Convolutional Neural Networks I

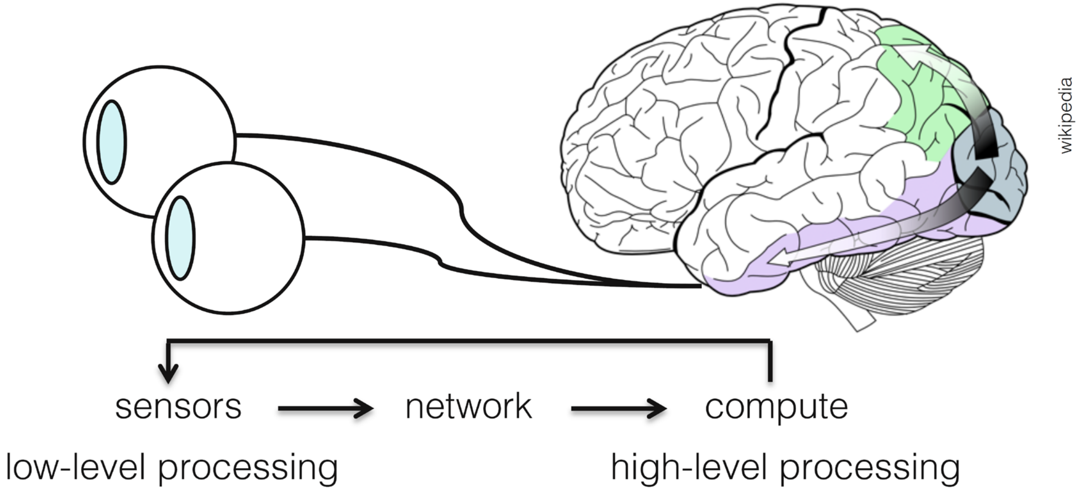
CS 6384 Computer Vision

Professor Yu Xiang

The University of Texas at Dallas

Some slides of this lecture are courtesy Stanford CS231n

Visual Perception vs. Computational Perception



Image

Neural Network

High-level information
- Depth
- Motion
- Object classes
- Object poses
- Etc.

Mathematic Models

- Try to model the human brain with computational models, e.g., neural networks

x

Image

$f(x)$

y

Object class

Mathematic Models

- What is the form of the function $f(x)$?
  - No idea!
  - Concatenate simple functions (neurons)

x

$$f(x)$$

$$y \in \{+1, -1\}$$

Dog

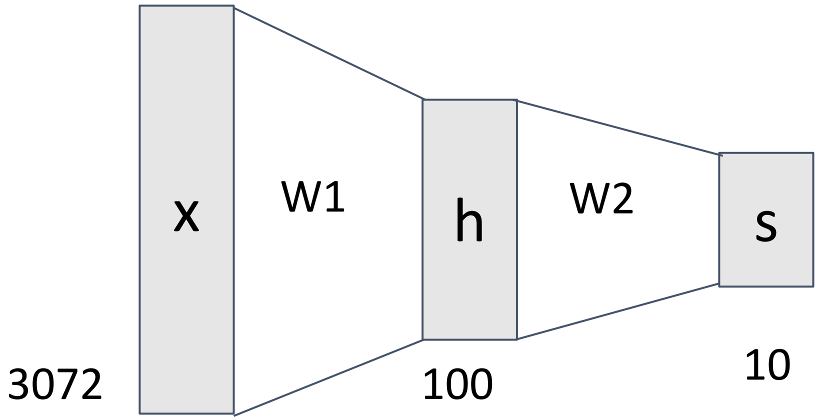
Neural Network: Concatenation of functions

Linear score function: $f = Wx$

2-layer Neural Network

$$f = f_2(f_1(x)) = W_2 \max(0, W_1 x)$$

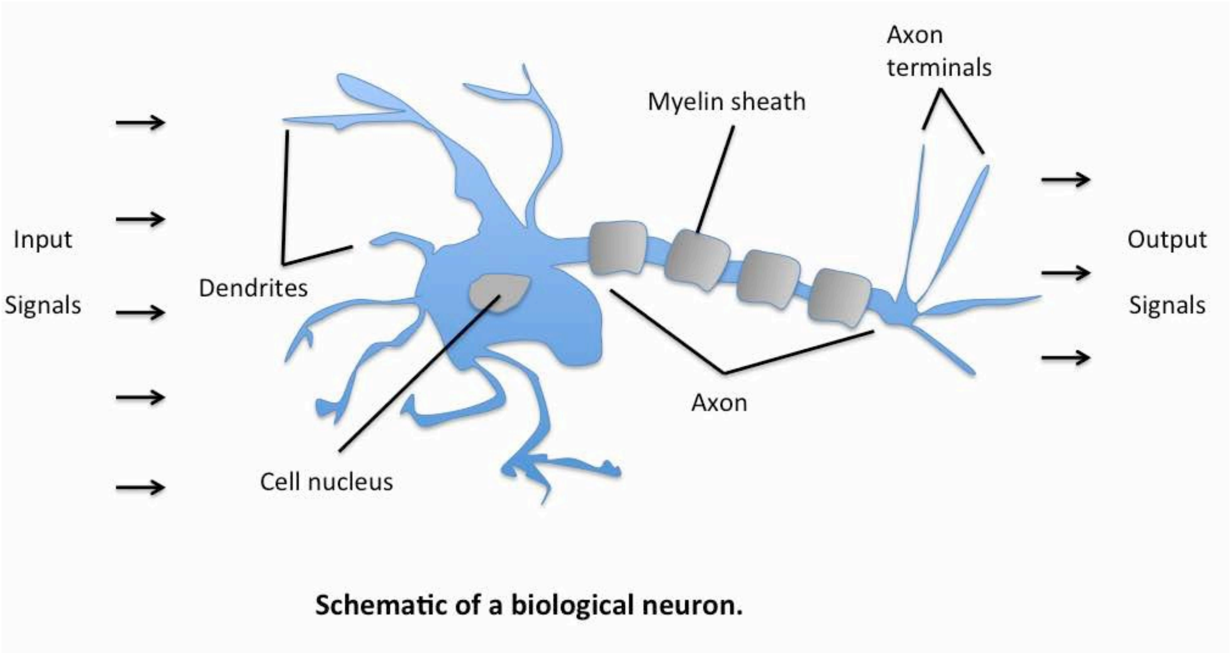
Non-linearity

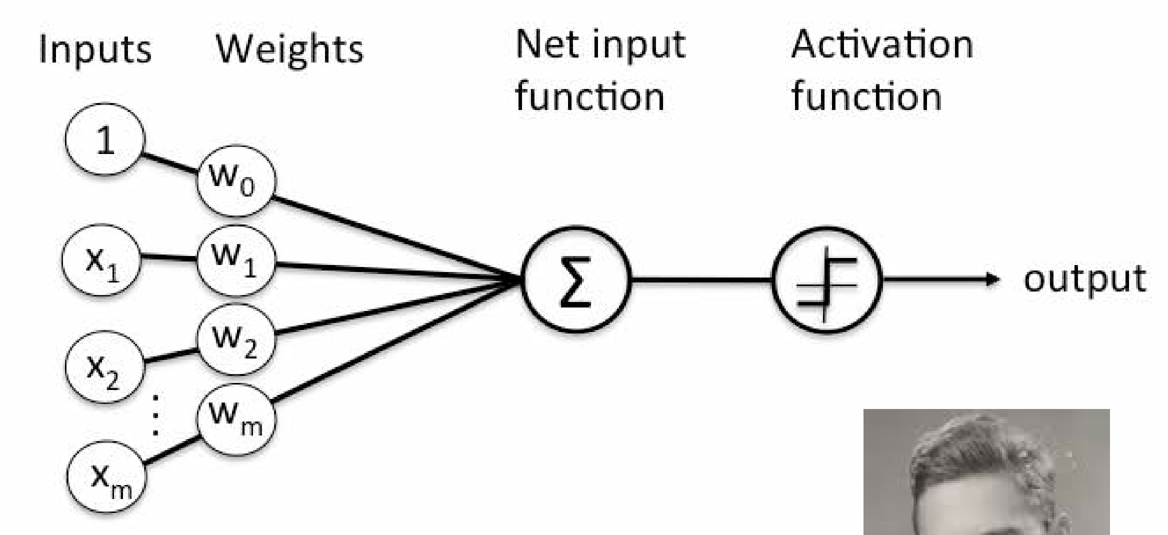


$$h = f_1(X)$$
$$s = f_2(h)$$

Need to learn the weights!

Frank Rosenblatt's Perceptron





$$\sigma(\mathbf{w}^T\mathbf{x}+b) = \begin{cases} 1 \text{ if } \mathbf{w}^T\mathbf{x}+b \geq 0, \\ 0 \text{ otherwise.} \end{cases}$$

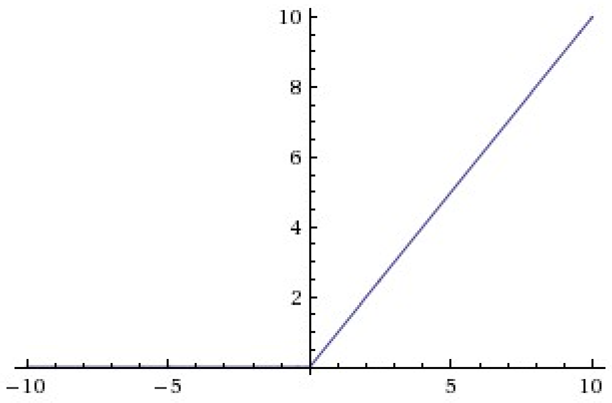
Frank Rosenblatt
(1928-1971)

Activation Functions

2-layer Neural Network

$$f = f_2(f_1(x)) = W_2 \max(0, W_1 x)$$

Rectified Linear Unit (ReLU)
max(0,x)

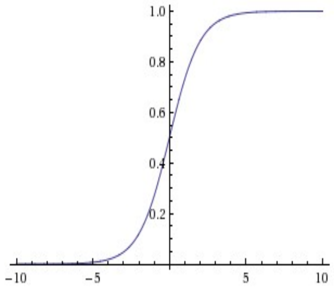


Introduce non-linearity to the network

Activation Functions

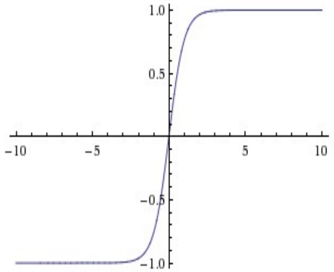
Sigmoid

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

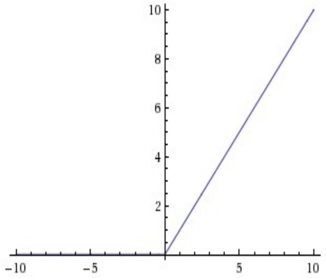


tanh tanh(x)

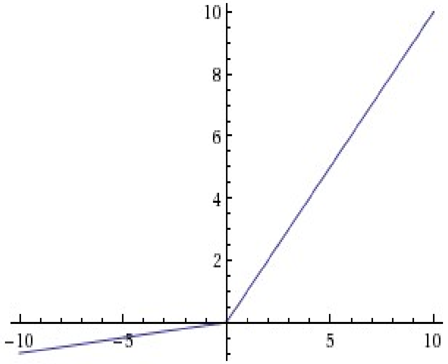
$$\frac{e^{2x}-1}{e^{2x}+1}$$



ReLU max(0,x)



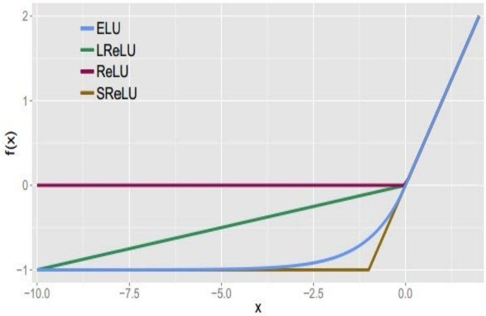
Leaky ReLU
max(0.1x, x)



Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU Exponential Linear Unit

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$$

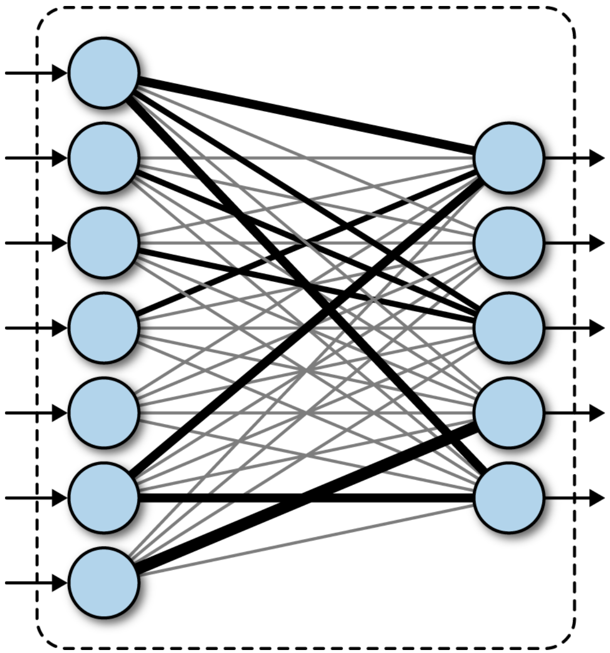


Fully Connected Layer

$$y = Wx$$

$x : n \times 1$ $\quad W : m \times n$ $\quad y : m \times 1$

$W : m \times n$



$x : n \times 1$ $\quad y : m \times 1$

Fully Connected Layer

- What is the drawback of only using fully connected layers?

$$y = Wx$$

- Consider an image with 640 x 480
  - x is with dimension 307,200
  - The weight matrix of the fully connect layer is too large

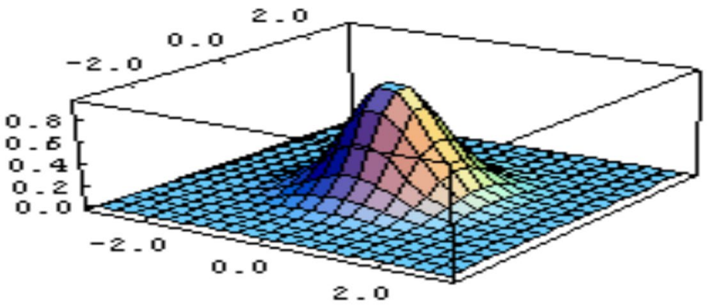
Convolutional Layers

- Consist of convolutional filters

- Share weights among different image locations

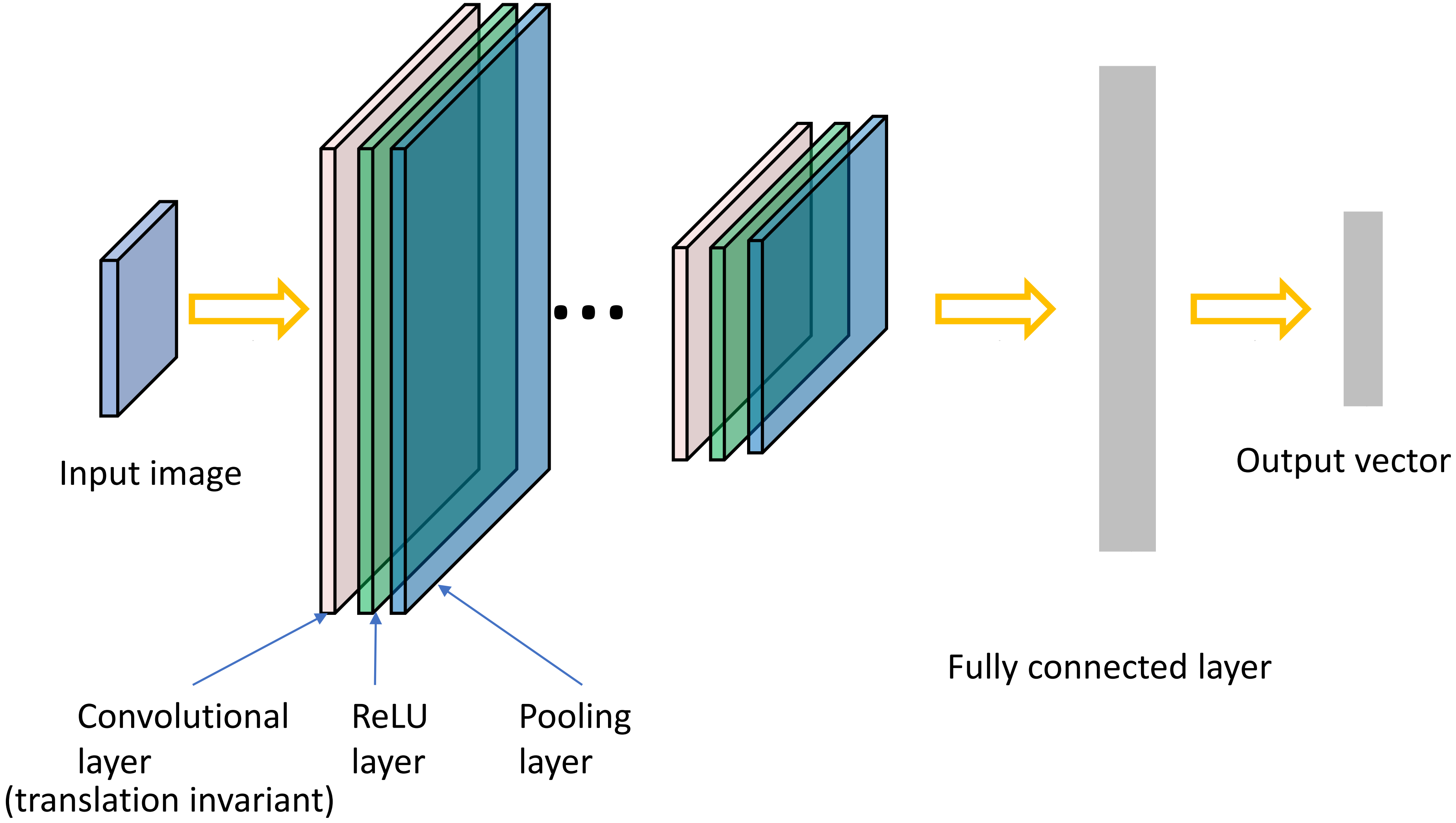
$$g(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Gaussian Filter

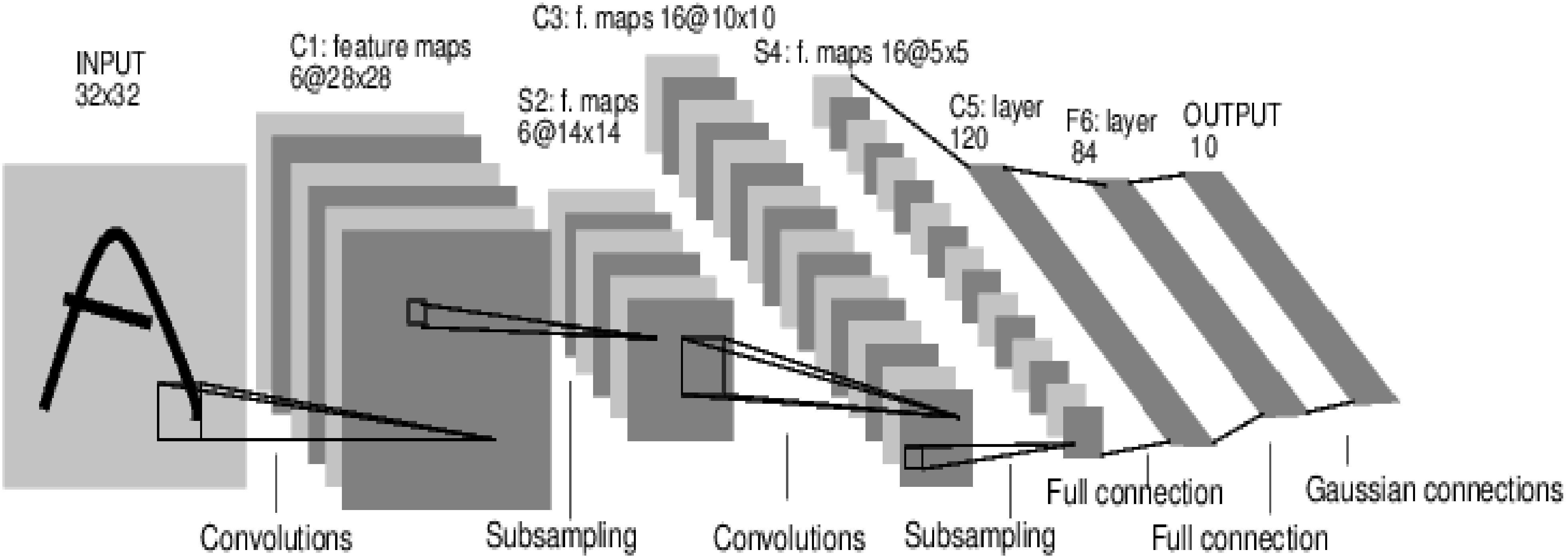


Learn the weights!

