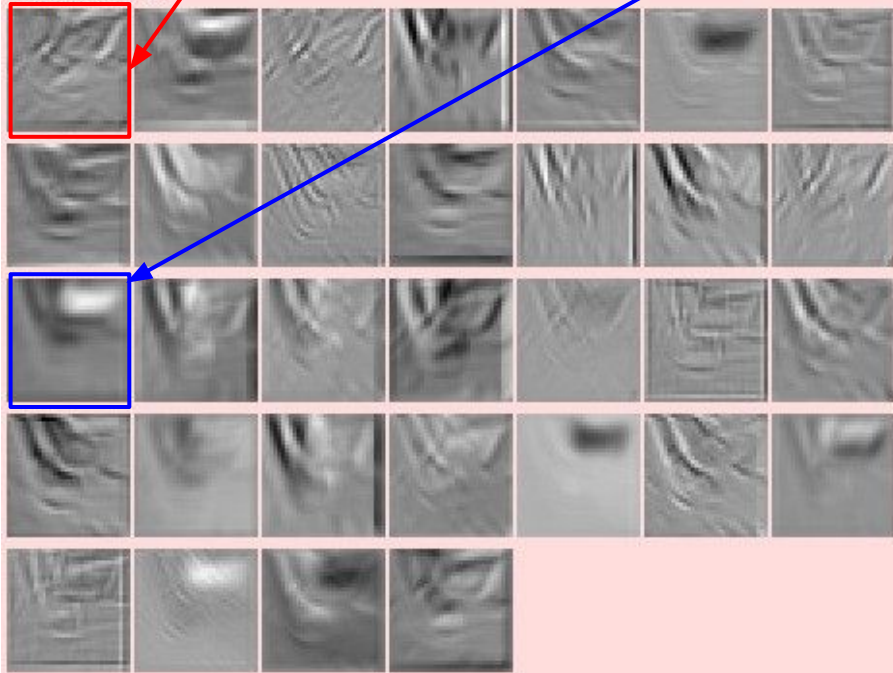




one filter =>  
one activation map

example 5x5 filters  
(32 total)

Activations:



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

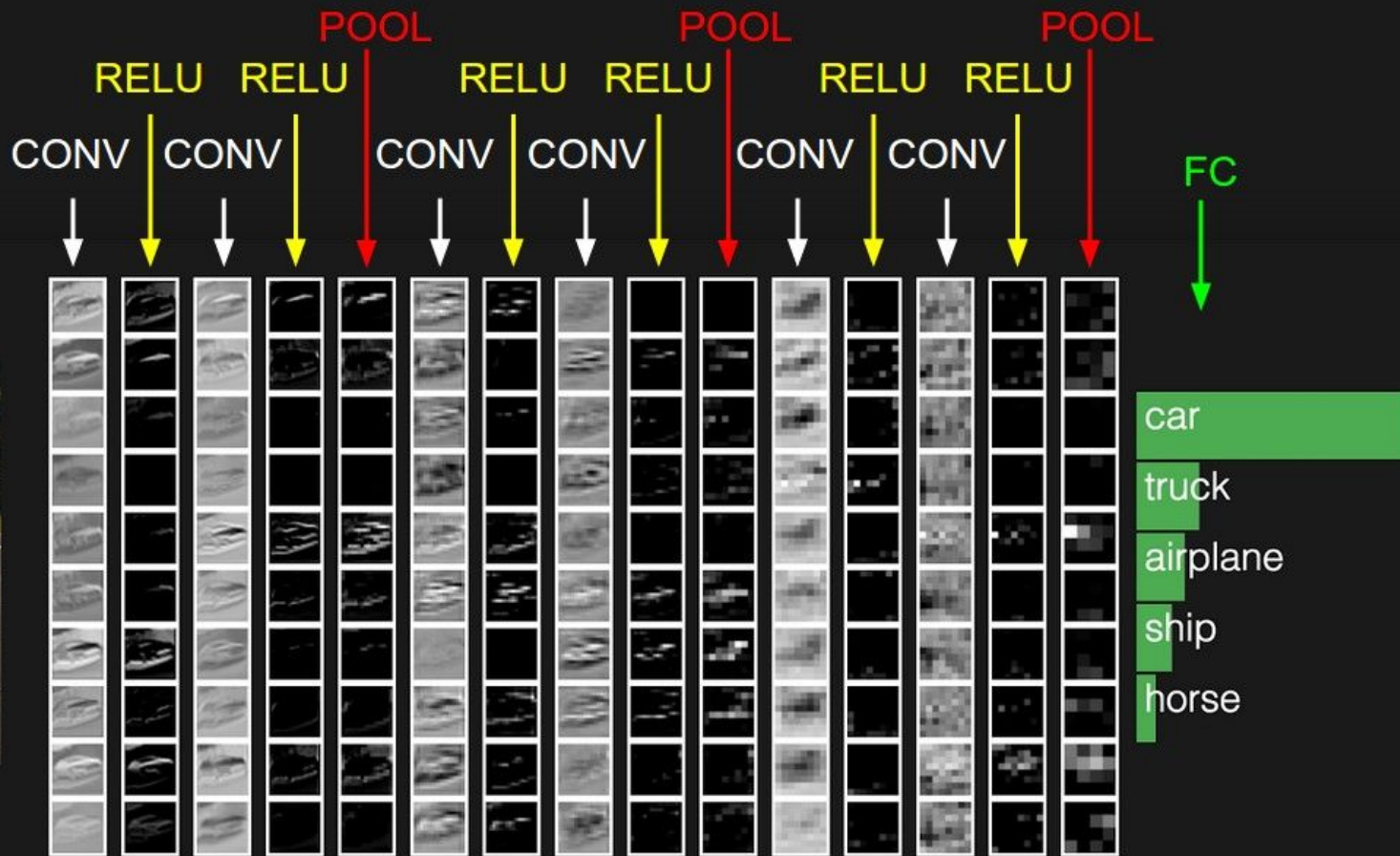
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1,y-n_2]$$



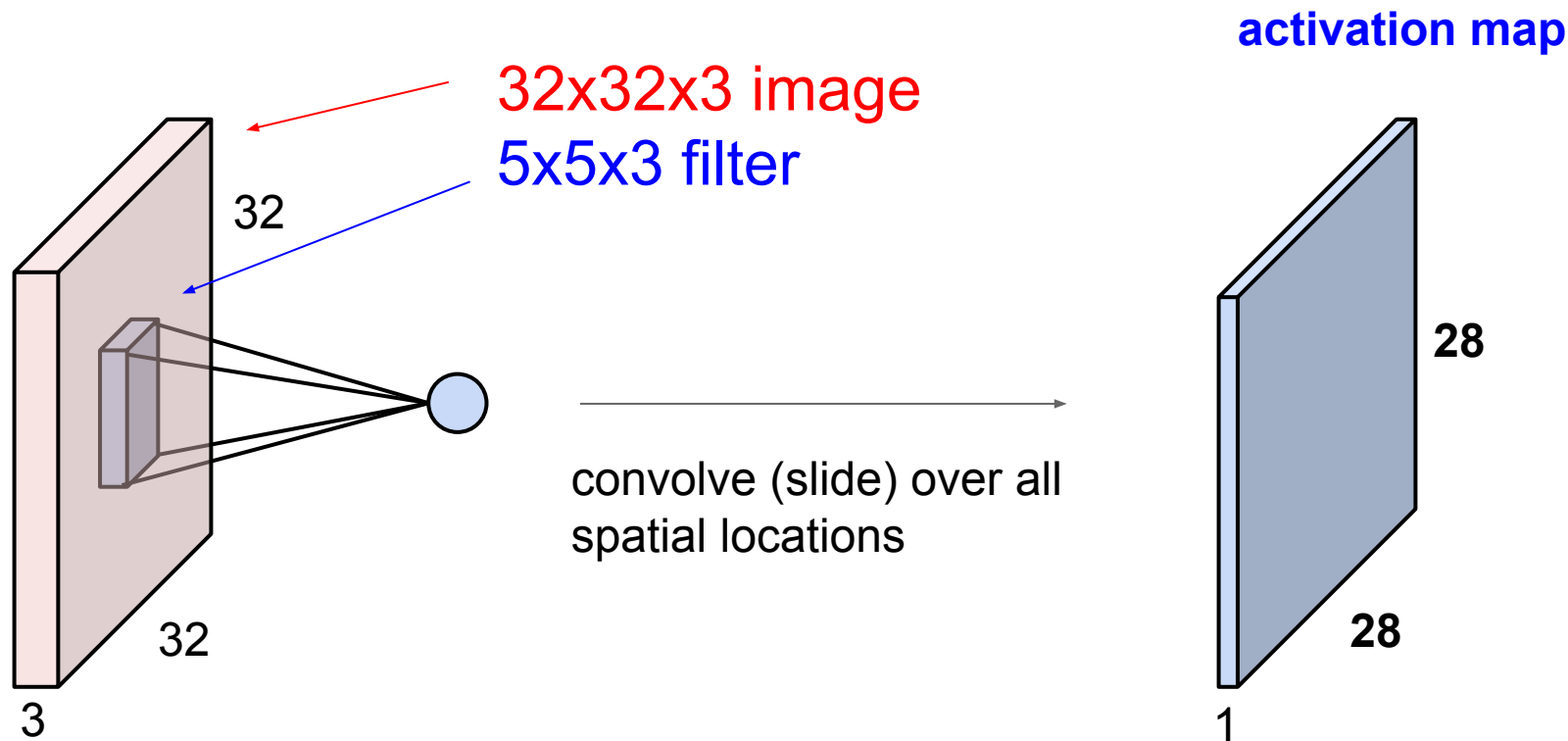
elementwise multiplication and sum of  
a filter and the signal (image)



preview:

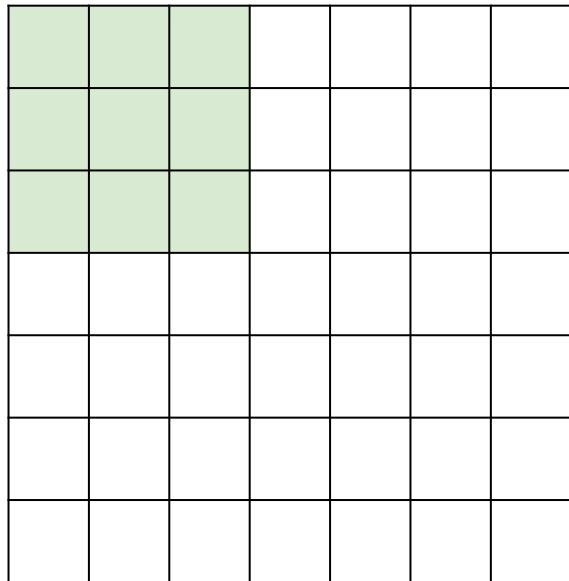


## A closer look at spatial dimensions:



## A closer look at spatial dimensions:

7



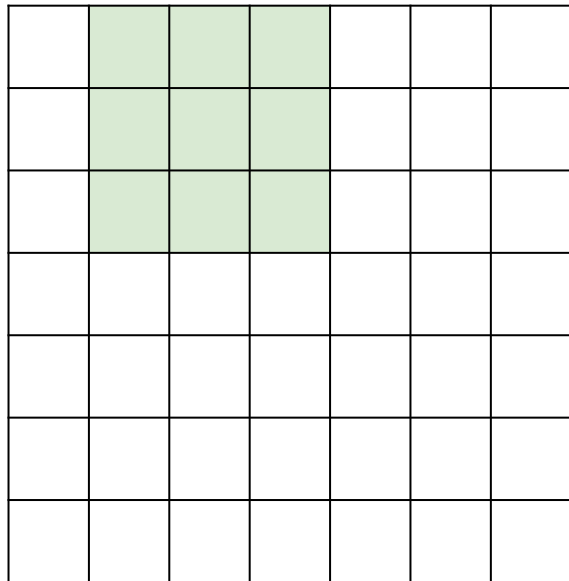
7

7x7 input (spatially)  
assume 3x3 filter



## A closer look at spatial dimensions:

7

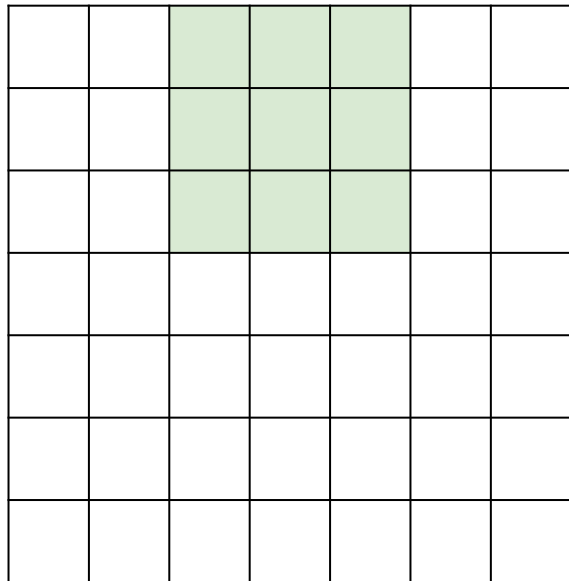


7

7x7 input (spatially)  
assume 3x3 filter

## A closer look at spatial dimensions:

7

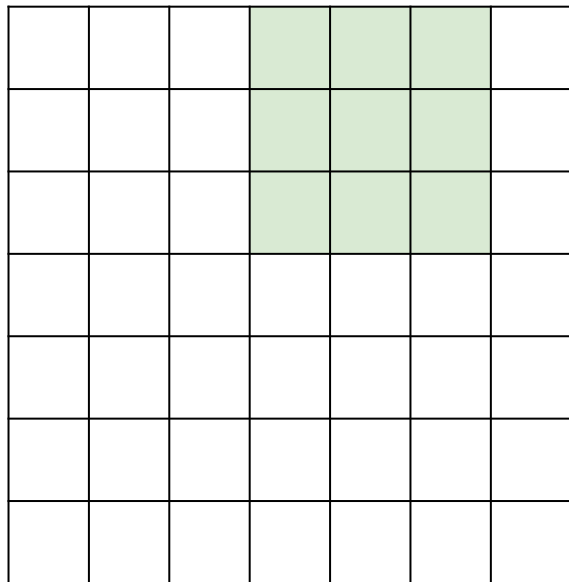


7

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:

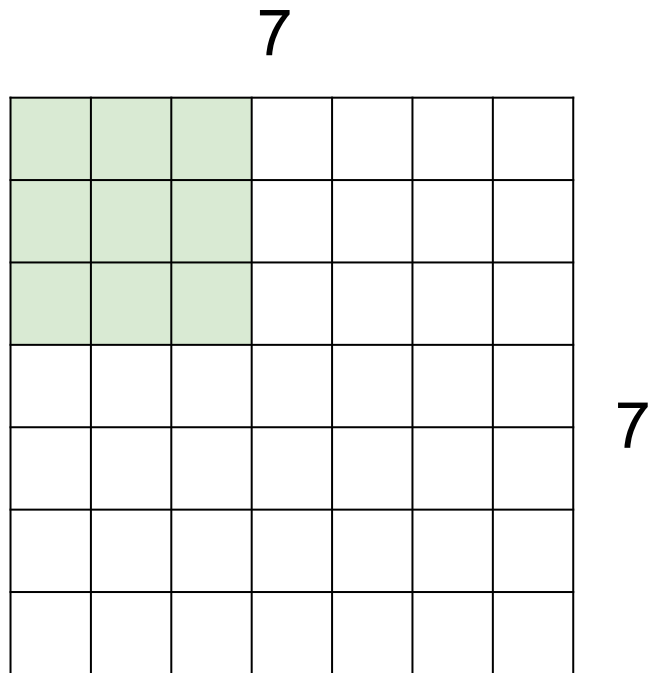
7


7

7x7 input (spatially)  
assume 3x3 filter

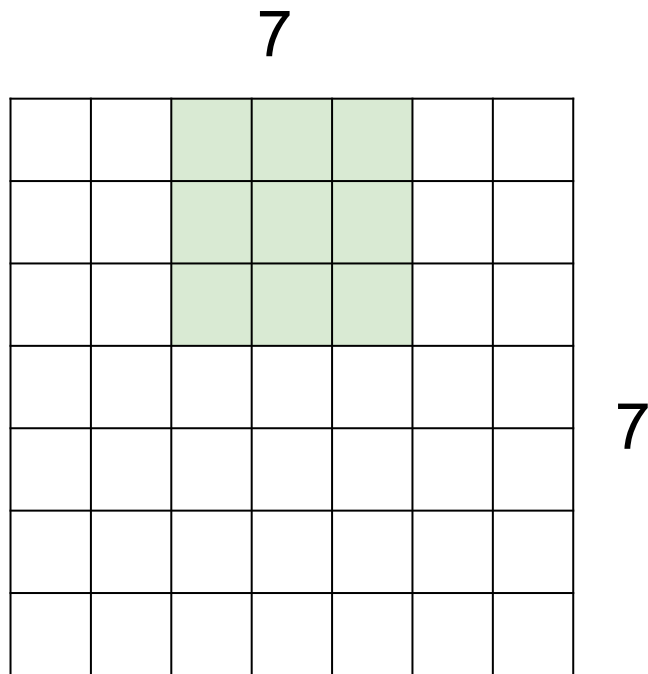
**=> 5x5 output**

## A closer look at spatial dimensions:



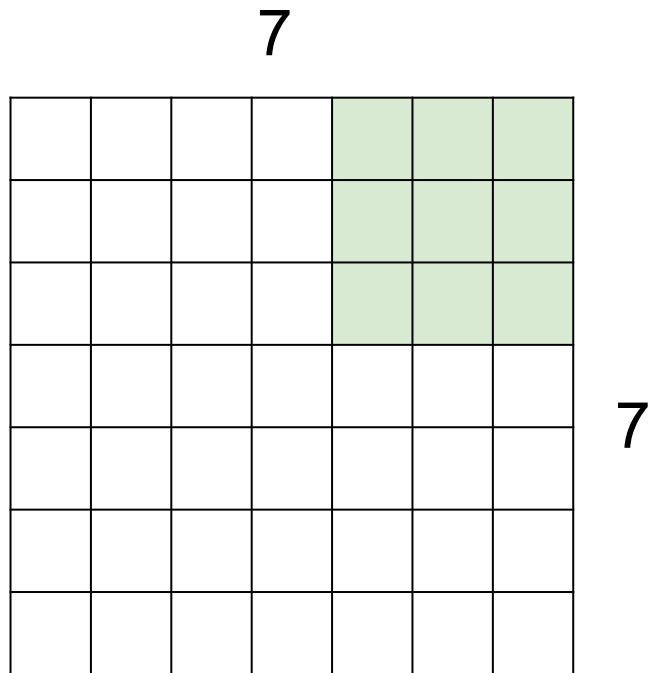
7x7 input (spatially)  
assume 3x3 filter  
applied **with stride 2**

## A closer look at spatial dimensions:



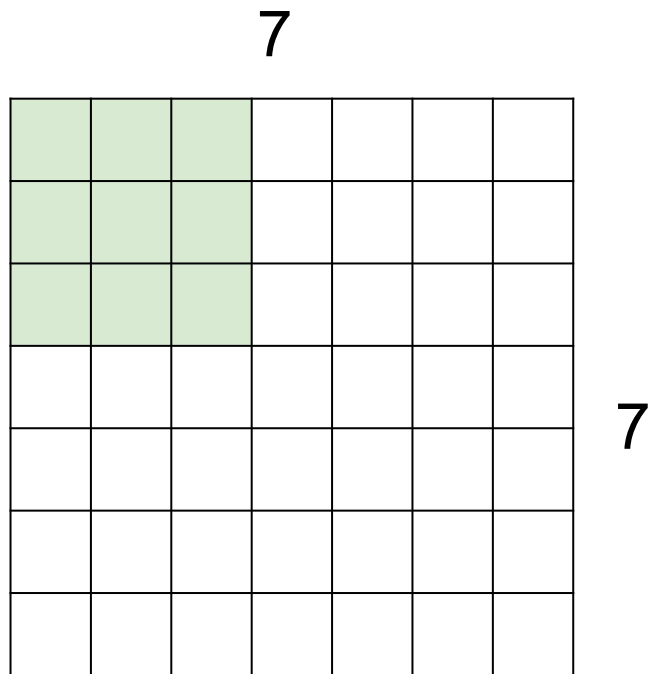
7x7 input (spatially)  
assume 3x3 filter  
applied **with stride 2**

## A closer look at spatial dimensions:



7x7 input (spatially)  
assume 3x3 filter  
applied **with stride 2**  
**=> 3x3 output!**

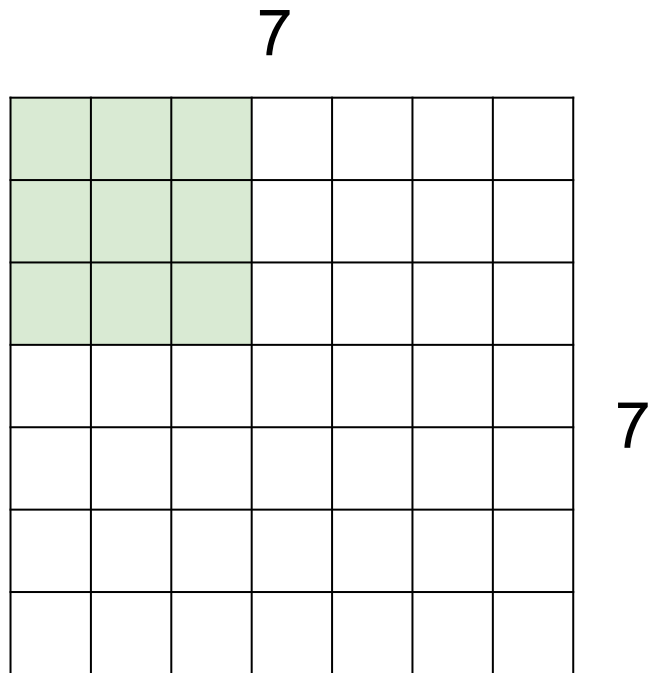
A closer look at spatial dimensions:



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

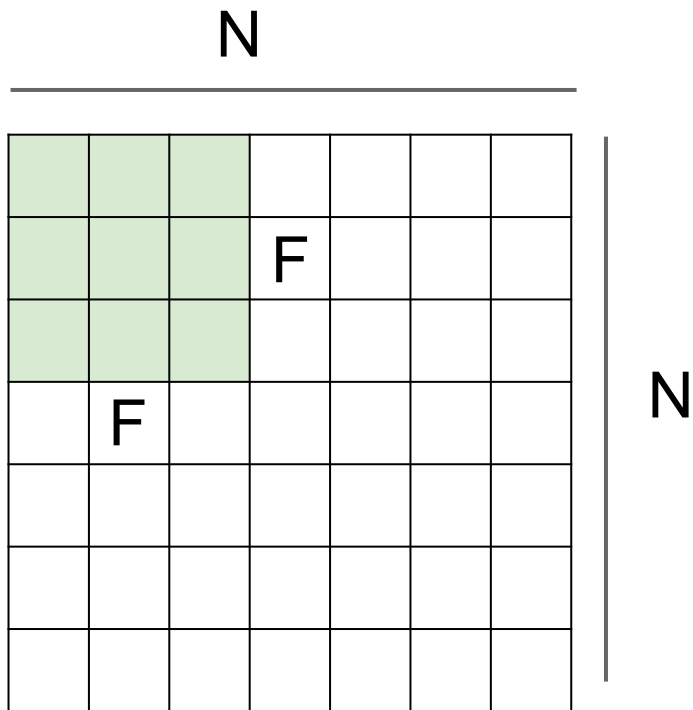


## A closer look at spatial dimensions:



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

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



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

e.g.  $N = 7, F = 3$ :

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

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

stride 3  $\Rightarrow (7 - 3) / 3 + 1 = 2.33 \therefore \backslash$

# 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?

(recall:)

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

# 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

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 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)

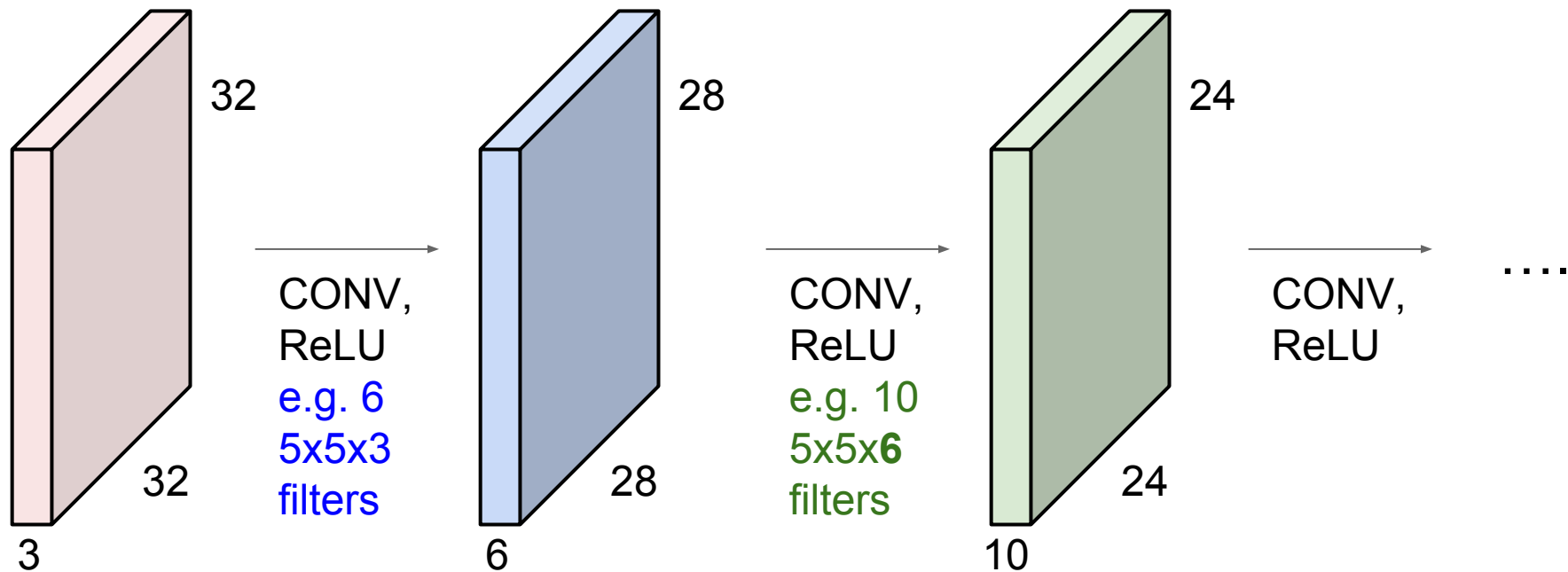
e.g.  $F = 3 \Rightarrow$  zero pad with 1

$F = 5 \Rightarrow$  zero pad with 2

$F = 7 \Rightarrow$  zero pad with 3

## Remember back to...

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32  $\rightarrow$  28  $\rightarrow$  24 ...). Shrinking too fast is not good, doesn't work well.

