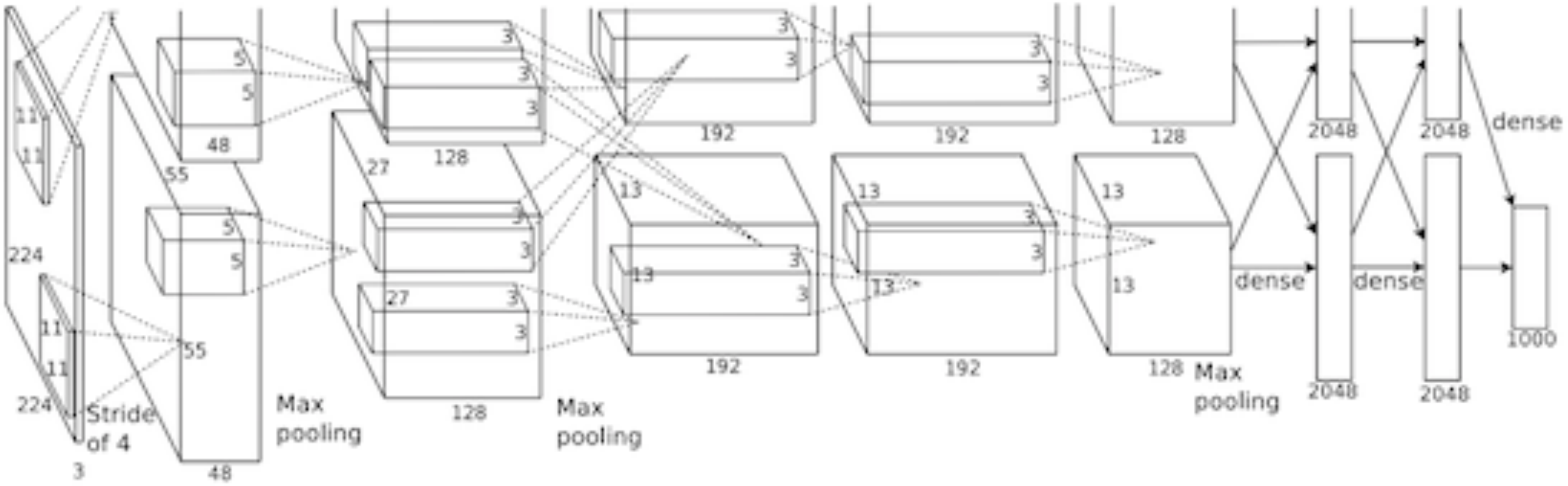
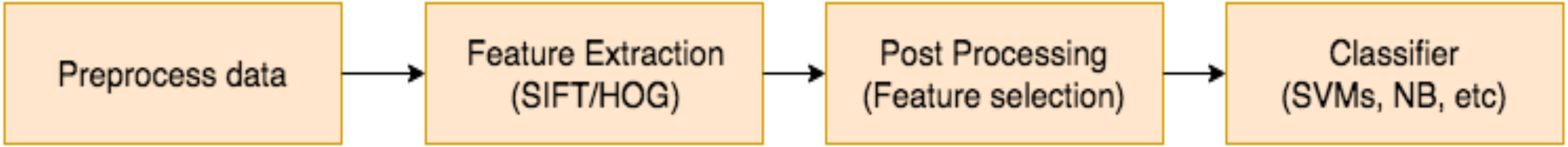


Convolutional Neural Networks & Deep Learning

Pre deep learning era



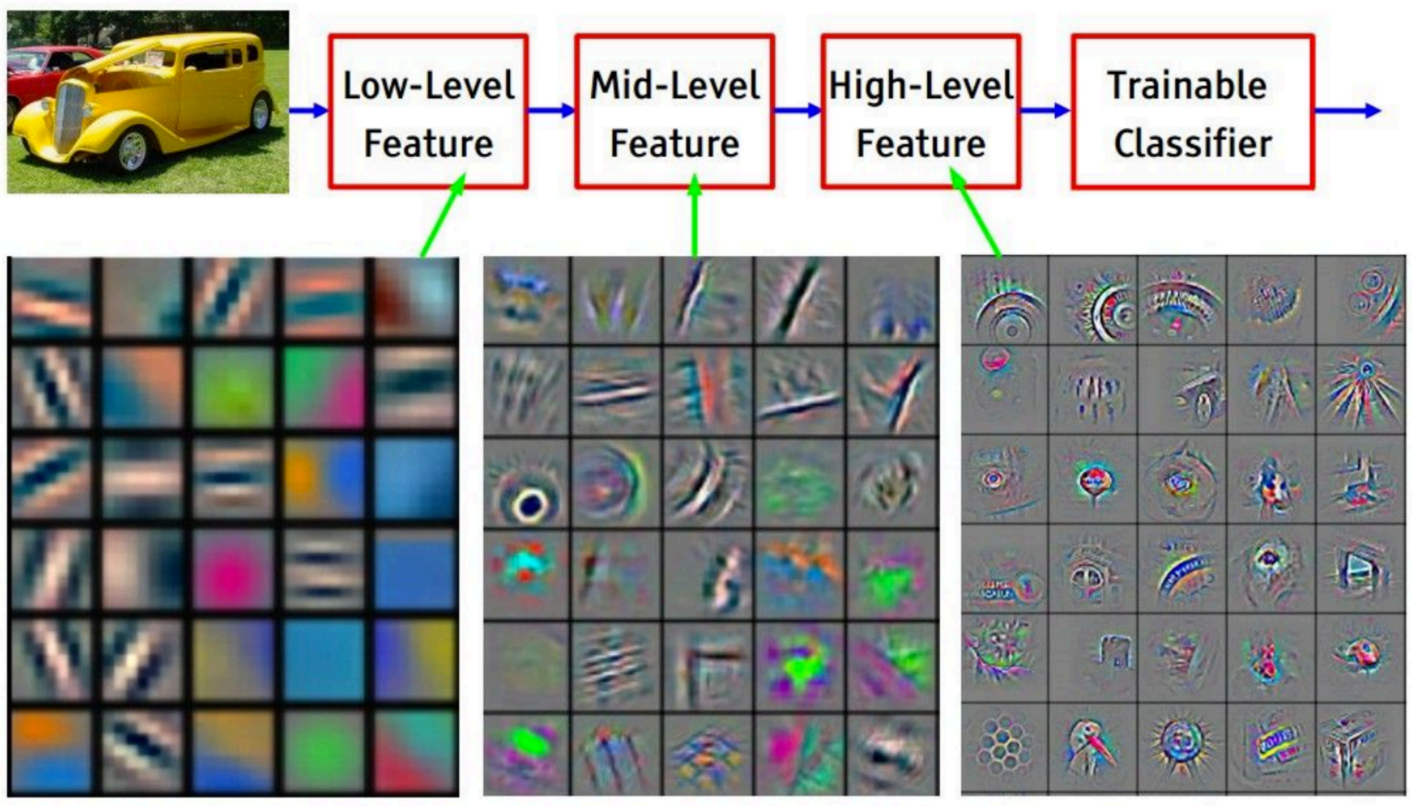
Cons:

- Hand crafted features are difficult to engineer!
- Time consuming process.
- Which set of features maximizes accuracy?
- Tends to overfit.

What is Deep Learning?

Composition of **non-linear** transformation of data

Why “deep”? Find **complex** patterns by learning **hierarchical** features



But deep learning is simple!

- Deep Learning builds an **end-to-end** recognition system.
- Non linear transformation of raw pixels directly to labels.
- Build a complex non-linear system by combining 4 simple **building blocks**.

Convolutions

Pooling

Activation functions

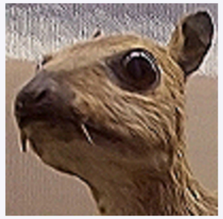
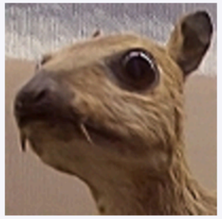
Softmax

Convolutions

Operation	Kernel	Image result
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Convolutions

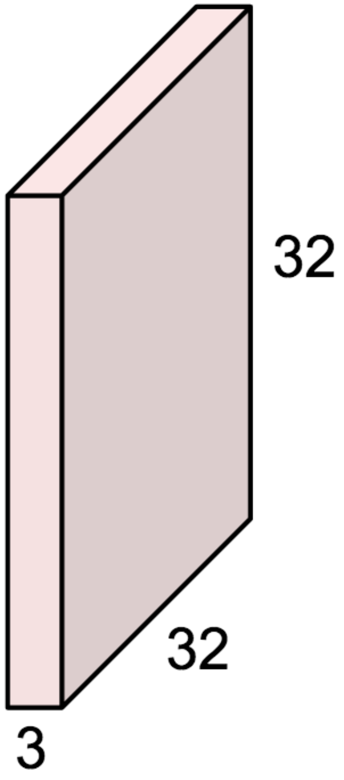
HW-1!

Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

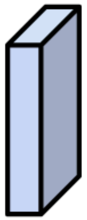
Convolutions – In deep learning

Filters always extend the full depth of the input volume

32x32x3 image



5x5x3 filter

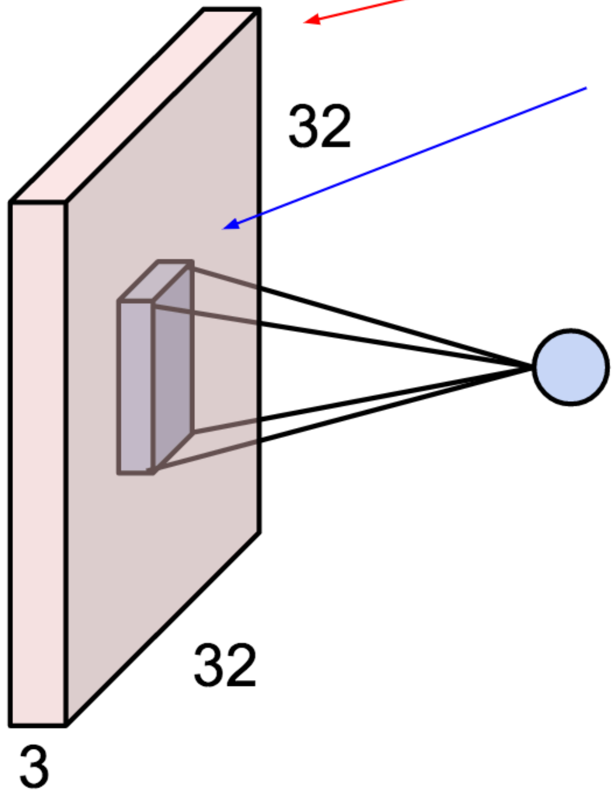


We need to learn these filters.

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

Figure from S-17 16-824 CMU

Convolutions – In deep learning



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)