Convolutional Neural Networks



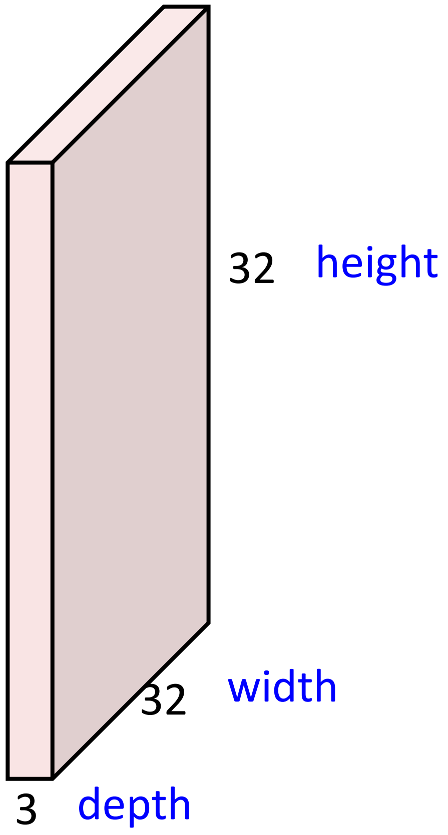
Convolutional Neural Networks



[LeNet-5, LeCun 1980]

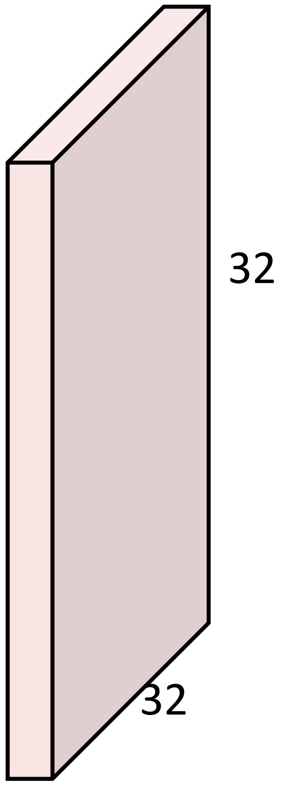
Convolutional Layer

32x32x3 image

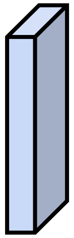


Convolutional Layer

32x32x3 image



5x5x3 filter

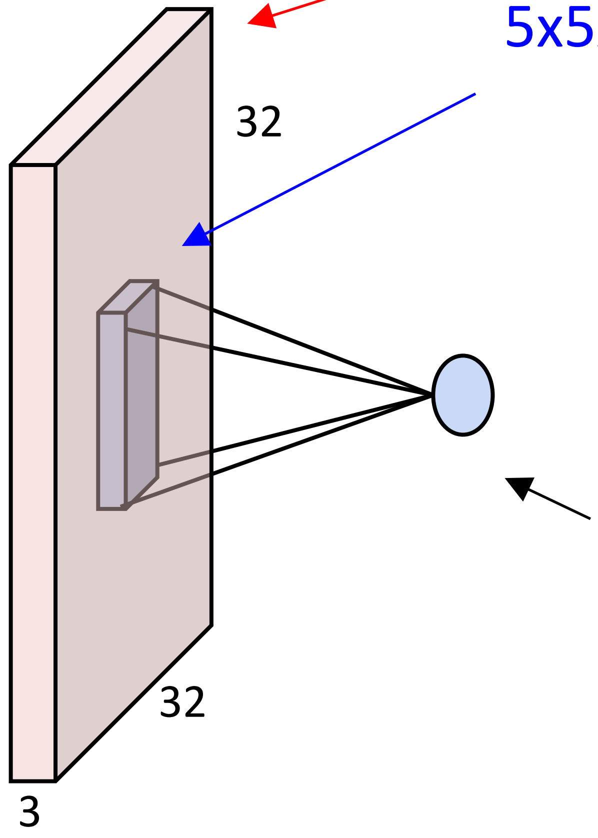


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolutional Layer

32x32x3 image

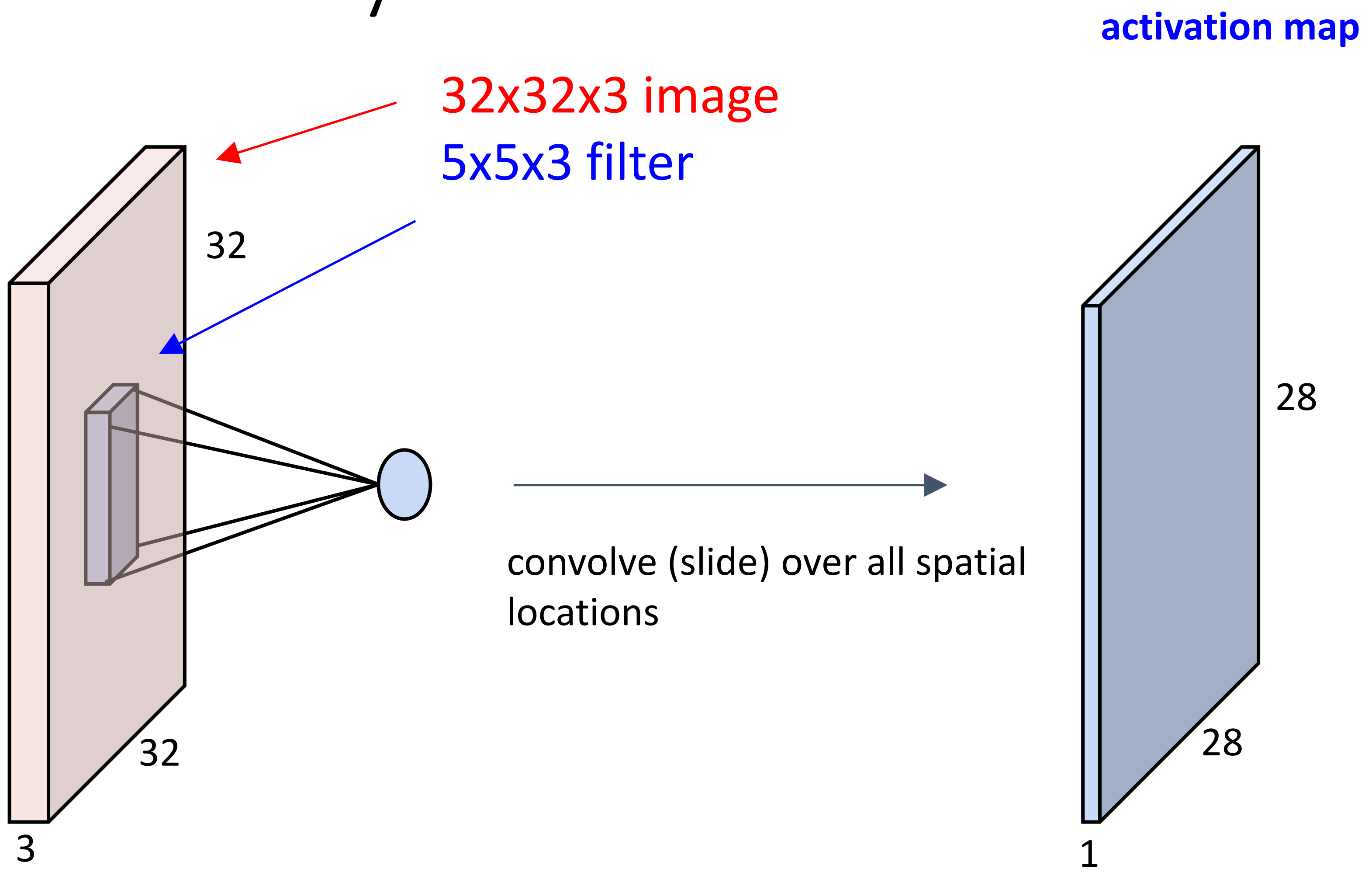
5x5x3 filter w



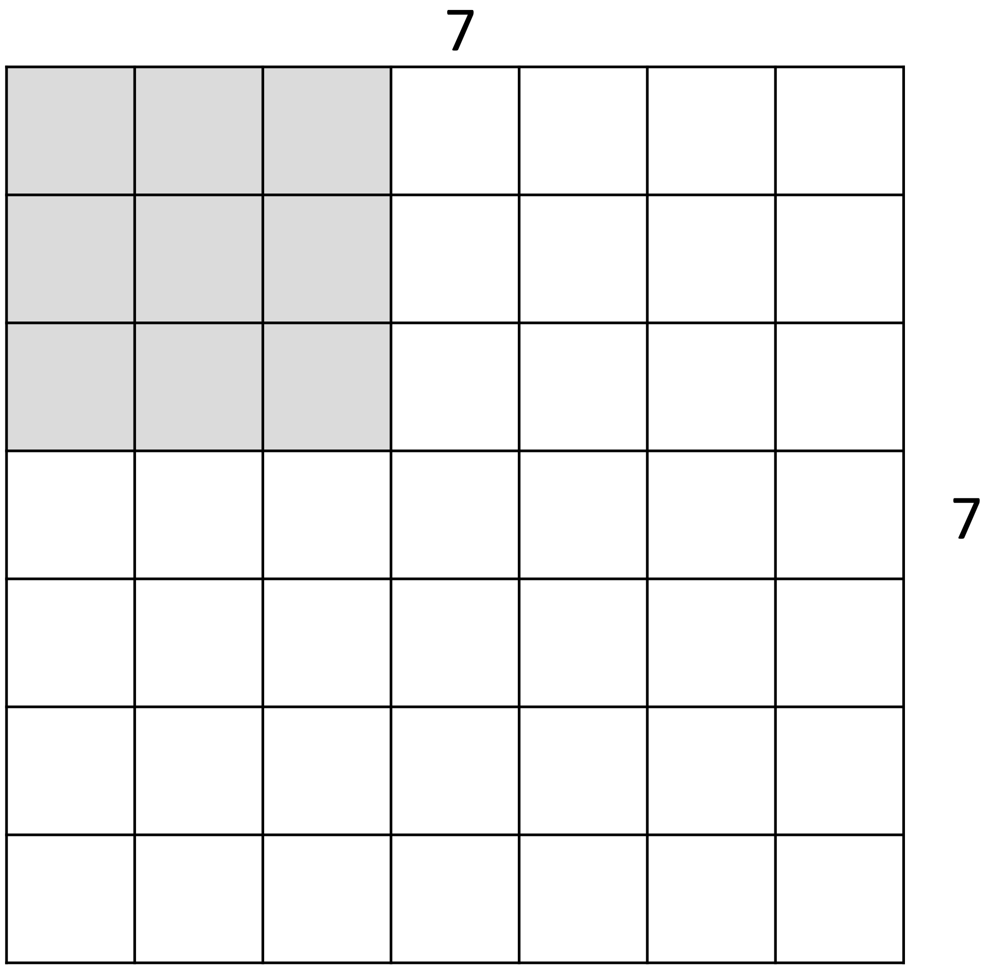
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolutional Layer



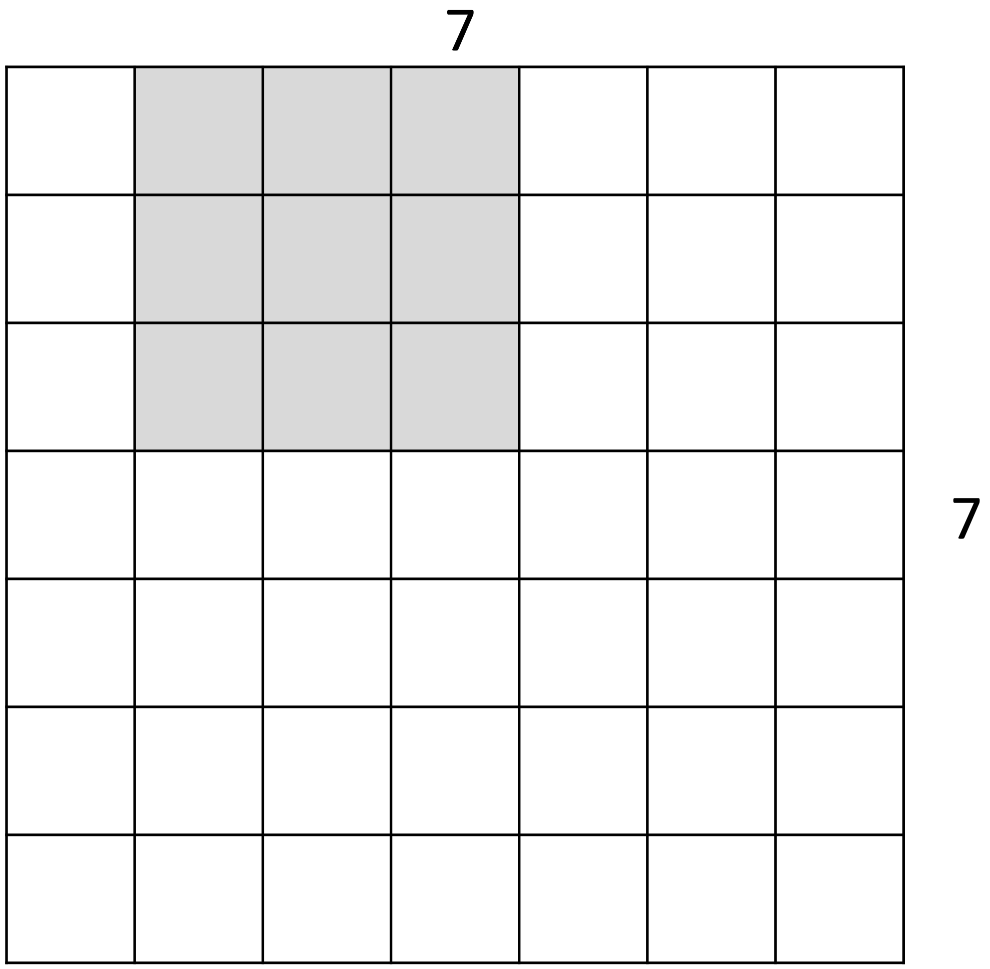
Convolutional Layer



A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 1

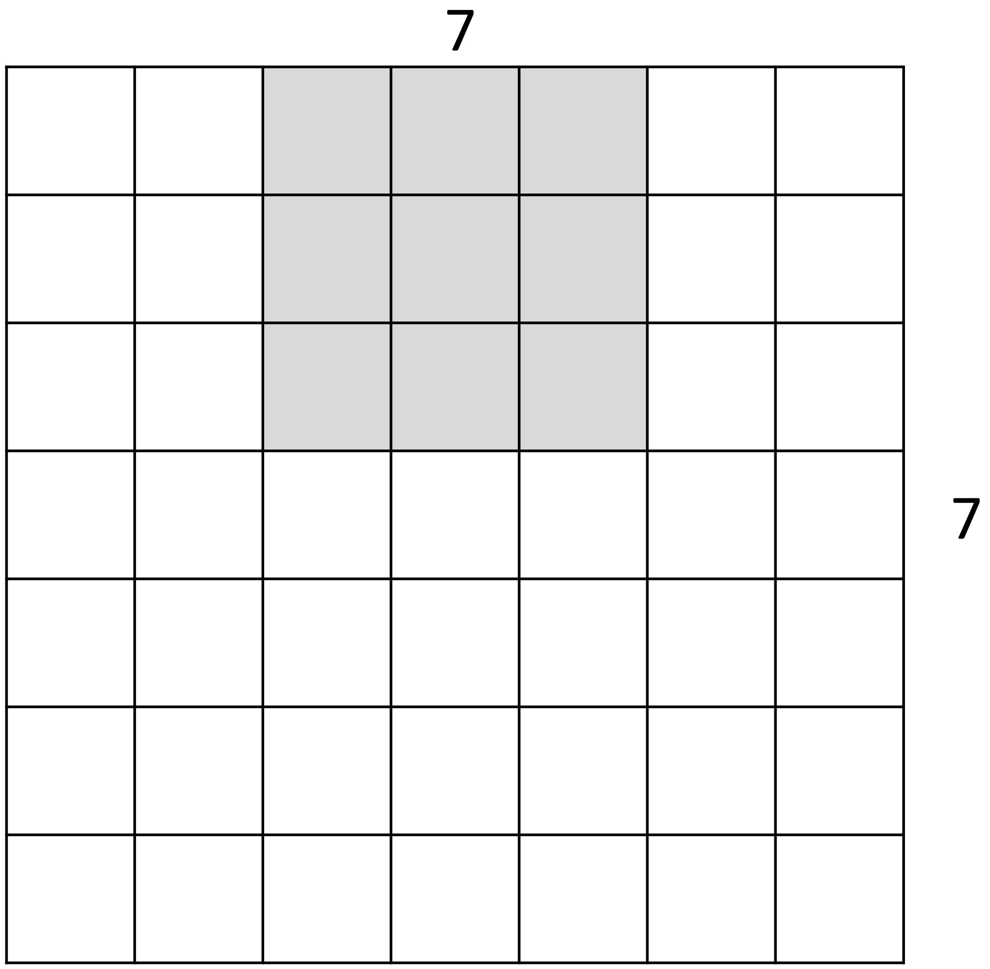
Convolutional Layer



A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 1

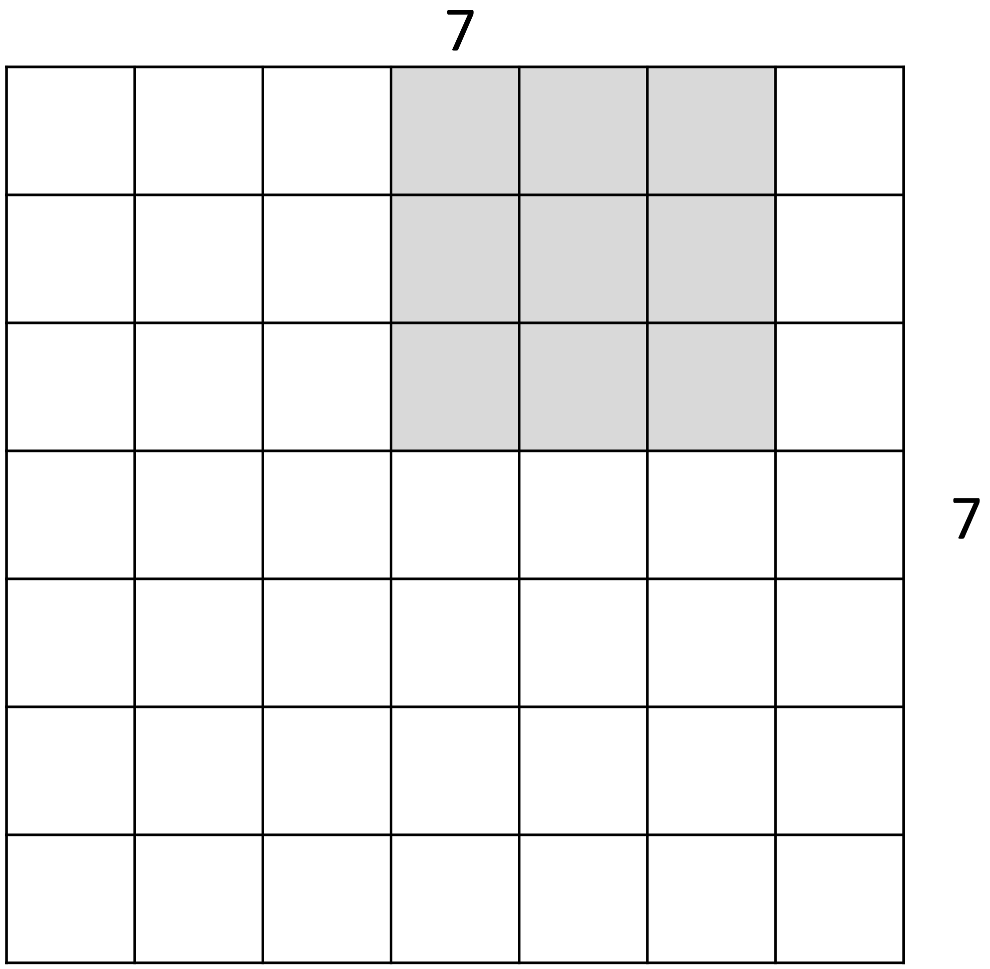
Convolutional Layer



A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 1

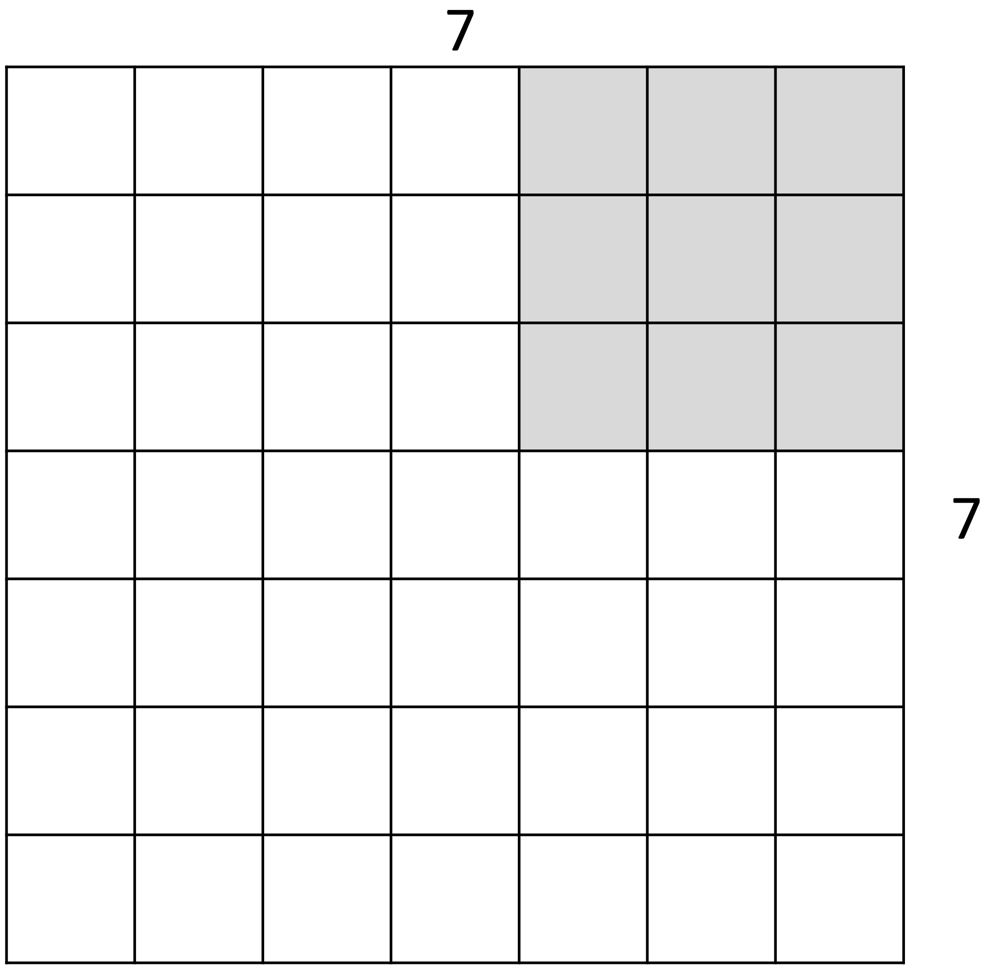
Convolutional Layer



A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 1

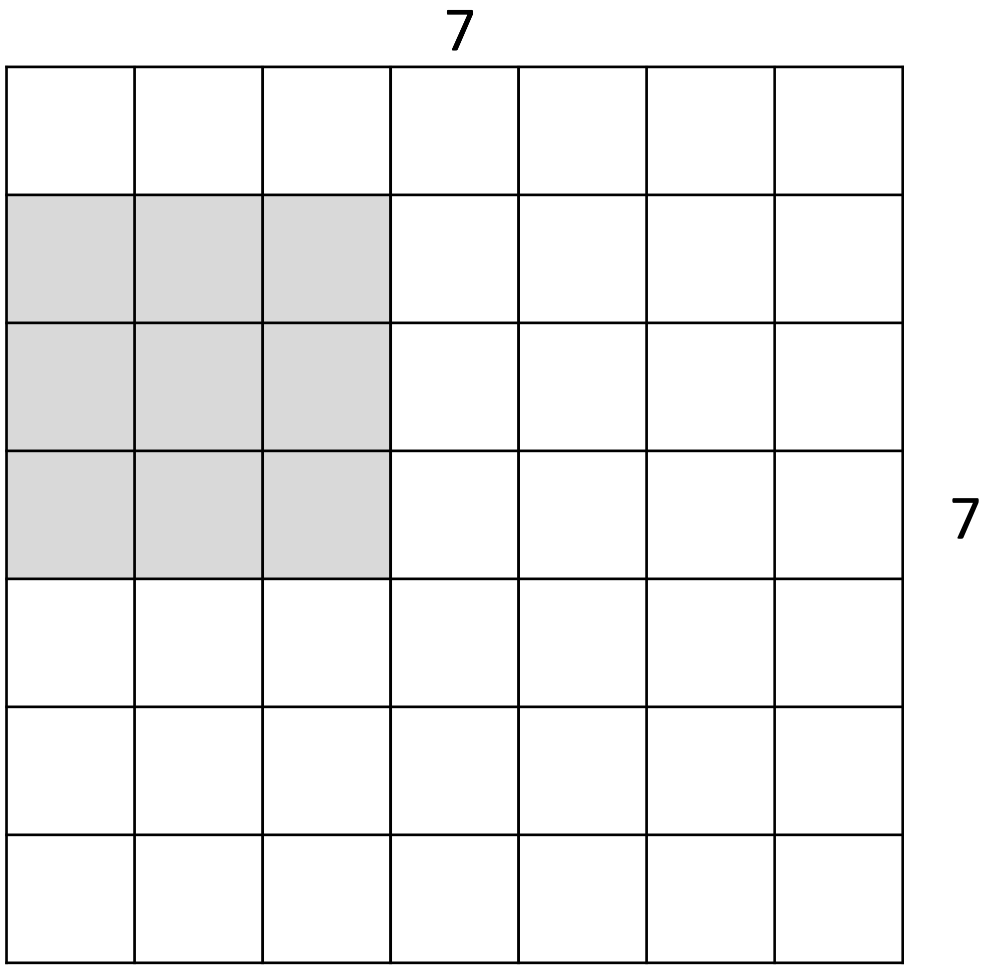
Convolutional Layer



A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 1

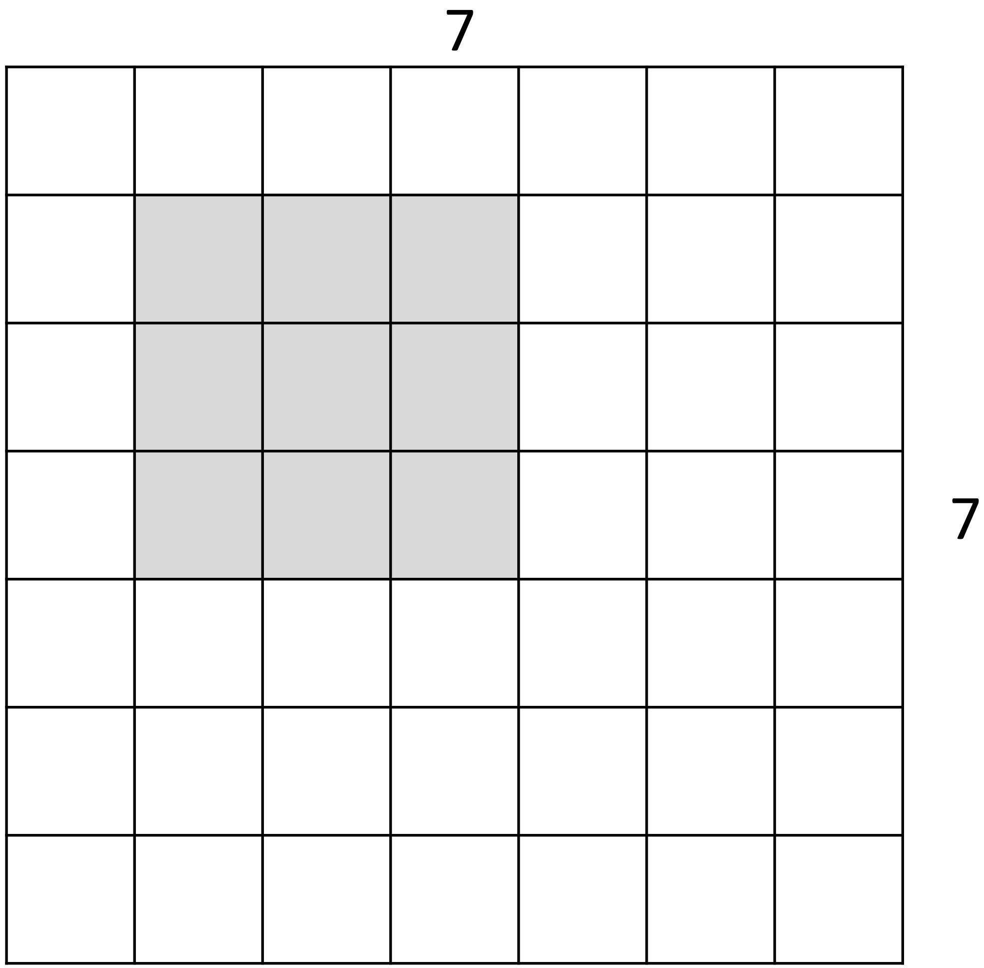
Convolutional Layer



A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 1

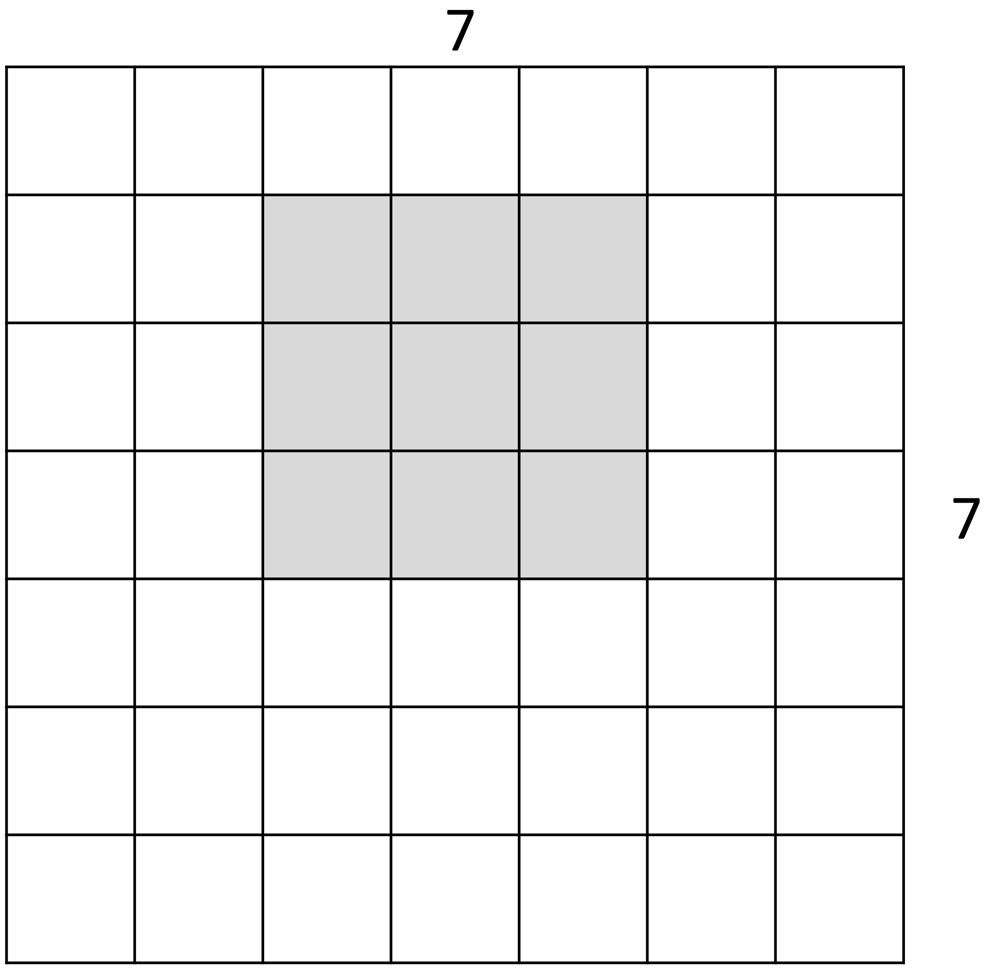
Convolutional Layer



A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 1

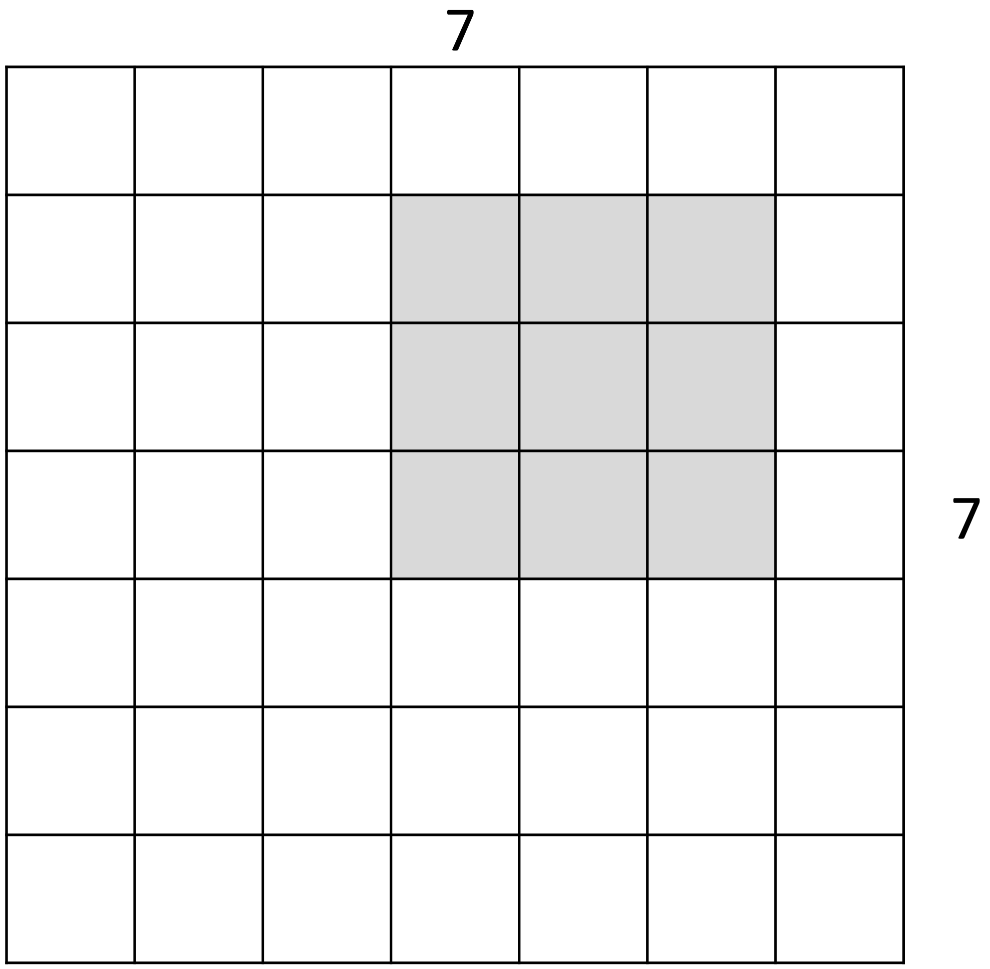
Convolutional Layer



A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 1

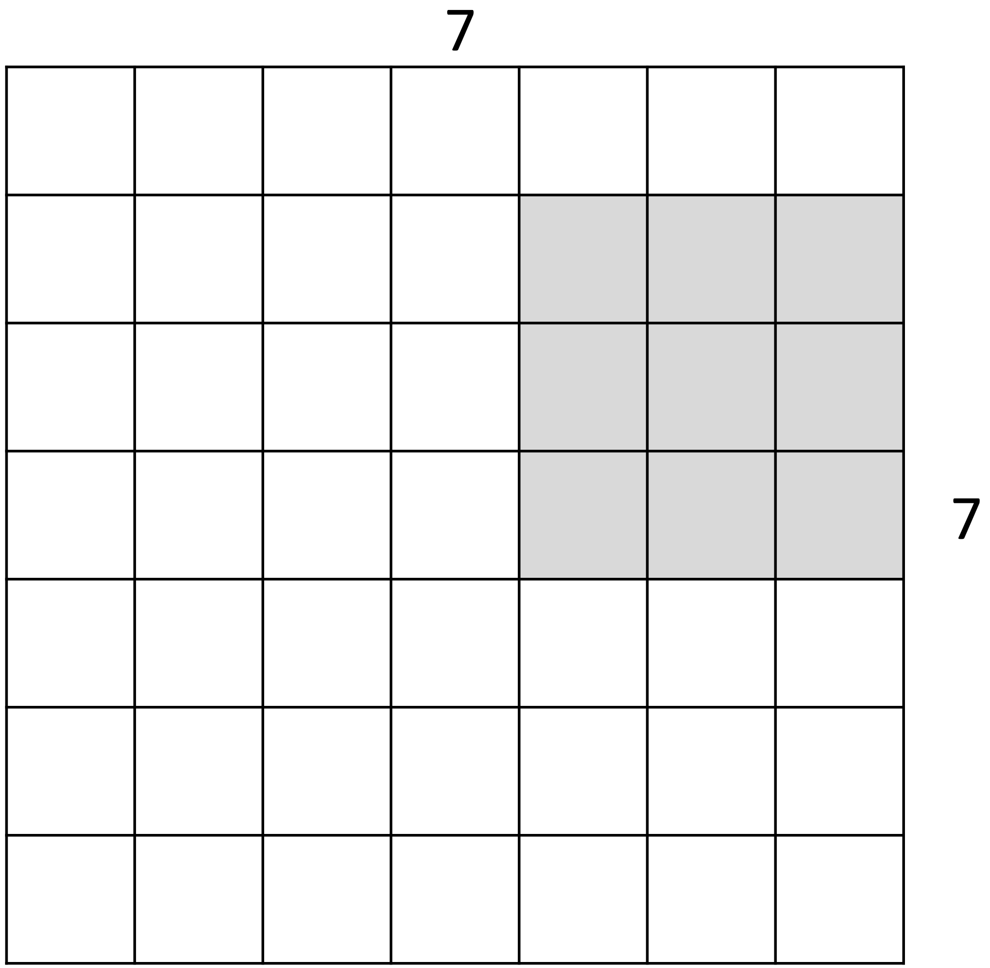
Convolutional Layer



A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 1

Convolutional Layer

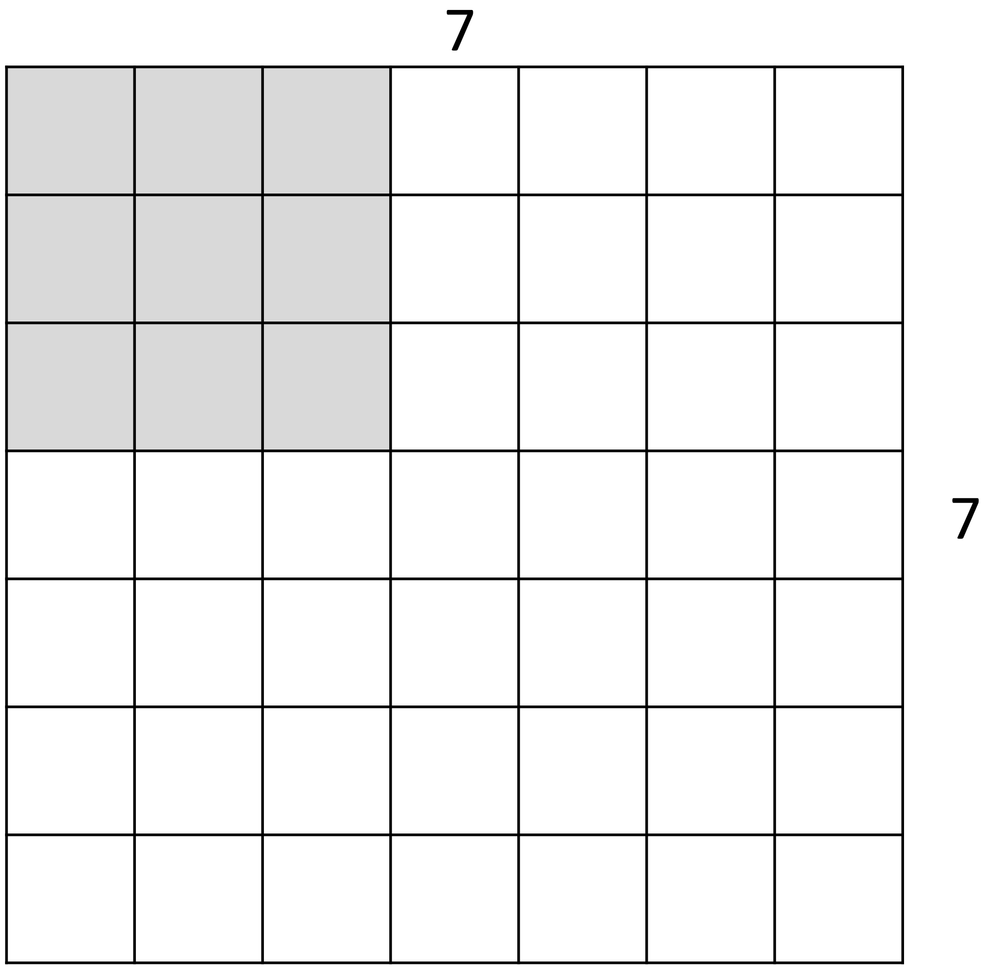


A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter, with stride 1

=> 5x5 output

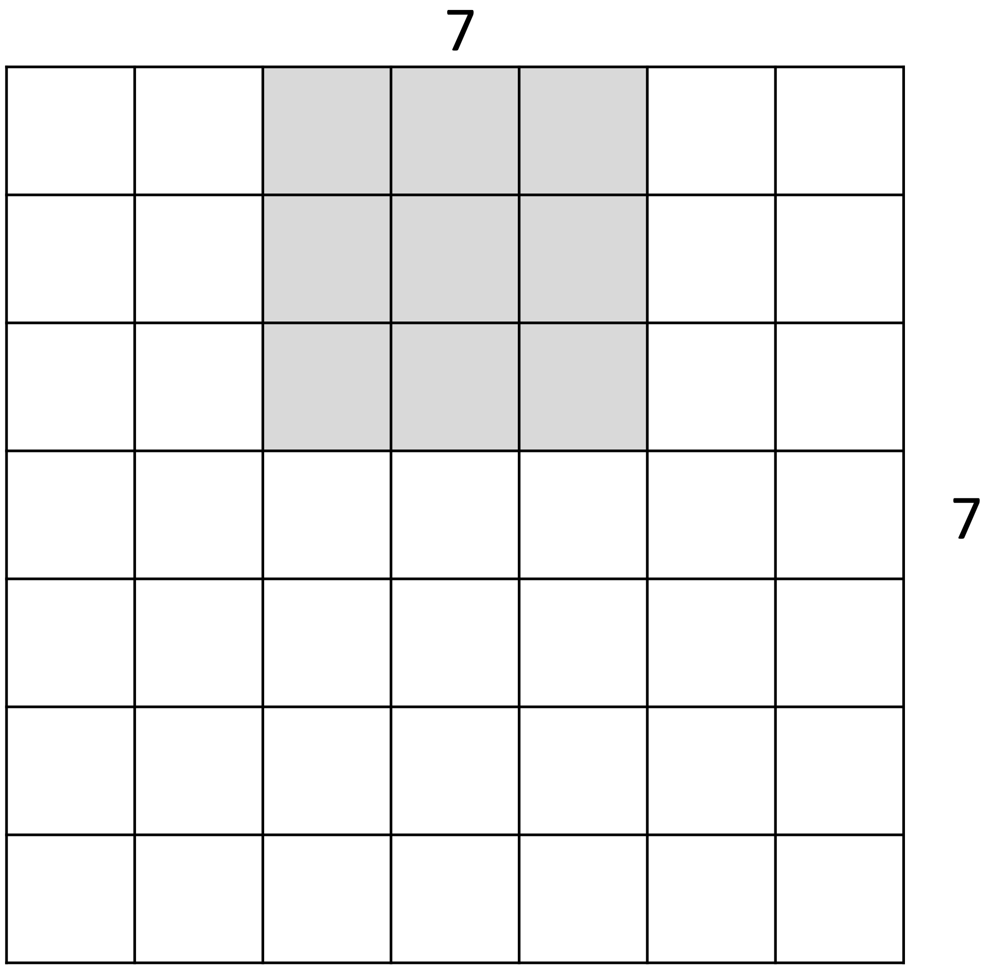
Convolutional Layer



A closer look at spatial dimensions:

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

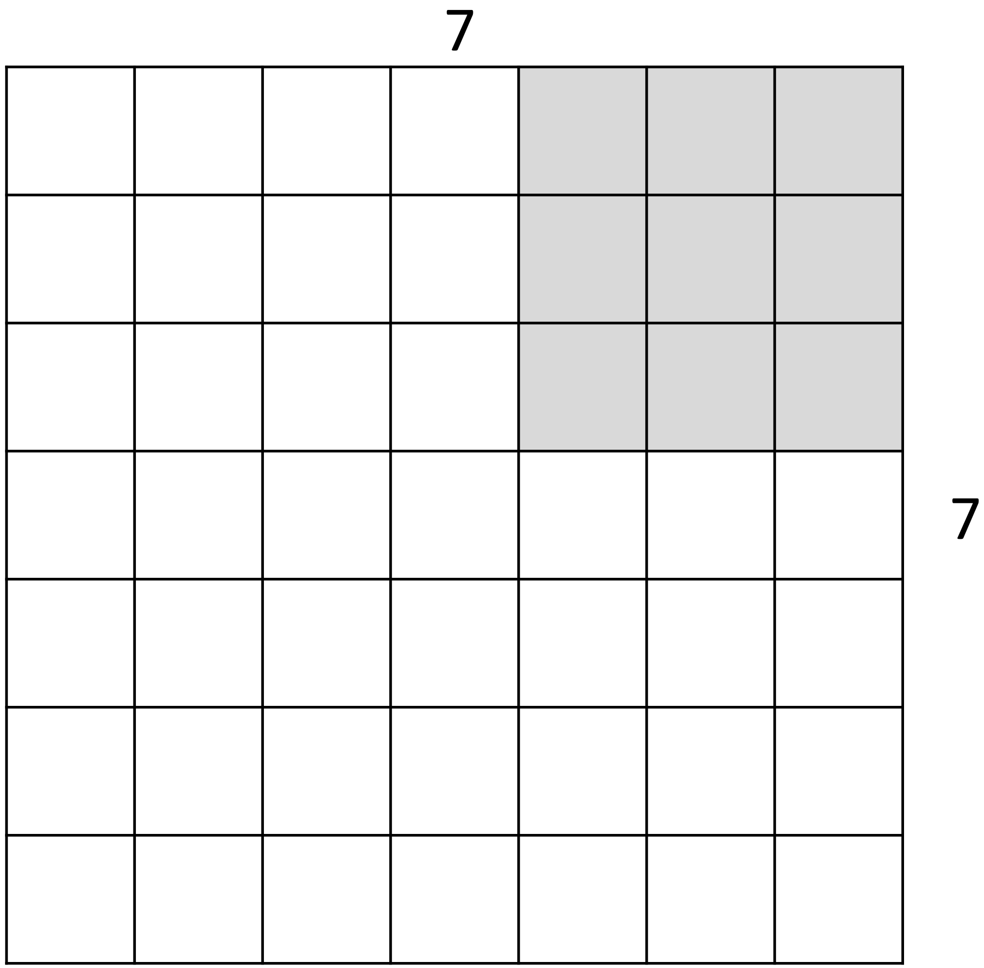
Convolutional Layer



A closer look at spatial dimensions:

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

Convolutional Layer

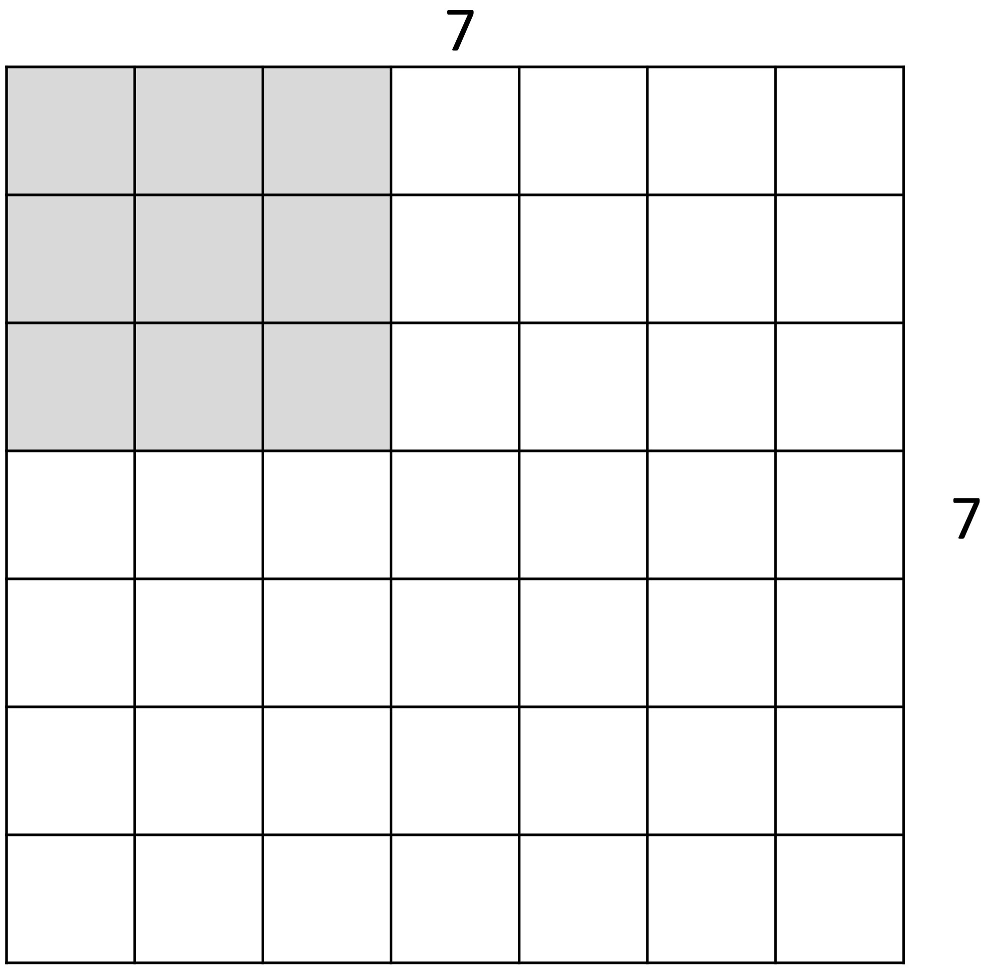


A closer look at spatial dimensions:

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

=> 3x3 output!

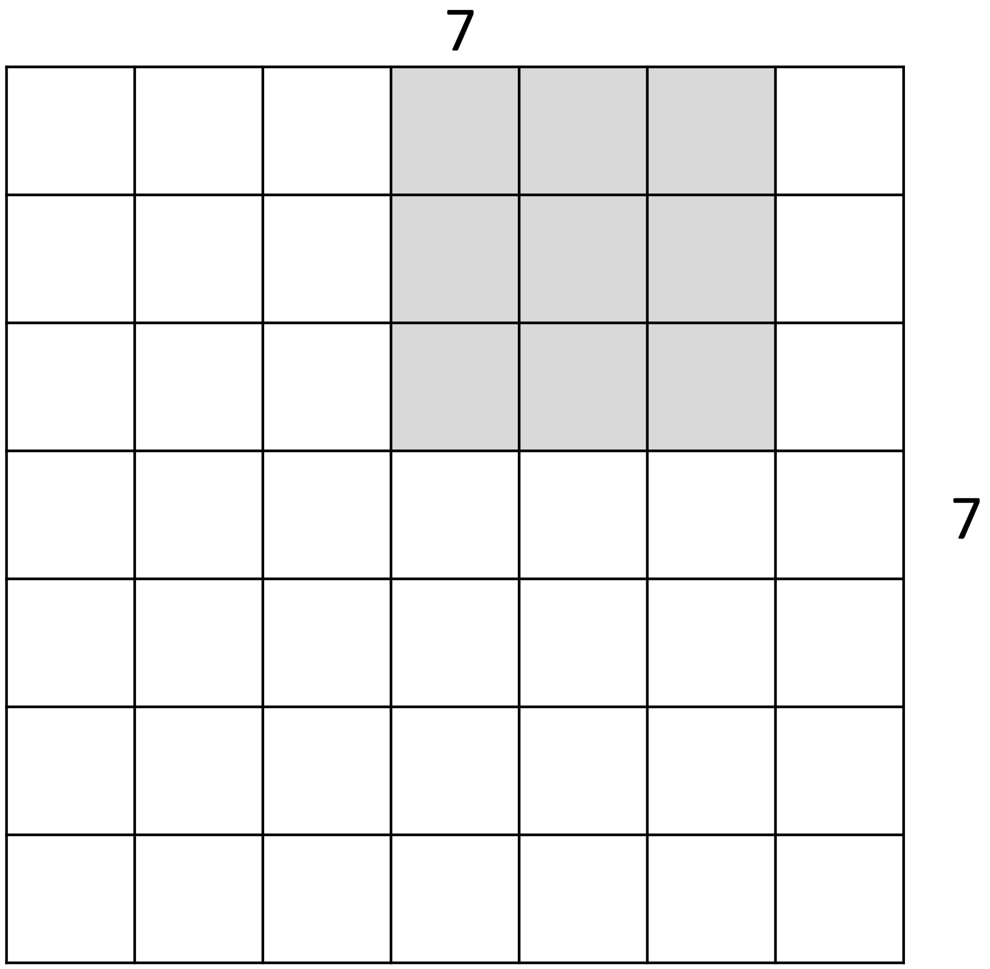
Convolutional Layer



A closer look at spatial dimensions:

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

Convolutional Layer

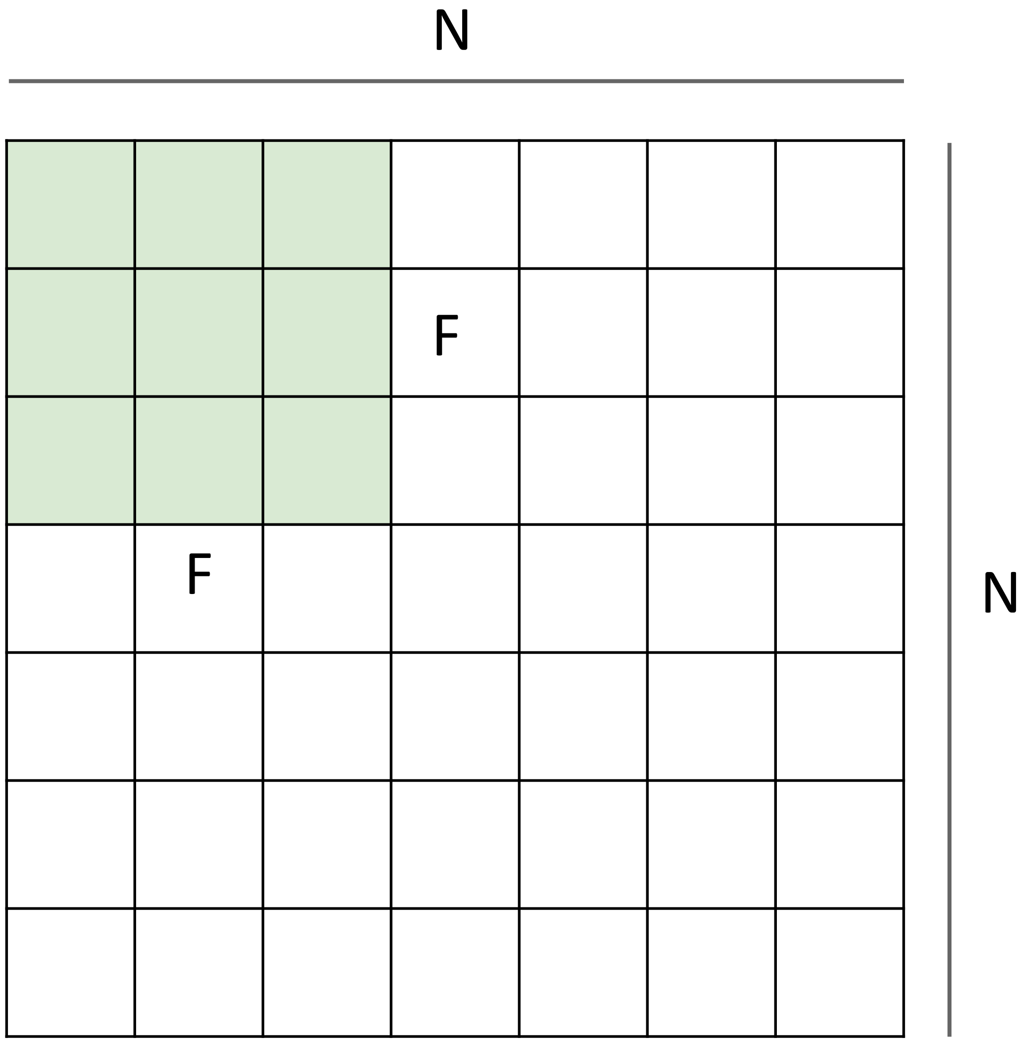


A closer look at spatial dimensions:

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

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

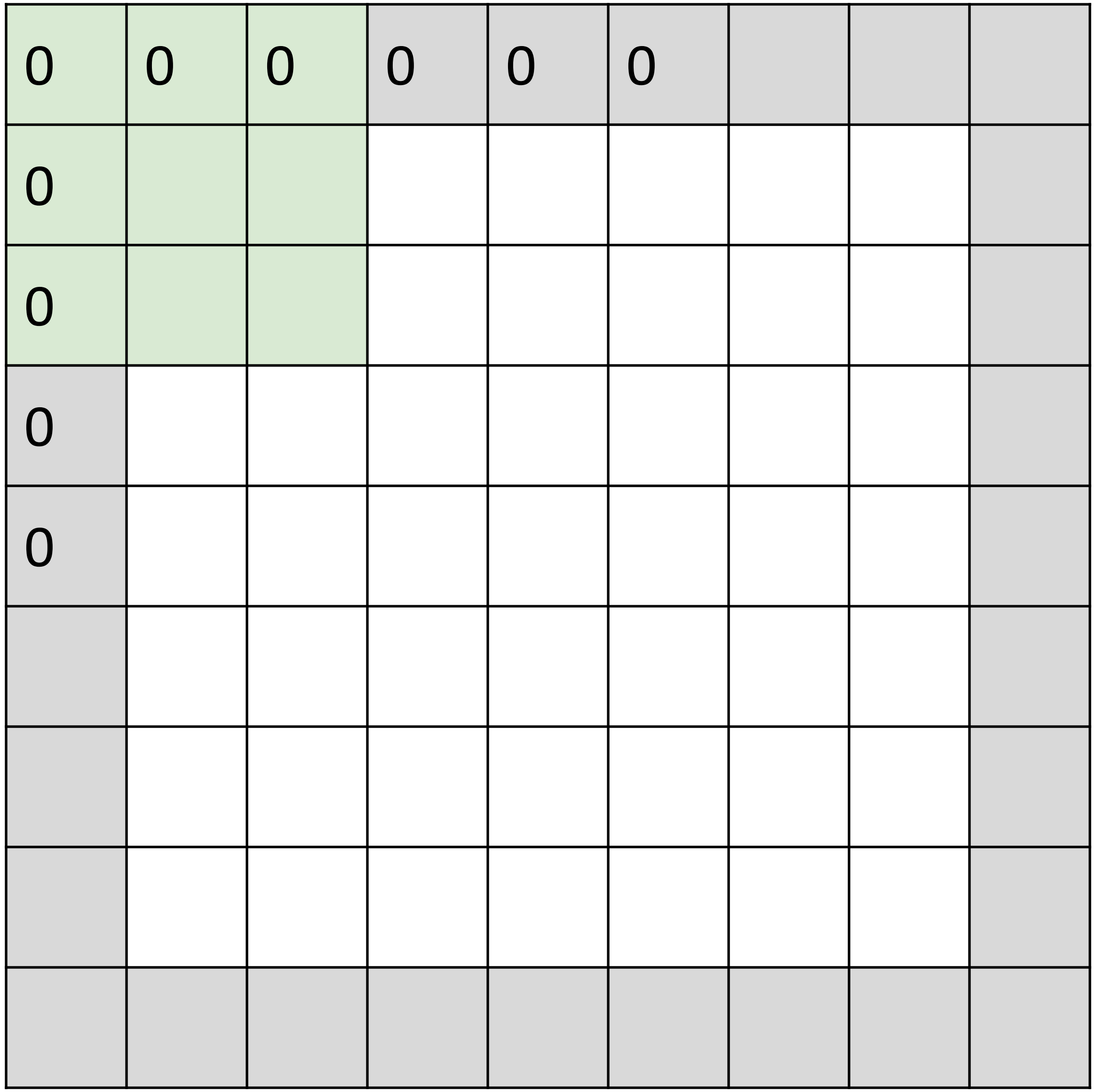
Convolutional Layer



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

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33

Convolutional Layer

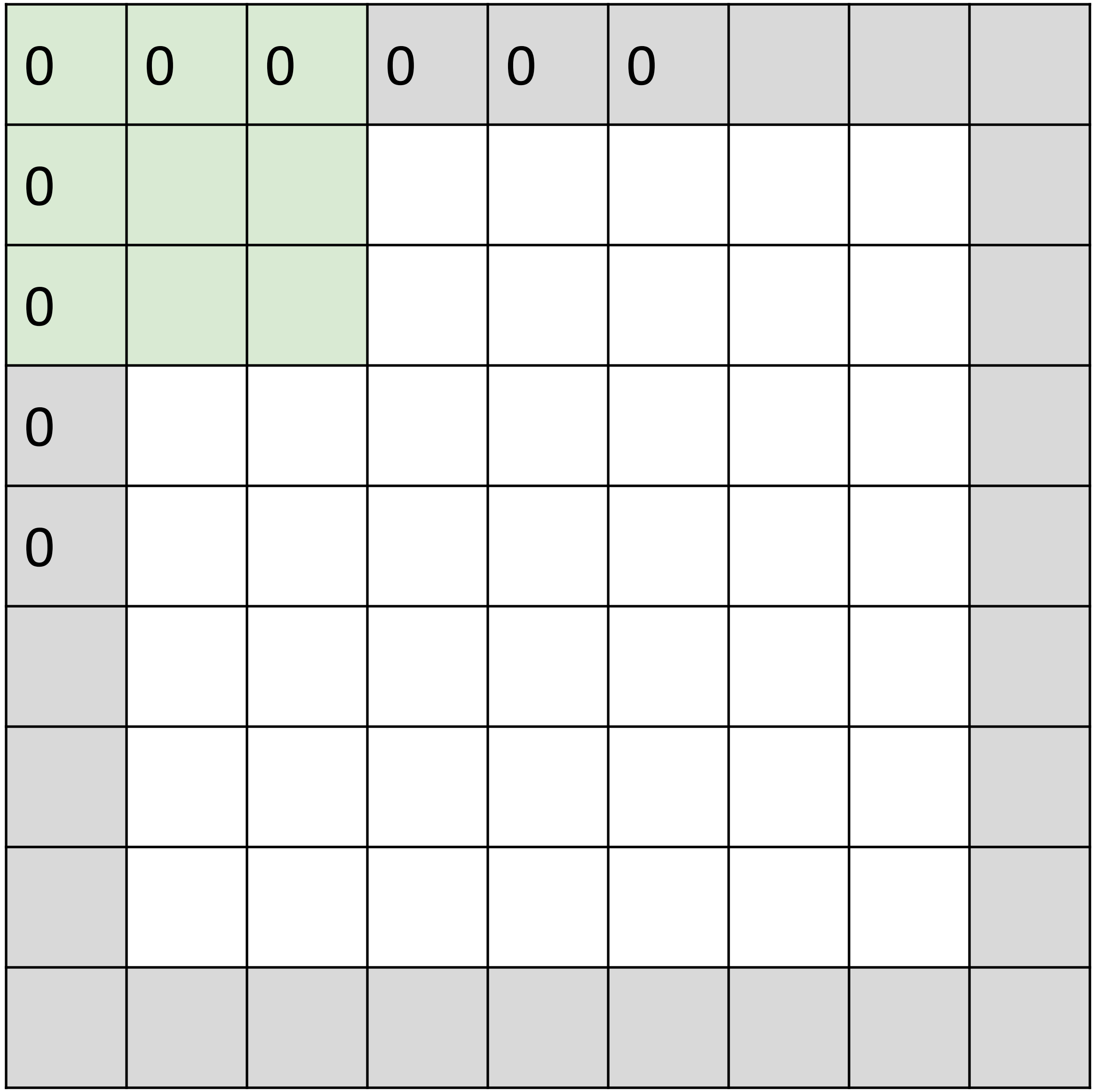


In practice: Common to zero pad the border

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

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

Convolutional Layer

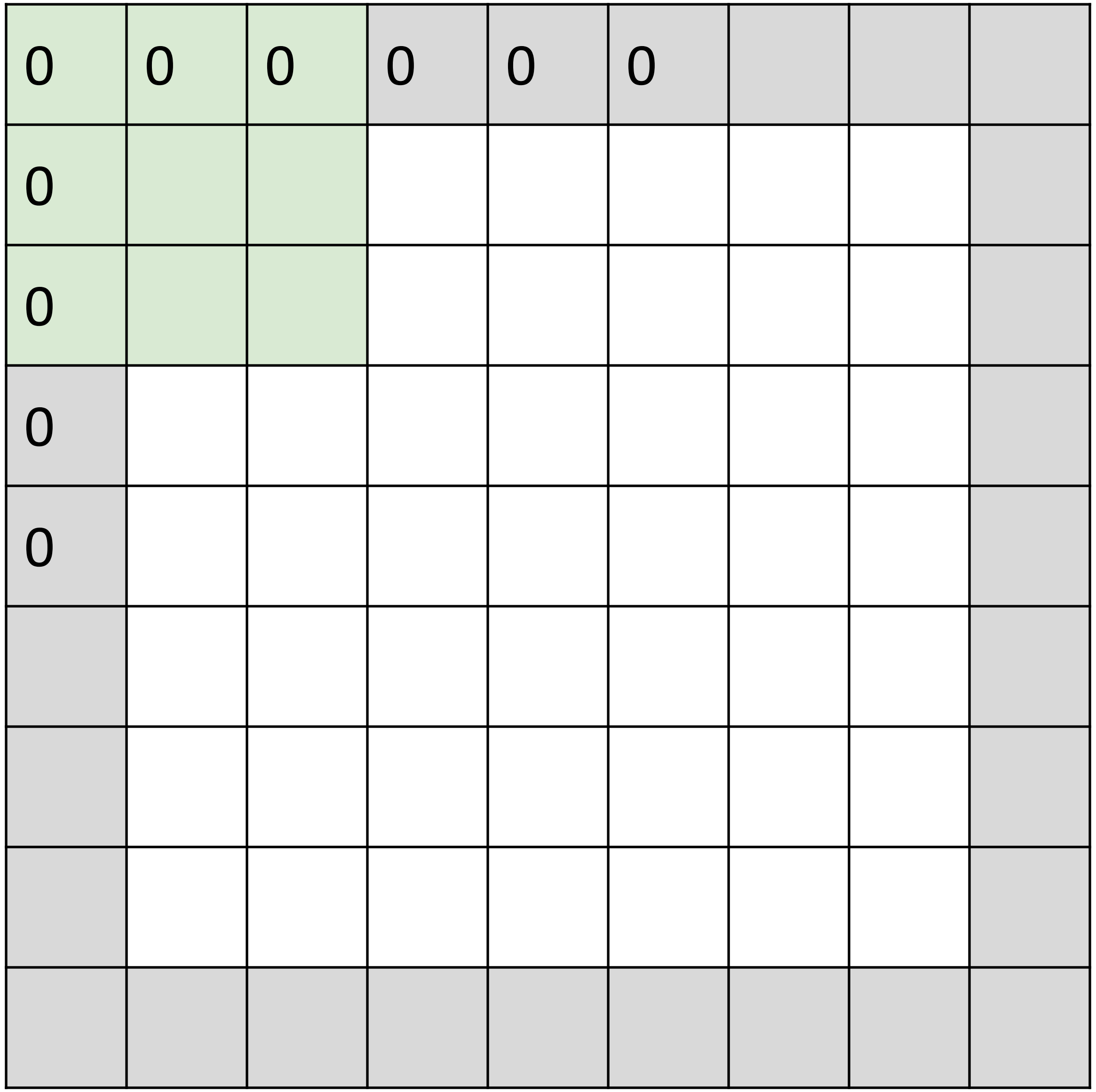


In practice: Common to zero pad the border

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

7x7 output!

Convolutional Layer

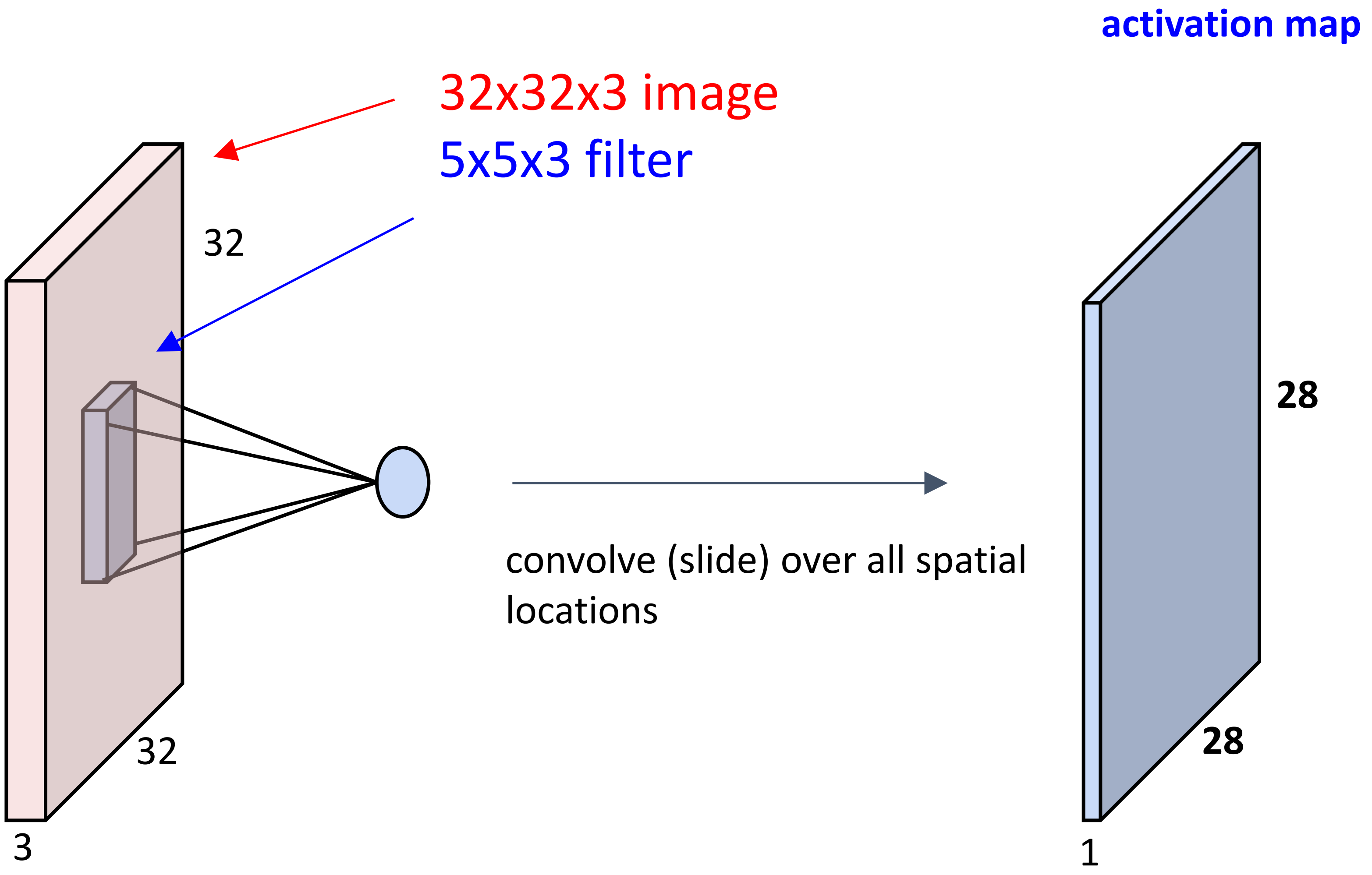


In practice: Common to zero pad the border

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

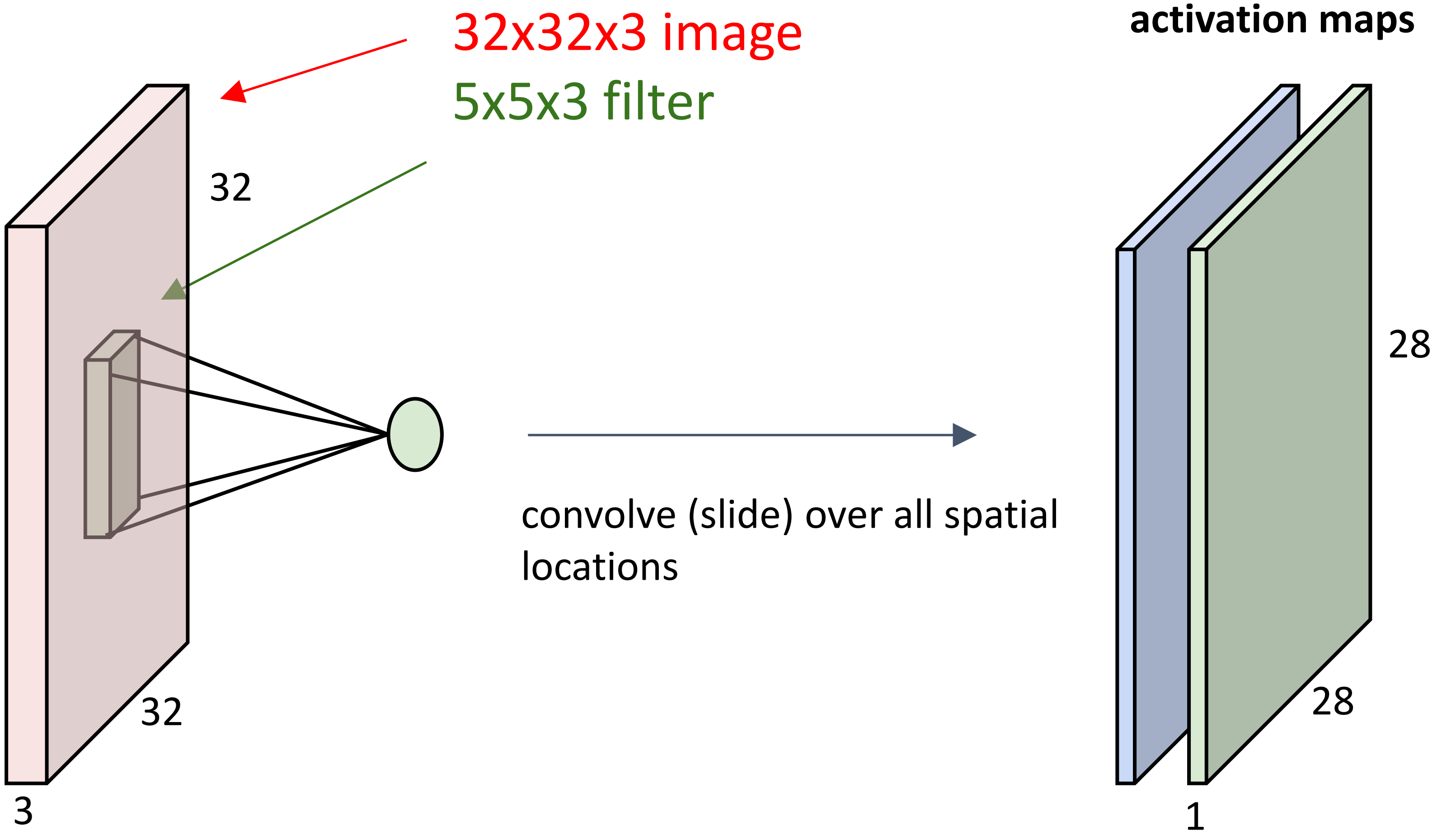
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Convolutional Layer

