Introduction to Convolutional Neural Networks

Karel Horák

Computer Vision Group
Department of Control and Instrumentation
Faculty of Electrical Engineering and Communications
Brno University of Technology

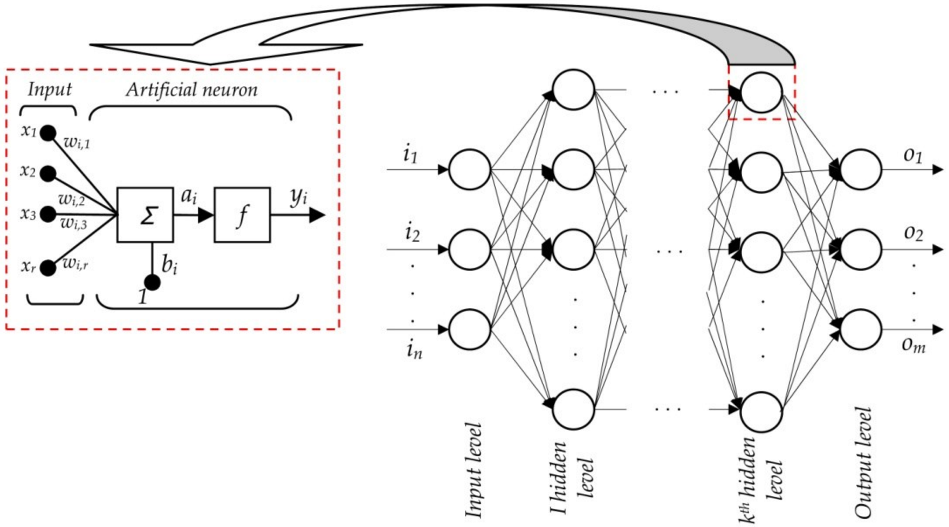
Contents:

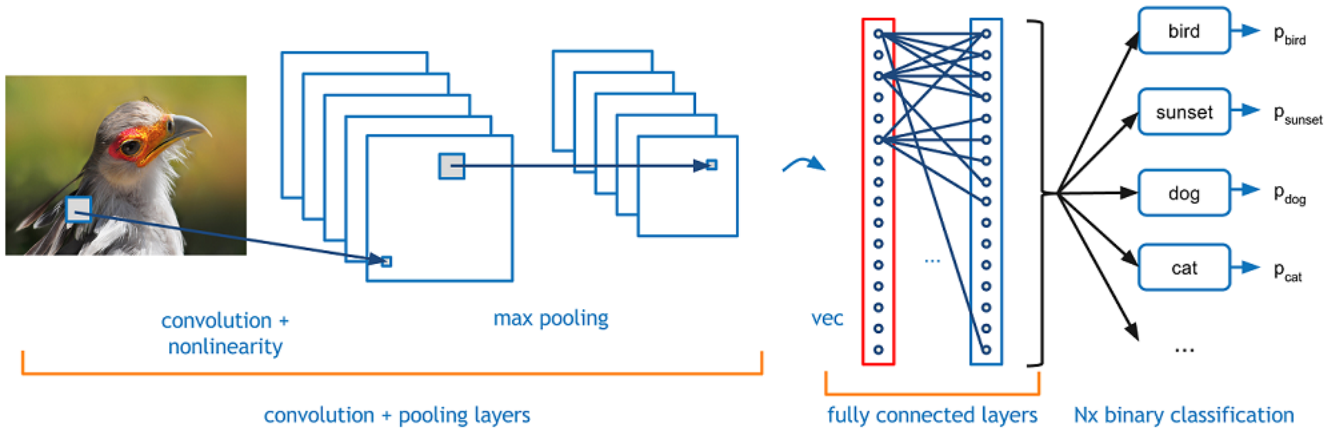
Slides courtesy of: K. Horak, BUT
V. Kalogeiton, INRIA
F.F. Li, Stanford

NN vs. CNN

NN vs. CNN

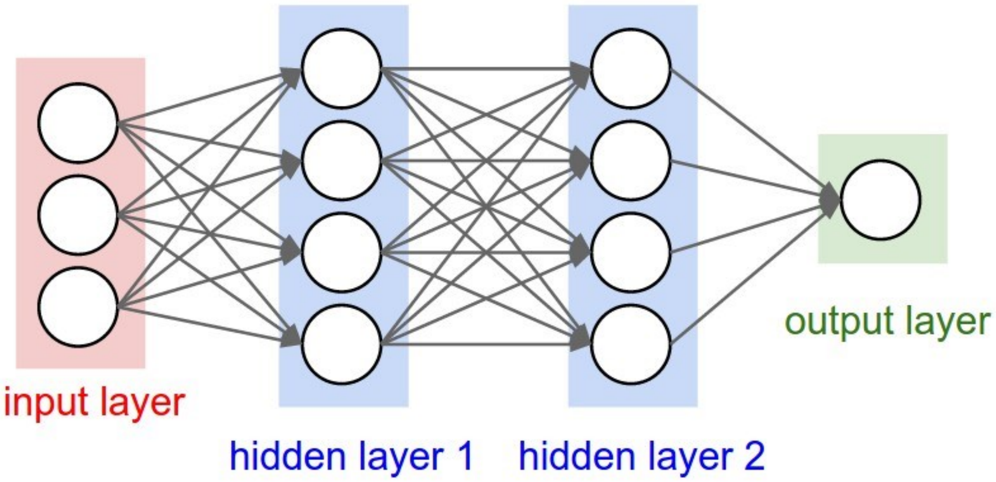
- Neural Network: a vector serves as an input
- Convolutional Neural Network: a volume (e.g. multichannel image) serves as an input

