Deep Learning and Convolutional Neural Networks

Computer Vision

Winter Semester 20/21

Goethe University

Acknowledgement: Some images are from various sources: UCF, Stanford cs231n, etc.

What we did last week

- Image classification:
    - Linear classifier
    - Gradient descent

Today’s class

- Deep Learning
- Convolutional Neural Networks

Linear Classifier Recap

- **Score function:**

$$s = f(x; W, b) = Wx + b$$

Linear Classifier

- **Score function:**

$$s = f(x; W, b) = Wx + b$$

- **Shorthand notation**

$$s = [W\ b][x\ 1]^T$$

$$W \qquad x$$

$$s = f(x; W) = Wx$$

Input x: (D+1)x1
Weight W: Kx(D+1)
Score s:Kx1

Linear Classifier

- **Score function:**

$$s = f(x; W, b) = Wx + b$$

- **Shorthand notation**

Input x: (D+1)x1
Weight W: Kx(D+1)
Score s:Kx1

$$s = [W\ b][x\ 1]^T$$

$$W \qquad x$$

$$s = f(x; W) = Wx$$

LIMITATIONS:

Rather insufficient to predict the class of x

- High dimensional input

- Highly nonlinear classification function

Classification

- Classification function for image is **complex, non-linear**

$$y = F(x)$$

x =

y = 1, 2, ... or K (class index)

Classification

- Classification function for image is **complex, non-linear**

$$y = F(x)$$

x =

y = 1, 2, ... or K (class index)

- Given data points (training examples)

$$y_i = F(x_i)$$

- Able to generalize to unseen example

Classification

- Classification function for image is **complex, non-linear**

$$y = F(x)$$

x =

y = 1, 2, ... or K (class index)

- Given data points (training examples)

$$y_i = F(x_i)$$

- Able to generalize to unseen example
- Our goal is to learn a good approximation of $F(x)$

Deep neural network: a class of function with large capacity to provide this approximation

  – With certain parameters learned in training

Stacking linear classifiers

- Stacking linear classifiers to improve representational power (to approximate $F(x)$)

$$s_1 = W_1 x$$

$$s_2 = W_2 s_1$$

Still linear, $W = W_2 W_1$

Stacking linear classifiers

- Stacking linear classifiers to improve representational power (to approximate $F(x)$)

$$s_1 = W_1 x$$
$$s_2 = W_2 s_1$$

Still linear, $W = W_2 W_1$

- Add non-linearity between layers (stages)

$$s_1 = W_1 x$$
$$s_2 = W_2 \sigma(s_1)$$

Can approximate any continuous function $F(x)$

- **Activation function** is applied element-wise

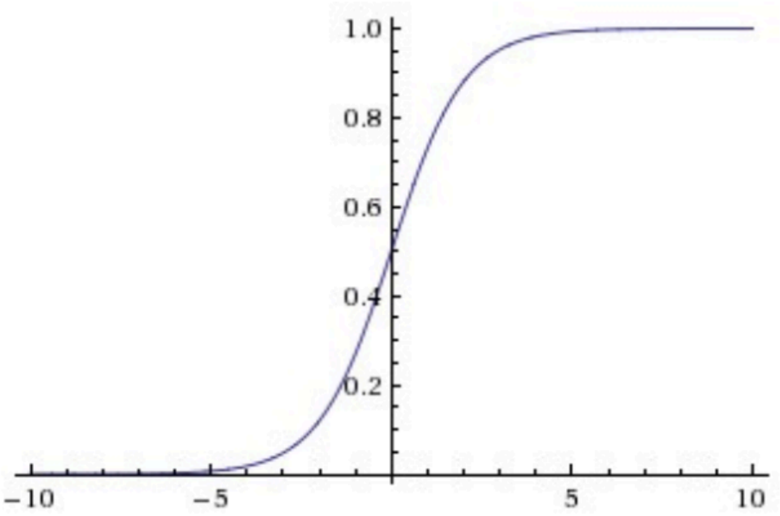
Stacking linear classifiers

$$s_1 = W_1 x$$
$$s_2 = W_2 \sigma(s_1)$$

Can approximate any continuous function $F(x)$

- **Activation function** is applied element-wise
- Example: Sigmoid

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

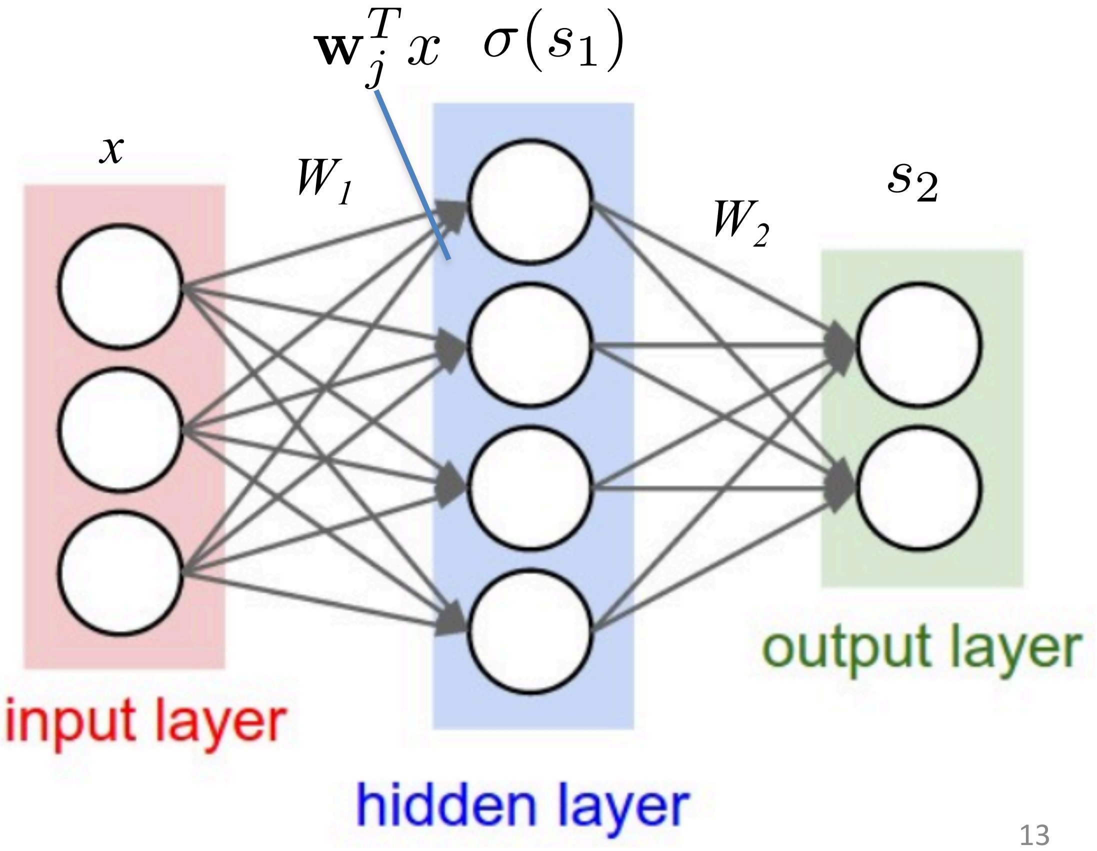


We obtain a neural network

Neural Network

- Neural network: collection of **neurons**
  - Connected in an acyclic graph
  - Output of a neuron can be input of another

$$s_1 = W_1 x$$
$$s_2 = W_2 \sigma(s_1)$$

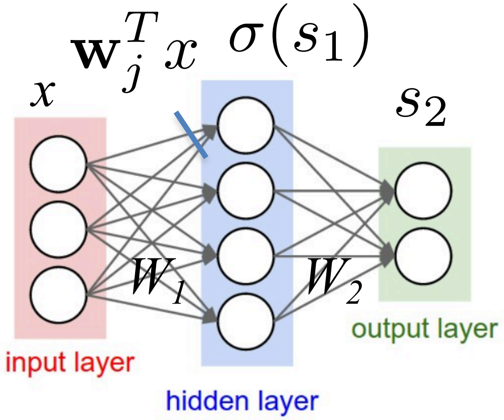


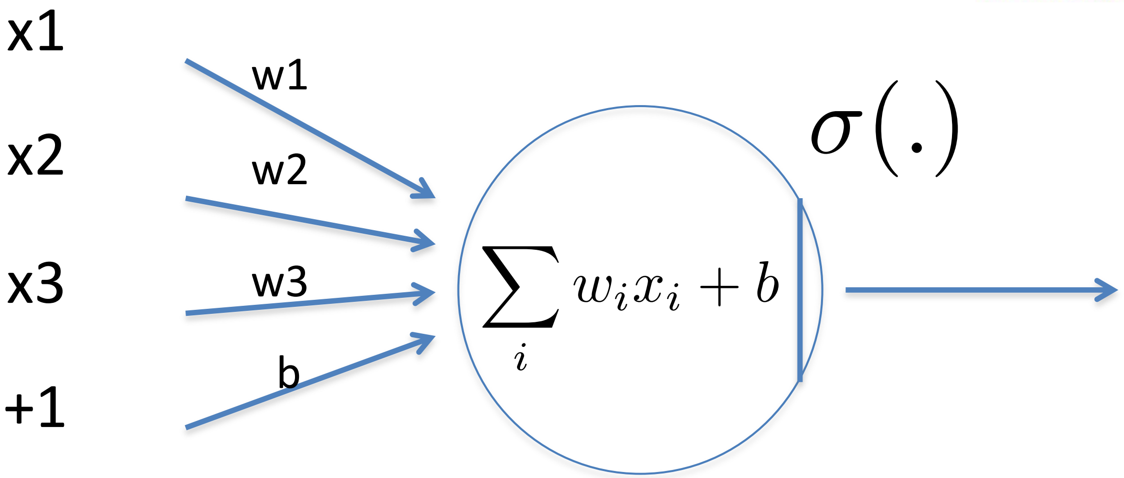
Neural Network

- Neural network: collection of **neurons**
  - Connected in an acyclic graph
  - Output of a neuron can be input of another

$$s_1 = W_1 x$$

$$s_2 = W_2 \sigma(s_1)$$

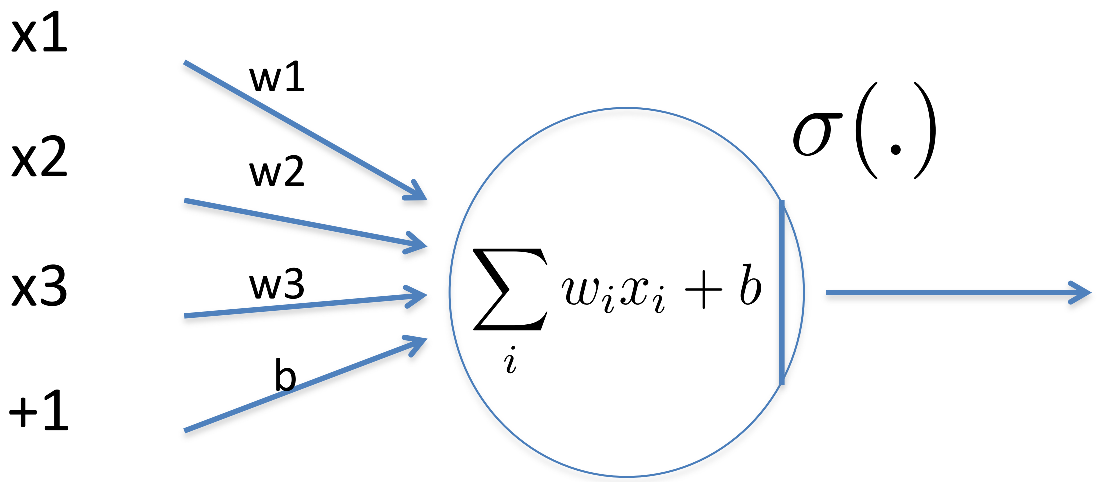




A neuron

- Neuron: a computational unit, take input x, output:

$$\sigma(\sum_i w_i x_i + b)$$

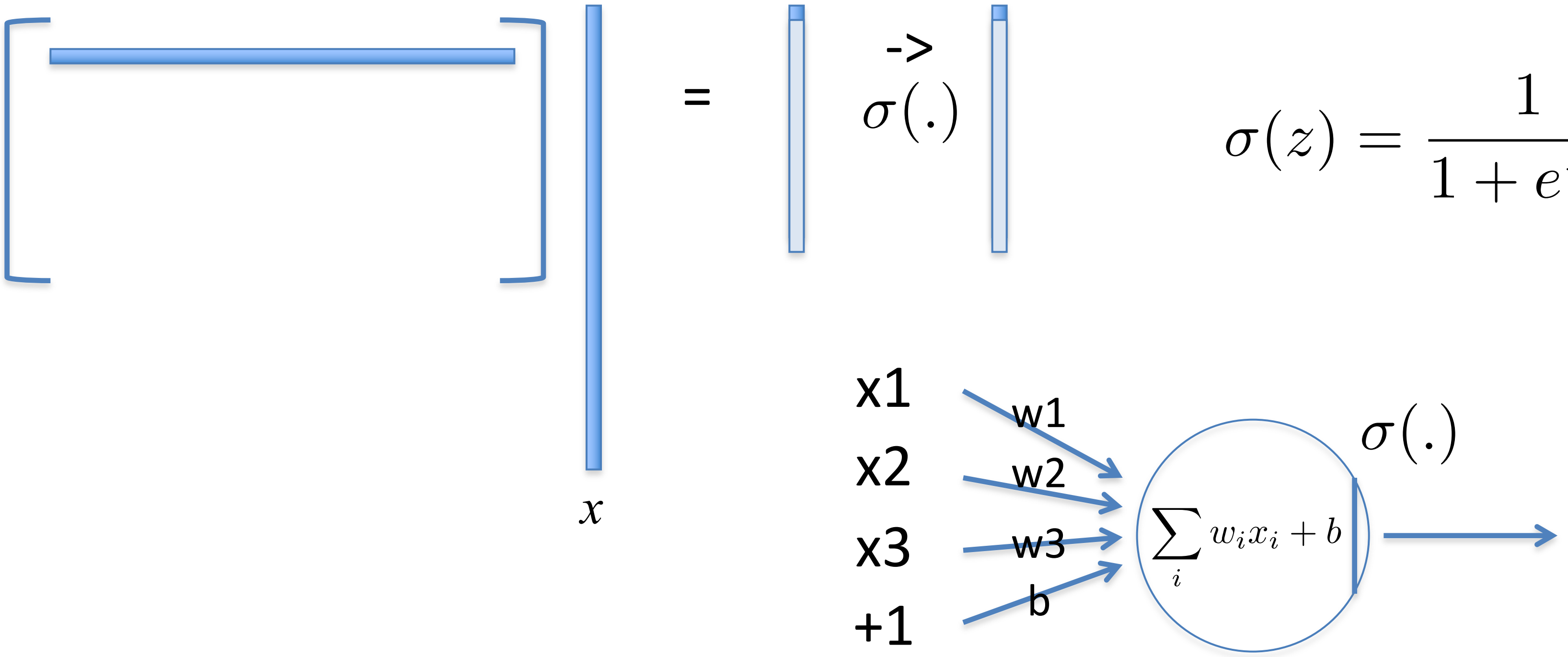


A neuron

- Neuron: a computational unit, take input x, output:

$$\sigma(\sum_i w_i x_i + b)$$

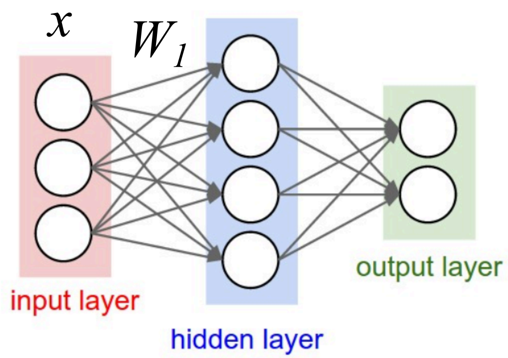
- Activation (Sigmoid) function is applied element-wise

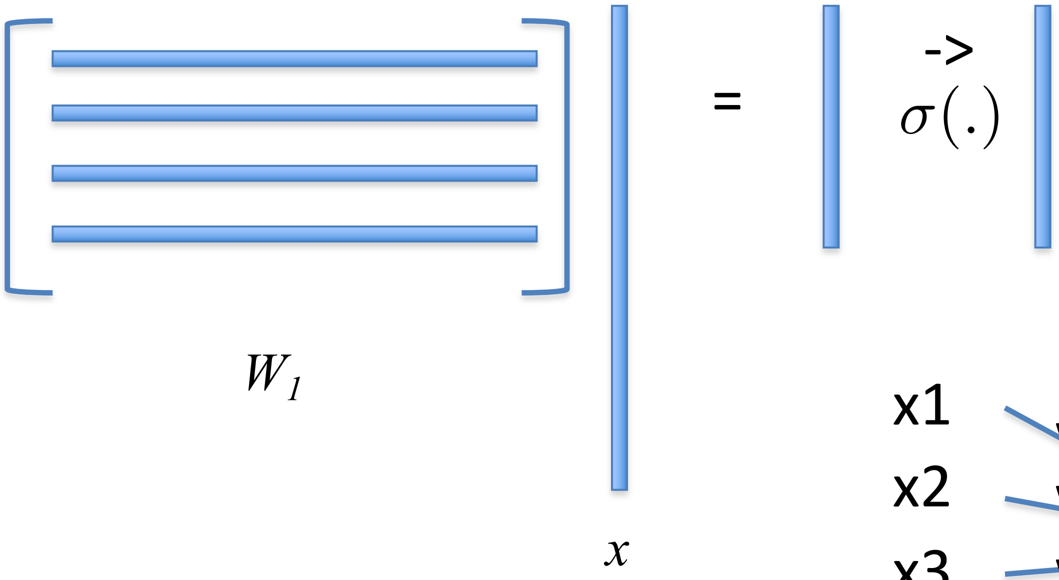


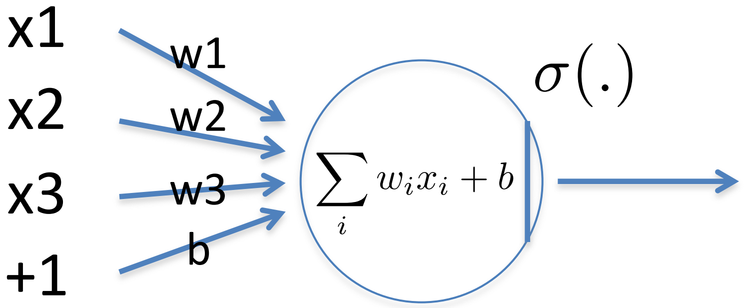
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Neural Network

- Neural network: collection of neurons
  - Connected in an acyclic graph
  - Output of a neuron can be input of another



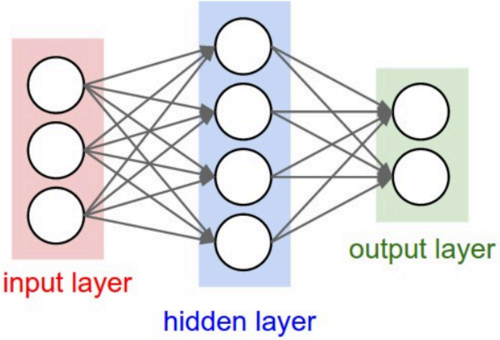




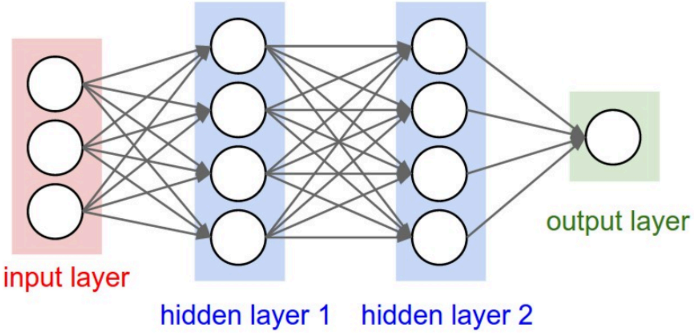
Neural Network

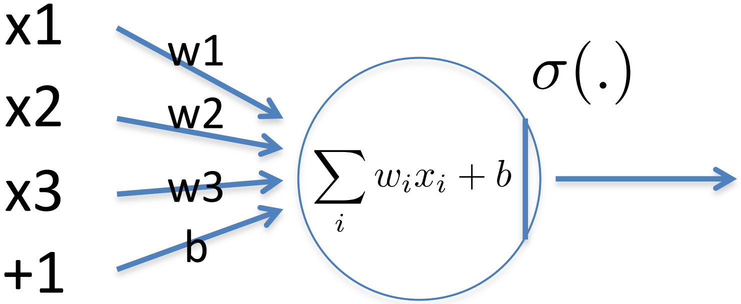
- Hidden layer: values are not observed in the training set
- Output layer: no activation