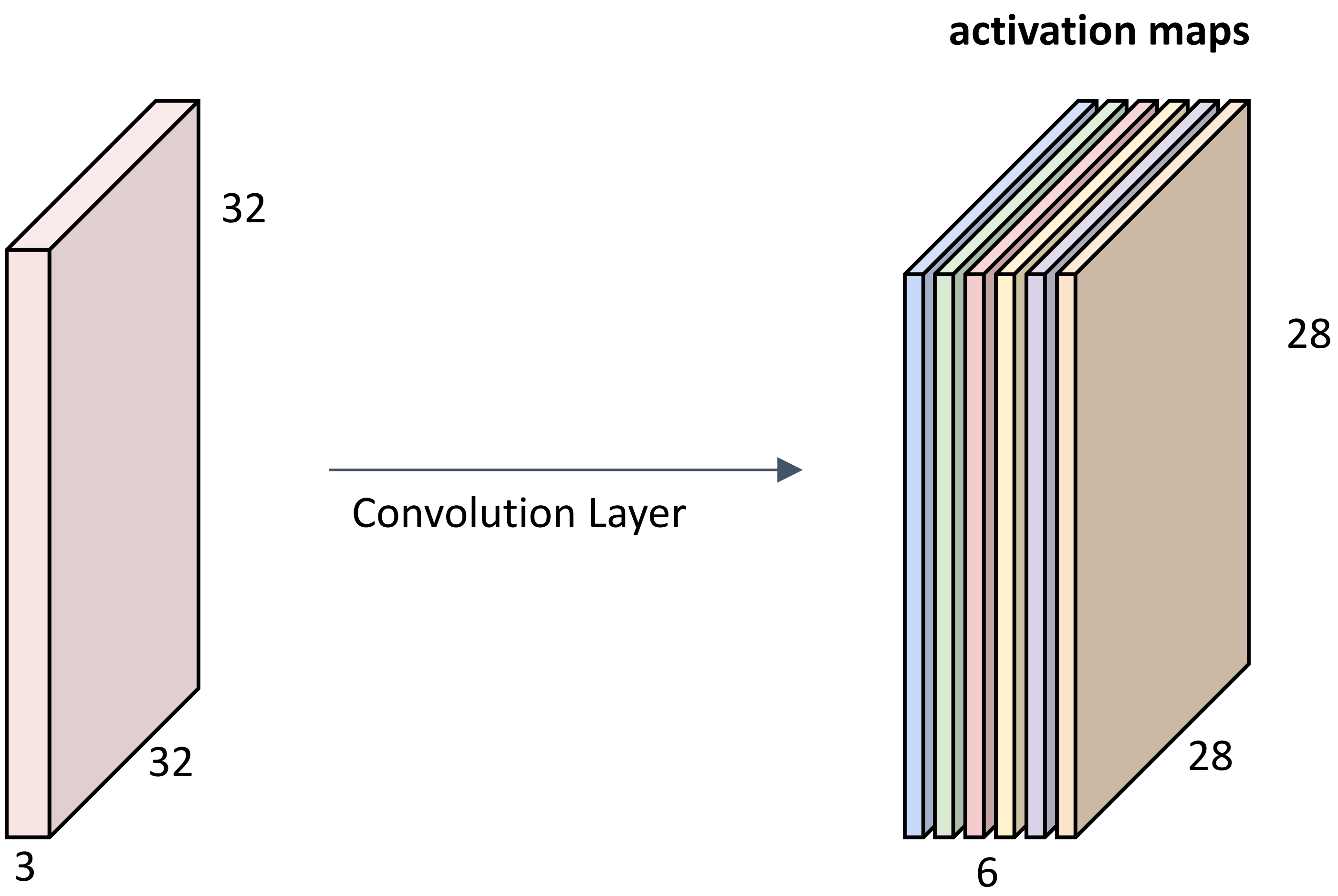


Convolutional Layer

consider a second, green filter



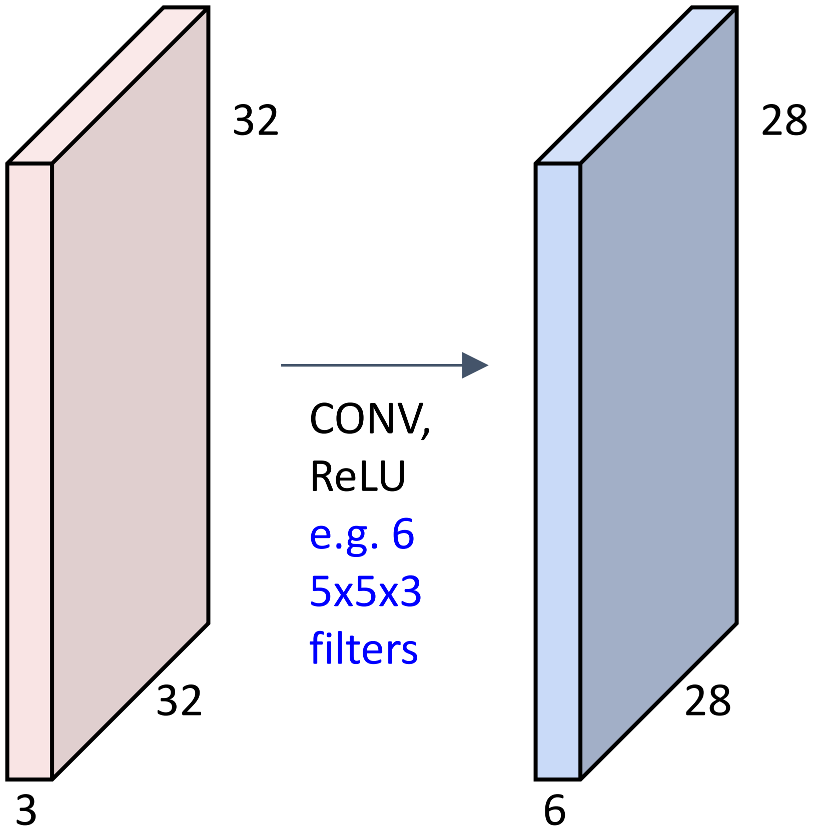
Convolutional Layer



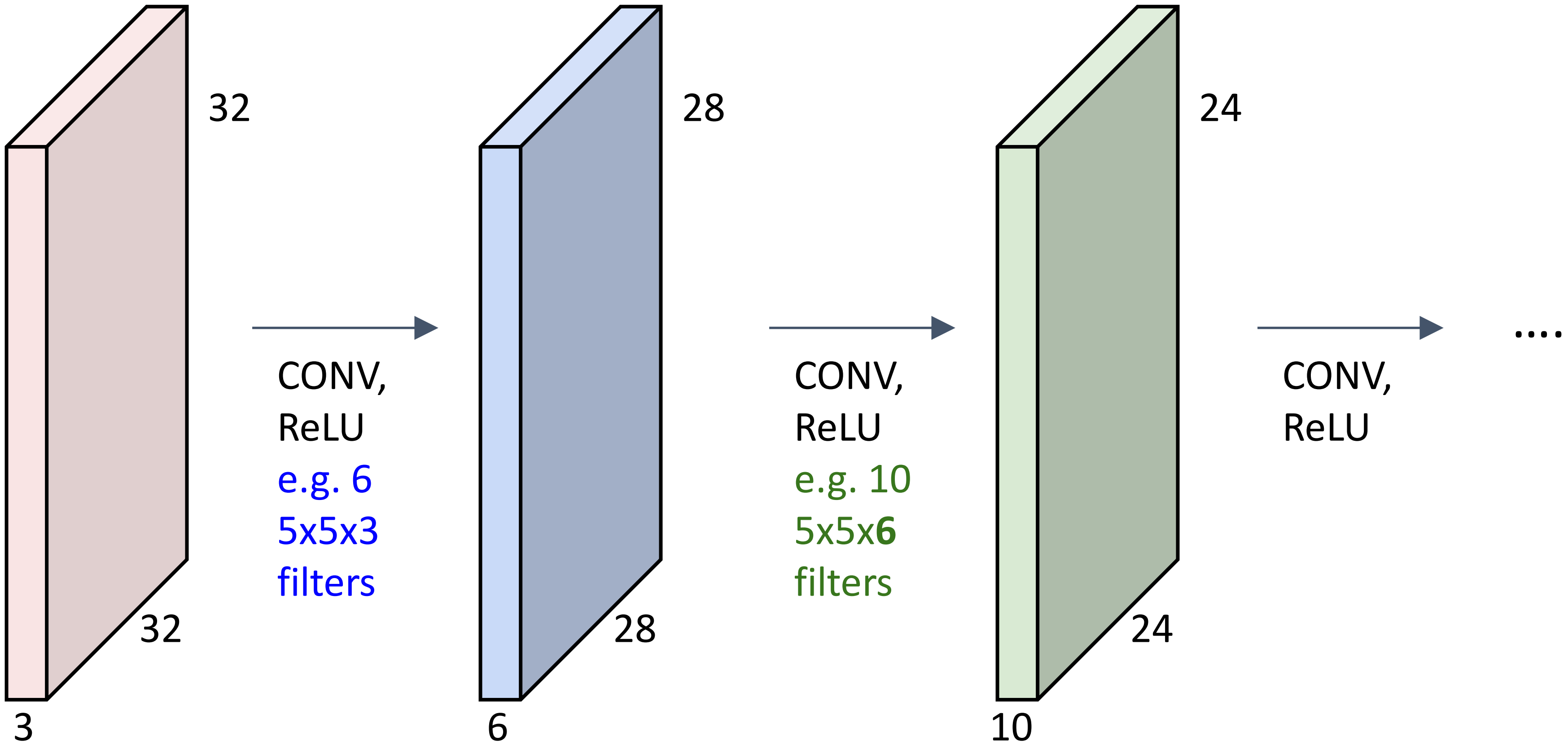
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps

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

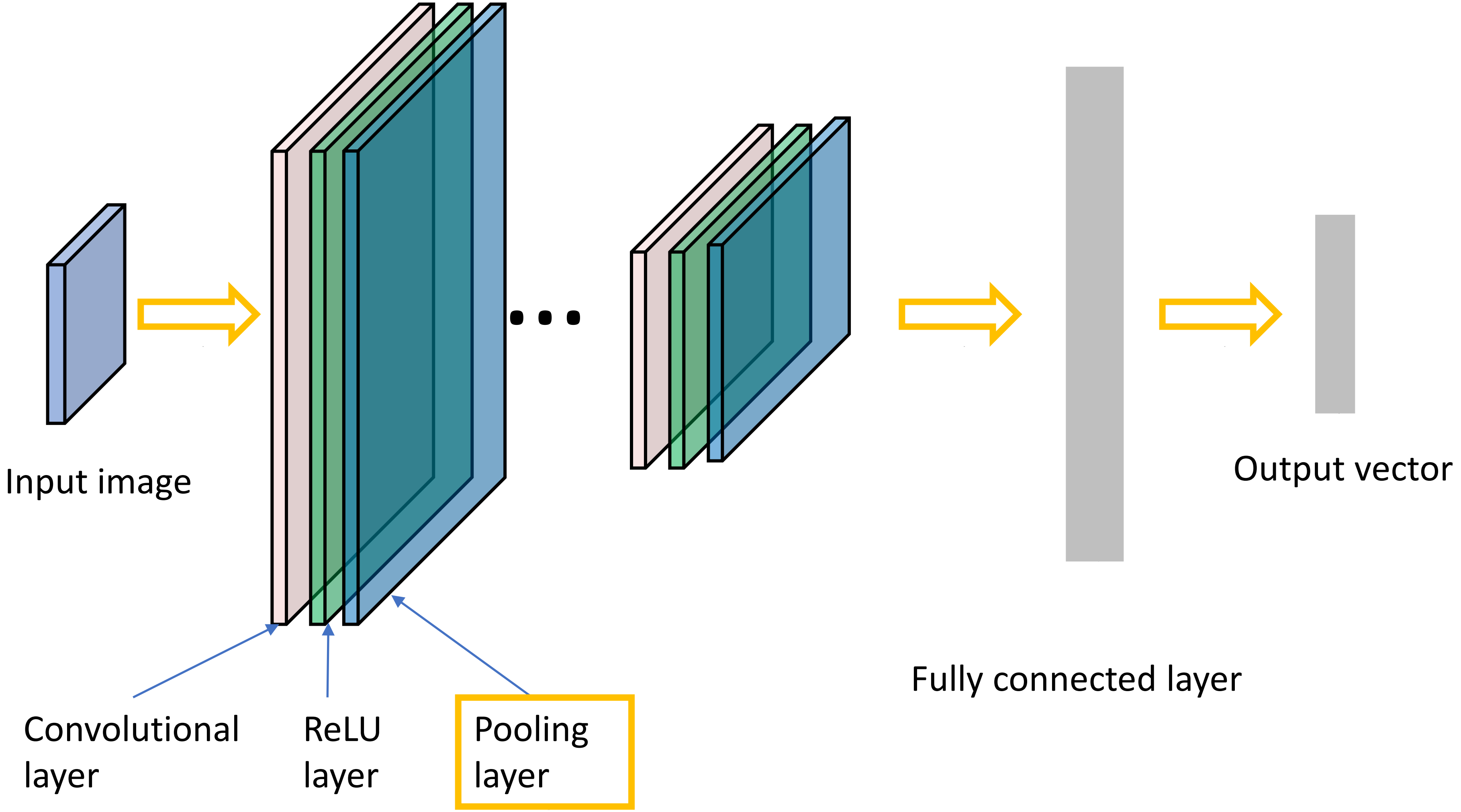
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



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

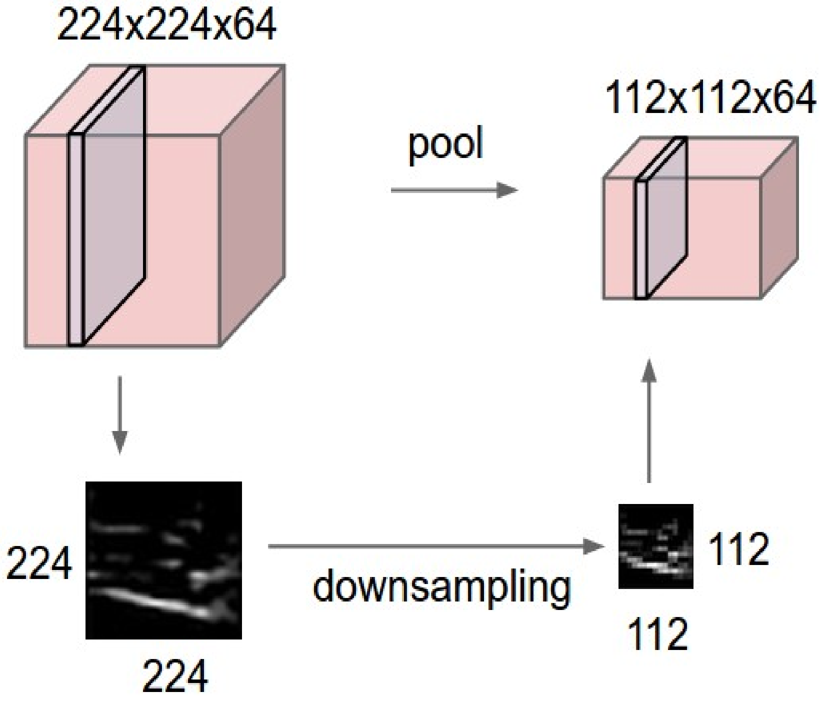


Convolutional Neural Networks

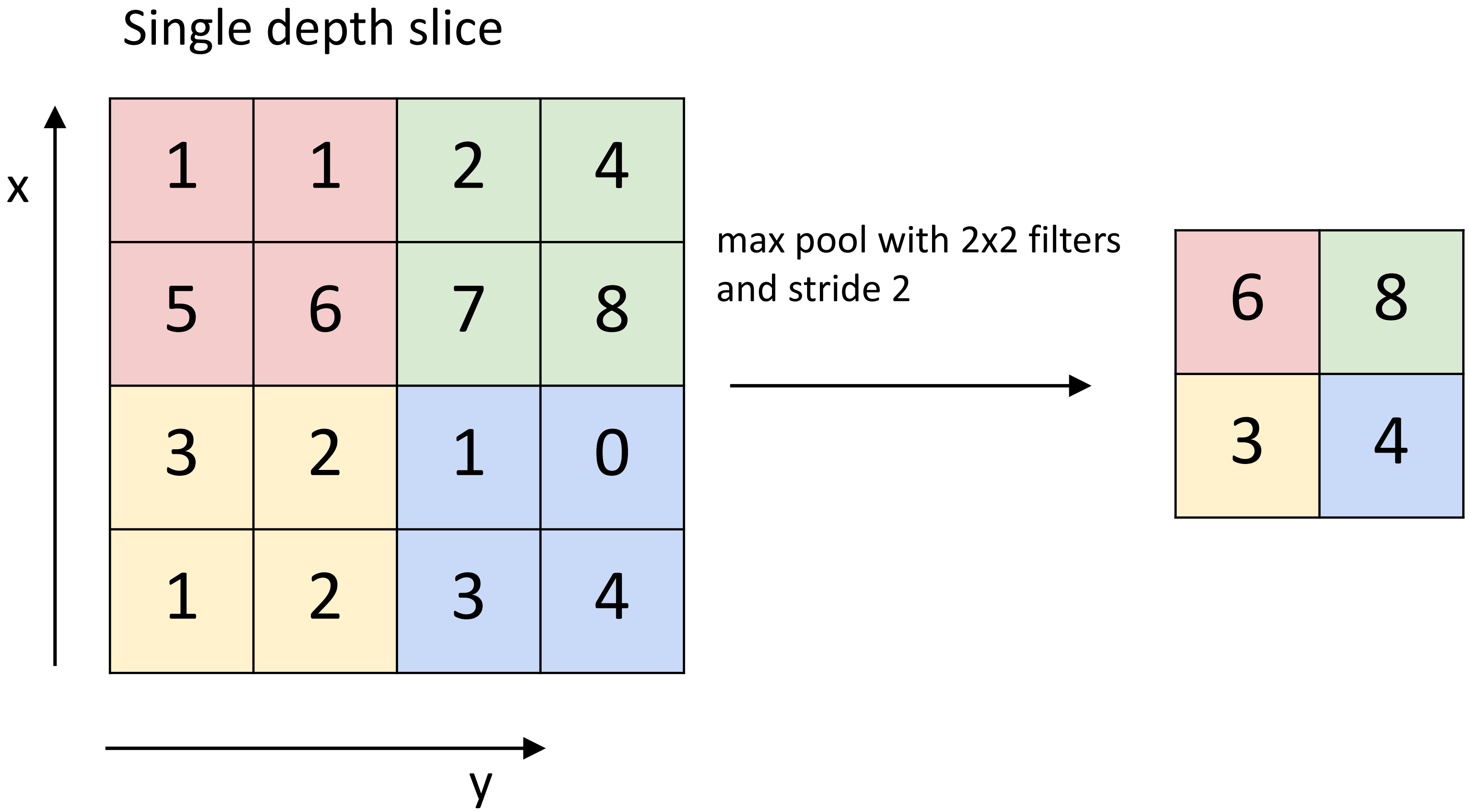


Pooling Layer

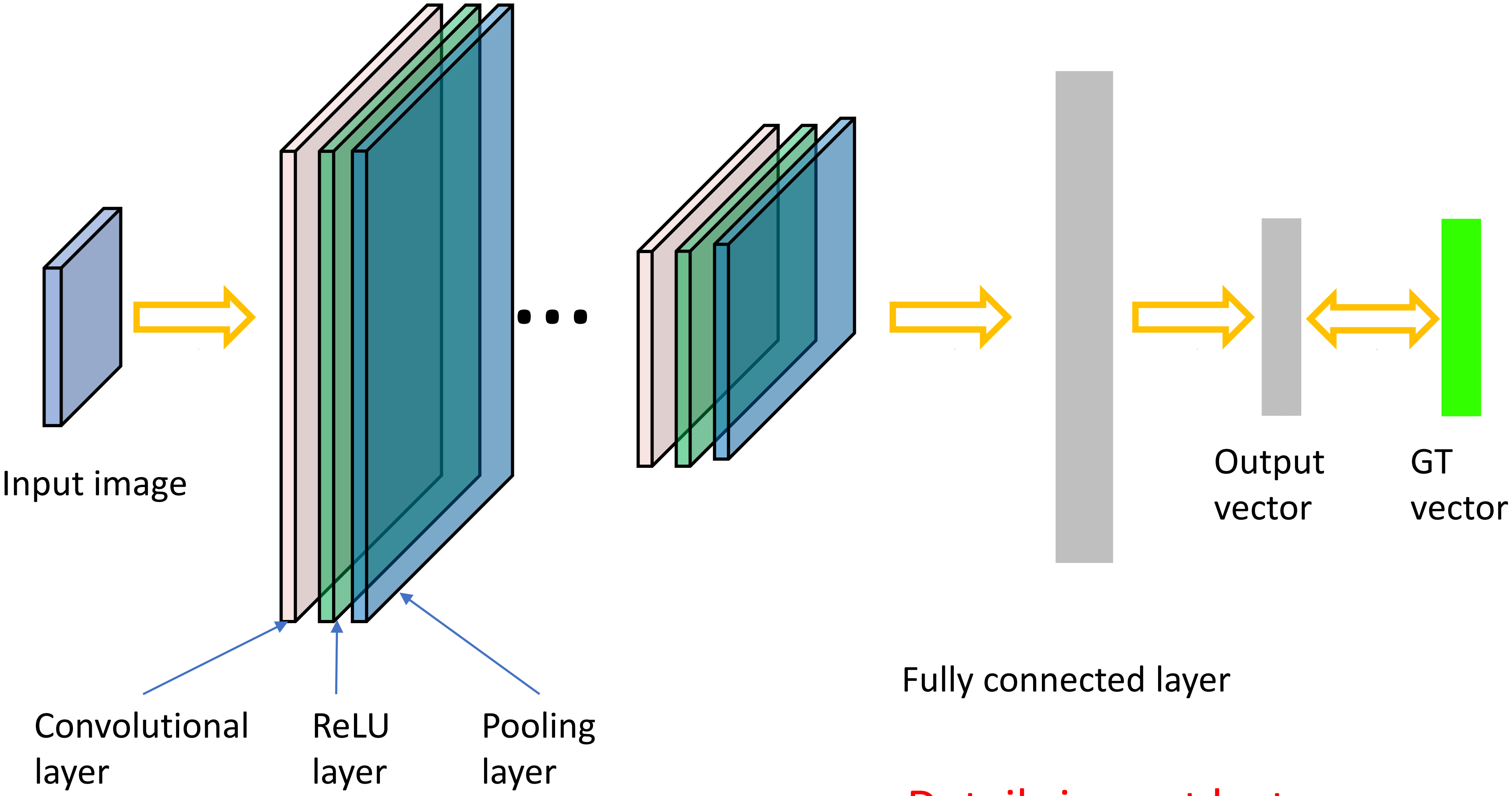
- makes the representations smaller and more manageable
- operates over each activation map independently:



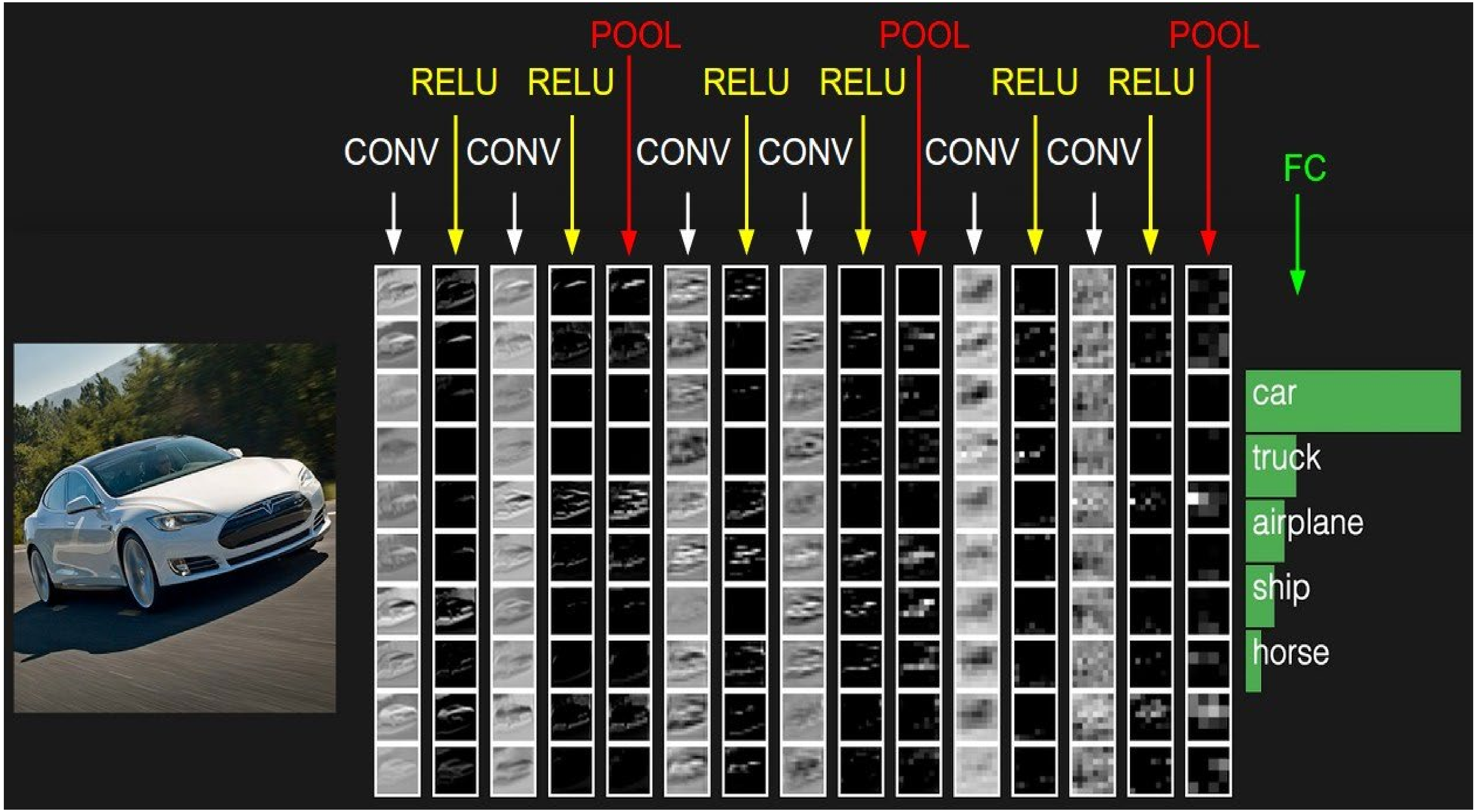
Max Pooling Layer



Training: back-propagate errors

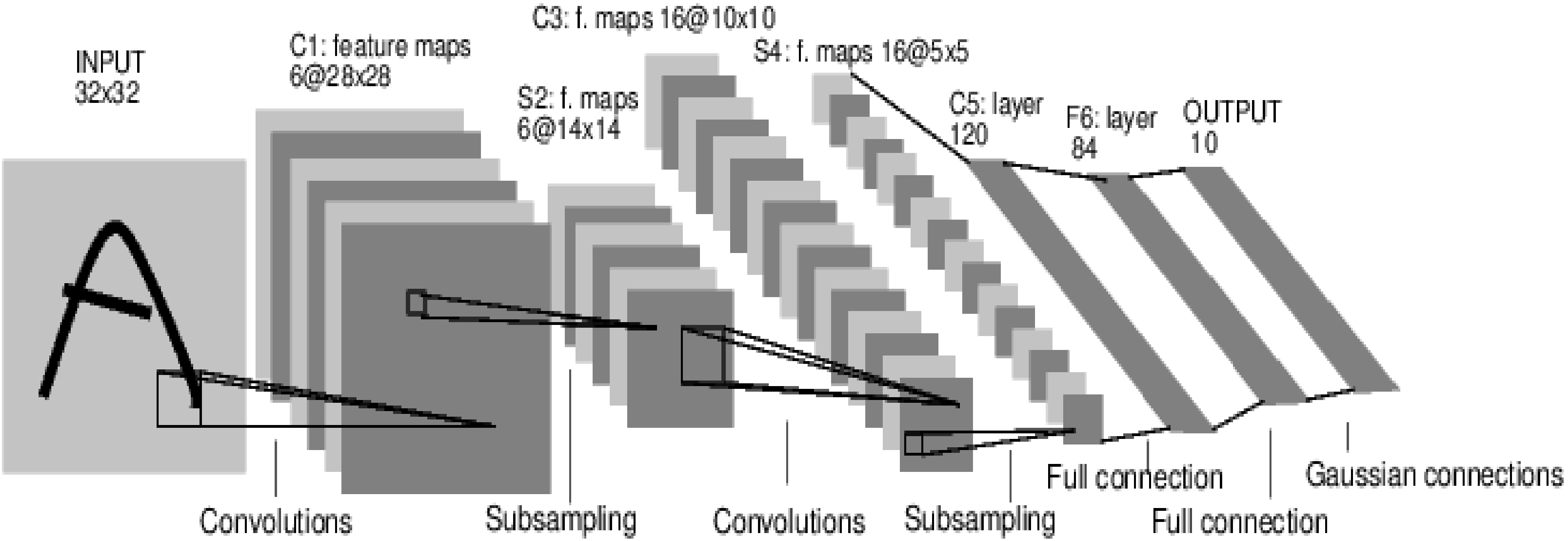


Details in next lecture


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

Case Study: LeNet-5

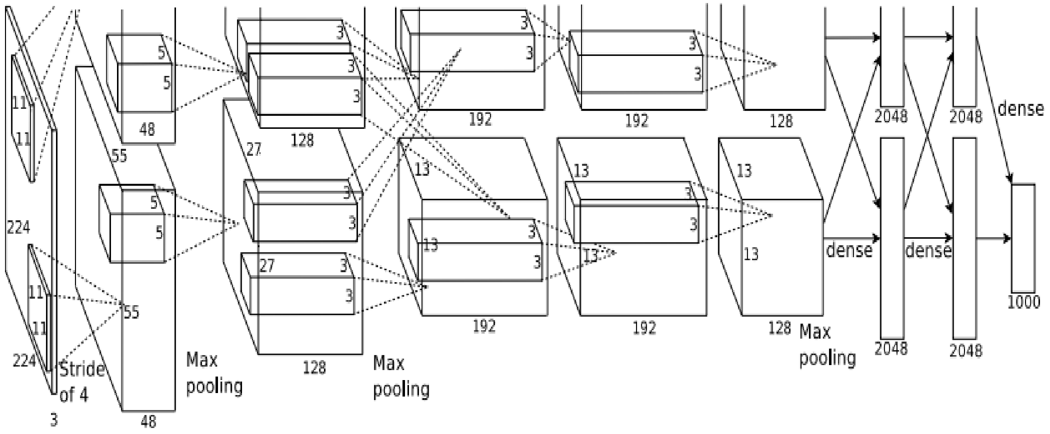
[LeCun et al., 1998]



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

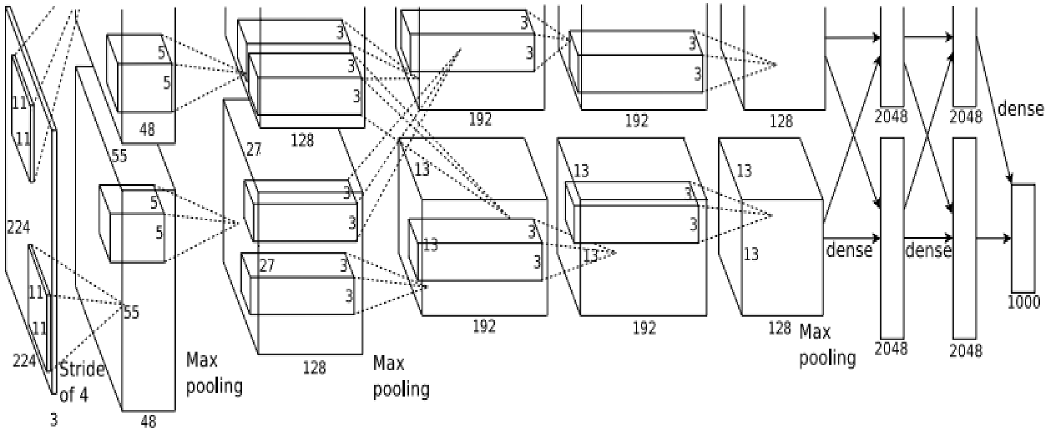
First layer (CONV1): 96 11x11 filters applied at stride 4

=>

Q: what is the output volume size? Hint: (227-11)/4+1 = 55

Case Study: AlexNet

[Krizhevsky et al. 2012]



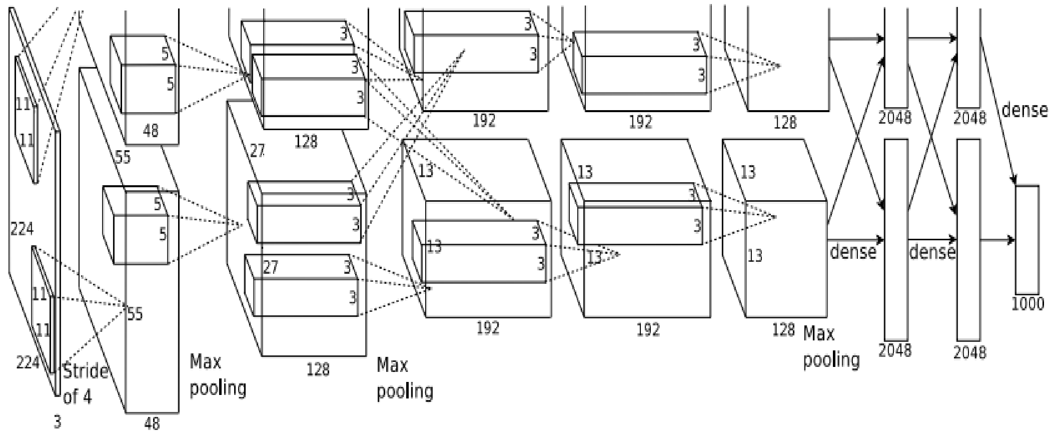
Input: 227x227x3 images

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]

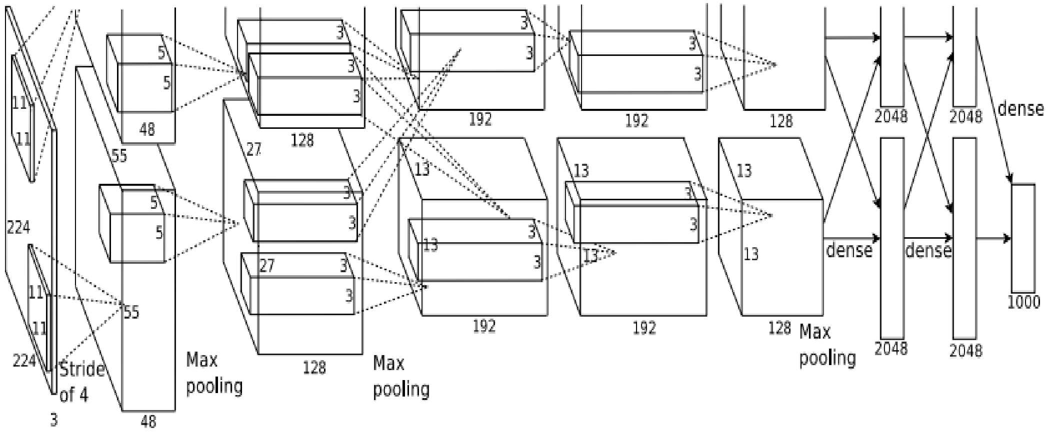


Input: 227x227x3 images

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

Case Study: AlexNet

[Krizhevsky et al. 2012]



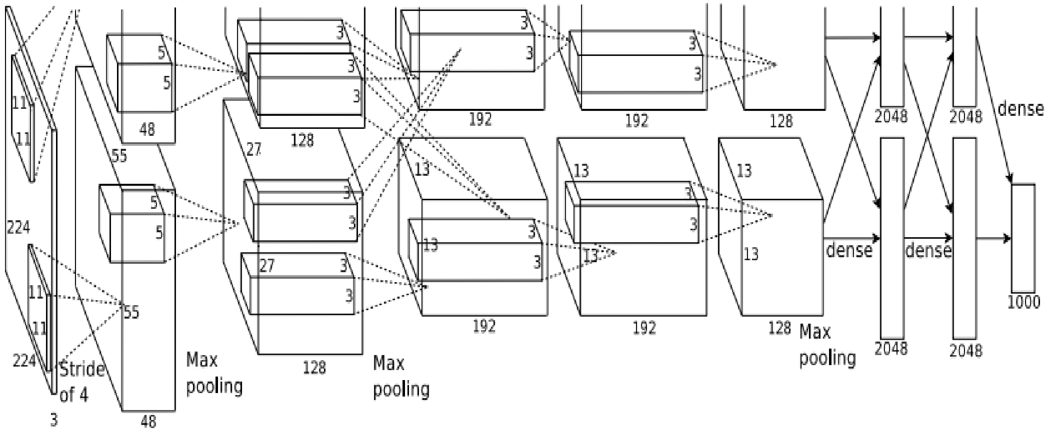
Input: 227x227x3 images
After CONV1: 55x55x96

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

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

Case Study: AlexNet

[Krizhevsky et al. 2012]



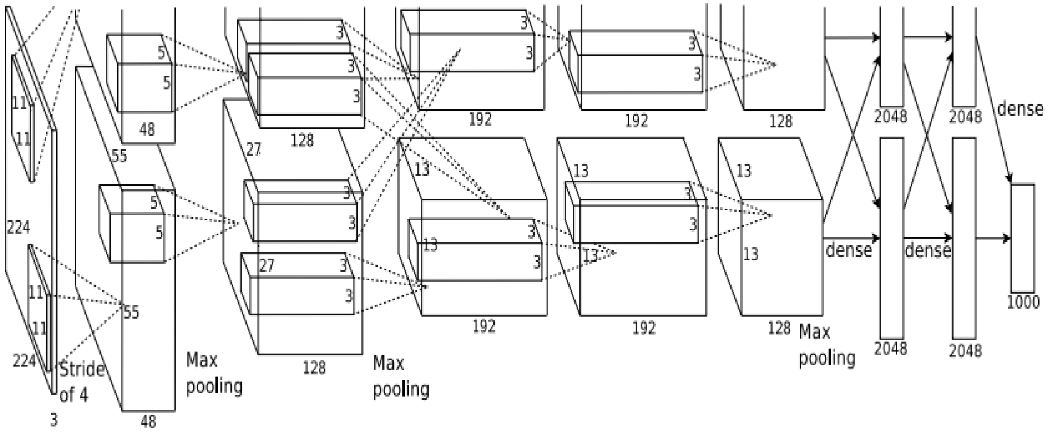
Input: 227x227x3 images
After CONV1: 55x55x96

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

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

Case Study: AlexNet

[Krizhevsky et al. 2012]

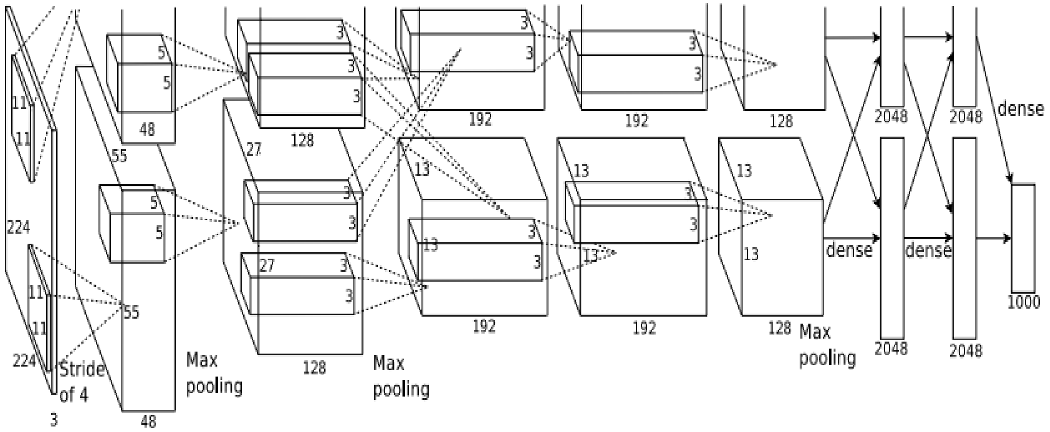


Input: 227x227x3 images
After CONV1: 55x55x96

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

Case Study: AlexNet

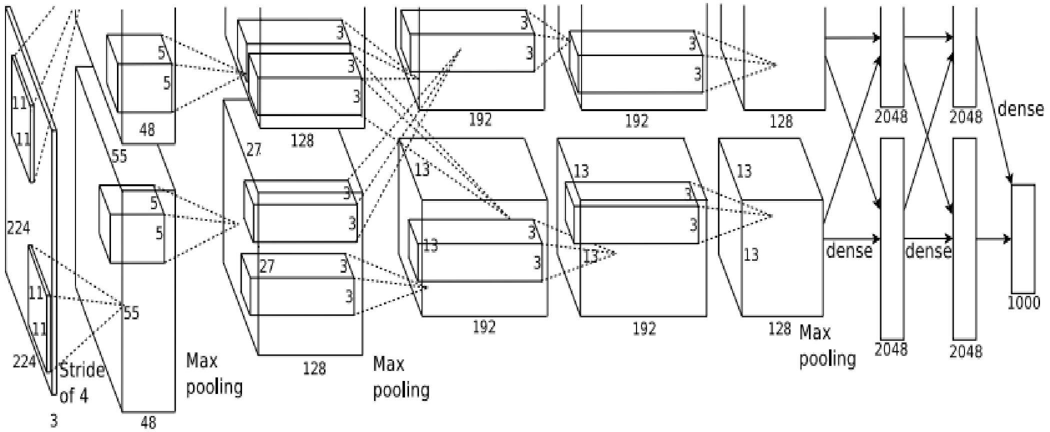
[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...

Case Study: AlexNet

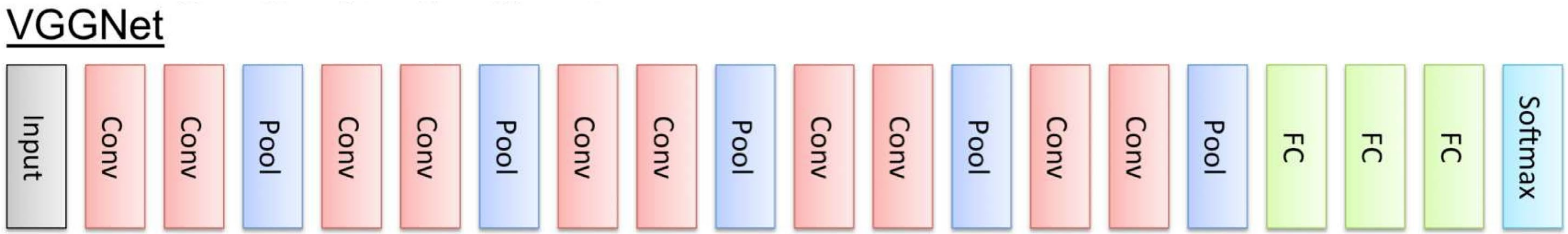
[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Case Study: VGGNet

[Simonyan and Zisserman, 2014]



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

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

Case Study: VGGNet

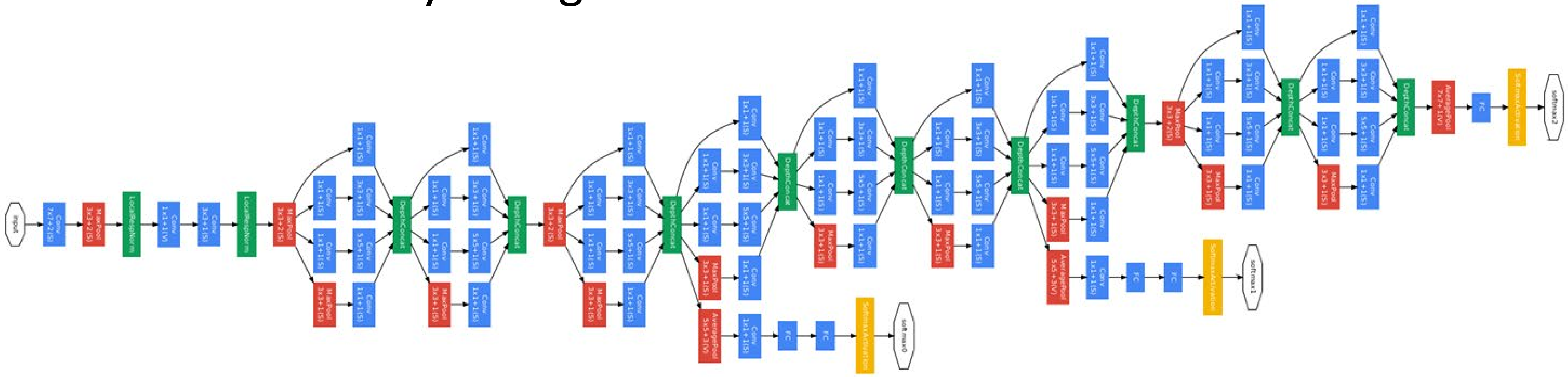
[Simonyan and Zisserman, 2014]

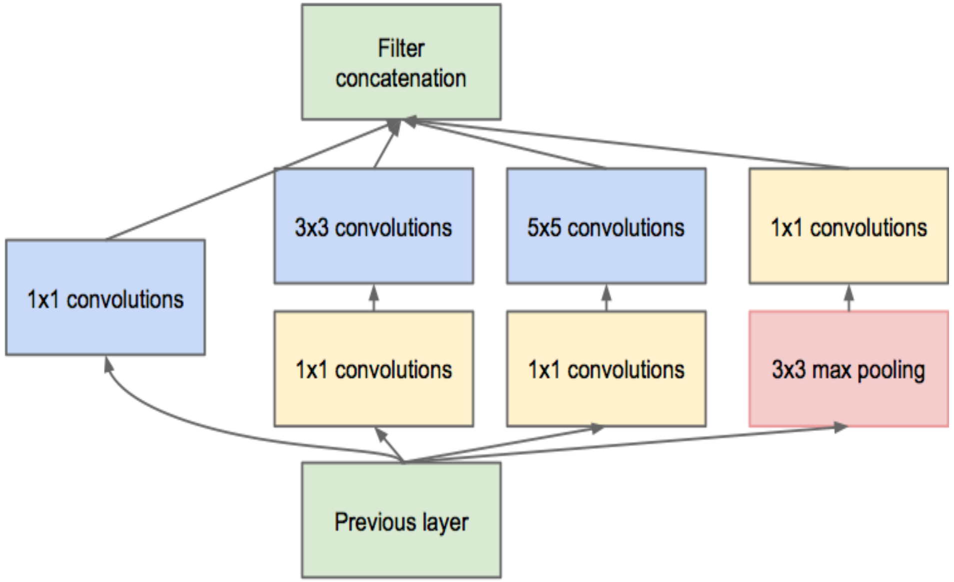
INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M 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

(not counting biases)

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:

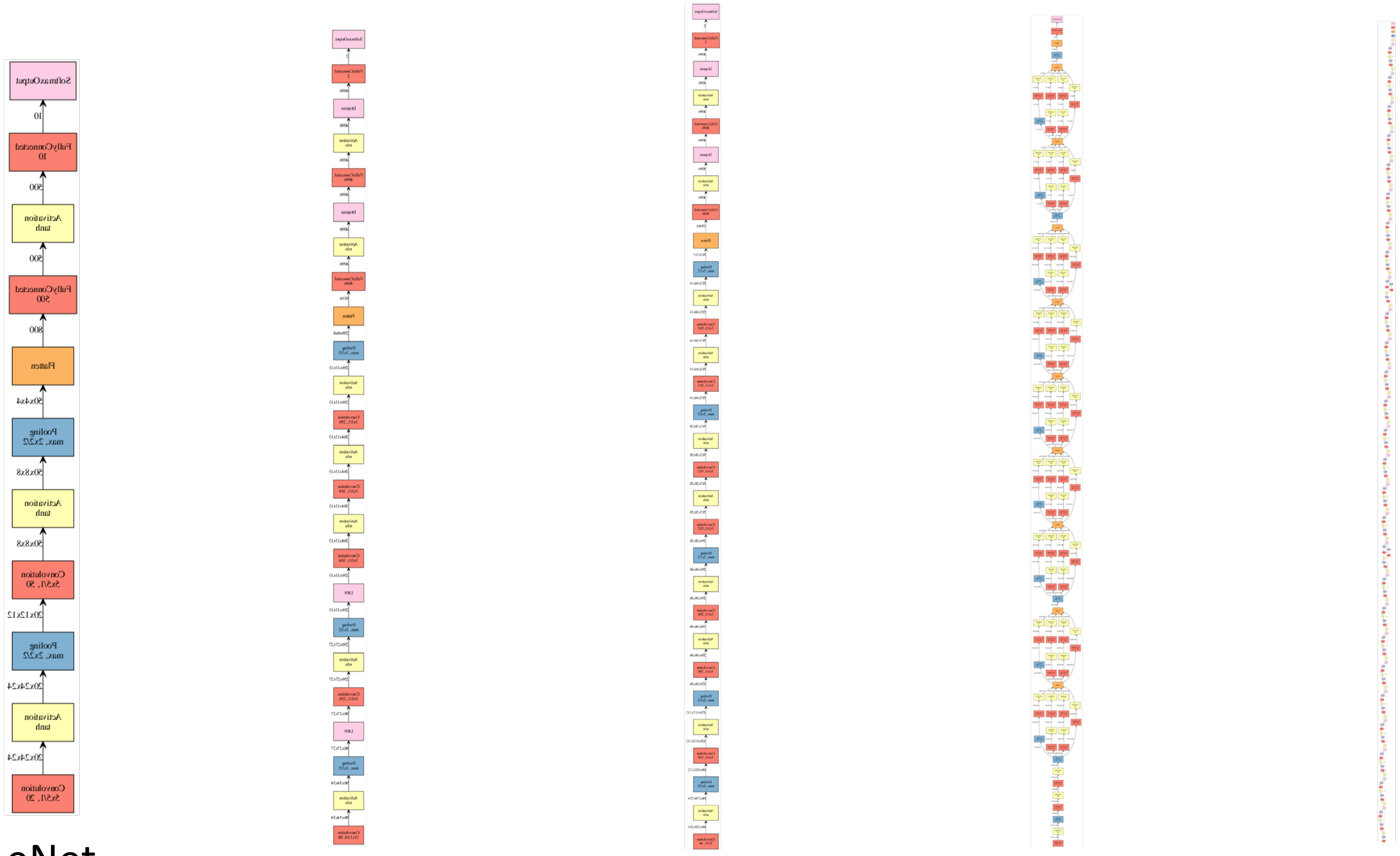
- Only 5 million params!

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)



LeNet
(5 layers)

AlexNet
(8 layers)

VGGNet
(19 layers)

GoogleNet

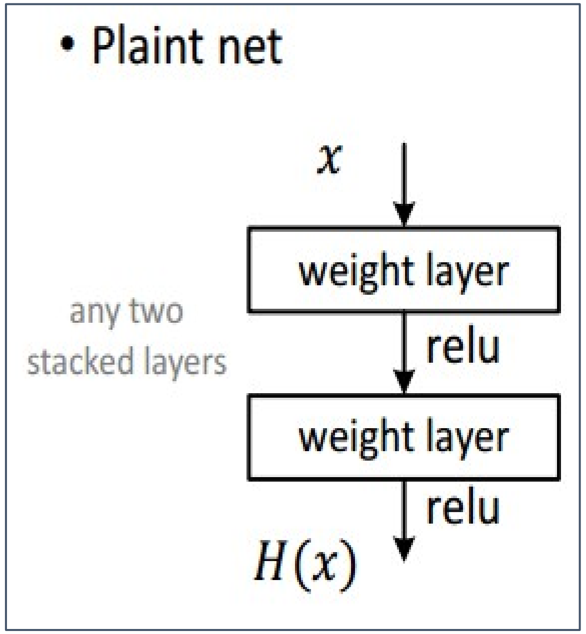
ResNet
(152 layers)

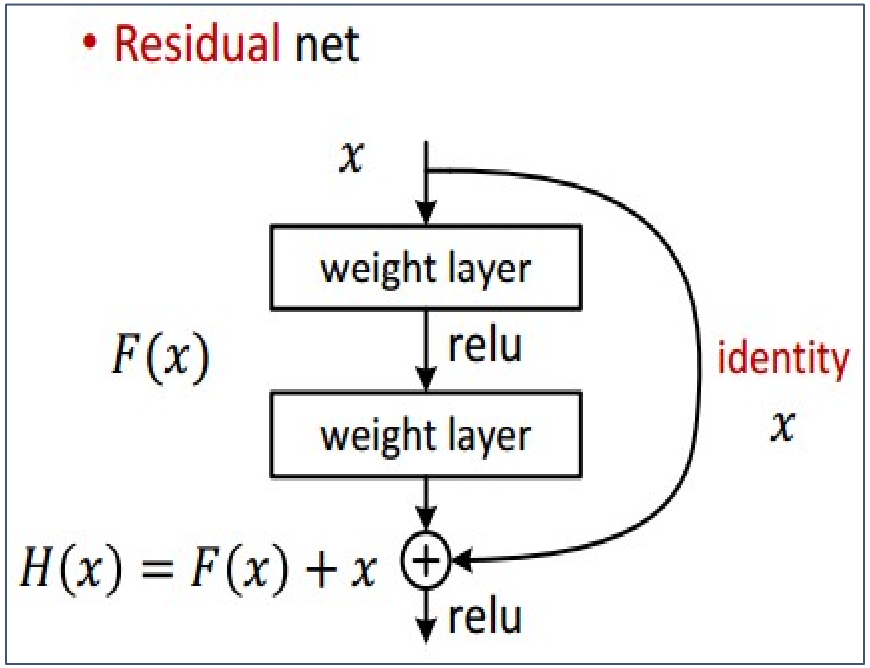
2-3 weeks of training on 8 GPU machine

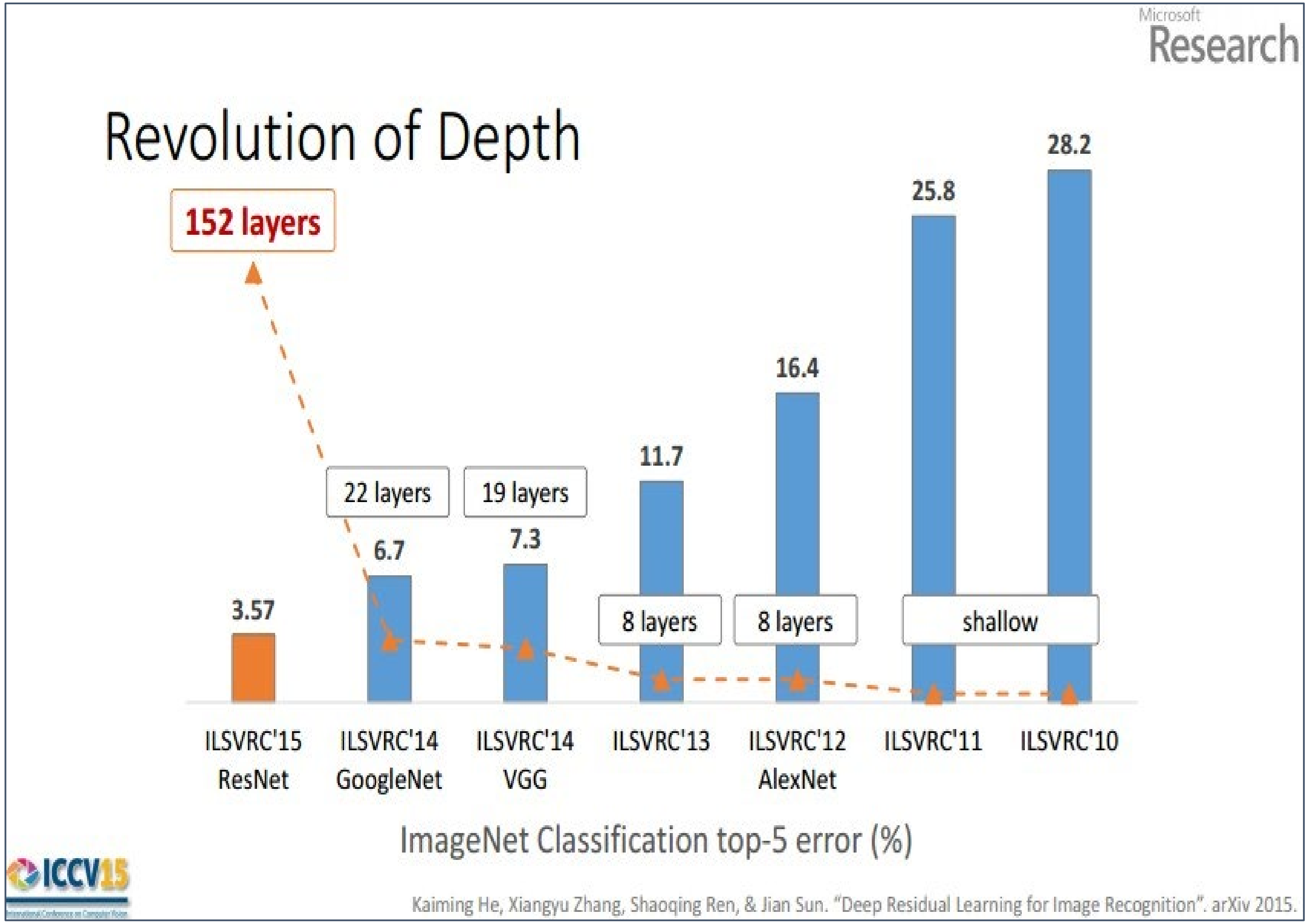
at runtime: faster than a VGGNet! (even though it has 8x more layers)

Case Study: ResNet

[He et al., 2015]







(slide from Kaiming He)

Further Reading

- Stanford CS231n, lecture 5, Convolutional Neural Networks http://cs231n.stanford.edu/schedule.html
- Deep learning with PyTorch https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
- AlexNet (2012): https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html
- Vgg16 (2014): https://arxiv.org/abs/1409.1556
- GoogleNet (2014): https://arxiv.org/abs/1409.4842
- ResNet (2015): https://arxiv.org/abs/1512.03385