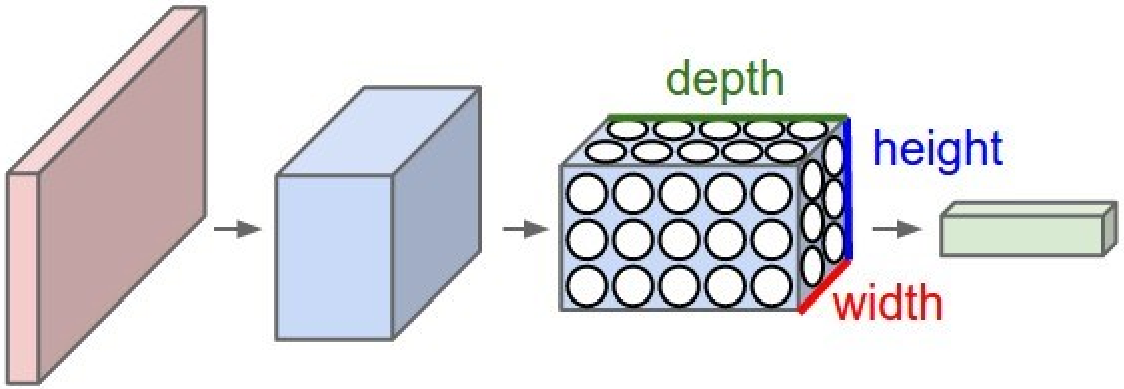




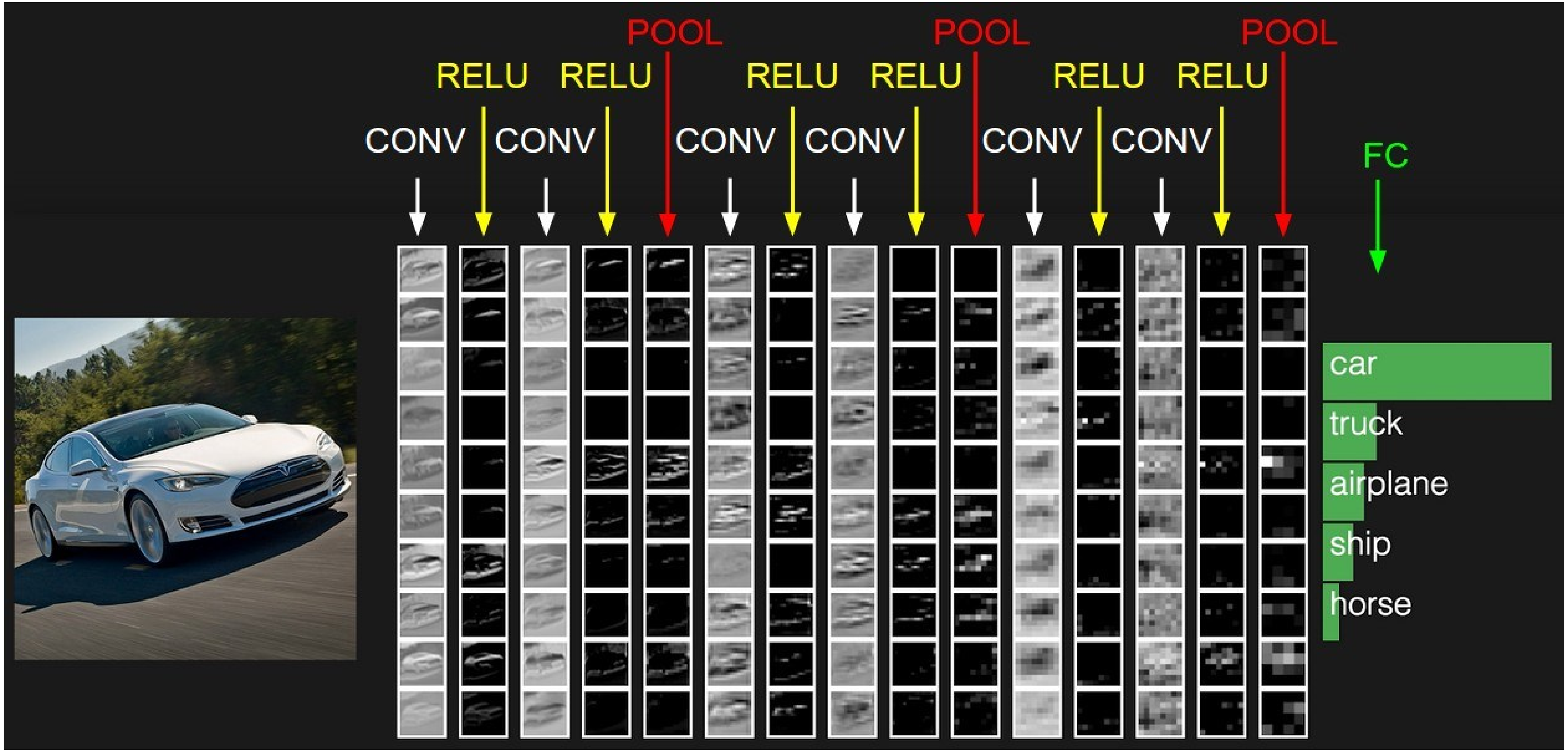
NN vs. CNN

- Weights between neurons of NN represent analogy to convolutional filters of CNN.
- Filters in one convolutional layer are mutually independent.
- Finally, what is the difference between NN or CNN and RNN (recurrent)?





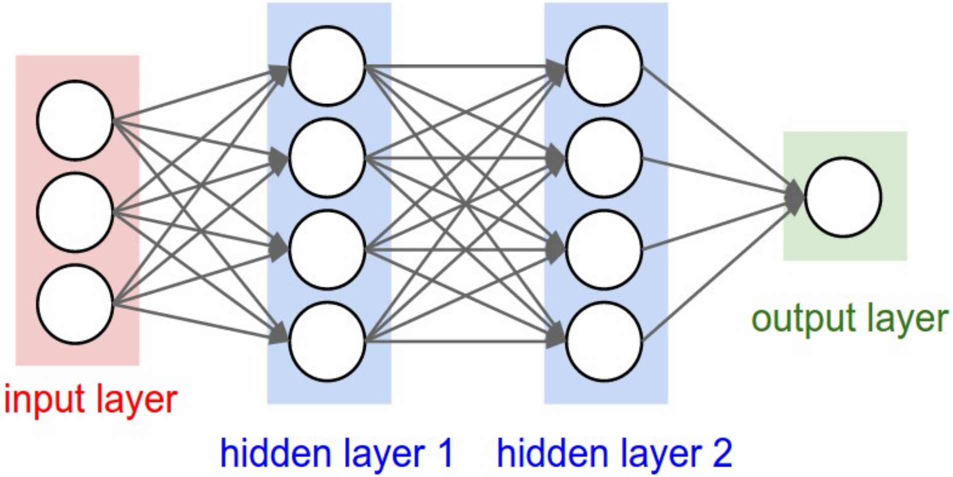
Convolutional Neural Network Layers



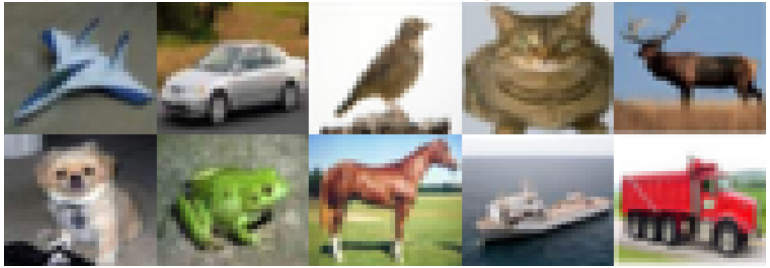
What are CNNs ?

CNN = Neural Network with a convolution operation
instead of matrix multiplication
in at least one of the layers