$$w^T x + b$$

Figure from S-17 16-824 CMU

Convolutions – In deep learning

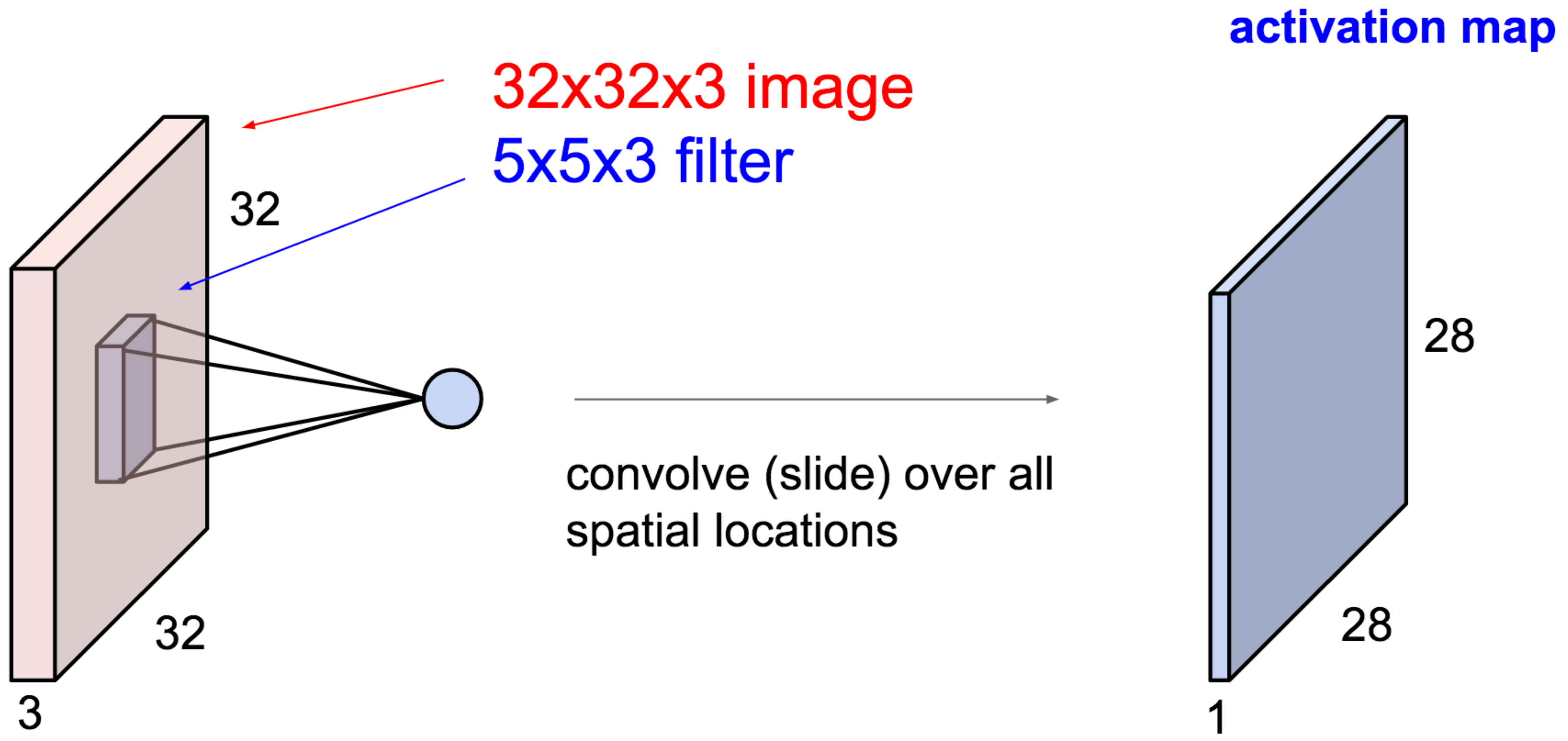


Figure from S-17 16-824 CMU