2-layer NN: 1 hidden, 1 output layer



?-layer NN: ? hidden, ? output layer

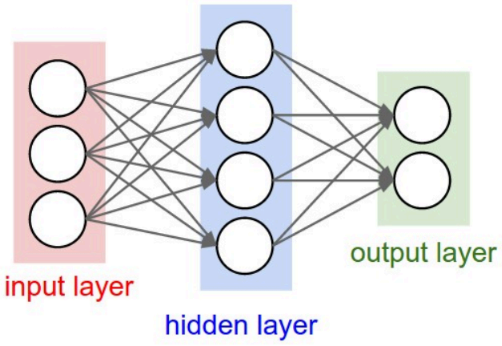




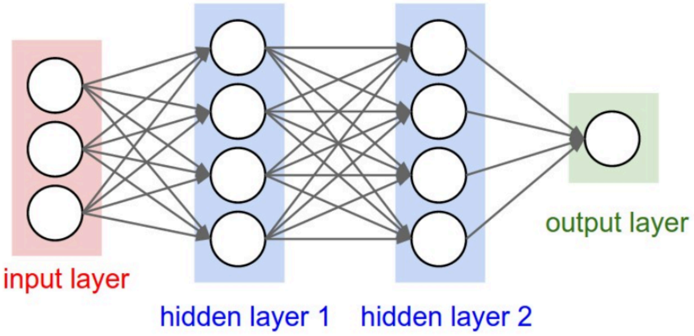
Neural Network

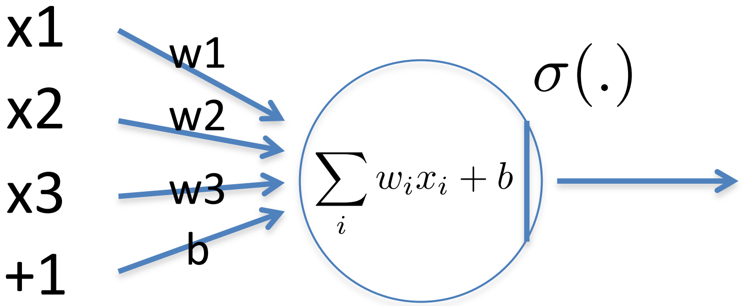
- Hidden layer: values are not observed in the training set
- Output layer: no activation

2-layer NN: 1 hidden, 1 output layer



3-layer NN: 2 hidden, 1 output layer





x1

x2

x3

+1

w1

w2

w3

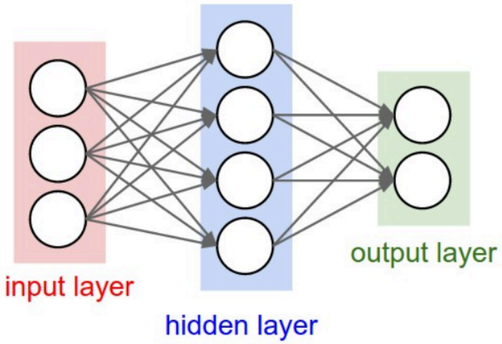
b

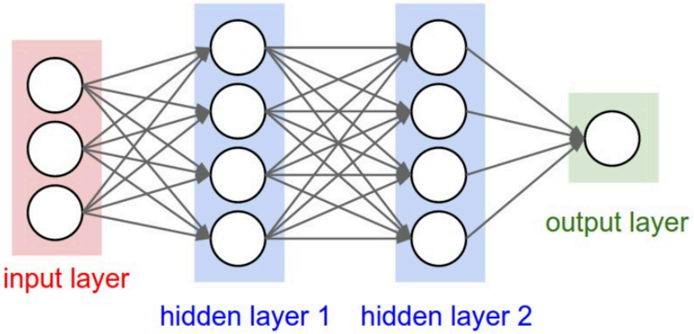
$\sum_i w_i x_i + b$

$\sigma(.)$

Neural Network

- Hidden layer: values are not observed in the training set
- Output layer: no activation


2-layer NN: 1 hidden, 1 output layer

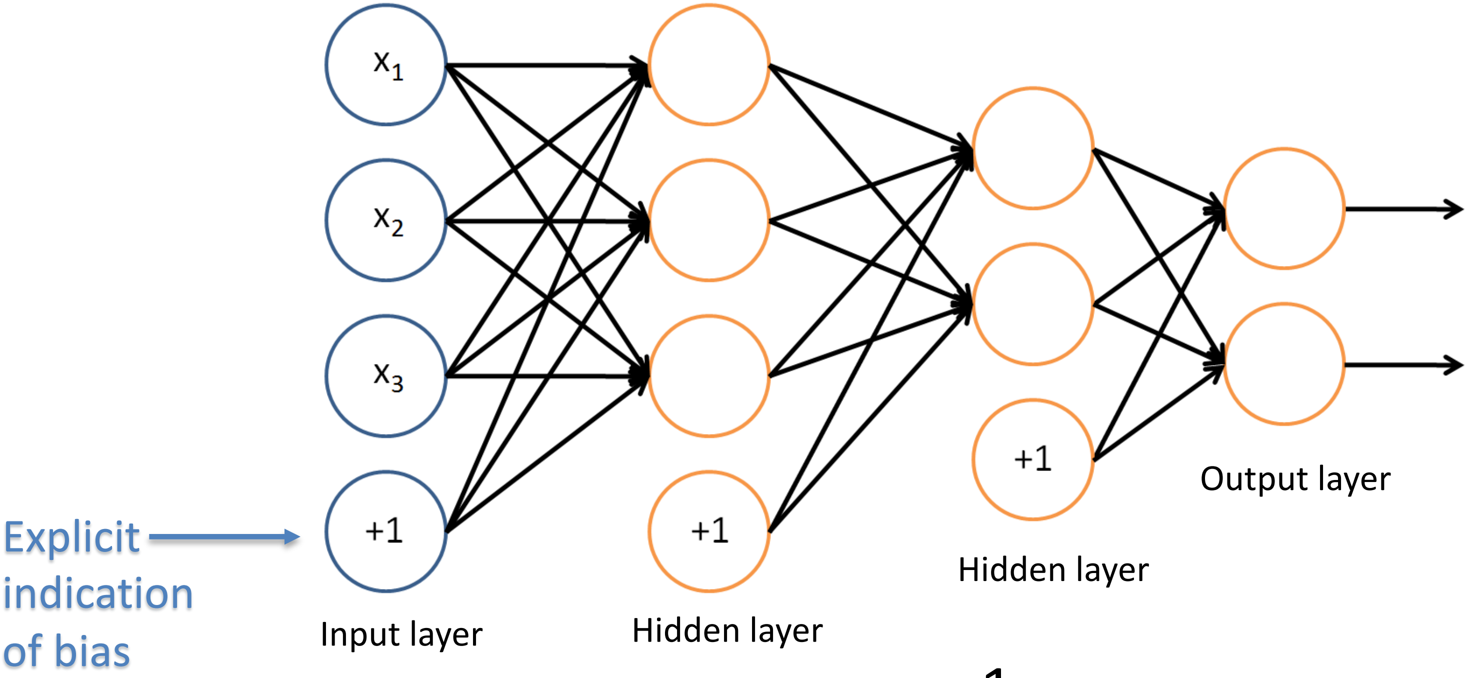

3-layer NN: 2 hidden, 1 output layer

Artificial neural network (ANN)

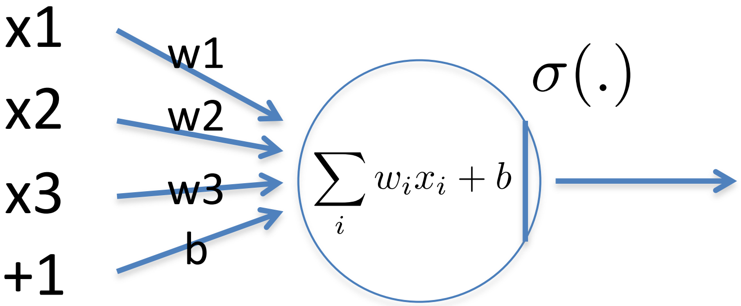
Multi-layer perceptrons (MLP)

Neural Network

- Hidden layer: values are not observed in the training set
- Output layer: no activation

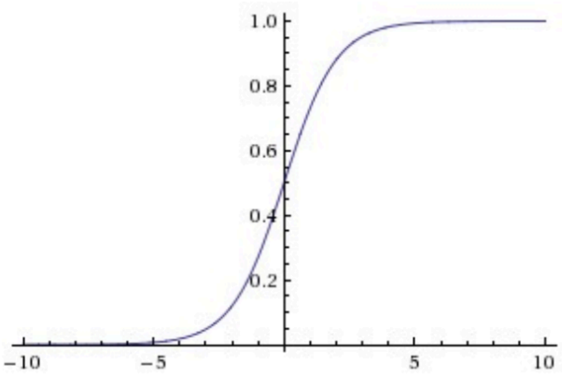


Explicit indication of bias



Activation function

Sigmoid

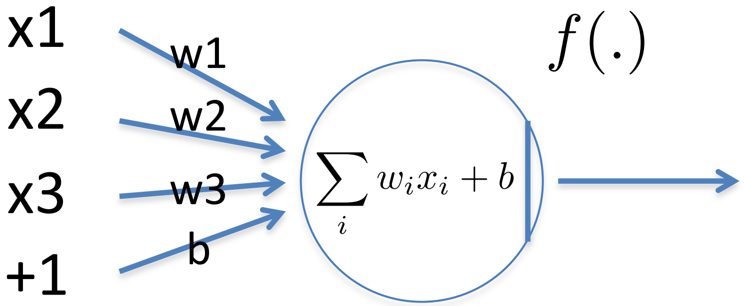


$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

-Incorporate non-linear

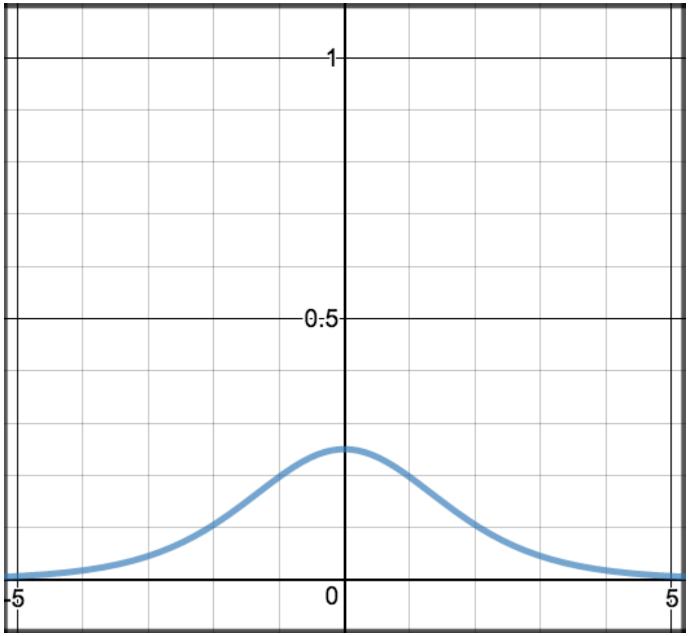
-Limit the output range (or additional normalization)

-Decision / probabilistic interpretation
  - -Detect feature or not
  - -Biological neuron: to fire or not



Activation function

Sigmoid derivative:



$$\sigma(z) = \frac{1}{1+e^{-z}}$$

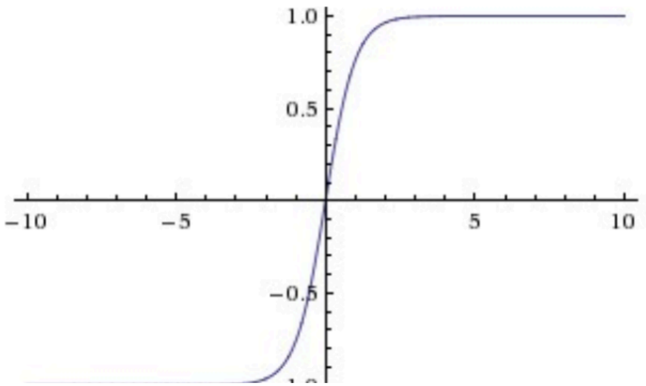
Easy to compute gradient:

$$\sigma'(z) = \sigma(z)(1-\sigma(z))$$

Plots for derivatives: https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html

Activation function

Hyperbolic tangent



$$f(z) = tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

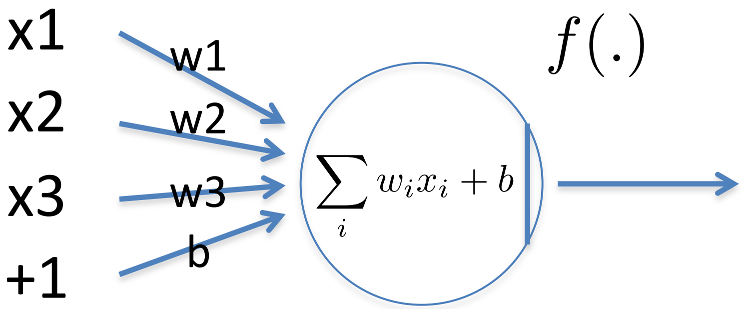
$$tanh(z) = 2\sigma(2z) - 1$$

PROBLEMS:

-Saturation, vanishing gradient (as the sigmoid)

-Slow and difficult to train with gradient descent

-Stronger gradient than the sigmoid



Activation function

Hyperbolic tangent derivative:

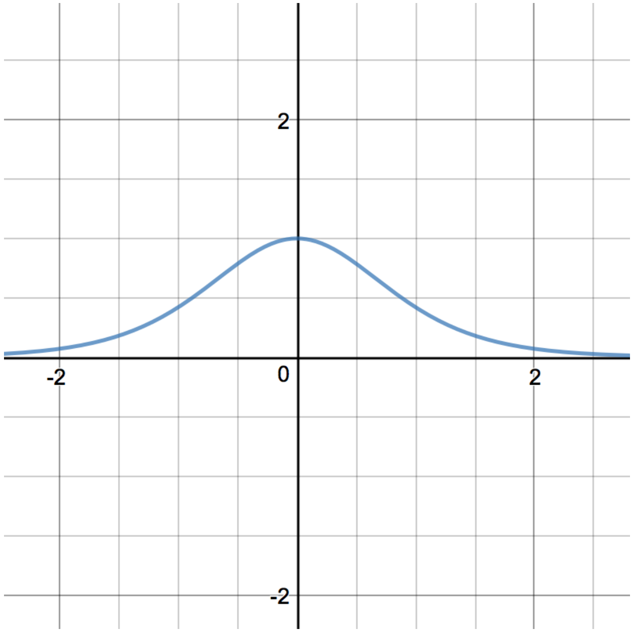
$$\frac{d}{dz}\left(\frac{e^z-e^{-z}}{e^z+e^{-z}}\right)=\frac{e^z+e^{-z}}{\left(e^z+e^{-z}\right)^2}d\left(e^z-e^{-z}\right)-\frac{e^z-e^{-z}}{\left(e^z+e^{-z}\right)^2}d\left(e^z+e^{-z}\right)$$

$$=\frac{\left(e^z+e^{-z}\right)\left(e^z+e^{-z}\right)}{\left(e^z+e^{-z}\right)^2}-\frac{\left(e^z-e^{-z}\right)\left(e^z-e^{-z}\right)}{\left(e^z+e^{-z}\right)^2}$$

$$=\frac{\left(e^z+e^{-z}\right)^2-\left(e^z-e^{-z}\right)^2}{\left(e^z+e^{-z}\right)^2}$$

$$=1-\left(\frac{e^z-e^{-z}}{e^z+e^{-z}}\right)^2$$

$$=1-tanh(z)^2$$

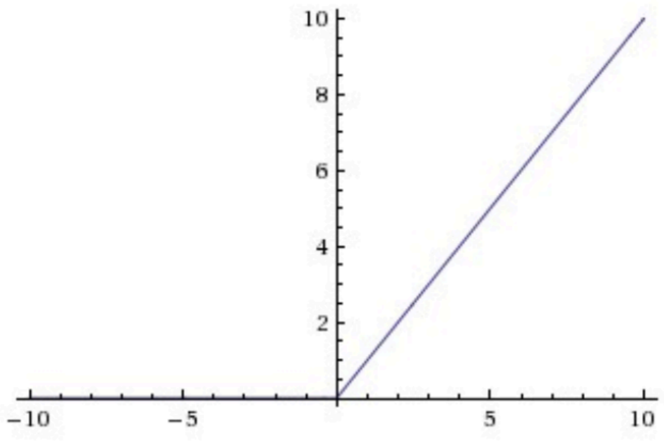


RECAP CHAIN RULE DER.:

$$\frac{q(z)}{g(z)}\rightarrow\frac{d(^q/_g)}{dz}=\frac{gq'-qg'}{g^2}$$

Activation function

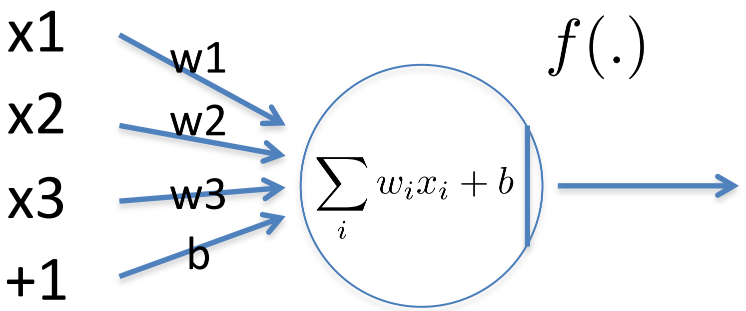
Rectified linear unit (ReLU)



MOST COMMONLY USED

- Avoids vanishing gradient problem
- If strong in negative area, there is no gradient an unit is dead

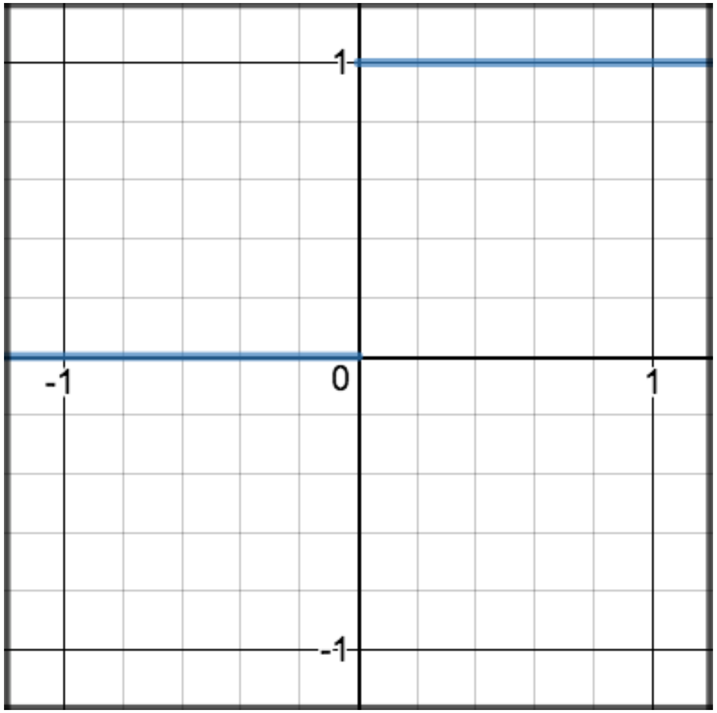
$$f(z) = max(0, z)$$



Activation function

Rectified linear unit (ReLU) derivative

$$f(z) = max(0, z)$$



$$\frac{df(z)}{dz} = \begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$$

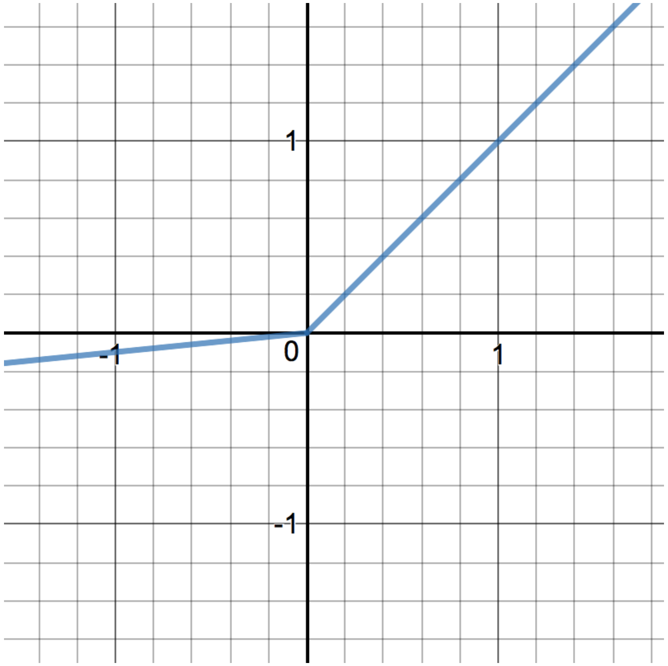
Undefined at 0:

$$\lim_{h \to 0^+} \frac{\max(0, h) - \max(0,0)}{h} = 1$$

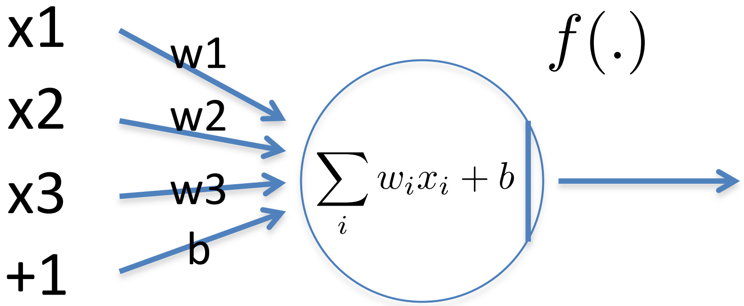
$$\lim_{h \to 0^-} \frac{\max(0, h) - \max(0,0)}{h} = 0$$