Neural Networks



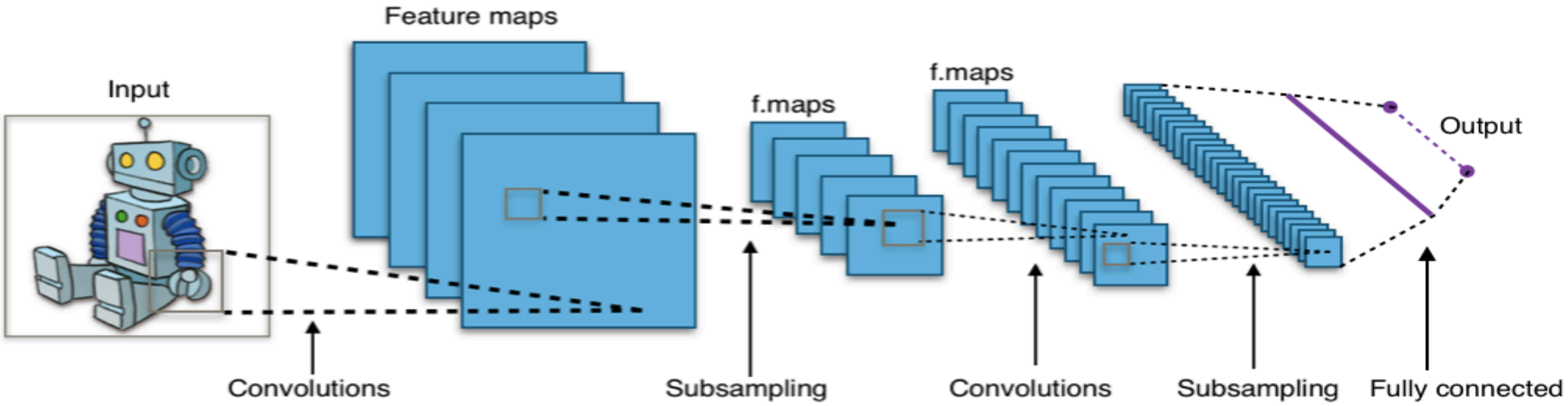
Input example : one image



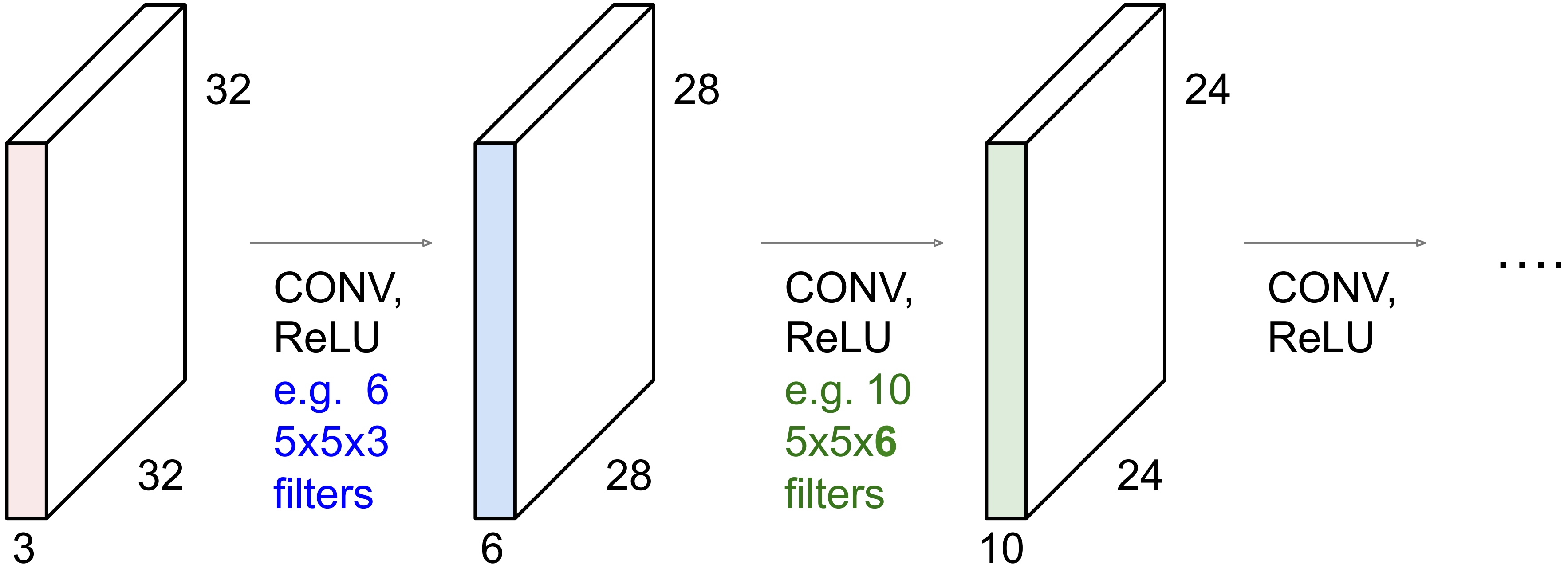
Output example : one class

airplane	dog
automobile	frog
bird	horse
cat	ship
deer	truck

A typical CNN architecture

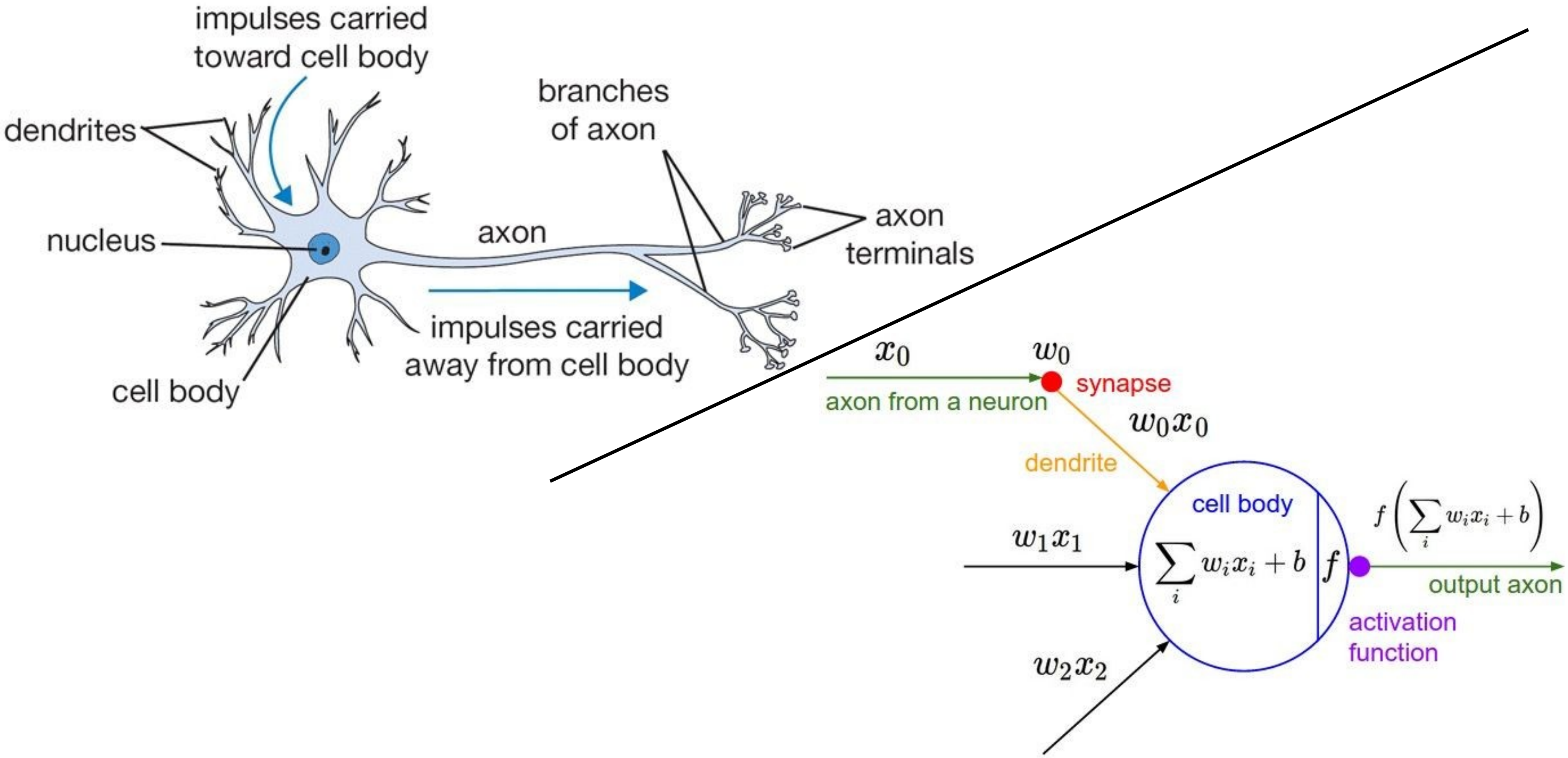


Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

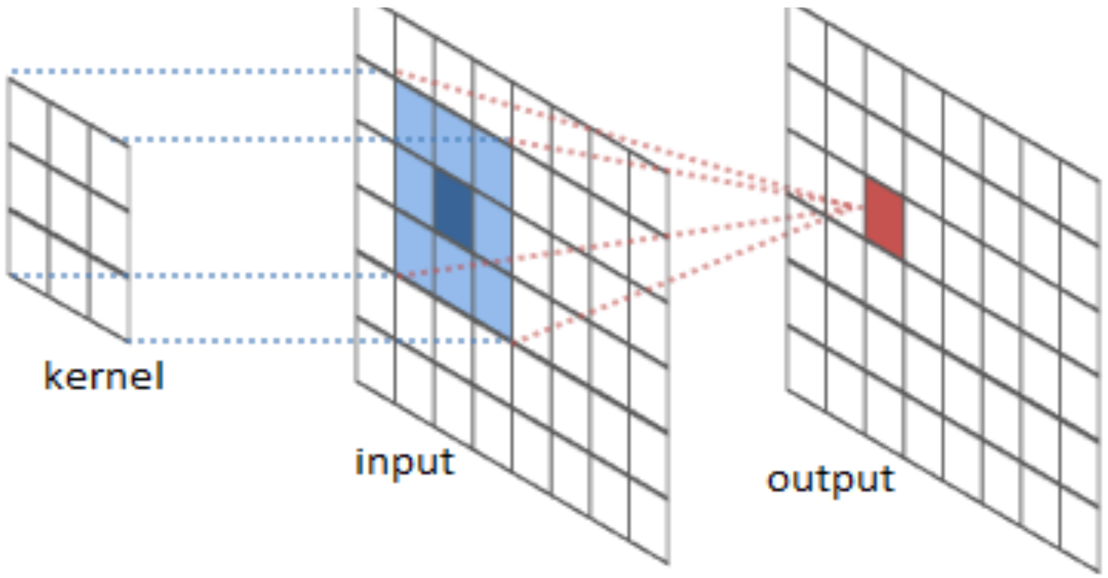
Biological neuron & mathematical model



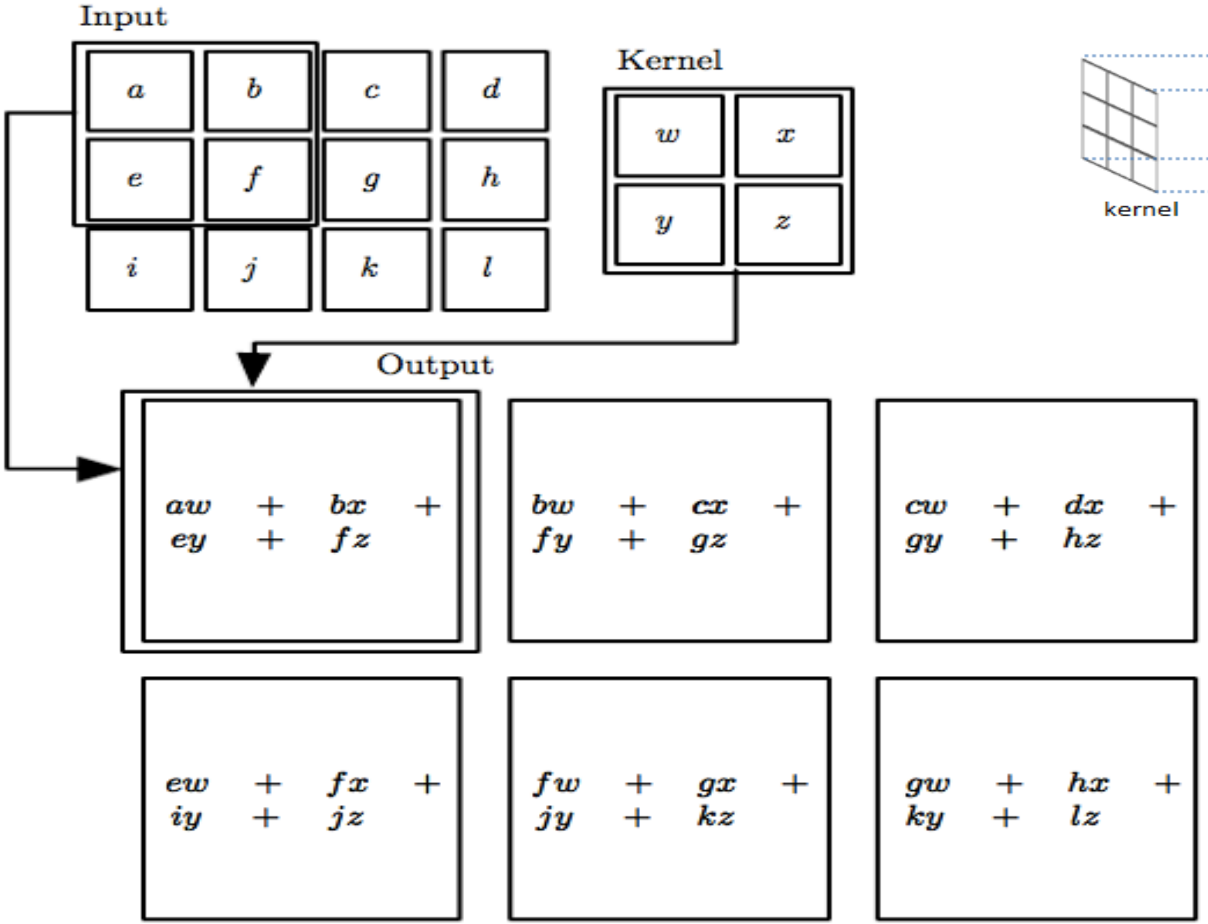
slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