Convolutions – In deep learning

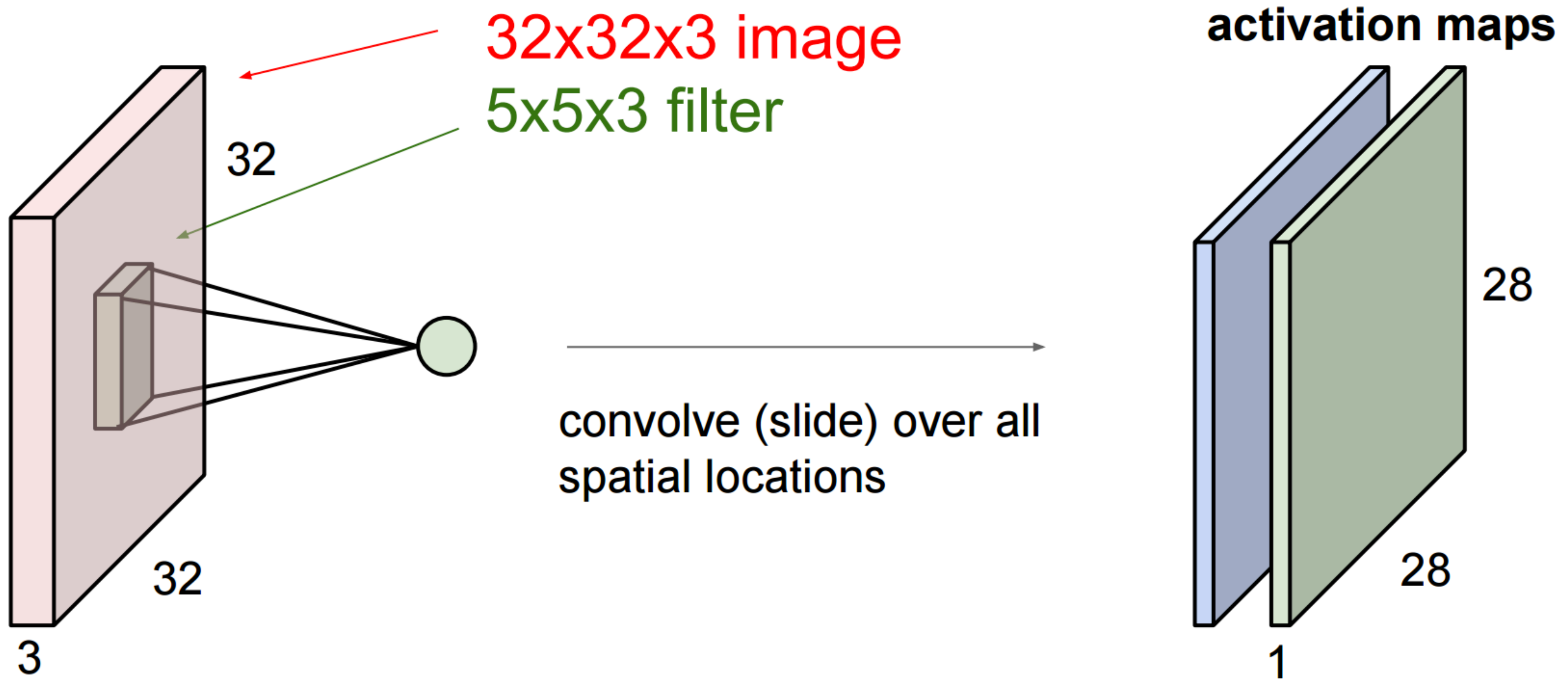
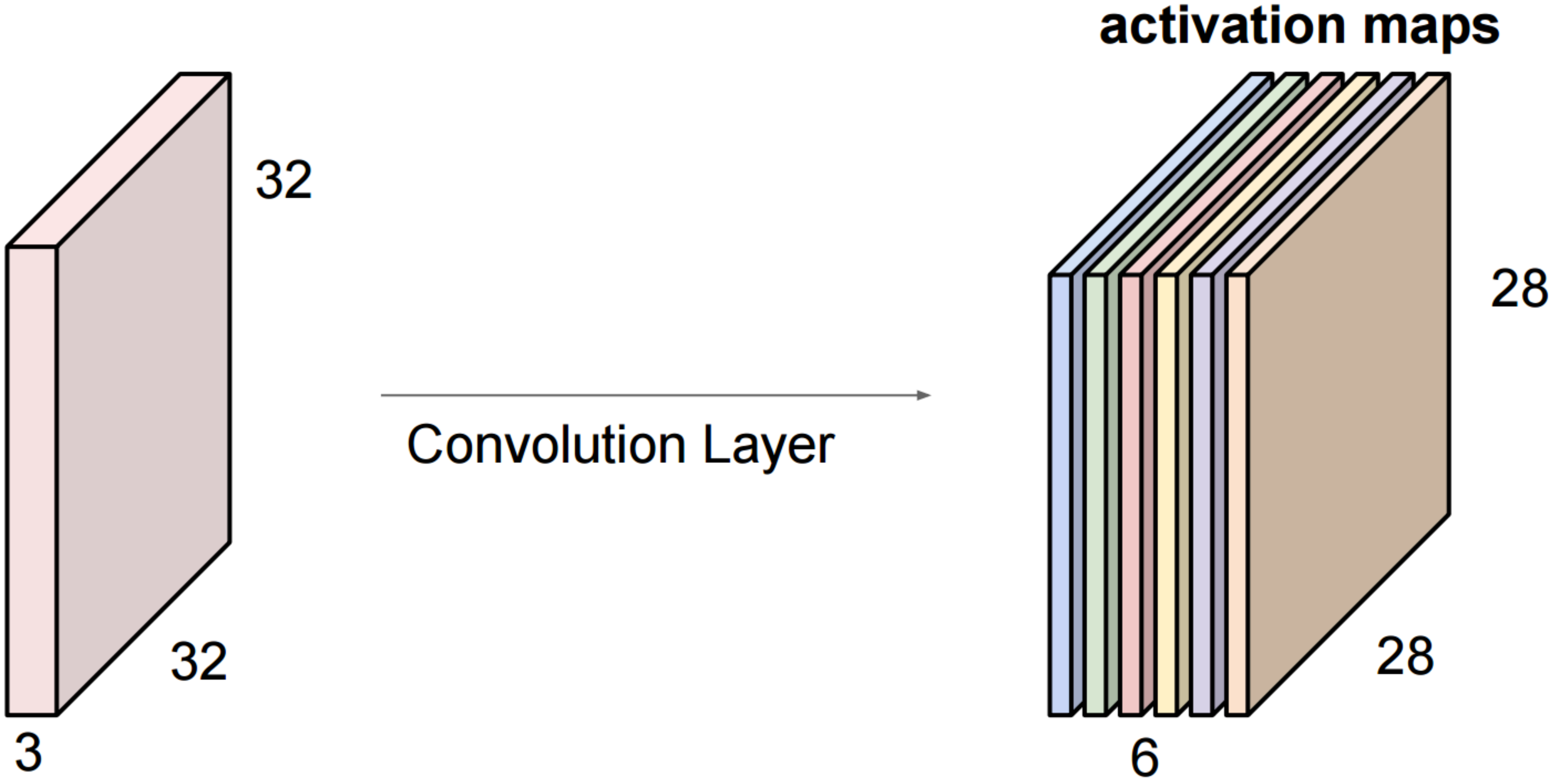
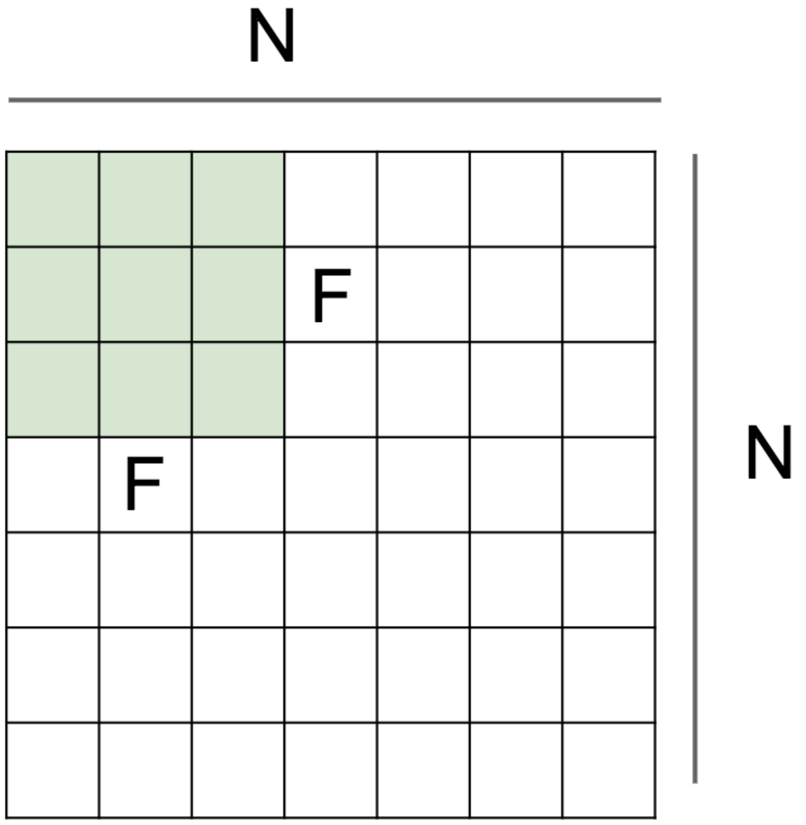