Activation function

Rectified linear unit (ReLU) variants: e.g. Leaky ReLU



$$f(z)=\begin{cases} z & z > 0 \\ \alpha z & z \leq 0 \end{cases}$$



x1
x2
x3
+1

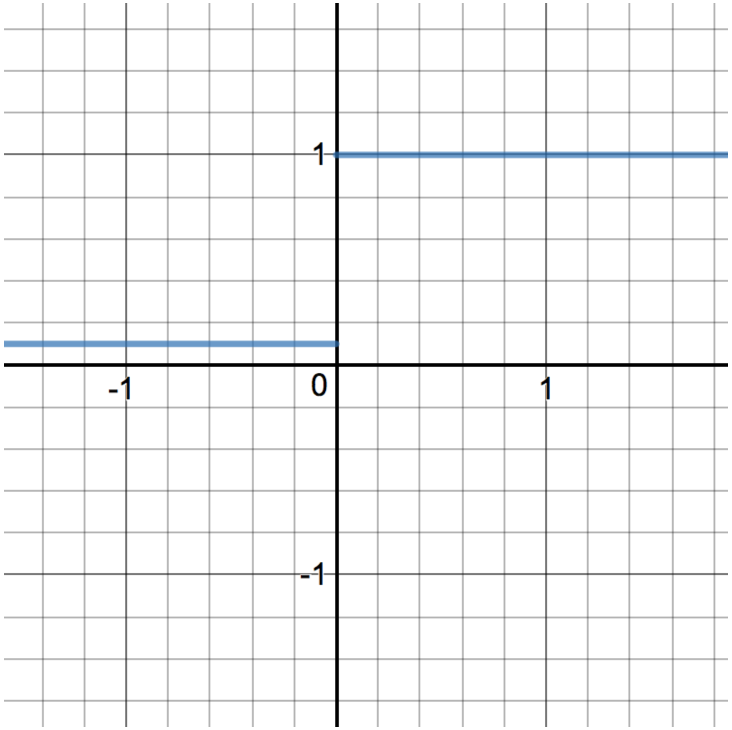
w1
w2
w3
b

$\sum_i w_i x_i + b$

$f(.)$

Activation function

Leaky ReLU derivative



$$f'(z) = \begin{cases} 1 & z > 0 \\ \alpha & z < 0 \end{cases}$$

Undefined at 0

NN as a function approximation

- NN with one hidden layer can approximate any continuous function F(x)
- Classification function in our case
- In practice, NN with multiple hidden layers performs better -> It’s an active research question: e.g. compositionality: f(f(f(x))) captures world structure

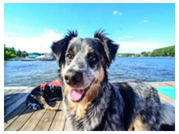
Today’s class

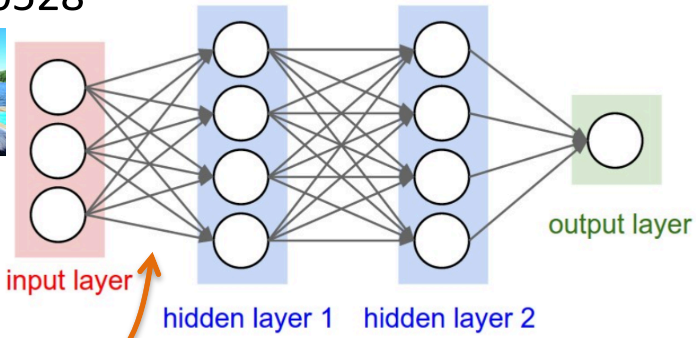
- Deep Learning
- **Convolutional Neural Networks**

Convolutional Neural Network

- CNN: similar to ordinary NN
- In most cases, the inputs are images
- Special network architecture for images
  - Less computation in the forward pass
  - Reduce the number of parameter
  - Better accuracy

224x224x3
=150528



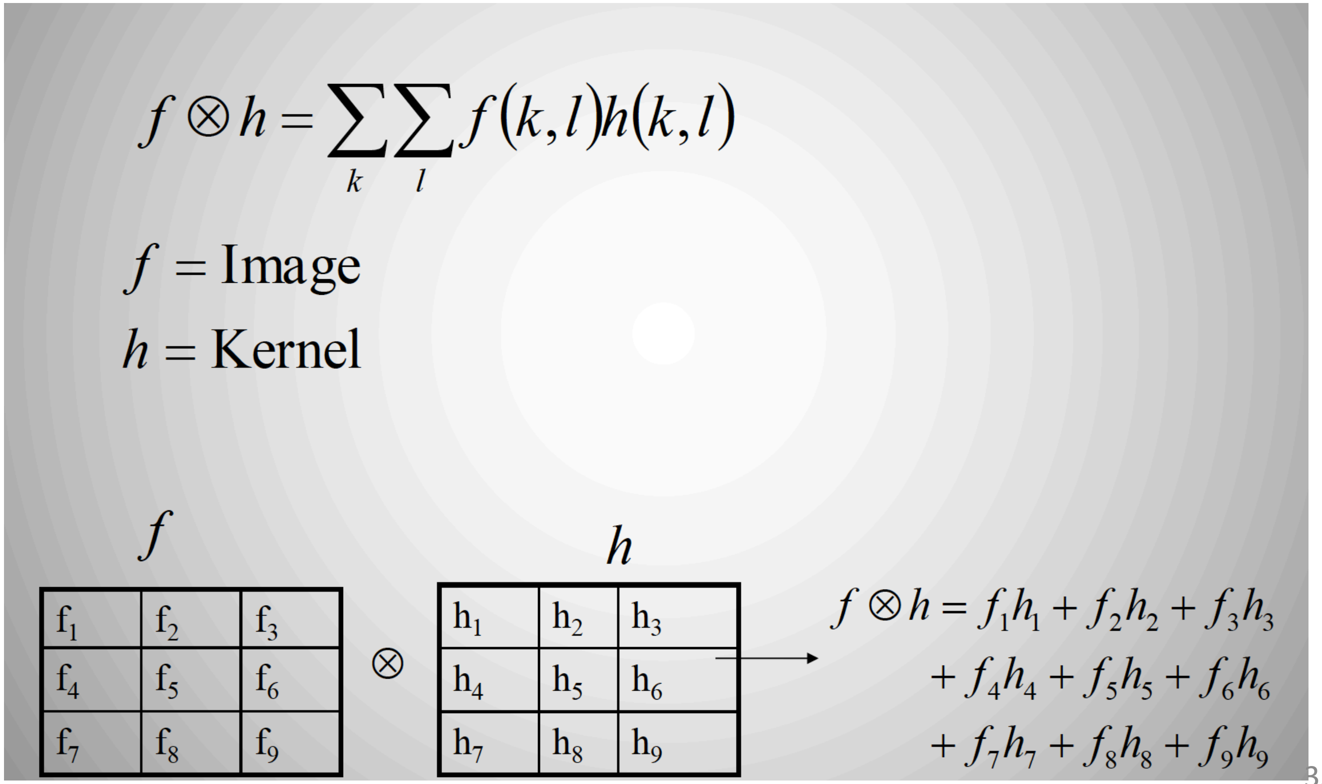


If using hidden layer of similar size, approximately 1x10^{10} parameters (only W_1)

Revisit Image Filtering

- Correlation / convolution (precisely, there are subtle differences)

Correlation / convolution



$$f \otimes h = \sum_{k} \sum_{l} f(k,l)\,h(k,l)$$

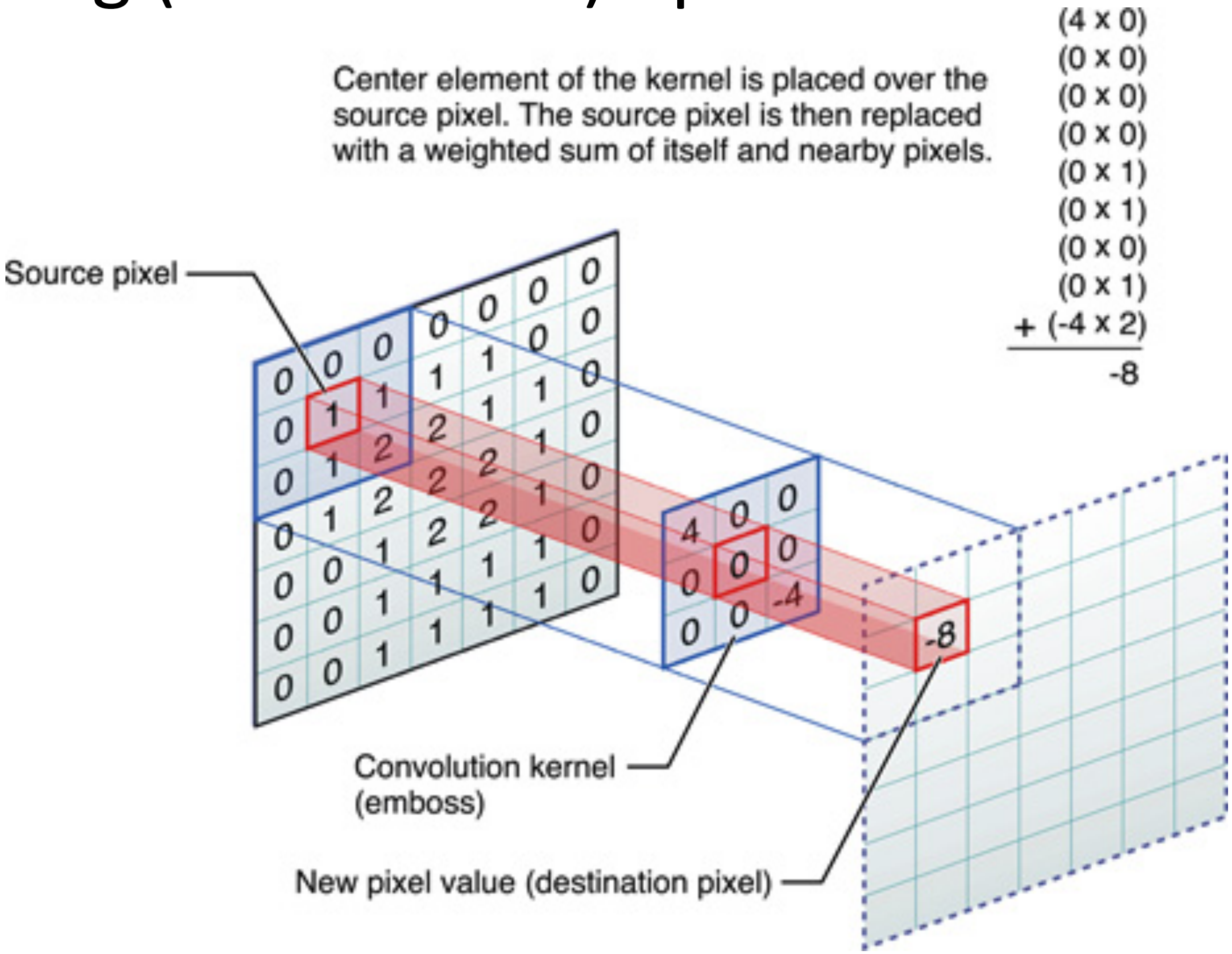
f = Image

h = Kernel

$$\overset{f}{\begin{array}{|c|c|c|}\hline f_1 & f_2 & f_3 \\ \hline f_4 & f_5 & f_6 \\ \hline f_7 & f_8 & f_9 \\ \hline \end{array}} \otimes \overset{h}{\begin{array}{|c|c|c|}\hline h_1 & h_2 & h_3 \\ \hline h_4 & h_5 & h_6 \\ \hline h_7 & h_8 & h_9 \\ \hline \end{array}} \longrightarrow \begin{aligned} f \otimes h = {} & f_1h_1 + f_2h_2 + f_3h_3 \\ & + f_4h_4 + f_5h_5 + f_6h_6 \\ & + f_7h_7 + f_8h_8 + f_9h_9 \end{aligned}$$

Image Filtering

- Filtering (convolution) operation



Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.

Image Filtering

- Filtering (convolution) operation
- Slide the filter kernel over the entire image to produce the output (image/activation)

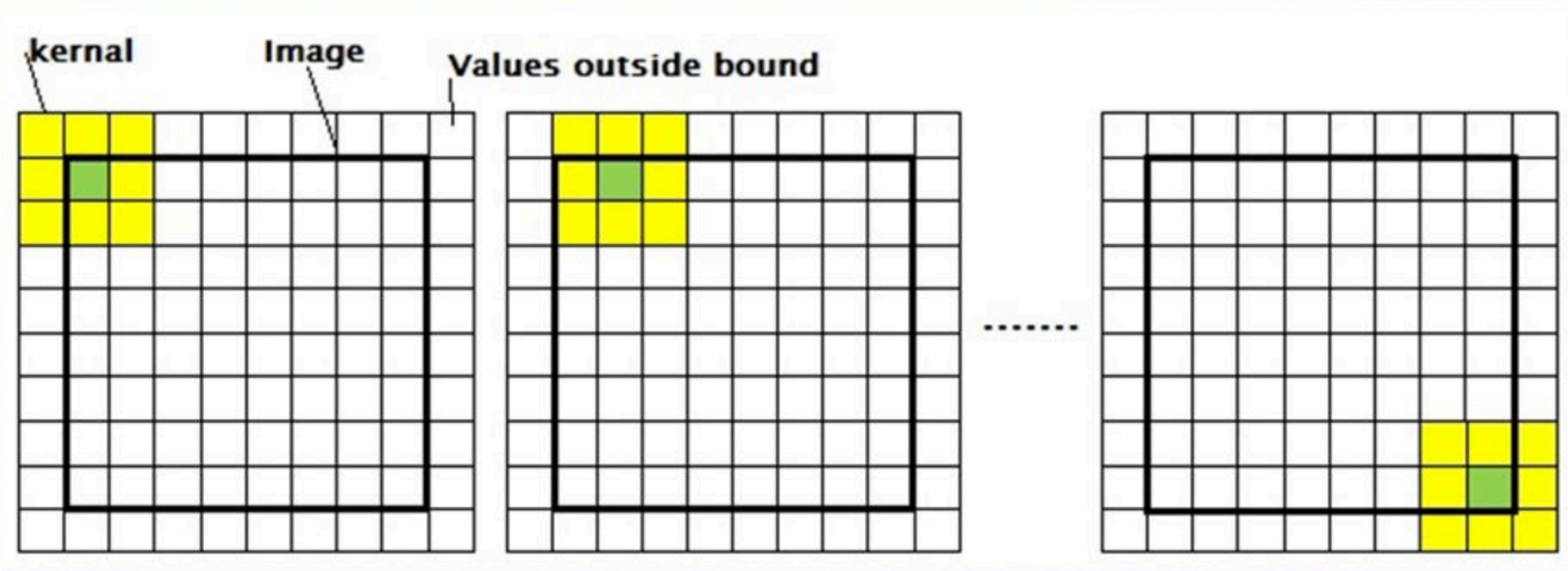
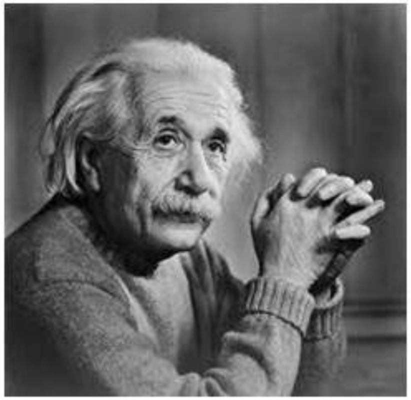


Image Filtering

- Filtering as feature detection / template matching



[[-1,0,1],
[-1,0,1],
[-1,0,1]]

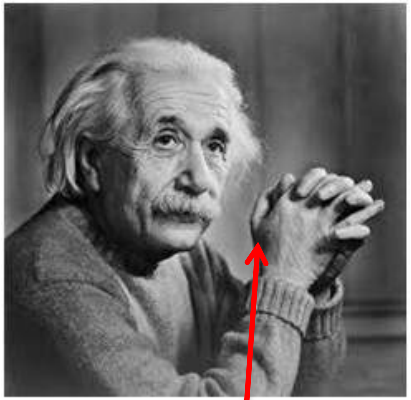
Detect vertical edge

[[-1,-1,-1],
[0, 0, 0],
[1, 1, 1]]

Detect horizontal edge

Image Filtering

- Filtering as feature detection / template matching



Activation / feature map

Vertical edge is detected here (large output)

[[-1,0,1],
[-1,0,1],
[-1,0,1]]

Detect vertical edge

[[-1,-1,-1],
[0, 0, 0],
[1, 1, 1]]

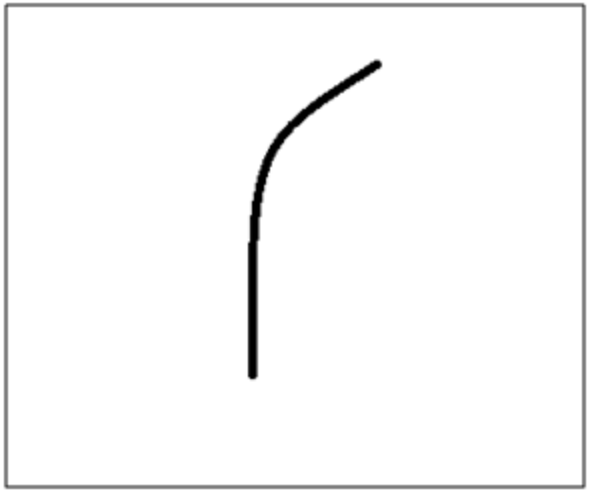
Detect horizontal edge

Image Filtering

- Filtering as feature detection / template matching

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



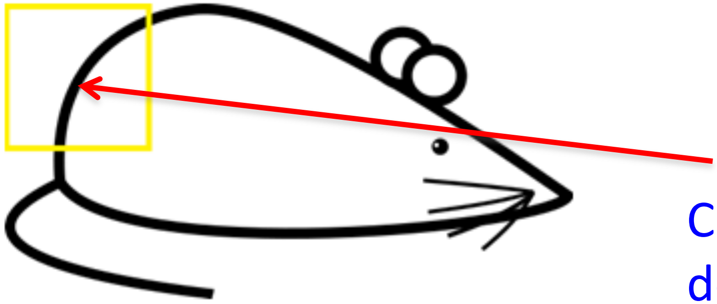
Visualization of a curve detector filter

Generalize to curve detector

Image Filtering

- Filtering as feature detection / template matching

Original image



Visualization of the filter on the image

Curve is detected here (large output)

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

$*$

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Image Filtering

- Filtering as feature detection / template matching

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

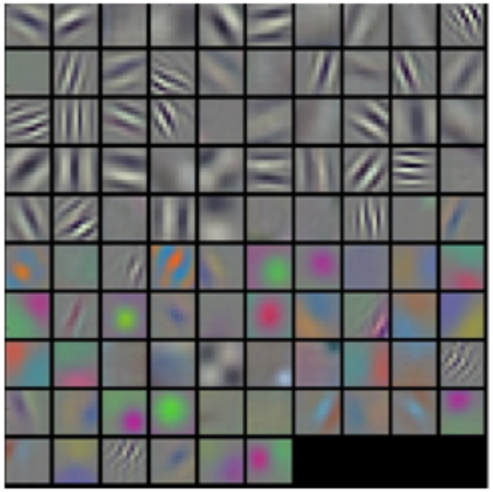
Small output: no curve is detected

Take away: Filtering (convolution) is an efficient mechanism for finding patterns

Filters respond most strongly to pattern elements that look like the filters

Image Filtering

- Filtering as feature detection / template matching

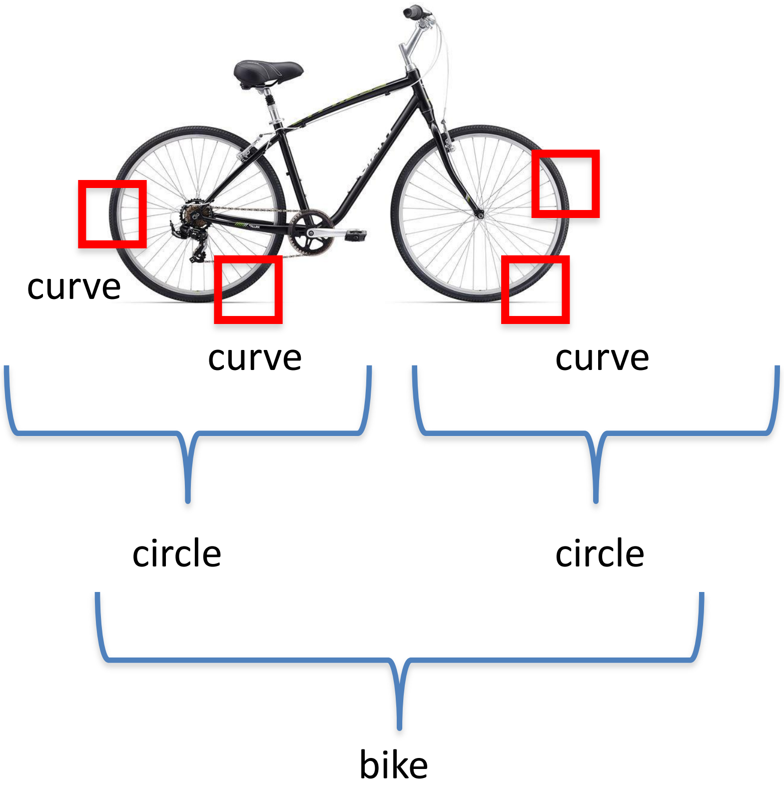


Visualizations of filters

-Need different filter kernels to detect different features
-Data driven approach: use training images to tell us what filter kernels are useful (learns the filters)

How to recognize an object?