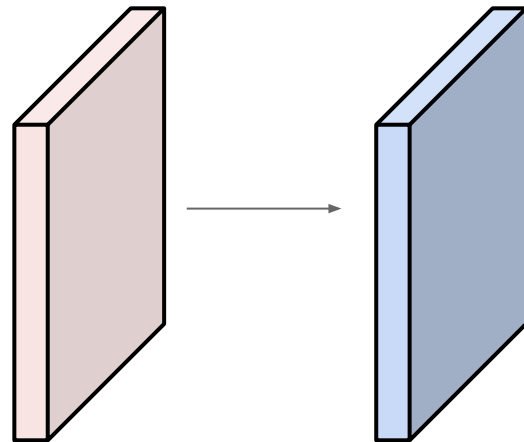


Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Output volume size: ?



Examples time:

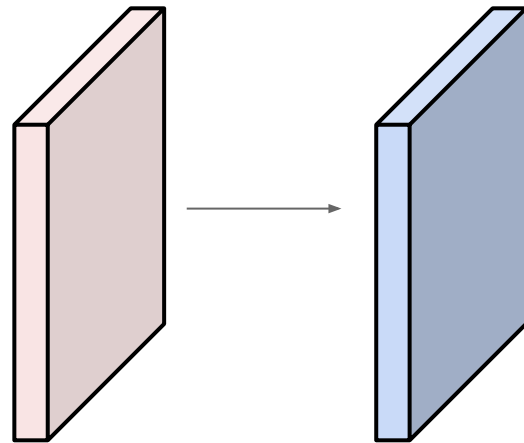
Input volume: **32x32x3**

**10** **5x5** filters with stride **1**, pad **2**

Output volume size:

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

**32x32x10**

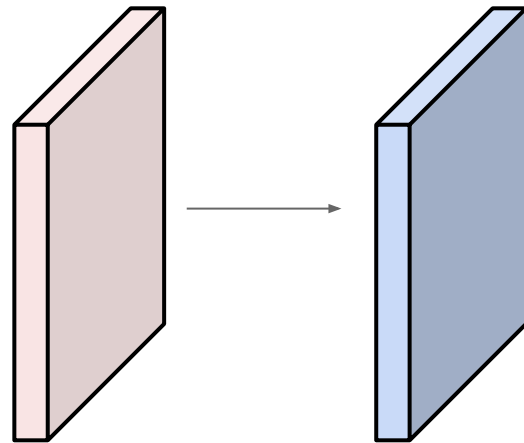




Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

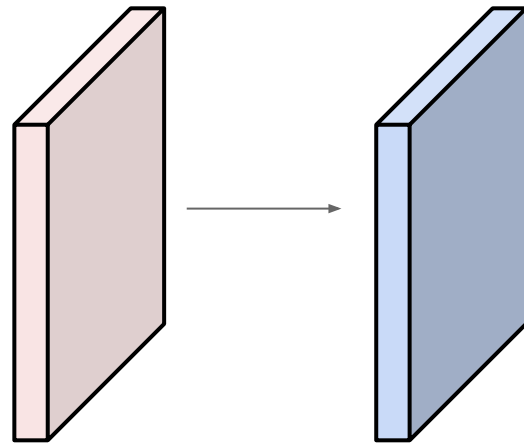


Number of parameters in this layer?

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.

## Common settings:

**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.

$K = (\text{powers of 2, e.g. 32, 64, 128, 512})$

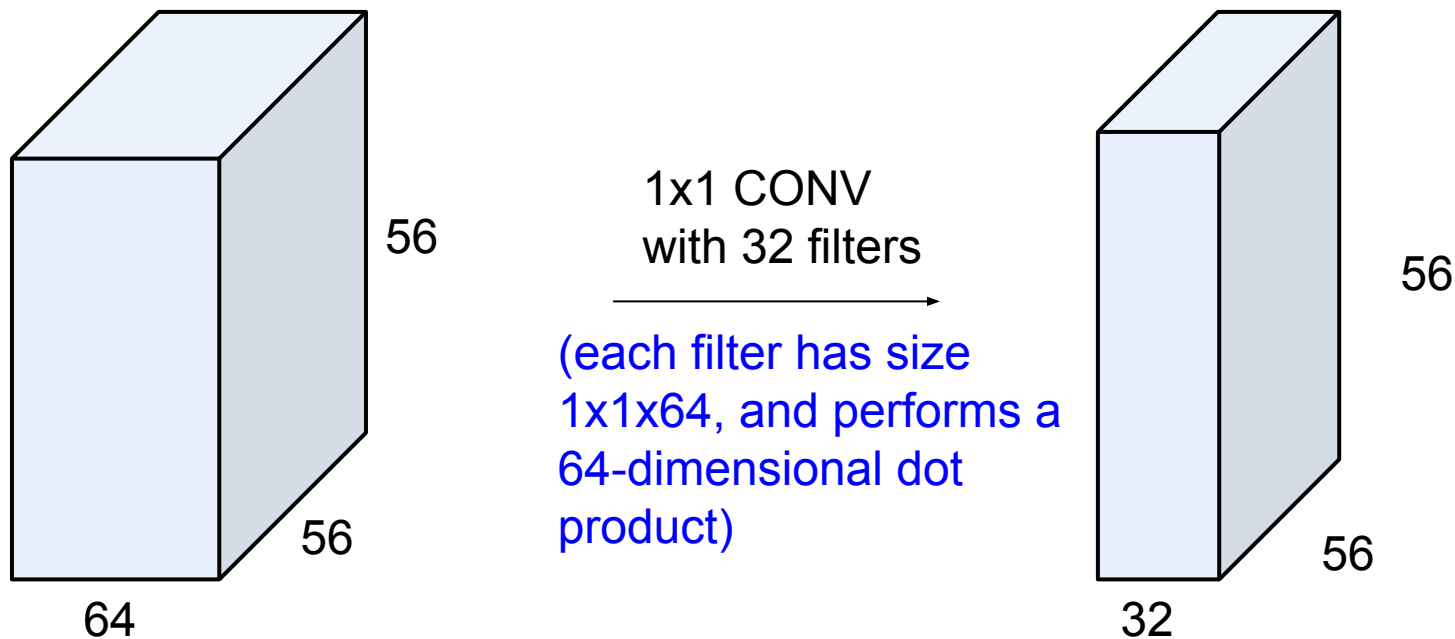
-  $F = 3, S = 1, P = 1$

-  $F = 5, S = 1, P = 2$

-  $F = 5, S = 2, P = ?$  (whatever fits)

-  $F = 1, S = 1, P = 0$

(btw, 1x1 convolution layers make perfect sense)



# Example: CONV layer in Torch

**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  $P$ .

## 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  $(kH-1)/2$ .

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)
```

# Example: CONV layer in Caffe

**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  $P$ .

```
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
    }
  }
}
```



# Example: CONV layer in Lasagne

**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  $P$ .