Figure from S-17 16-824 CMU

Convolutions – In deep learning



Convolution – Spatial Dimensions



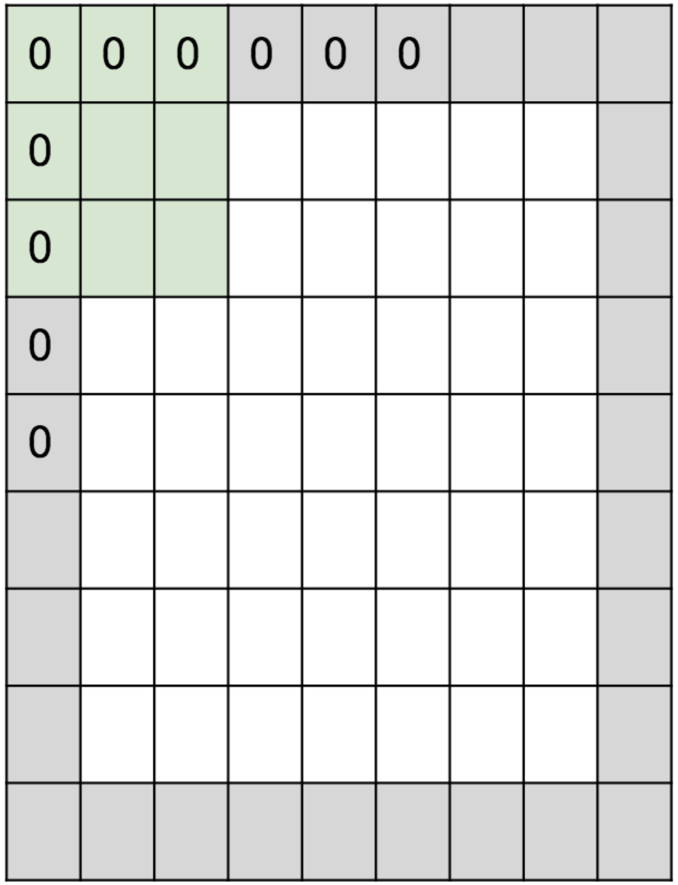
Output size:
(N - F) / stride + 1

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 :\

Figure from S-17 16-824 CMU

Convolution – Spatial Dimensions

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

Figure from S-17 16-824 CMU

Convolution : Example

Why not use FCs for learning image features?

- Huge number of parameters in Fully connected network.
- Full connectivity is wasteful. Leads to overfitting.
- (200x200x3) x 5 neurons = 120,000x5 parameters in FC!
- No spatial relation in FCs.
- Just learn several filters (weights in CNNs).
- 5x5x100 = 2500 parameters for learning 100 filters in CNNs.

Max-pooling

- Non-linear down sampling.
- Input is partitioned into non-overlapping patches and maximum value in each partition is chosen.

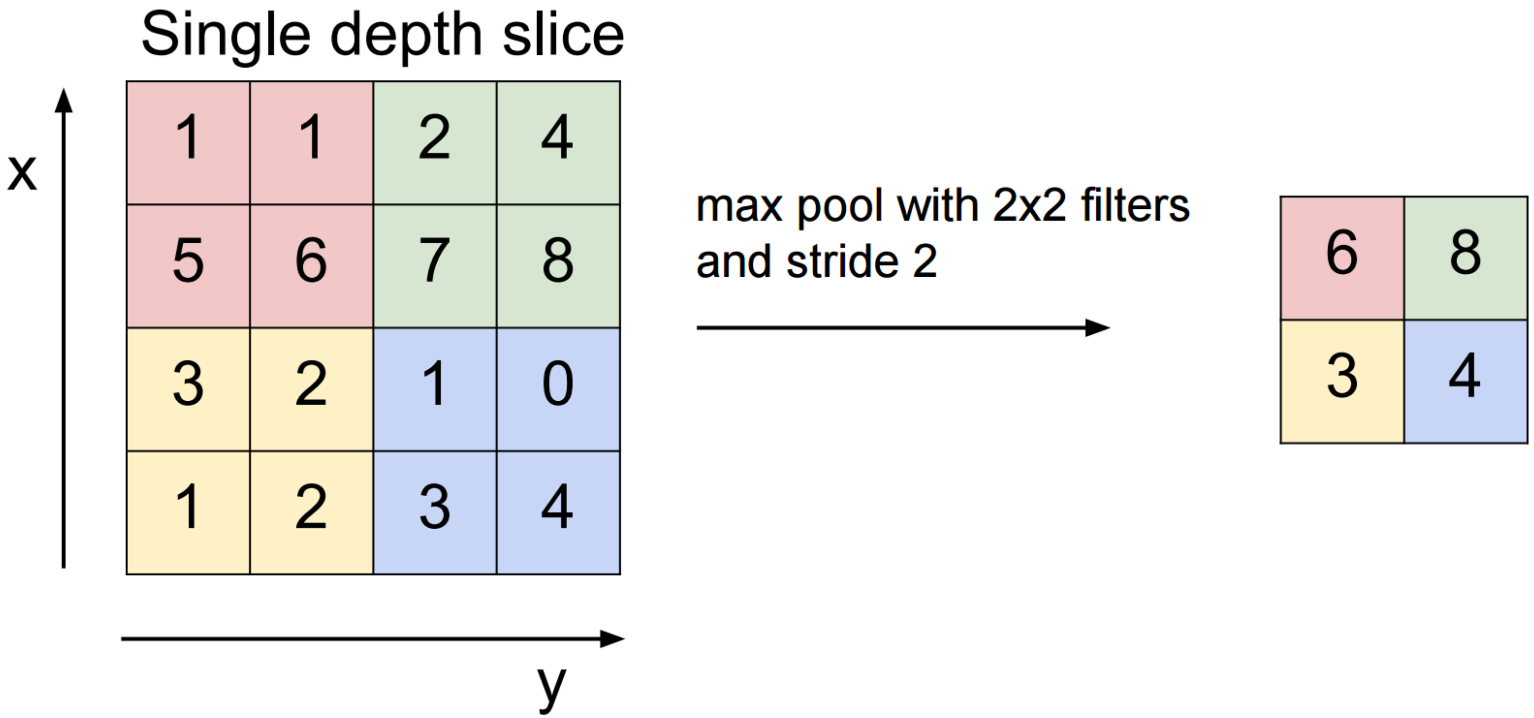


Figure from Fei-Fei Li & Andrej Karpathy & Justin Johnson (CS231N)

Max-pooling

Depth doesn't change!