- Use feature detection (image filtering) in a hierarchical manner



curve

curve

curve

Low level feature

circle

circle

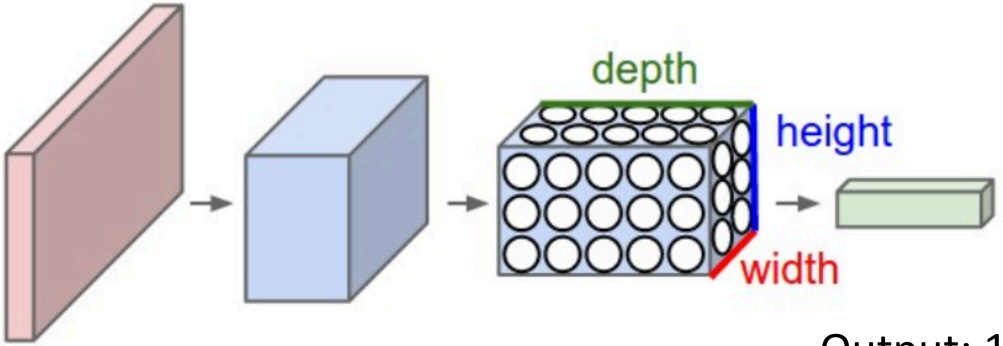
High level feature

bike

Implement this approach using CNN

CNN

- 3D volumes of neurons



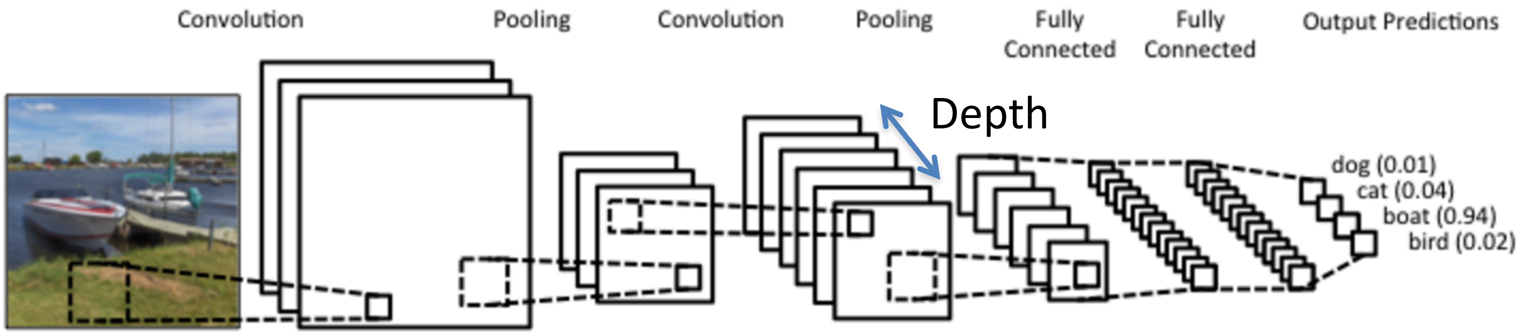
Input: width x height x numChannel

Output: 1 x 1 x numClass

- Stack of
  - Convolutional layer
  - Fully connected layer
  - Pooling layer

CNN

- One example



Input:
width x height x numChannel

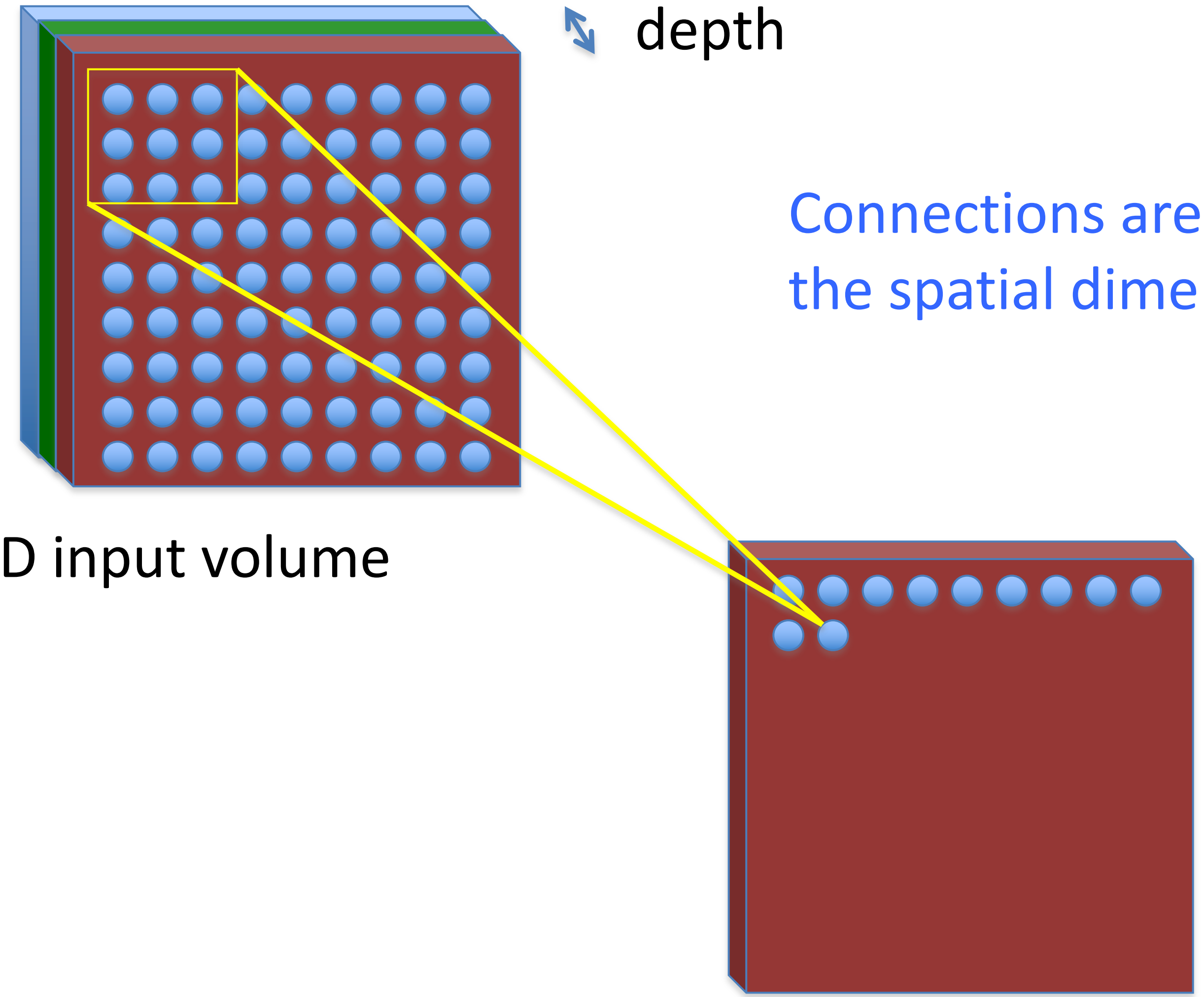
Output:
1 x 1 x numClass

- Stack of
  - Convolutional layer
  - Fully connected layer
  - Pooling layer

Convolutional Layer

- Conv layer: core component
- Local connectivity
  - Spatial extent: receptive field
  - Extent of connectivity along the depth dimension = depth of the input
- Parameter sharing
- Filtering / convolution
  - Instead of matrix multiplication