```
class lasagne.layers.Conv2DLayer(incoming, num_filters, filter_size, stride=(1, 1), pad=0,
untie_biases=False, W=lasagne.init.GlorotUniform(), b=lasagne.init.Constant(0),
nonlinearity=lasagne.nonlinearities.rectify, flip_filters=True, convolution=theano.tensor.nnet.conv2d,
**kwargs) [source]
```

2D convolutional layer

Performs a 2D convolution on its input and optionally adds a bias and applies an elementwise nonlinearity.

Parameters: **incoming**: a `Layer` instance or a tuple

The layer feeding into this layer, or the expected input shape. The output of this layer should be a 4D tensor, with shape `(batch_size, num_input_channels, input_rows, input_columns)`.

**num\_filters**: int

The number of learnable convolutional filters this layer has.

**filter\_size**: int or iterable of int

An integer or a 2-element tuple specifying the size of the filters.

**stride**: int or iterable of int

An integer or a 2-element tuple specifying the stride of the convolution operation.

**pad**: int, iterable of int, 'full', 'same' or 'valid' (default: 0)

By default, the convolution is only computed where the input and the filter fully overlap (a valid convolution). When `stride=1`, this yields an output that is smaller than the input by `filter_size - 1`. The `pad` argument allows you to implicitly pad the input with zeros, extending the output size.

A single integer results in symmetric zero-padding of the given size on all borders, a tuple of two integers allows different symmetric padding per dimension.

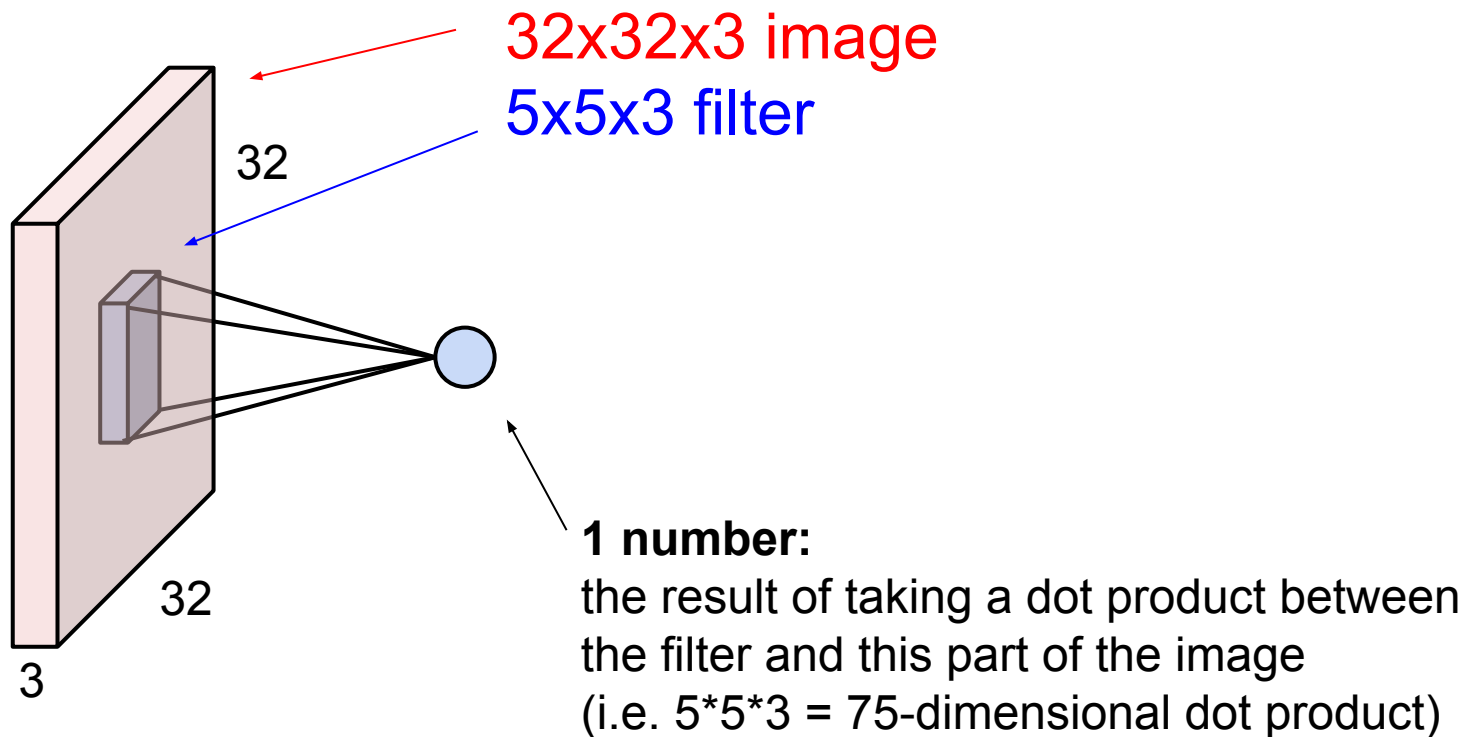
`'full'` pads with one less than the filter size on both sides. This is equivalent to computing the convolution wherever the input and the filter overlap by at least one position.

`'same'` pads with half the filter size (rounded down) on both sides. When `stride=1` this results in an output size equal to the input size. Even filter size is not supported.

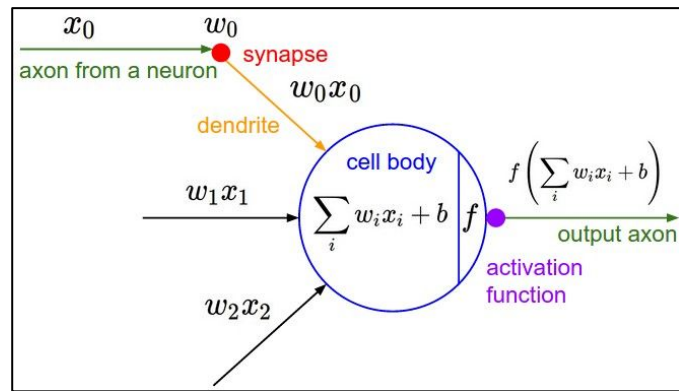
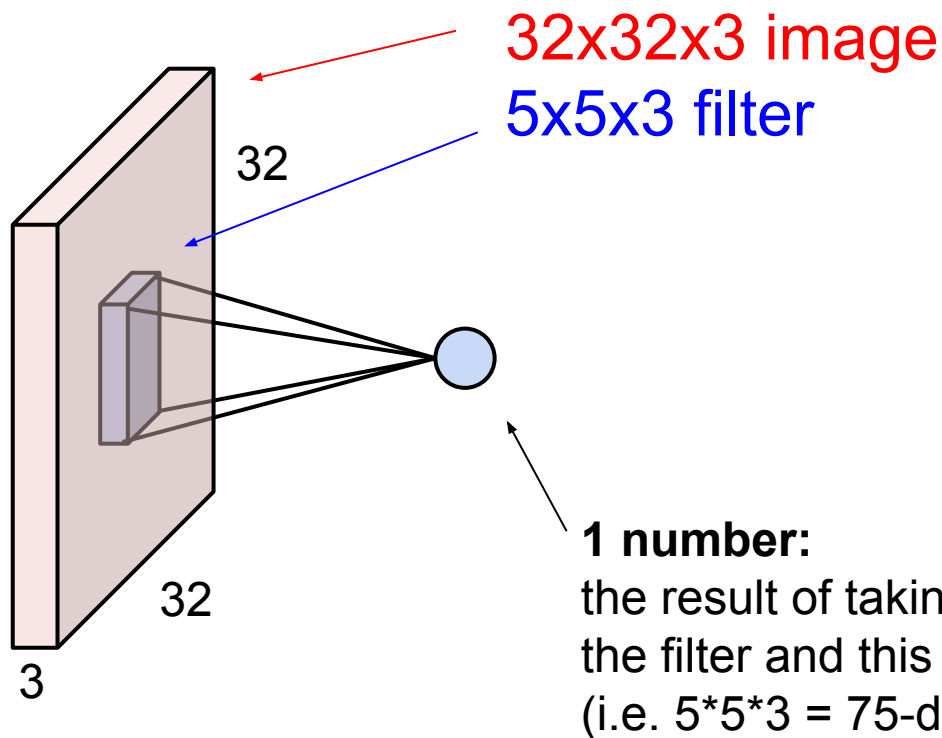
`'valid'` is an alias for `0` (no padding / a valid convolution).



# The brain/neuron view of CONV Layer

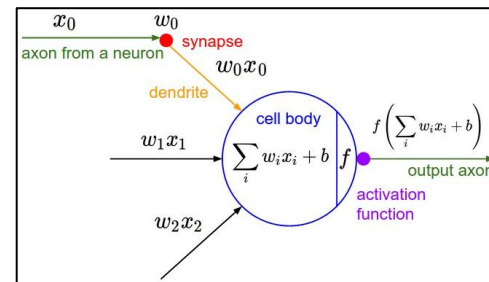
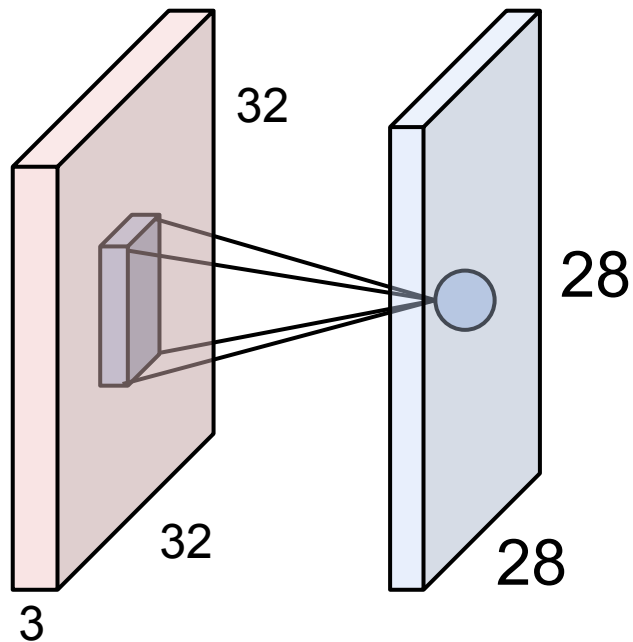


# The brain/neuron view of CONV Layer



It's just a neuron with local connectivity...

# The brain/neuron view of CONV Layer

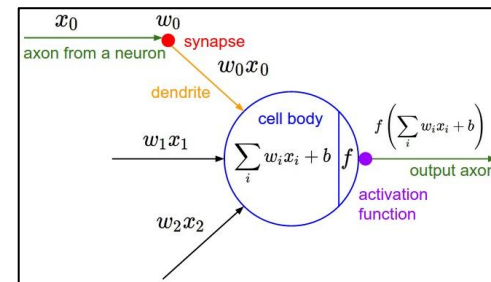
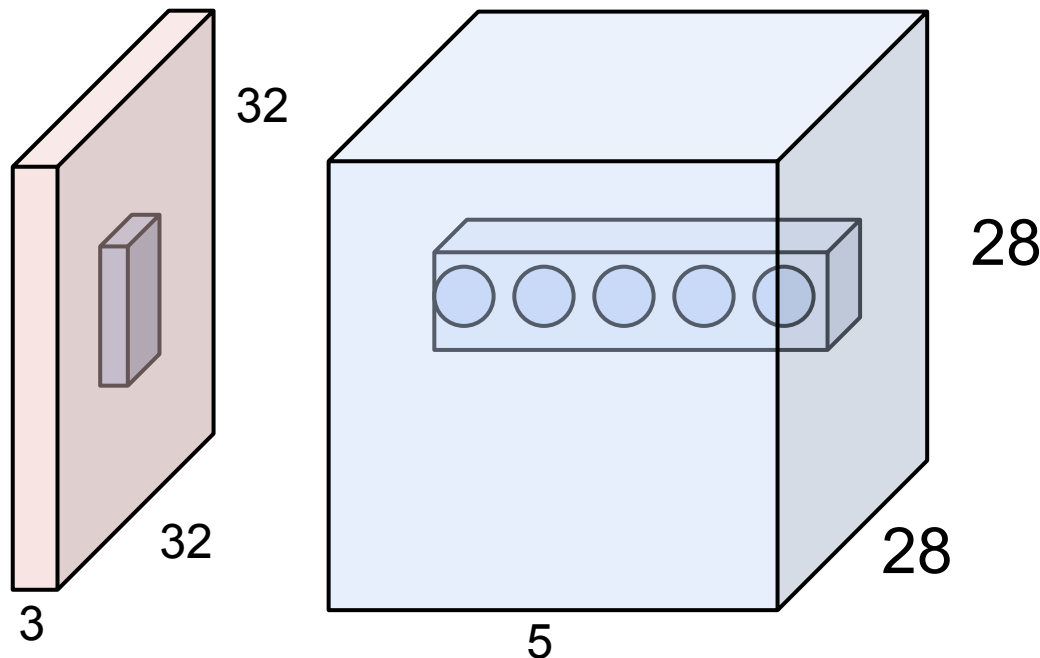


An activation map is a 28x28 sheet of neuron outputs:

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



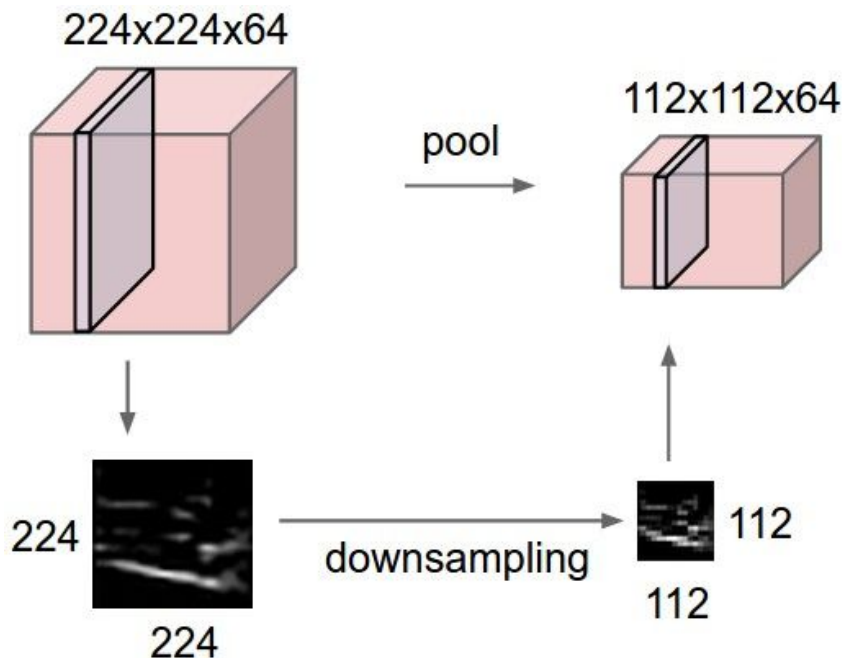
E.g. with 5 filters,  
CONV layer consists of  
neurons arranged in a 3D grid  
(28x28x5)

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

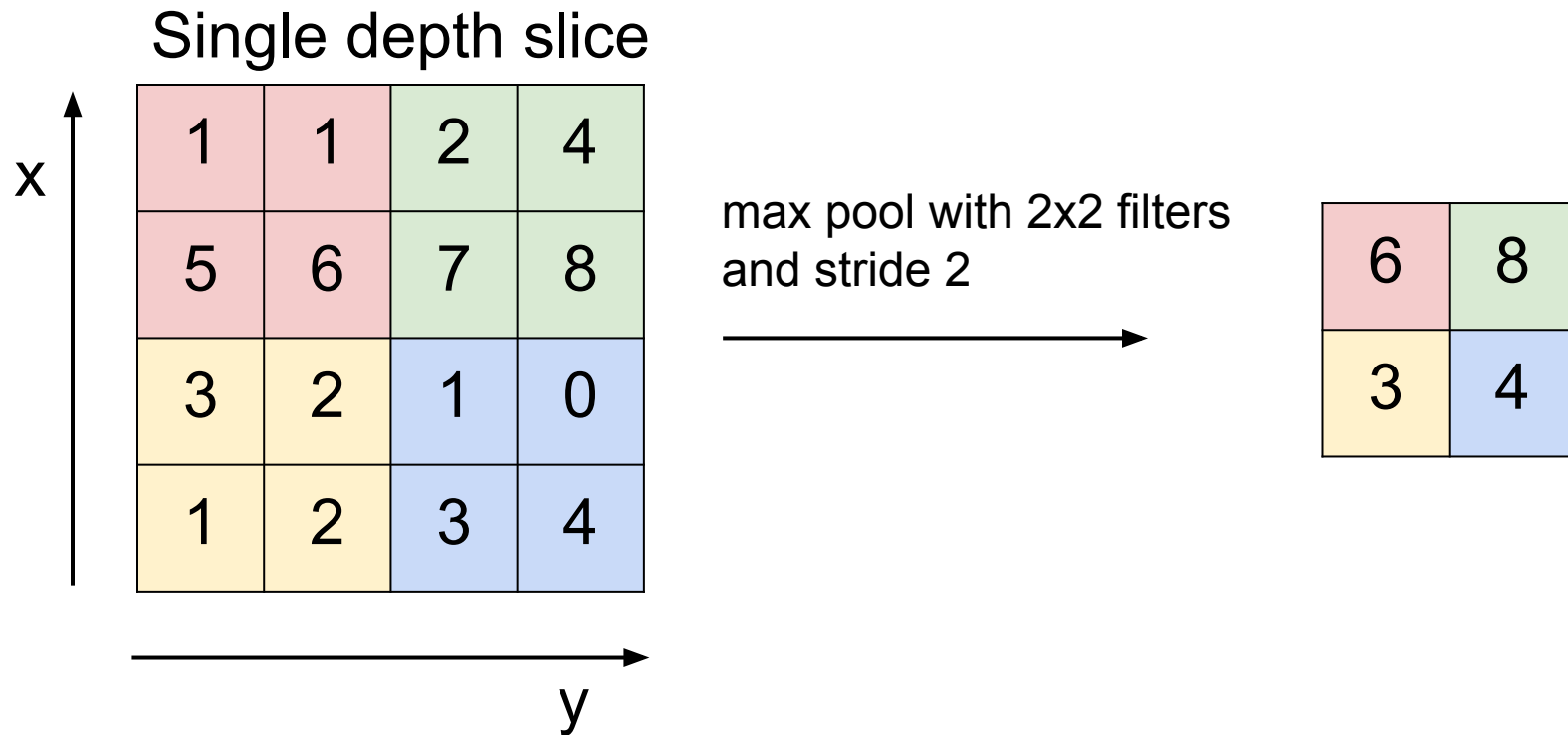
A white Tesla Model S is shown from a front-three-quarter view, driving on a paved road. The car is in motion, as indicated by the blurred background of green trees and distant mountains. The car's sleek design, including its large glass windows and distinctive front grille with the Tesla logo, is clearly visible. The lighting suggests a bright, sunny day.

# Pooling layer

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



# MAX POOLING



- 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



## Common settings:

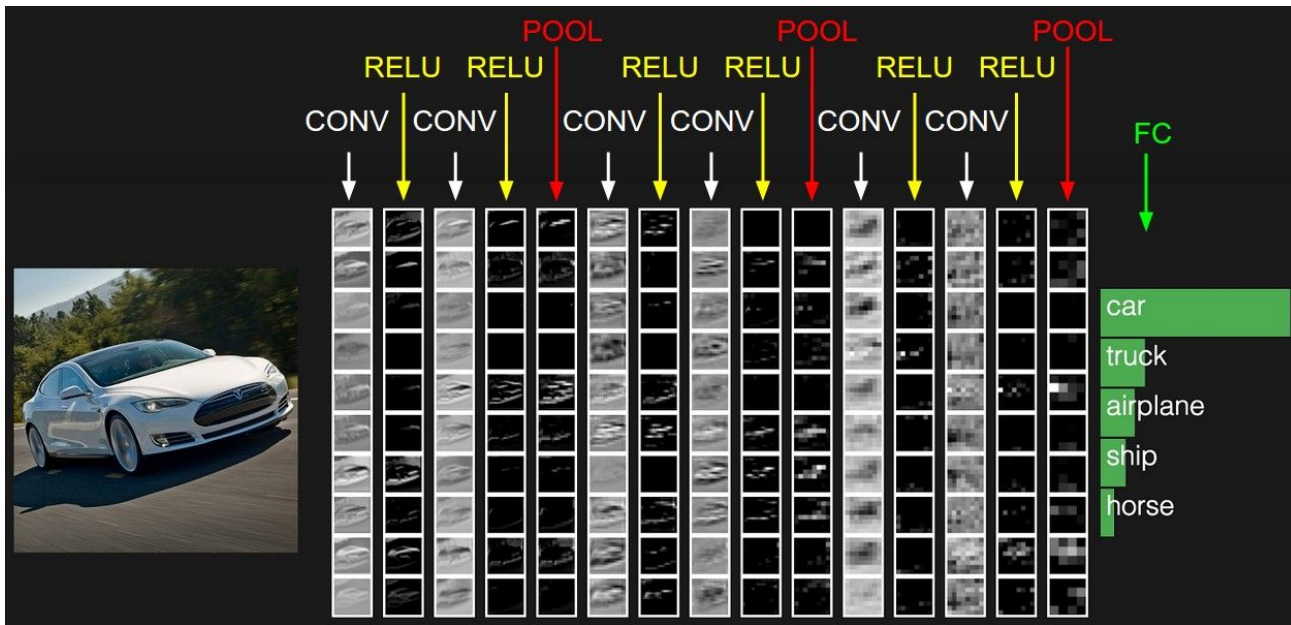
$$F = 2, S = 2$$

$$F = 3, S = 2$$

- 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

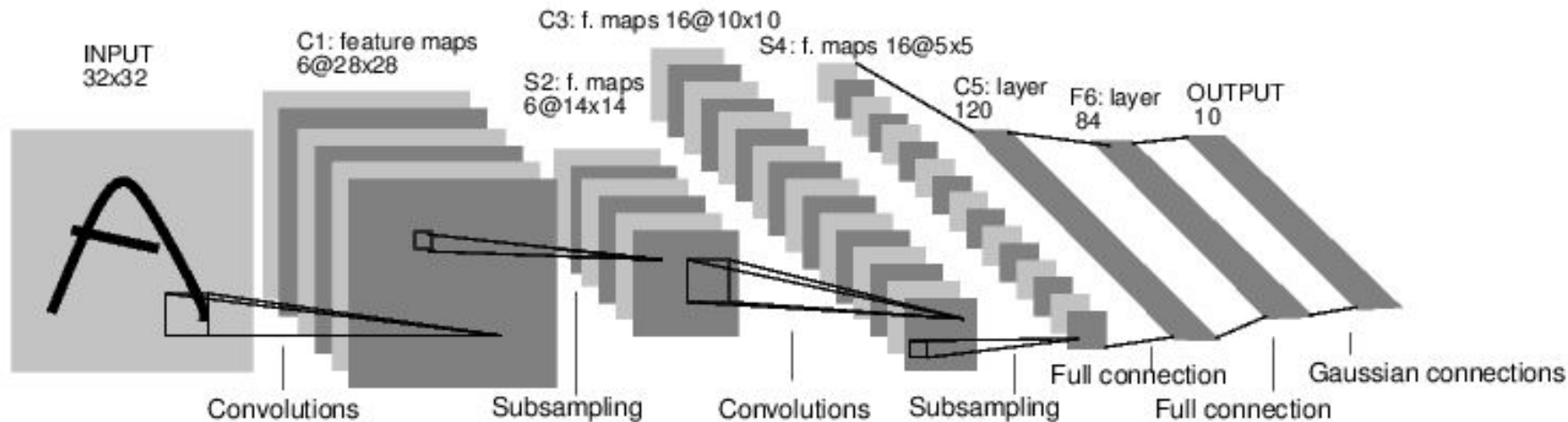


[ConvNetJS demo: training on CIFAR-10]

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

# Case Study: LeNet-5

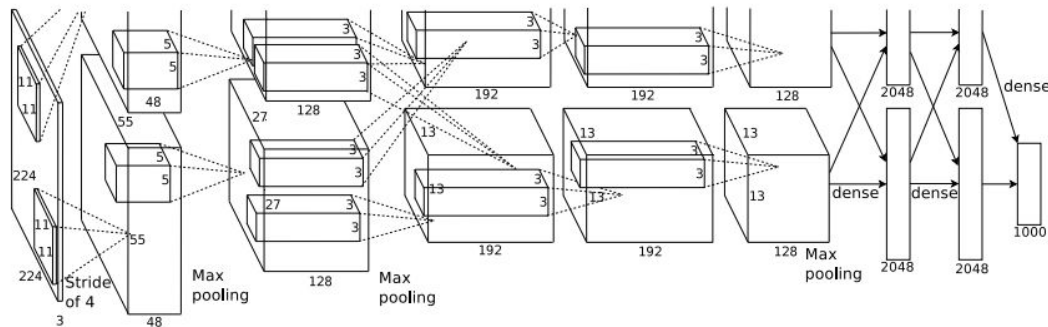
[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1  
Subsampling (Pooling) layers were 2x2 applied at stride 2  
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

# Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

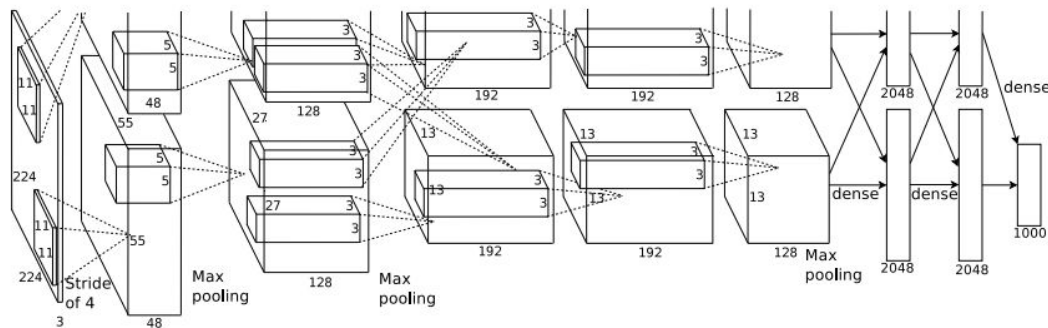
**First layer (CONV1):** 96 11x11 filters applied at stride 4

=>

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

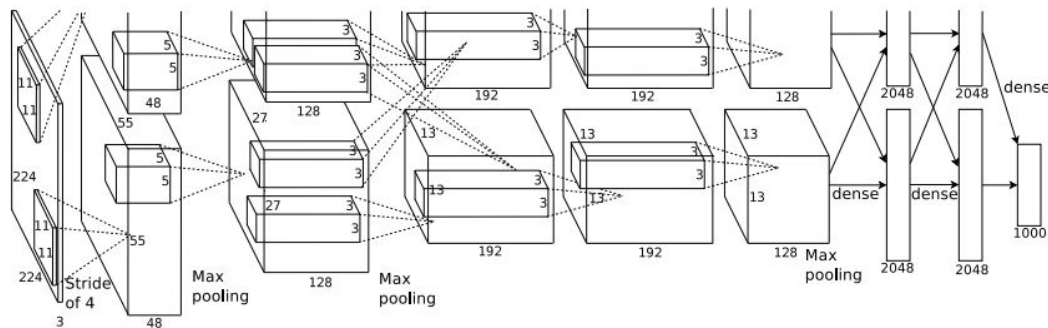
=>

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

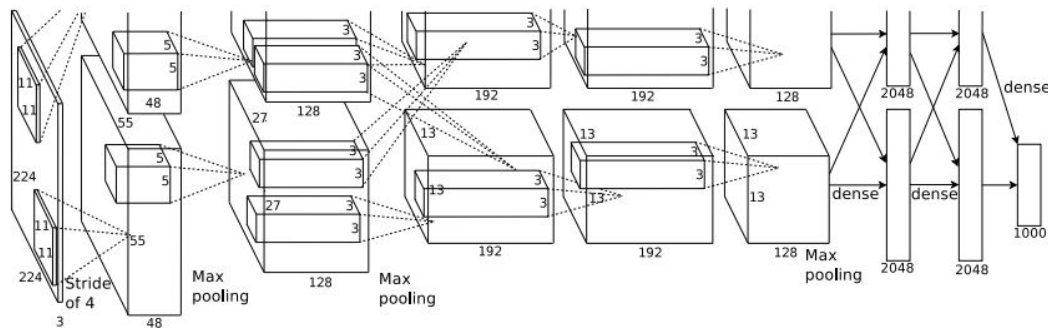
=>

Output volume **[55x55x96]**

Parameters:  $(11*11*3)*96 = \mathbf{35K}$

# Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

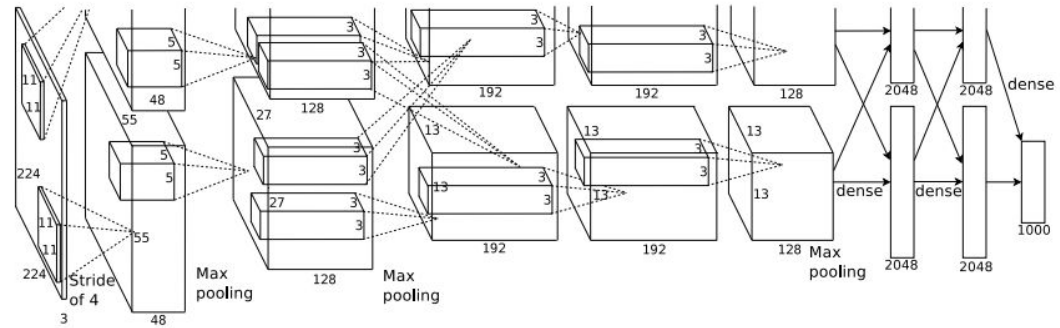
**Second layer (POOL1):** 3x3 filters applied at stride 2

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



# Case Study: AlexNet

[Krizhevsky et al. 2012]



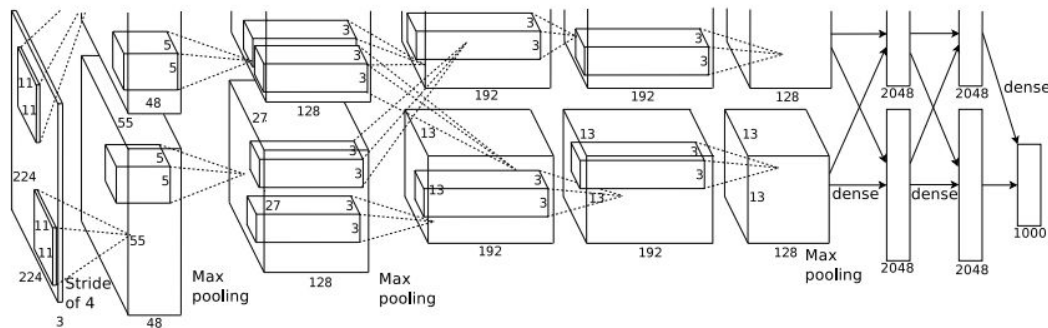
Input: 227x227x3 images  
After CONV1: 55x55x96

**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

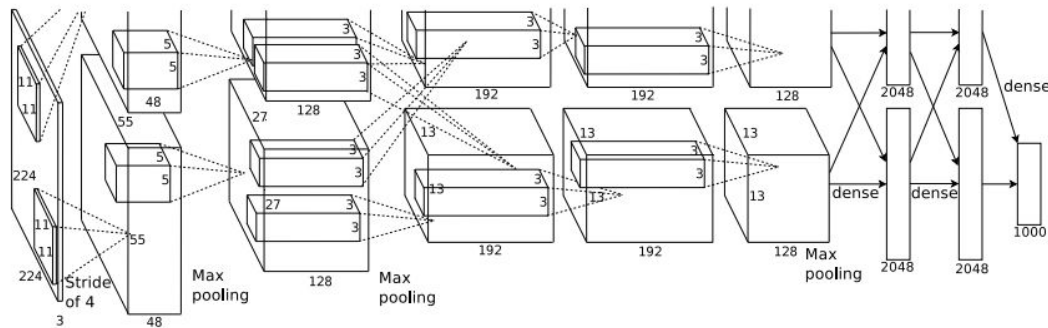
**Second layer (POOL1):** 3x3 filters applied at stride 2

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

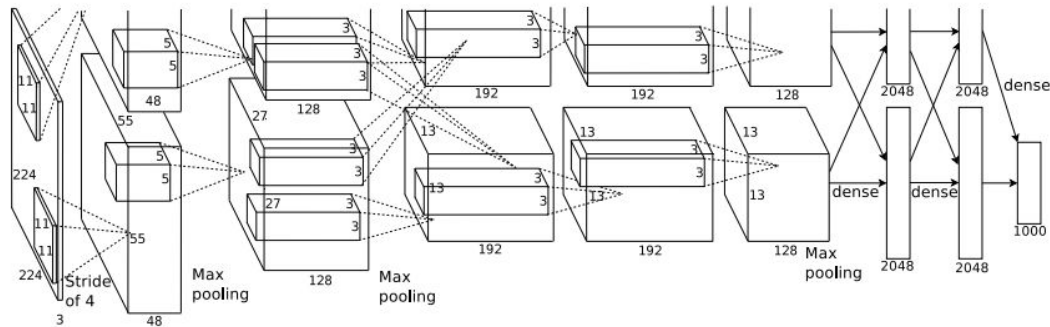
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

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

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)



# Case Study: 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

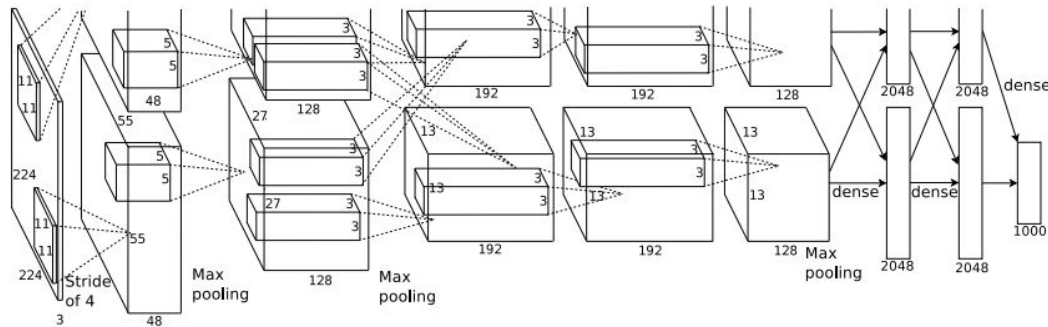
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

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

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)

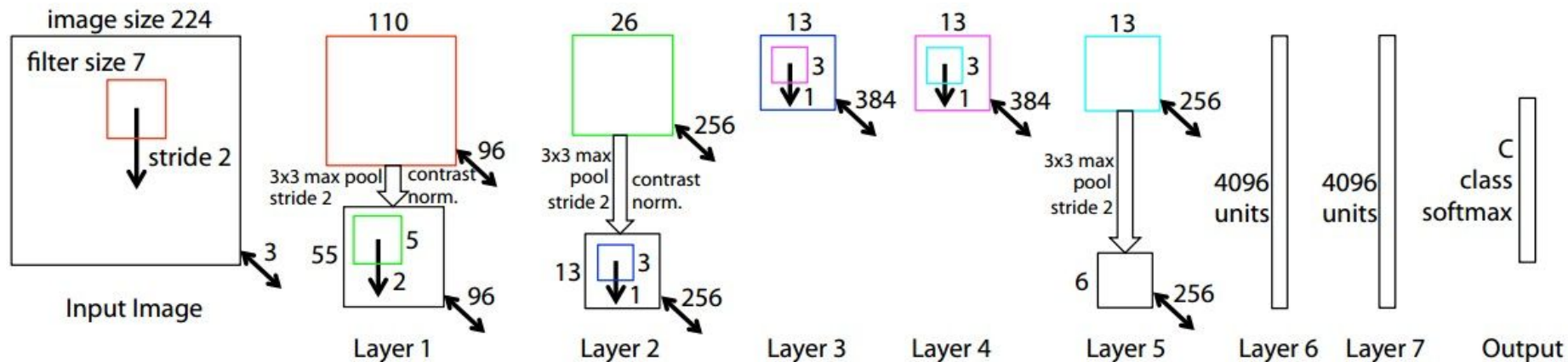


## Details/Retrospectives:

- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

# Case Study: ZFNet

[Zeiler and Fergus, 2013]



AlexNet but:

CONV1: change from (11x11 stride 4) to (7x7 stride 2)

CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 15.4% -> 14.8%

# 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 ( $224 \times 224$ RGB image)					
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
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
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
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
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:  $224*224*3=150\text{K}$  params: 0 (not counting biases)

CONV3-64: [224x224x64] memory:  $224*224*64=3.2\text{M}$  params:  $(3*3*3)*64 = 1,728$

CONV3-64: [224x224x64] memory:  $224*224*64=3.2\text{M}$  params:  $(3*3*64)*64 = 36,864$

POOL2: [112x112x64] memory:  $112*112*64=800\text{K}$  params: 0

CONV3-128: [112x112x128] memory:  $112*112*128=1.6\text{M}$  params:  $(3*3*64)*128 = 73,728$

CONV3-128: [112x112x128] memory:  $112*112*128=1.6\text{M}$  params:  $(3*3*128)*128 = 147,456$

POOL2: [56x56x128] memory:  $56*56*128=400\text{K}$  params: 0

CONV3-256: [56x56x256] memory:  $56*56*256=800\text{K}$  params:  $(3*3*128)*256 = 294,912$

CONV3-256: [56x56x256] memory:  $56*56*256=800\text{K}$  params:  $(3*3*256)*256 = 589,824$

CONV3-256: [56x56x256] memory:  $56*56*256=800\text{K}$  params:  $(3*3*256)*256 = 589,824$

POOL2: [28x28x256] memory:  $28*28*256=200\text{K}$  params: 0

CONV3-512: [28x28x512] memory:  $28*28*512=400\text{K}$  params:  $(3*3*256)*512 = 1,179,648$

CONV3-512: [28x28x512] memory:  $28*28*512=400\text{K}$  params:  $(3*3*512)*512 = 2,359,296$

CONV3-512: [28x28x512] memory:  $28*28*512=400\text{K}$  params:  $(3*3*512)*512 = 2,359,296$

POOL2: [14x14x512] memory:  $14*14*512=100\text{K}$  params: 0

CONV3-512: [14x14x512] memory:  $14*14*512=100\text{K}$  params:  $(3*3*512)*512 = 2,359,296$

CONV3-512: [14x14x512] memory:  $14*14*512=100\text{K}$  params:  $(3*3*512)*512 = 2,359,296$

CONV3-512: [14x14x512] memory:  $14*14*512=100\text{K}$  params:  $(3*3*512)*512 = 2,359,296$

POOL2: [7x7x512] memory:  $7*7*512=25\text{K}$  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	
13 weight layers	16 weight layers	16 weight layers	19
Input (224 × 224 RGB image)			
conv3-64	conv3-64	conv3-64	conv3-64
<b>conv3-64</b>	conv3-64	conv3-64	conv3-64
maxpool			
conv3-128	conv3-128	conv3-128	conv3-128
<b>conv3-128</b>	conv3-128	conv3-128	conv3-128
maxpool			
conv3-256	conv3-256	conv3-256	conv3-256
conv3-256	conv3-256	conv3-256	conv3-256
	<b>conv1-256</b>	<b>conv3-256</b>	conv3-256
maxpool			
conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512
	<b>conv1-512</b>	<b>conv3-512</b>	conv3-512
maxpool			
conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512
	<b>conv1-512</b>	<b>conv3-512</b>	conv3-512
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			



INPUT: [224x224x3] memory:  $224*224*3=150\text{K}$  params: 0 (not counting biases)

CONV3-64: [224x224x64] memory:  $224*224*64=3.2\text{M}$  params:  $(3*3*3)*64 = 1,728$

CONV3-64: [224x224x64] memory:  $224*224*64=3.2\text{M}$  params:  $(3*3*64)*64 = 36,864$

POOL2: [112x112x64] memory:  $112*112*64=800\text{K}$  params: 0

CONV3-128: [112x112x128] memory:  $112*112*128=1.6\text{M}$  params:  $(3*3*64)*128 = 73,728$

CONV3-128: [112x112x128] memory:  $112*112*128=1.6\text{M}$  params:  $(3*3*128)*128 = 147,456$

POOL2: [56x56x128] memory:  $56*56*128=400\text{K}$  params: 0

CONV3-256: [56x56x256] memory:  $56*56*256=800\text{K}$  params:  $(3*3*128)*256 = 294,912$

CONV3-256: [56x56x256] memory:  $56*56*256=800\text{K}$  params:  $(3*3*256)*256 = 589,824$

CONV3-256: [56x56x256] memory:  $56*56*256=800\text{K}$  params:  $(3*3*256)*256 = 589,824$

POOL2: [28x28x256] memory:  $28*28*256=200\text{K}$  params: 0

CONV3-512: [28x28x512] memory:  $28*28*512=400\text{K}$  params:  $(3*3*256)*512 = 1,179,648$

CONV3-512: [28x28x512] memory:  $28*28*512=400\text{K}$  params:  $(3*3*512)*512 = 2,359,296$

CONV3-512: [28x28x512] memory:  $28*28*512=400\text{K}$  params:  $(3*3*512)*512 = 2,359,296$

POOL2: [14x14x512] memory:  $14*14*512=100\text{K}$  params: 0

CONV3-512: [14x14x512] memory:  $14*14*512=100\text{K}$  params:  $(3*3*512)*512 = 2,359,296$

CONV3-512: [14x14x512] memory:  $14*14*512=100\text{K}$  params:  $(3*3*512)*512 = 2,359,296$

CONV3-512: [14x14x512] memory:  $14*14*512=100\text{K}$  params:  $(3*3*512)*512 = 2,359,296$

POOL2: [7x7x512] memory:  $7*7*512=25\text{K}$  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:  $24\text{M} * 4 \text{ bytes} \approx 93\text{MB} / \text{image}$  (only forward!  $\sim 2$  for bwd)

TOTAL params: 138M parameters

ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64	conv3-64	conv3-64	cc
<b>conv3-64</b>	conv3-64	conv3-64	cc
maxpool			
conv3-128	conv3-128	conv3-128	co
<b>conv3-128</b>	conv3-128	conv3-128	co
maxpool			
conv3-256	conv3-256	conv3-256	co
conv3-256	conv3-256	conv3-256	co
	<b>conv1-256</b>	<b>conv3-256</b>	co
			co
maxpool			
conv3-512	conv3-512	conv3-512	co
conv3-512	conv3-512	conv3-512	co
	<b>conv1-512</b>	<b>conv3-512</b>	co
			co
maxpool			
conv3-512	conv3-512	conv3-512	co
conv3-512	conv3-512	conv3-512	co
	<b>conv1-512</b>	<b>conv3-512</b>	co
			co
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

INPUT: [224x224x3] memory:  $224*224*3=150\text{K}$  params: 0 (not counting biases)

CONV3-64: [224x224x64] memory:  $224*224*64=3.2\text{M}$  params:  $(3*3*3)*64 = 1,728$

CONV3-64: [224x224x64] memory:  $224*224*64=3.2\text{M}$  params:  $(3*3*64)*64 = 36,864$

POOL2: [112x112x64] memory:  $112*112*64=800\text{K}$  params: 0

CONV3-128: [112x112x128] memory:  $112*112*128=1.6\text{M}$  params:  $(3*3*64)*128 = 73,728$

CONV3-128: [112x112x128] memory:  $112*112*128=1.6\text{M}$  params:  $(3*3*128)*128 = 147,456$

POOL2: [56x56x128] memory:  $56*56*128=400\text{K}$  params: 0

CONV3-256: [56x56x256] memory:  $56*56*256=800\text{K}$  params:  $(3*3*128)*256 = 294,912$

CONV3-256: [56x56x256] memory:  $56*56*256=800\text{K}$  params:  $(3*3*256)*256 = 589,824$

CONV3-256: [56x56x256] memory:  $56*56*256=800\text{K}$  params:  $(3*3*256)*256 = 589,824$

POOL2: [28x28x256] memory:  $28*28*256=200\text{K}$  params: 0

CONV3-512: [28x28x512] memory:  $28*28*512=400\text{K}$  params:  $(3*3*256)*512 = 1,179,648$

CONV3-512: [28x28x512] memory:  $28*28*512=400\text{K}$  params:  $(3*3*512)*512 = 2,359,296$

CONV3-512: [28x28x512] memory:  $28*28*512=400\text{K}$  params:  $(3*3*512)*512 = 2,359,296$

POOL2: [14x14x512] memory:  $14*14*512=100\text{K}$  params: 0

CONV3-512: [14x14x512] memory:  $14*14*512=100\text{K}$  params:  $(3*3*512)*512 = 2,359,296$

CONV3-512: [14x14x512] memory:  $14*14*512=100\text{K}$  params:  $(3*3*512)*512 = 2,359,296$

CONV3-512: [14x14x512] memory:  $14*14*512=100\text{K}$  params:  $(3*3*512)*512 = 2,359,296$

POOL2: [7x7x512] memory:  $7*7*512=25\text{K}$  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:  $24\text{M} * 4 \text{ bytes} \approx 93\text{MB} / \text{image}$  (only forward!  $\sim 2$  for bwd)

TOTAL params: 138M parameters

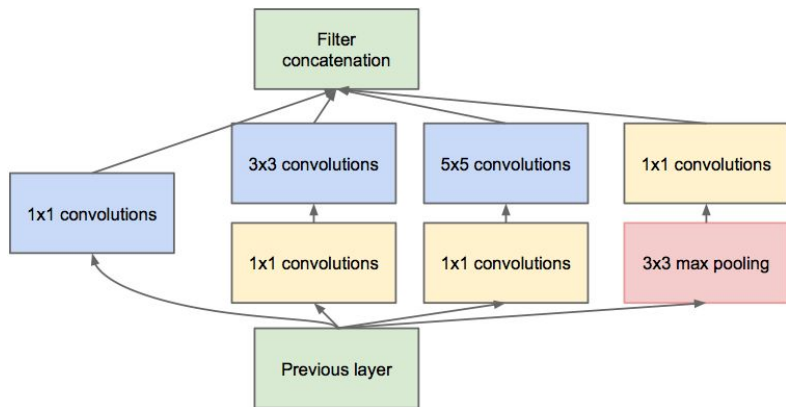
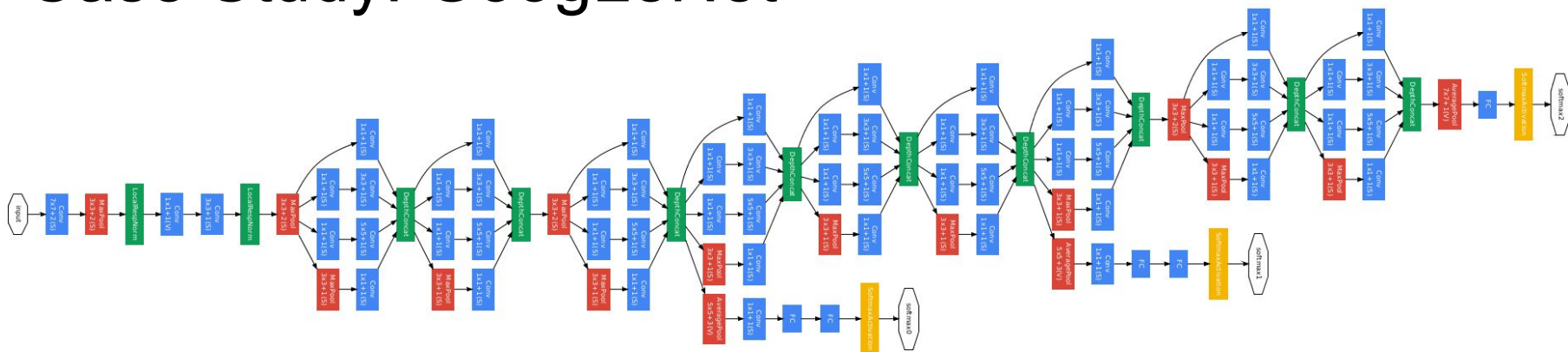
Note:

Most memory is in  
early CONV

Most params are  
in late FC

# Case Study: GoogLeNet

[Szegedy et al., 2014]



## Inception module

ILSVRC 2014 winner (6.7% top 5 error)

# Case Study: GoogLeNet

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Fun features:


- Only 5 million params!  
(Removes FC layers completely)

**Compared to AlexNet:**

- 12X less params
- 2x more compute
- 6.67% (vs. 16.4%)


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

ILSVRC 2015 winner (3.6% top 5 error)

Microsoft  
Research

## MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: “*Ultra-deep*” (quote Yann) **152-layer** nets
  - ImageNet Detection: **16%** better than 2nd
  - ImageNet Localization: **27%** better than 2nd
  - COCO Detection: **11%** better than 2nd
  - COCO Segmentation: **12%** better than 2nd

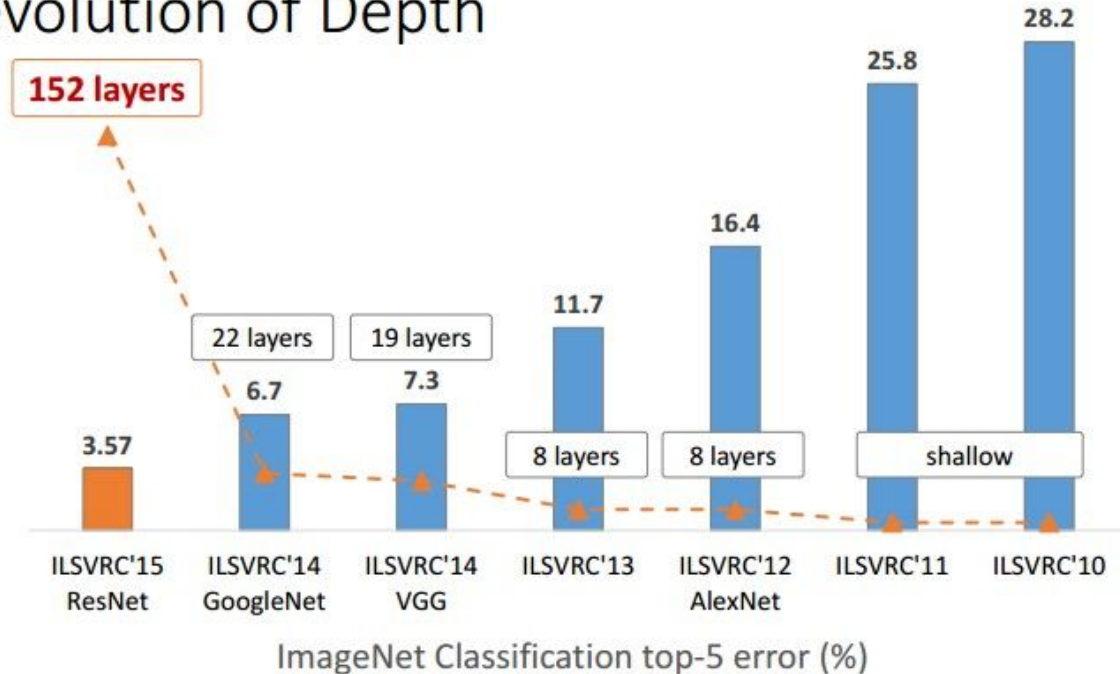
ICCV 15  
International Conference on Computer Vision

\*improvements are relative numbers

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. arXiv 2015.

Slide from Kaiming He’s recent presentation <https://www.youtube.com/watch?v=1PGLj-uKT1w>

# Revolution of Depth



ImageNet Classification top-5 error (%)

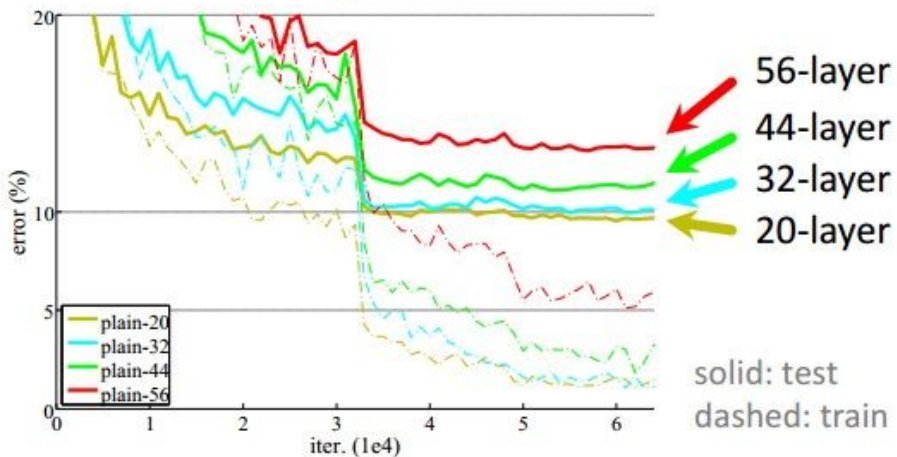
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

(slide from Kaiming He's recent presentation)

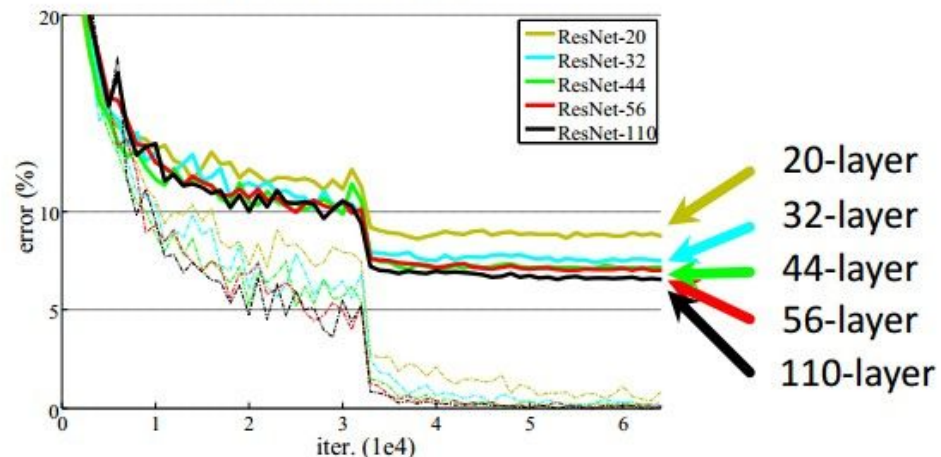


# CIFAR-10 experiments

CIFAR-10 plain nets

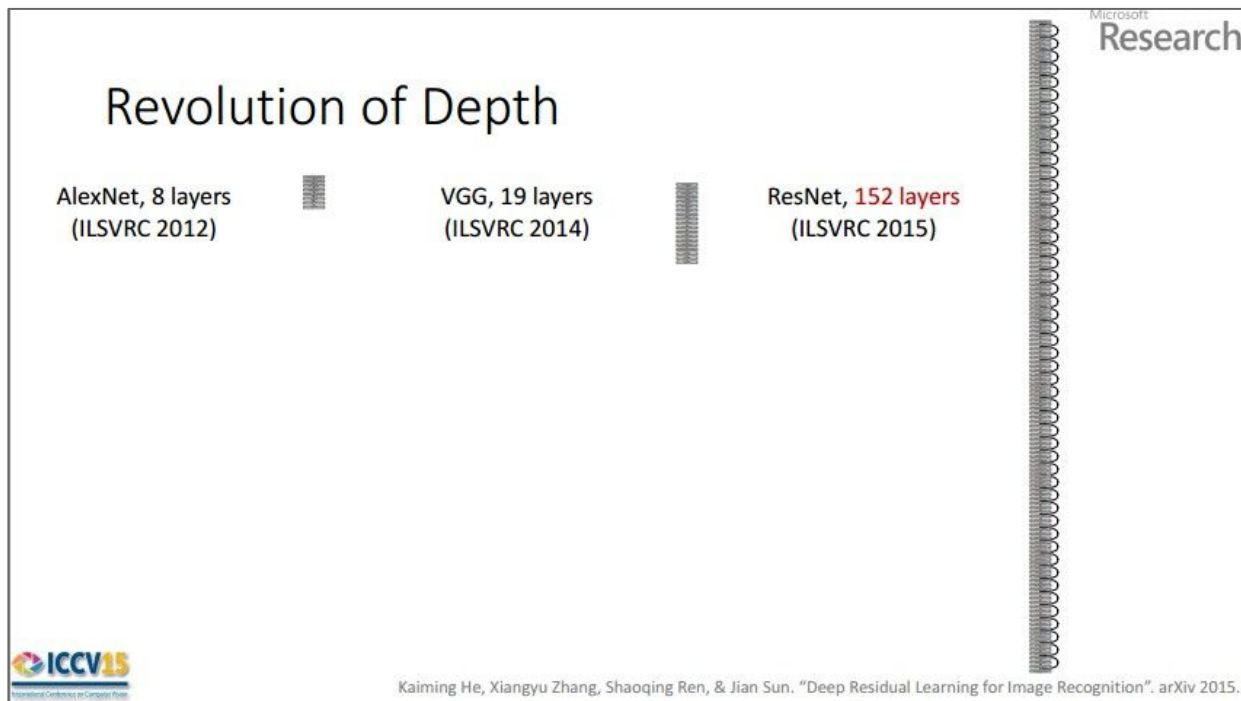


CIFAR-10 ResNets



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

ILSVRC 2015 winner (3.6% top 5 error)



2-3 weeks of training  
on 8 GPU machine

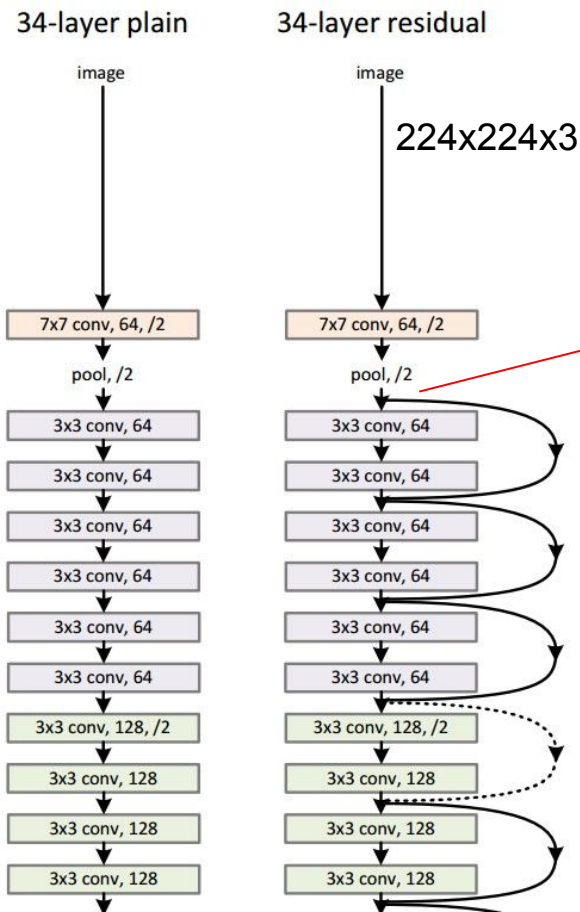
at runtime: faster  
than a VGGNet!  
(even though it has  
8x more layers)

(slide from Kaiming He's recent presentation)



# Case Study: ResNet

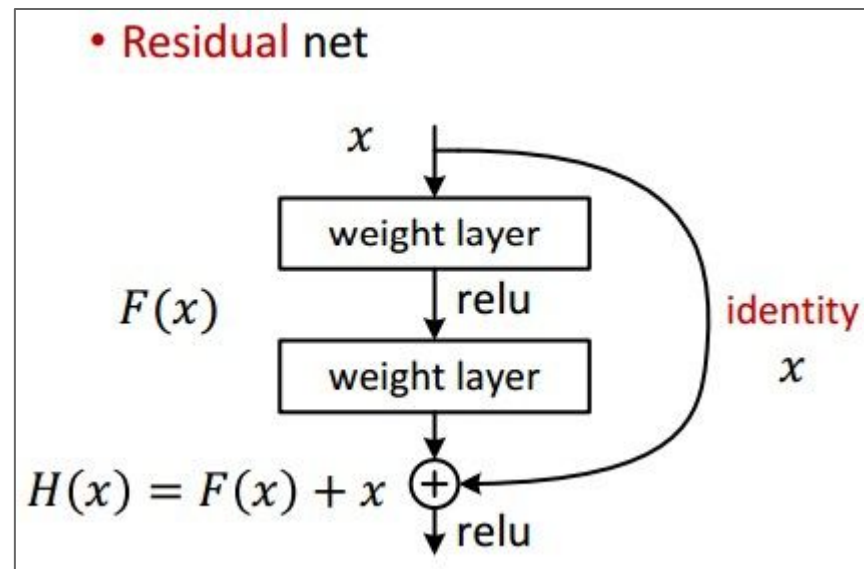
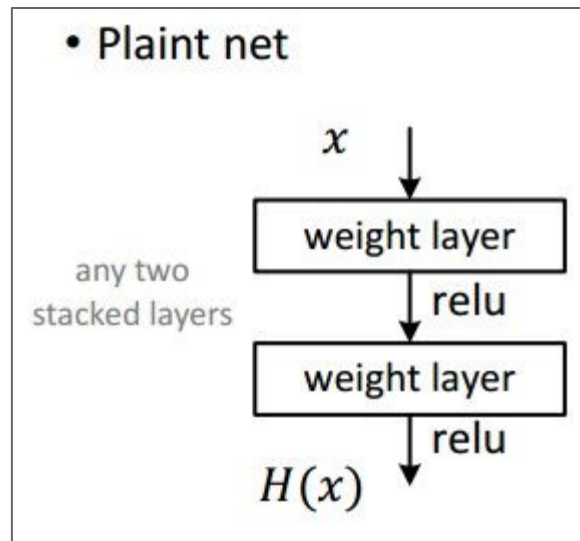
[He et al., 2015]



spatial dimension  
only 56x56!

# Case Study: ResNet

[He et al., 2015]

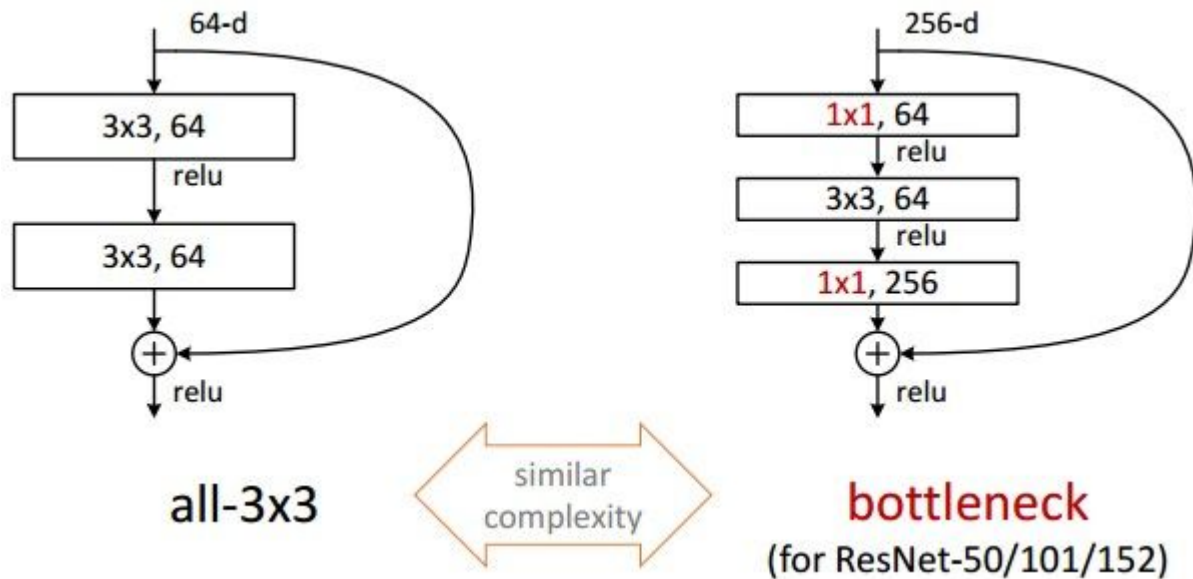


# Case Study: ResNet *[He et al., 2015]*

- 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

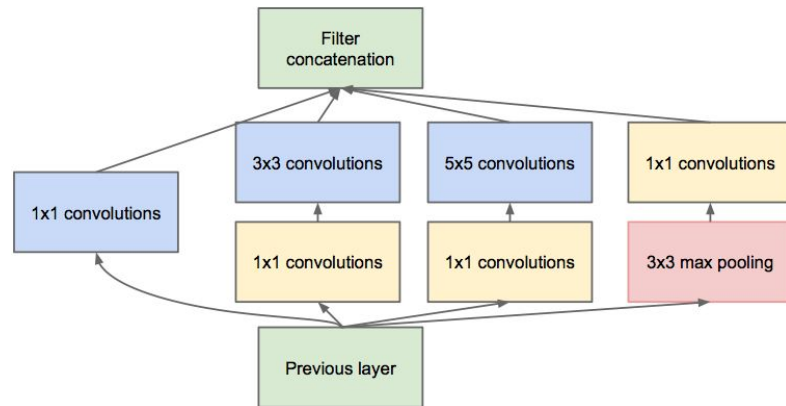
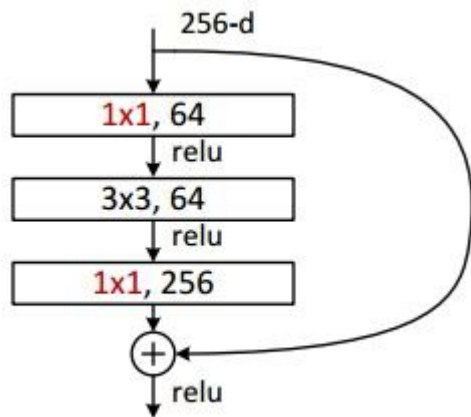
# Case Study: ResNet

[He et al., 2015]



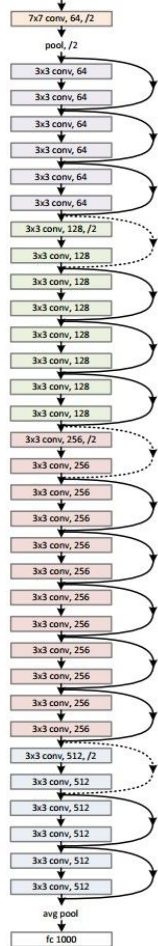
# Case Study: ResNet

[He et al., 2015]



(this trick is also used in GoogLeNet)

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



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				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		$1.8 \times 10^9$	$3.6 \times 10^9$	$3.8 \times 10^9$	$7.6 \times 10^9$	$11.3 \times 10^9$

# 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  
 **$[(\text{CONV-RELU})^*N\text{-POOL?}]^*M\text{-(FC-RELU)}^*K, \text{SOFTMAX}$**   
where N is usually up to ~5, M is large,  $0 \leq K \leq 2$ .
  - but recent advances such as ResNet/GoogLeNet challenge this paradigm