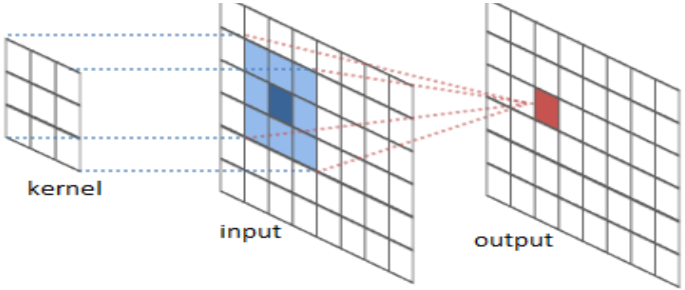
Convolution

The convolution operation



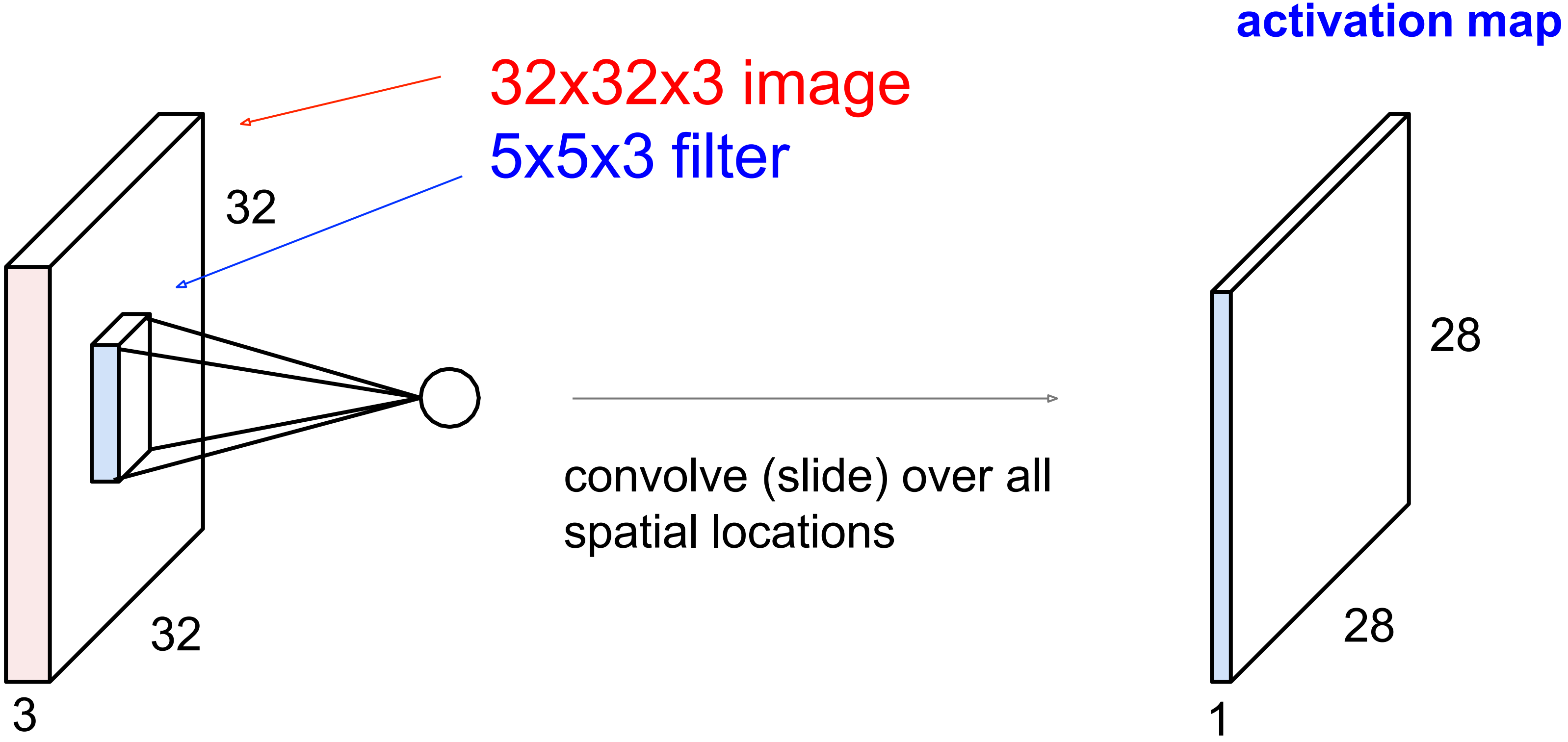
The convolution operation





Convolution Layers

Convolution Layer



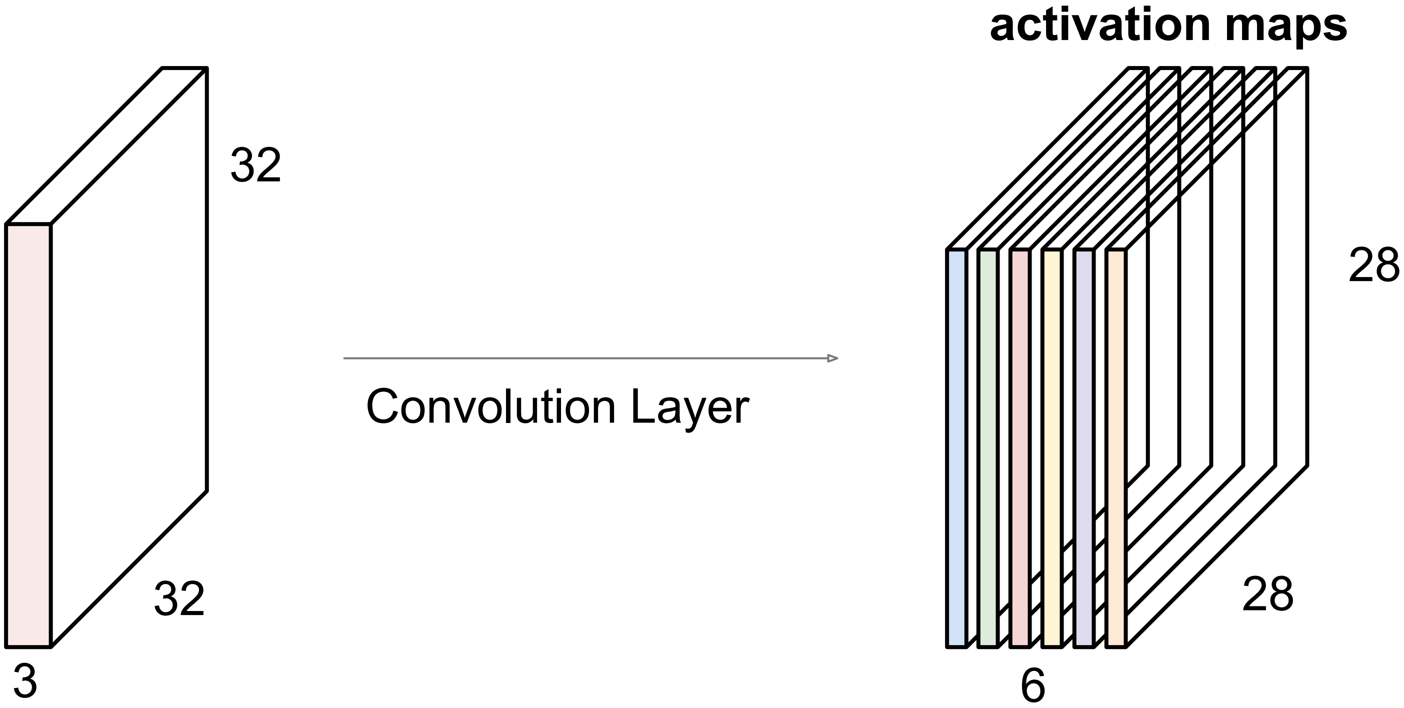
slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Convolution Layer