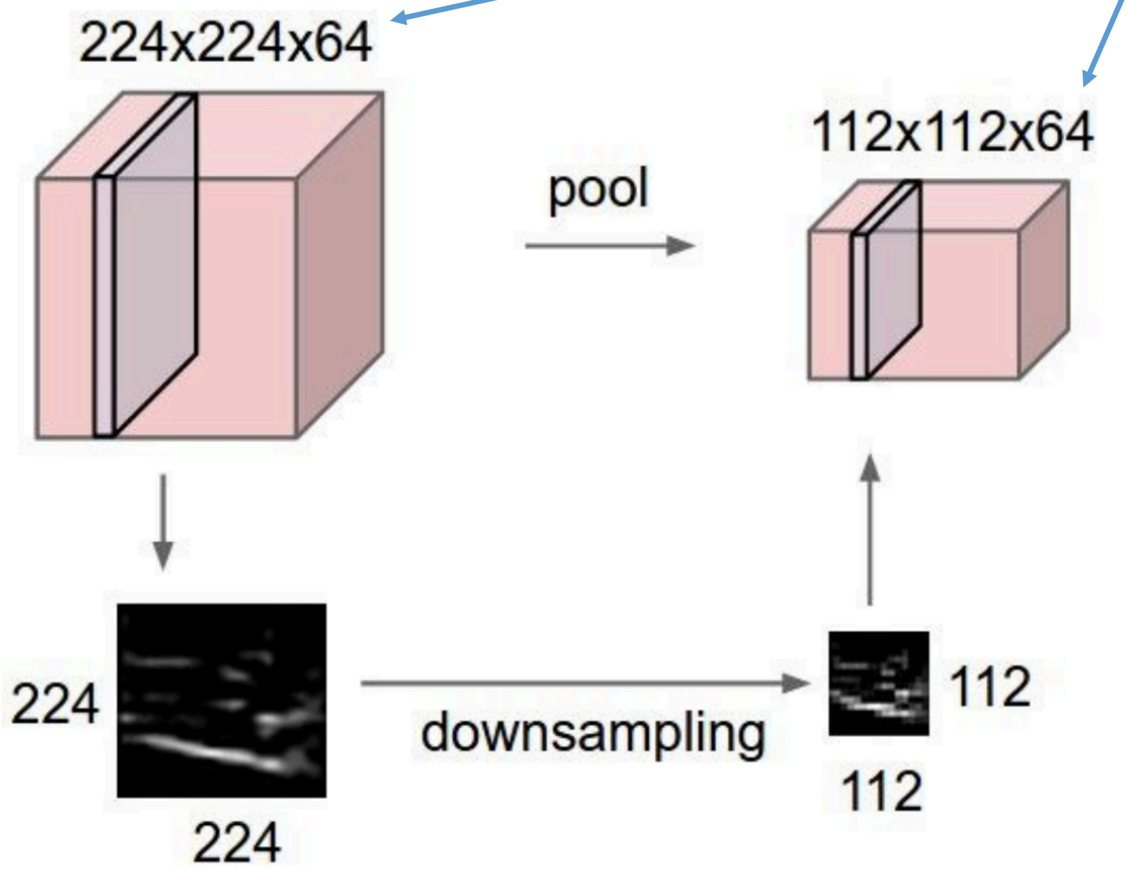


Figure from Fei-Fei Li & Andrej Karpathy & Justin Johnson (CS231N)

Why Max-pool?

- Reduce spatial size of representation.
- Reduce the number of parameters drastically.
- 2x2 filter with stride = 2 discards 75% of the activations!
- Control overfitting.
- Provides translation invariance.

Linear Activations

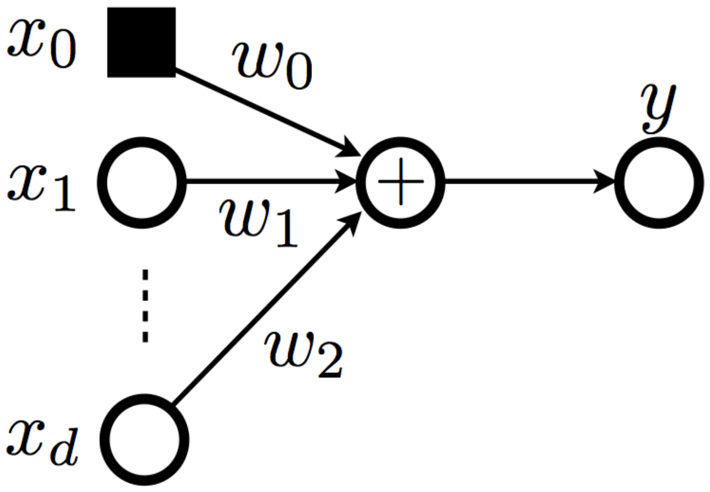
$$y(\mathbf{x}) = \sum_{i=1}^{d} w_i x_i + w_0$$

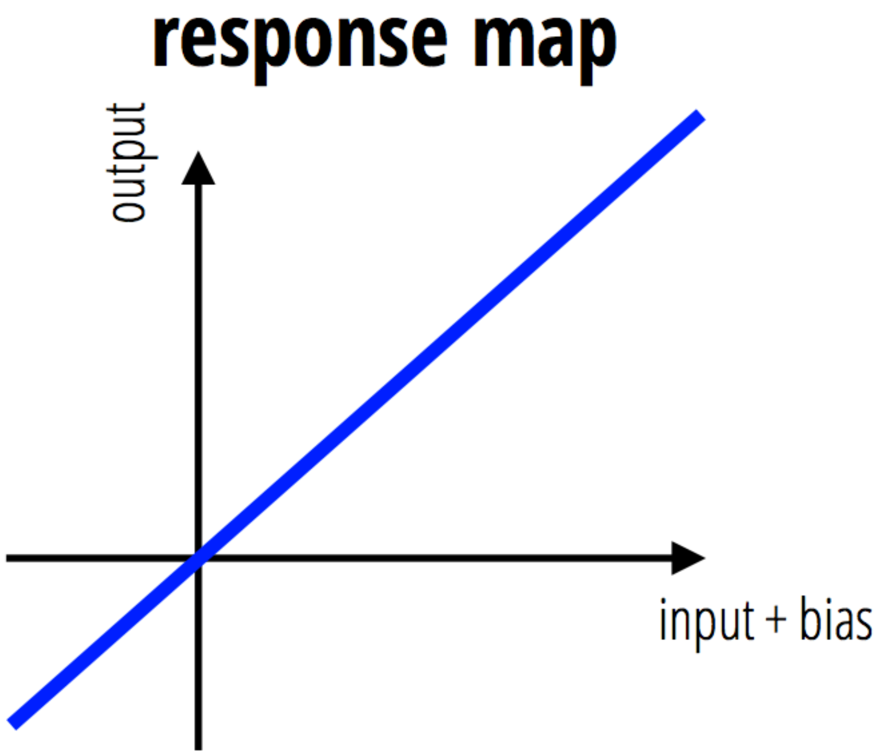
output

i-th input

i-th weight

bias





Why *non-linear* activation functions?

We need a non-linear transformation of data such that the output is a complex, non-linear transformation of the input.

History of Activation Functions

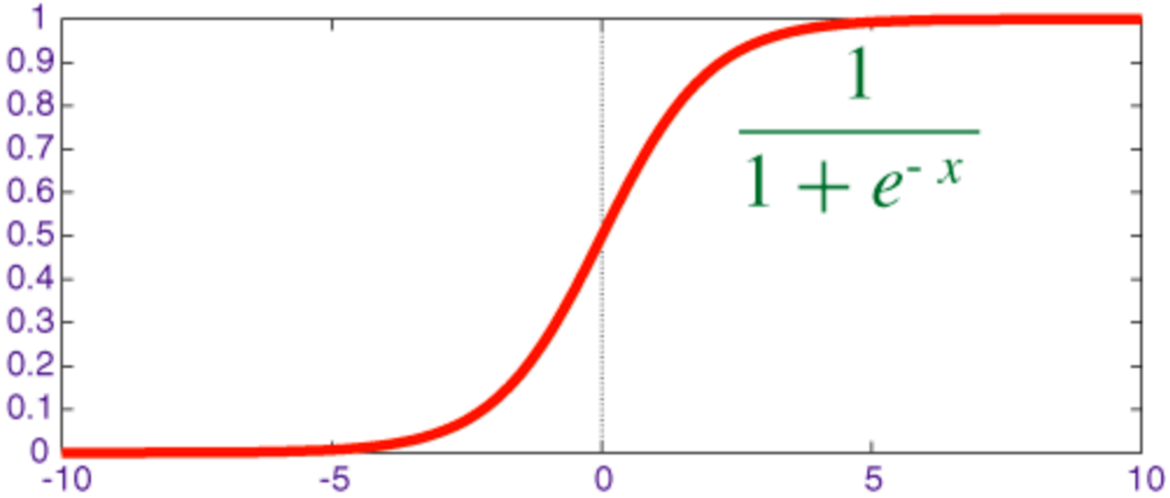
Name	Formula	Year
none	$y = x$	-
sigmoid	$y = \frac{1}{1+e^{-x}}$	1986
tanh	$y = \frac{e^{2x}-1}{e^{2x}+1}$	1986
ReLU	$y = \max(x, 0)$	2010
(centered) SoftPlus	$y = \ln\left(e^{x} + 1\right) - \ln 2$	2011
LReLU	$y = \max(x, \alpha x)$, $\alpha \approx 0.01$	2011
maxout	$y = \max(W_1 x + b_1, W_2 x + b_2)$	2013
APL	$y = \max(x,0) + \sum_{s=1}^{S} a_i^s \max\left(0, -x + b_i^s\right)$	2014
VLReLU	$y = \max(x, \alpha x)$, $\alpha \in 0.1, 0.5$	2014
RReLU	$y = \max(x, \alpha x)$, $\alpha = \text{random}(0.1, 0.5)$	2015
PReLU	$y = \max(x, \alpha x)$, α is learnable	2015
ELU	$y = x$, if $x \geq 0$, else $\alpha(e^{x} - 1)$	2015

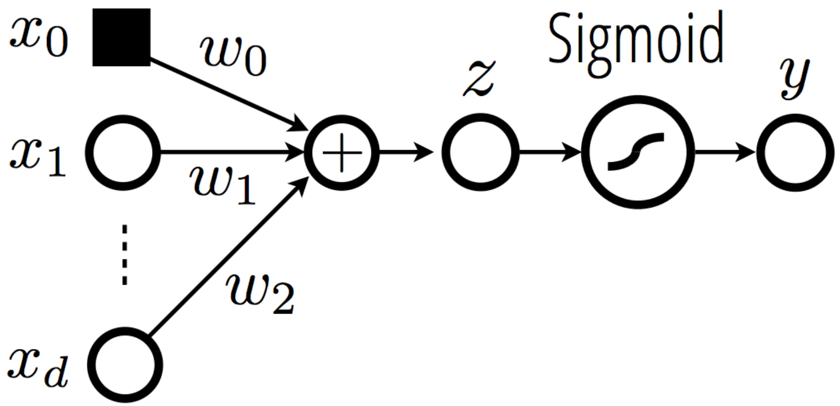
Sigmoid

$$z = \sum_{i=0}^{d} w_i x_i \qquad y = \frac{1}{1+e^{-z}}$$

Logistic Function

$$y = \sigma\left(\sum_{i=0}^{d} w_i x_i\right) \qquad \frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))$$





Sigmoid

- Squashes numbers to range [0,1] – can kill gradients. (Vanishing gradient)
- Best for learning “logical” functions – i.e. functions on binary inputs.
- Not as good for image networks (replaced by RELU)

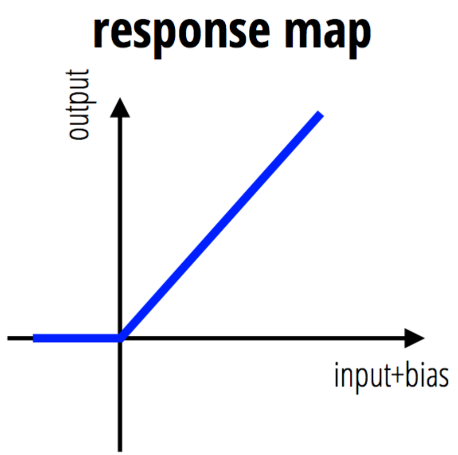
Rectified Linear Unit

$$z = \sum_{i=0}^{d} w_i x_i \qquad y = \begin{cases} z, & \text{if } z > 0 \\ 0, & \text{otherwise} \end{cases}$$

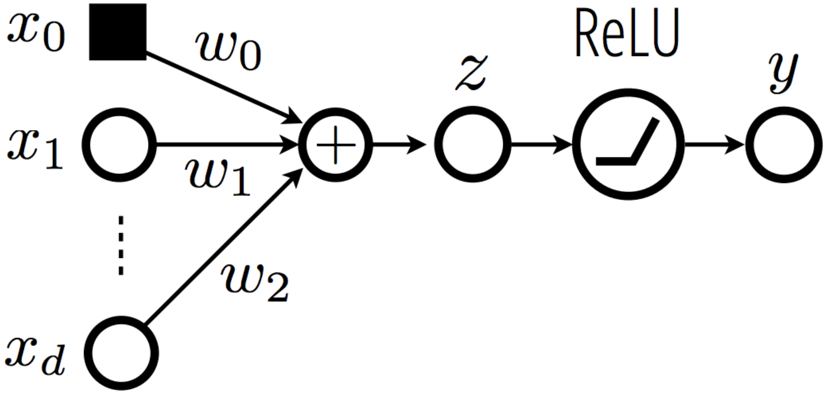
ReLU: $y = \max(0, z)$

Noisy ReLu: $y = \max(0, z + \epsilon) \quad \epsilon \sim \mathcal{N}(0, \sigma)$

Leaky ReLu: $y = \begin{cases} z, & \text{if } z > 0 \\ az, & \text{otherwise} \end{cases}$



Note: Output is a nonlinear function of input, but is linear above zero



Why ReLu?

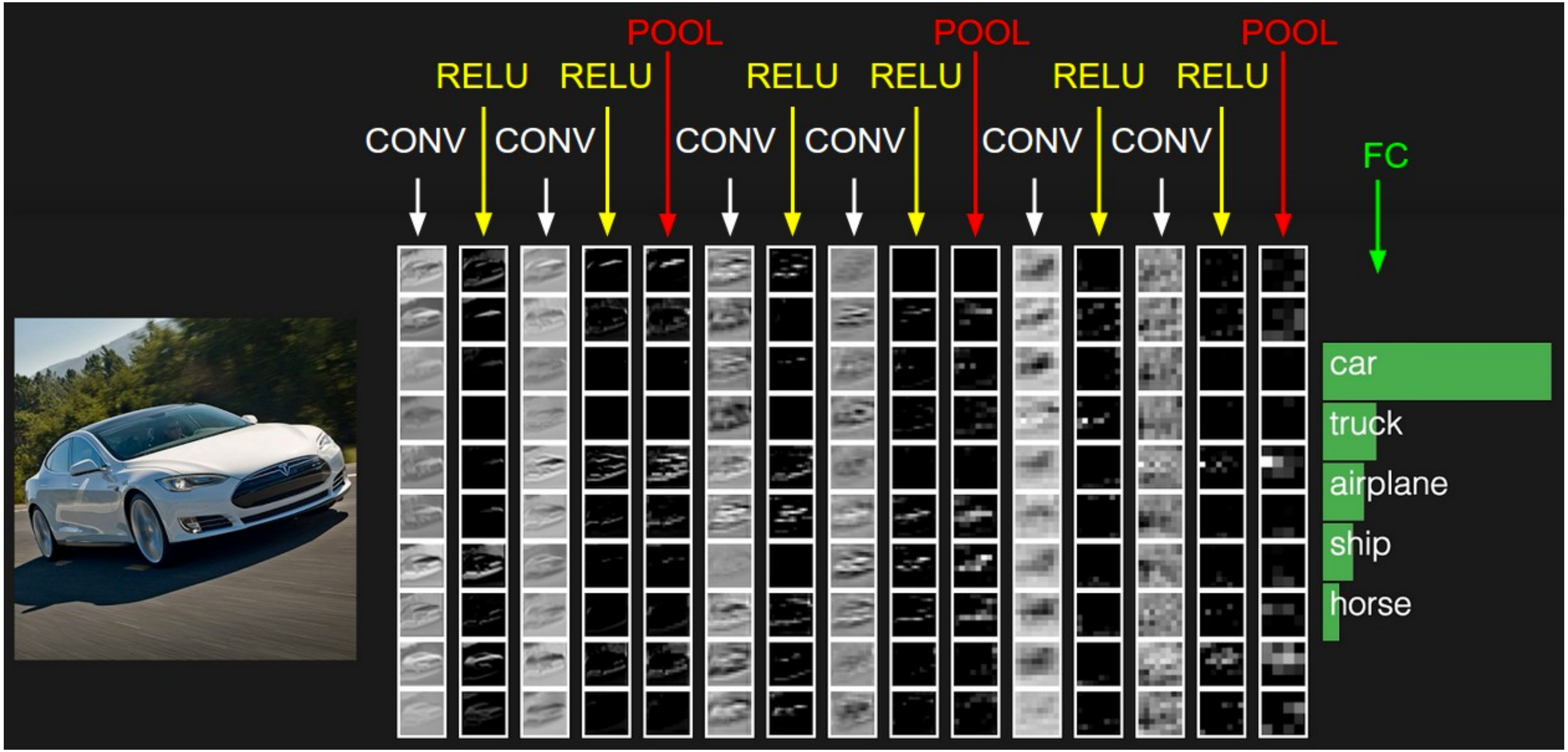
- Inexpensive computations. (Almost 6x faster than sigmoid!)
- No vanishing gradient!
- Leaky ReLus used to prevent “dying” neurons.
- Sparse gradients. (Skip computations where input < 0)

Softmax Function

- All positive values which sum to 1.
- Final layer after output layer.
- Neat probabilistic interpretation – gives probabilities of each class.

$$\boxed{\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}}$$

Deep Learning is just a combination of Convolutions + Pooling + ReLu



Network Initialization

How do you initialize all the weights in the network?

We do not know the final values of the weights..

All weights = 0?

- No learning.
- All outputs are 0.
- Errors are not backpropagated.
- No updates.

Initialized to small random values

- We want the weights close to 0, but not exactly 0.
- Initialize to small random values to *break symmetry*.
- Recommended : Sample from Uniform(-r, r)

$$r = 4\sqrt{\frac{6}{in + out}}$$

Top deep learning libraries

Caffe

K

TensorFlow™

dmlc
mxnet

torch

theano

Terminologies

- **Iteration** : 1 forward pass
- **Epochs** : 1 full training cycle on data set
- **Batch-size** : Number of samples trained per iteration
- **Learning Rate** : Update = Learning Rate x Gradient
- **Max-Epochs** : Usually 20. (Depends on data set)