

CMP717

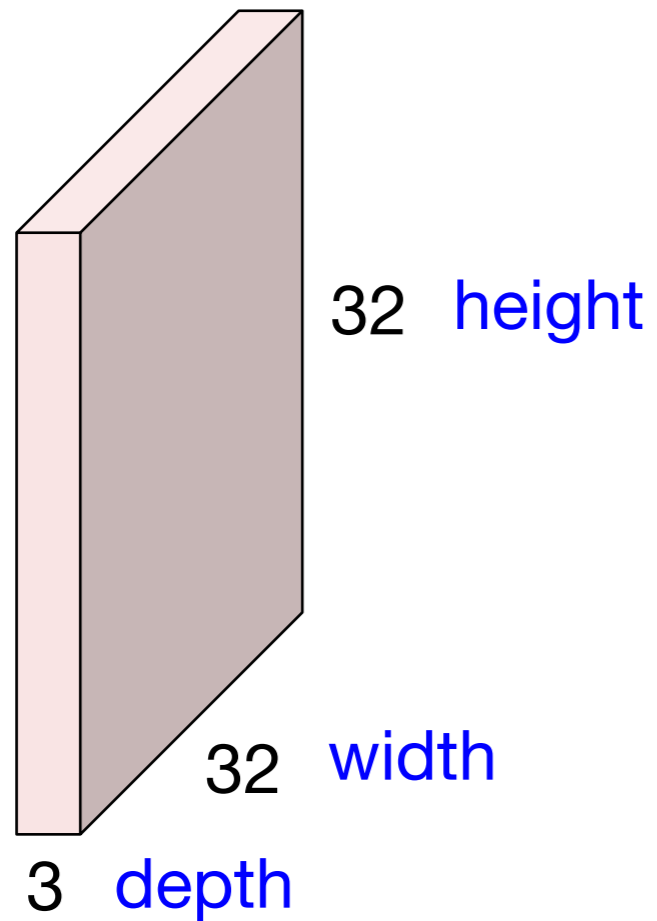
Image Processing

Convolutional Neural Networks

Erkut Erdem
Hacettepe University
Computer Vision Lab (HUCVL)

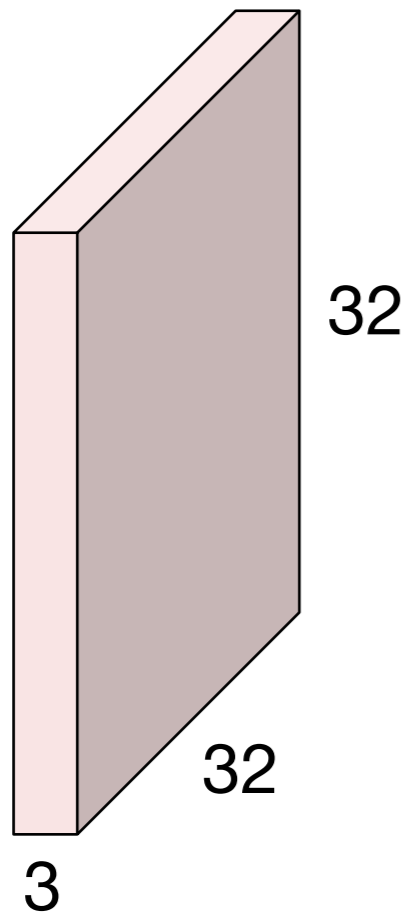
Convolution Layer

32x32x3 image



Convolution Layer

32x32x3 image



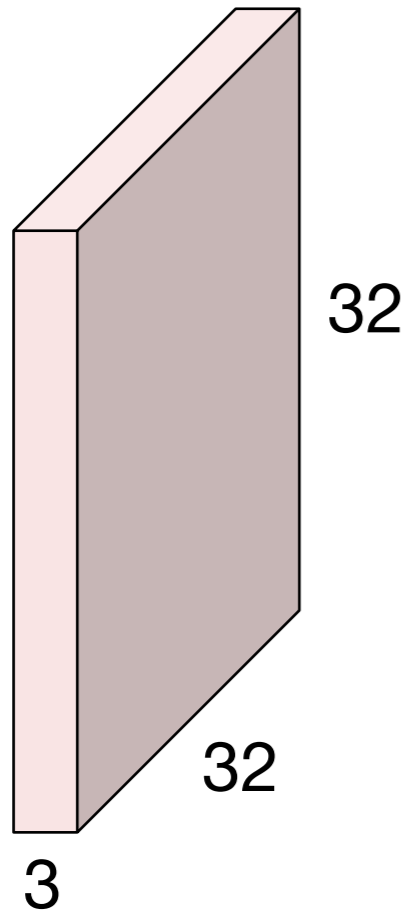
5x5x3 filter



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

Convolution Layer

32x32x3 image



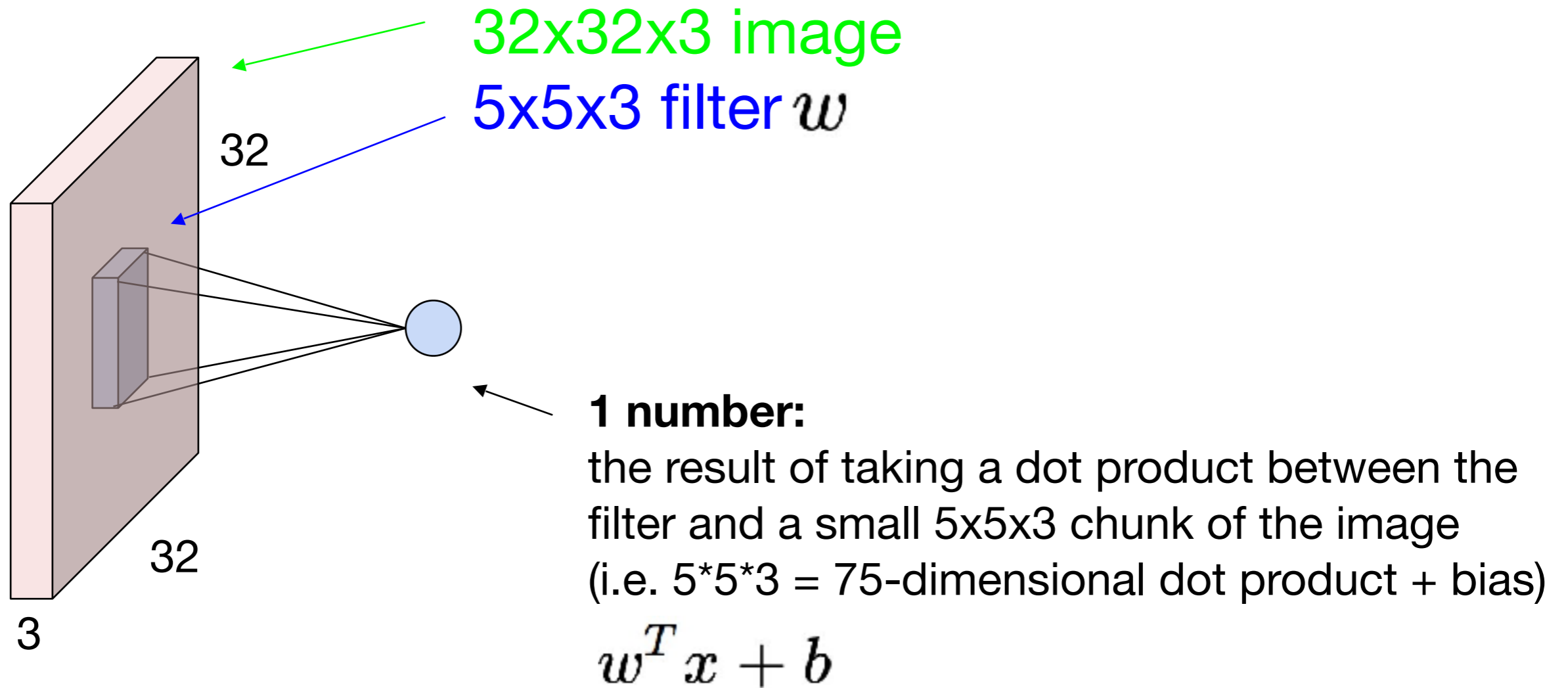
Filters always extend the full depth of the input volume

5x5x3 filter

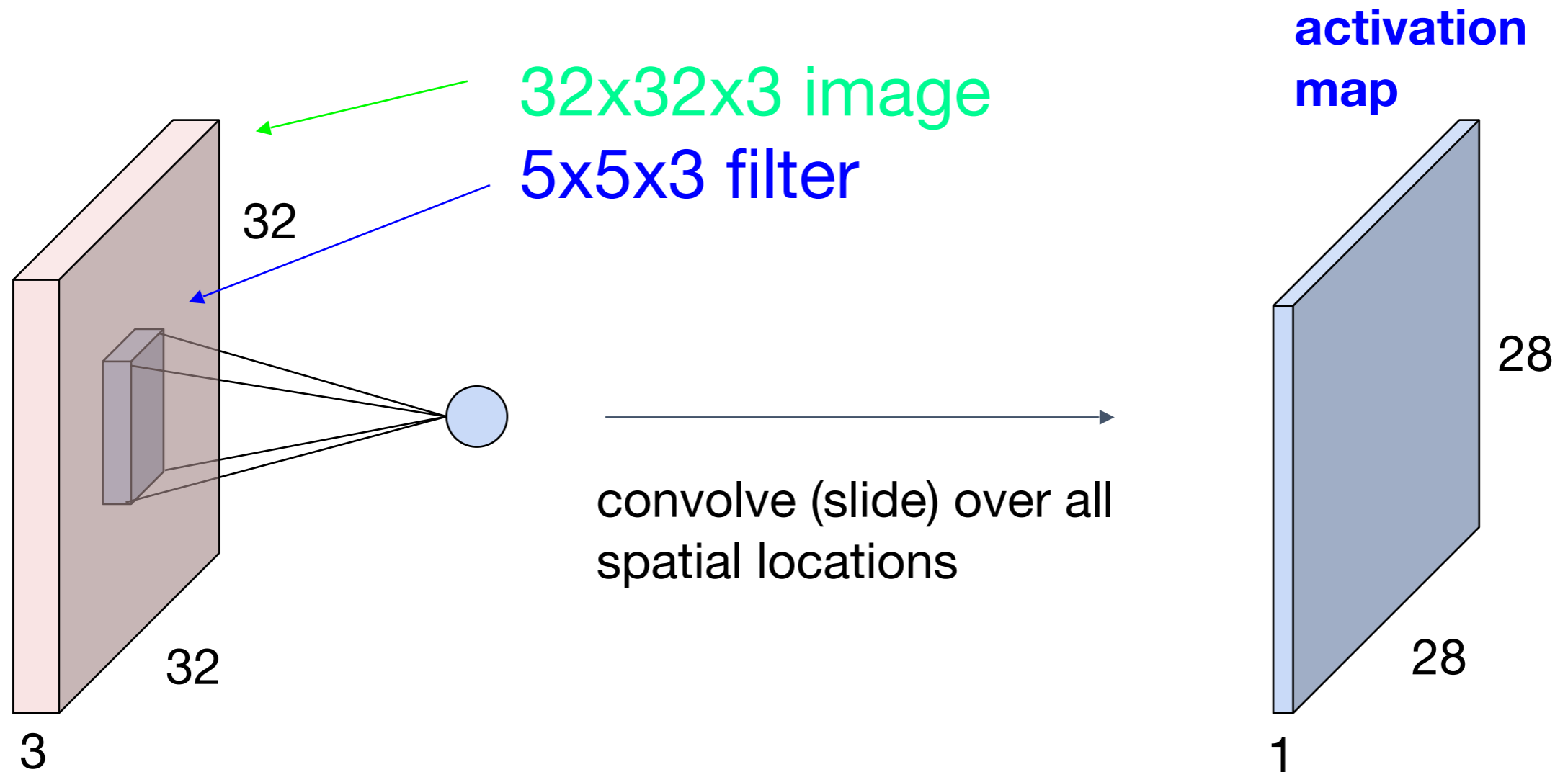


Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

Convolution Layer

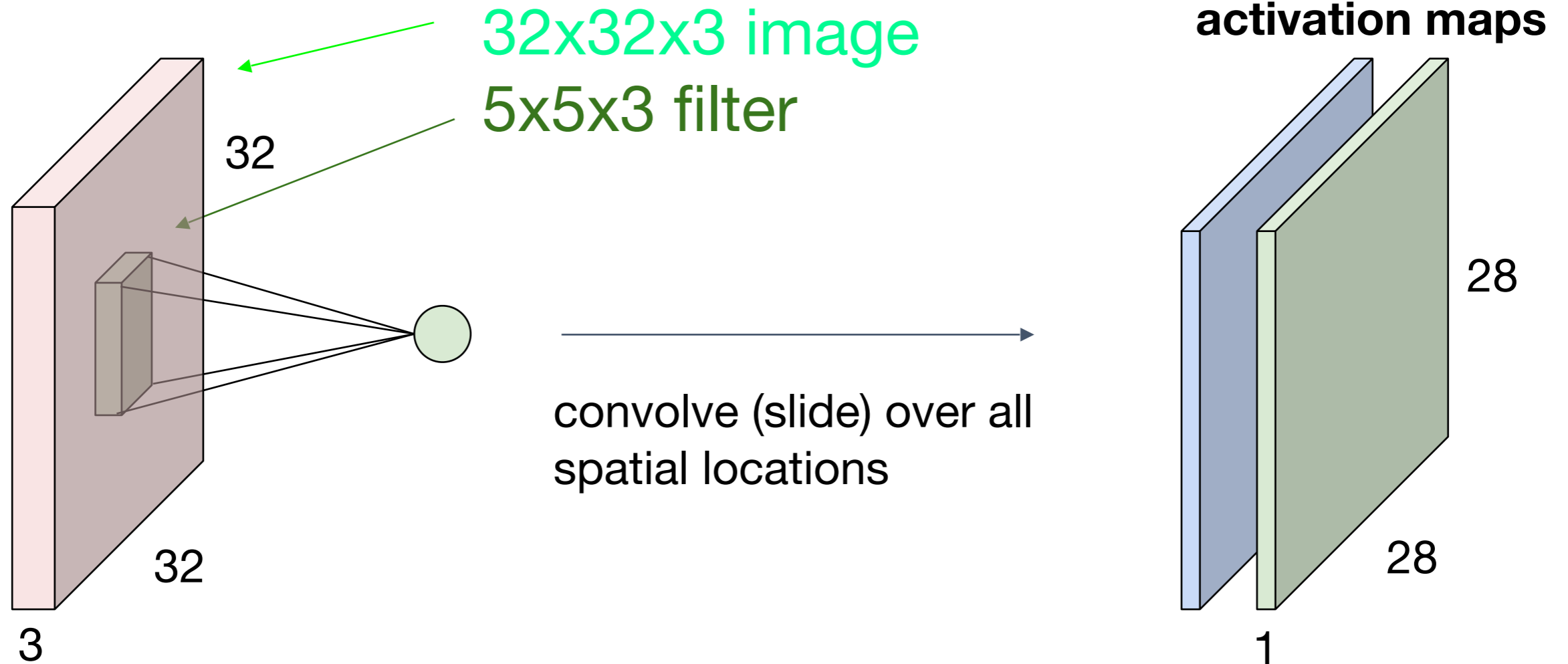


Convolution Layer

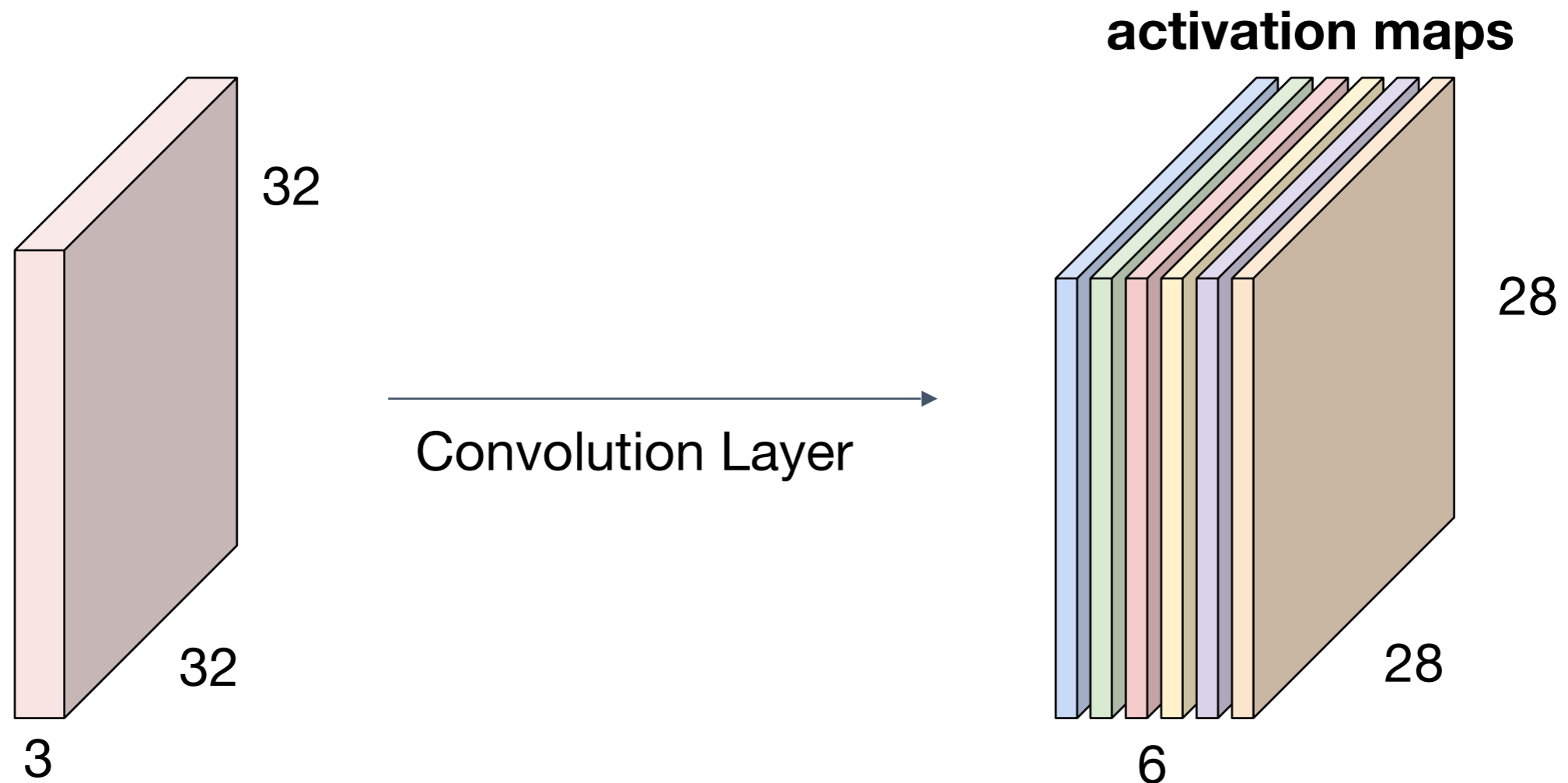


Convolution Layer

consider a second, **green** filter

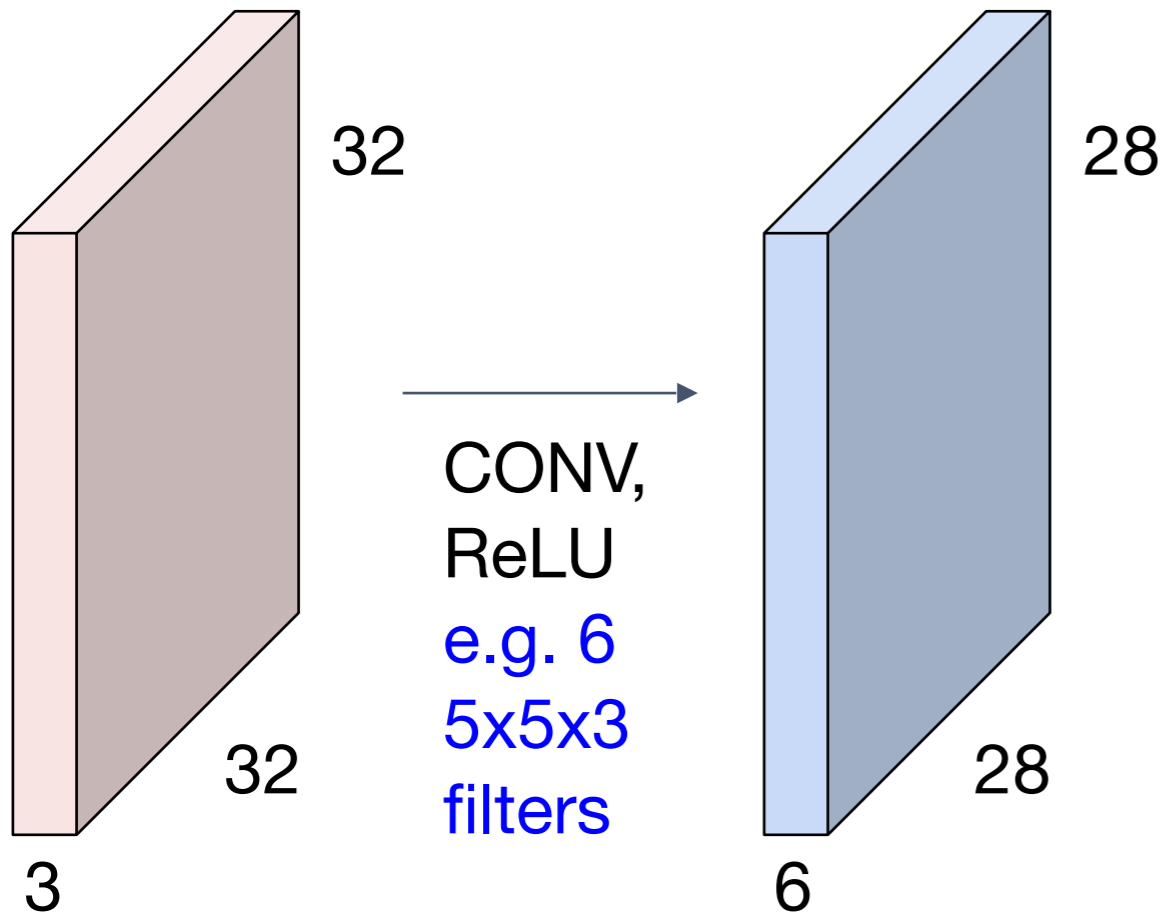


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

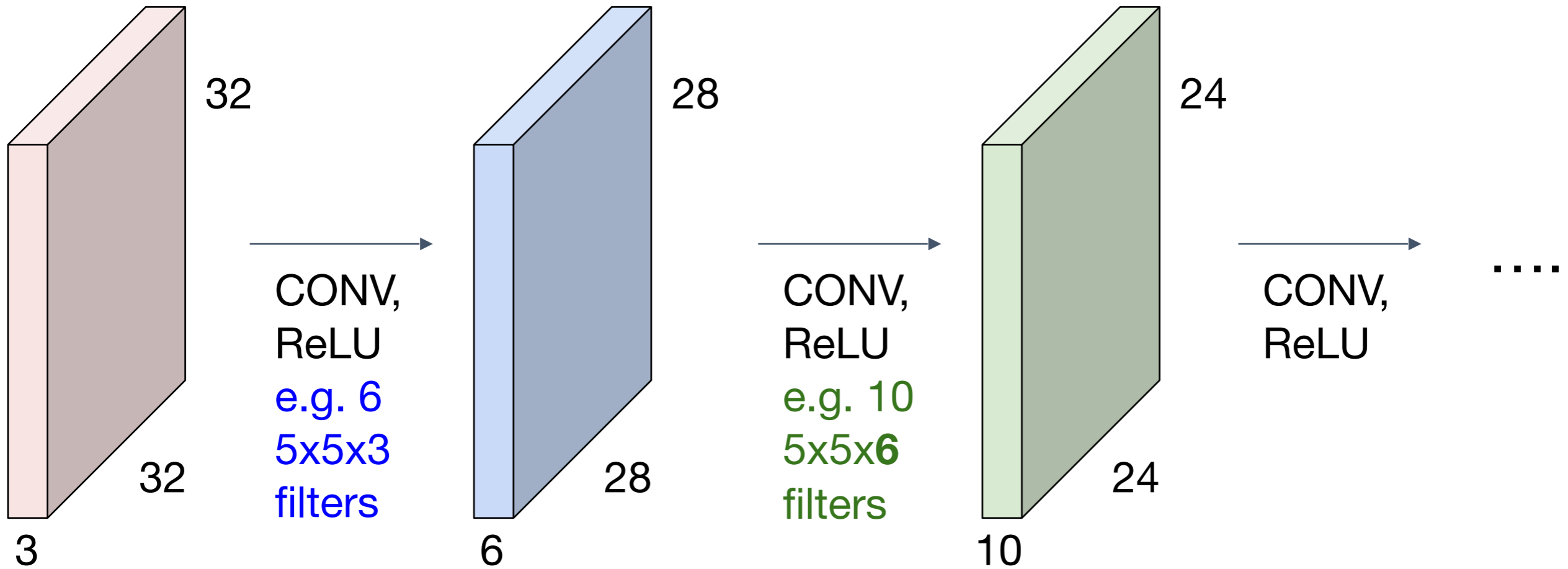


We stack these up to get a “new image” of size 28x28x6!

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

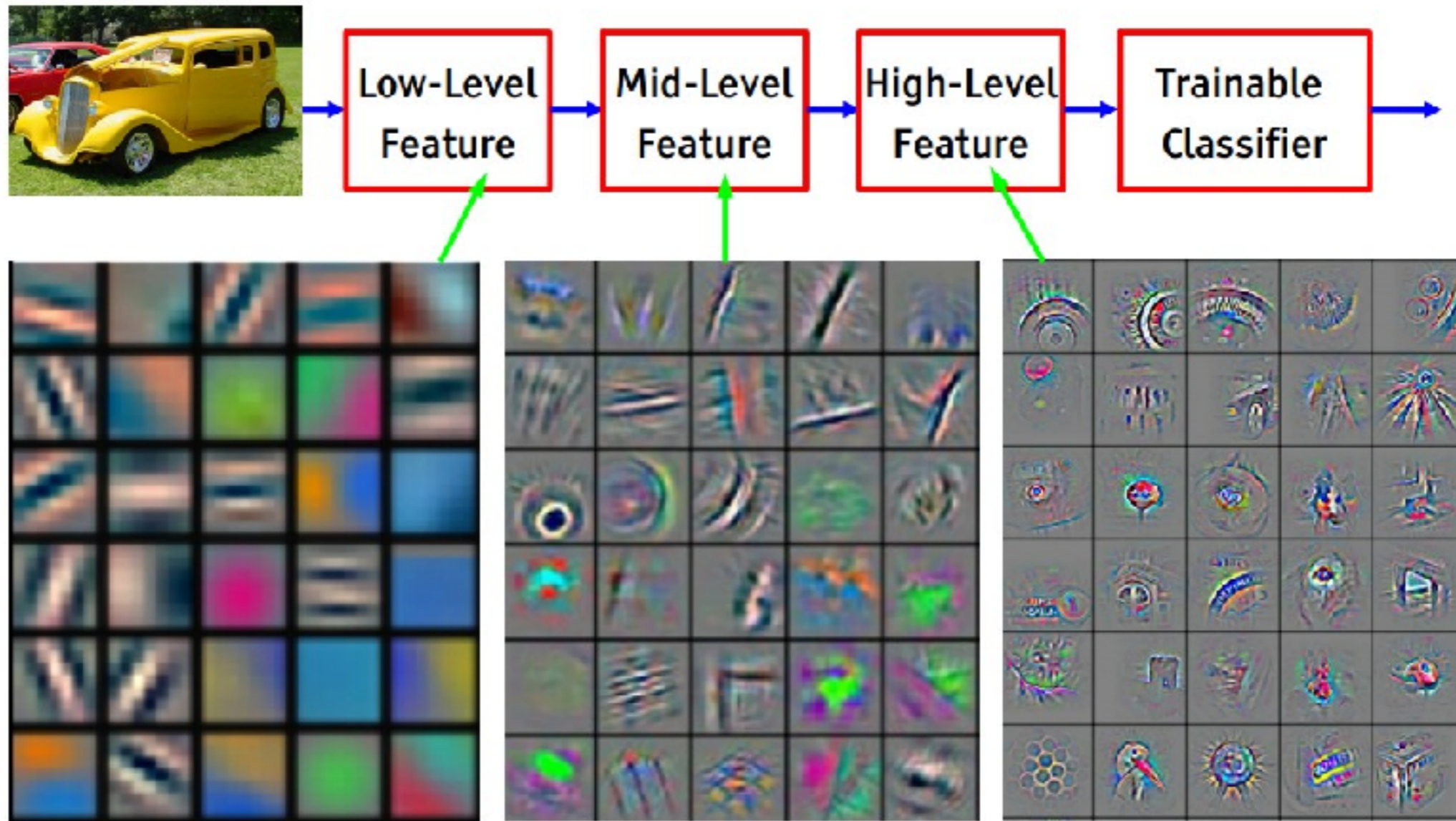


Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Preview

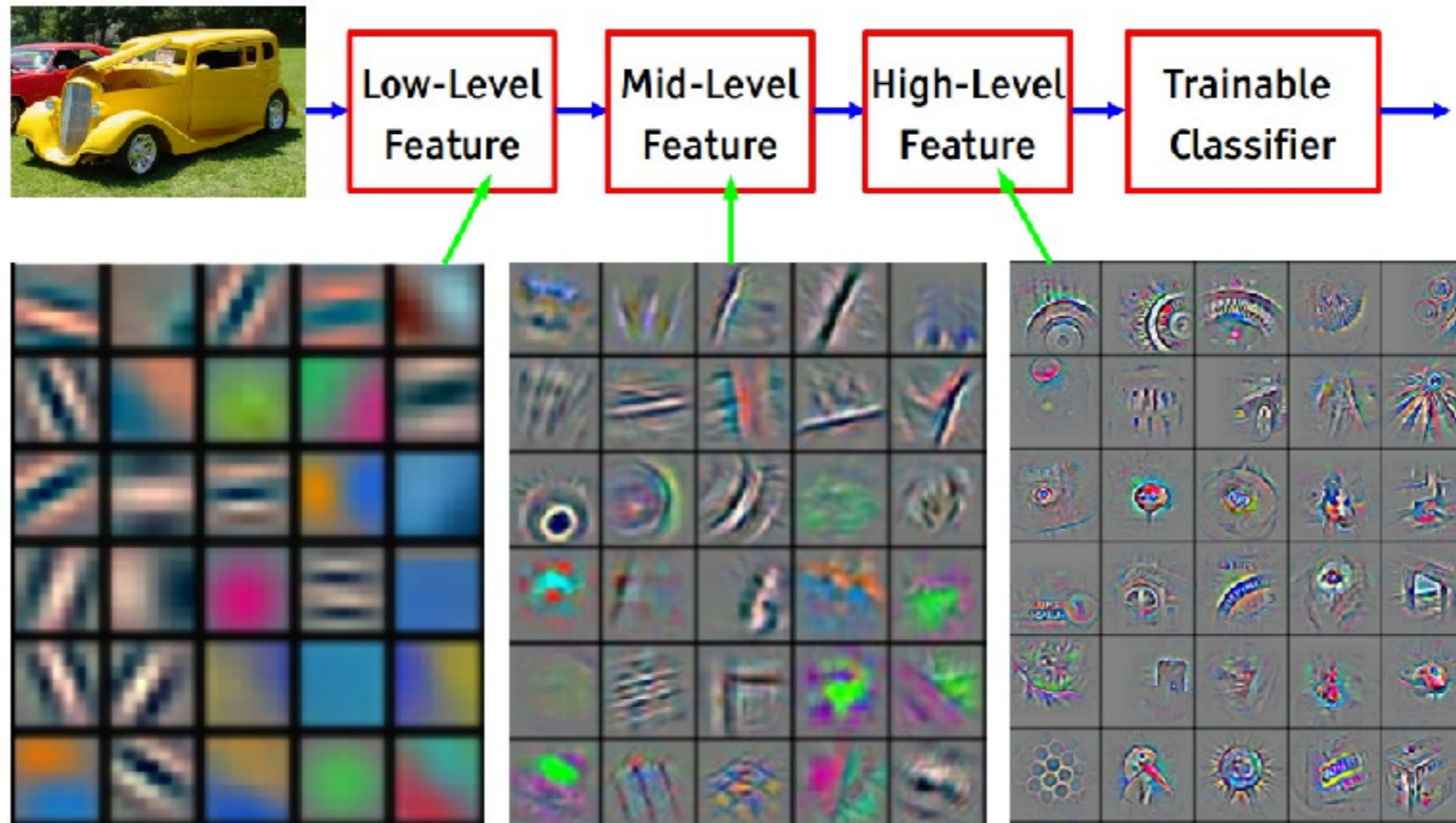
[From recent Yann
LeCun slides]