consider a second, green filter

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

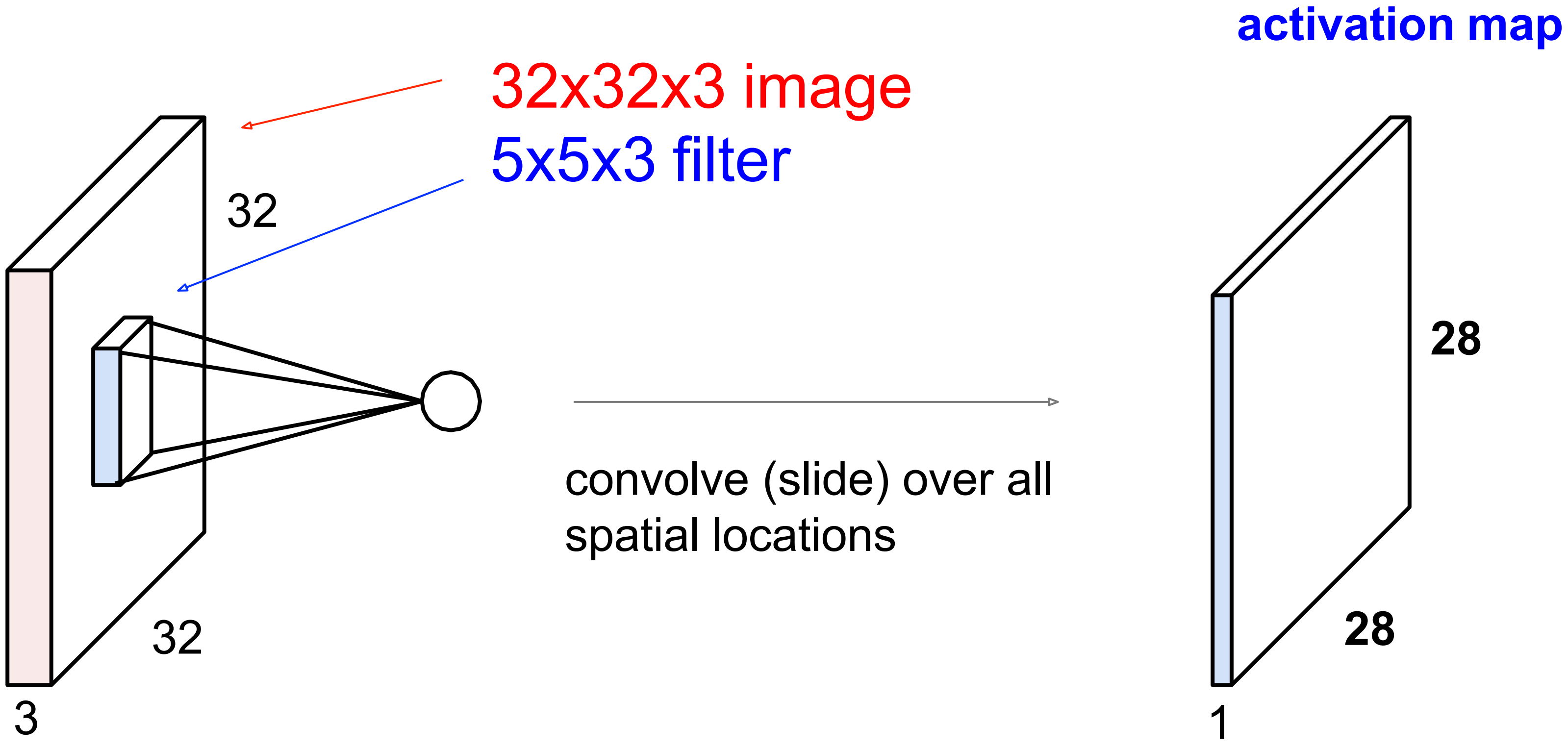
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

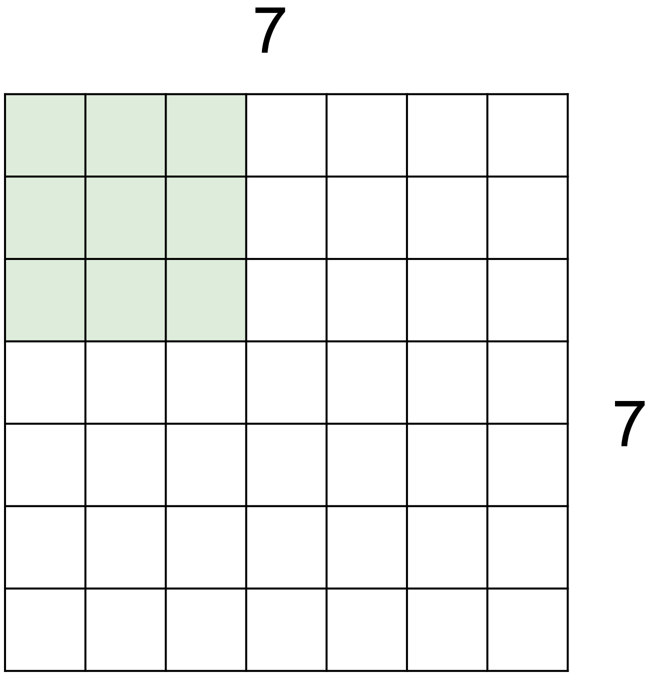
slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

A closer look at spatial dimensions:



slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

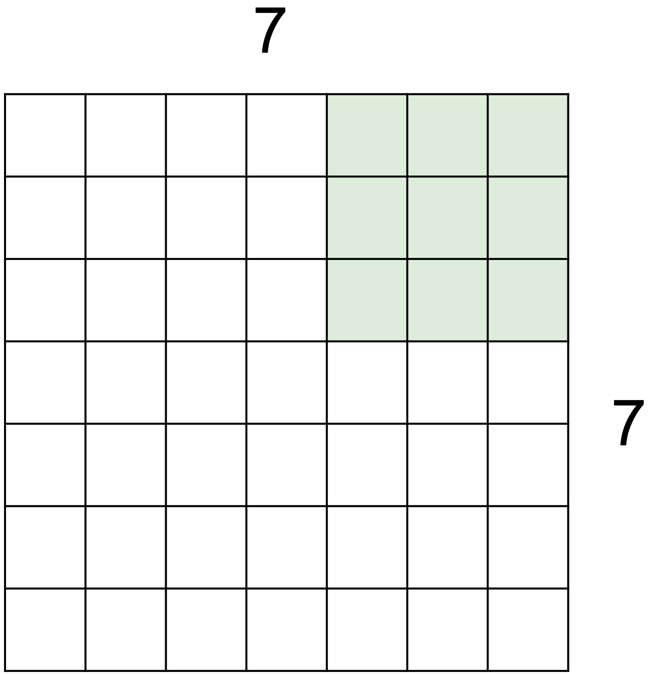
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

A closer look at spatial dimensions:

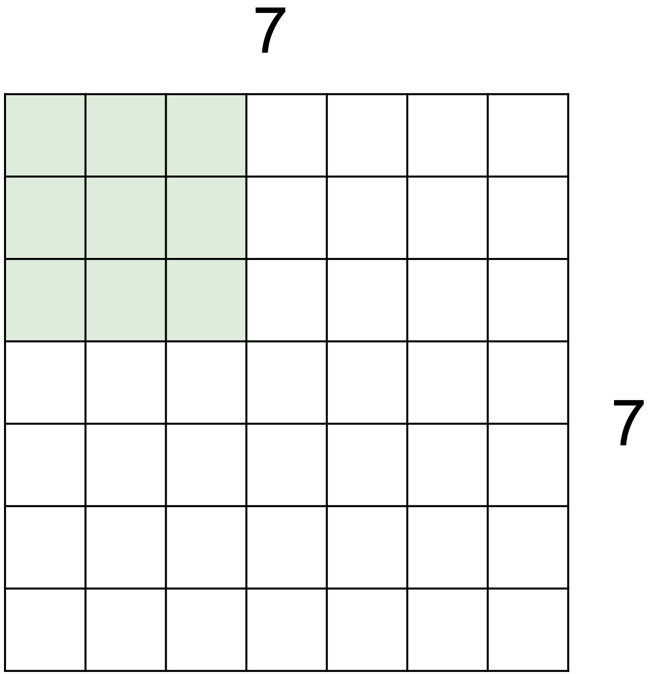


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

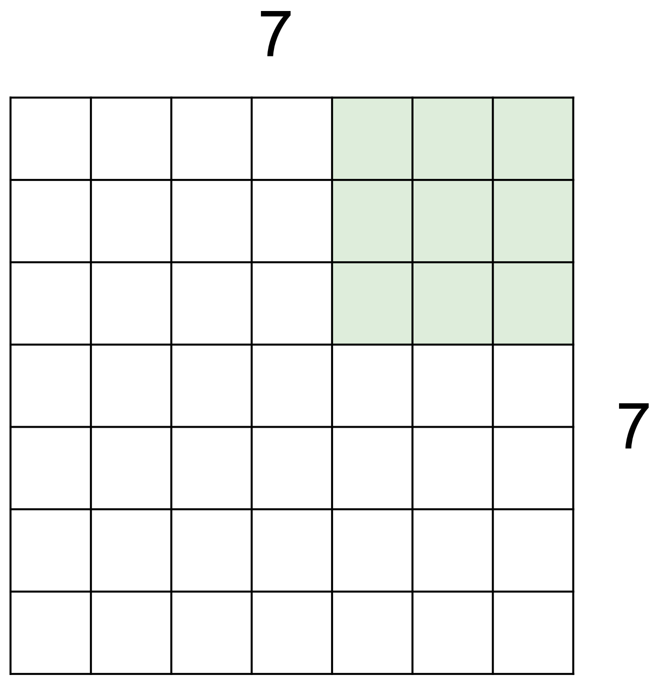
A closer look at spatial dimensions:



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

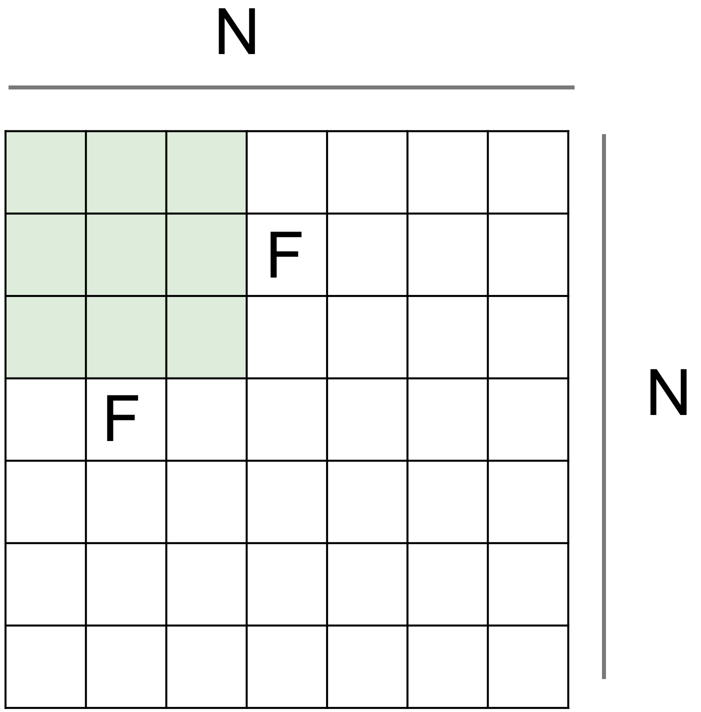
slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

A closer look at spatial dimensions:



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

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

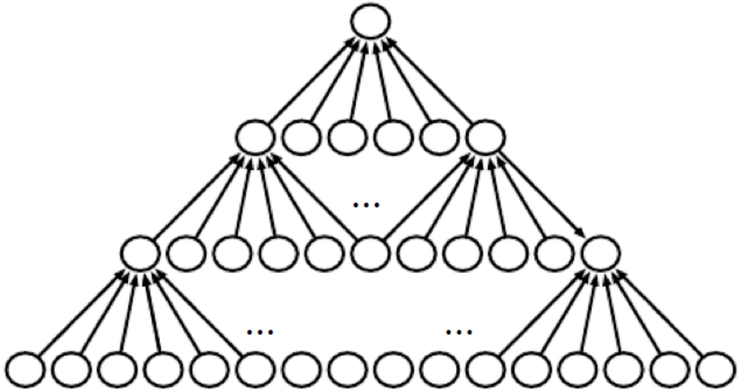


Output size:
(N - F) / stride + 1

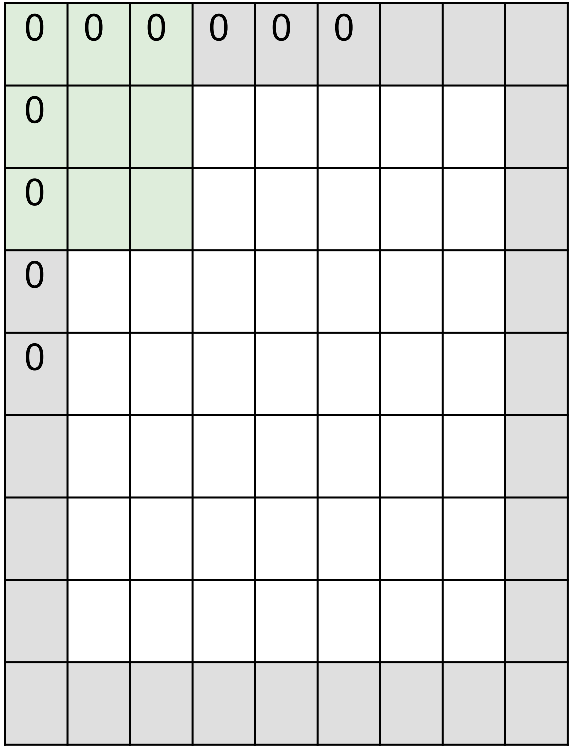
e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Zero-Padding



Zero-Padding: common to the border



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

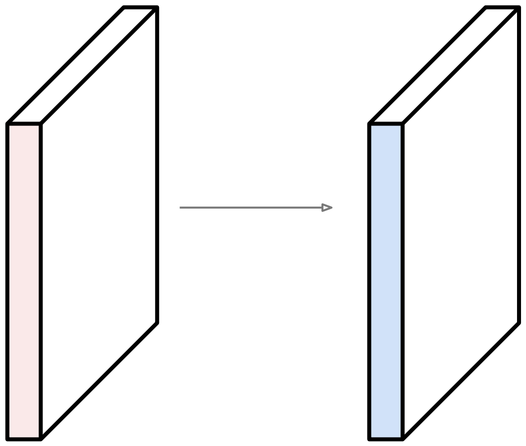
7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2



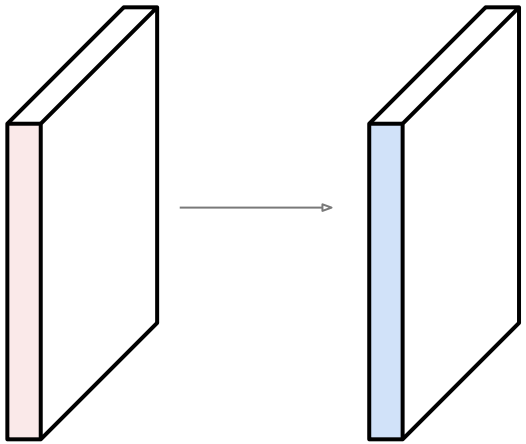
Output volume size:

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

32x32x10

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

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**

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

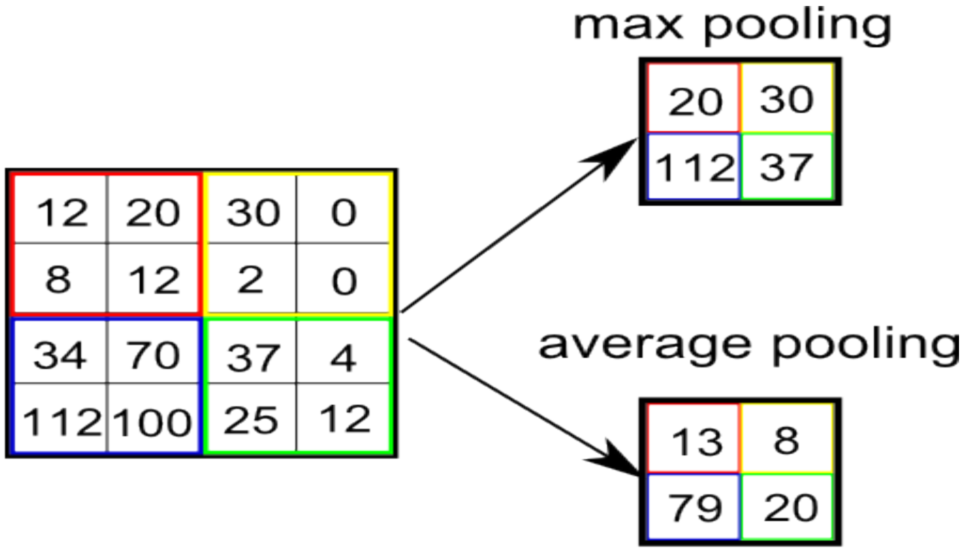
Summary. To summarize, the Conv Layer:

- 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.