Convolutional Layer



depth

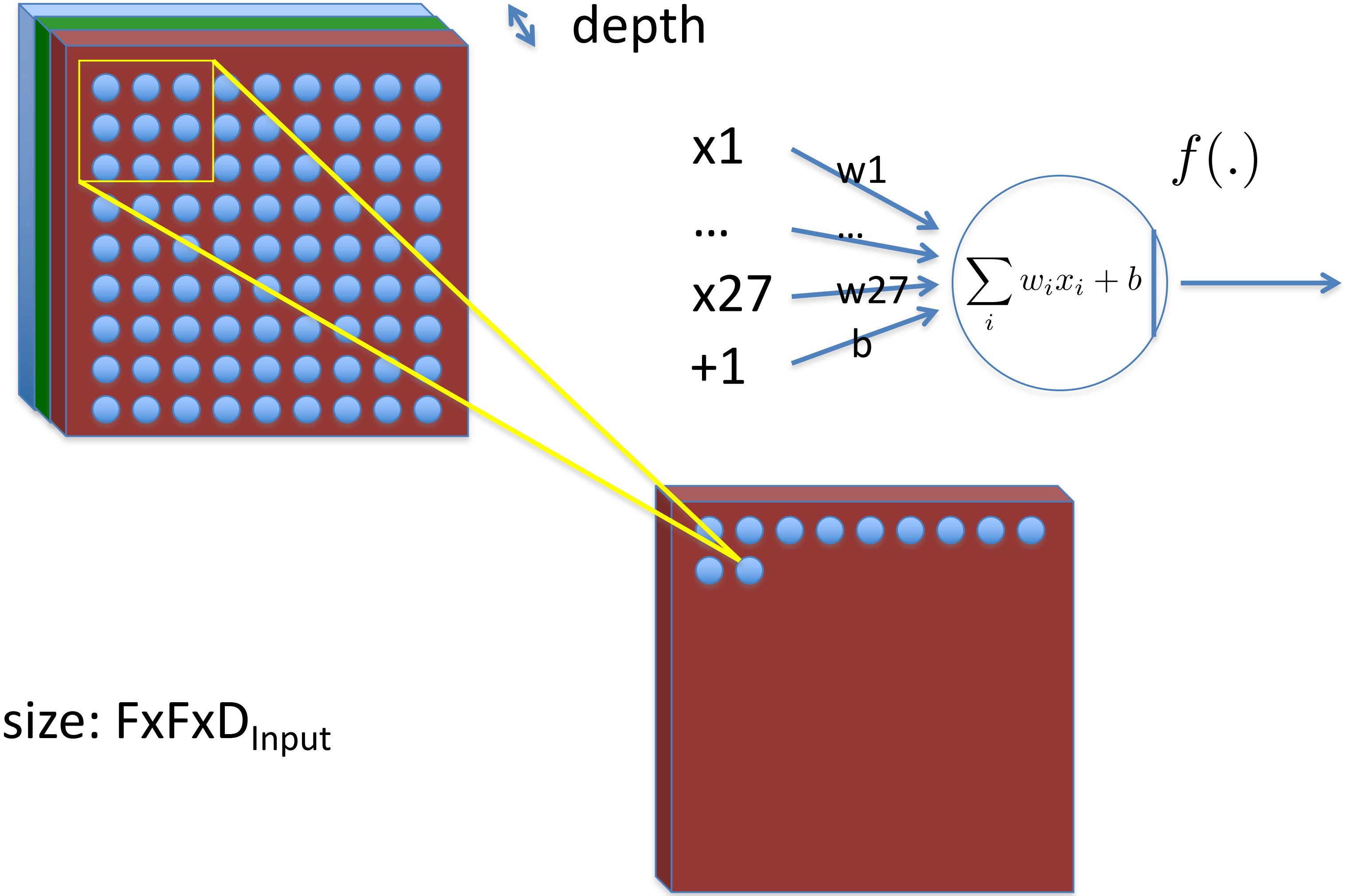
receptive
field

Connections are local in
the spatial dimension

3D input volume

3D output
volume of
neuron
activation

Convolutional Layer



depth

x1

…

x27

+1

w1

…

w27

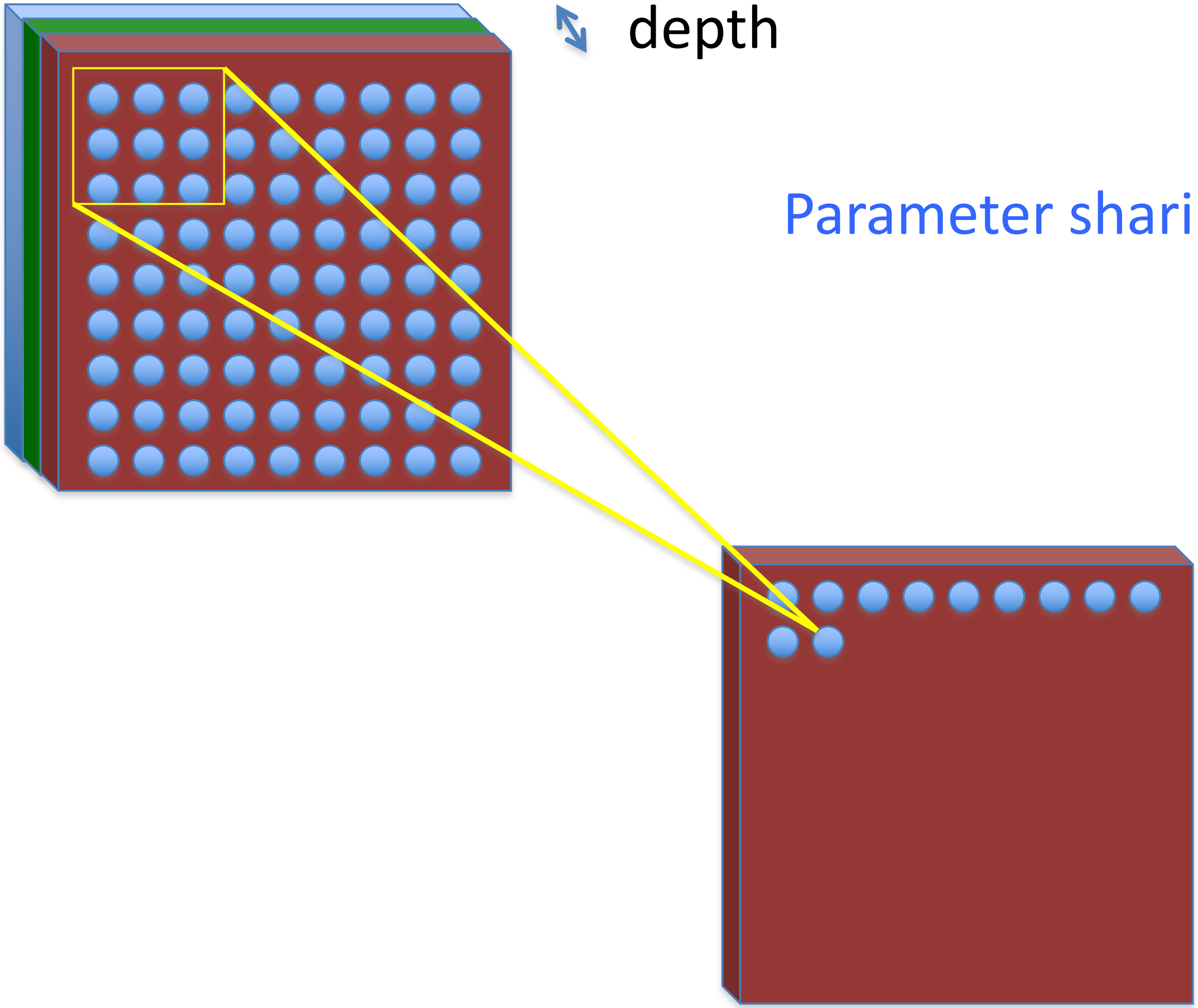
b

$\sum_i w_i x_i + b$

$f(.)$

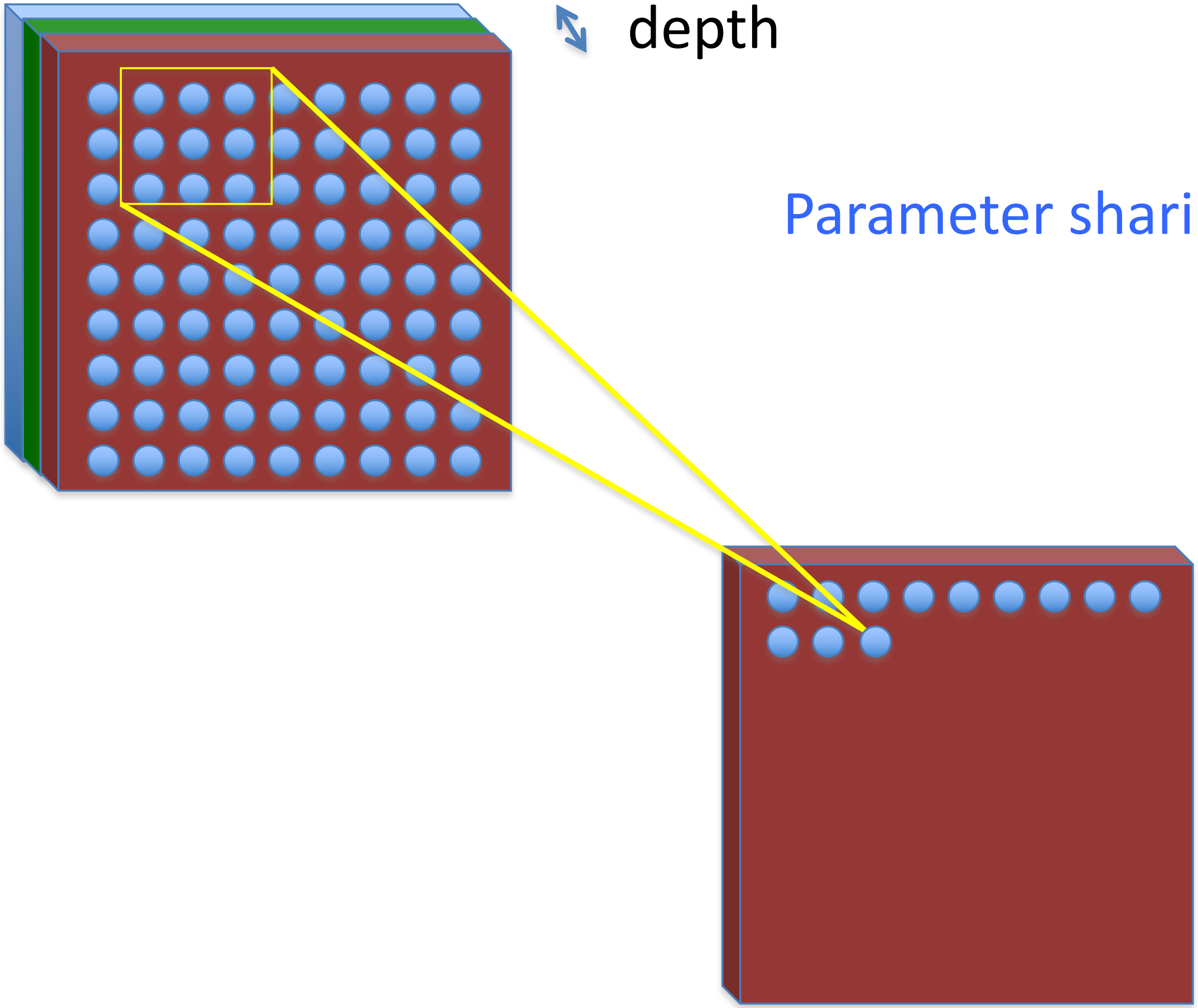
Filter size: FxFxD_{Input}

Convolutional Layer



Parameter sharing

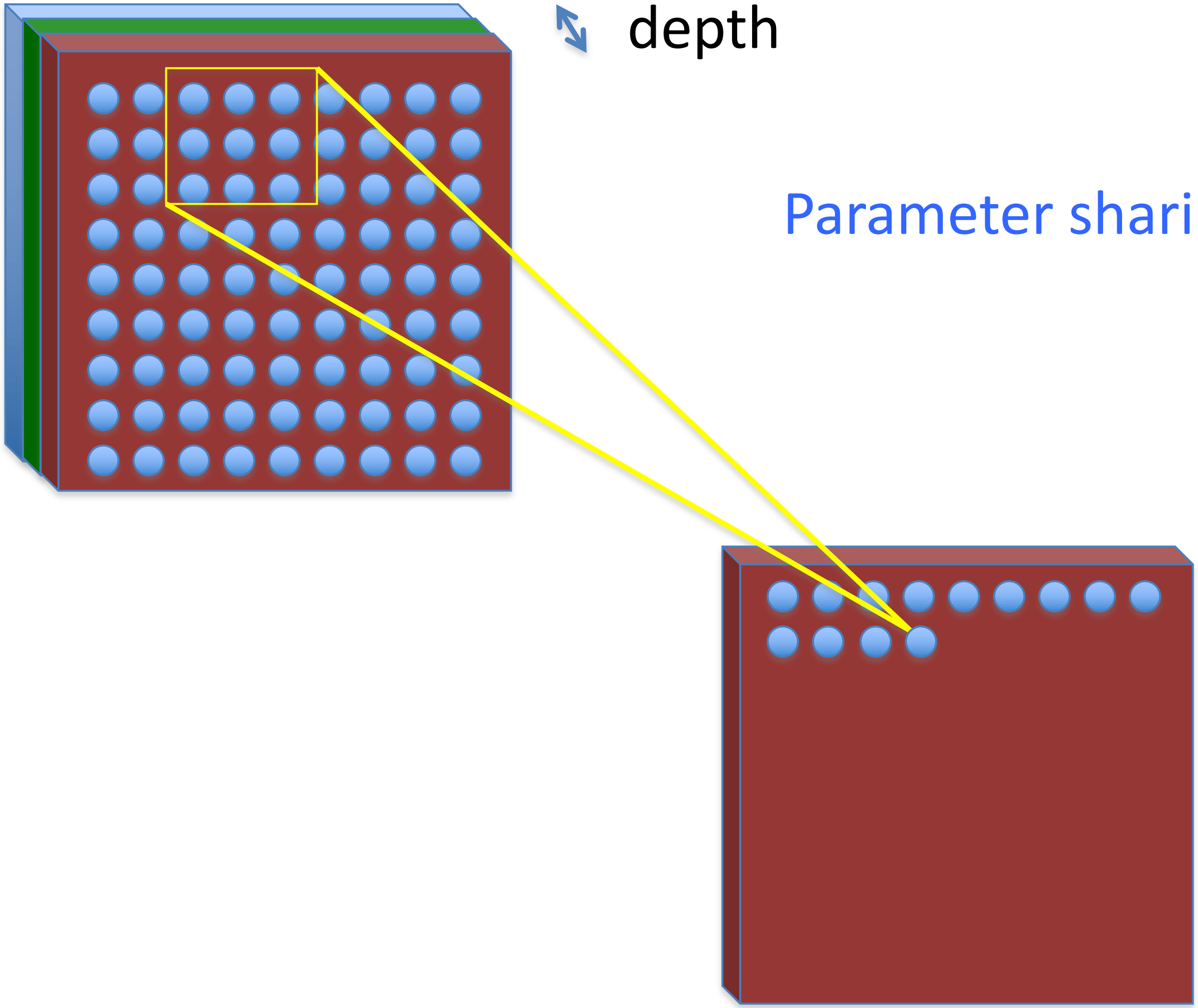
Convolutional Layer



depth

Parameter sharing

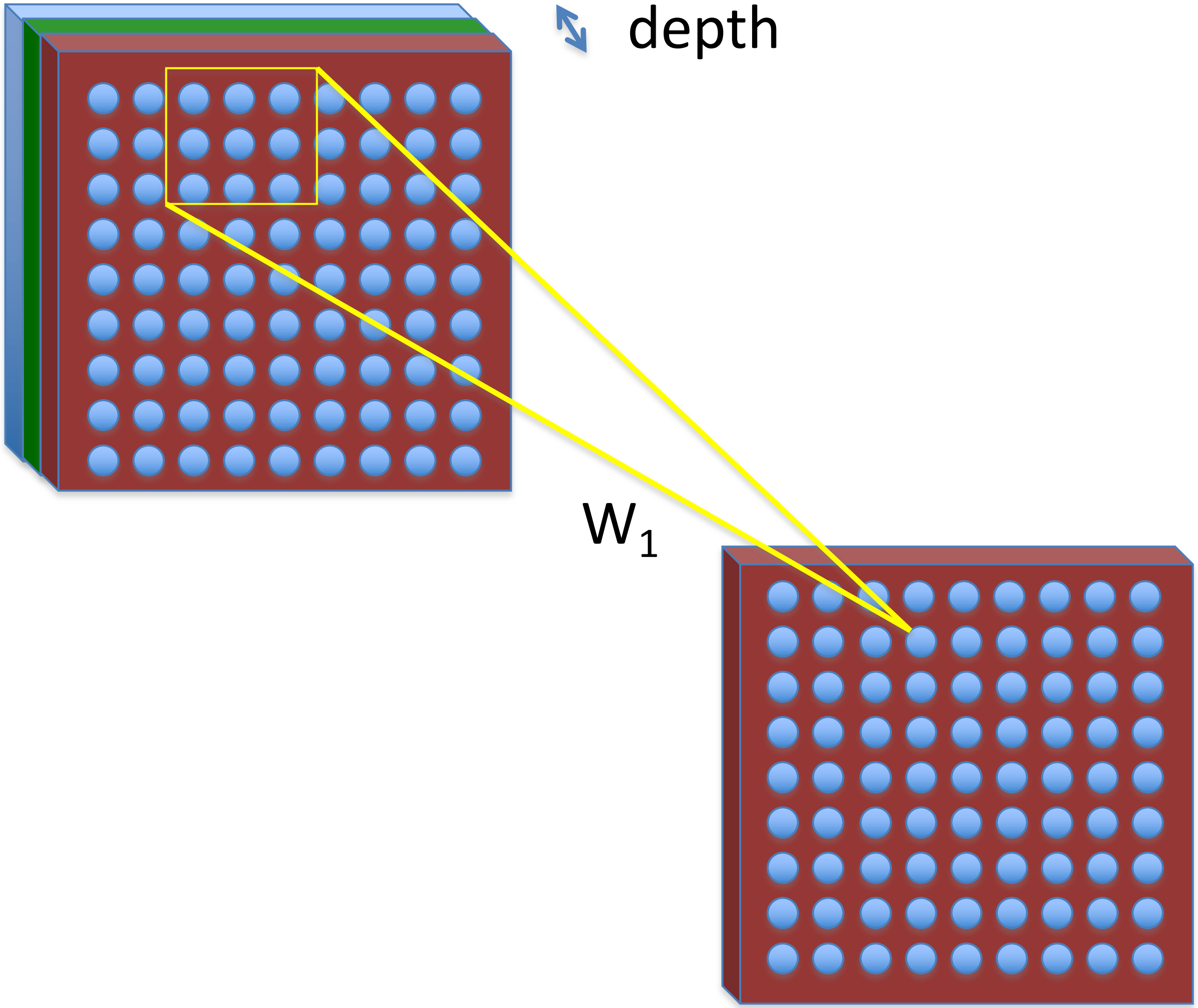
Convolutional Layer



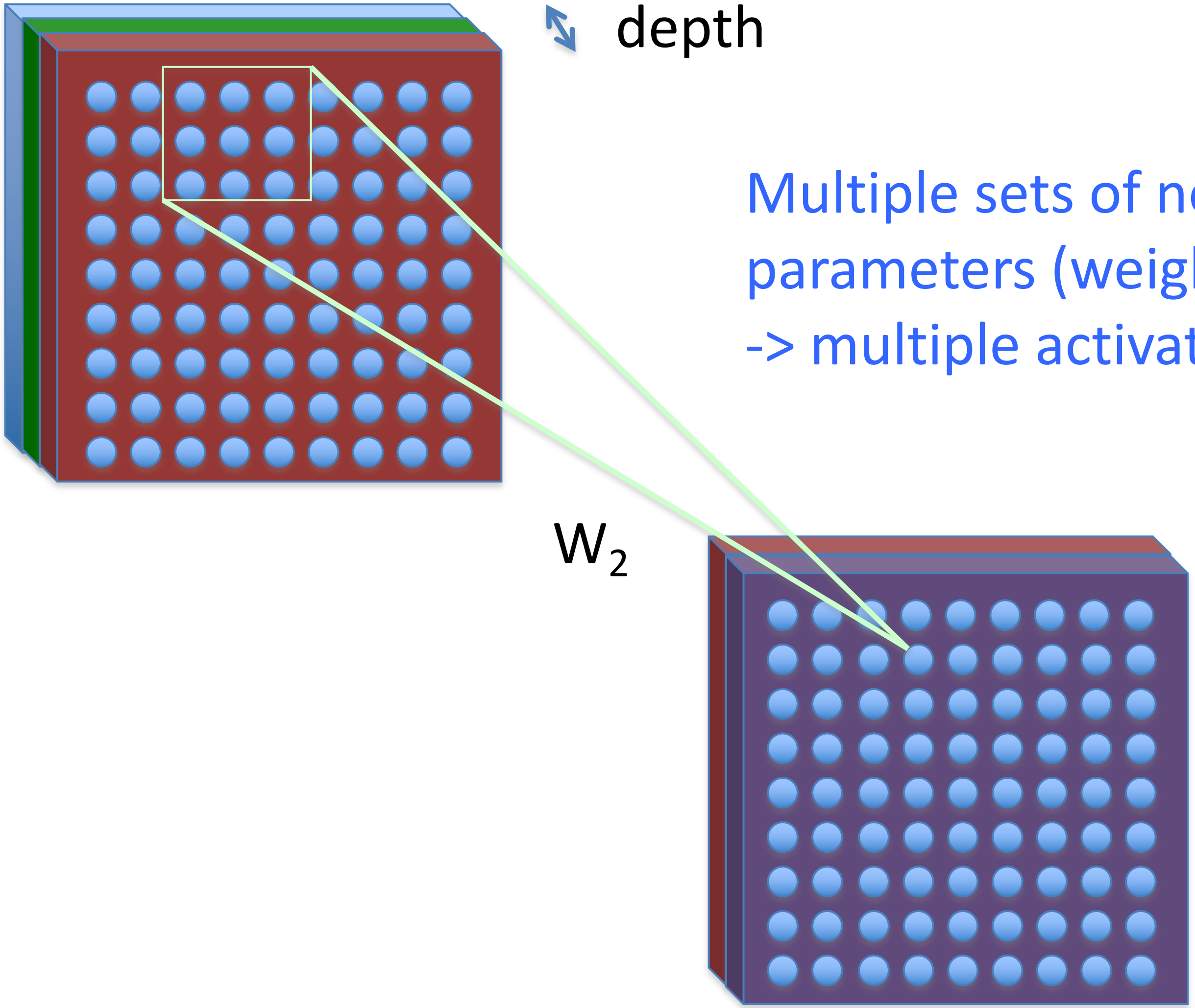
depth

Parameter sharing

Convolutional Layer

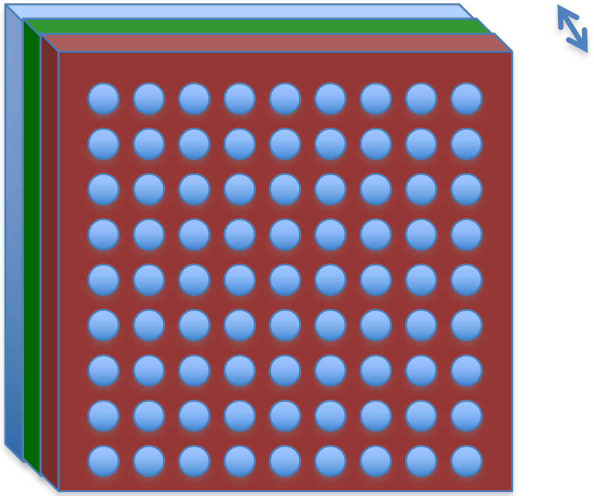


Convolutional Layer



Multiple sets of neuron parameters (weights and bias) -> multiple activation maps

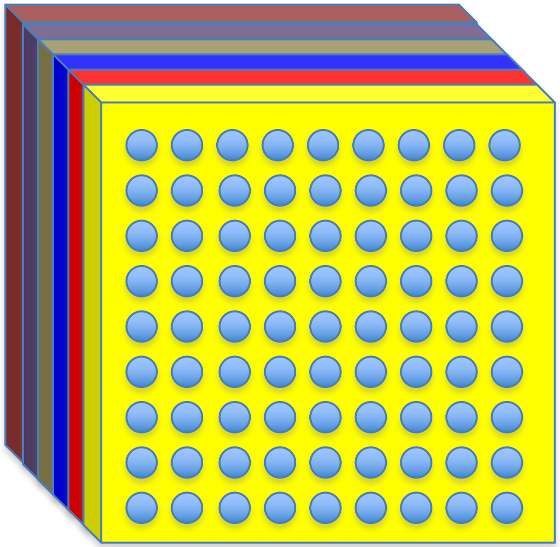
Convolutional Layer



depth

3D input volume

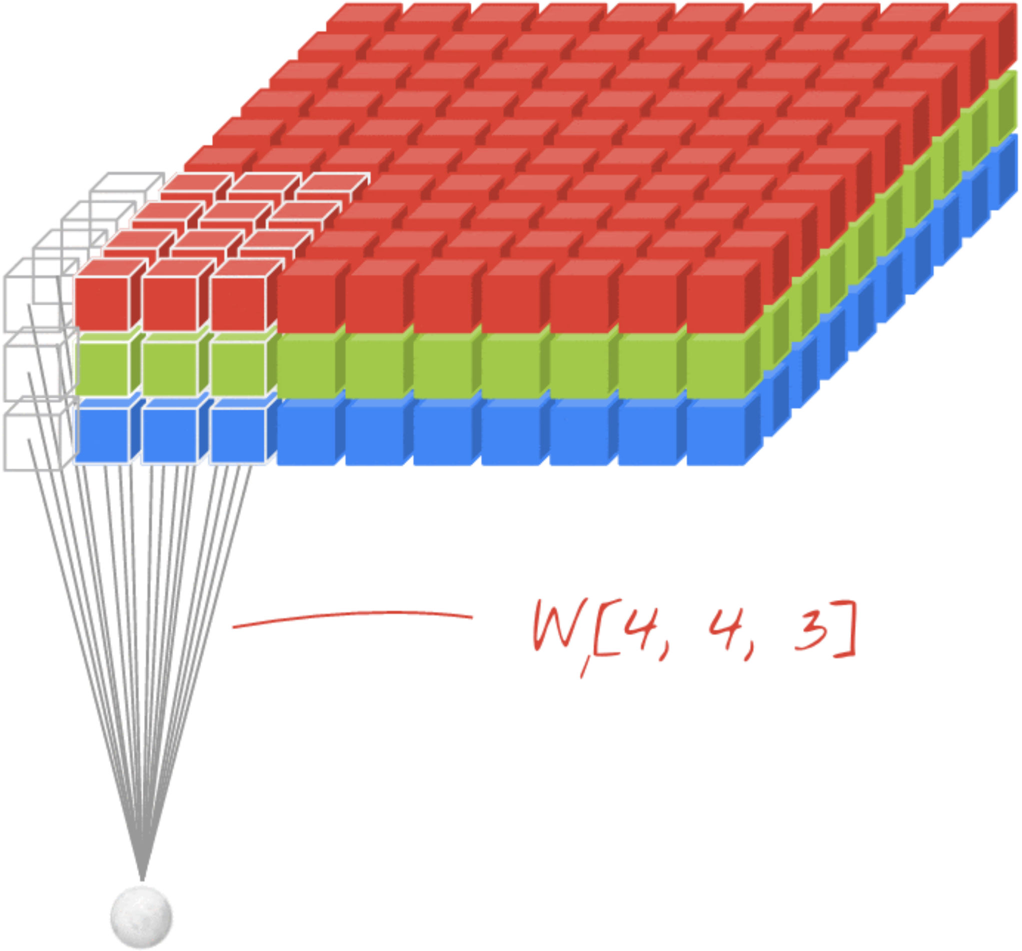
Multiple sets of neuron parameters (weights and bias) -> multiple activation maps



3D output volume of neuron activation

Num of filter kernels = num of output activation maps (depth)

Convolutional Layer



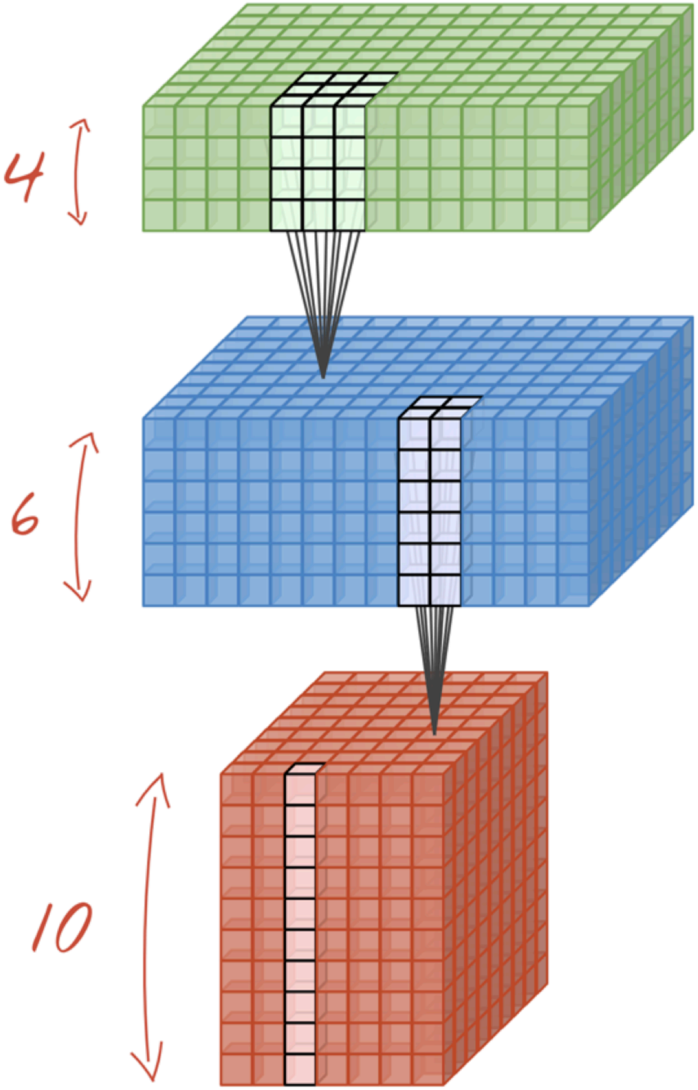
W[4, 4, 3]

$$\sum_{i=1,j=1,c=1}^{i=4,j=4,c=3} x^{p}_{i,j,c} w^{k}_{i,j,c}$$

For each image patch p, $\mathbf{x}^p$ and kernel k, $\mathbf{w}^k$

Image credit: codelabs google

Convolutional in deeper layers



$W_1[3, 3, 4, 6]$

Width x height x channels x # filters

$W_2[2, 2, 6, 10]$

$W_2[1, 1, 10, \dots]$

stride 2

Image credit: codelabs google

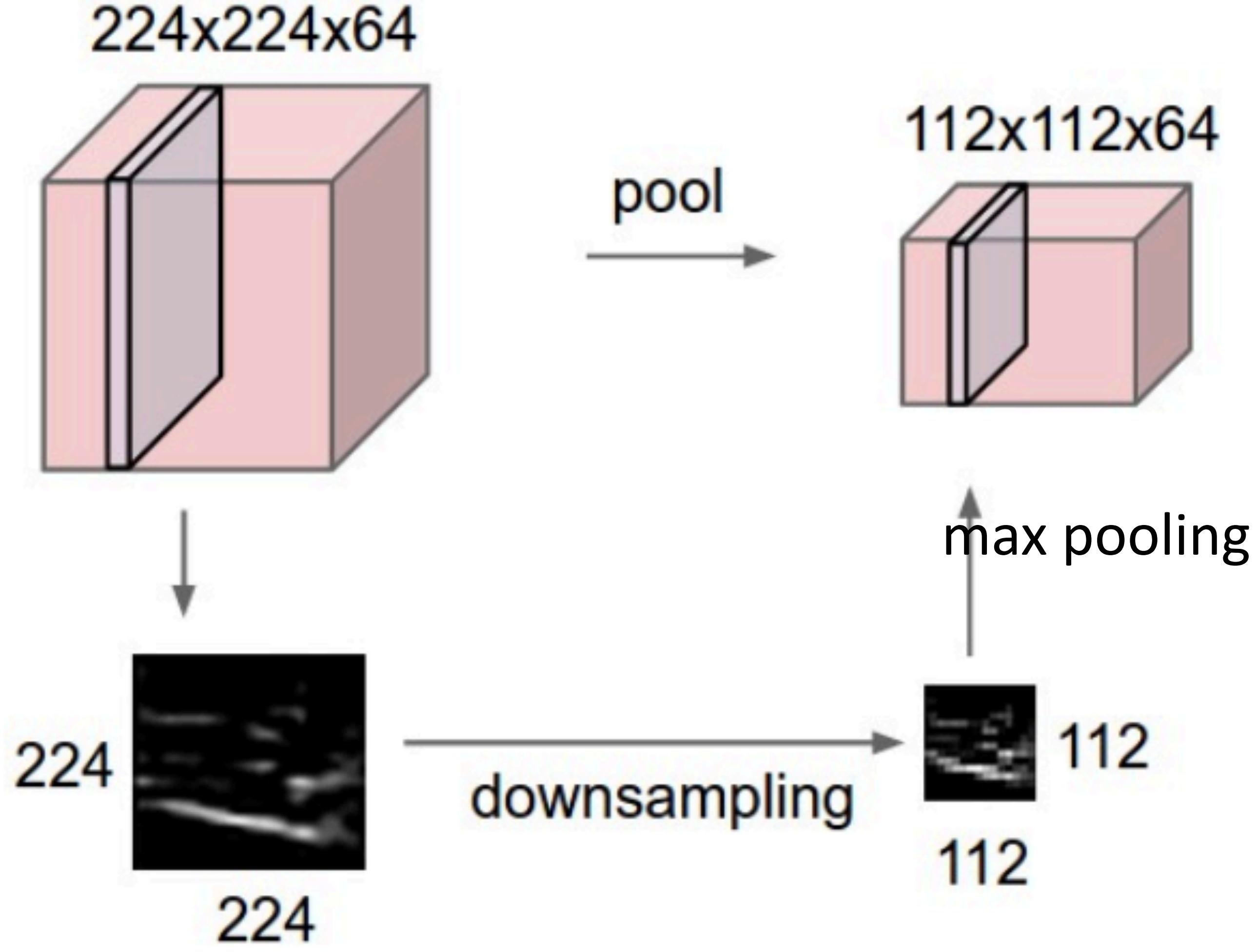
Convolutional Layer

- Filtering (Convolution)
- Matched Filter to identify certain image features
  - Edges or corners (low level layers)
  - Faces or cars (high level layers)
- Assumption of image
  - Locality of pixel dependencies
  - Stationary of image statistics
  - Translation invariance
  - Use the same set of filters for the whole image

Pooling layer

- Progressively reduce the spatial size of the feature map
- Reduce model parameters
- Operate independently on every feature map of the input
- Overlap or non-overlap
- Average or max pooling
- Translation invariant: same pooled feature even when the image undergoes small translations
  - Same label even when the image is translated

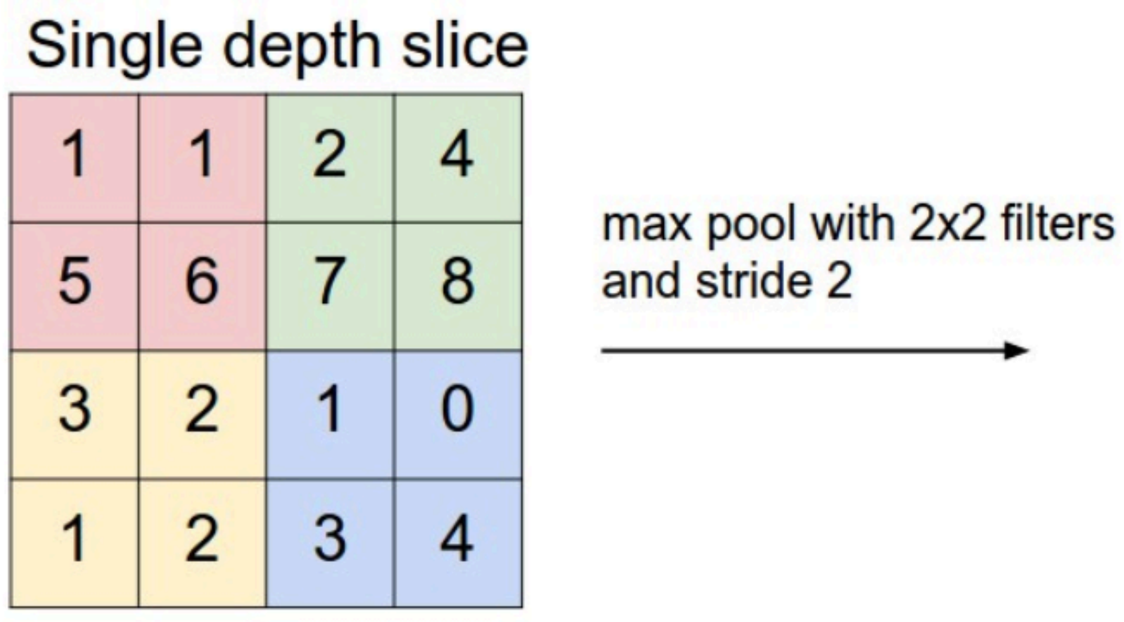
Pooling layer



max pooling

Pooling layer

max pooling



Single depth slice

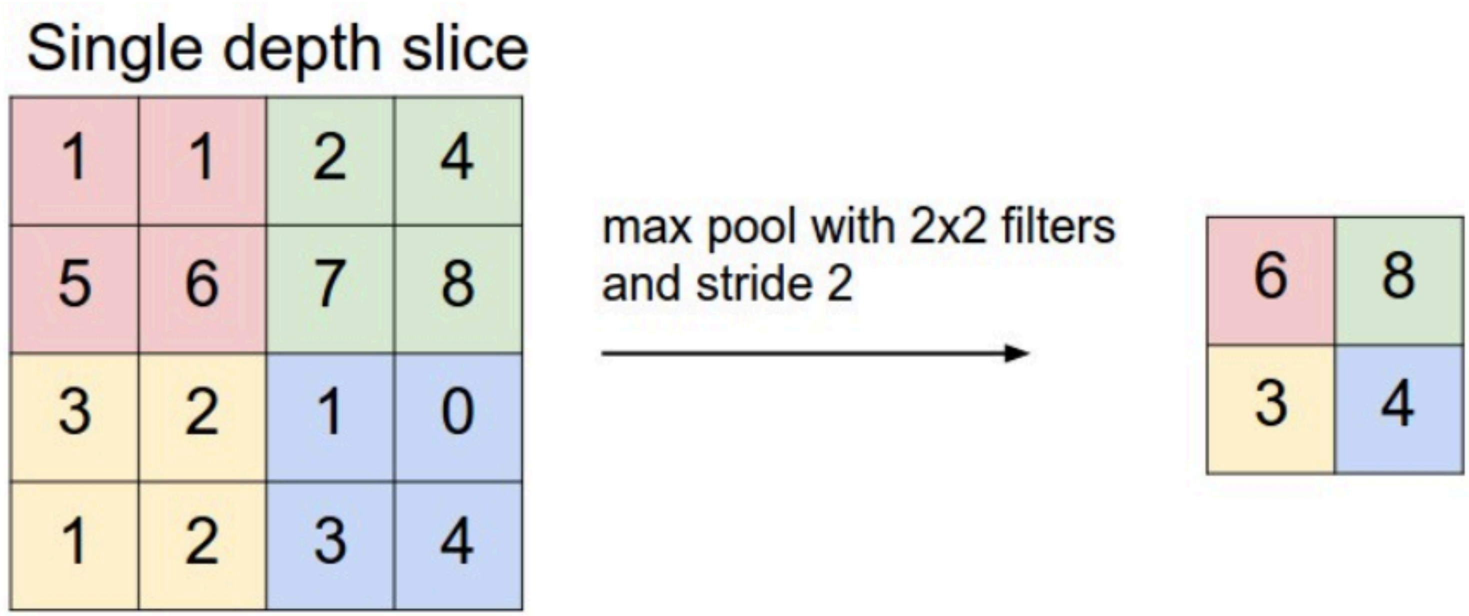
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters
and stride 2

?

Pooling layer

max pooling



Single depth slice

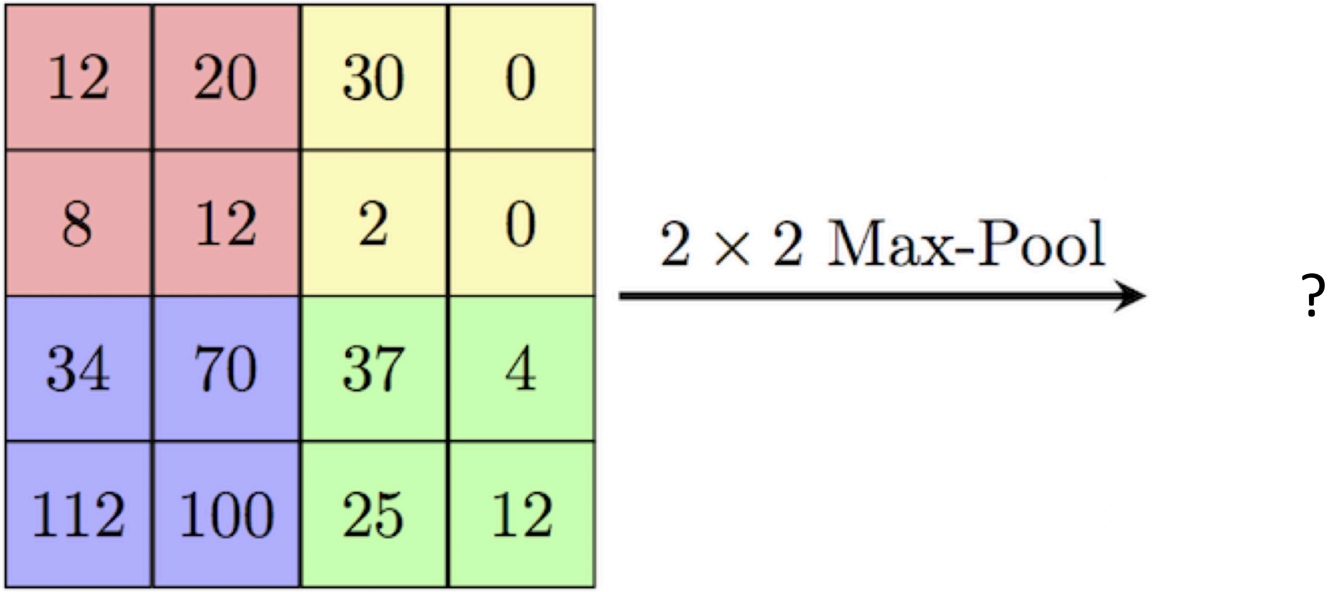
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters
and stride 2

6	8
3	4

Pooling layer

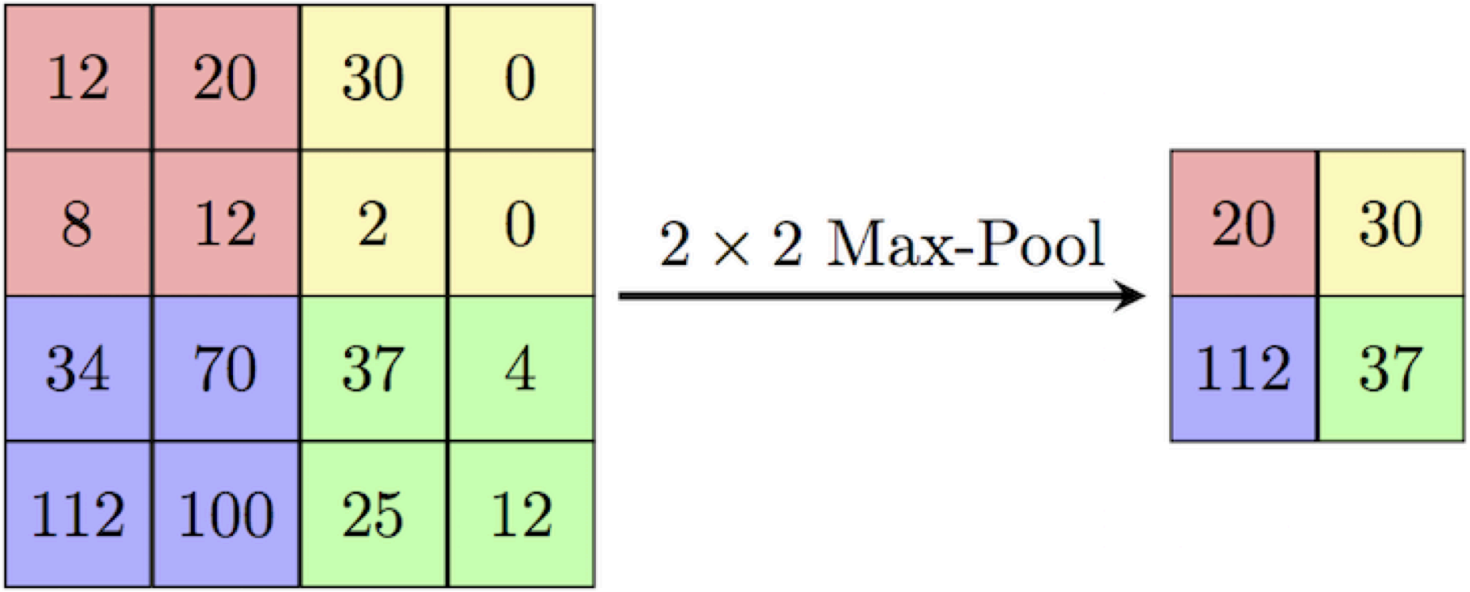
max pooling



12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

Pooling layer

max pooling



12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

2 × 2 Max-Pool

20	30
112	37

QUESTION: How would be the result if applying average pooling instead of max pooling?

Pooling layer

Pooling animation

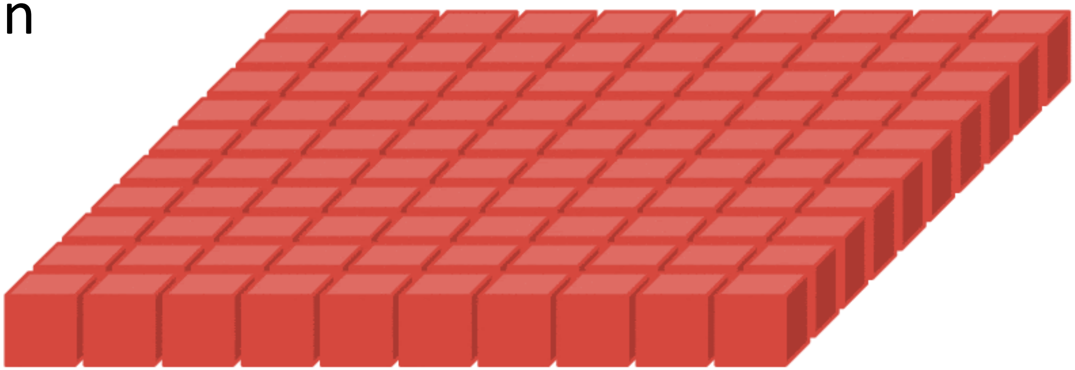
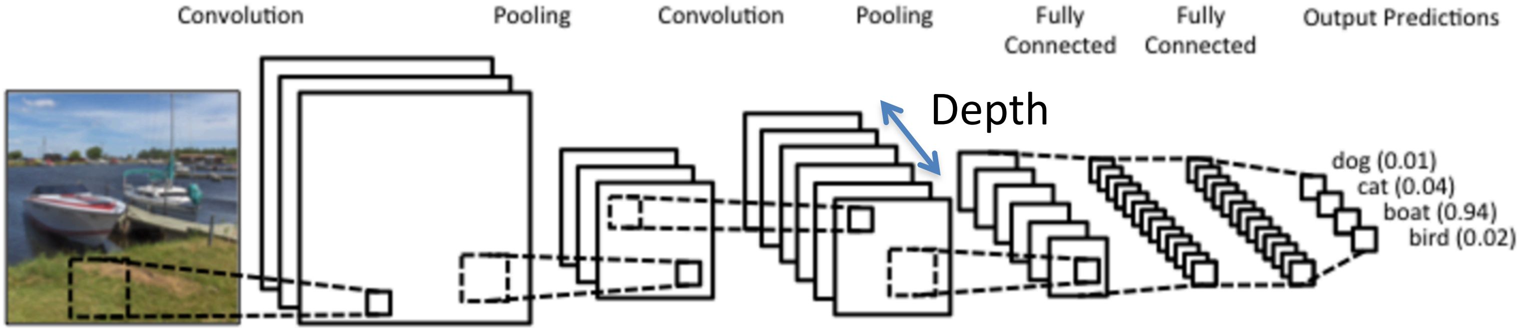


Image credit: codelabs google

CNN

- Stack the layers

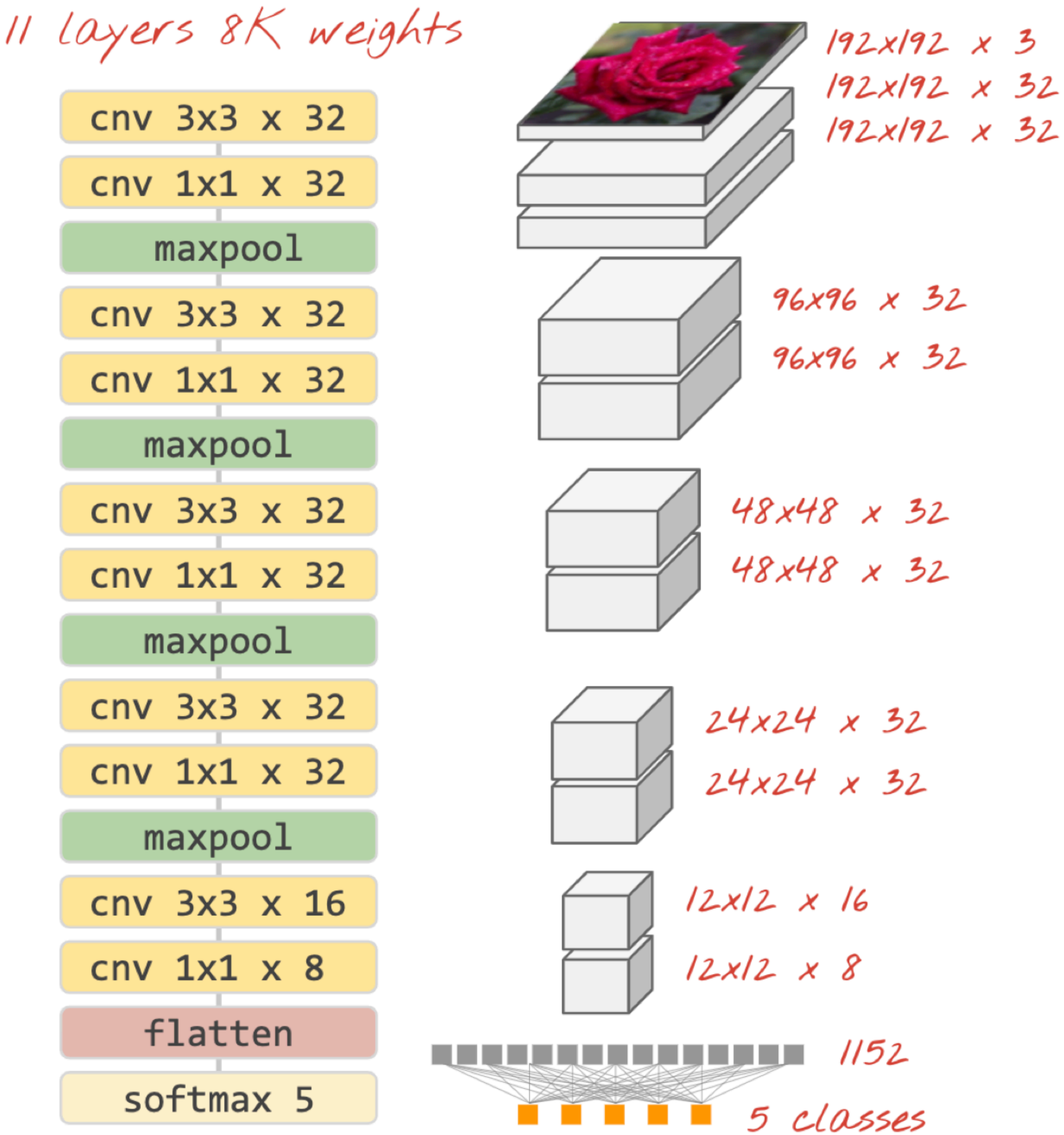


Input:
width x height x numChannel

Output:
1 x 1 x numClass

CNN

- Stack the layers



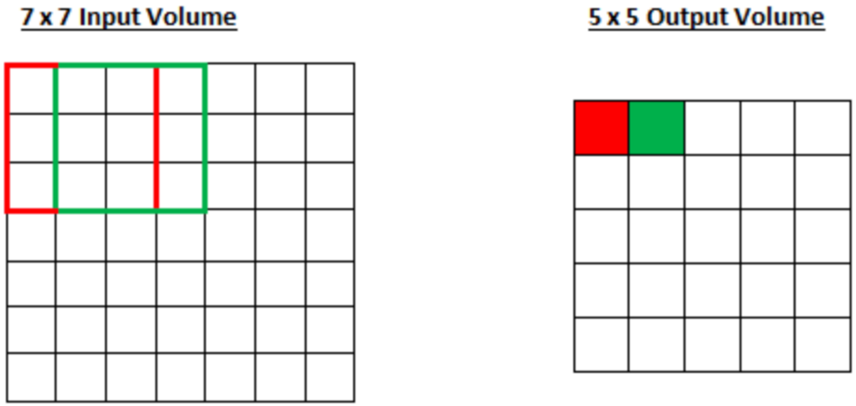
cnv 3x3 x 32
cnv 1x1 x 32
maxpool
cnv 3x3 x 32
cnv 1x1 x 32
maxpool
cnv 3x3 x 32
cnv 1x1 x 32
maxpool
cnv 3x3 x 32
cnv 1x1 x 32
maxpool
cnv 3x3 x 16
cnv 1x1 x 8
flatten
softmax 5

Image credit: codelabs google

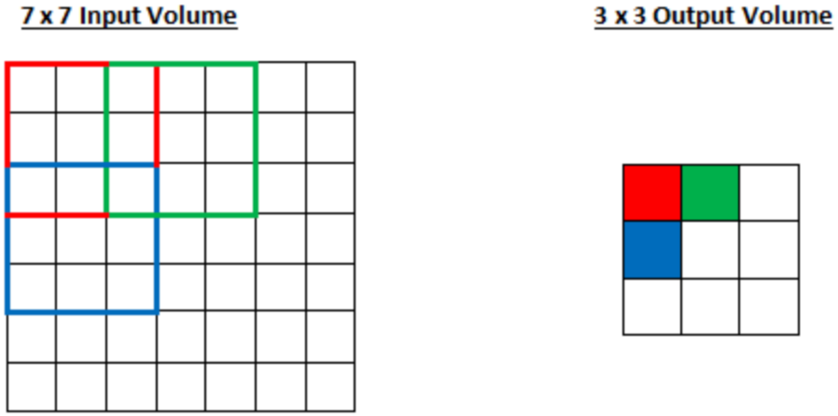
Stride

- Stride = the number of pixels (input units) by which the filter shifts

Stride = 1

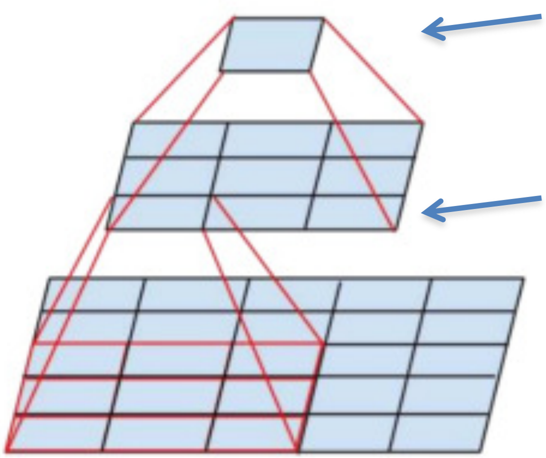


Stride = 2



Receptive field

Receptive field: part of the image that is visible to a neuron
Inspired by visual cortex architecture



Stride = 1

Conv 2

Conv 1

Input layer (image)

- 3x3 receptive field size w.r.t. conv1
- 5x5 receptive field size w.r.t. input image

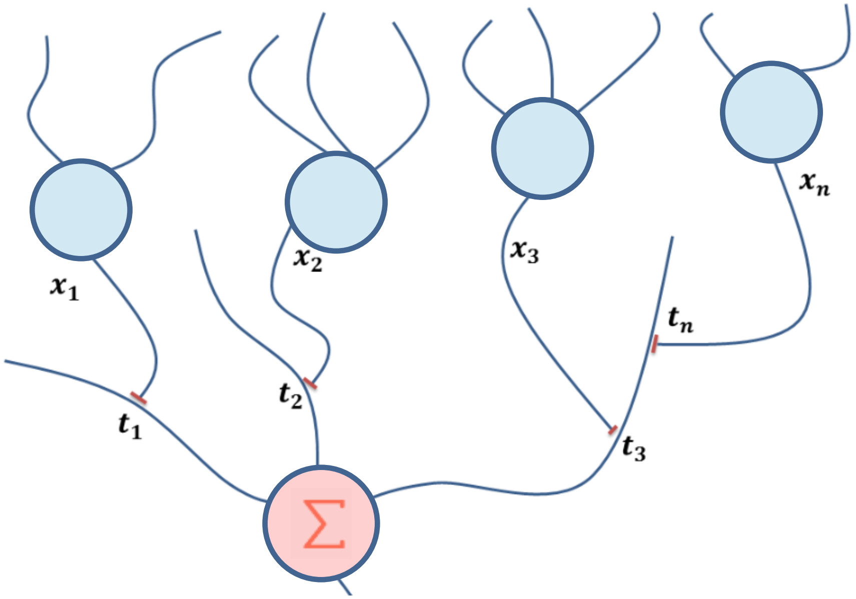
3x3 receptive field size (filter size)

Although connections are local, neurons in the higher layers could see large portions of the image (able to recognize object)

CNN – inspired by neuroscience

Simplified neuroscience: a neuron computes a dot product between its inputs and the synaptic weights

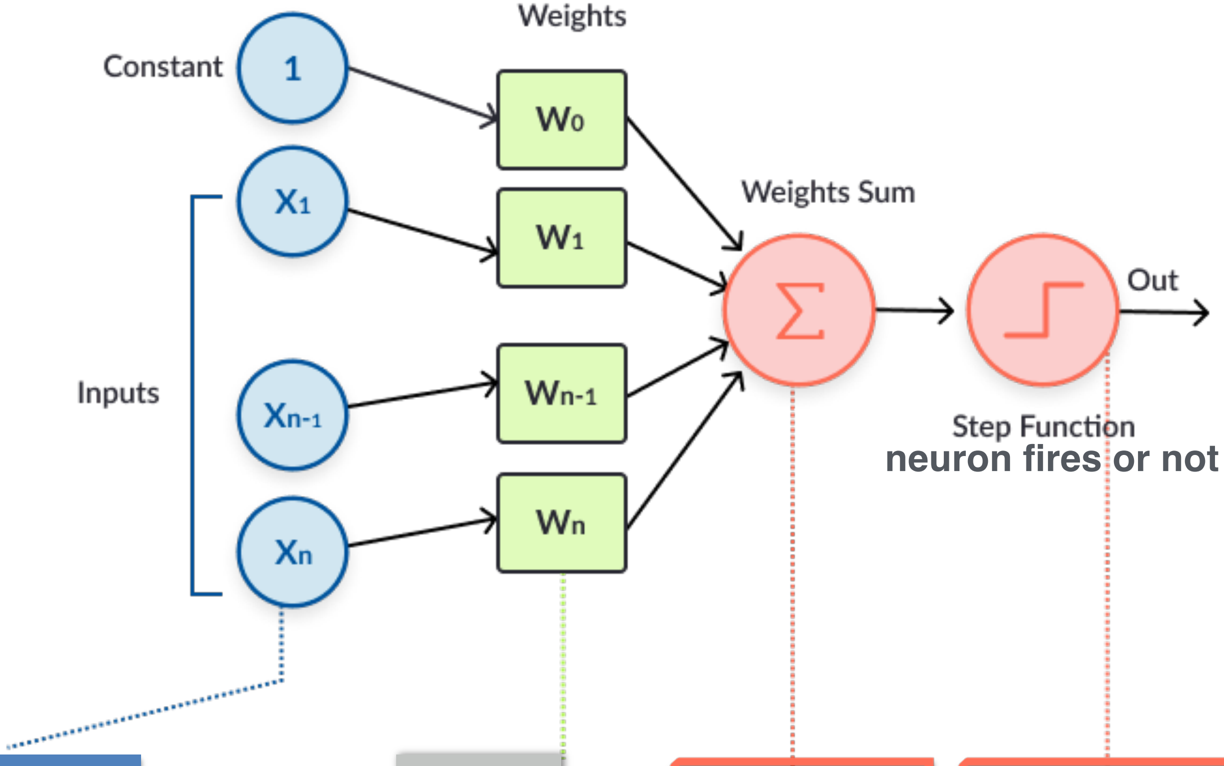
$$< x, t >= \sum_{i=1}^{n} x_i t_i \longleftrightarrow$$



Simple perceptron

F. Rosenblatt 1957

- One layer NN



neuron fires or not

Input +
constant for bias

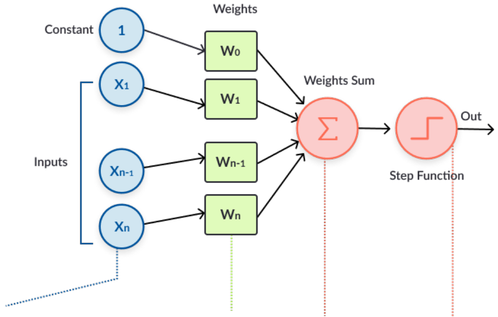
Weights
Learned

$$\sum_{i=0}^{n} x_i w_i$$

$$Out = sgn(\sum_{i=0}^{n} x_i w_i)$$

Image credit: missinglink.ai

Simple perceptron



1. Takes the inputs which are fed into the perceptrons in the input layer, multiplies them by their weights, and computes the sum.

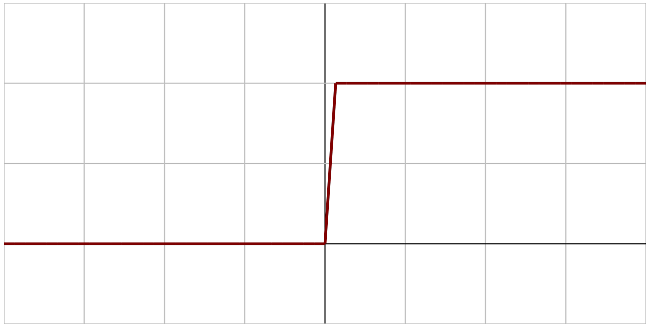
2. Adds the number one, multiplied by a “bias weight”. This is a technical step that makes it possible to move the output function of each perceptron (the activation function) up, down, left and right on the number graph.

3. Feeds the sum through the activation function—in a simple perceptron system, the activation function is a step function.

4. The result of the step function is the output.

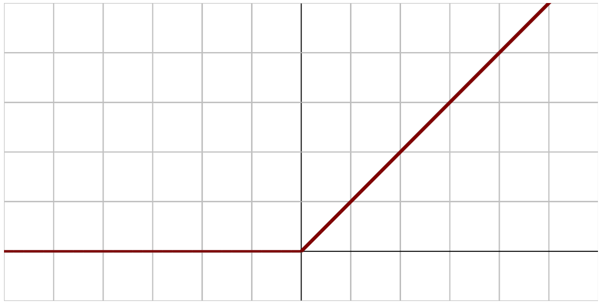
Simple perceptron

Types of Nonlinearities



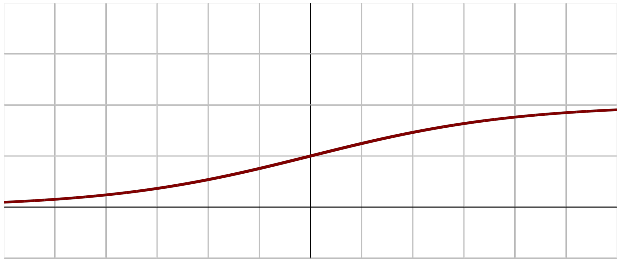
Step function

$$f(x) = \begin{cases} 0 & : x < 0 \\ 1 & : x \geq 0 \end{cases}$$



Linear Rectifier (ReLu)

$$f(x) = \begin{cases} 0 & : x < 0 \\ x & : x \geq 0 \end{cases}$$

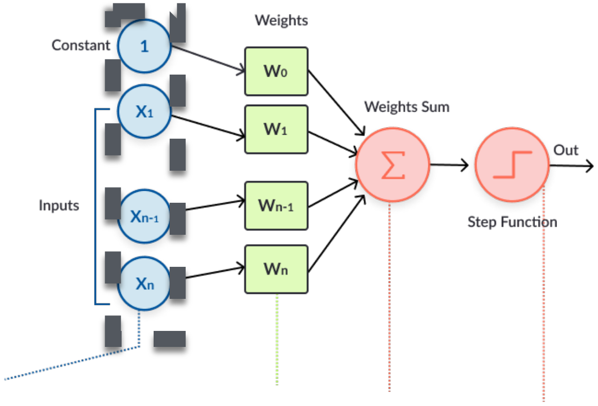


Sigmoid

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

etc.

Simple perceptron



Given training samples $\{\mathbf{x}_i, y_i\}_{\forall i}$

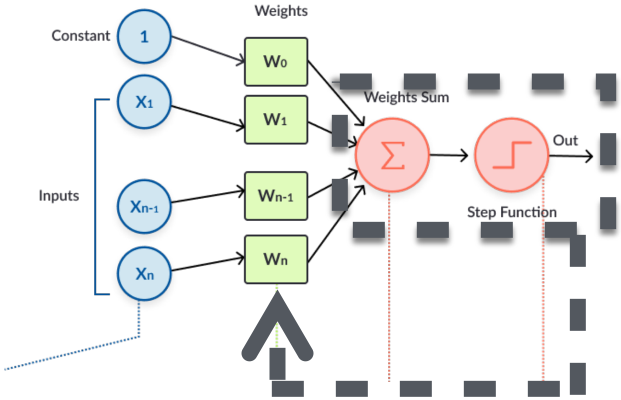
$\mathbf{x}_i$ -> input of example *i*,

y_i -> groundtruth target of example *i*

Simple perceptron

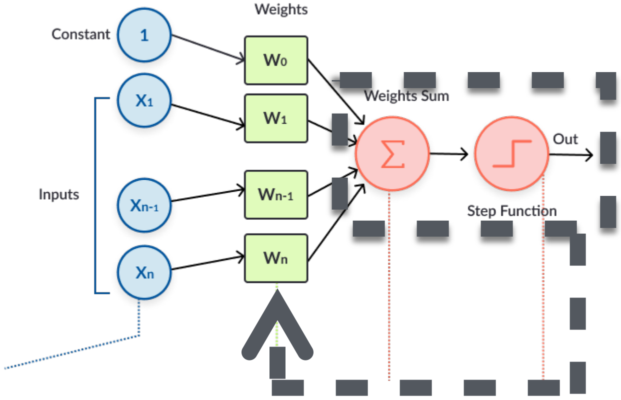
Initialization:

Initialize the weights W to 0 or small random numbers.



Simple perceptron

Initialization:



Initialize the weights W to 0 or small random numbers.

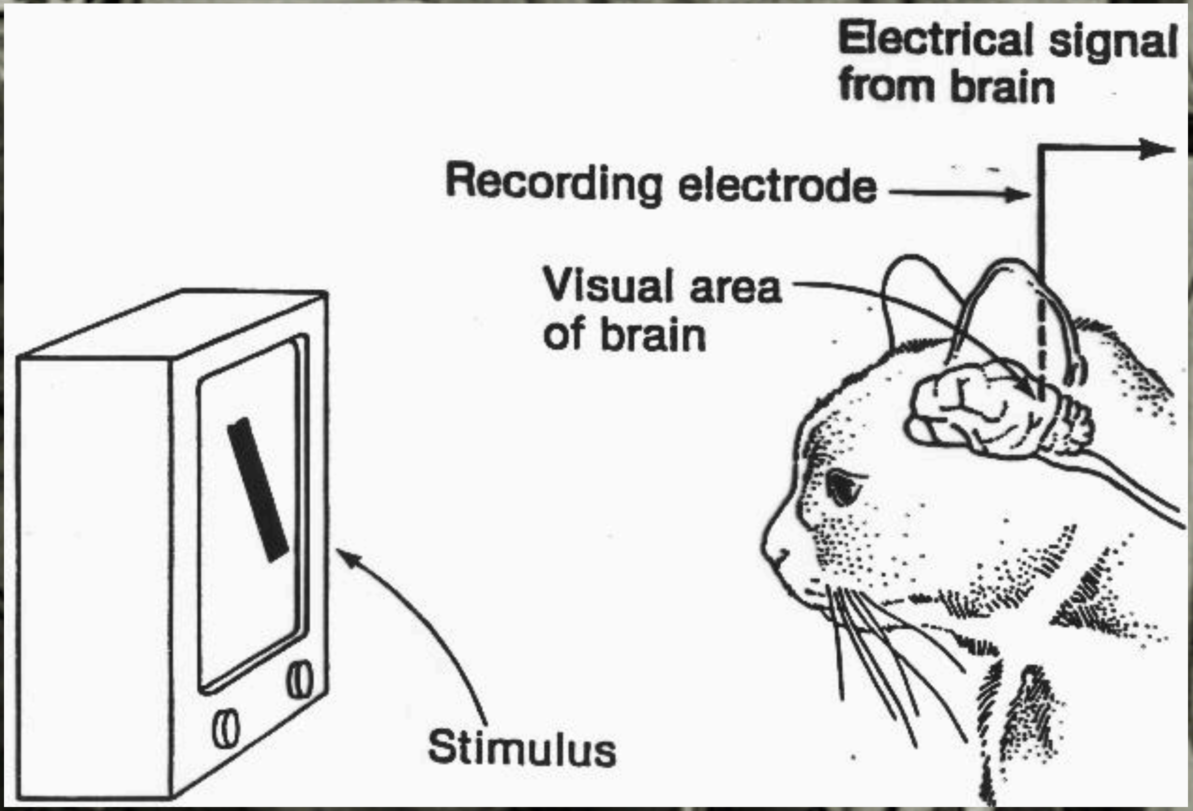
Iterate:

For each training sample $\mathbf{x}_i$:

1. Calculate the output value: $out = sgn(\sum_{i=0}^{n} x_i w_i)$
2. Update the weights. $\mathbf{w} = \mathbf{w} + \eta \mathbf{x_i}(y_i - out)$

In case of linear separable data, the learning converges in a bounded number of iterations.

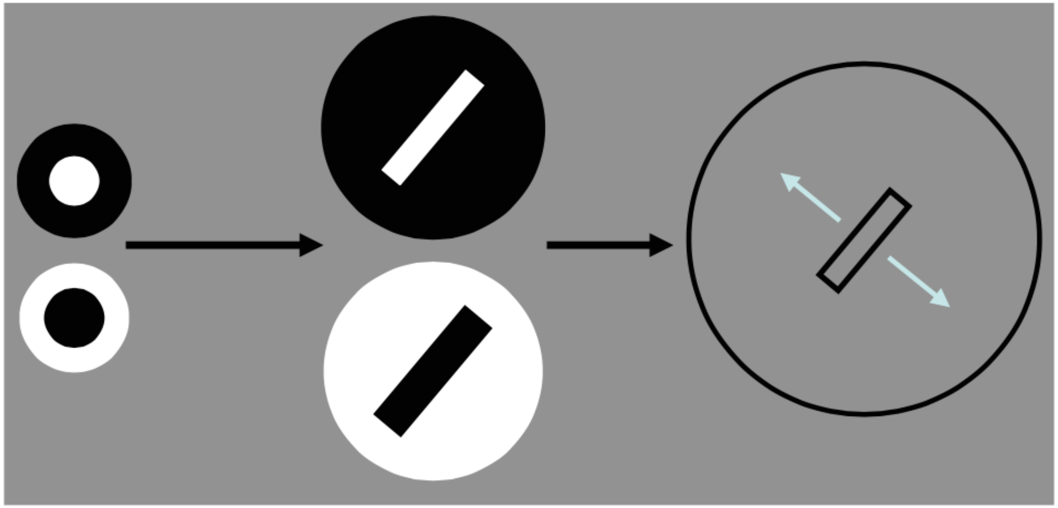
Hubel and Wiesel



Nobel prize

(1981)

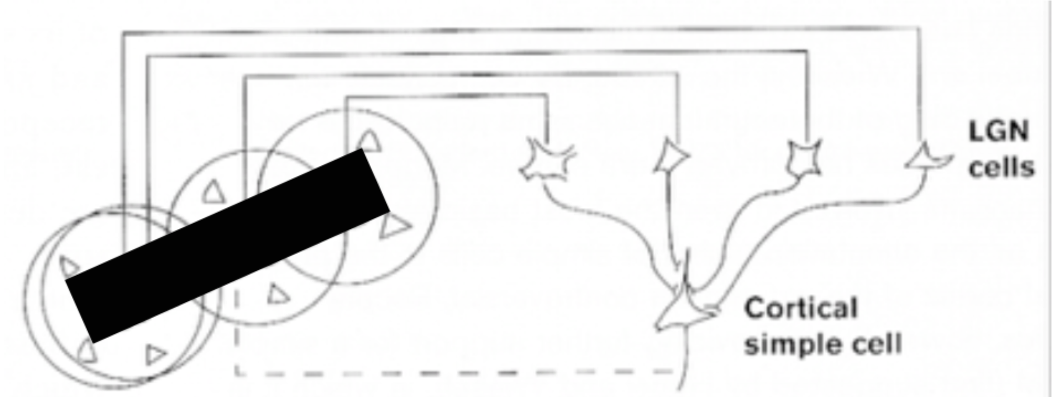
Hubel and Wiesel

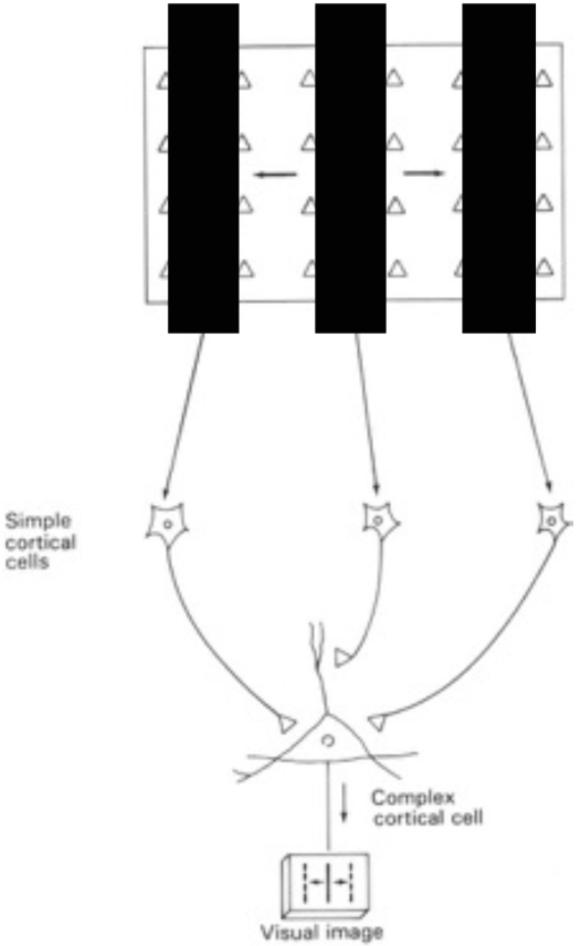


LGN−type cells

Simple cells

Complex cells





(Hubel & Wiesel 1959)

Simple and Complex Cells

- Tuning operation (Gaussian-like, AND-like)

$$y = e^{-|x-w|^2}$$

$$or$$

$$y \sim \frac{x \bullet w}{|x|}$$

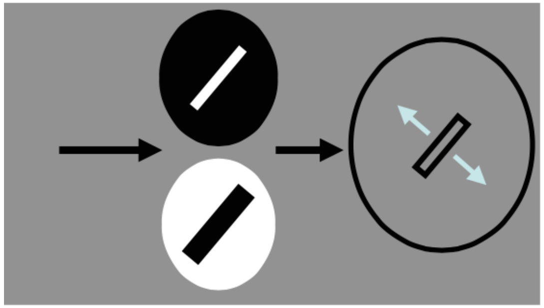
- Simple units

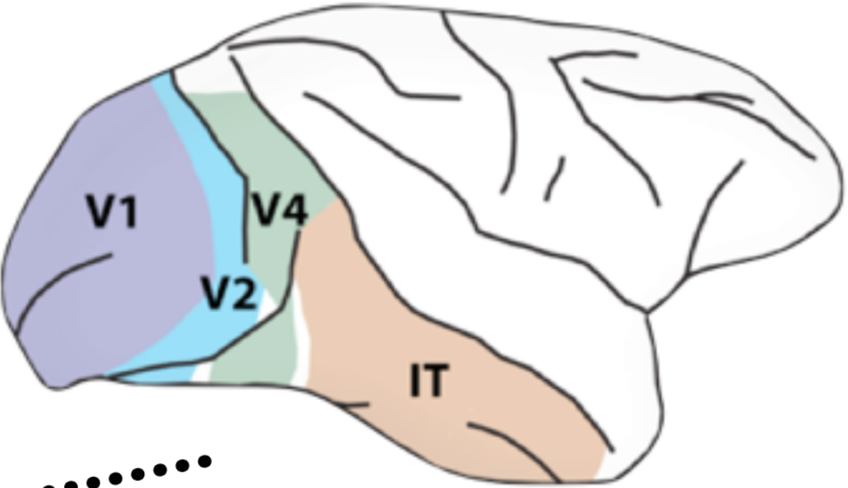
- Max-like operation (OR-like)

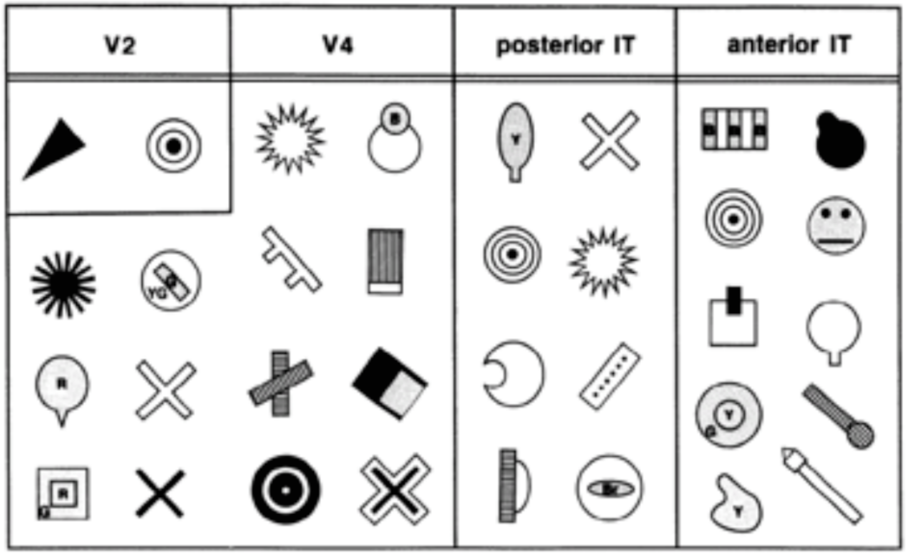
$$y = \max\{x1, x2, \ldots\}$$

- Complex units

The visual ventral stream



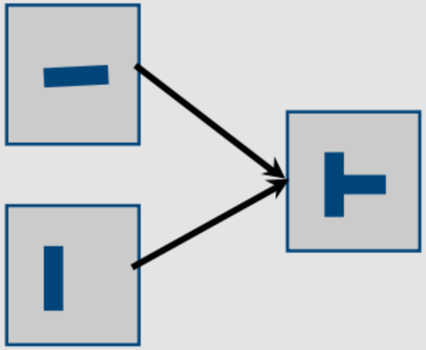
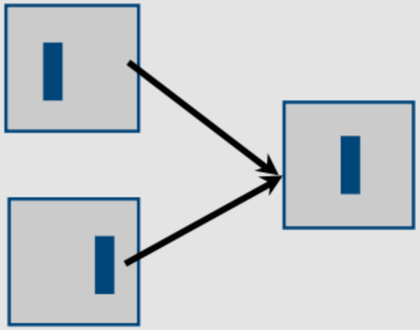




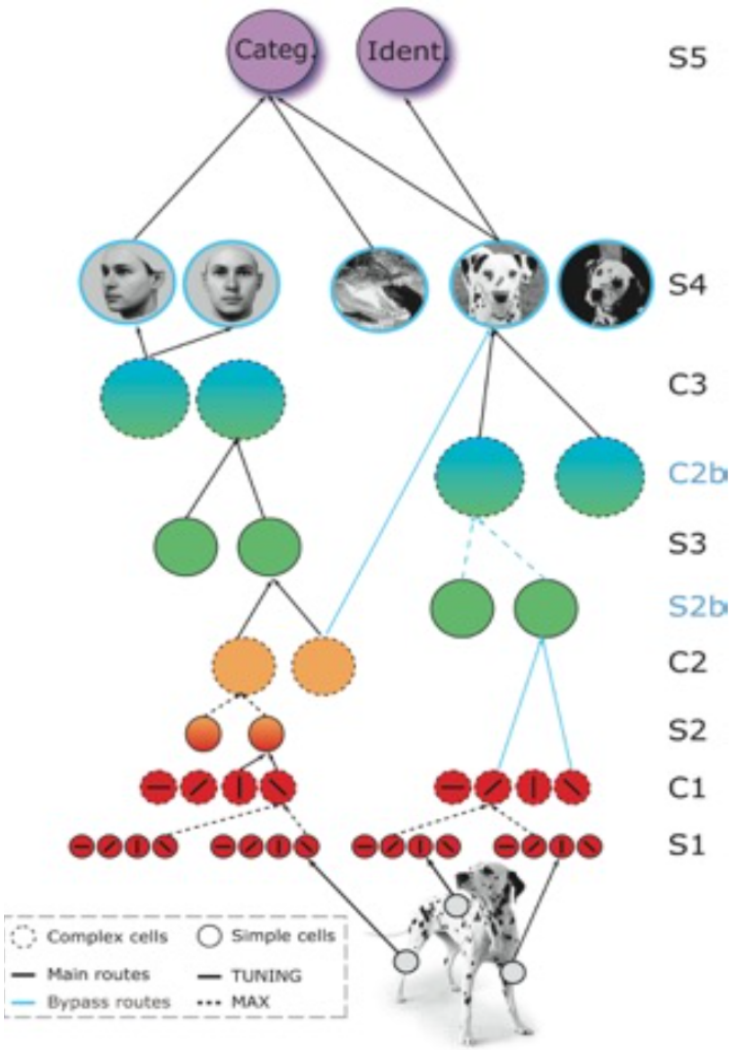
The ventral stream hierarchy: V1, V2, V4, IT
A gradual increase in the receptive field size, in the complexity of the preferred stimulus, in tolerance to position and scale changes

Kobatake & Tanaka, 1994

Simple and Complex Cells

Unit	Pooling	Computation
Simple		Selectivity / template matching
Complex		Invariance

HMAX



Riesenhuber & Poggio 1999, 2000; Serre Kouh Cadieu Knoblich Kreiman & Poggio 2005; Serre Oliva Poggio 2007

Two operations (~OR, ~AND): disjunctions of conjunctions

- Tuning operation (Gaussian-like, AND-like)

$$y = e^{-|x-w|^2}$$

or

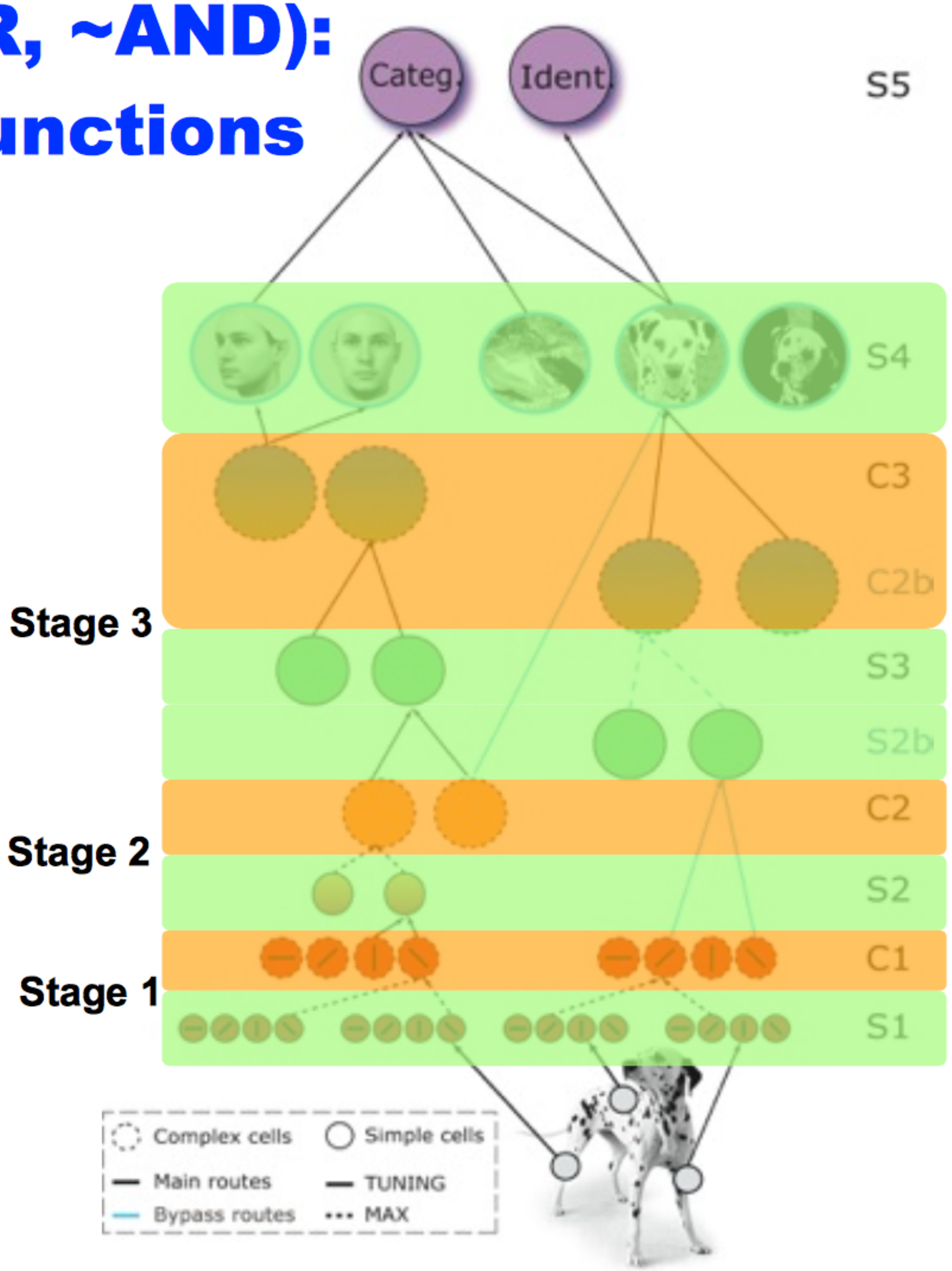
$$y \sim \frac{x \bullet w}{|x|}$$

- Simple units

- Max-like operation (OR-like)

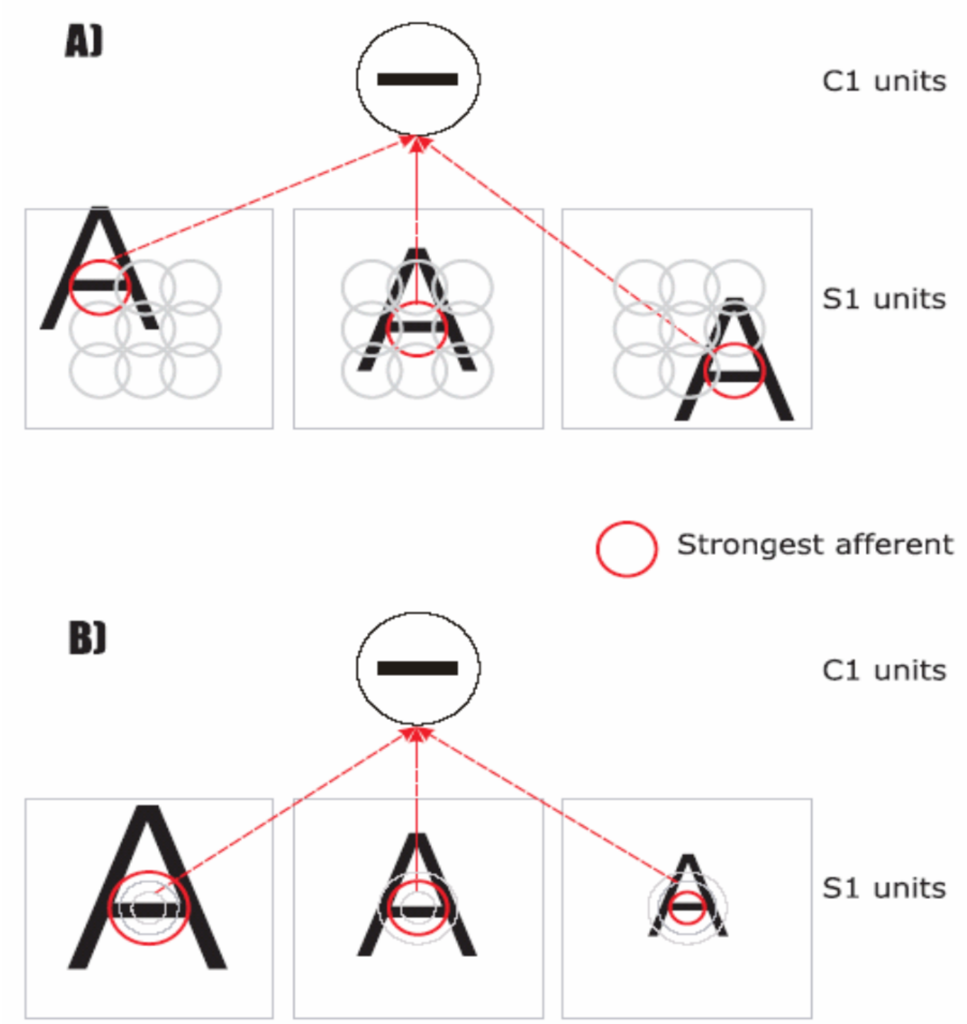
$$y = \max\{x1, x2, \ldots\}$$

- Complex units



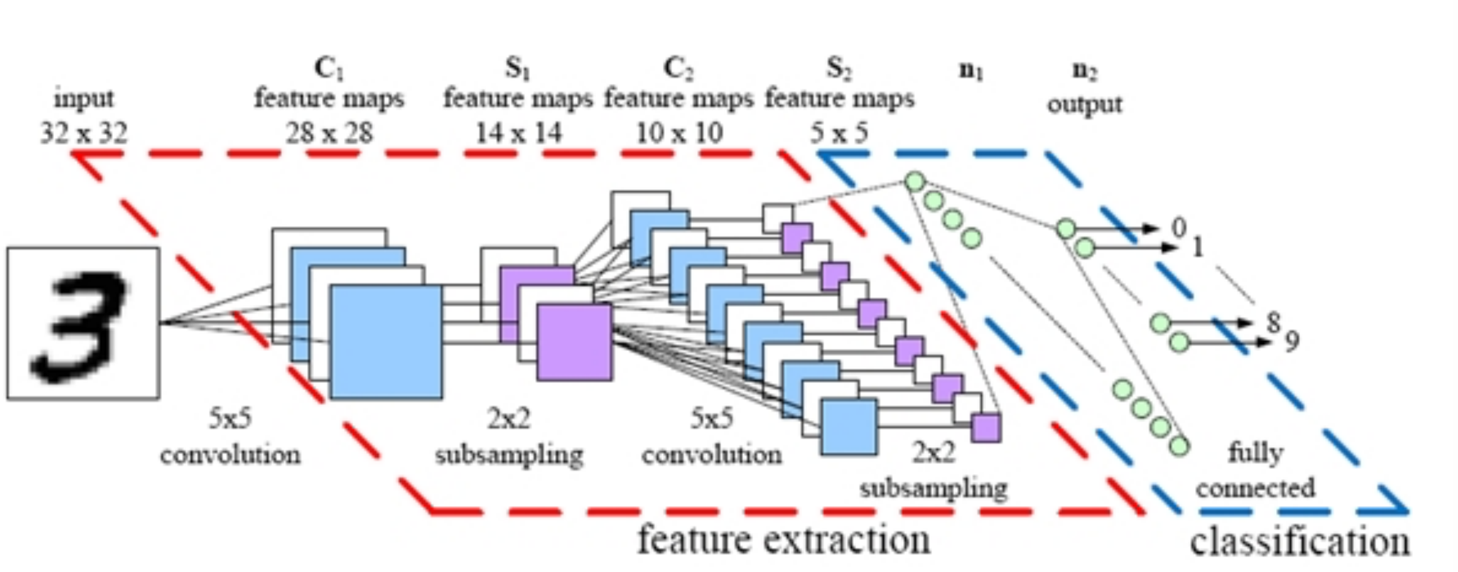
Each operation ~microcircuits of ~100 neurons

Invariance



Serre, T., and Riesenhuber, M. (2004)

Convolutional Neural Networks (CNNs)



Convolutional assumption

LeCun et al. 98

Today’s class

- Deep Learning
- Convolutional Neural Networks

Next week's class

- CNN architectures