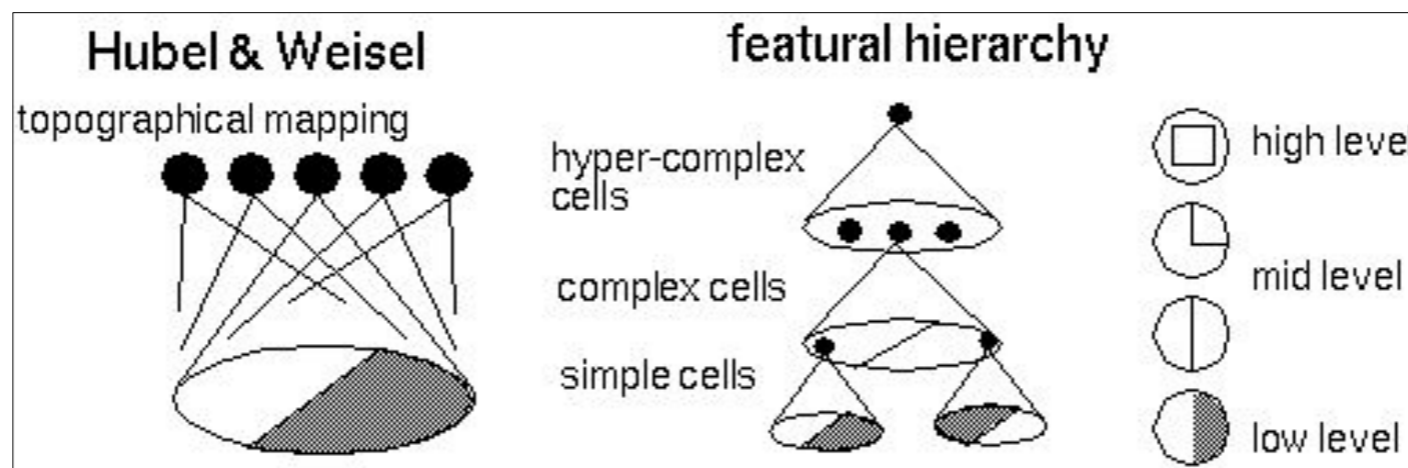
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Preview

[From recent Yann LeCun slides]



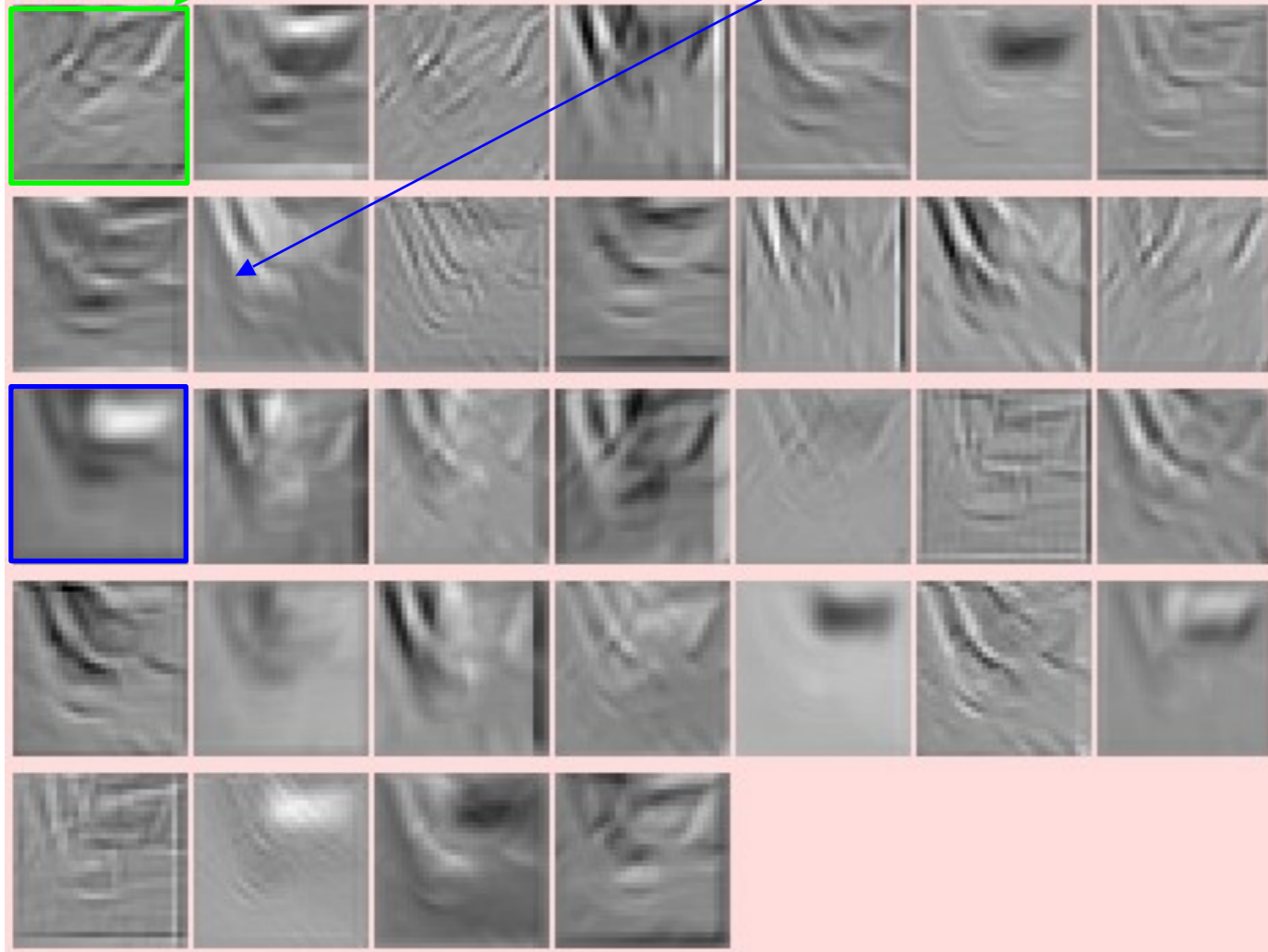
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]





one filter =>
one activation map

Activations:



example 5x5 filters

(32 total)

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

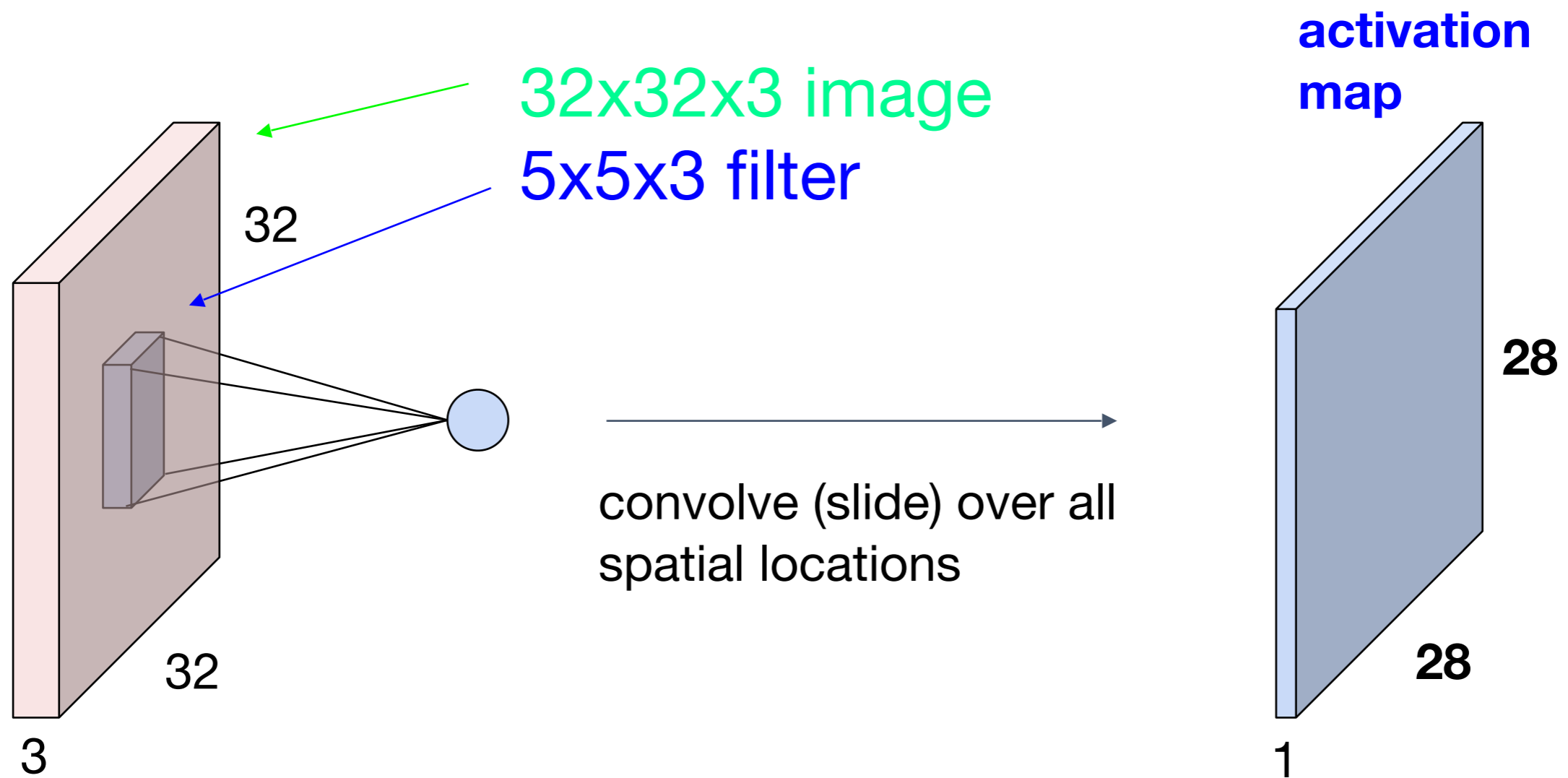
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1,y-n_2]$$

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

Preview

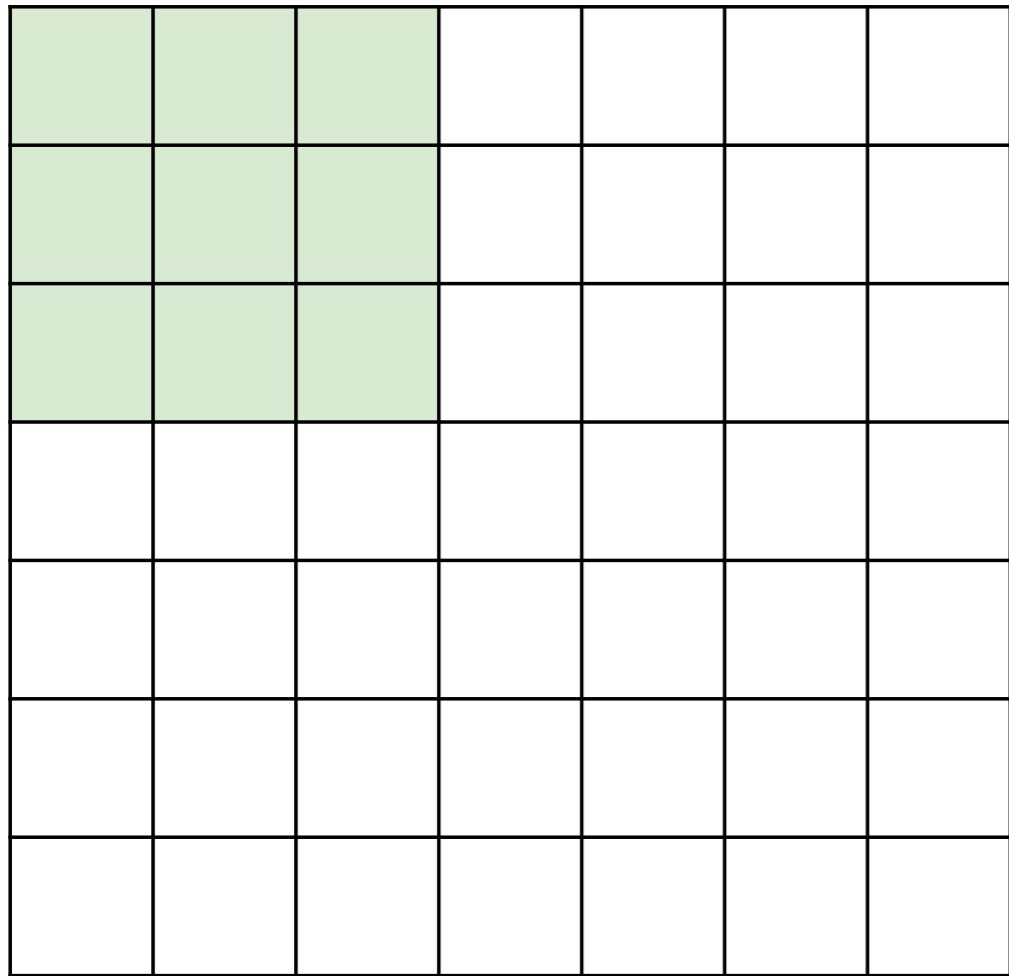


A closer look at spatial dimensions:



A closer look at spatial dimensions:

7

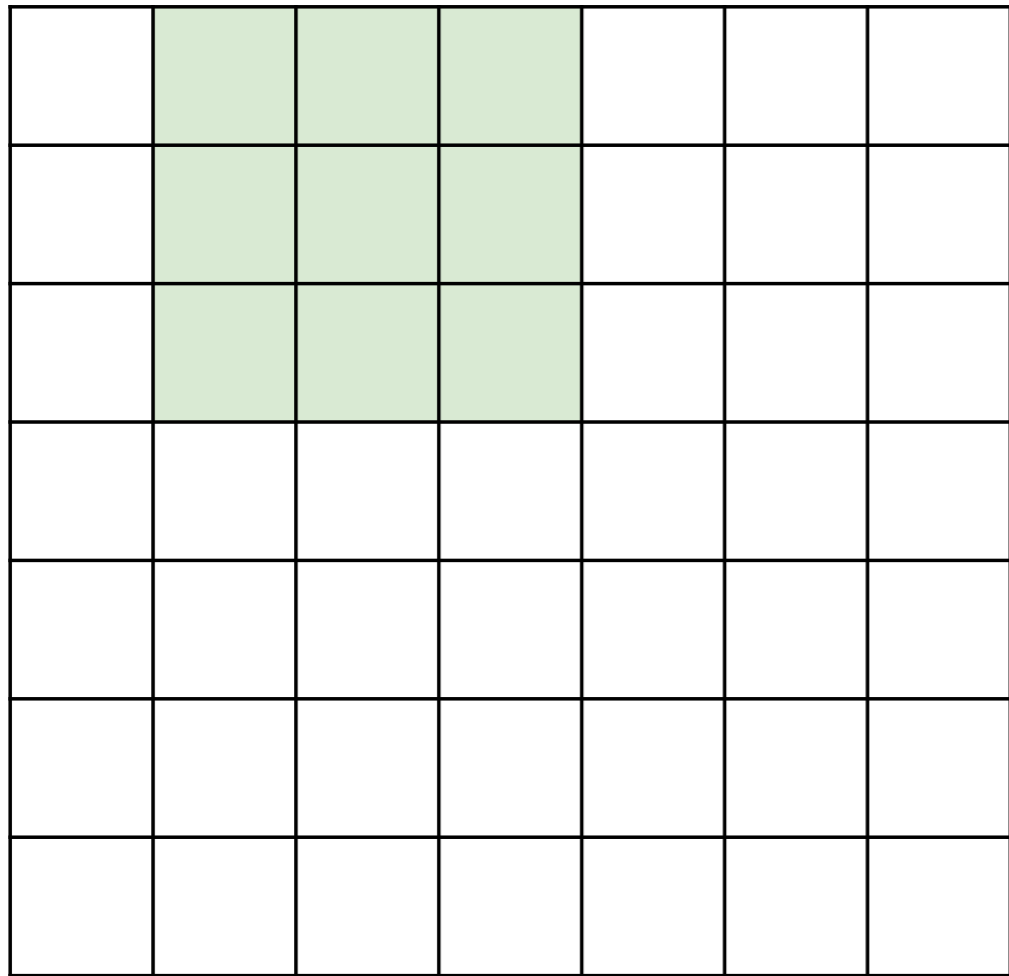


7

7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

7

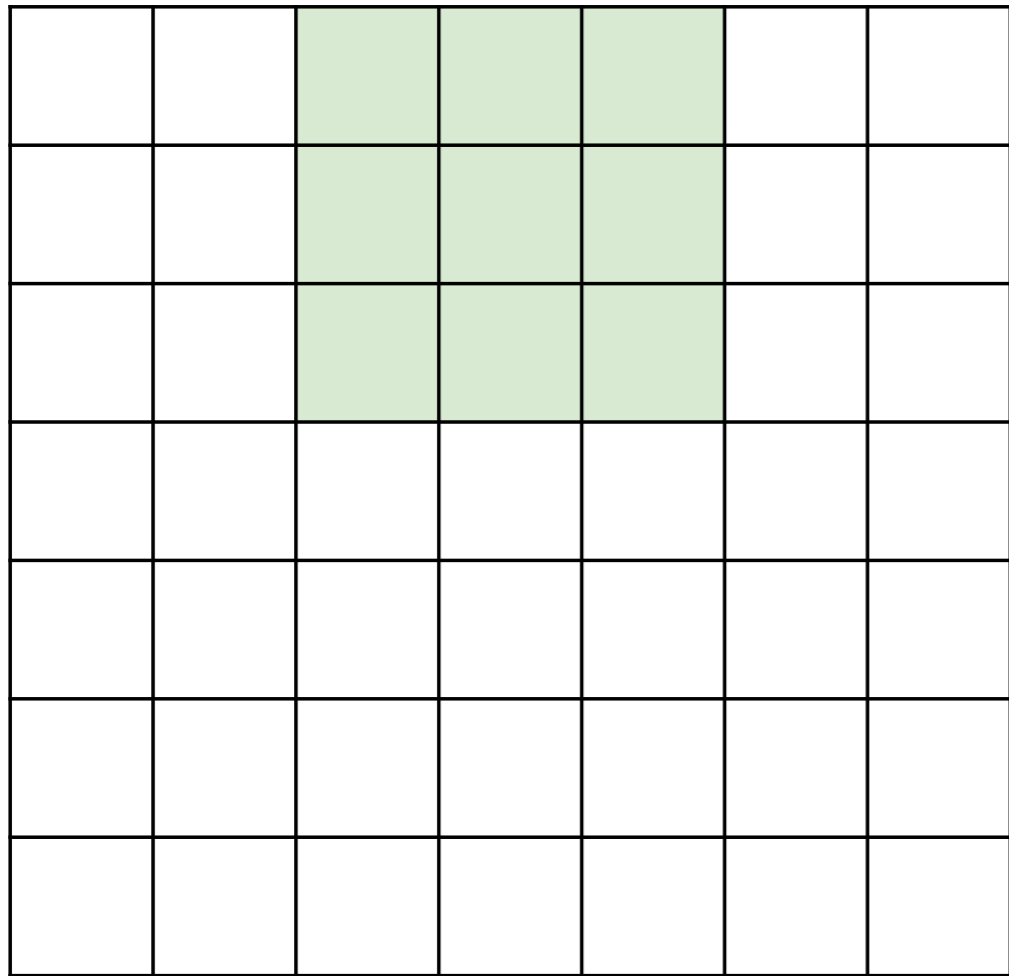


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

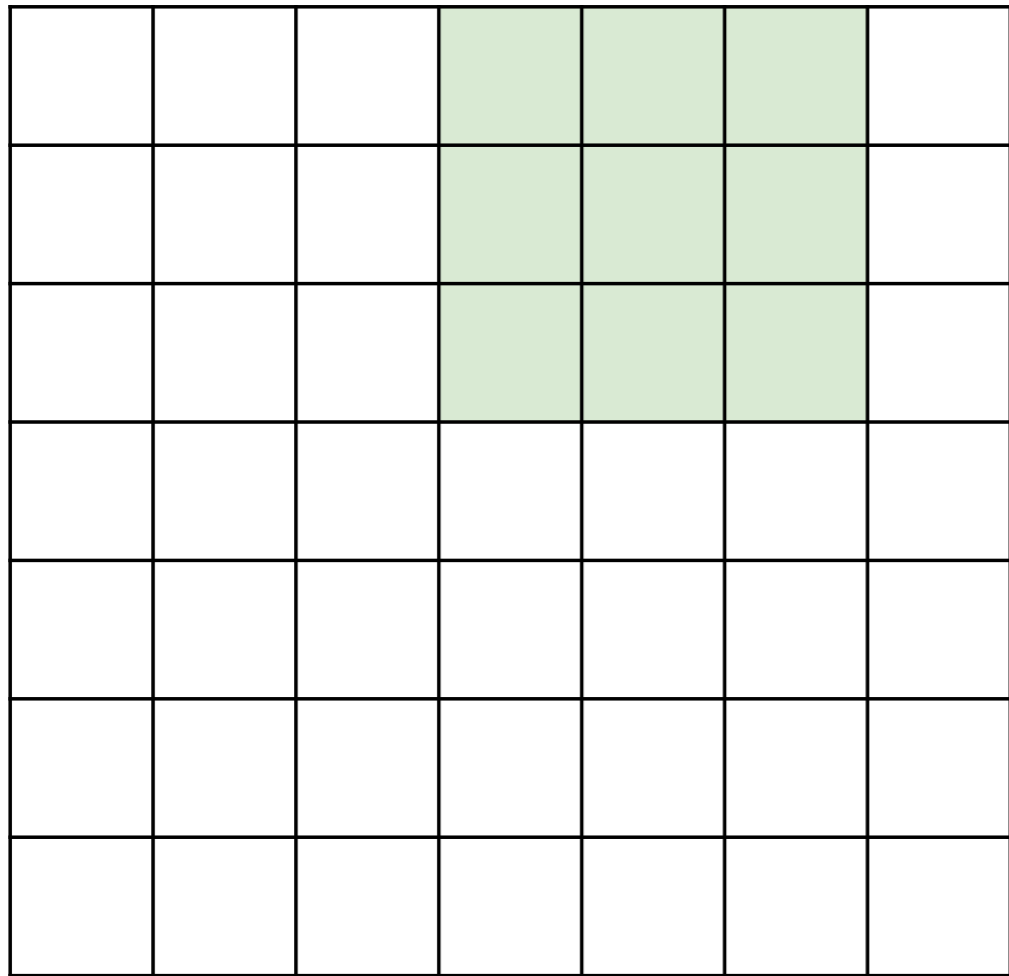


7

7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

7

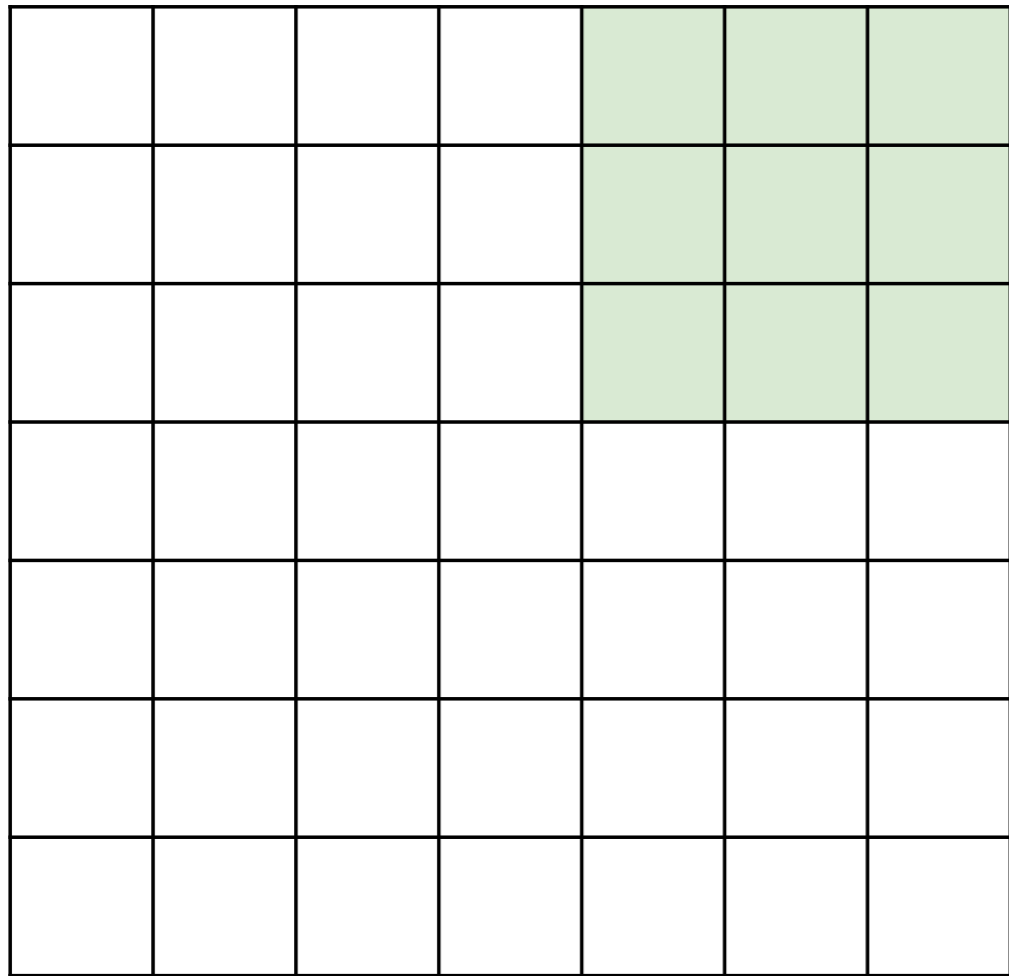


7

7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

7

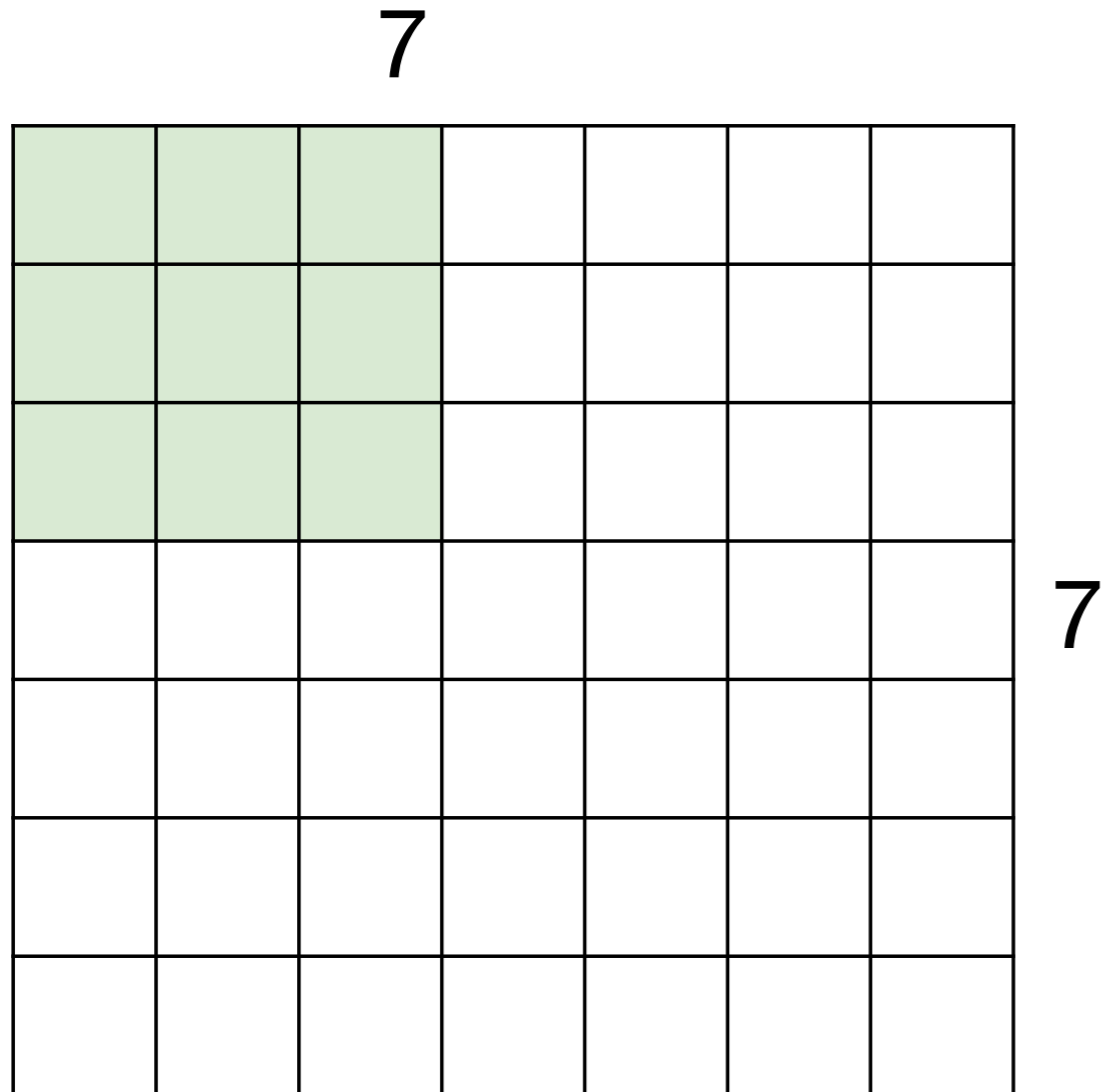


7

7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

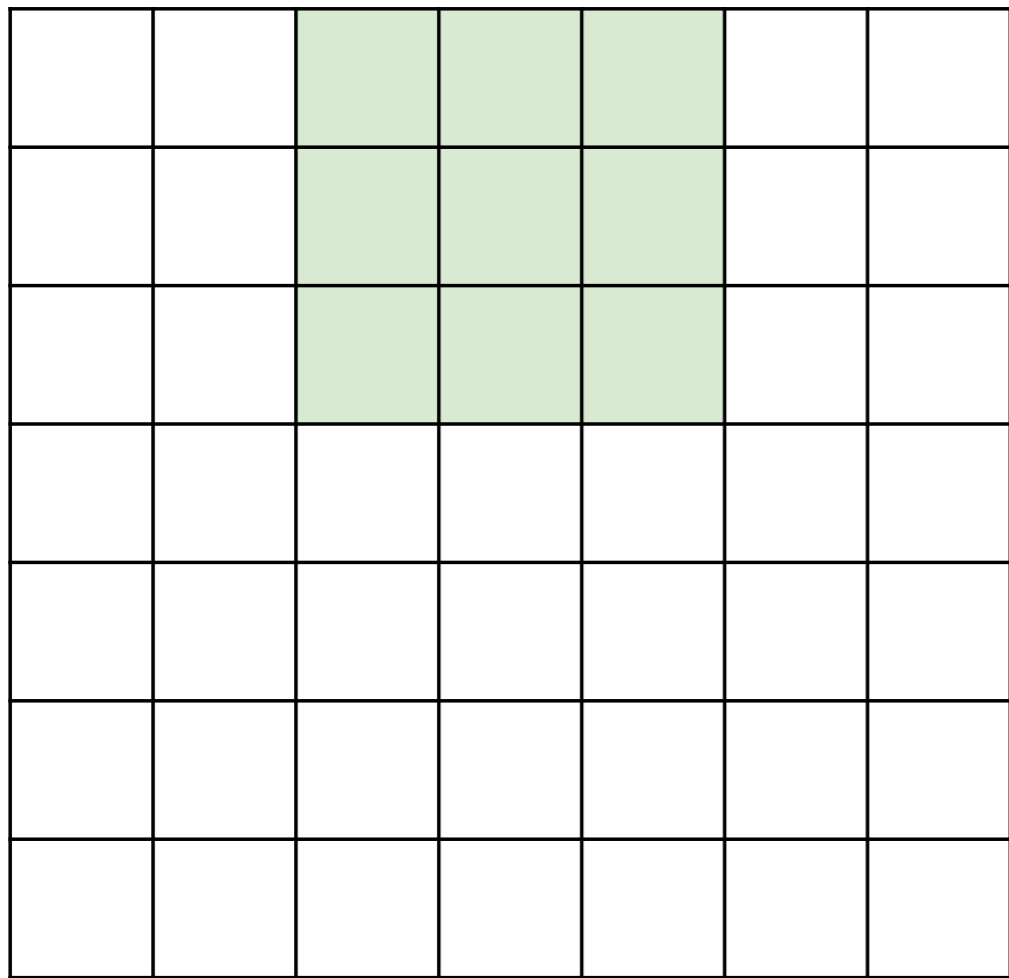
A closer look at spatial dimensions:



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

A closer look at spatial dimensions:

7

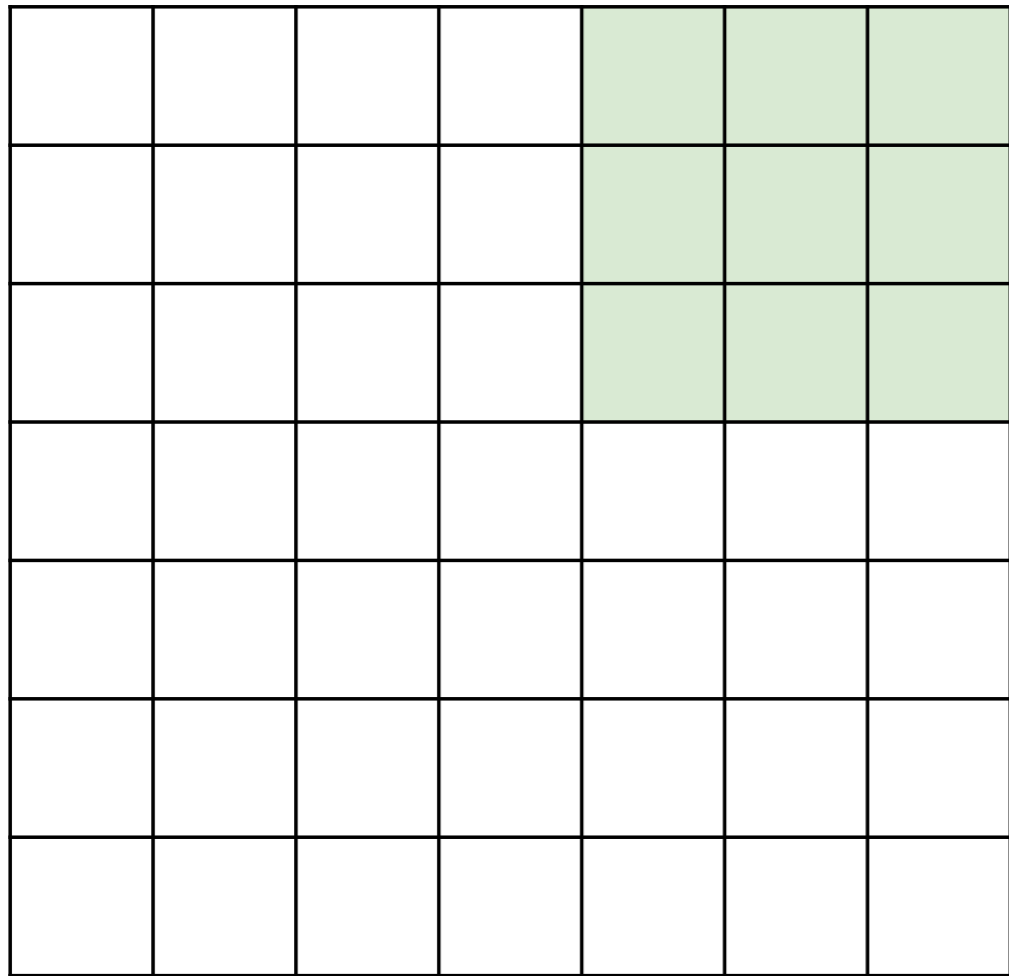


7

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

A closer look at spatial dimensions:

7

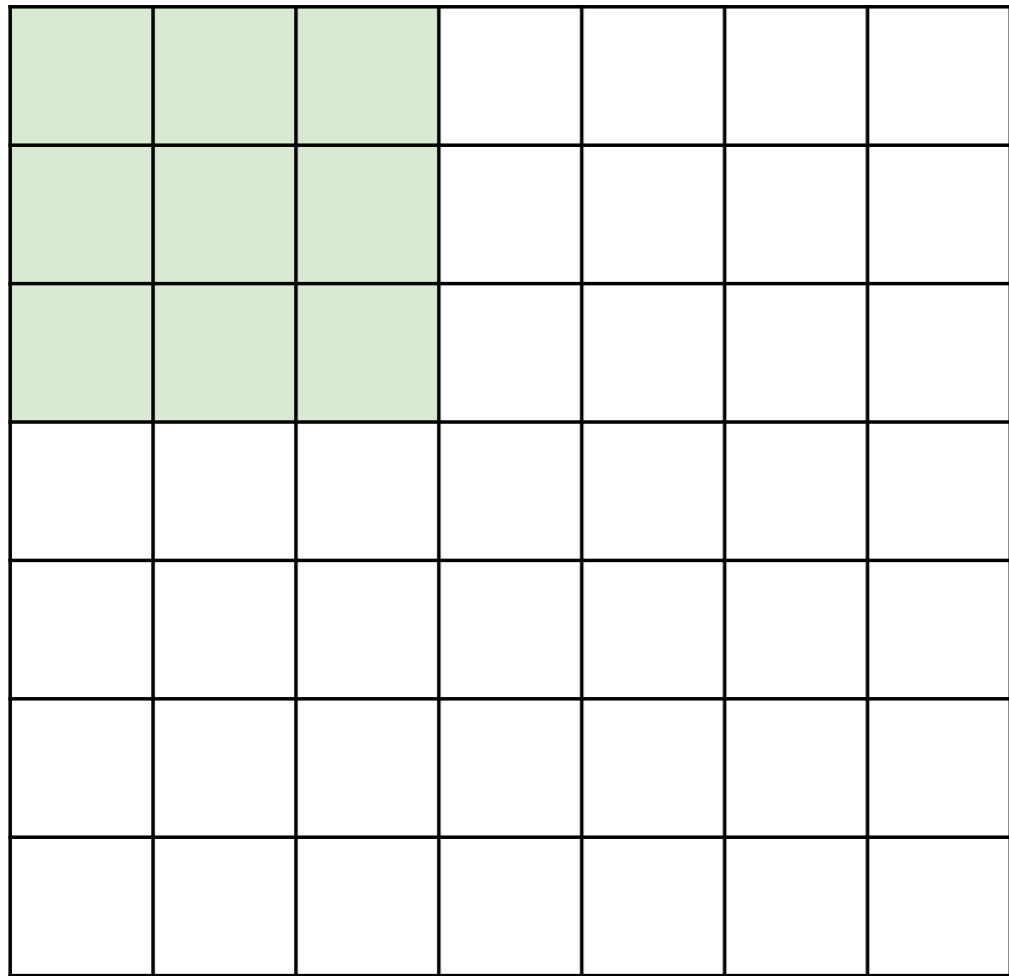


7

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

A closer look at spatial dimensions:

7

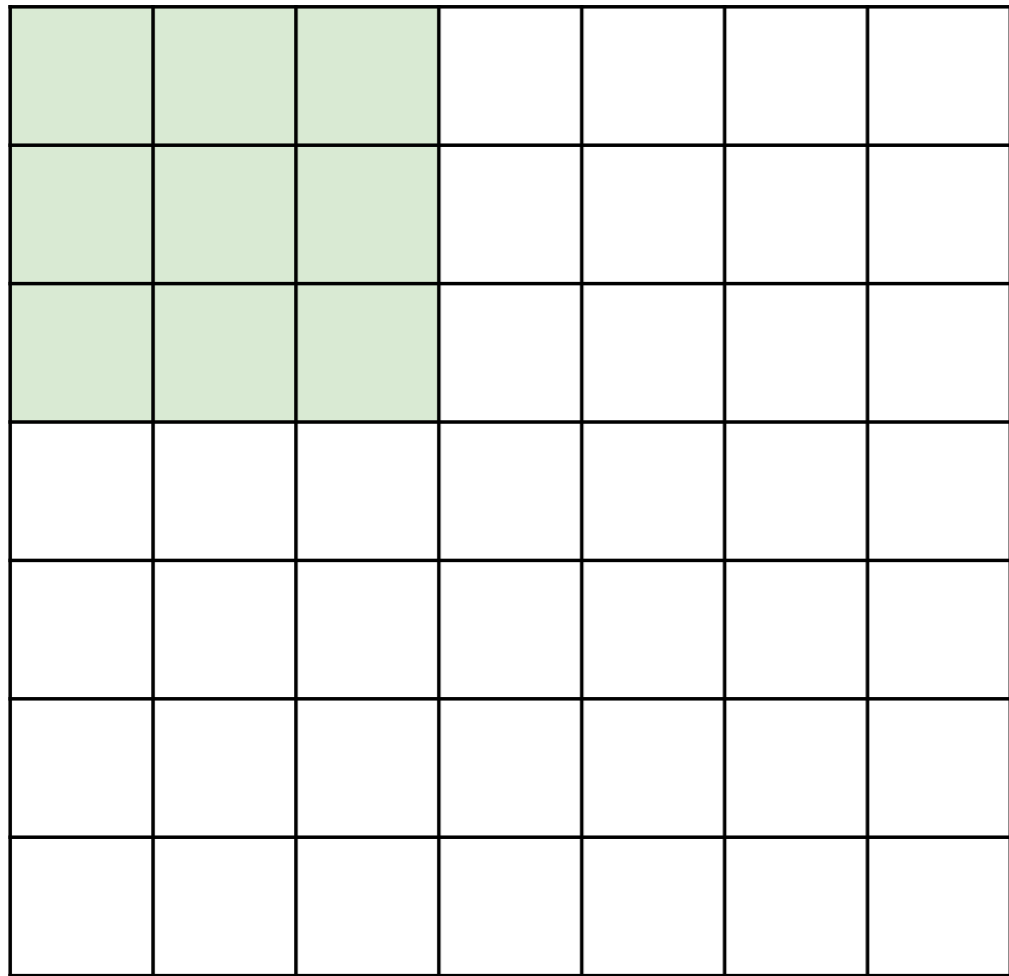


7

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

A closer look at spatial dimensions:

7

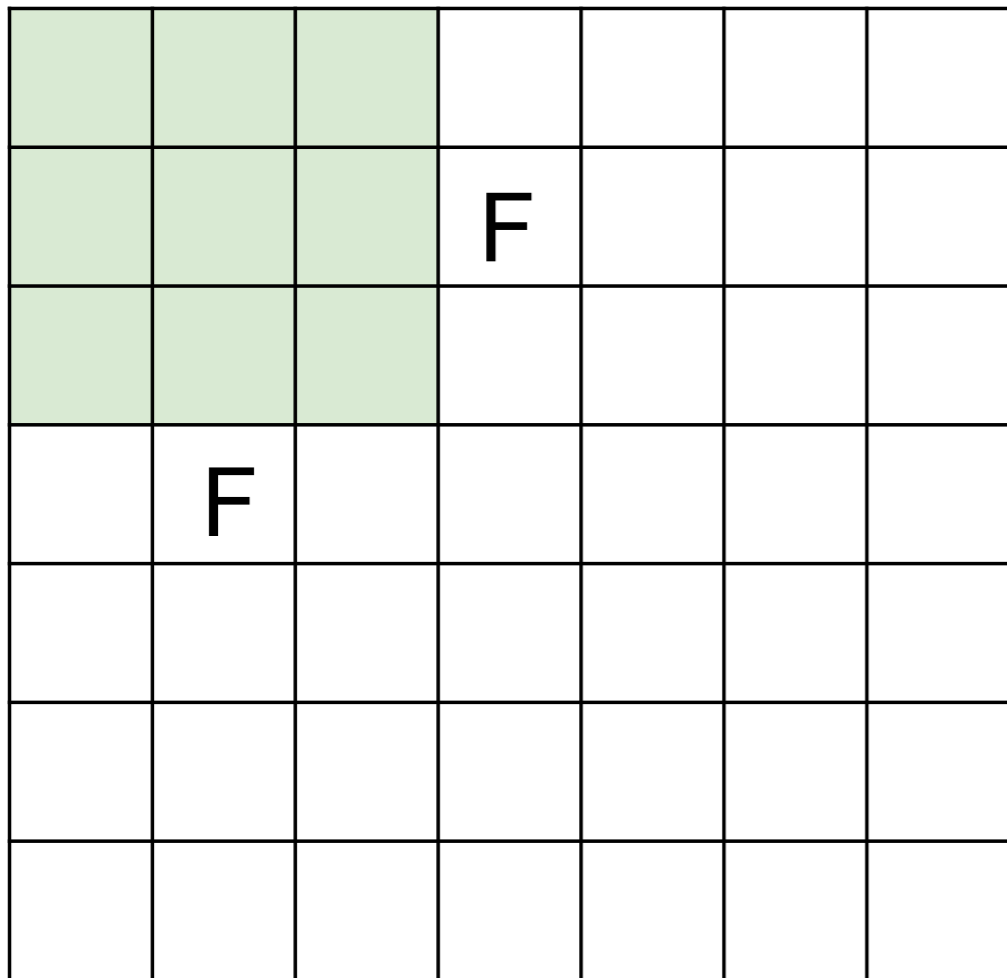


7

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

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

N



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 $\Rightarrow (7 - 3) / 3 + 1 = 2.33 :$

\

N

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

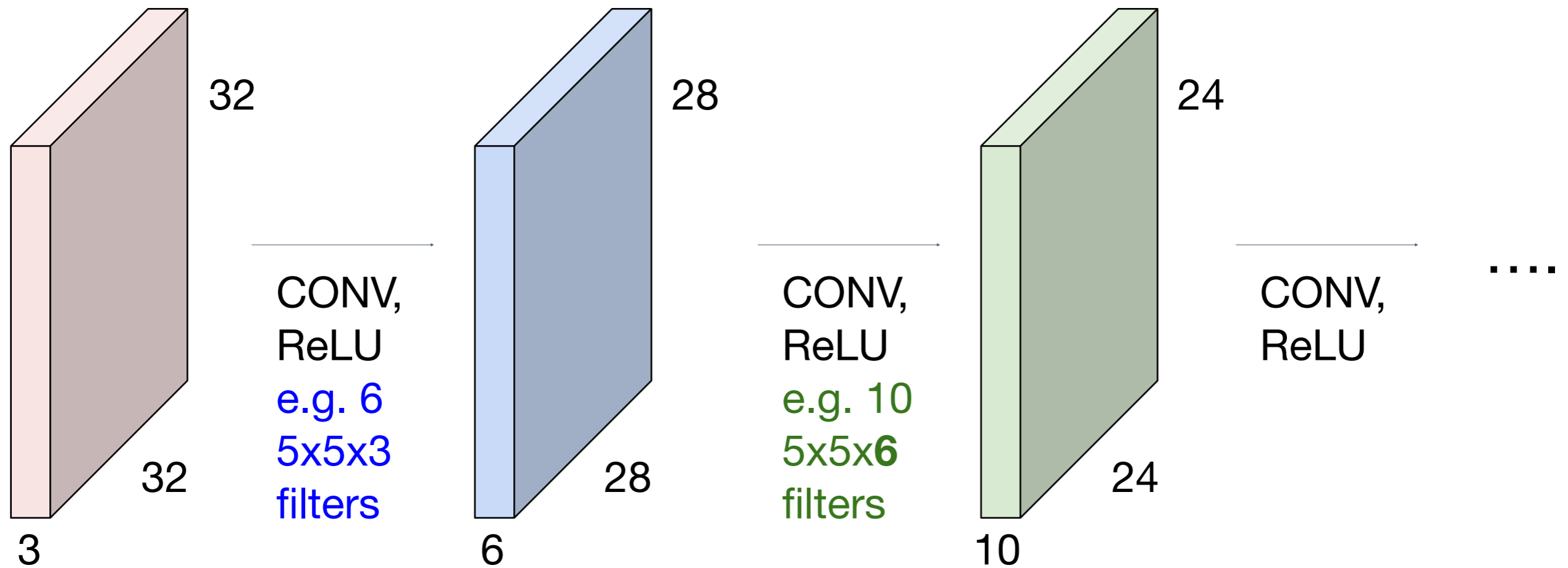
$F = 5 \Rightarrow$ zero pad with 2

$F = 7 \Rightarrow$ zero pad with 3

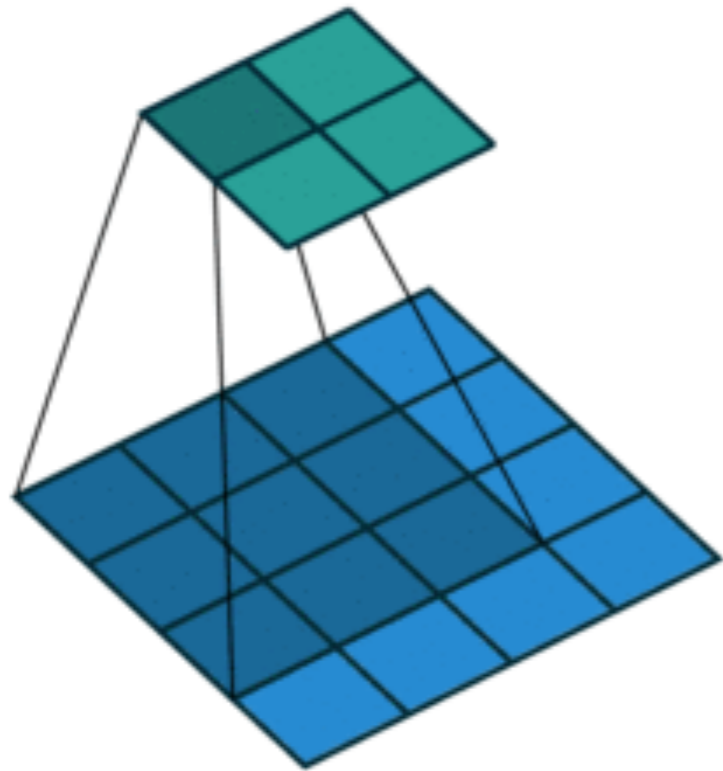
Remember back to...

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!

(32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



Recap: Convolution Layer



$$f = Wx$$

$$W =$$

$$\begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix}$$

(No padding, no strides)

Convoluting a 3×3 kernel over a 4×4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$).

Computing the output values of a 2D discrete convolution

$i_1 = i_2 = 5, k_1 = k_2 = 3, s_1 = s_2 = 2, \text{ and } p_1 = p_2 = 1$

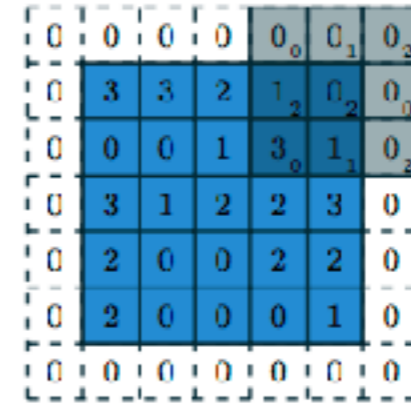
0	1	2
2	2	0
0	1	2



6.0	14.0	17.0
14.0	12.0	12.0
8.0	10.0	17.0



6.0	14.0	17.0
14.0	12.0	12.0
8.0	10.0	17.0



6.0	14.0	17.0
14.0	12.0	12.0
8.0	10.0	17.0



6.0	14.0	17.0
14.0	12.0	12.0
8.0	10.0	17.0



6.0	14.0	17.0
14.0	12.0	12.0
8.0	10.0	17.0



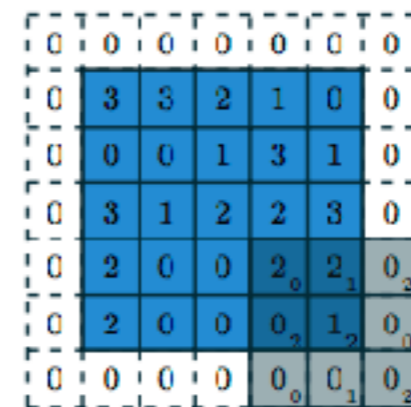
6.0	14.0	17.0
14.0	12.0	12.0
8.0	10.0	17.0



6.0	14.0	17.0
14.0	12.0	12.0
8.0	10.0	17.0



6.0	14.0	17.0
14.0	12.0	12.0
8.0	10.0	17.0



6.0	14.0	17.0
14.0	12.0	12.0
8.0	10.0	17.0

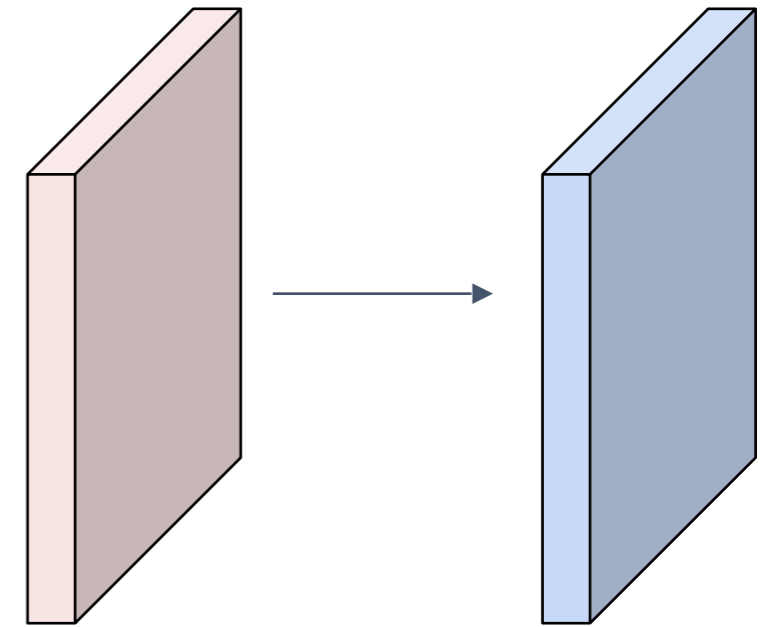
Examples

time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Output volume size: ?



Examples

time:

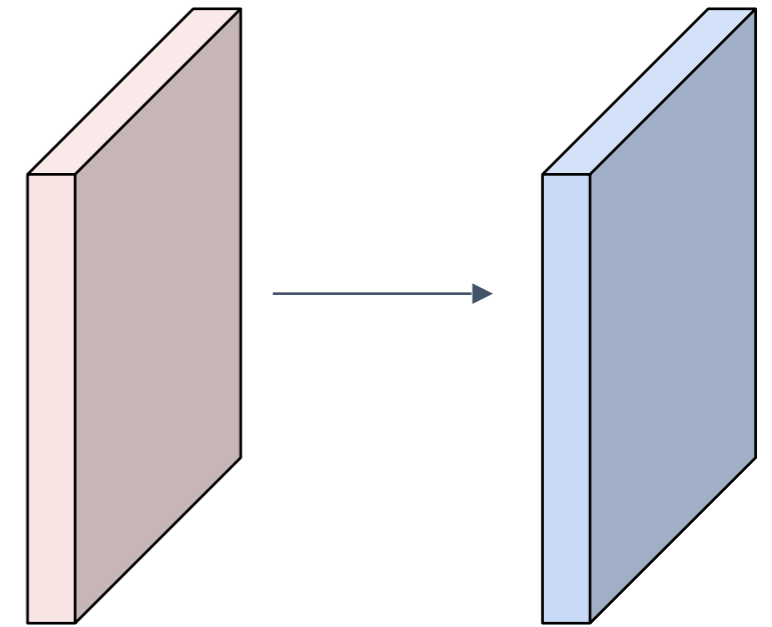
Input volume: **32x32x3**

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

Output volume size:

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

32x32x10



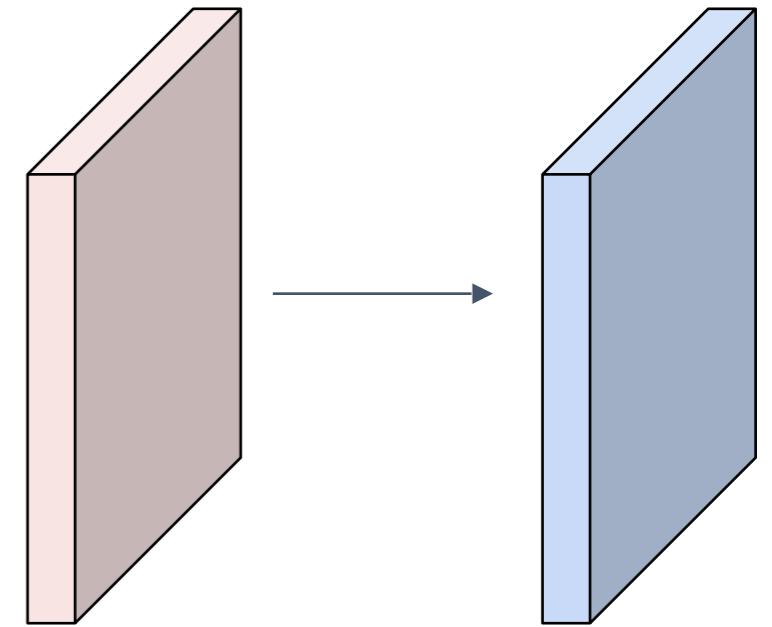
Examples

time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

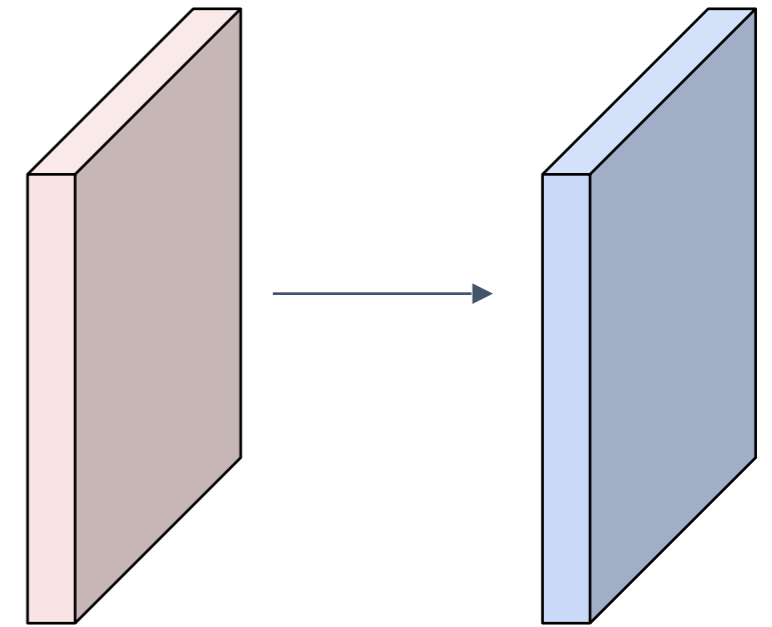


Examples

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)

$\Rightarrow 76*10 = 760$

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:

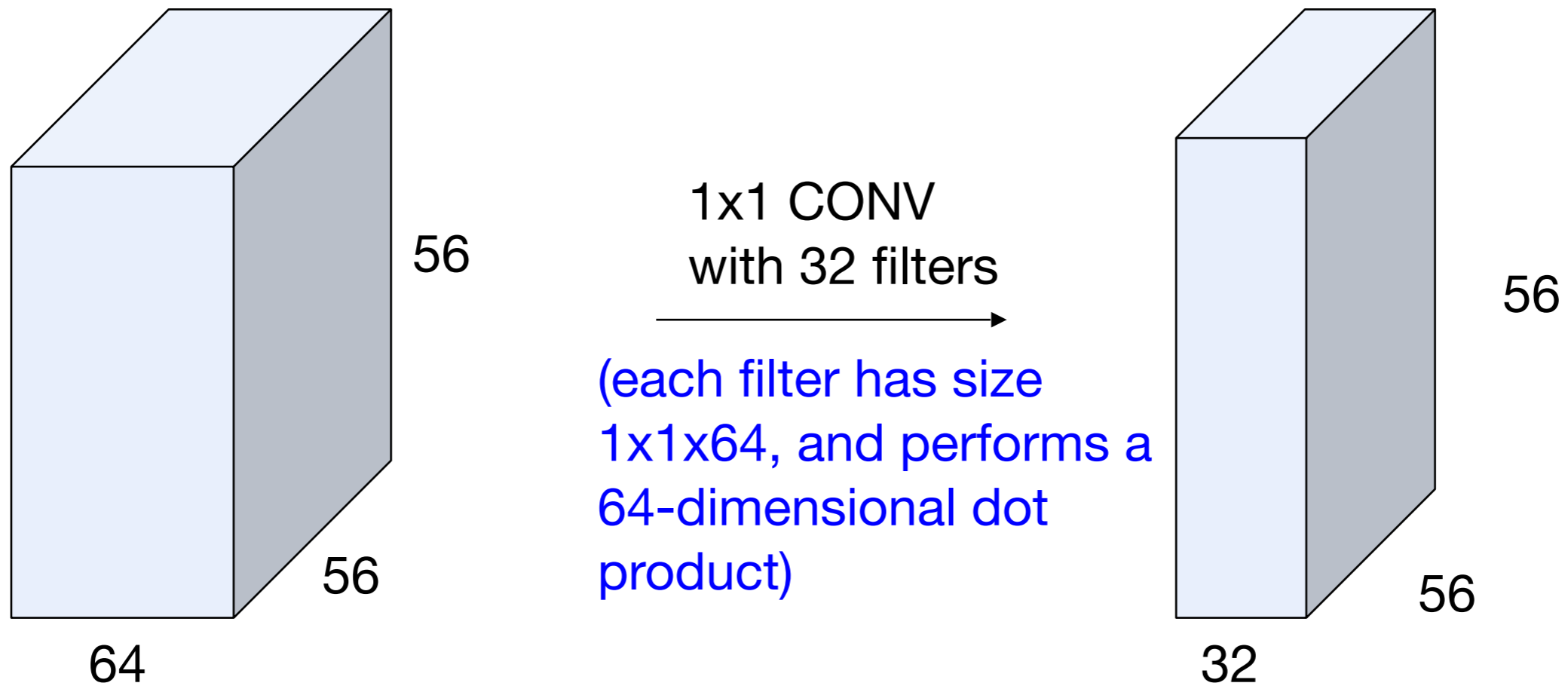
$K =$ (powers of 2, e.g. 32, 64, 128, 512)

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

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)



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 $(kH - 1)/2$.

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)
```

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 .

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 .

Example: CONV layer in Lasagne

```
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.

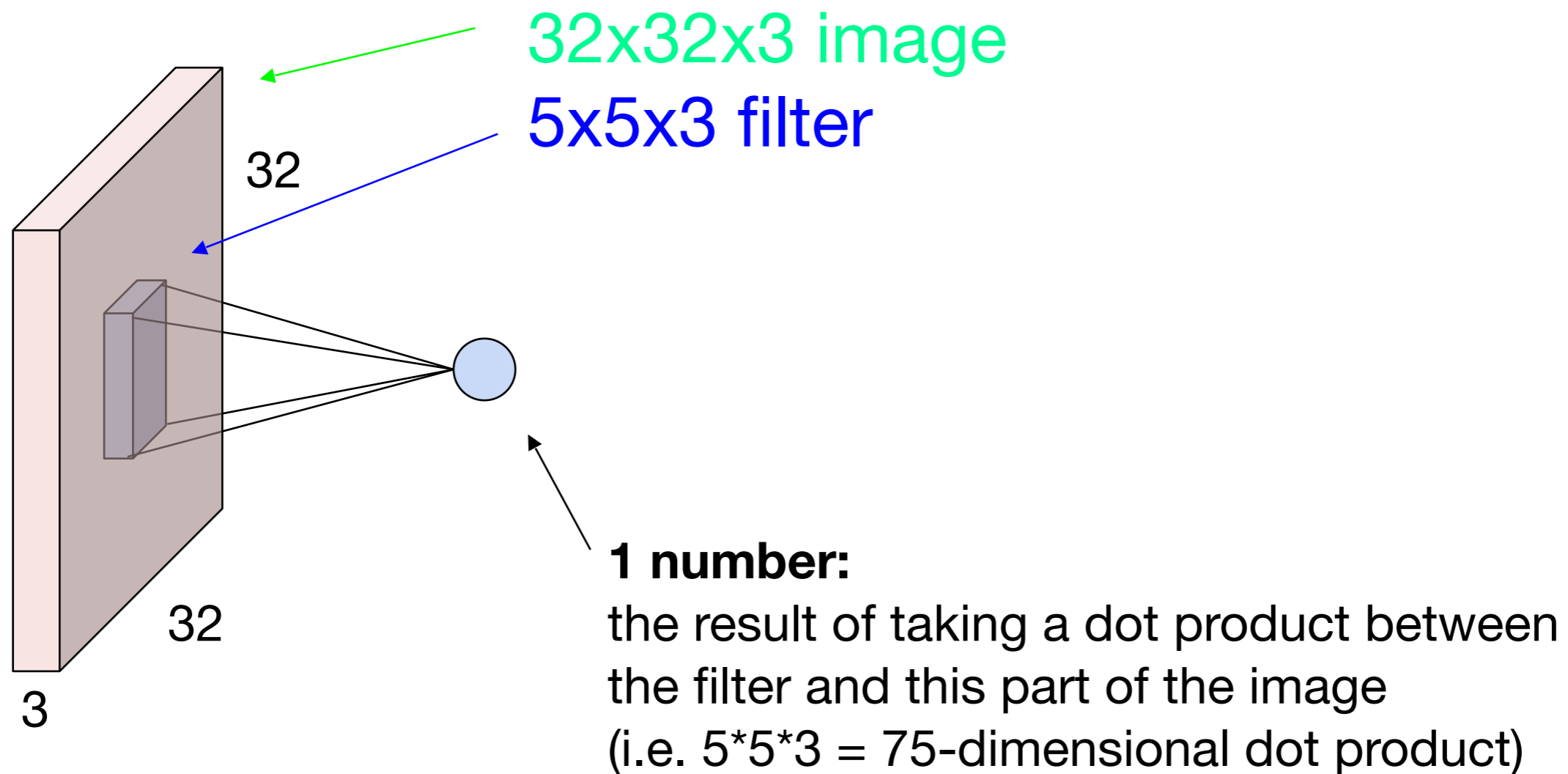
`'same'` pads with half the filter size (rounded down) on both sides. When `stride=1` this results in an output size equal to the input size. Even filter size is not supported.

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

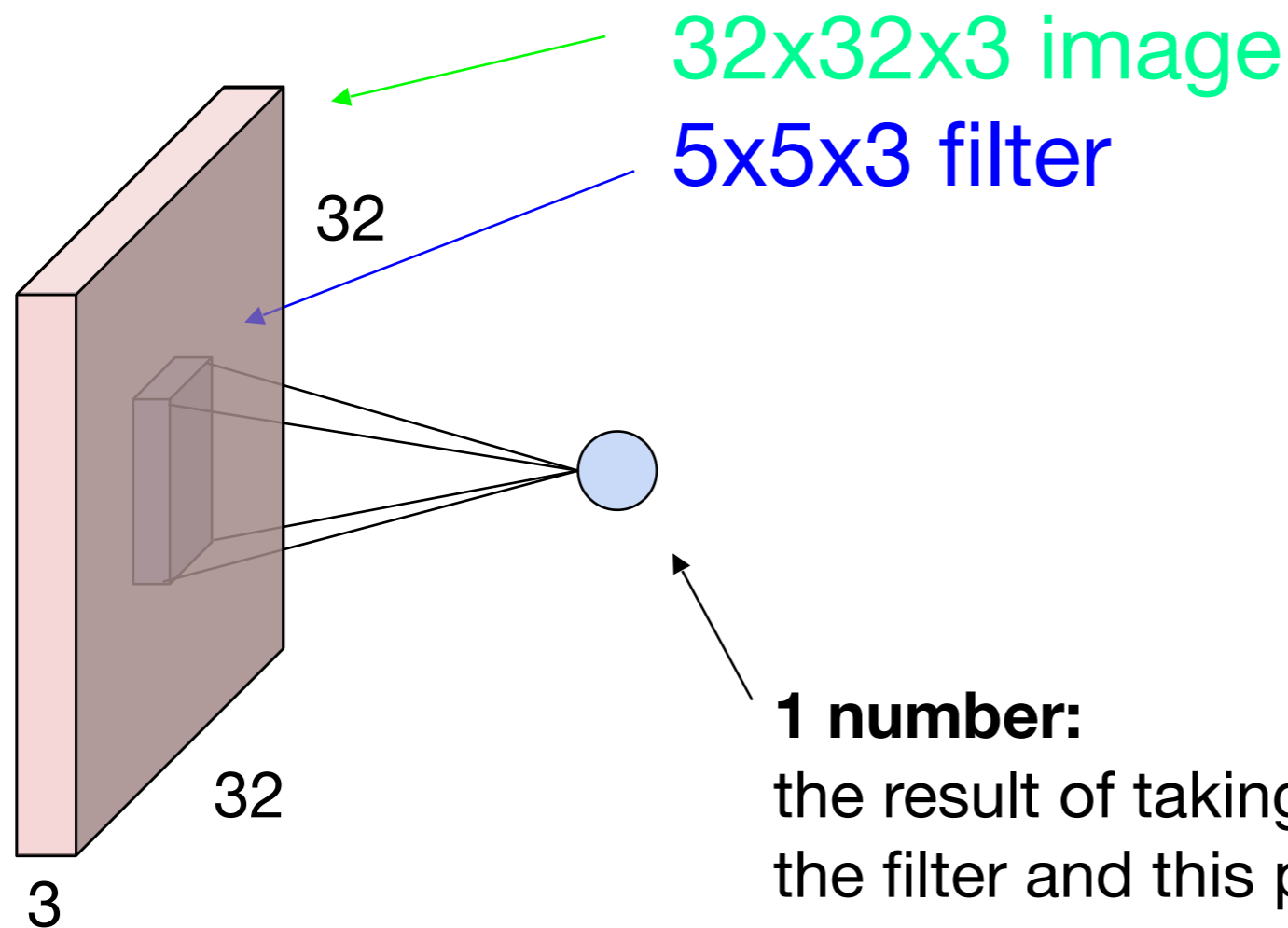
Summary. To summarize, the Conv Layer:

- 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

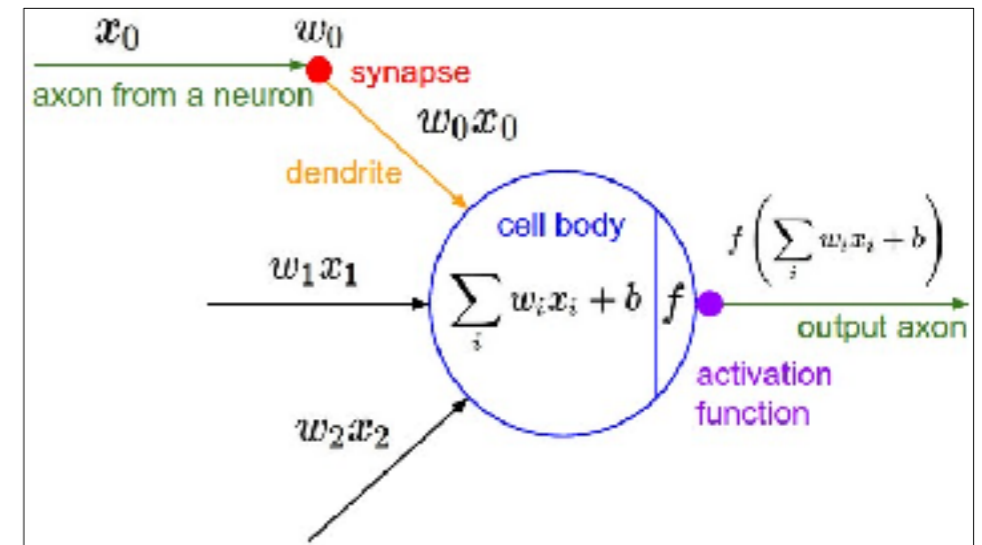


The brain/neuron view of CONV Layer



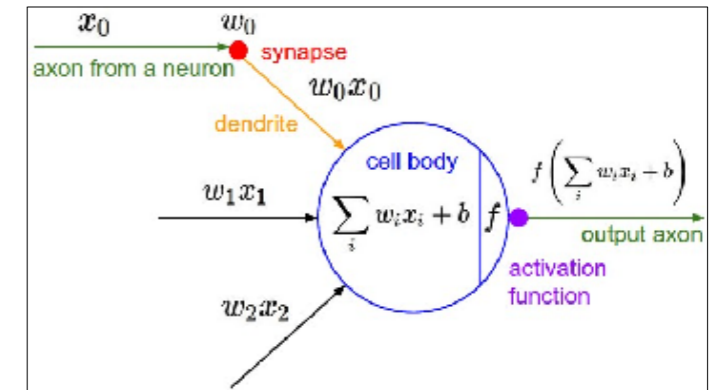
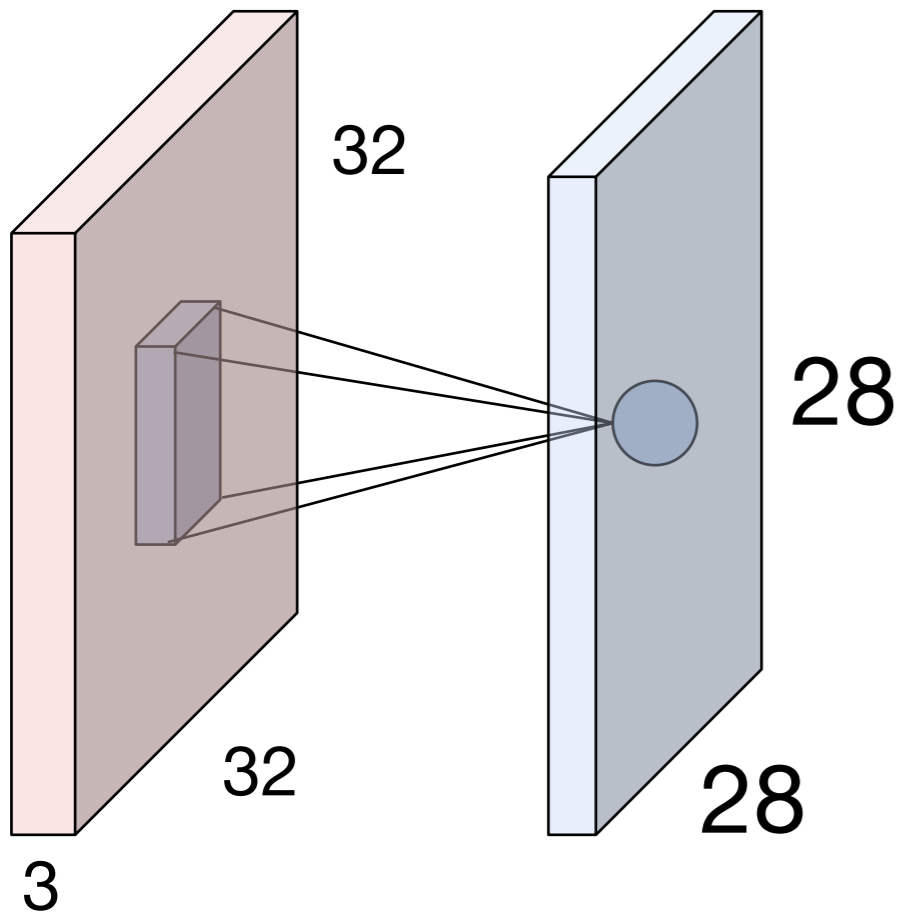
1 number:

the result of taking a dot product between the filter and this part of the image (i.e. $5*5*3 = 75$ -dimensional dot product)



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

The brain/neuron view of CONV Layer

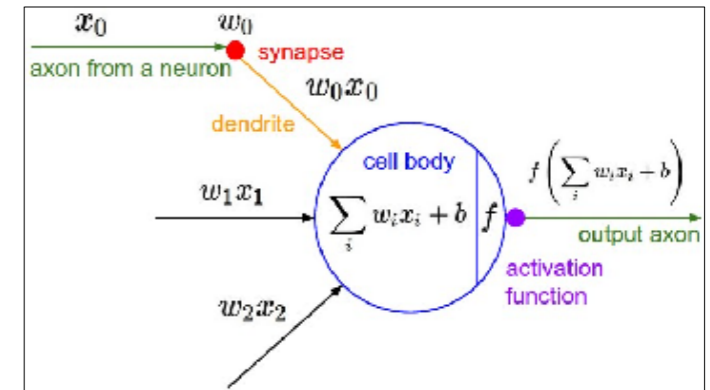
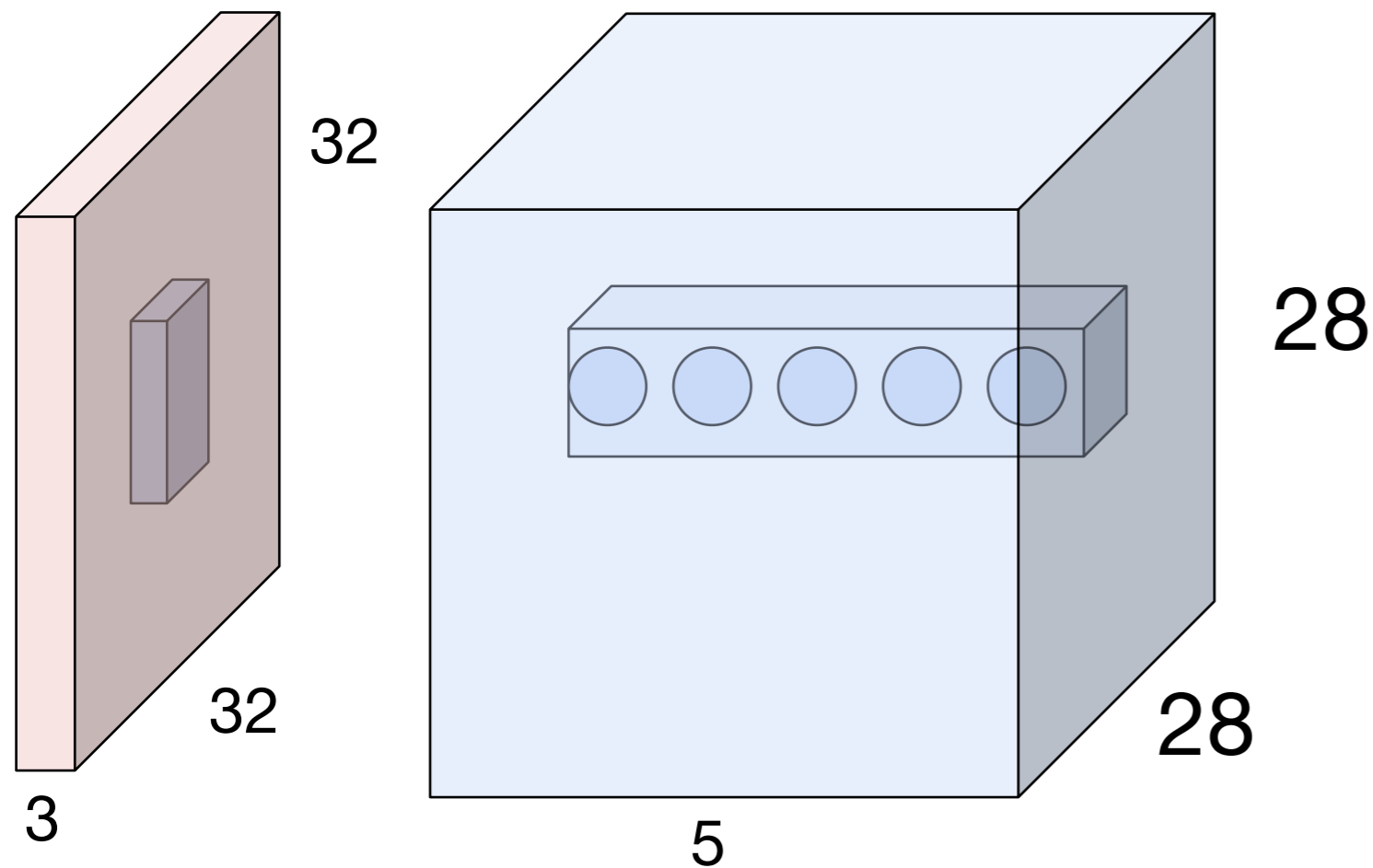


An activation map is a 28x28 sheet of neuron outputs:

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

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

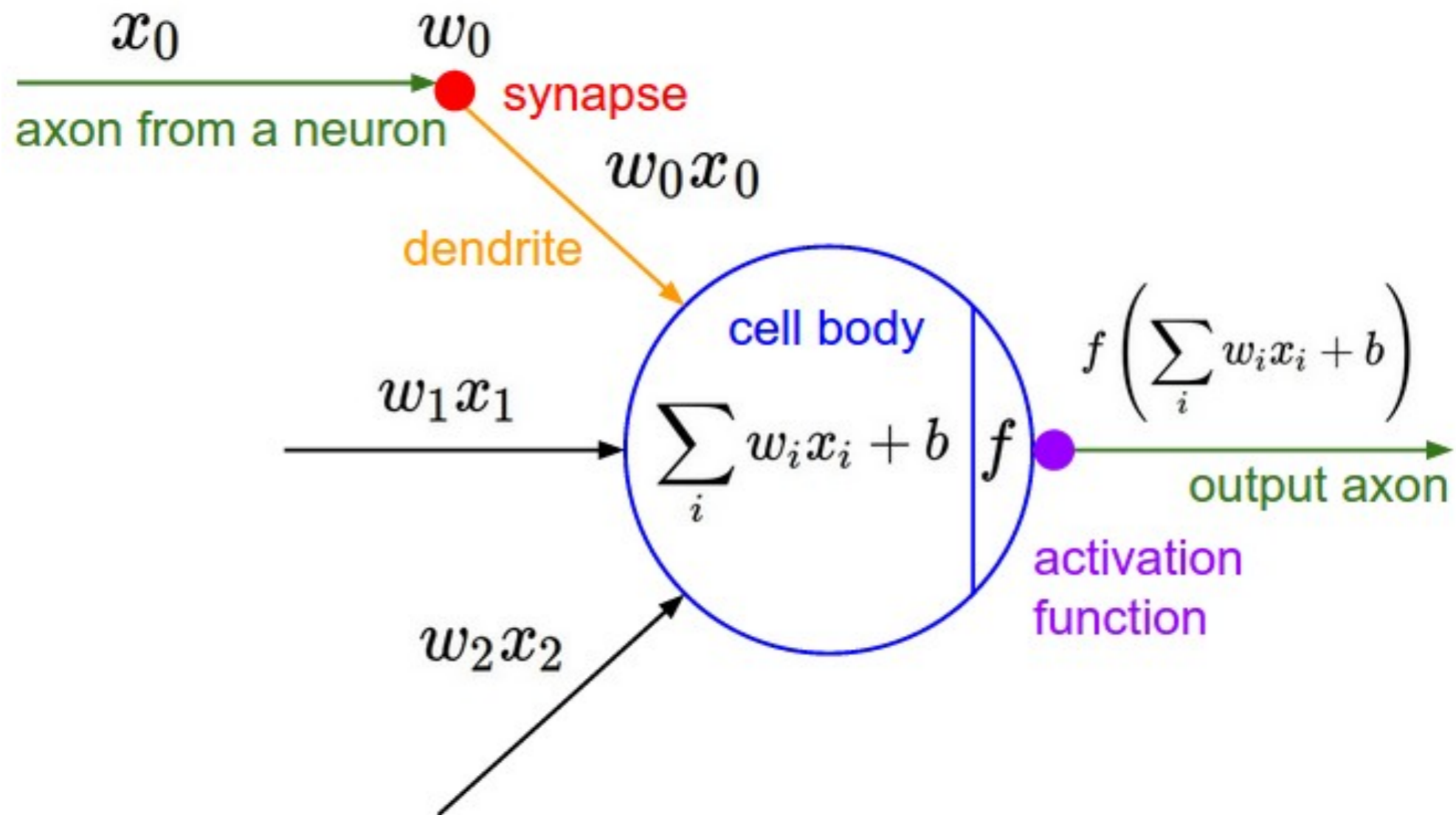
The brain/neuron view of CONV Layer



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

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

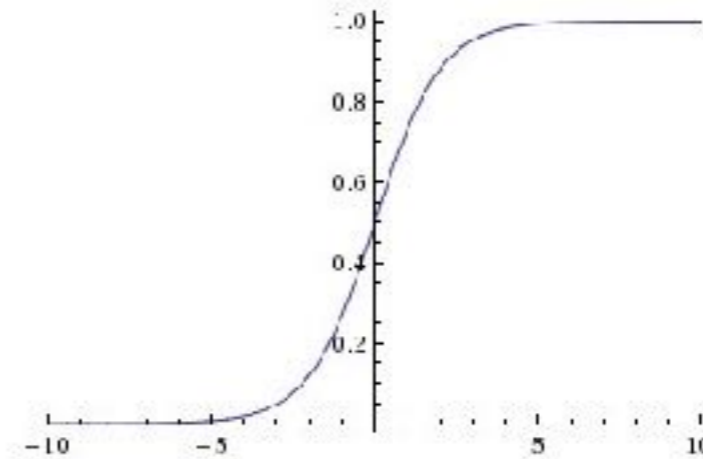
Activation Functions



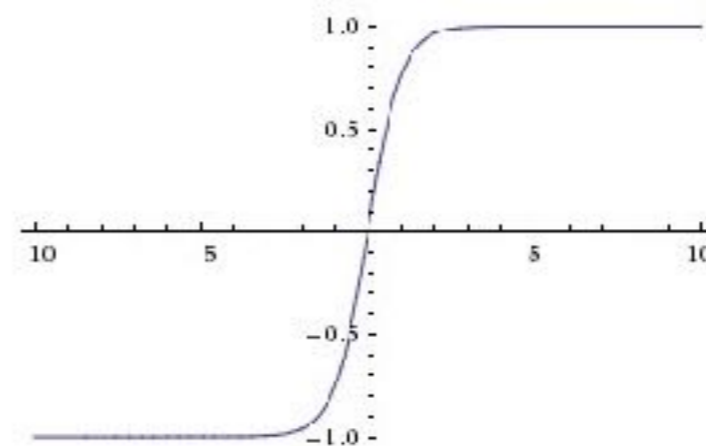
Activation Functions

Sigmoid

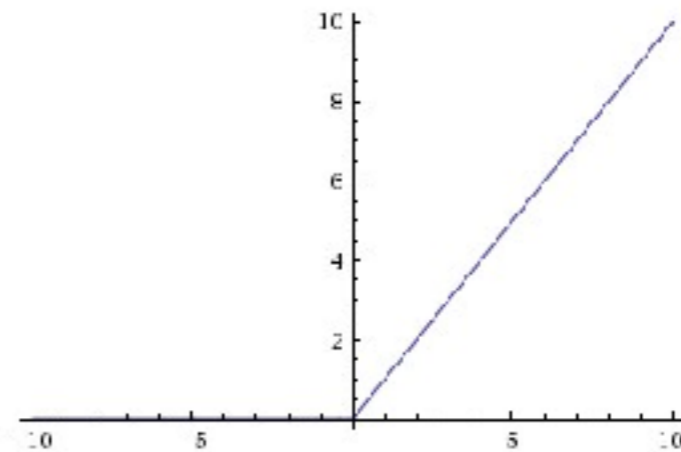
$$\sigma(x) = 1 / (1 + e^{-x})$$



tanh $\tanh(x)$

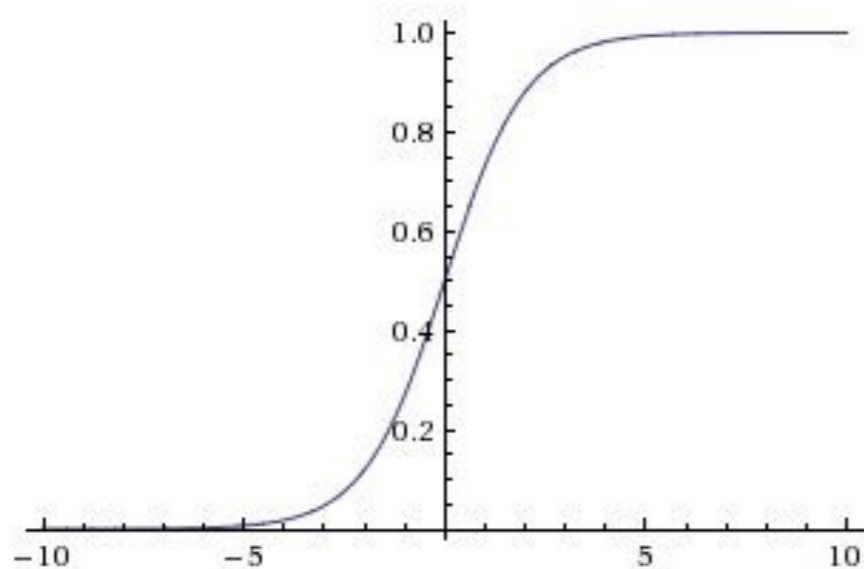


ReLU $\max(0, x)$



Activation Functions

$$\sigma(x) = 1/(1 + e^{-x})$$



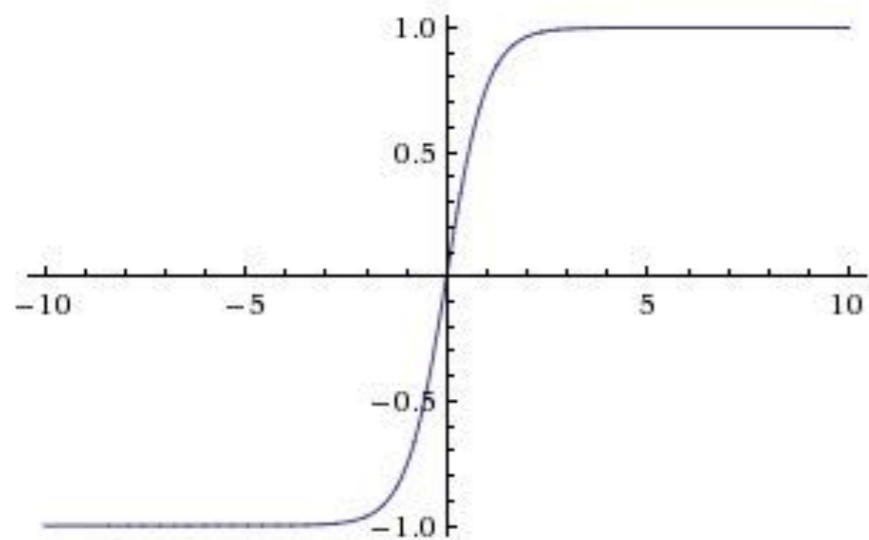
Sigmoid

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

3 problems:

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered
3. $\exp()$ is a bit compute expensive

Activation Functions



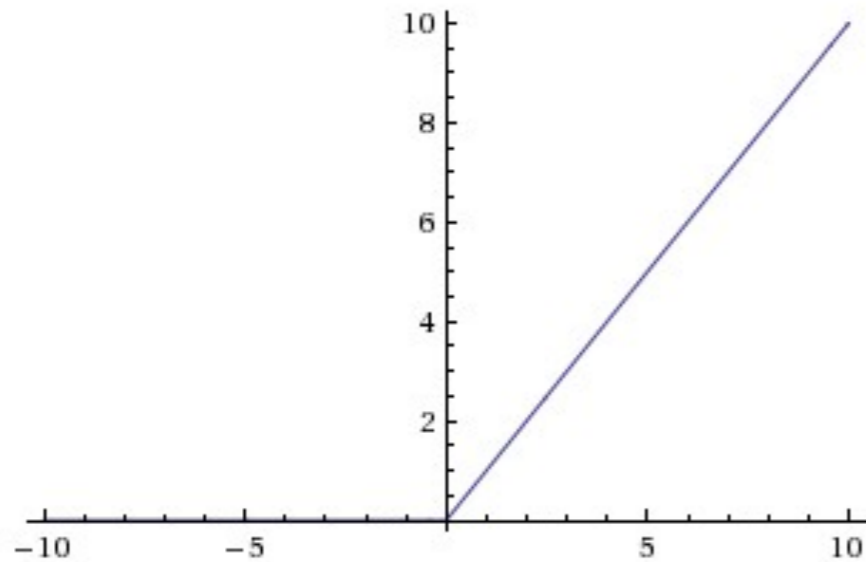
$\tanh(x)$

- Squashes numbers to range [-1,1]
- zero centered (nice)
- still kills gradients when saturated :(

[LeCun et al., 1991]

Activation Functions

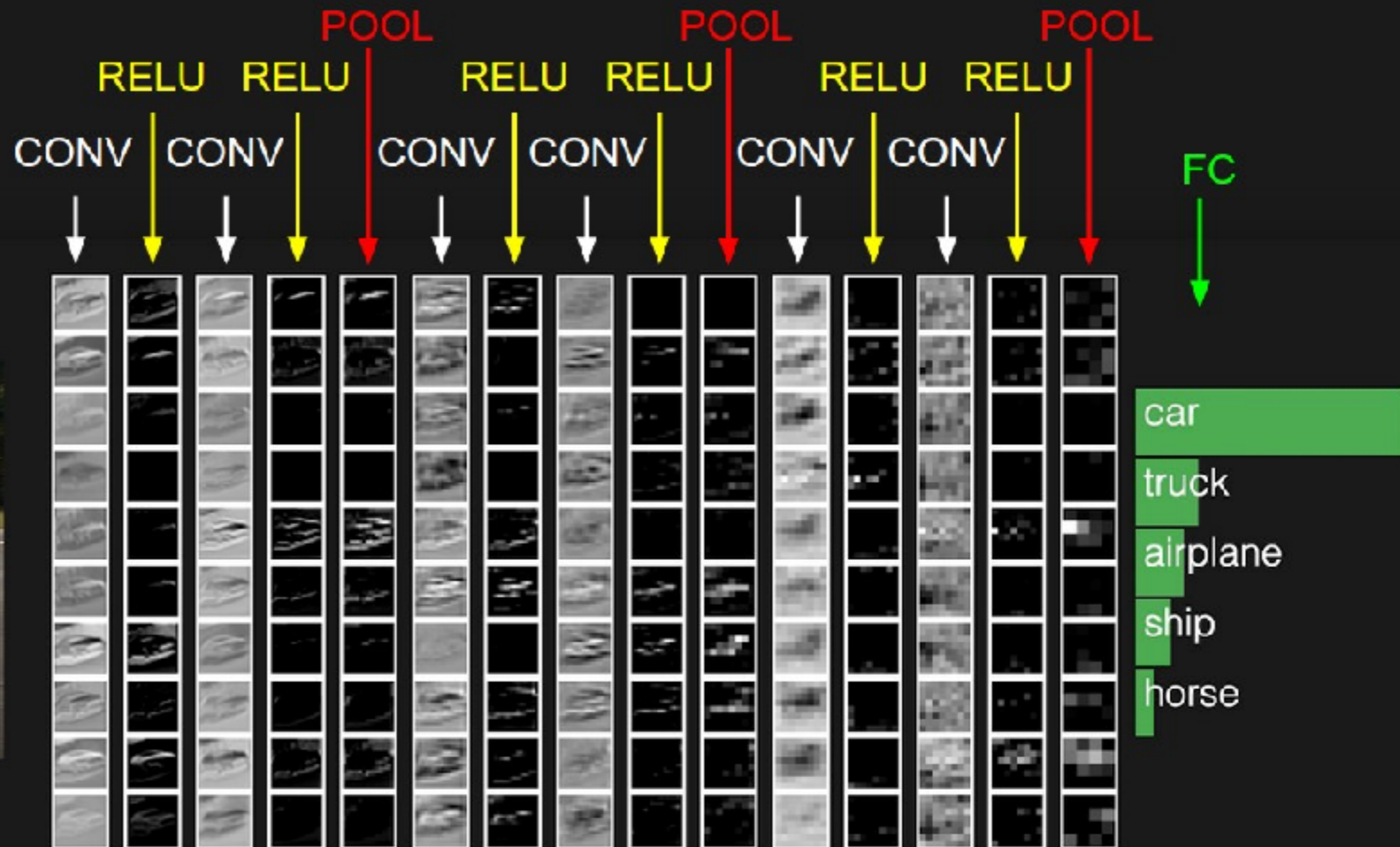
- Computes $f(x) = \max(0, x)$
- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)



ReLU (Rectified Linear Unit)

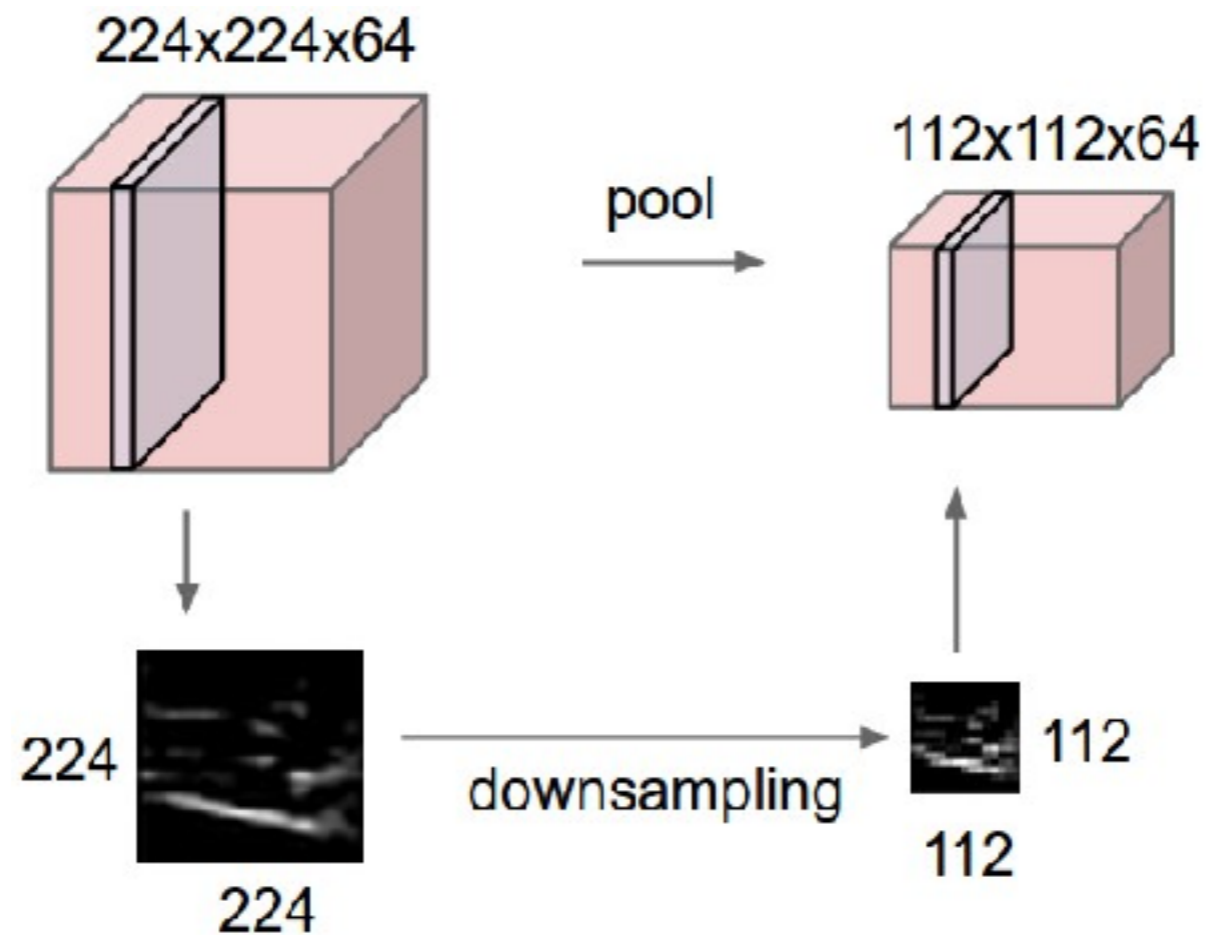
[Krizhevsky et al., 2012]

two more layers to go: POOL/FC



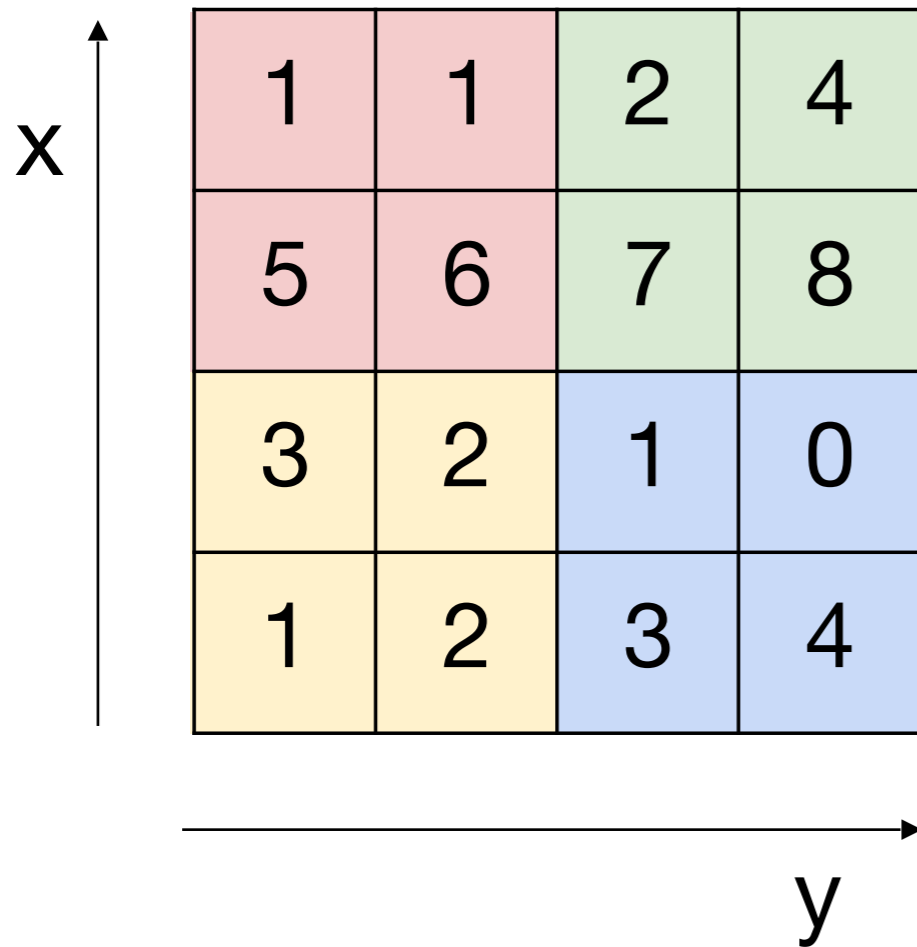
Pooling layer

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

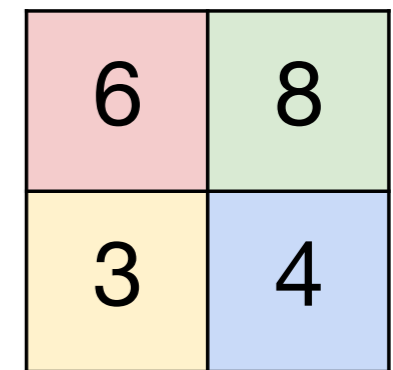
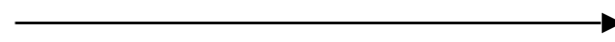


Max Pooling

Single depth



max pool with 2x2 filters and stride 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

Common settings:

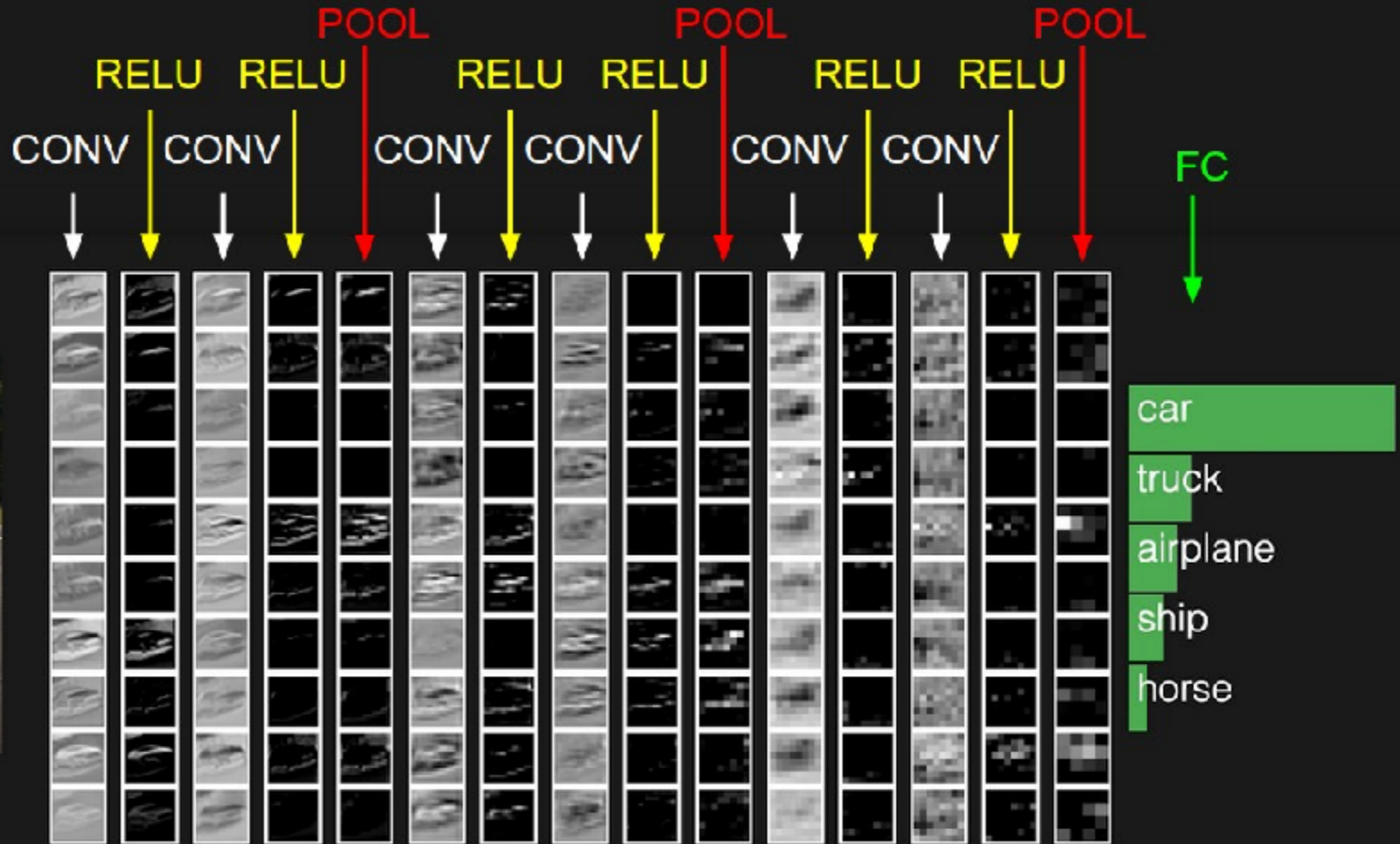
$$F = 2, S = 2$$

$$F = 3, S = 2$$

- 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

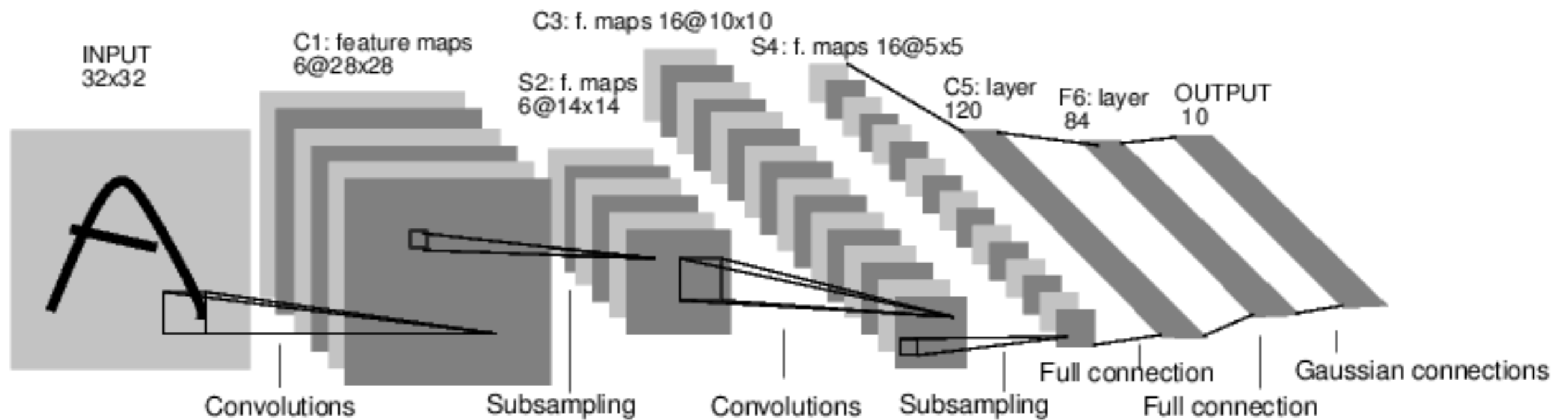


[ConvNetJS demo: training on CIFAR-10]

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

Case studies

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



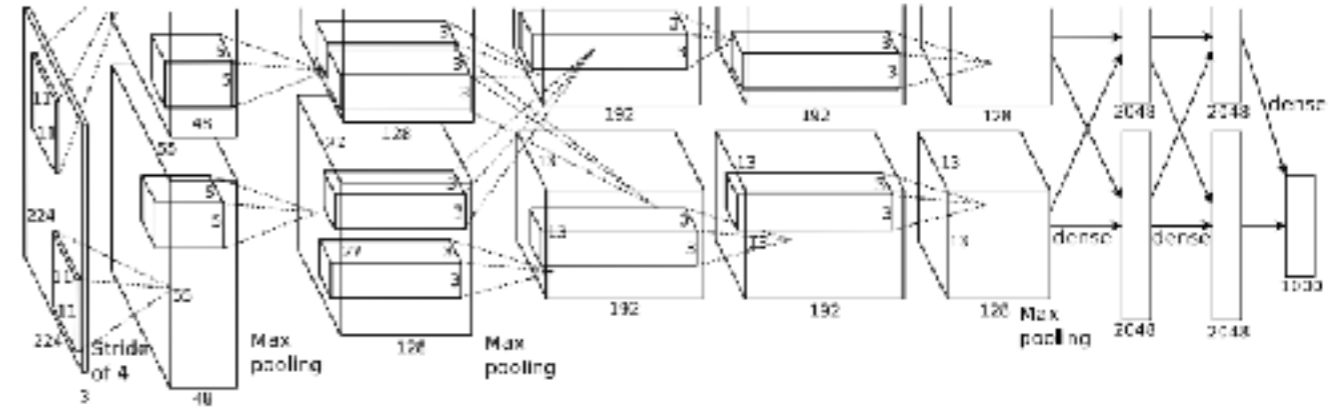
Conv filters were 5x5, applied at stride 1

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

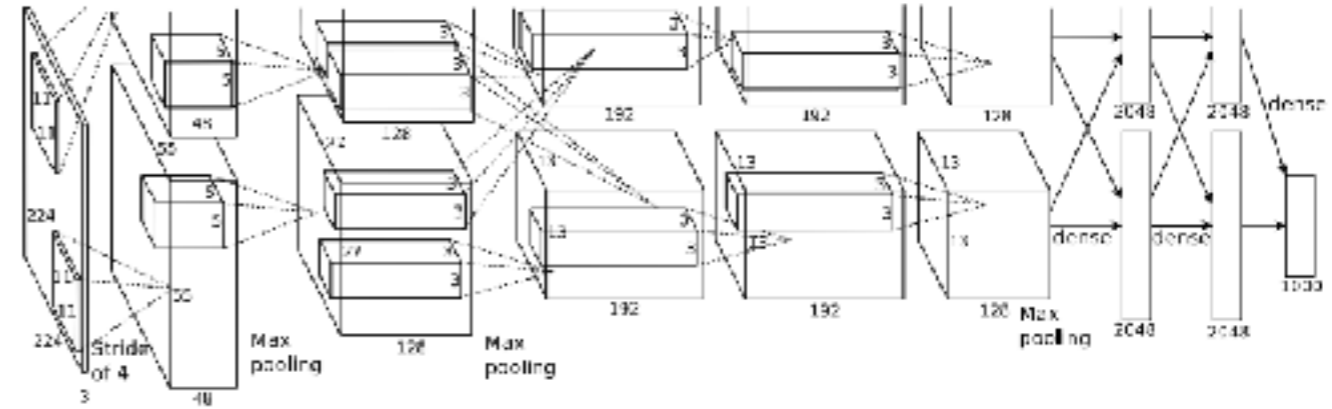
First layer (CONV1): 96 11x11 filters applied at stride 4

=>

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

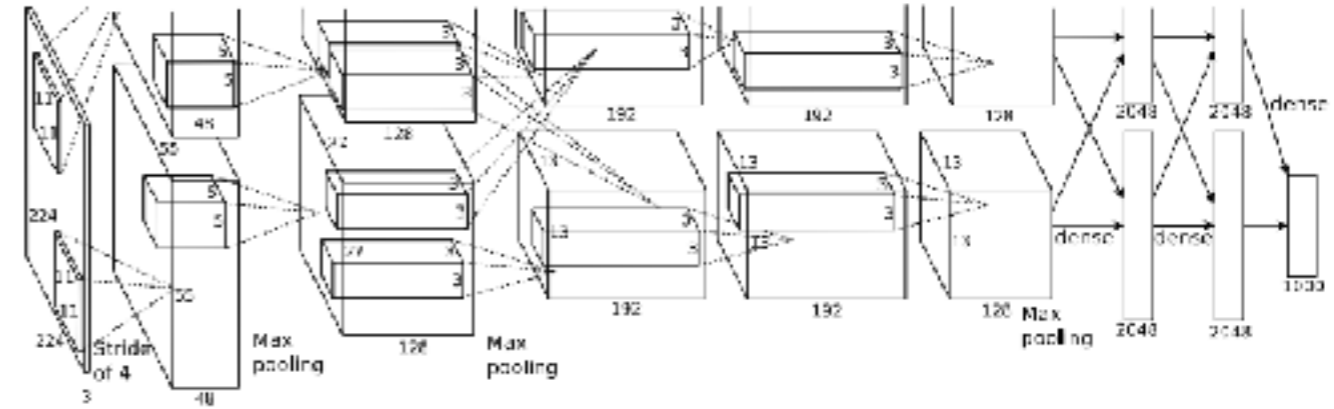
=>

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

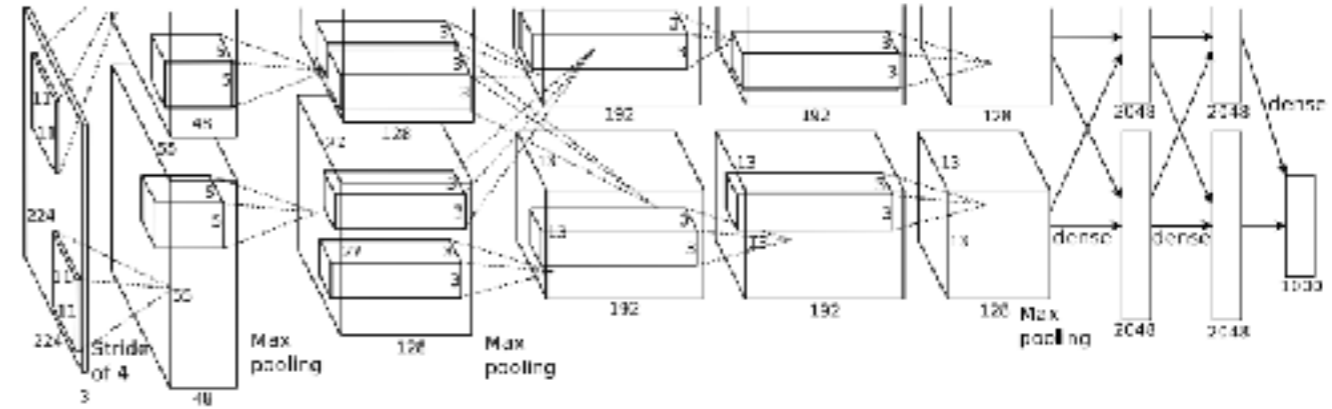
=>

Output volume **[55x55x96]**

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

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

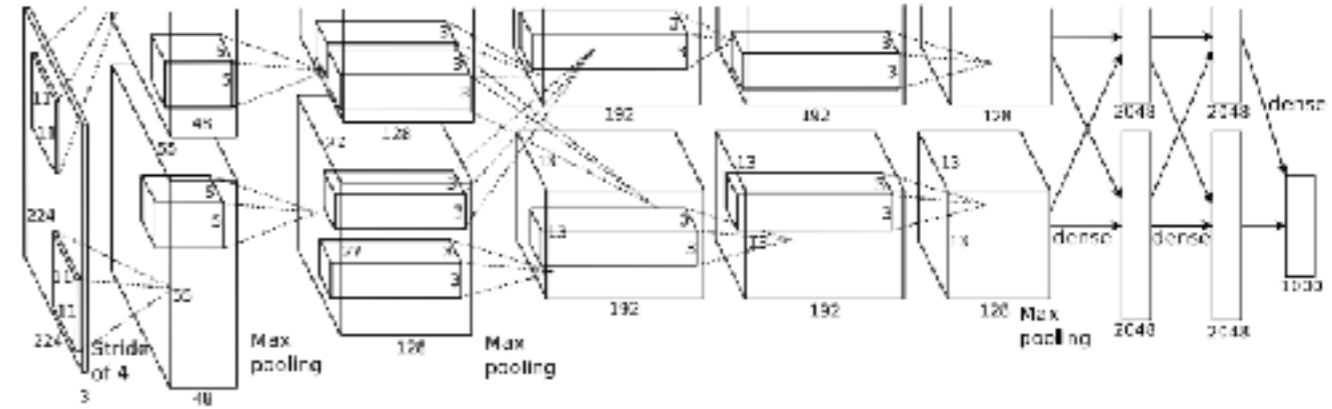
After CONV1: 55x55x96

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

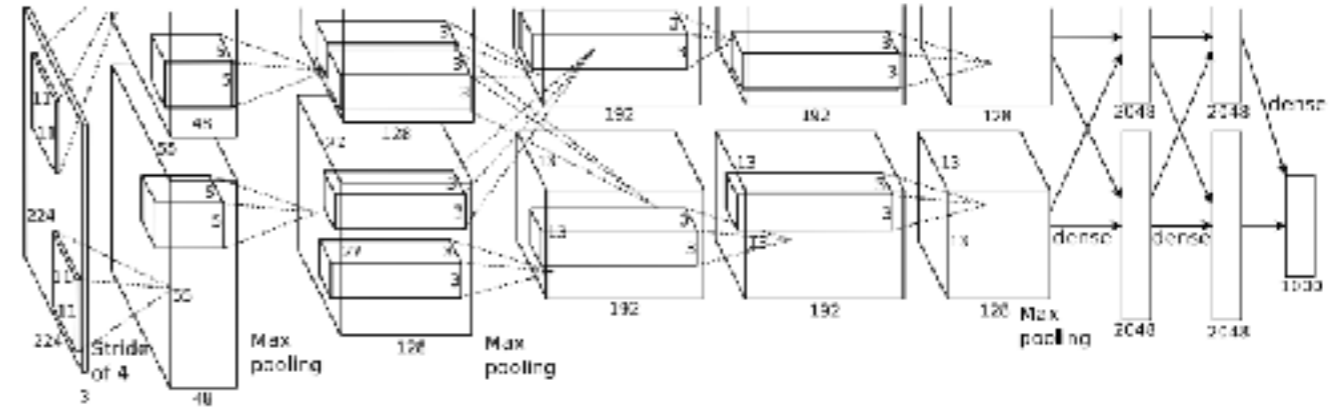
Second layer (POOL1): 3x3 filters applied at stride 2

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

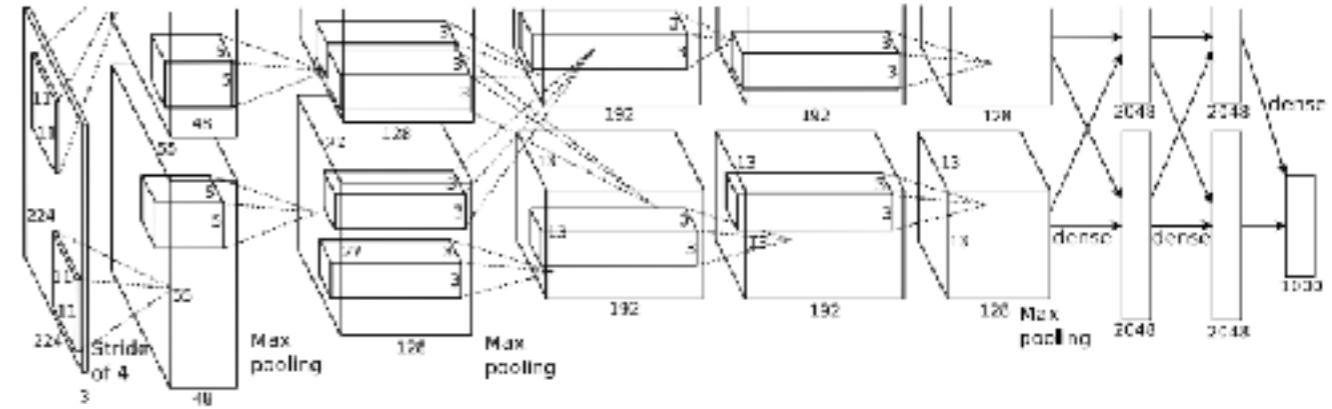
Second layer (POOL1): 3x3 filters applied at stride 2

Output volume: 27x27x96

Parameters: 0!

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

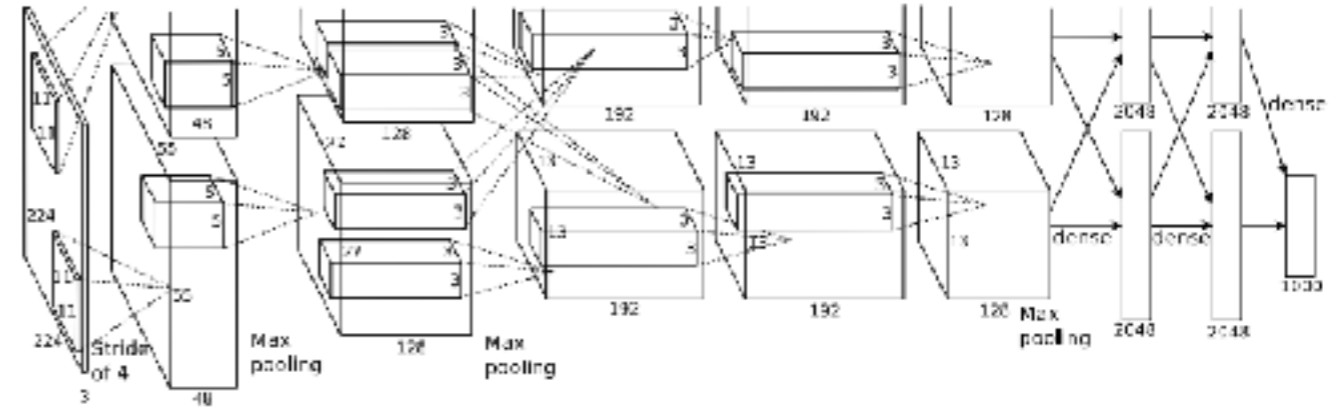
After CONV1: 55x55x96

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

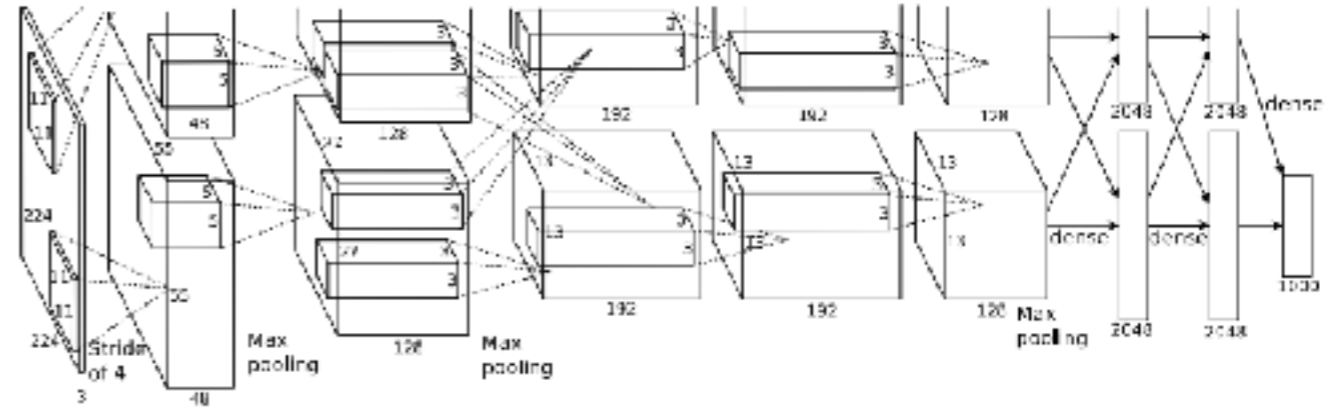
[4096] **FC6**: 4096 neurons

[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

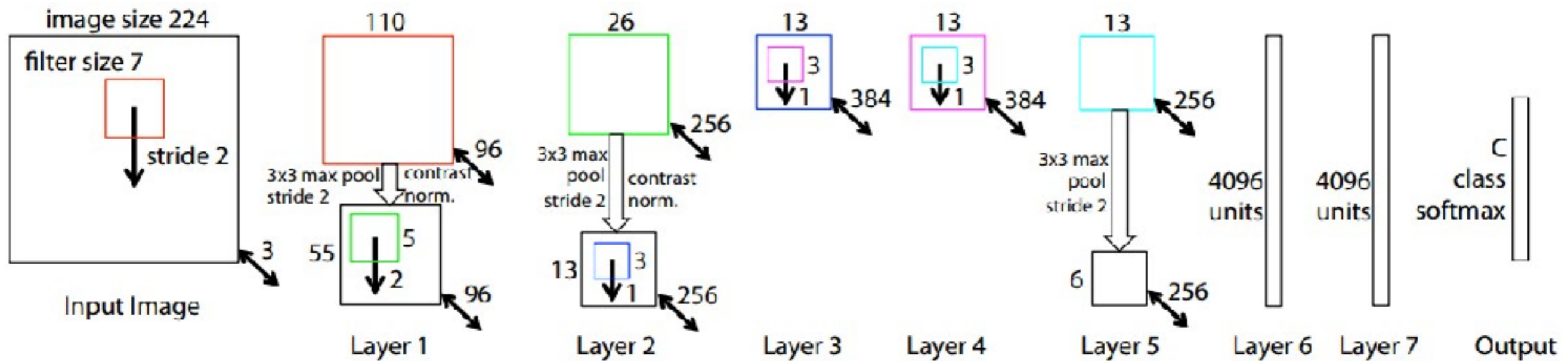
[4096] **FC7**: 4096 neurons

[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% \rightarrow 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 × 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
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: Number of parameters (in millions).

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

(not counting biases)

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: $4096*1000 = 4,096,000$

ConvNet Configuration			
B	C	D	E
13 weight layers	16 weight layers	16 weight layers	19
Input (224 x 224 RGB image)			
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
maxpool			
conv3-256	conv3-256	conv3-256	conv3-256
conv3-256	conv3-256	conv3-256	conv3-256
	conv1-256	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
maxpool			
conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512
	conv1-512	conv3-512	conv3-512
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

(not counting biases)

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: $4096*1000 = 4,096,000$

ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 x 224 RGB image)			
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
maxpool			
conv3-256	conv3-256	conv3-256	conv3-256
conv3-256	conv3-256	conv3-256	conv3-256
	conv1-256	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
maxpool			
conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512
	conv1-512	conv3-512	conv3-512
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

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

TOTAL params: 138M parameters

(not counting biases)

Note:

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: $4096*1000 = 4,096,000$

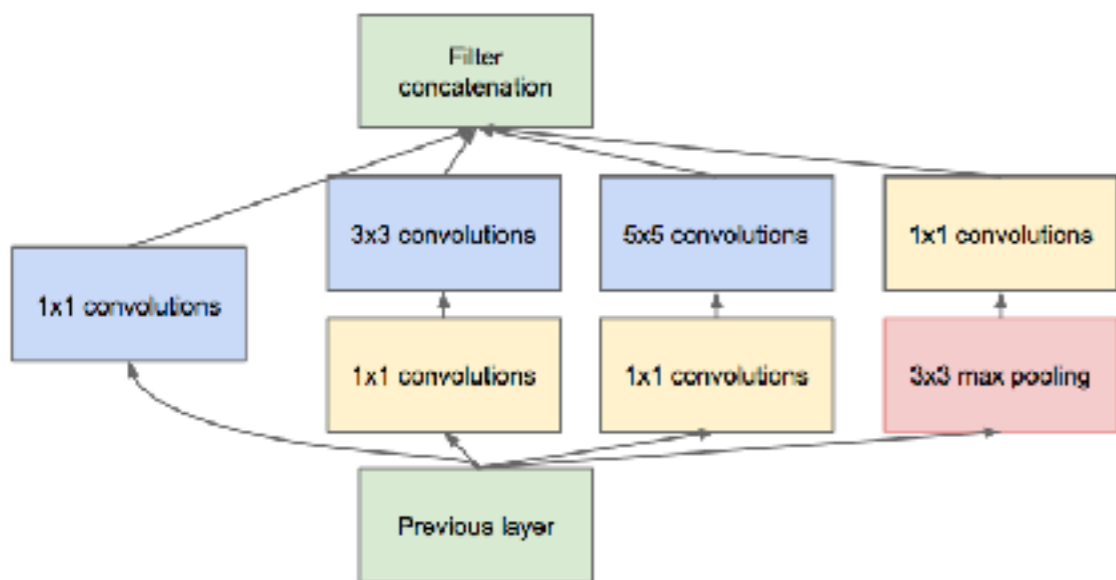
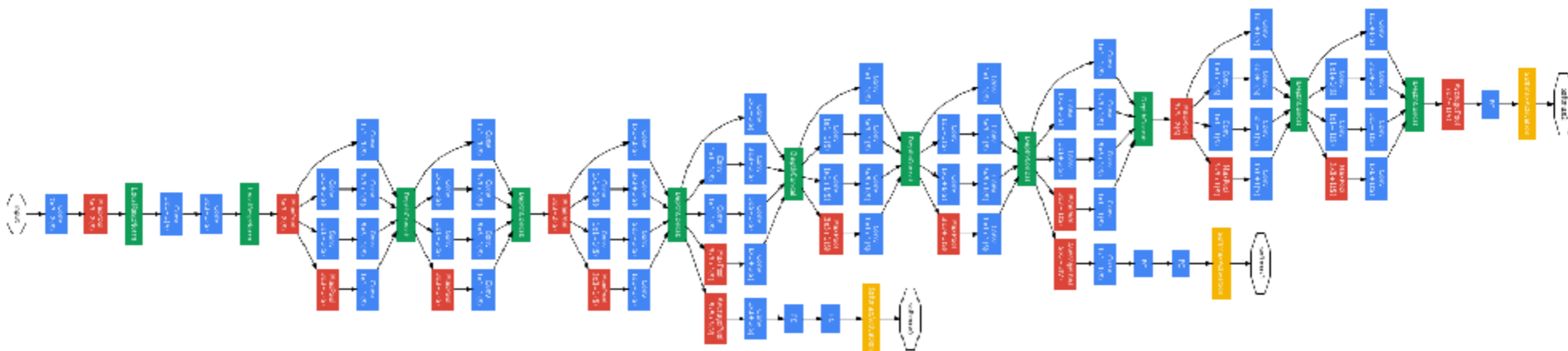
Most memory is in early CONV

Most params are in late FC

TOTAL memory: 24M * 4 bytes \approx 93MB / image (only forward! \sim *2 for bwd)

TOTAL params: 138M parameters

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



Inception module

ILSVRC 2014 winner (6.7% top 5 error)

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

*improvements are relative numbers

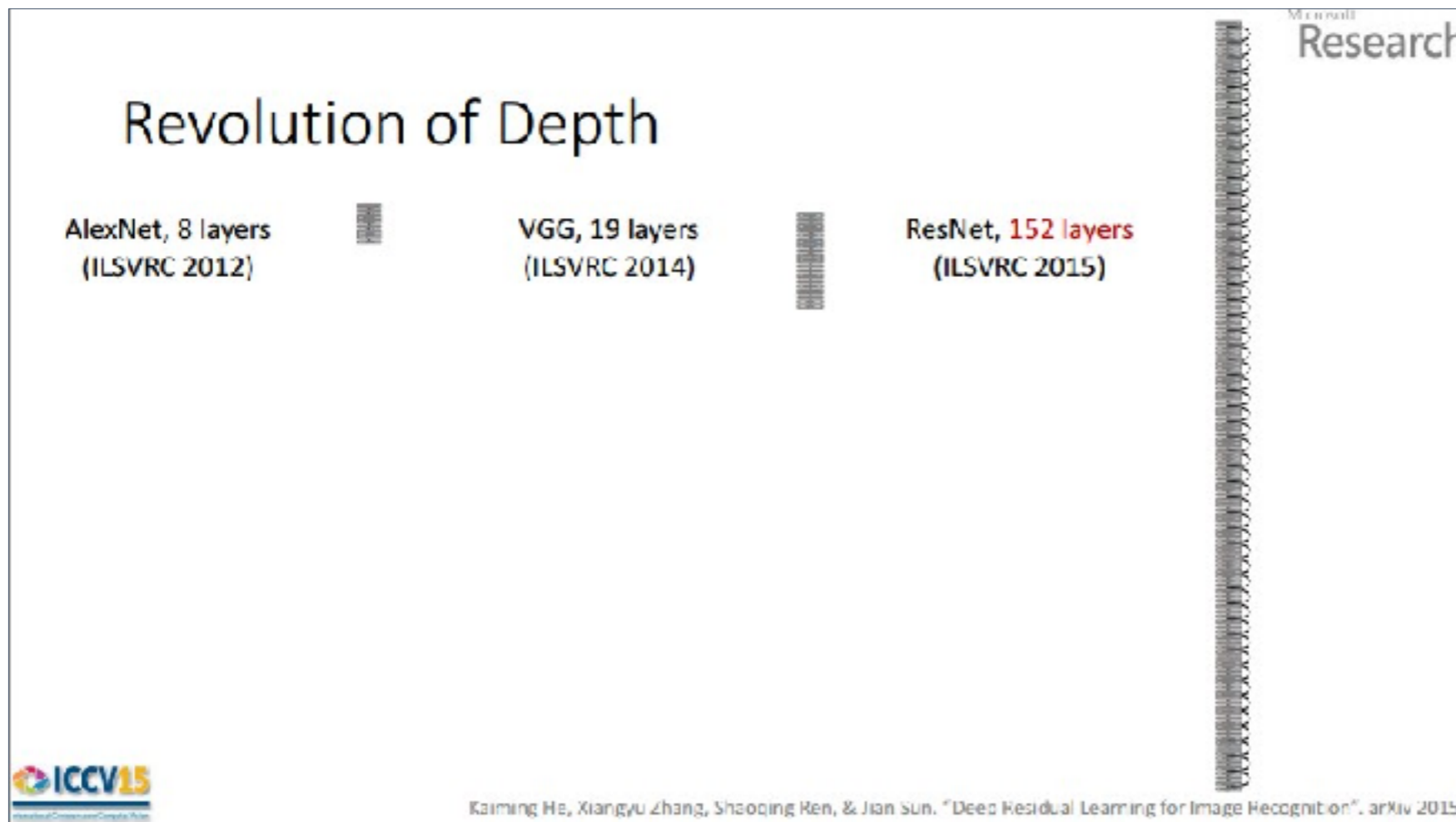


Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

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

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

ILSVRC 2015 winner
(3.6% top 5 error)

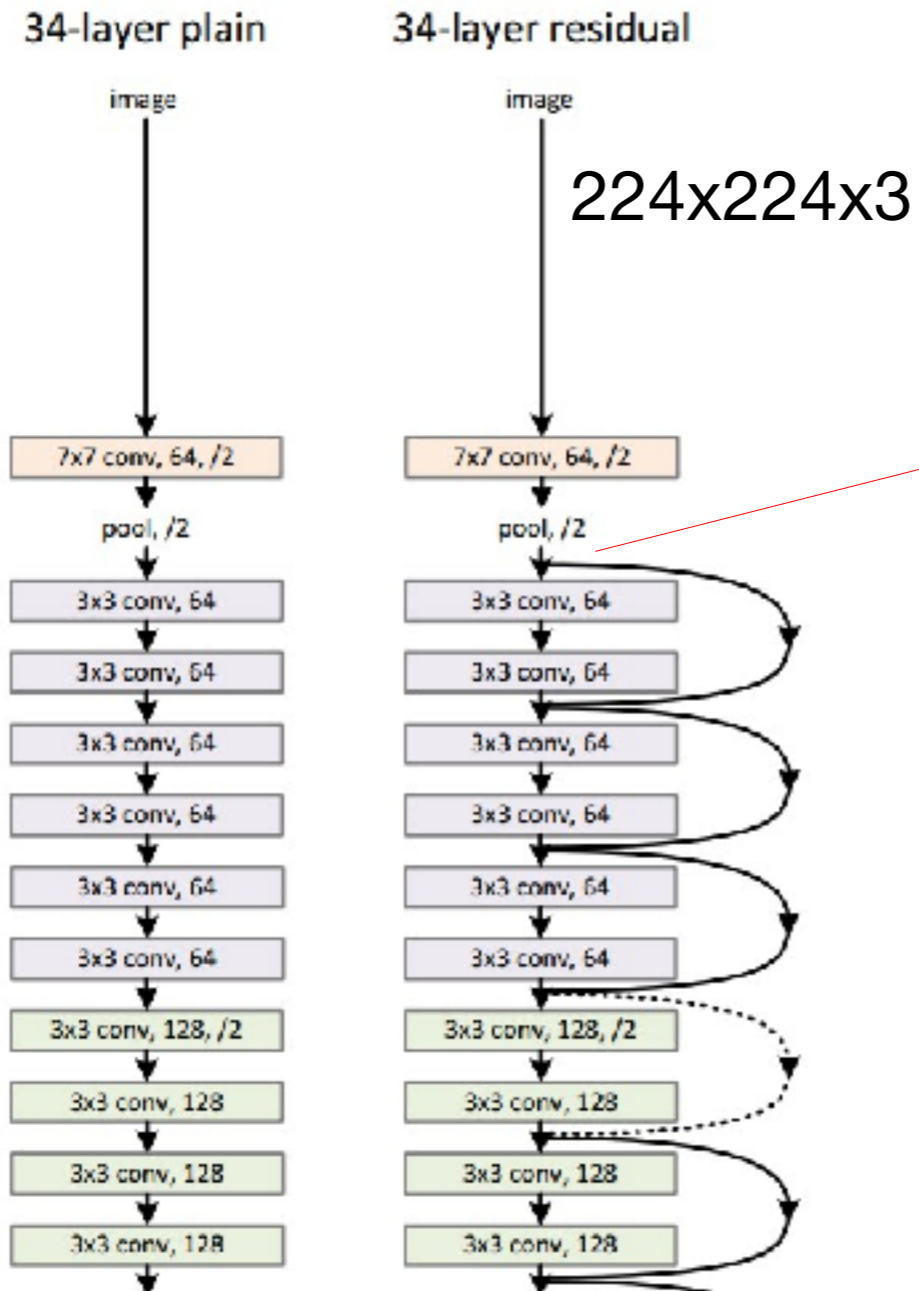


2-3 weeks of
training on 8 GPU
machine

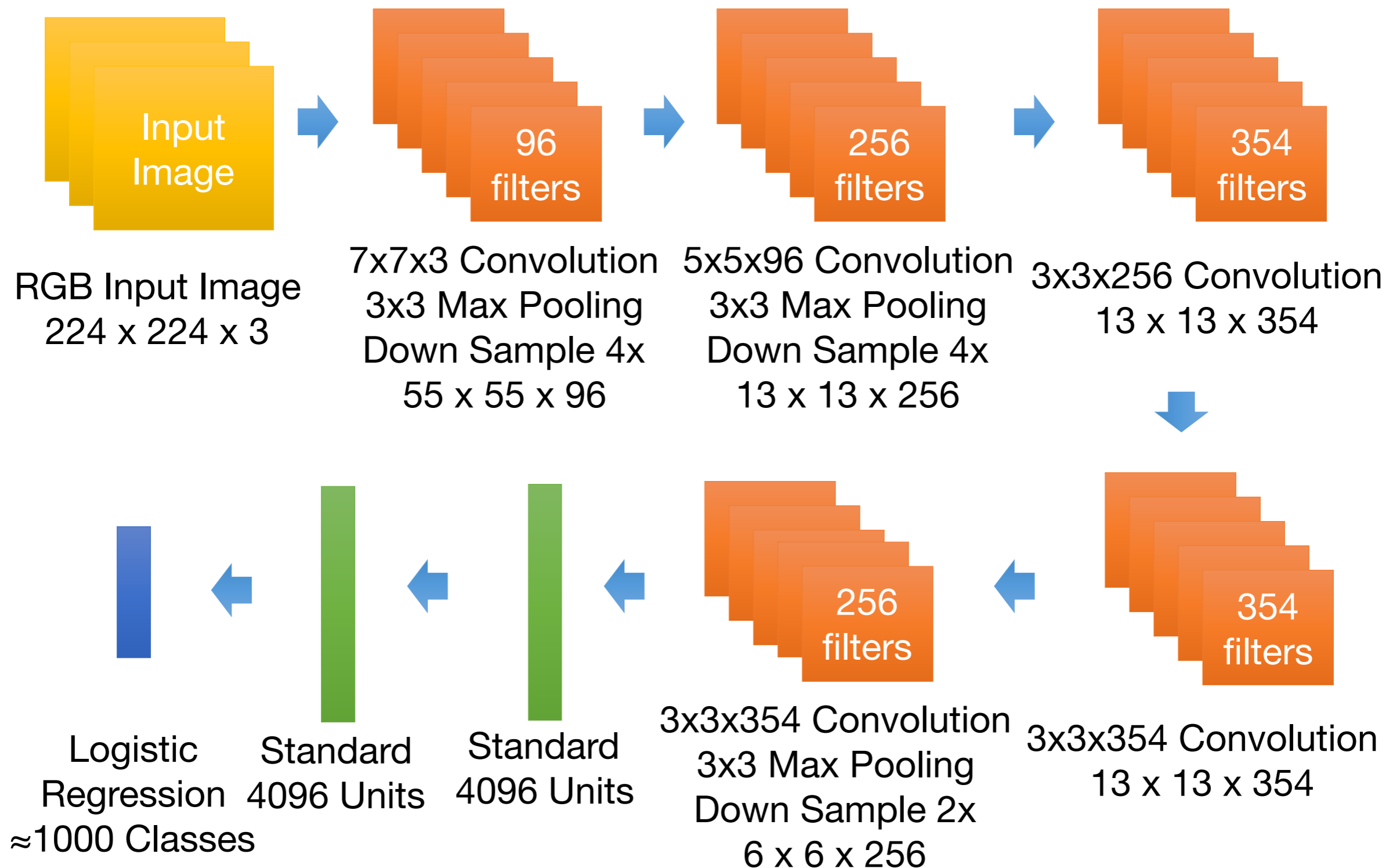
at runtime: faster
than a VGGNet!
(even though it has
8x more layers)

(slide from Kaiming He's recent presentation)

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



spatial dimension
only 56x56!

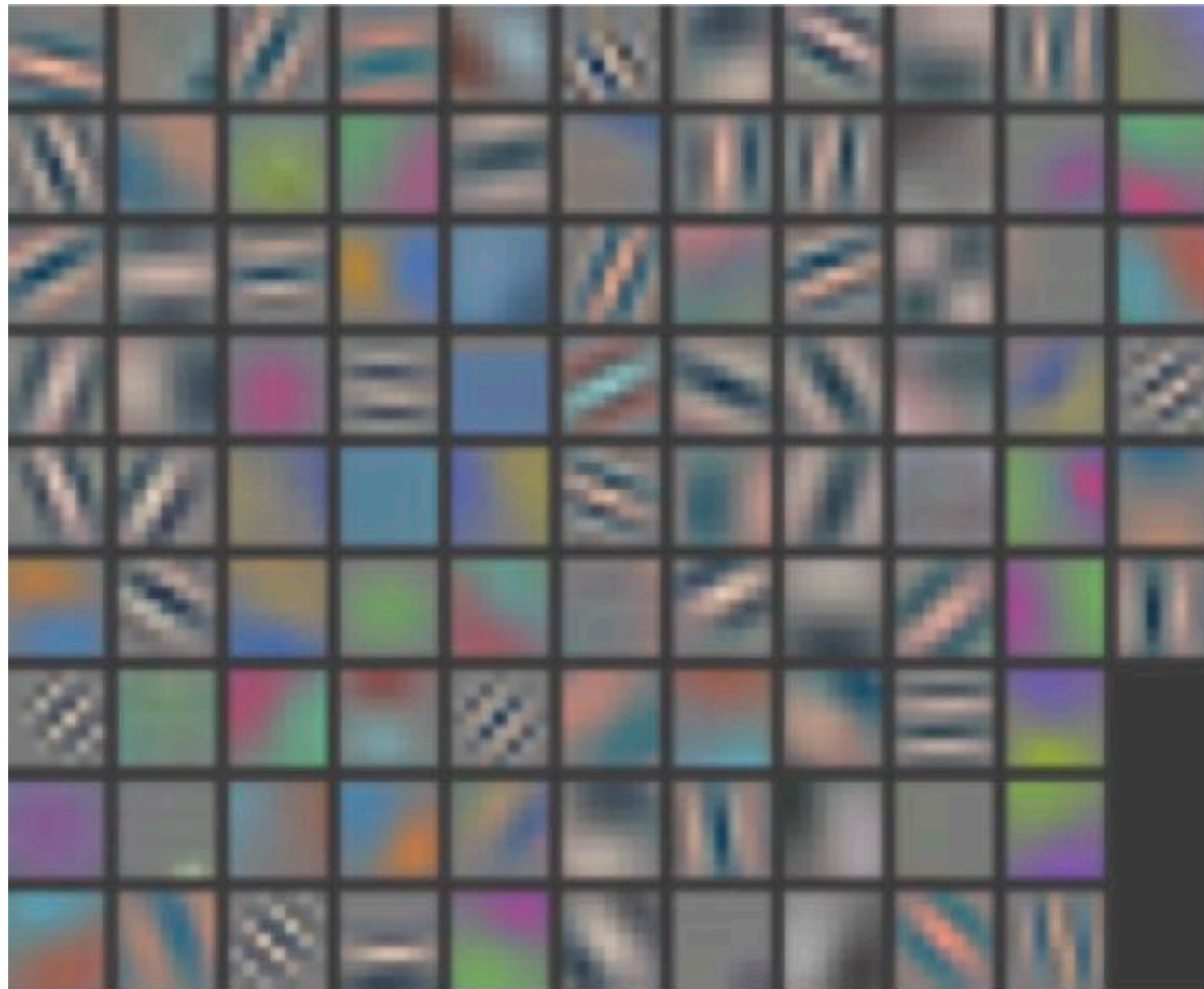


<http://www.image-net.org/>

<http://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf>

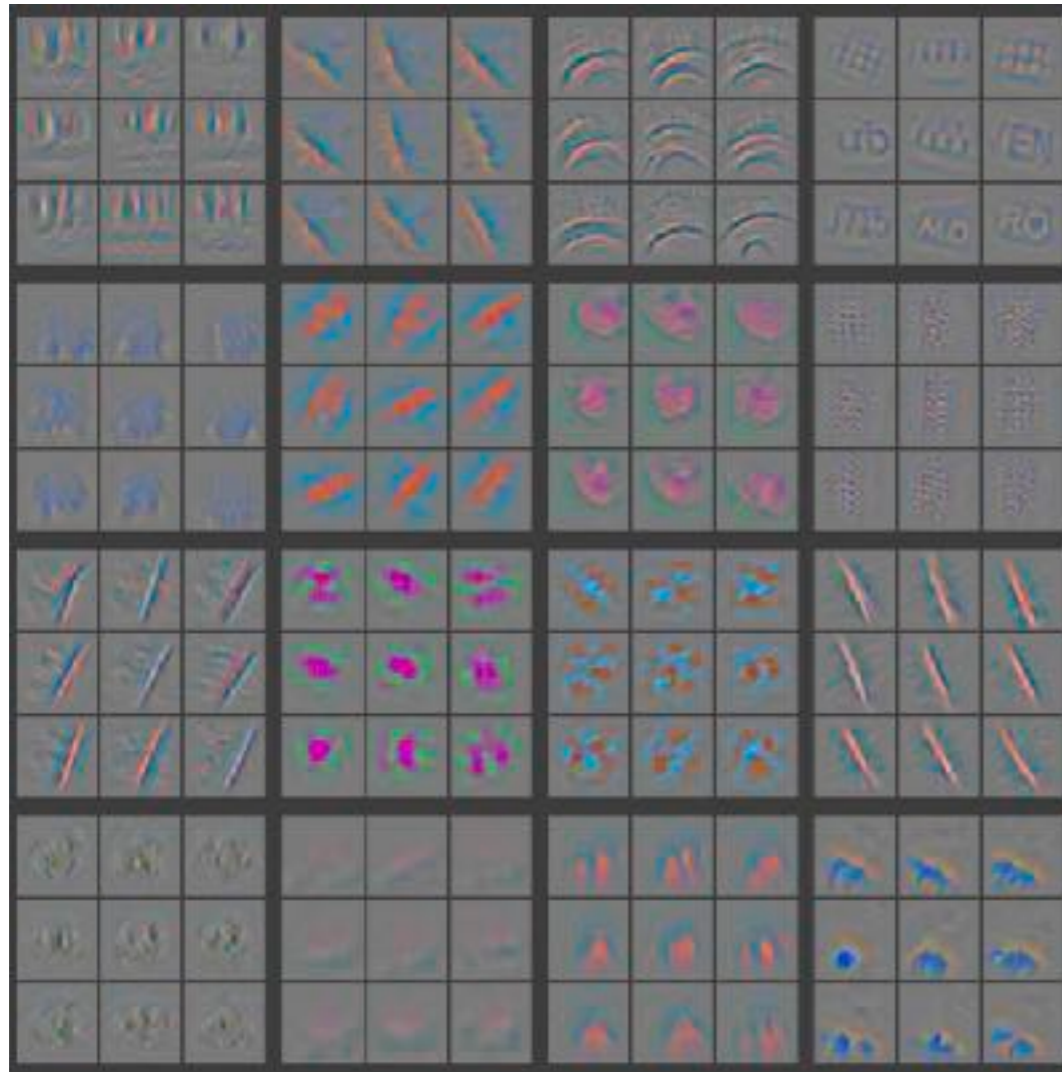
http://cs.nyu.edu/~fergus/presentations/nips2013_final.pdf

Visualizing CNN (Layer 1)

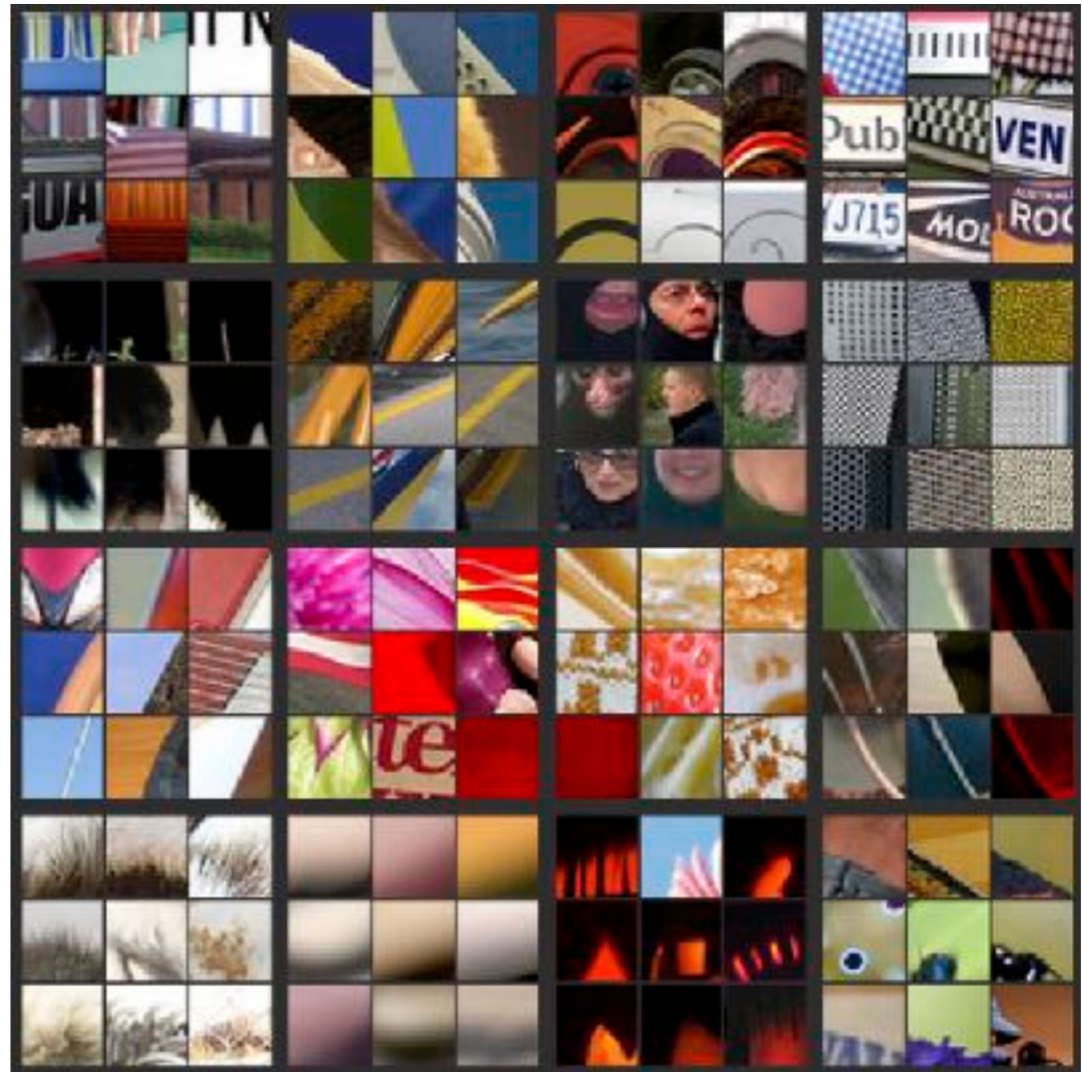


<http://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf>
http://cs.nyu.edu/~fergus/presentations/nips2013_final.pdf

Visualizing CNN (Layer 2)



Part that Triggered Filter

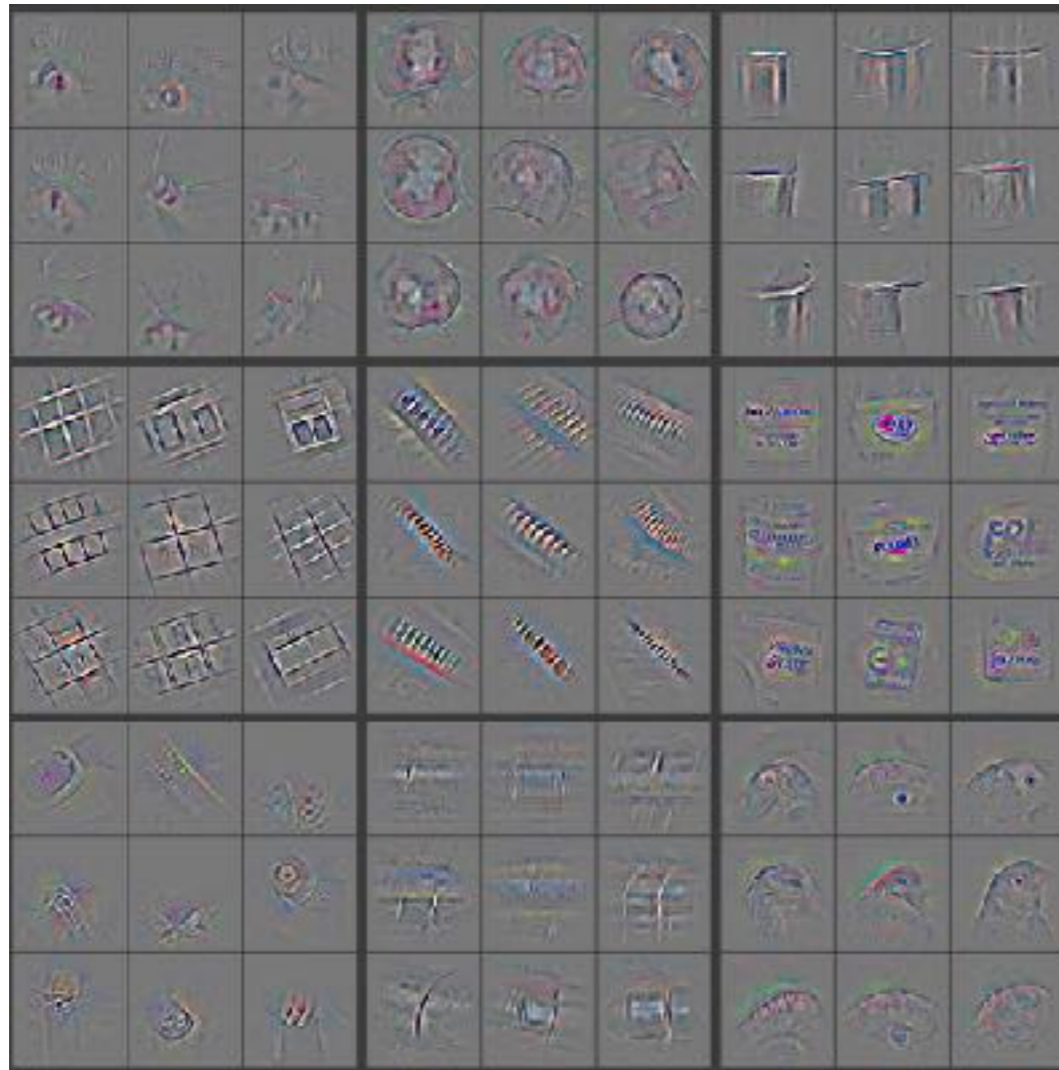


Top Image Patches

<http://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf>

http://cs.nyu.edu/~fergus/presentations/nips2013_final.pdf

Visualizing CNN (Layer 3)



Part that Triggered Filter

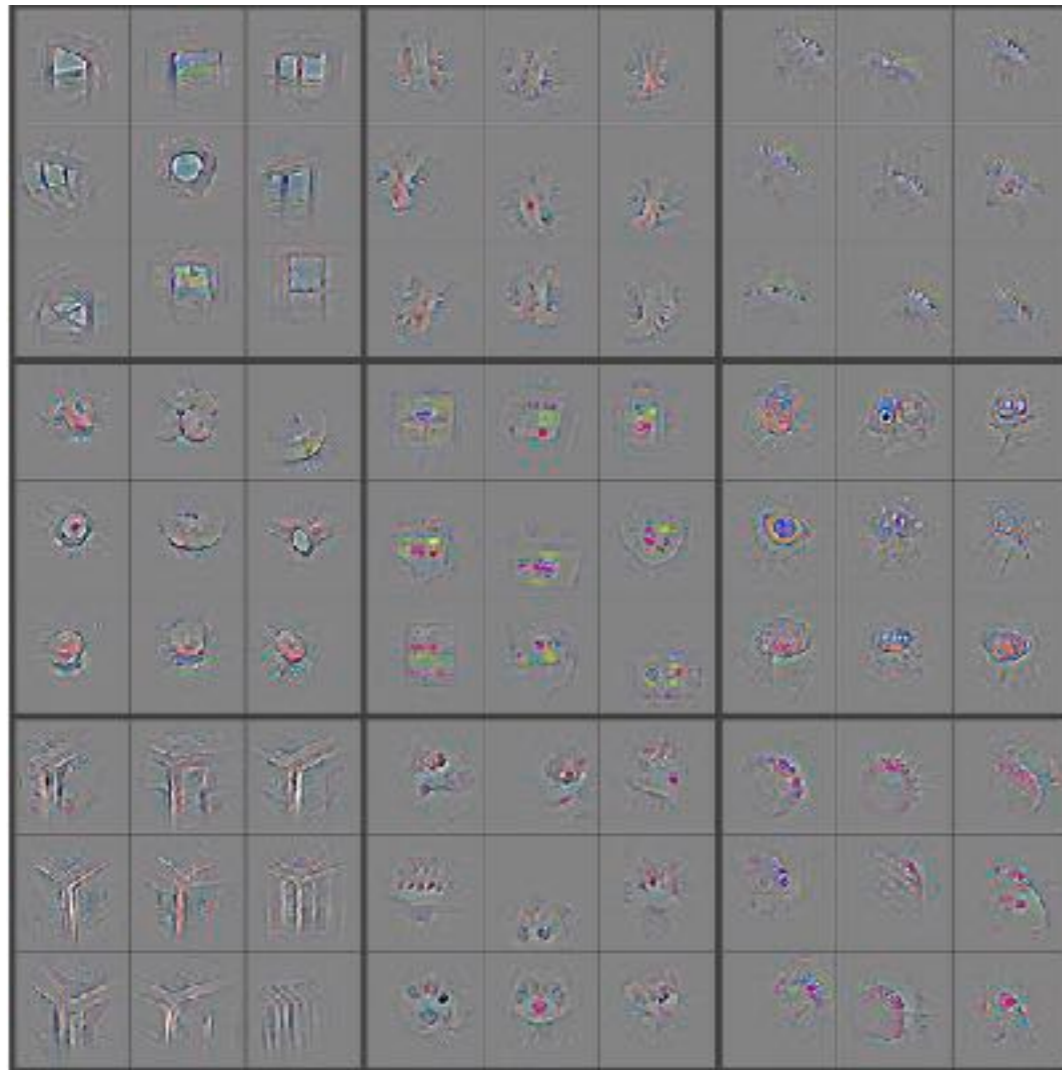


Top Image Patches

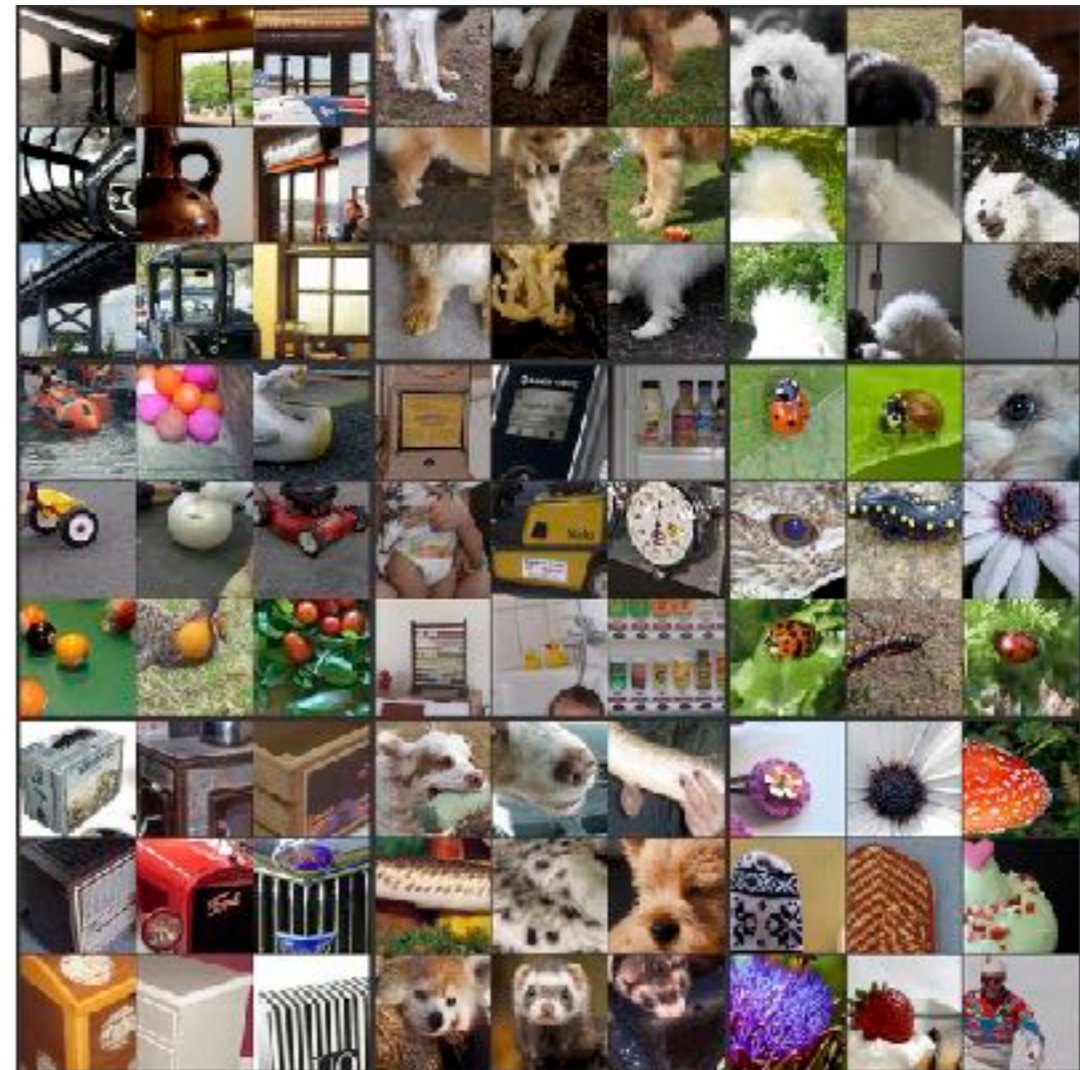
<http://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf>

http://cs.nyu.edu/~fergus/presentations/nips2013_final.pdf

Visualizing CNN (Layer 4)



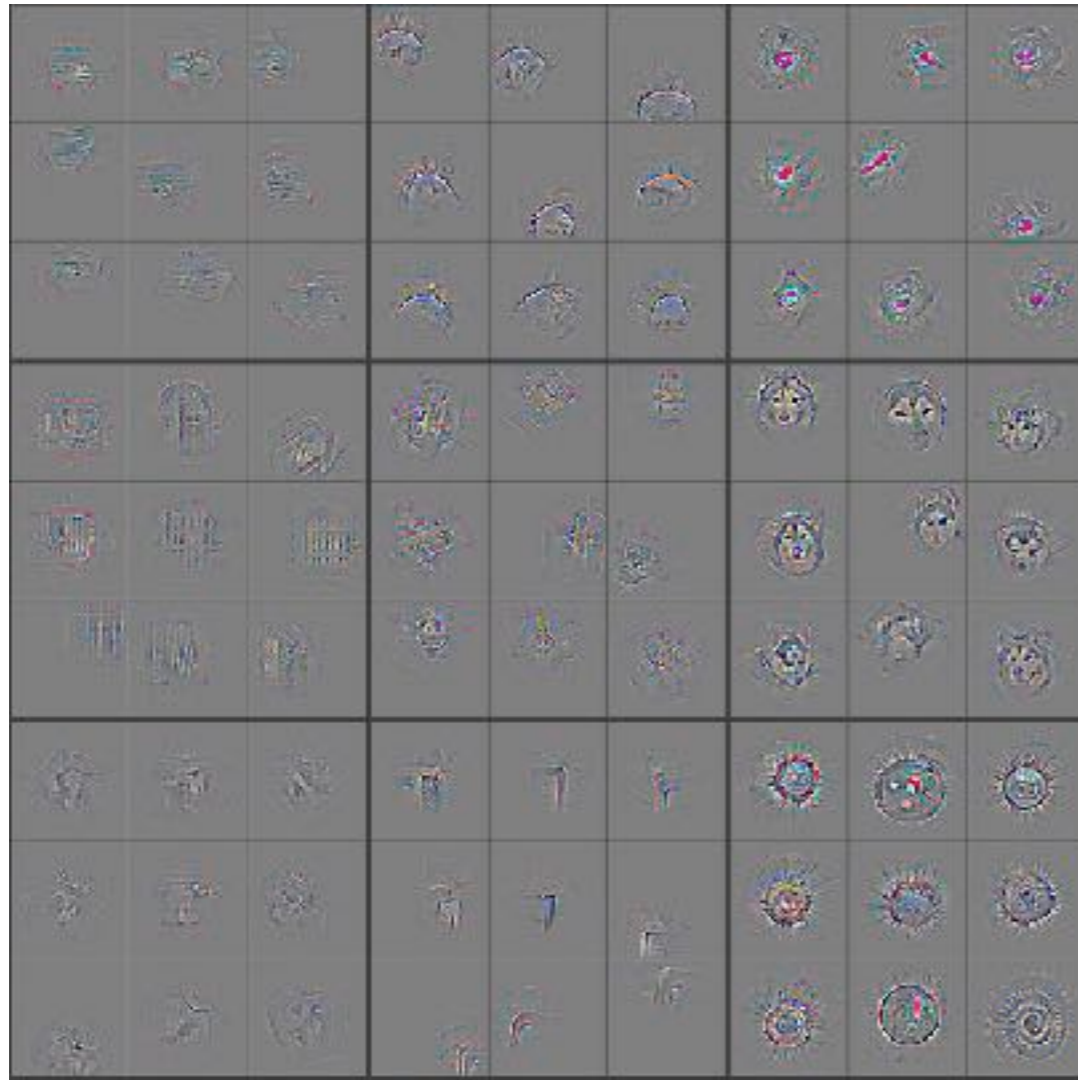
Part that Triggered Filter



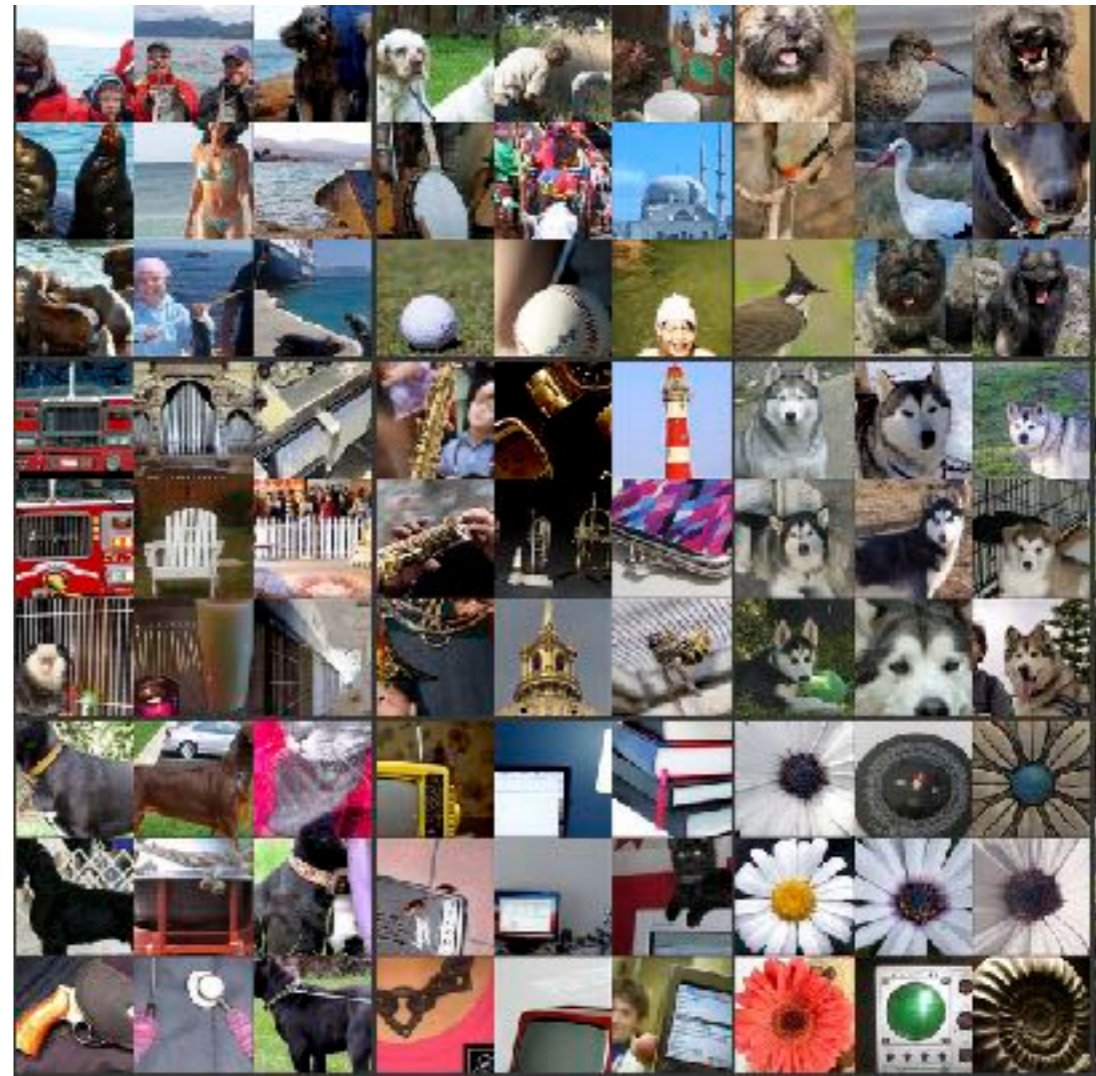
Top Image Patches

<http://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf>
http://cs.nyu.edu/~fergus/presentations/nips2013_final.pdf

Visualizing CNN (Layer 5)



Part that Triggered Filter



Top Image Patches

<http://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf>

http://cs.nyu.edu/~fergus/presentations/nips2013_final.pdf

Tips and Tricks

- Shuffle the training samples
- Use Dropout and Batch Normalization for regularization

Input representation

“Given a rectangular image, we first rescaled the image such that the shorter side was of length 256, and then cropped out the central 256×256 patch from the resulting image”

- Centered (0-mean) RGB values.



An input image (256x256)



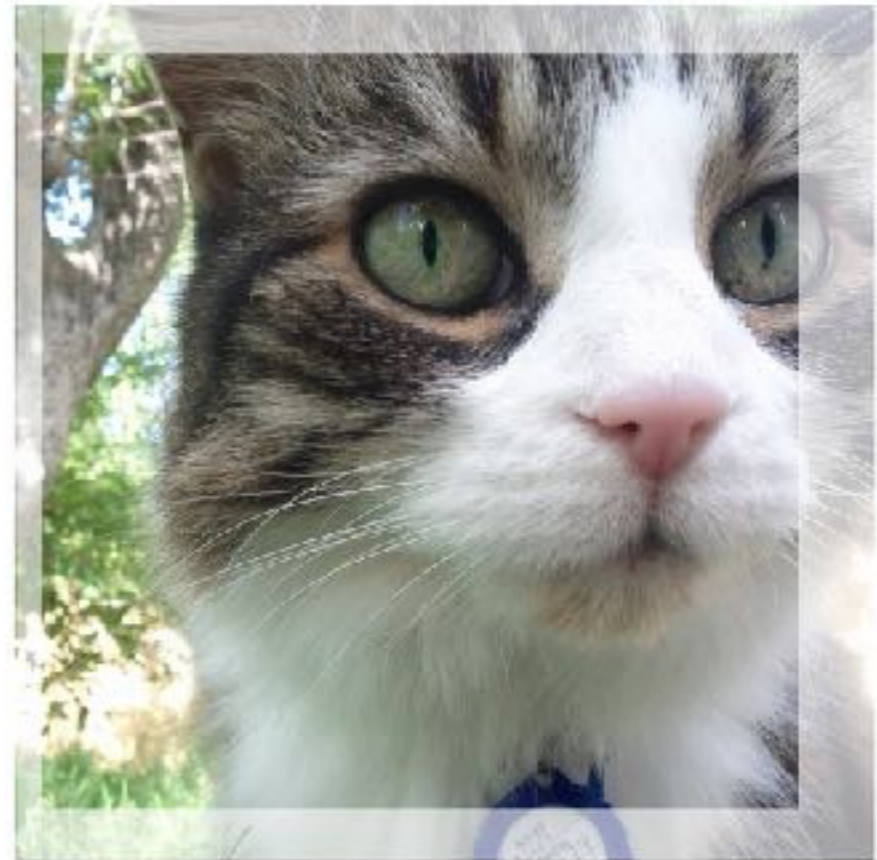
Minus sign



The mean input image

Data Augmentation

- The neural net has 60M real-valued parameters and 650,000 neurons
- It overfits a lot. Therefore, they train on 224×224 patches extracted randomly from 256×256 images, and also their horizontal reflections.



“This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly inter- dependent.”

[Krizhevsky et al. 2012]

Data Augmentation

- Alter the intensities of the RGB channels in training images.

“Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components, with magnitudes proportional to the corresponding eigenvalues times a random variable drawn from a Gaussian with mean zero and standard deviation 0.1... This scheme approximately captures an important property of natural images, namely, that object identity is invariant to changes in the intensity and color of the illumination. This scheme reduces the top-1 error rate by over 1%.”



[Krizhevsky et al. 2012]

Data Augmentation

Horizontal flips



Data Augmentation

Get creative!

Random mix/combinations of :

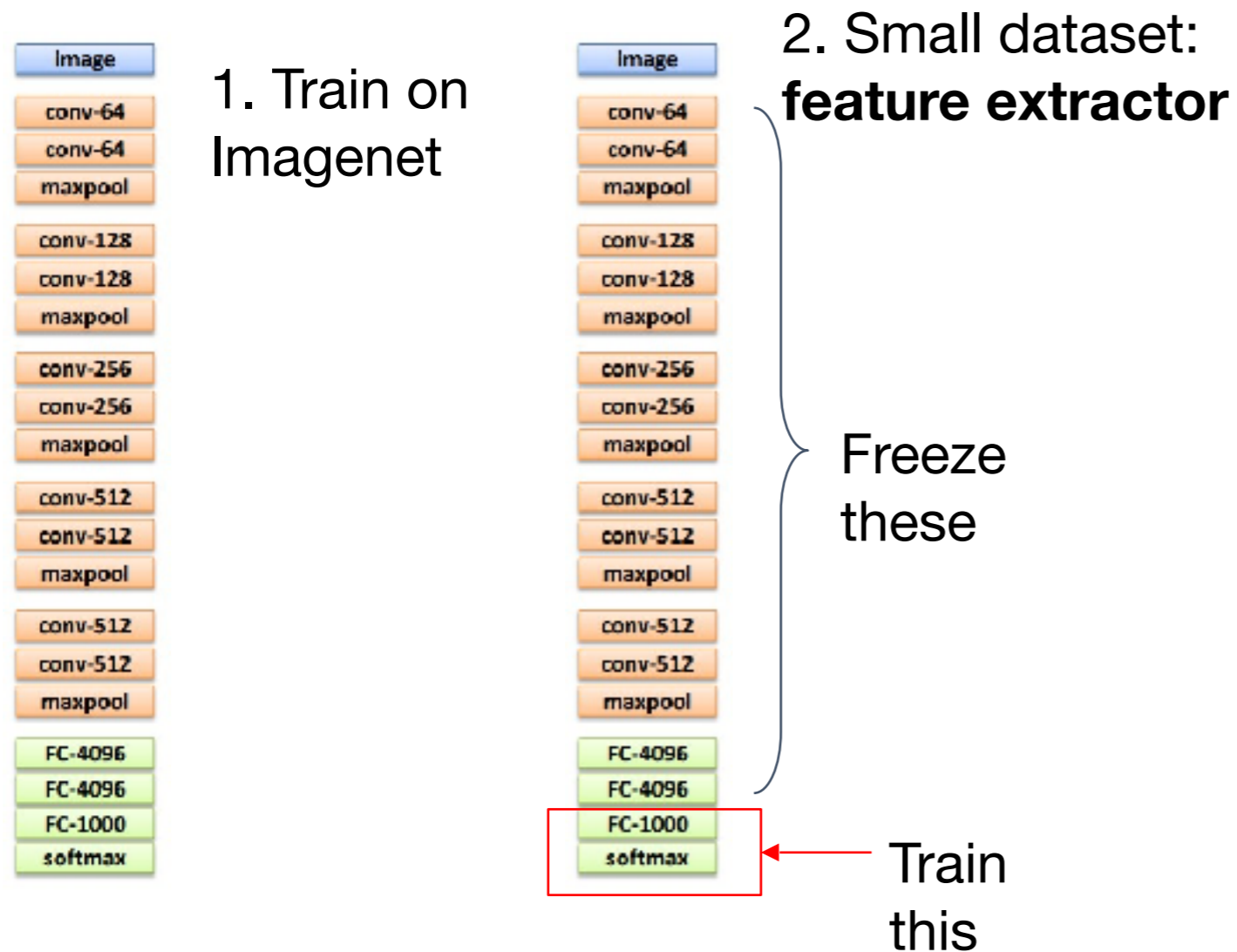
- translation
- rotation
- stretching
- shearing,
- lens distortions, ... (go crazy)

Transfer Learning with ConvNets

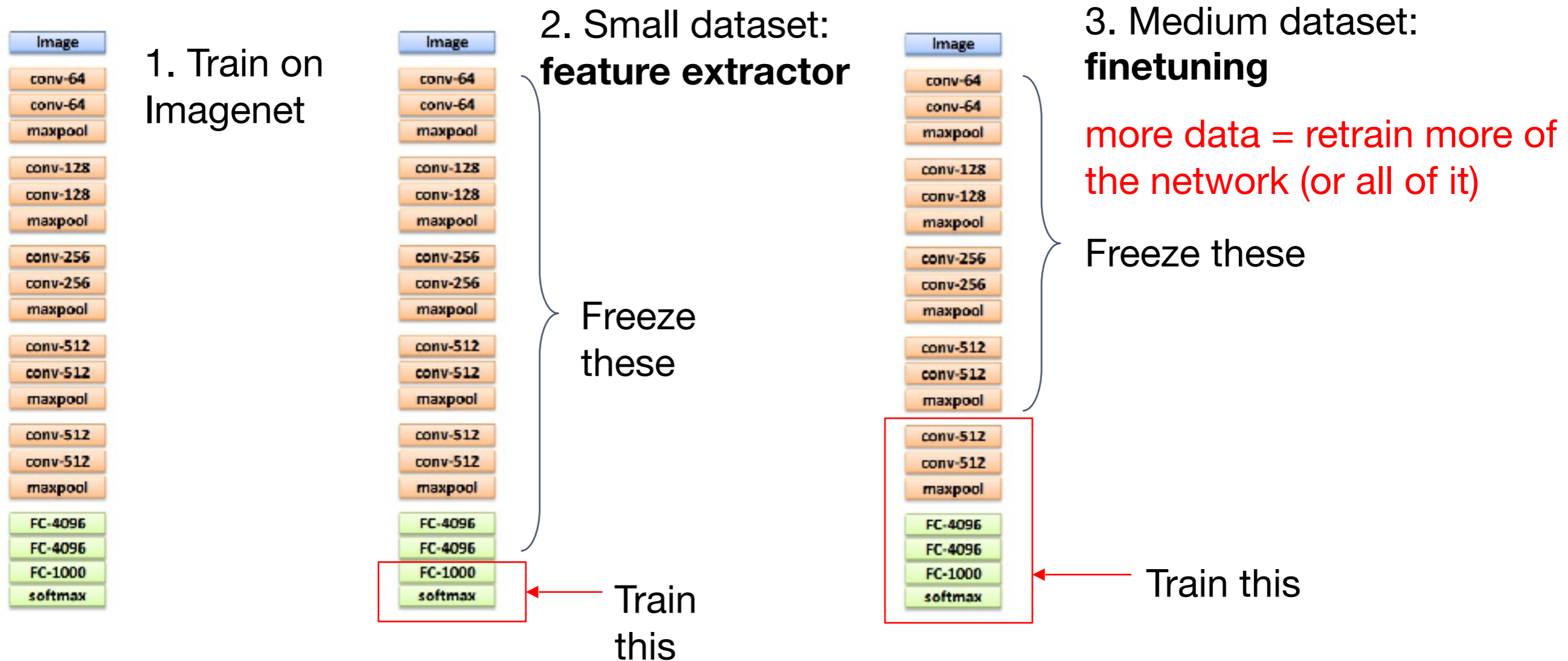


1. Train on
Imagenet

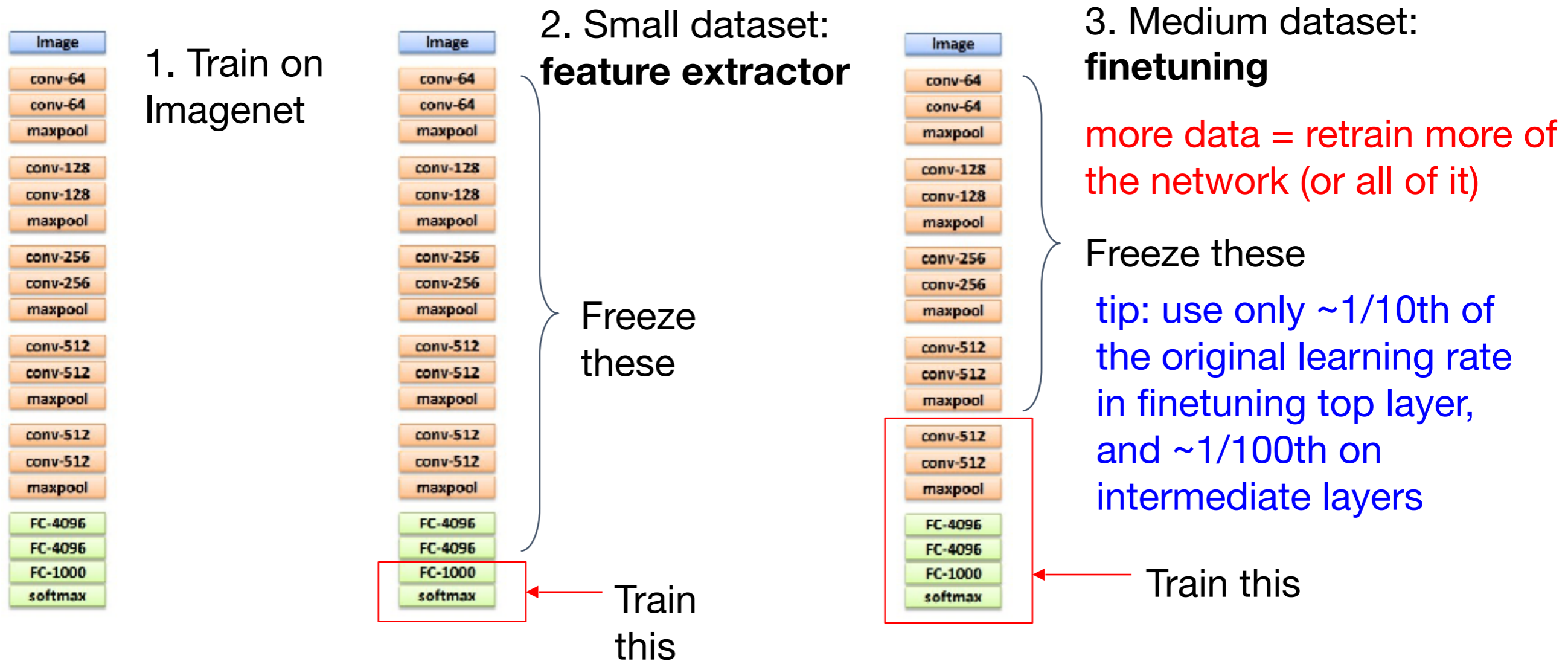
Transfer Learning with ConvNets



Transfer Learning with ConvNets











Transfer Learning with ConvNets

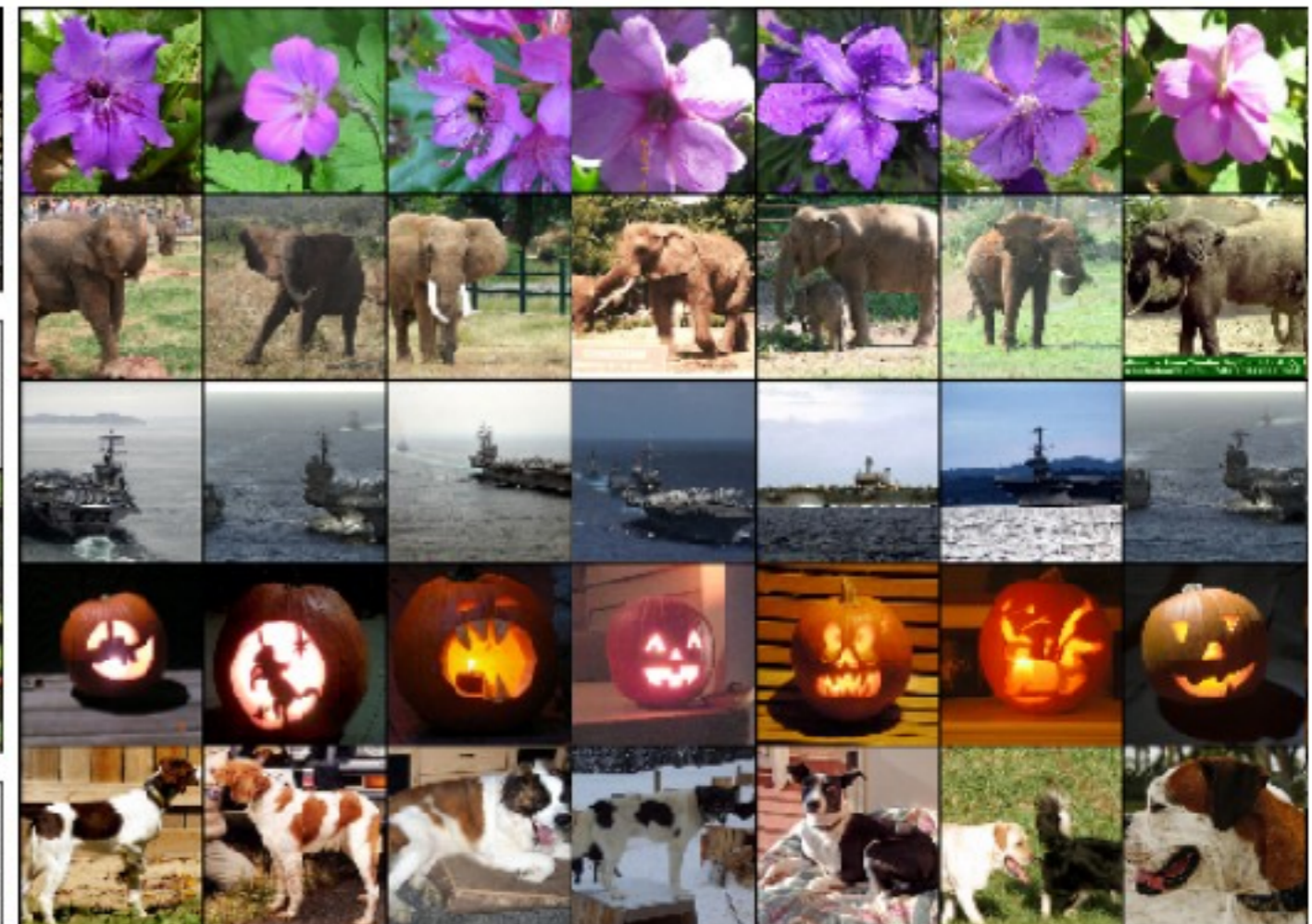


Today ConvNets are everywhere

Classification

			
mite	container ship	motor scooter	leopard
<ul style="list-style-type: none"> mite black widow cockroach tick starfish 	<ul style="list-style-type: none"> container ship lifeboat amphibian fireboat drilling platform 	<ul style="list-style-type: none"> motor scooter go-kart moped bumper car golfcart 	<ul style="list-style-type: none"> leopard jaguar cheetah snow leopard Egyptian cat
			
grille	mushroom	cherry	Madagascar cat
<ul style="list-style-type: none"> convertible grille pickup heach wagon fire engine 	<ul style="list-style-type: none"> agaric mushroom jelly fungus gill fungus dead-man's-fingers 	<ul style="list-style-type: none"> dalmatian grape elderberry ffordshire bullterrier currant 	<ul style="list-style-type: none"> squirrel monkey spider monkey titi indri howler monkey

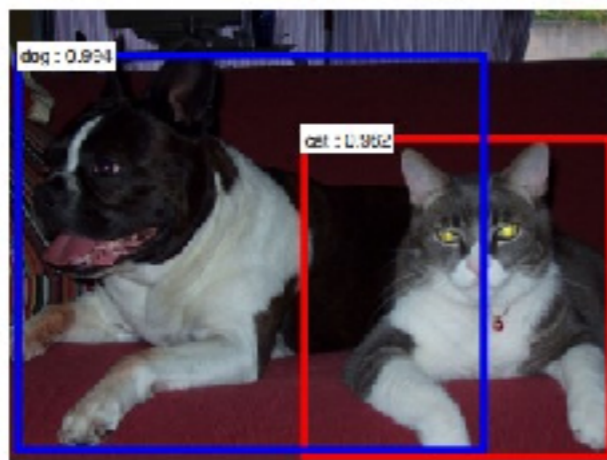
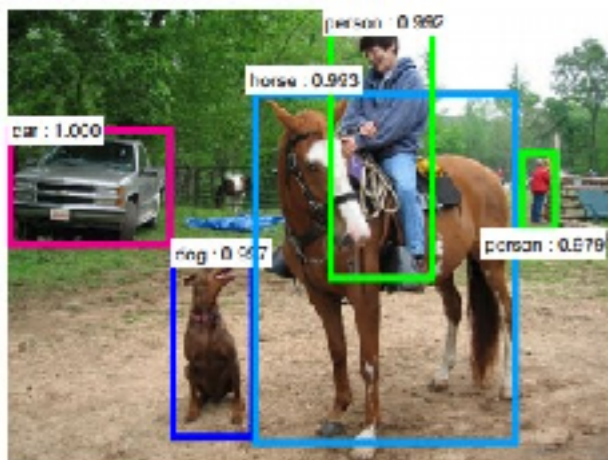
Retrieval



[Krizhevsky 2012]

Today ConvNets are everywhere

Detection



Segmentation



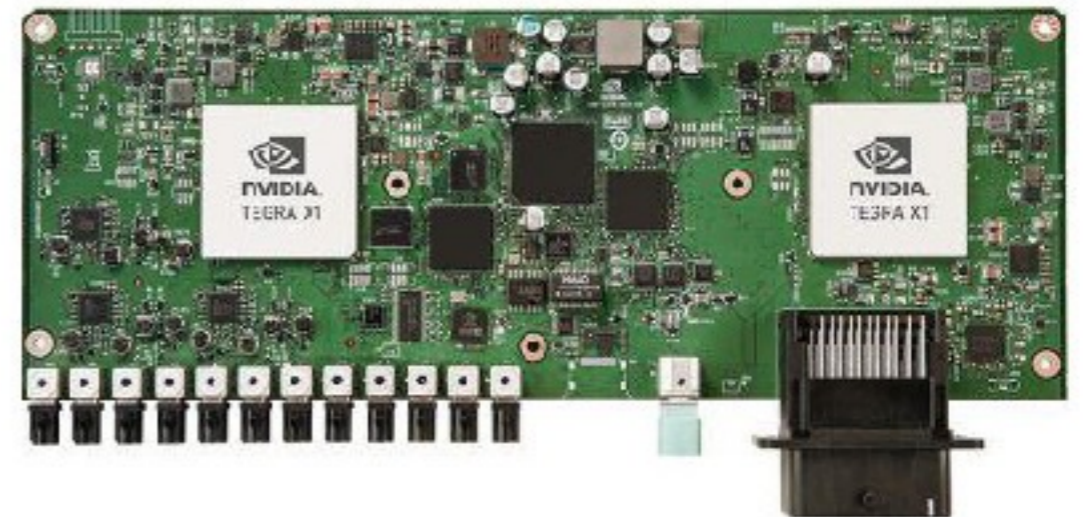
[Faster R-CNN: Ren, He, Girshick, Sun 2015]

[Farabet et al., 2012]

Today ConvNets are everywhere

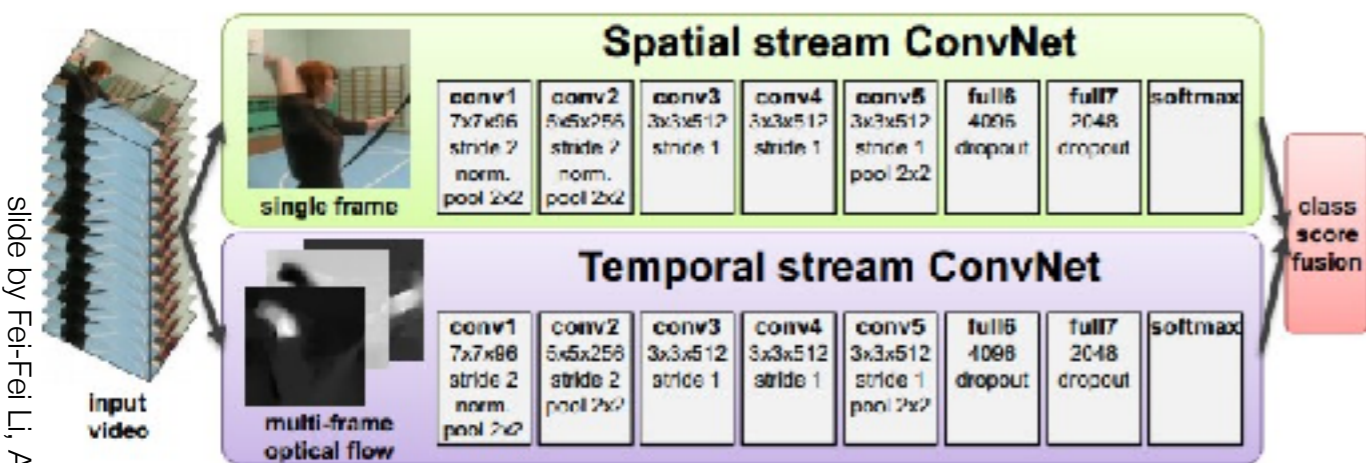
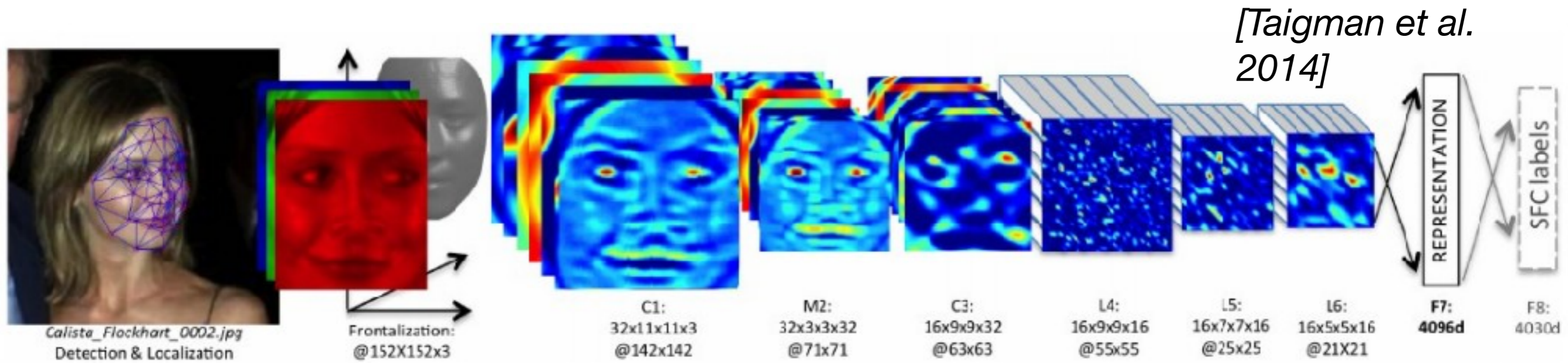


self-driving cars



NVIDIA Tegra X1

Today ConvNets are everywhere



[Simonyan et al. 2014]



[Goodfellow 2014]

Today ConvNets are everywhere

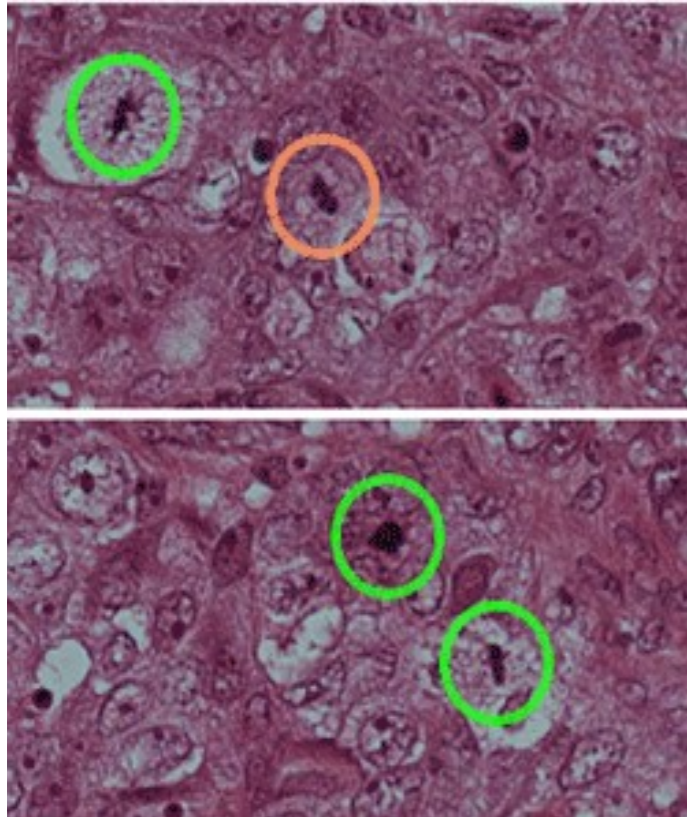


[Toshev, Szegedy 2014]



[Mnih 2013]

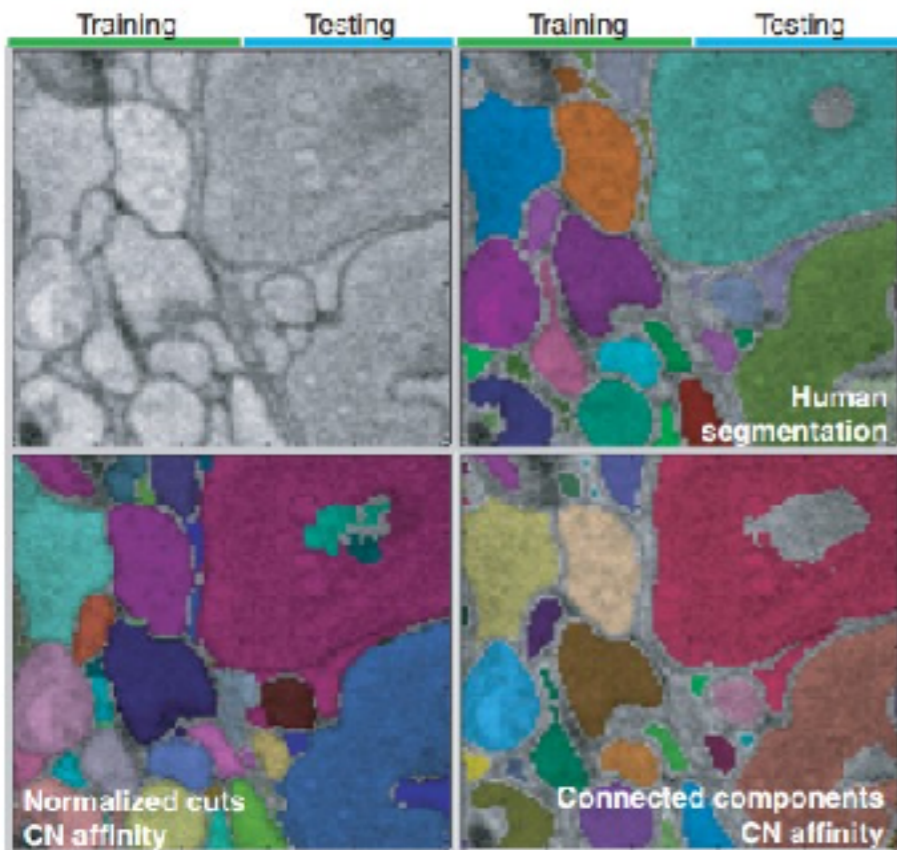
Today ConvNets are everywhere



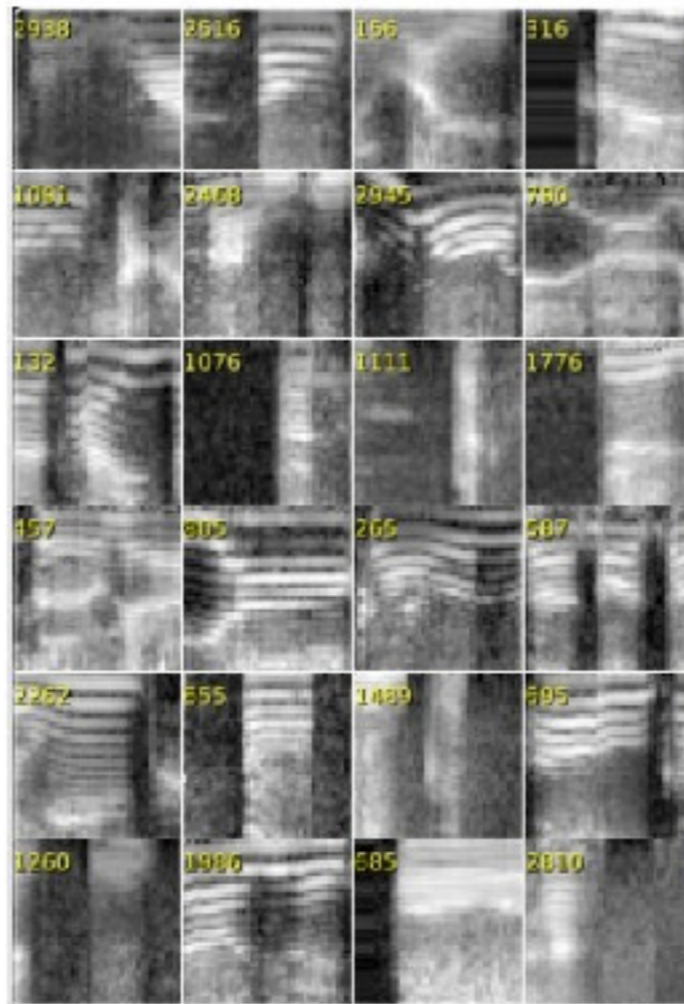
[Ciresan et al. 2013]

[Sermanet et al. 2011]
[Ciresan et al.]

Today ConvNets are everywhere



[Turaga et al., 2010]



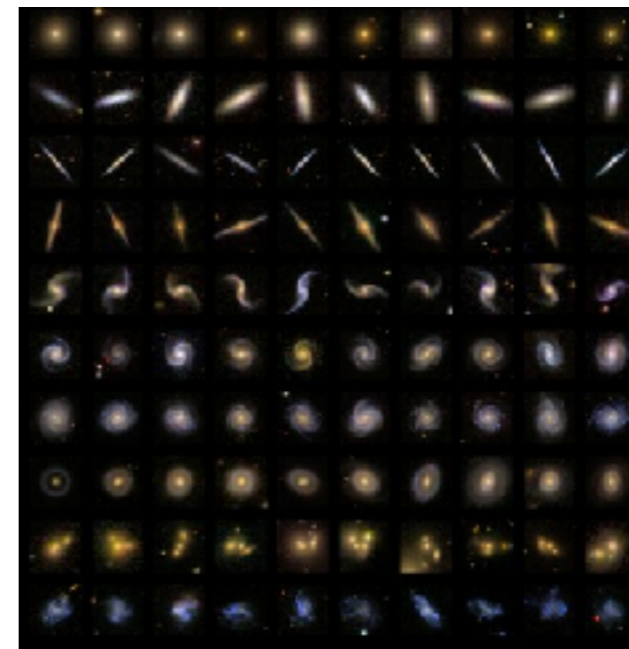
I caught this movie on the Sci-Fi channel recently. I actually turned out to be pretty decent as far as B-list horror/suspense films go. Two guys (one naive and one loud mouthed a **) take a road trip to stop a wedding but have the worst possible luck when a maniac in a fraky, make-shift tank/truck hybrid decides to play cat-and-mouse with them. Things are further complicated when they pick up a ridiculously whorish hitchhiker. What makes this film unique is that the combination of comedy and terror actually work in this movie, unlike so many others. The two guys are likable enough and there are some good chase/suspense scenes. Nice pacing and comic timing make this movie more than passable for the home/dvdster buff. **Definitely worth checking out.**

I just saw this on a local independent station in the New York City area. The cast showed promise but when I saw the director, George Cosmatos, I became suspicious. And sure enough, it was every bit as bad, every bit as pointless and stupid as every George Cosmatos movie I ever saw. He's like a stupid man's Michael Bay - with all the awfulness that accolade promises. There's no point to the conspiracy, no burning issues that urge the conspirator-on. We are left to ourselves to connect the dots from one bit of gratuitous various walks in the film to the next. Thus, the current budget crisis, the war in Iraq, Islamic extension, the fate of social security, 47 million Americans without health care, stagnating wages, and the death of the middle class are all subsumed by the sheer terror of graffiti. A truly, stunningly idiotic film.

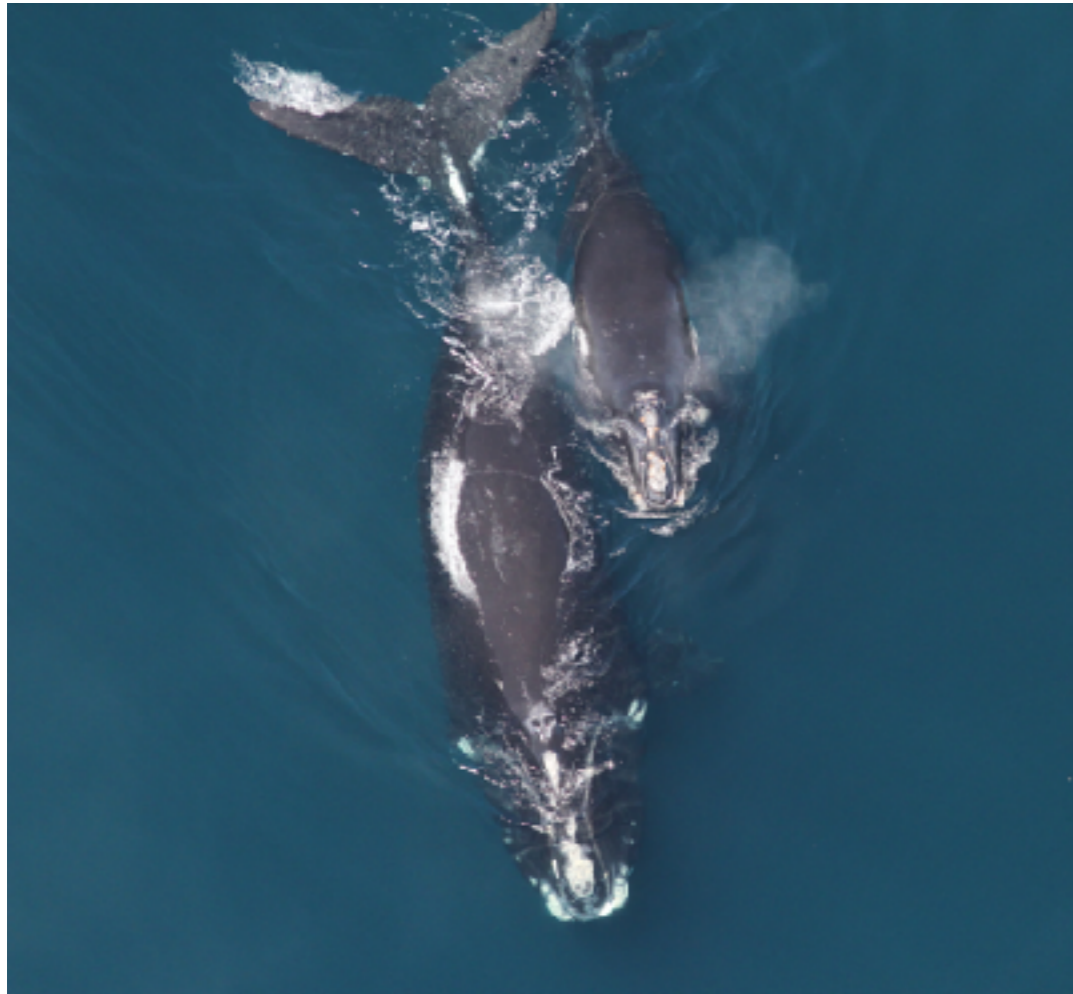
Graphics is far from the best part of the game. **This is the number one best THX game in the series.** Next to Underground. **It deserves strong love. It is an insane game.** There are massive levels, massive unlockable characters... it's just a massive game. **Waste your money on this game. This is the kind of money that is wasted properly.** And even though graphics suck, that doesn't make a game good. Actually, the graphics were good at the time. Today the graphics are crap. WOO C/RES! As they say in Canada, This is the fun game, aye. (You get to go to Canada in THPS!) Well, I don't know if they say that, but they might. who knows. Well, Canadian people do. Wait aminate, I'm getting off topic. This game rocks. Buy it, play it, enjoy it, love it. It's PURE BRILLIANCE.

The first was good and original. I was a not bad horror/comedy movie. So I heard a second one was made and I had to watch it. What really makes this movie work is Judd Nelson's character and the sometimes clever script. **A pretty good script for a person who wrote the Final Destination films and the direction was okay.** Sometimes there's scenes where it looks like it was filmed using a home video camera with a prissy -look. **Great made - for - TV movie. It was worth the rental and probably worth trying just to get that nice eerie feeling and watch Judd Nelson's Stanley doing what he does best.** I suggest newcomers to watch the first one before watching the sequel, just so you'll have an idea what Stanley is like and get a little history background.

[Denil et al. 2014]



Today ConvNets are everywhere















Whale recognition, Kaggle Challenge



Mnih and Hinton, 2010

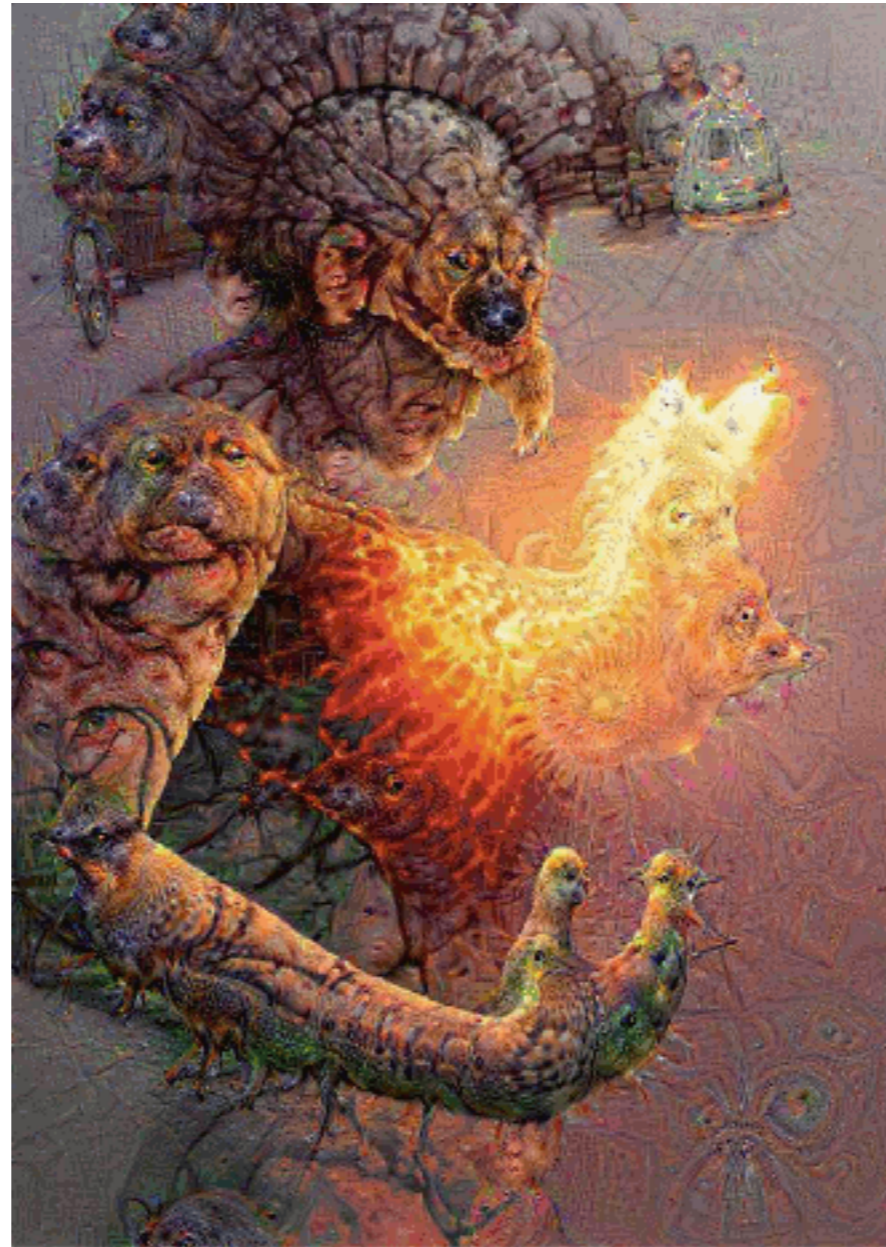
Today ConvNets are everywhere

Image Captioning

Describes without errors	Describes with minor errors	Somewhat related to the image	Unrelated to the image
 <p data-bbox="109 848 480 919">A person riding a motorcycle on a dirt road.</p>	 <p data-bbox="589 848 974 899">Two dogs play in the grass.</p>	 <p data-bbox="1111 848 1495 930">A skateboarder does a trick on a ramp.</p>	 <p data-bbox="1632 848 2016 930">A dog is jumping to catch a frisbee.</p>
 <p data-bbox="109 1257 480 1338">A group of young people playing a game of frisbee.</p>	 <p data-bbox="589 1257 1029 1328">Two hockey players are fighting over the puck.</p>	 <p data-bbox="1111 1257 1495 1338">A little girl in a pink hat is blowing bubbles.</p>	 <p data-bbox="1632 1277 2044 1359">A refrigerator filled with lots of food and drinks.</p>
 <p data-bbox="109 1676 507 1757">A herd of elephants walking across a dry grass field.</p>	 <p data-bbox="617 1676 974 1747">A close up of a cat laying on a couch.</p>	 <p data-bbox="1111 1686 1522 1757">A red motorcycle parked on the side of the road.</p>	 <p data-bbox="1632 1686 2044 1757">A yellow school bus parked in a parking lot.</p>

[Vinyals et al., 2015]

Today ConvNets are everywhere



[reddit.com/r/deepdream](https://www.reddit.com/r/deepdream)