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

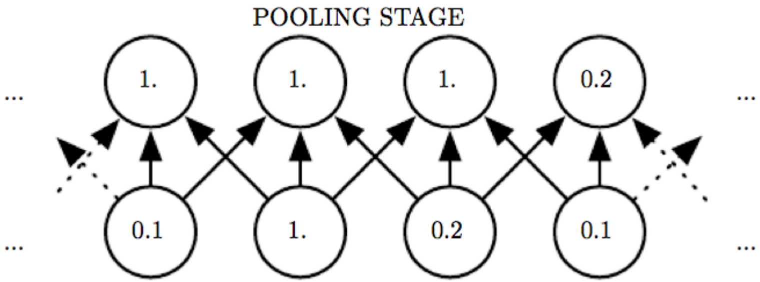
Pooling

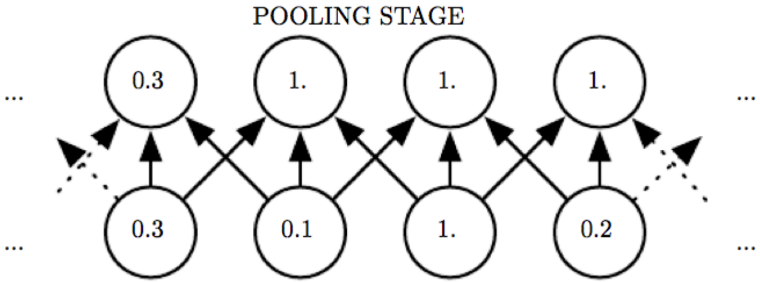
Pooling



Effect = invariance to small translations of the input

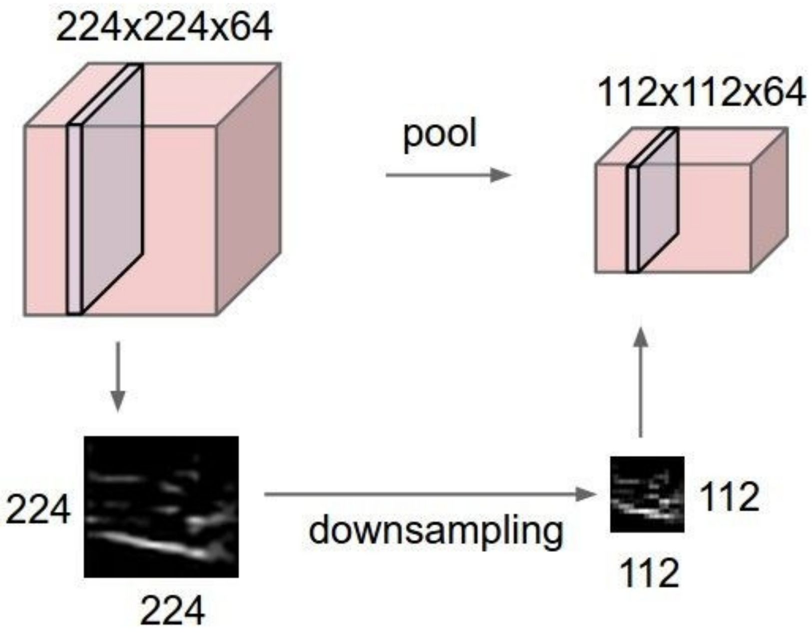
Pooling





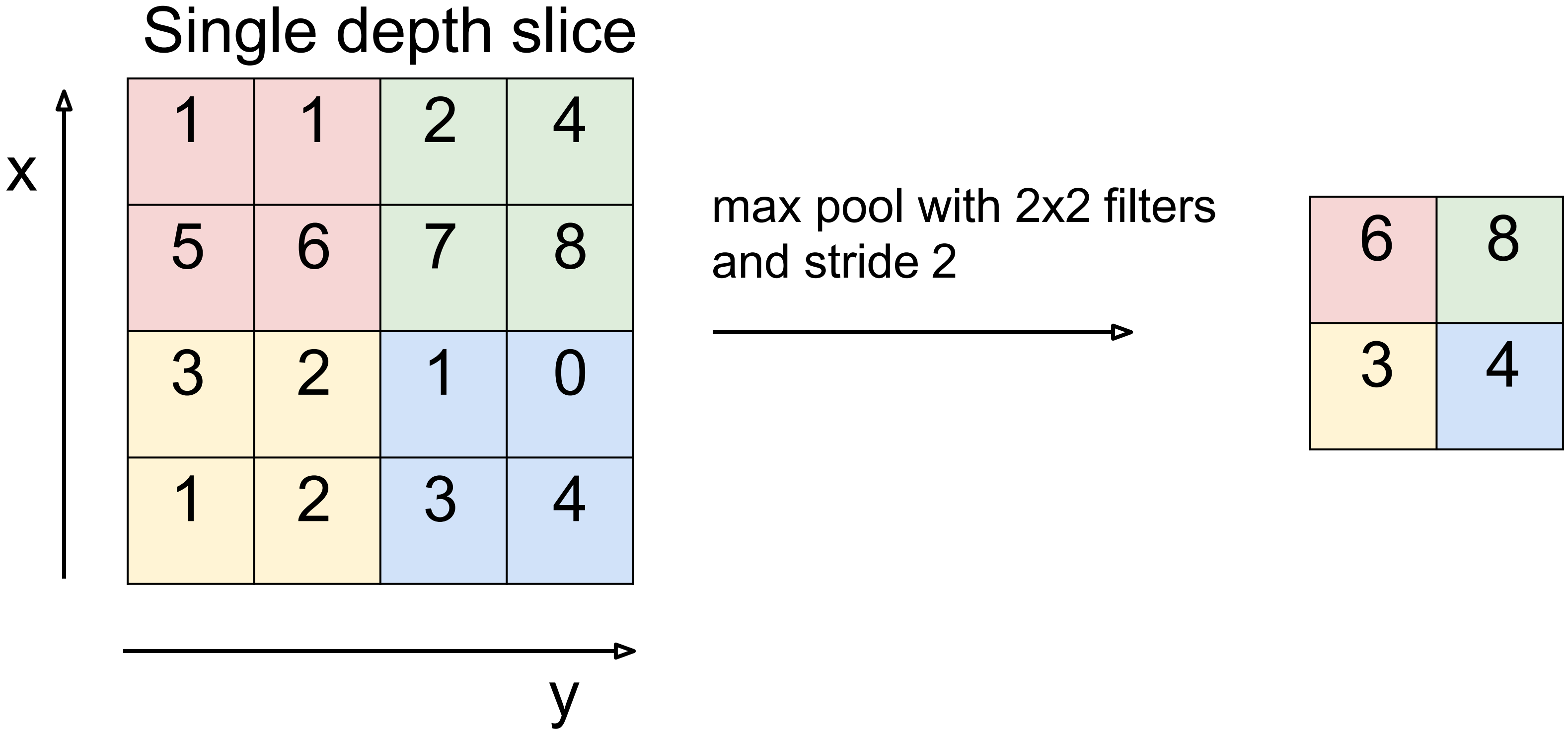
Pooling

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



slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Max Pooling



slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Summary

- 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

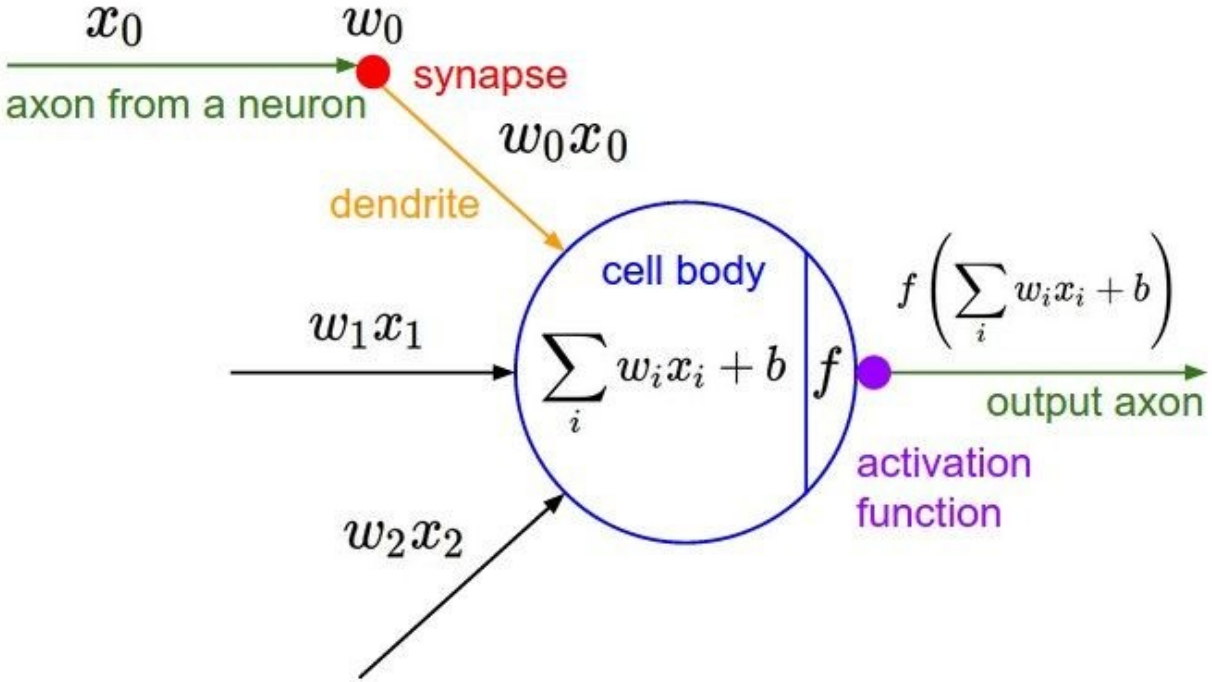
Common settings:

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

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Activation function

Activation Functions

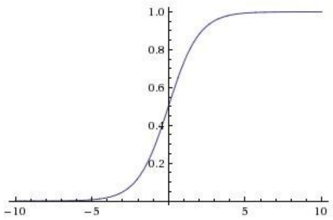


slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

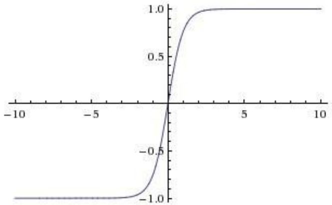
Activation Functions

Sigmoid

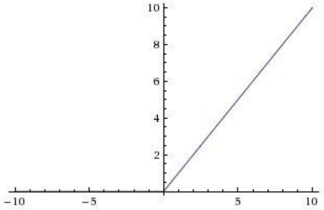
$$\sigma(x) = 1/(1+e^{-x})$$



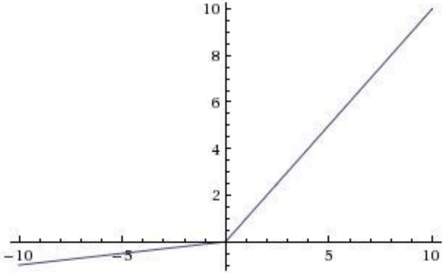
tanh tanh(x)



ReLU max(0,x)



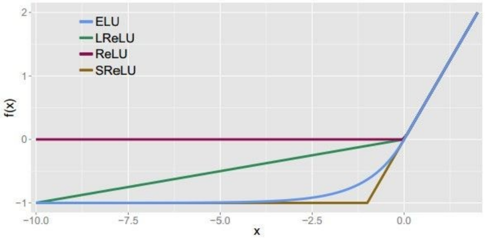
Leaky ReLU



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

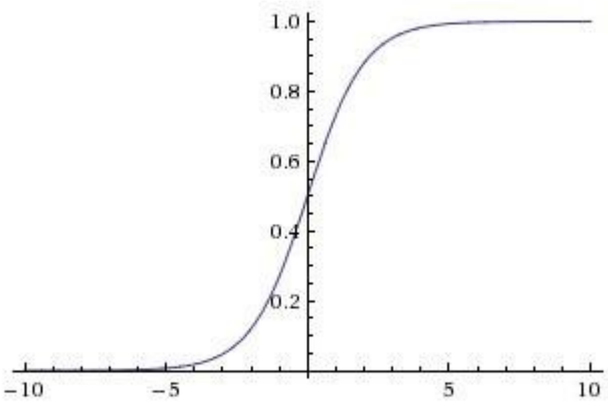
ELU

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



slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Activation Functions



Sigmoid

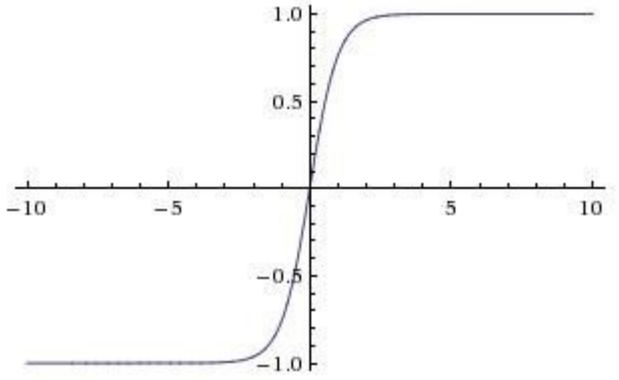
$$\sigma(x) = 1/(1 + e^{-x})$$

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

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

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Activation Functions



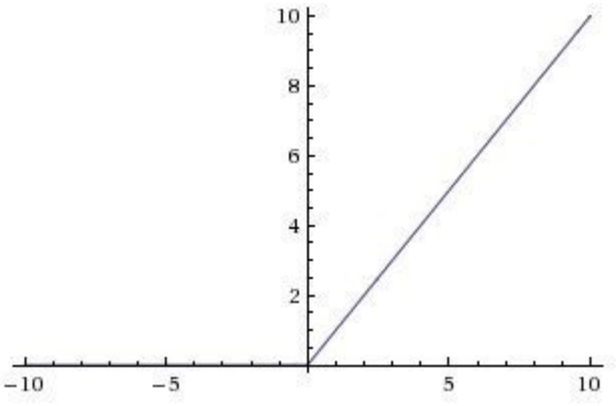
tanh(x)

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

[LeCun et al., 1991]

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Activation Functions



ReLU
(Rectified Linear Unit)

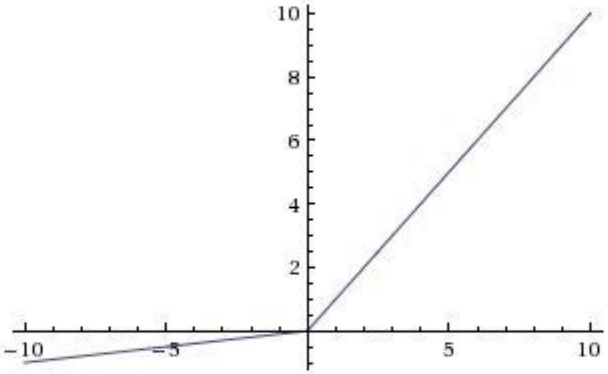
Computes **f(x) = max(0,x)**

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)

- Not zero-centered output
- ReLU units can “die”

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Activation Functions



Leaky ReLU

$$f(x) = \max(0.01x, x)$$

- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not “die”.**

[Mass et al., 2013] [He et al., 2015]

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

In practice

- Use ReLU. Be careful with your learning rates
- Try out Leaky ReLU / Maxout / ELU
- Try out tanh but don’t expect much
- Don’t use sigmoid

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Weights initialization

Weights initialization

- If the weights in a network start too small,
  then the signal shrinks as it passes through each layer until it's too tiny to be useful.
- If the weights in a network start too large,
  then the signal grows as it passes through each layer until it's too massive to be useful.

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Weights initialization

- All zero initialization

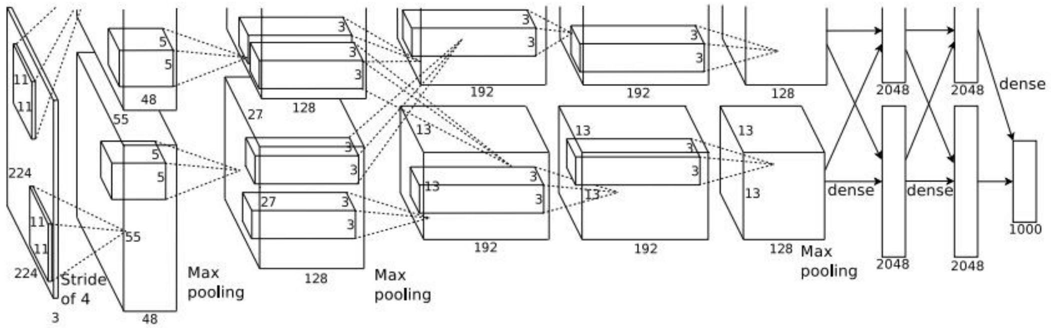
- Small random numbers

- Draw weights from a Gaussian distribution
  with standard deviation of sqrt(2/n),
  where n is the number of outputs to the neuron

AlexNet example

Case Study: AlexNet

[Krizhevsky et al. 2012]



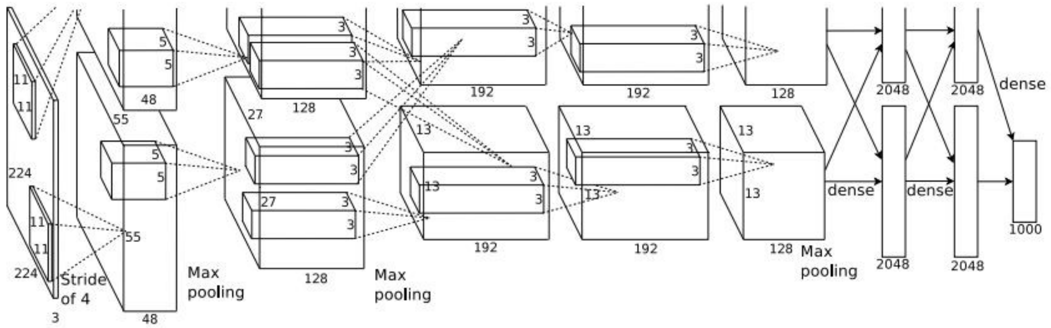
Input: 227x227x3 images

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

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Case Study: AlexNet

[Krizhevsky et al. 2012]



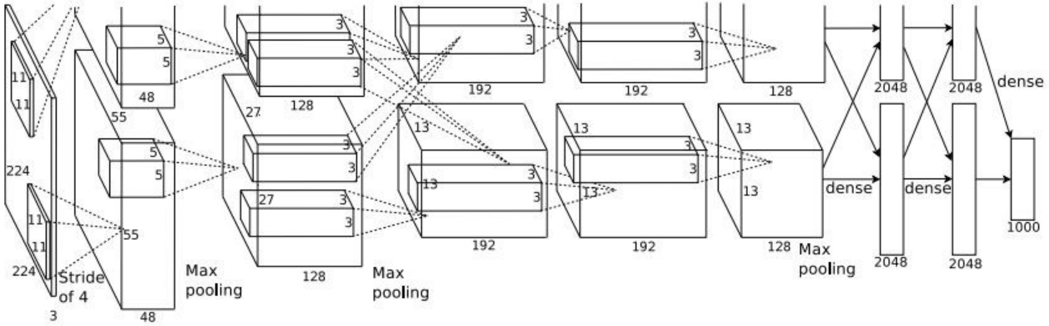
Input: 227x227x3 images
After CONV1: 55x55x96

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

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

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%

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson