Deep Learning

Sewoong Oh

CSE/STAT 416
University of Washington

- Feature engineering is critical in achieving good performance
- e.g. seasonal trends captured by sinusoids

$$f(x) \;=\; w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \tilde{w}_4 \sin\left(\frac{2\pi x}{12}\right) + \tilde{w}_5 \cos\left(\frac{2\pi x}{12}\right)$$

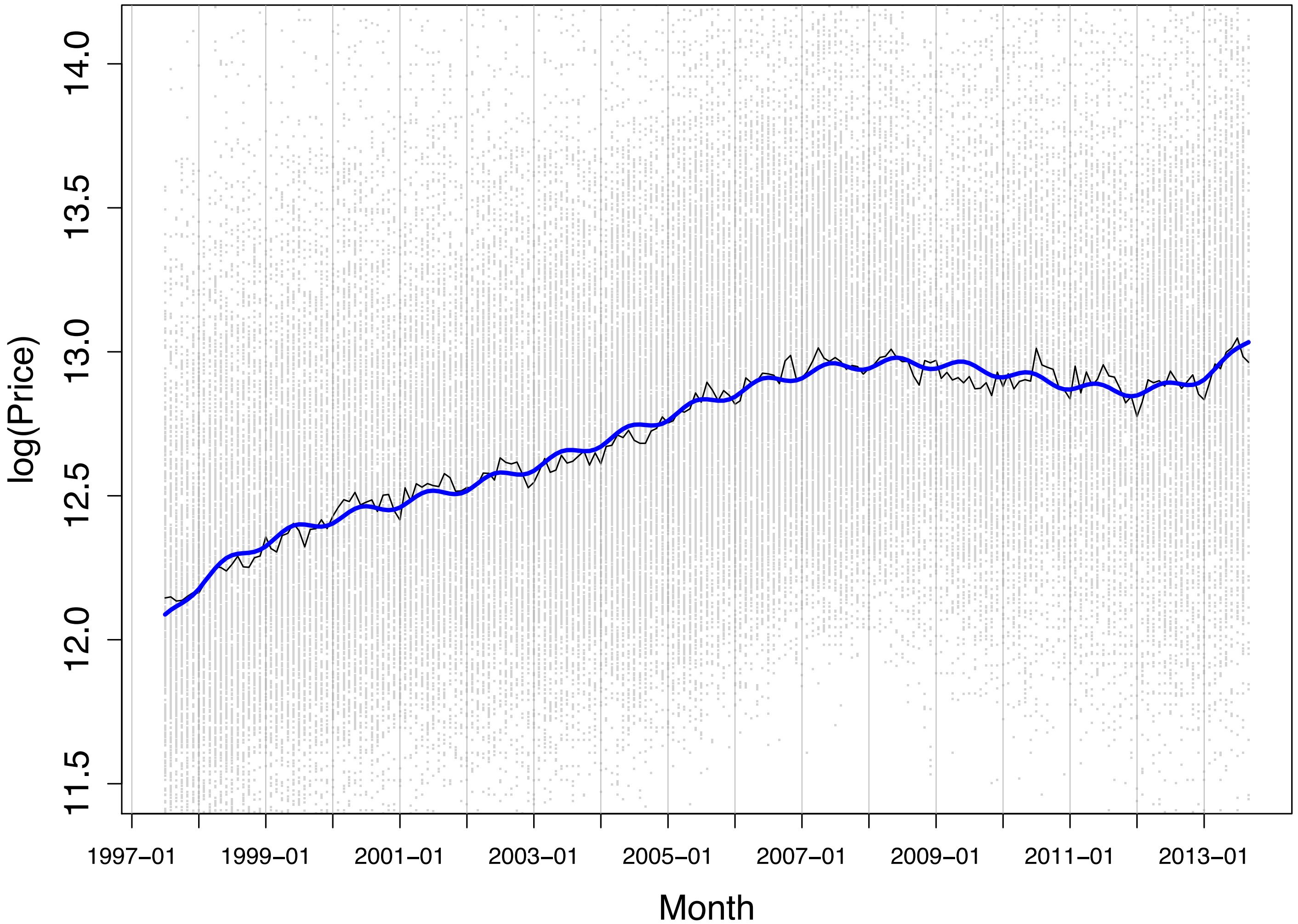


Image classification

Input: **x**
Image pixels

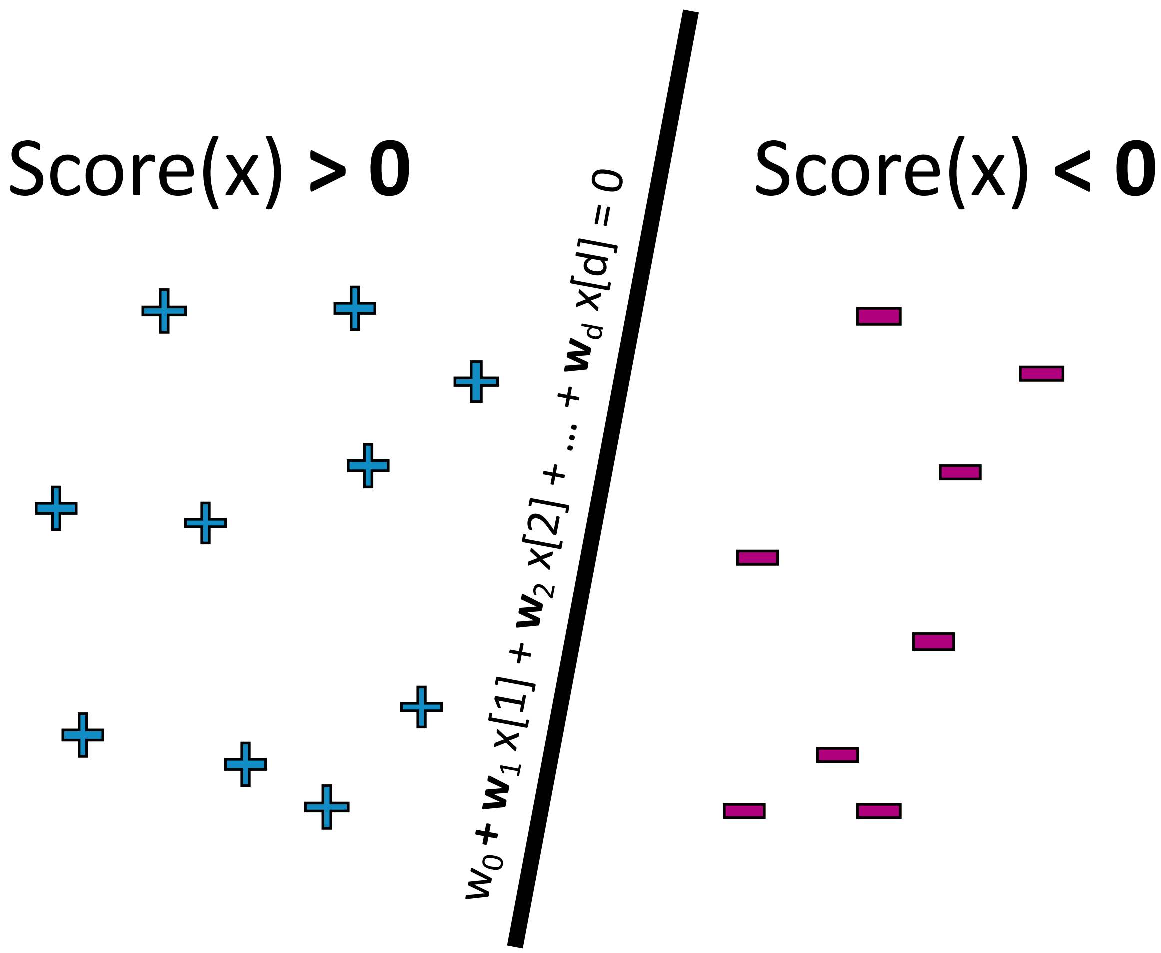
Output: y
Predicted object

- Feature engineering is extremely challenging
  - For real-data that is high-dimensional and complex
- Neural networks allow us to learn features that are non-linear

Recall: linear classification

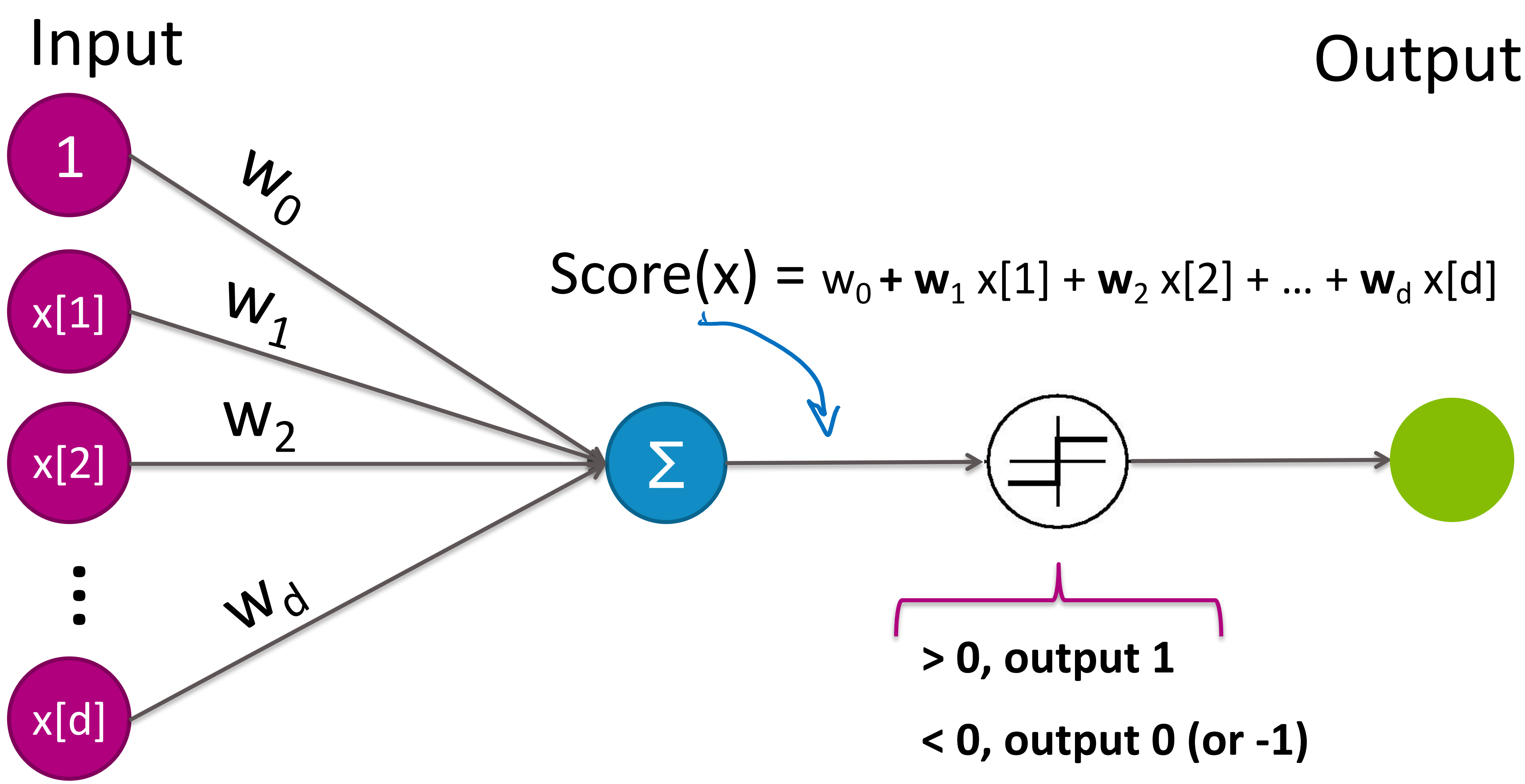
- Input is d-dimensional data
- Output is a partition of the space into two, separated by a hyperplane (line in 2-d)
- Training searches for the best line

Score(x) = $w_0 + \mathbf{w}_1\, x[1] + \mathbf{w}_2\, x[2] + \ldots + \mathbf{w}_d\, x[d]$



Graph representation of classifier: useful for defining neural networks

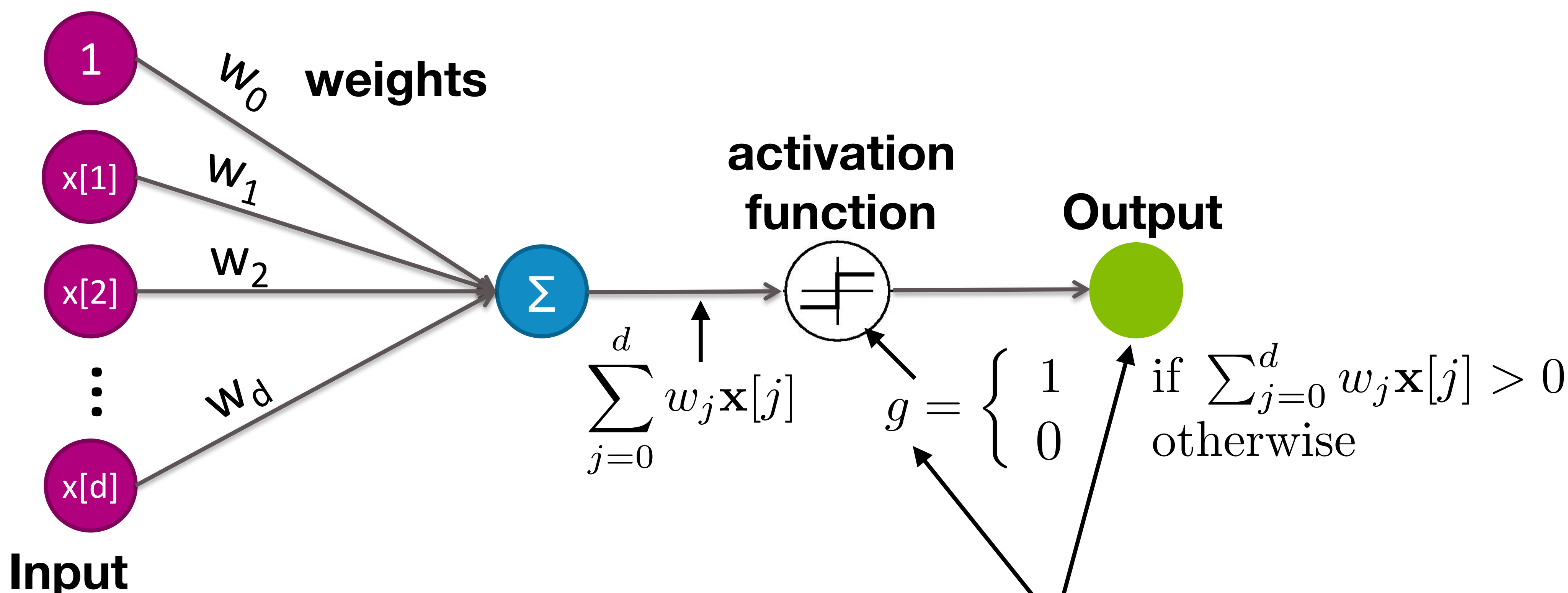
- We study an alternative representation of a linear classifier
- This graphical representation paves the way for designing deep neural networks
- This allow one to compactly represent a function (as a composition of many simple operations)



Single-layer neural network

This is a **single-layer** and **one-neuron** neural network

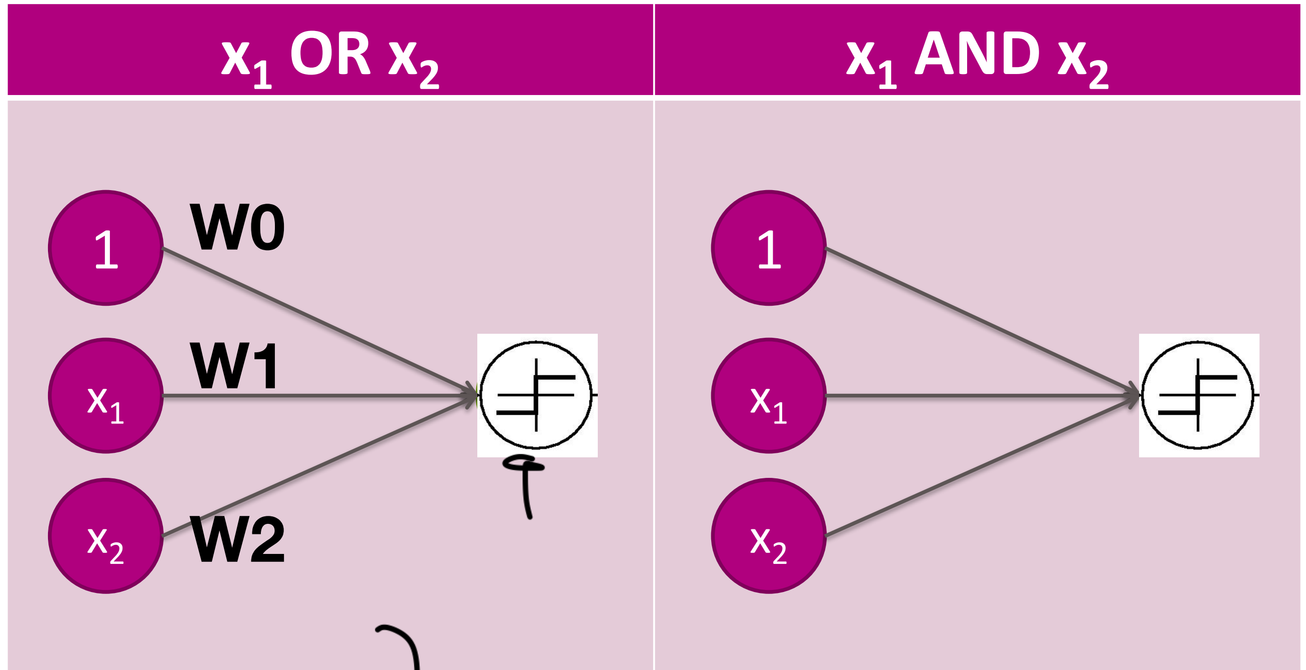
$$f(x) \;=\; \mathrm{sign}(w_0 + w_1 x[1] + \cdots w_d x[d])$$



- This is a generic formula for one **neuron**:
  - input: 1,x[1],..,x[d]
  - We take weighted sum
  - And pass it through an **activation function *g()***

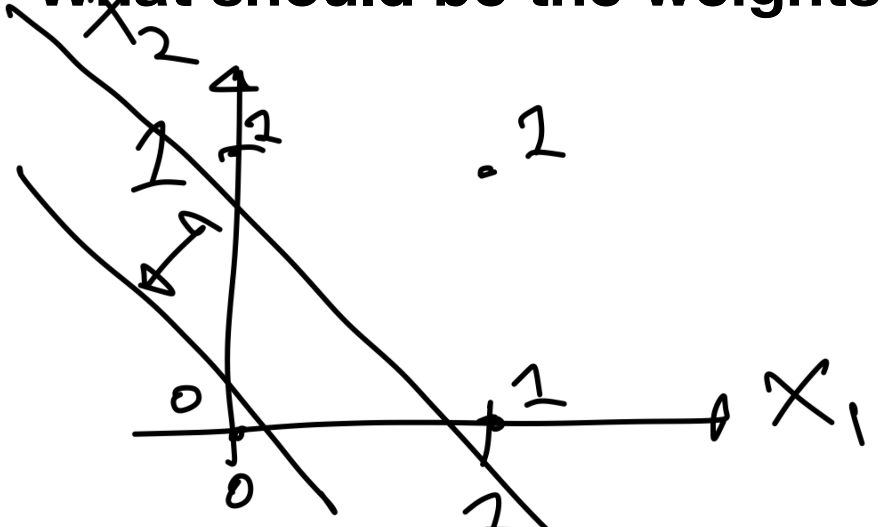
What can be represented by a linear classifier?

- x1 x2 y
- 0 0 0
- 0 1 1
- 1 0 1
- 1 1 1



- x1 x2 y
- 0 0 0
- 0 1 0
- 1 0 0
- 1 1 1

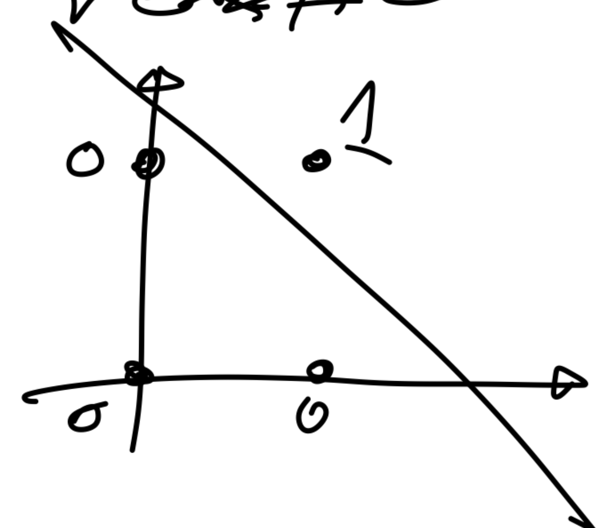
What should be the weights?



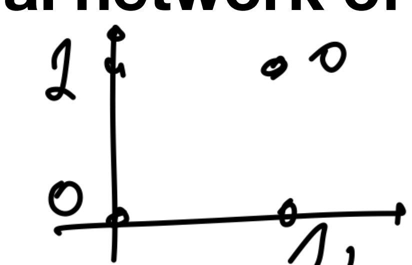
$$\boxed{\mathrm{sign}(w_0 + w_1 x_1 + w_2 x_2) = Y}$$

$-0.5 + 0.5x_1 + 0.5x_2 \rightarrow$ OR

$+0.5 + 0.5x_1 + 0.5x_2 \rightarrow$ ~~OR~~ AND

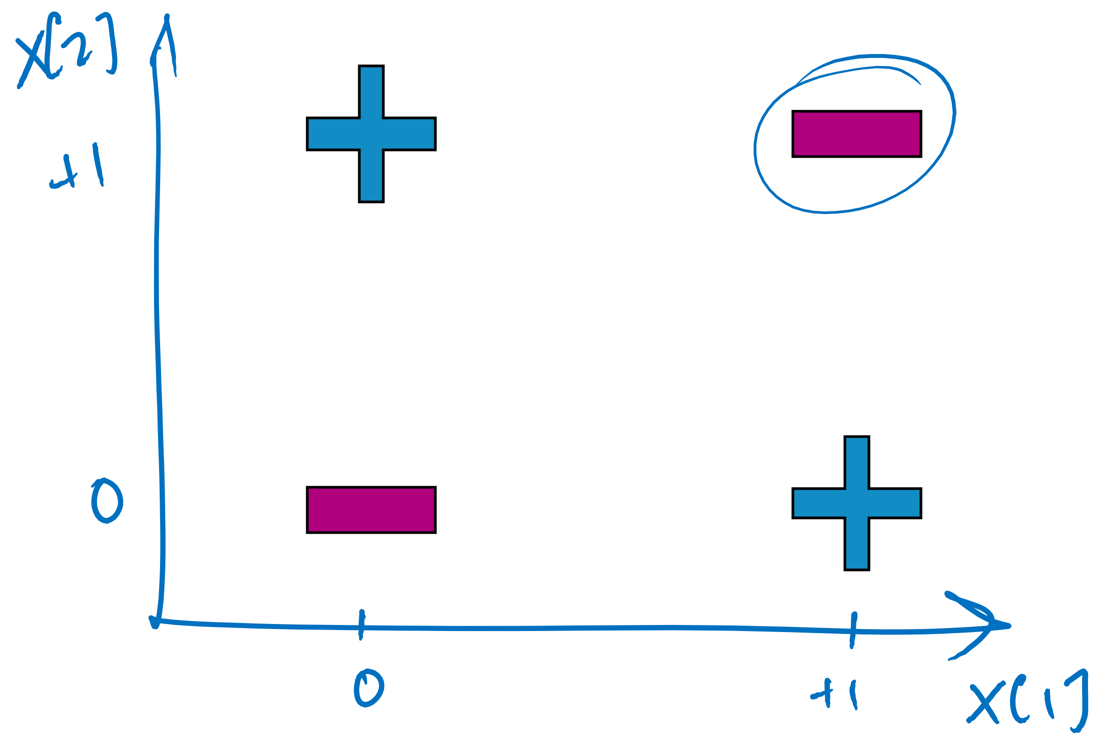


Note that there is a one-to-one correspondence between a linear classifier and a neural network of the above form



What cannot be learned?

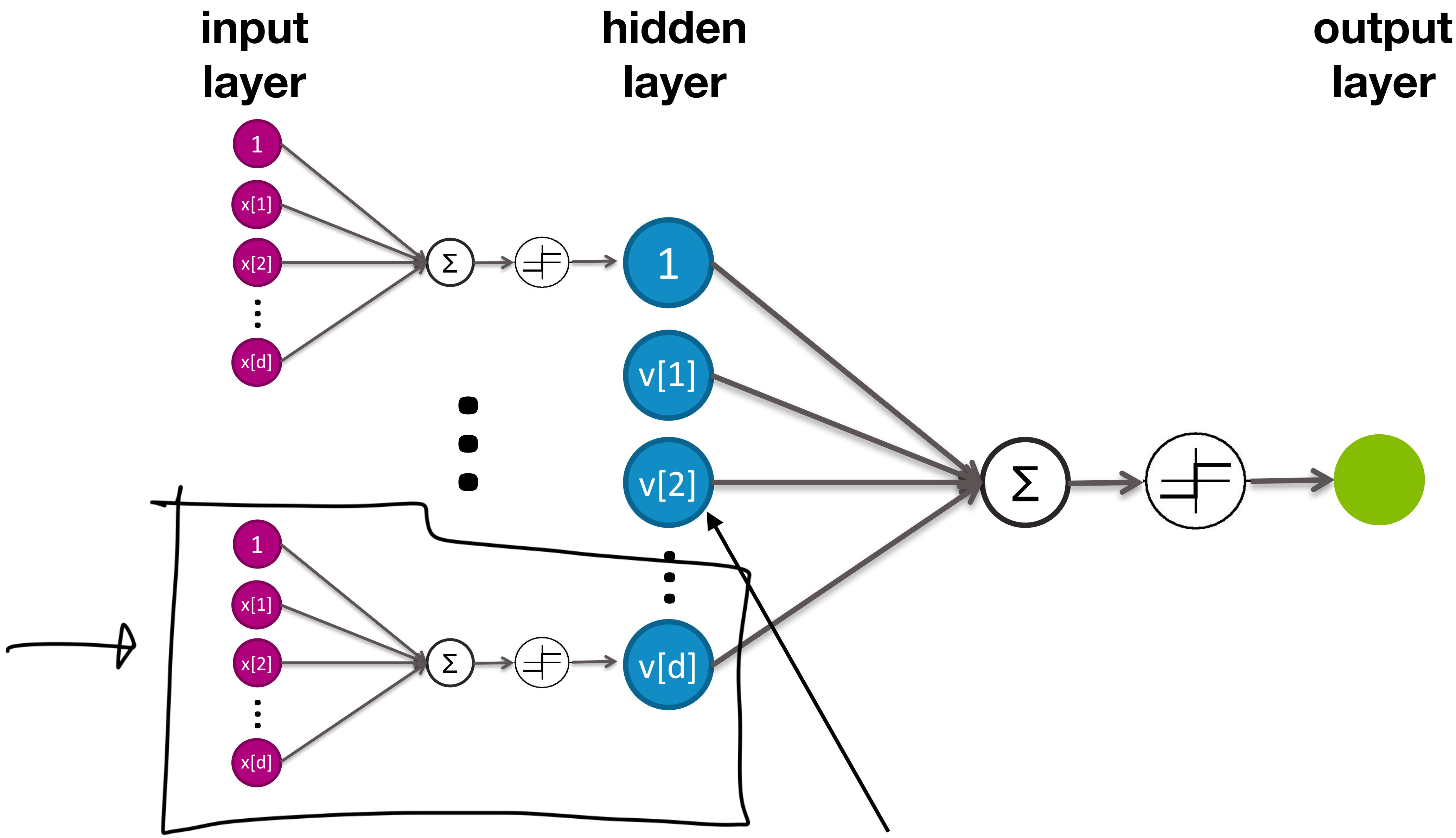
How can we get higher representation power?



- How can we build upon the single-layer, one-neuron function, to get a class of functions that can represent more complex functions?

Hidden layer

- We compose neurons to create a network of neurons
  -> neural network

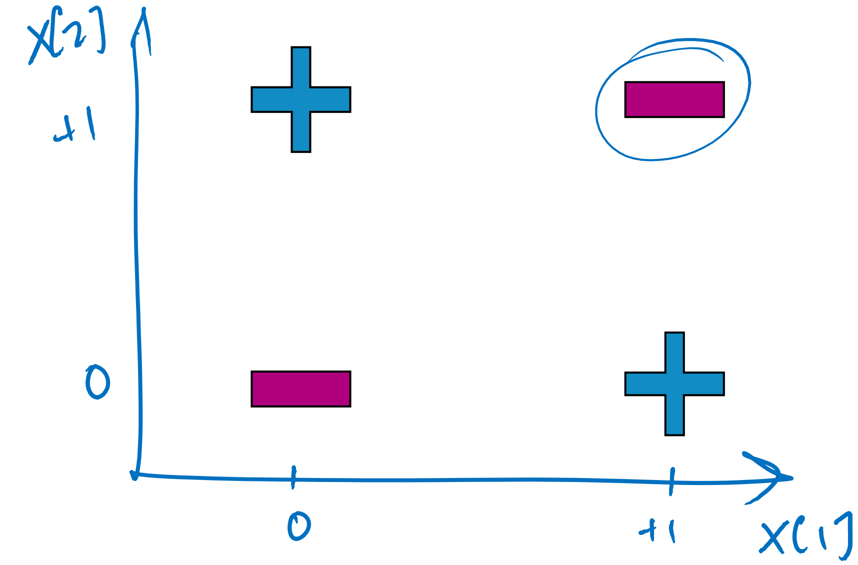


$$h_2(x) = \text{sign}(\underbrace{w_{20} + w_{21}x[1] + \cdots w_{2d}x[d]}_{w_2^T x})$$

$$f(x) = \text{sign}\Big((w^{(2)})^T \underbrace{\text{sign}((W^{(1)})^T x)}_{=h(x)} \Big)$$

Example: XOR function

$$\text{XOR} = \underbrace{x[1] \text{ AND NOT } x[2]}_{v[1]} \text{ OR } \underbrace{\text{NOT } x[1] \text{ AND } x[2]}_{v[2]}$$



XOR as a 2-layer neural network

y = x[1] XOR x[2]   =(x[1] AND ¬x[2]) OR (x[2] AND ¬x[1])

v[1] = (x[1] AND ¬x[2])

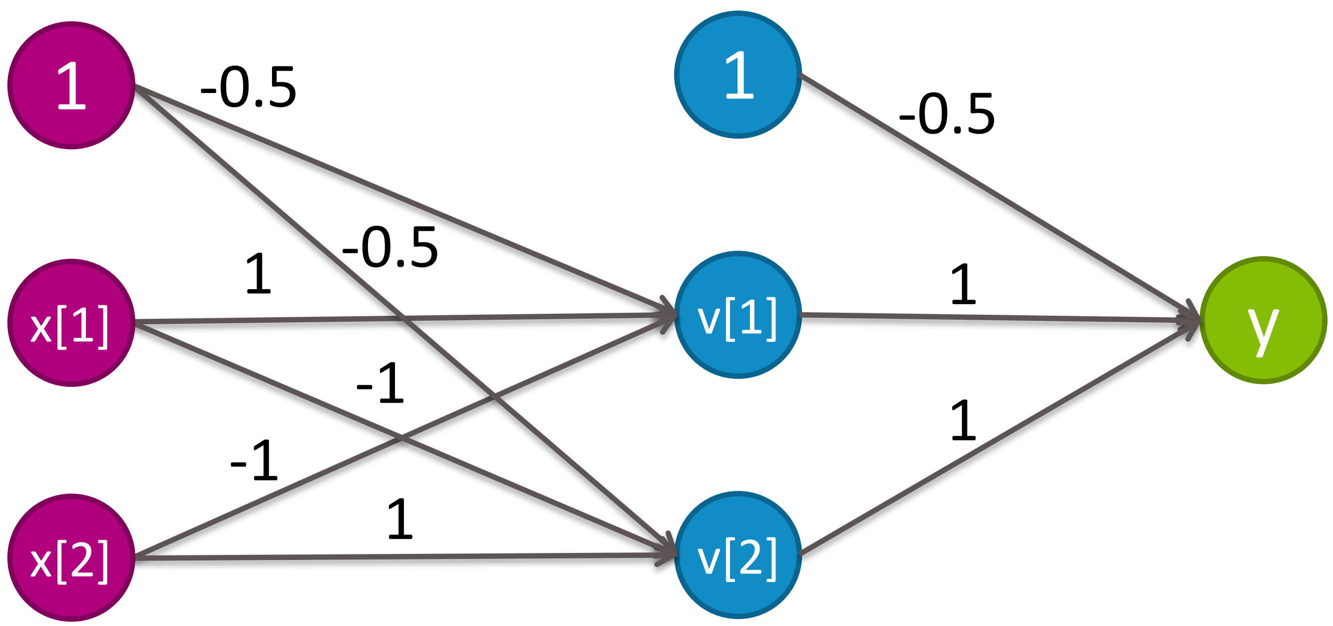
  = g(-0.5+x[1]-x[2])

v[2] = (x[2] AND ¬x[1])

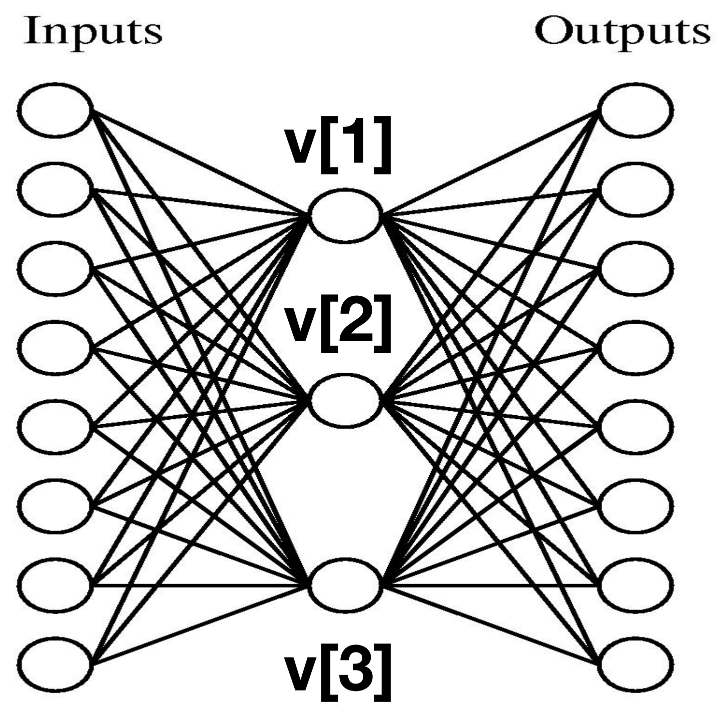
  = g(-0.5+x[2]-x[1])

y = v[1] OR v[2]

  = g(-0.5+v[1]+v[2])



Two-layer neural network (= one-hidden layer neural network)



Single unit:

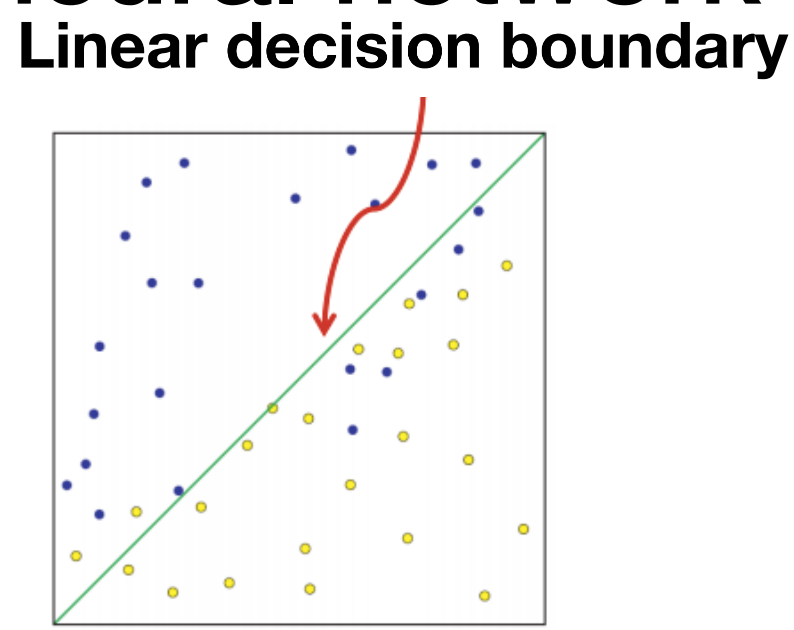
$$out(\mathbf{x}) = g(w_0 + \sum_j w_j \mathbf{x}[j])$$

1-hidden layer:

$$out(\mathbf{x}) = g(w_0 + \sum_k w_k \underbrace{\boxed{g(w_0^k + \sum_j w_j^k \mathbf{x}[j])}}_{\mathbf{v}[k]})$$

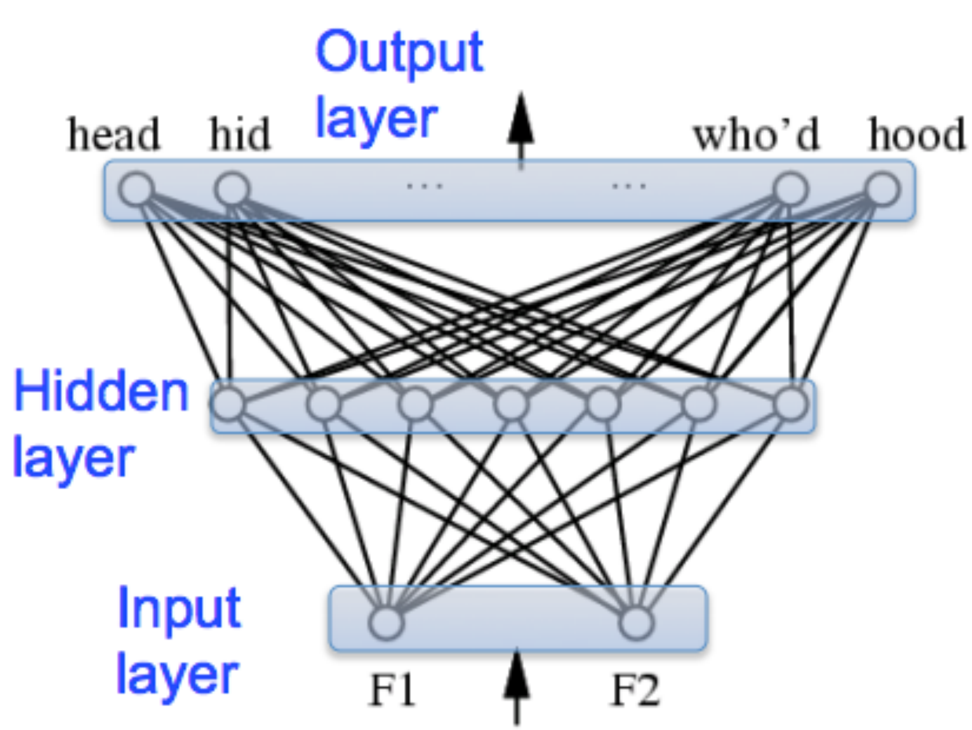
Example of 2-layer neural network in action

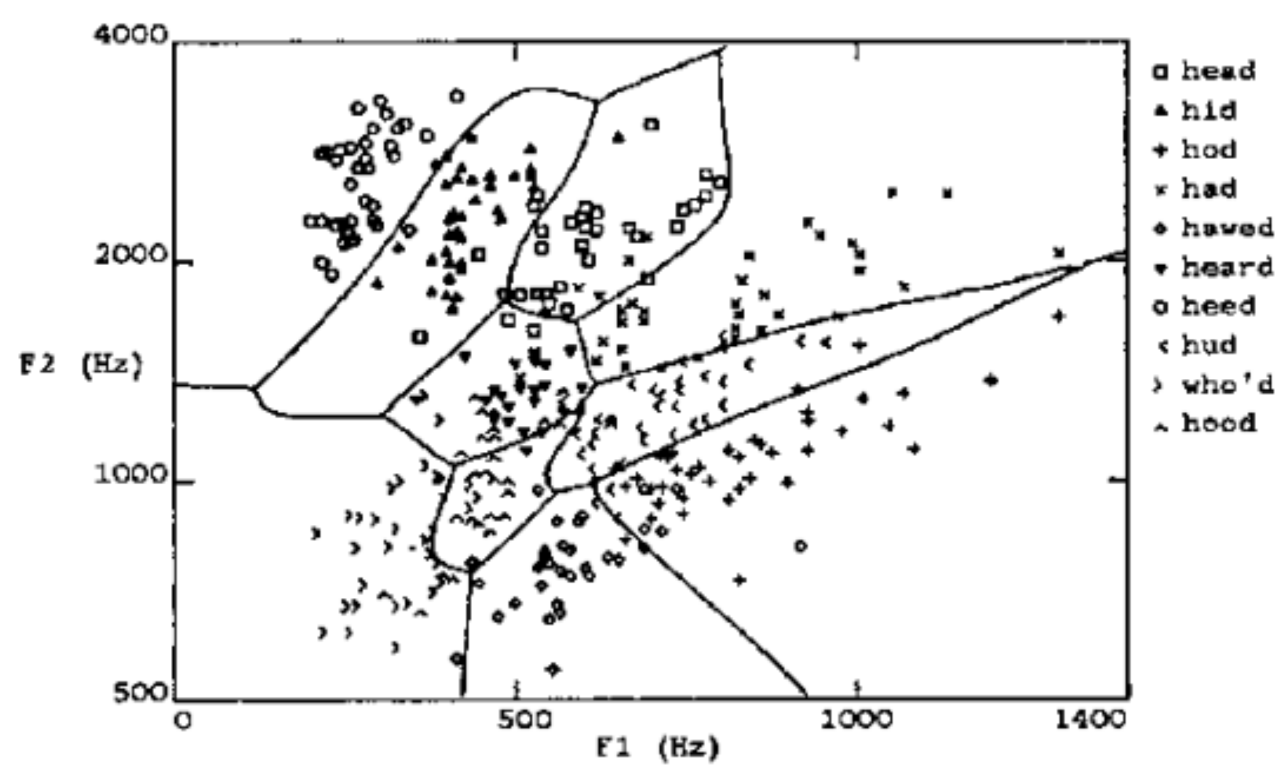
1-layer neural networks
only represents linear classifiers



Example: 2-layer neural network trained to distinguish vowel sounds using 2 formants (features)

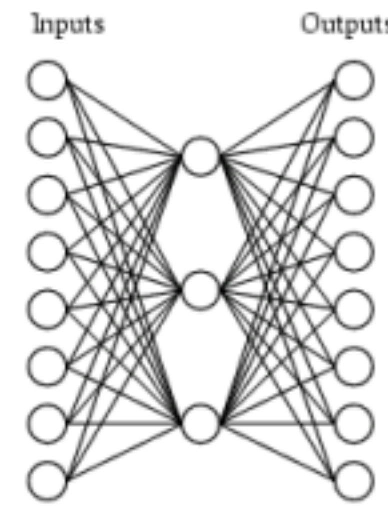
a highly non-linear decision boundary can be learned from 2-layer neural networks





Representation power of a 2-layer neural network

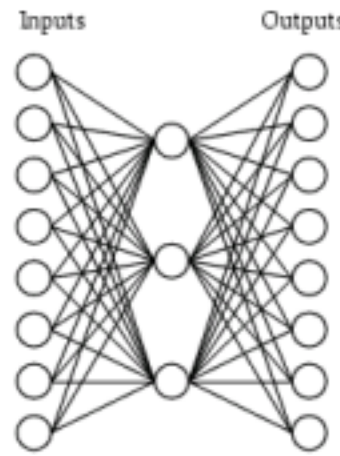
- Can such function be learned?
- If we are manually designing functions, then 3 hidden layer is enough.
- The reason is that there is some simplicity or pattern in the data that we want to represent: it only has basis vectors!



A target function:

	Input		Output
000	10000000	$\rightarrow$	10000000
001	01000000	$\rightarrow$	01000000
010	00100000	$\rightarrow$	00100000
100	00010000	$\rightarrow$	00010000
011	00001000	$\rightarrow$	00001000
	00000100	$\rightarrow$	00000100
	00000010	$\rightarrow$	00000010
111	00000001	$\rightarrow$	00000001

A network:

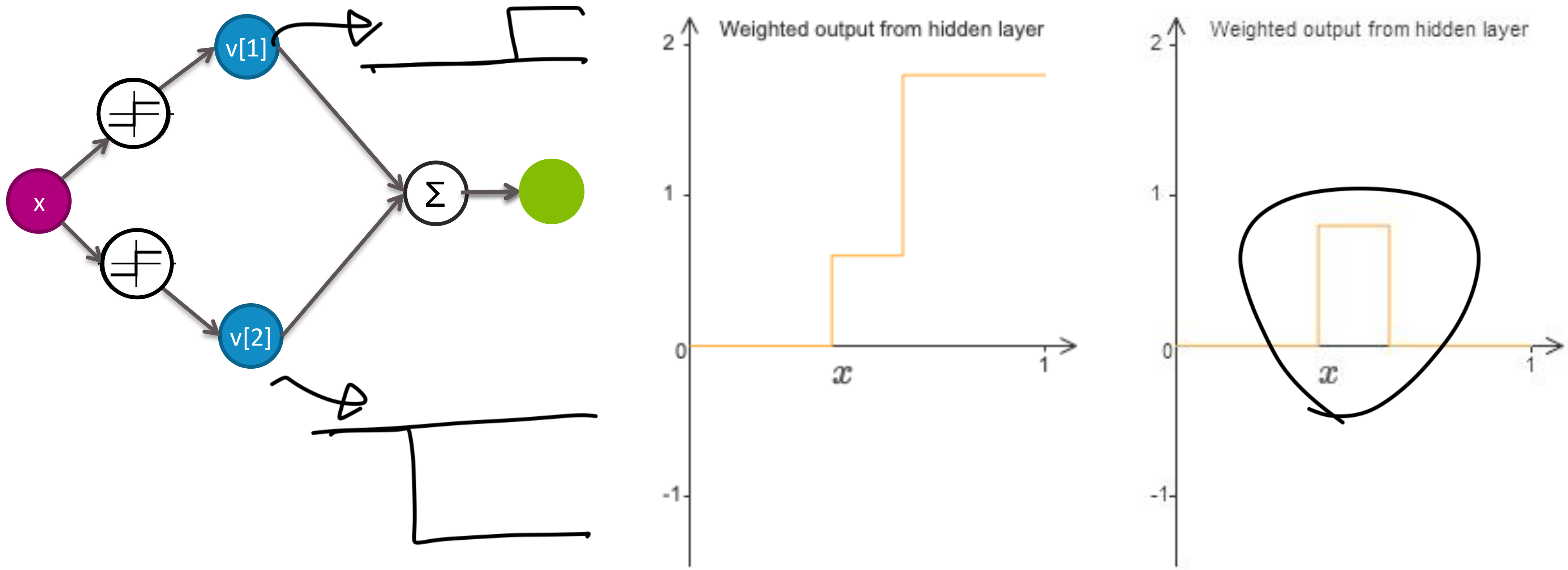


Learned hidden layer representation:

Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

A 2-layer neural network can represent any function, if we allow enough units in the hidden layer

One-dimensional input/output example for illustration

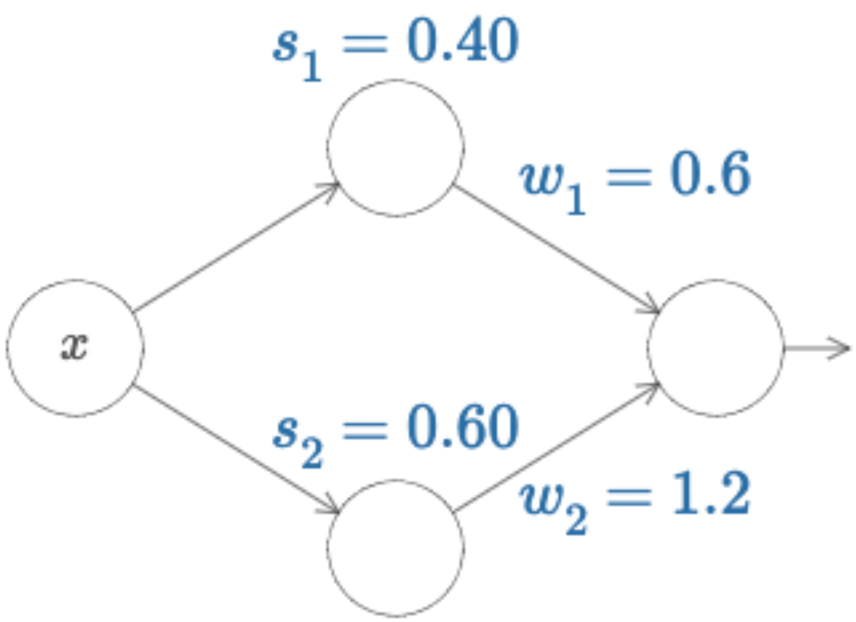


- We can compose step functions
  to approximate piece constant functions
  and use them to approximate any function

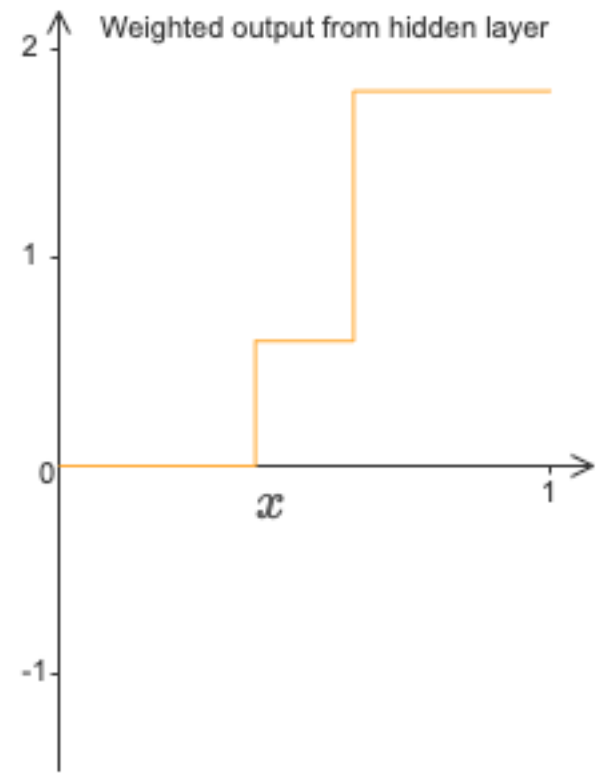
- More pieces (more hidden units) give better approximation

- demo: http://neuralnetworksanddeeplearning.com/chap4.html

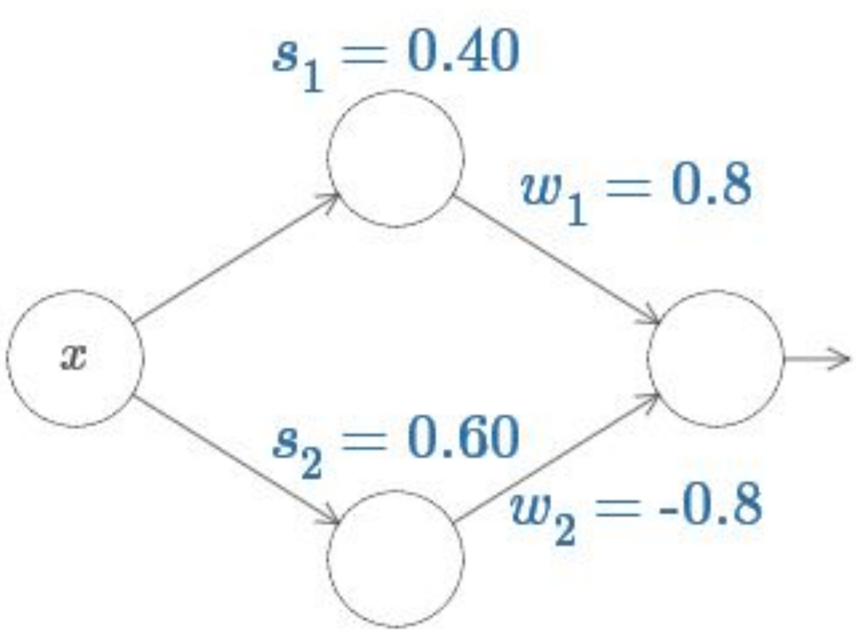
Example



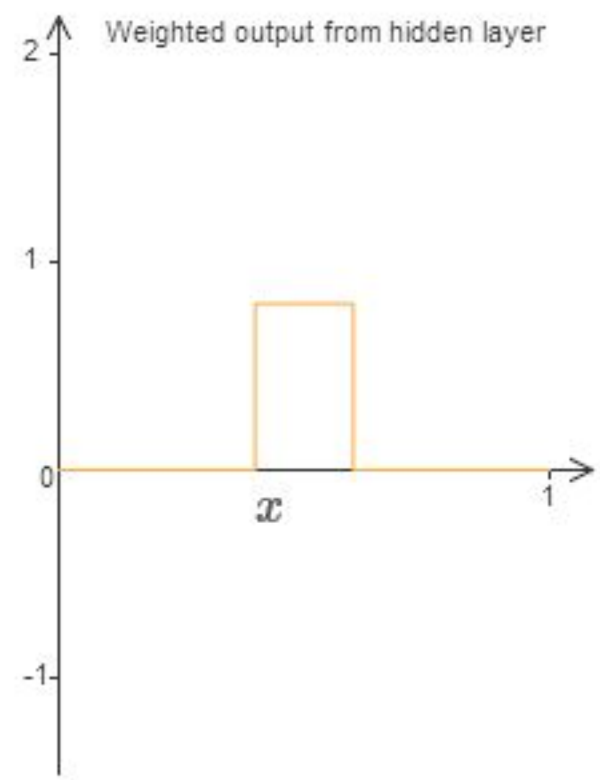
$s_1 = 0.40$
$w_1 = 0.6$
x
$s_2 = 0.60$
$w_2 = 1.2$



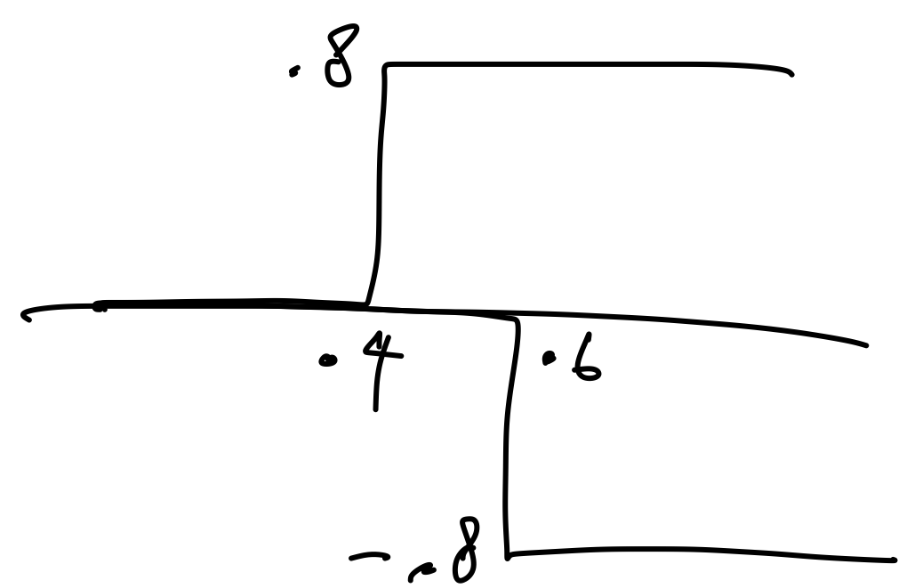
Weighted output from hidden layer
2
1
0
-1
x
1



$s_1 = 0.40$
$w_1 = 0.8$
x
$s_2 = 0.60$
$w_2 =$ -0.8

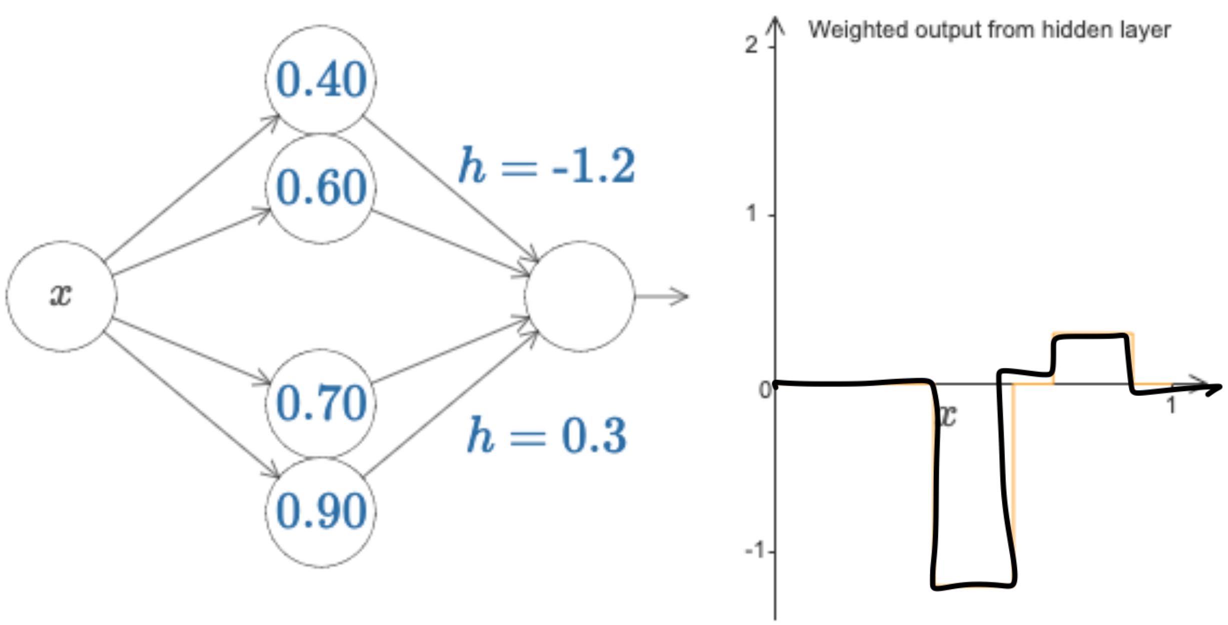


Weighted output from hidden layer
2
1
0
-1
x
1

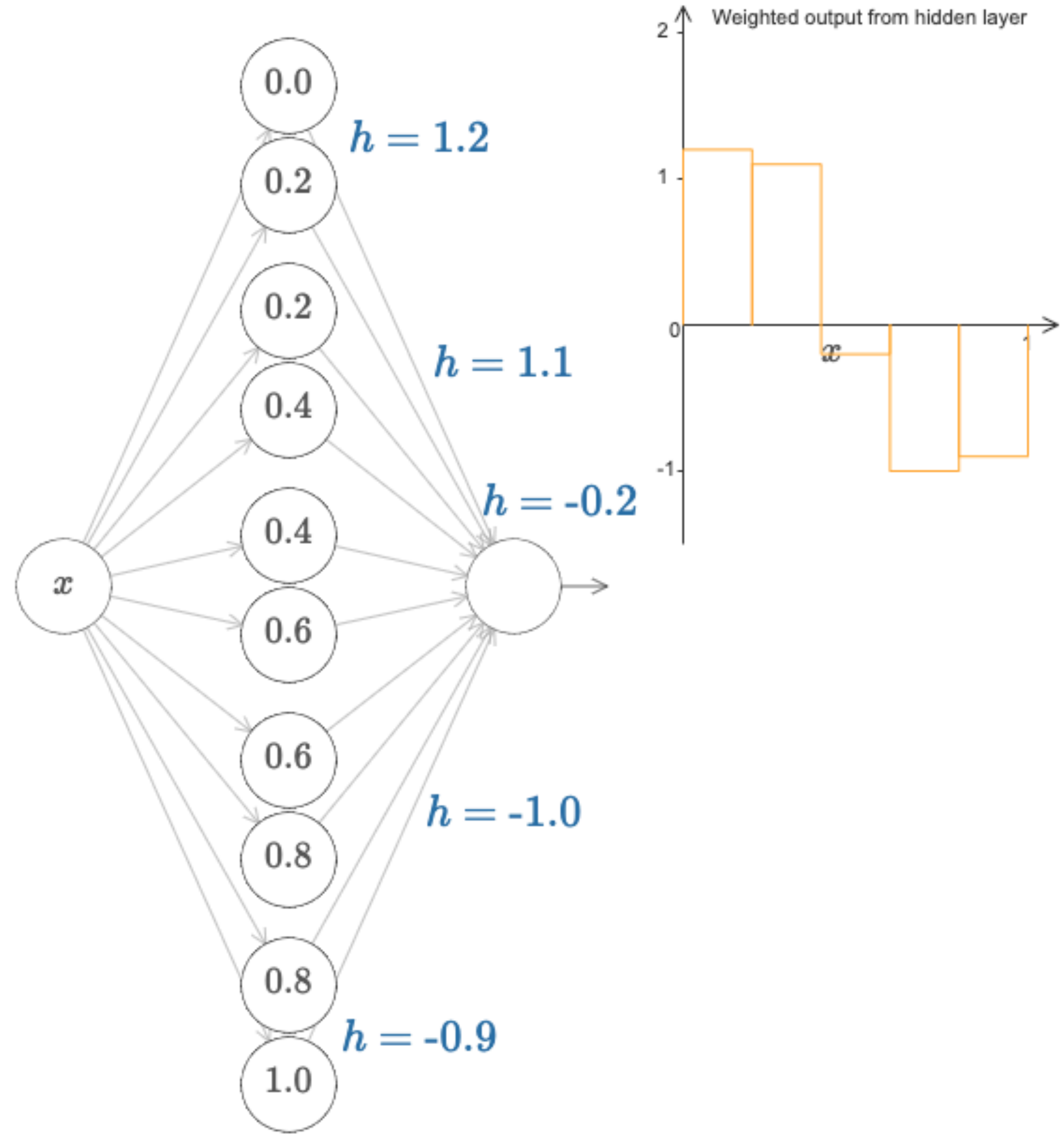


.8
.4
.6
-.8

Example

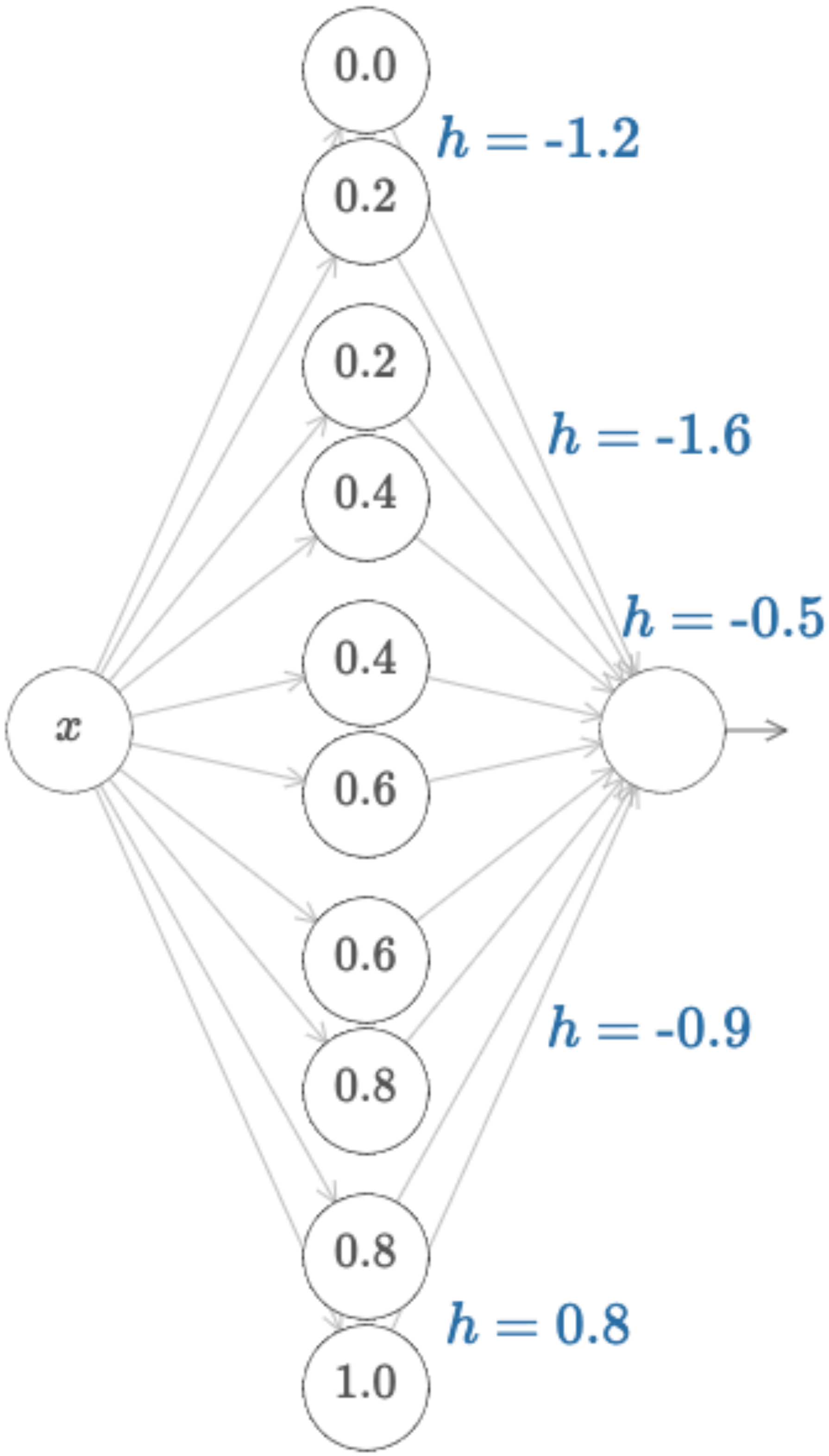


x
0.40
0.60
0.70
0.90
$h =$ -1.2
$h =$ 0.3
Weighted output from hidden layer
2
1
0
-1
x
1

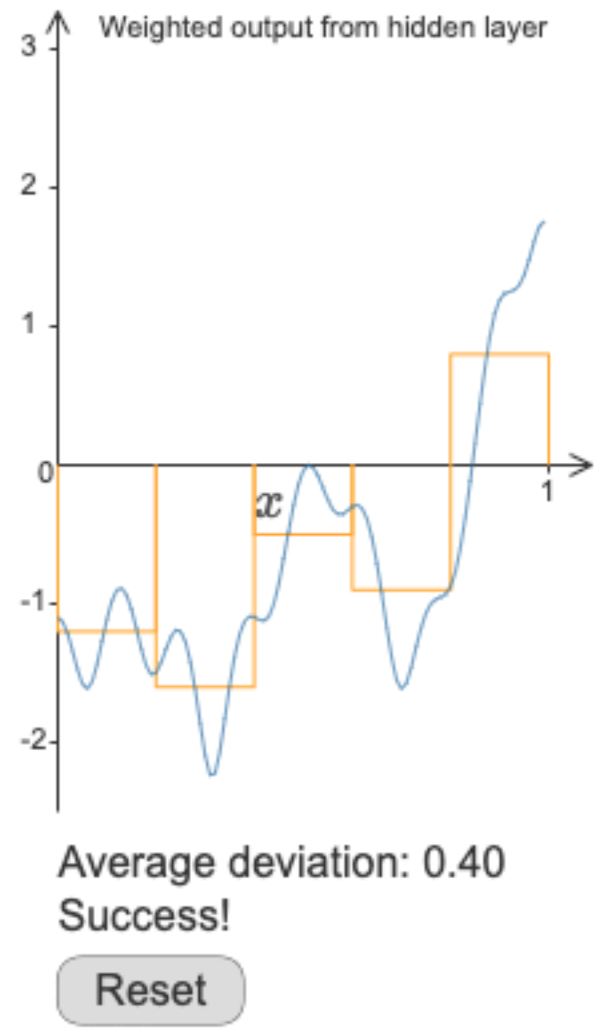


x
0.0
0.2
0.2
0.4
0.4
0.6
0.6
0.8
0.8
1.0
$h =$ 1.2
$h =$ 1.1
$h =$ -0.2
$h =$ -1.0
$h =$ -0.9
Weighted output from hidden layer
2
1
0
-1
x

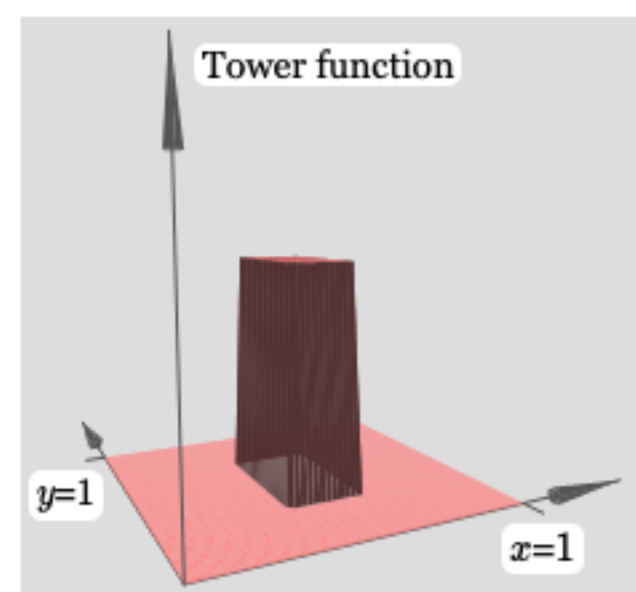
Example



0.0
$h =$ -1.2
0.2
0.2
$h =$ -1.6
0.4
$h =$ -0.5
0.4
x
0.6
0.6
$h =$ -0.9
0.8
0.8
$h = 0.8$
1.0



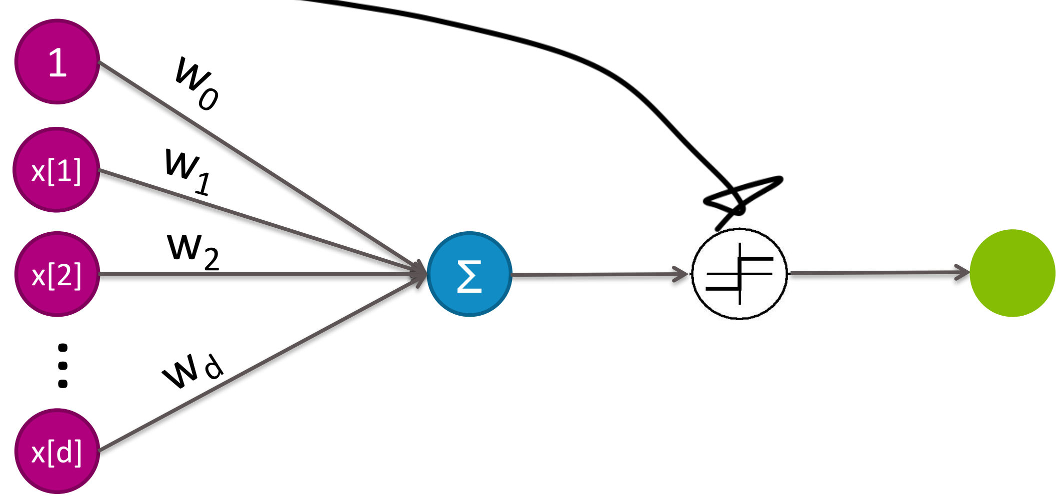
Weighted output from hidden layer
3
2
1
0
-1
-2
x
1
Average deviation: 0.40
Success!
Reset



Tower function
y=1
x=1

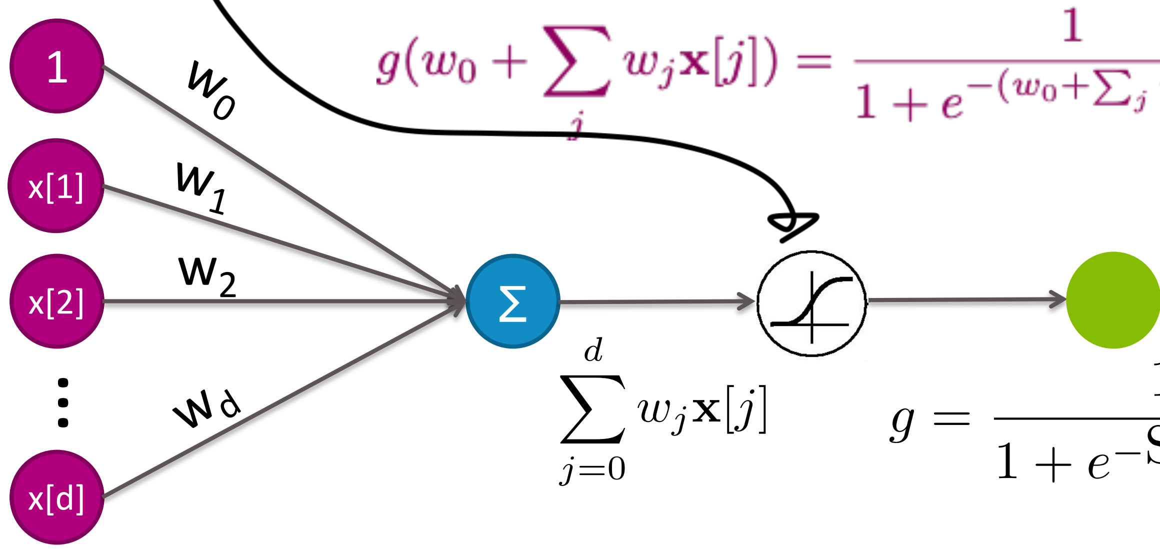
General neural networks

- **Sign** activation function is never used in practice because the gradient is zero almost everywhere

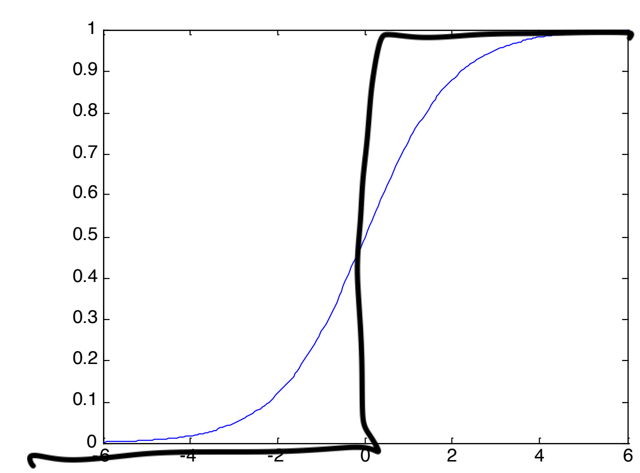


- instead, **sigmoids** can be used because it is differentiable, and can approximate the sign function

$$g(w_0 + \sum_j w_j \mathbf{x}[j]) = \frac{1}{1 + e^{-(w_0 + \sum_j w_j \mathbf{x}[j])}}$$

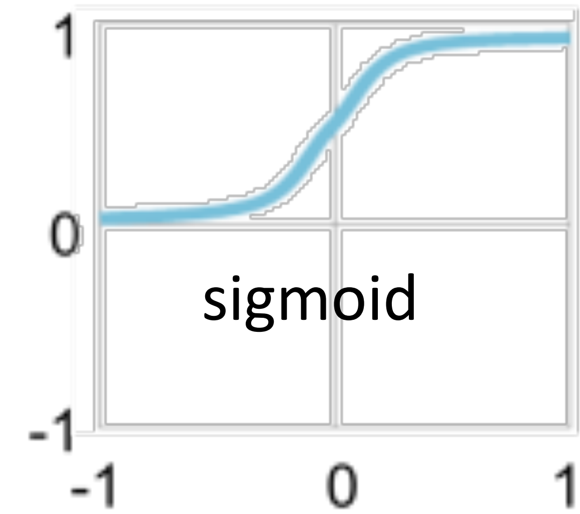


$$\sum_{j=0}^{d} w_j \mathbf{x}[j] \qquad g = \frac{1}{1 + e^{-\mathrm{Score}(\mathbf{x})}}$$

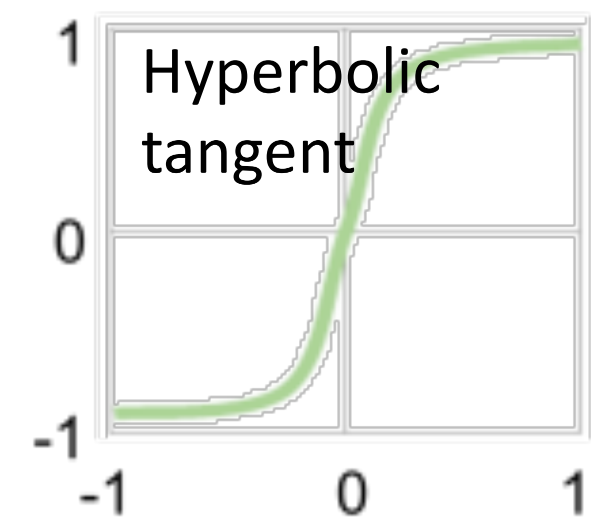


Activation functions

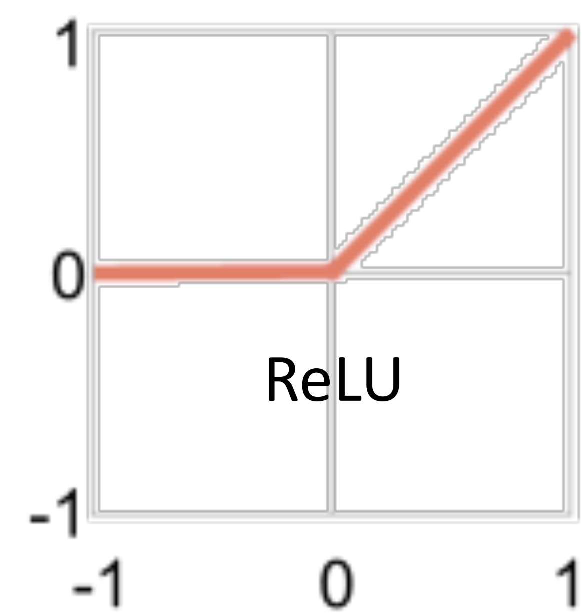
•Sigmoid
-Historically popular, but (mostly) fallen out of favor
•Neuron's activation saturates
(weights get very large -> gradients get small)
•Not zero-centered -> other issues in the gradient steps
-When put on the output layer, called "softmax" because interpreted as class probability (soft assignment)



•Hyperbolic tangent $g(x) = \tanh(x)$
-Saturates like sigmoid unit, but zero-centered



•Rectified linear unit (ReLU) $g(x) = x^+ = \max(0,x)$
-Most popular choice these days
-Fragile during training and neurons can "die off"… be careful about learning rates
-"Noisy" or "leaky" variants



•Softplus $g(x) = \log(1+\exp(x))$
-Smooth approximation to rectifier activation

General neural networks

- Layers and layers and layers of linear models and non-linear transformations

- Around for about 50 years
  - Fell in “disfavor” in 90s

- In last few years, big resurgence
  - Impressive accuracy on several benchmark problems
  - Powered by huge datasets, GPUs, & modeling/learning algorithm improvements

Overfitting

Are NNs likely to overfit?

- *Yes*, they can represent arbitrary functions!!!

Avoiding overfitting?

- More training data
- Fewer hidden nodes / better topology
  - **Rule of thumb:** 3-layer NNs outperform 2-layer NNs, but going deeper rarely helps (different story for convolutional networks!)
- Regularization
- Early stopping

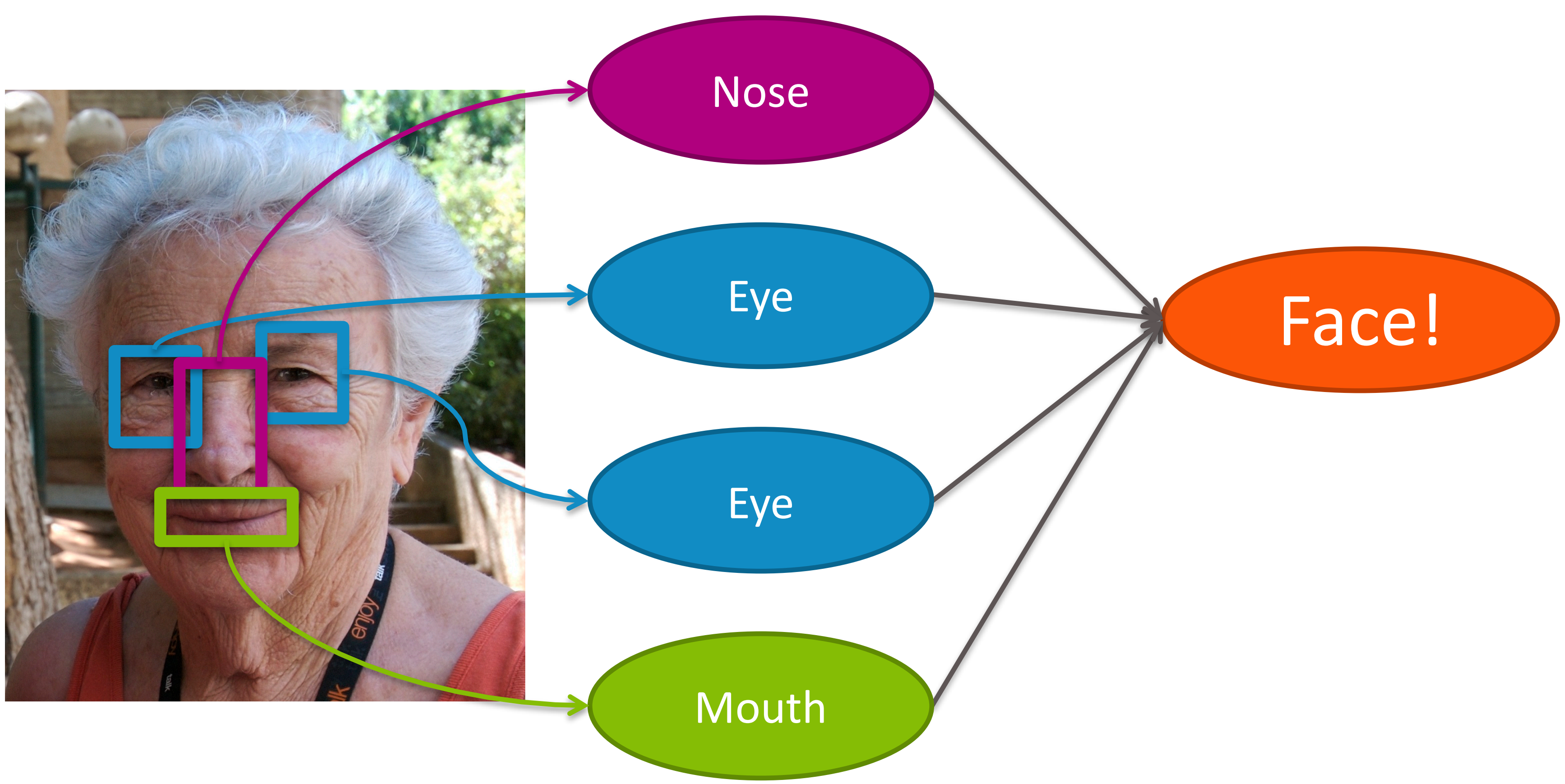
Applications to vision problems

- Classical image processing manually extracts features

Features = local detectors

−Combined to make prediction

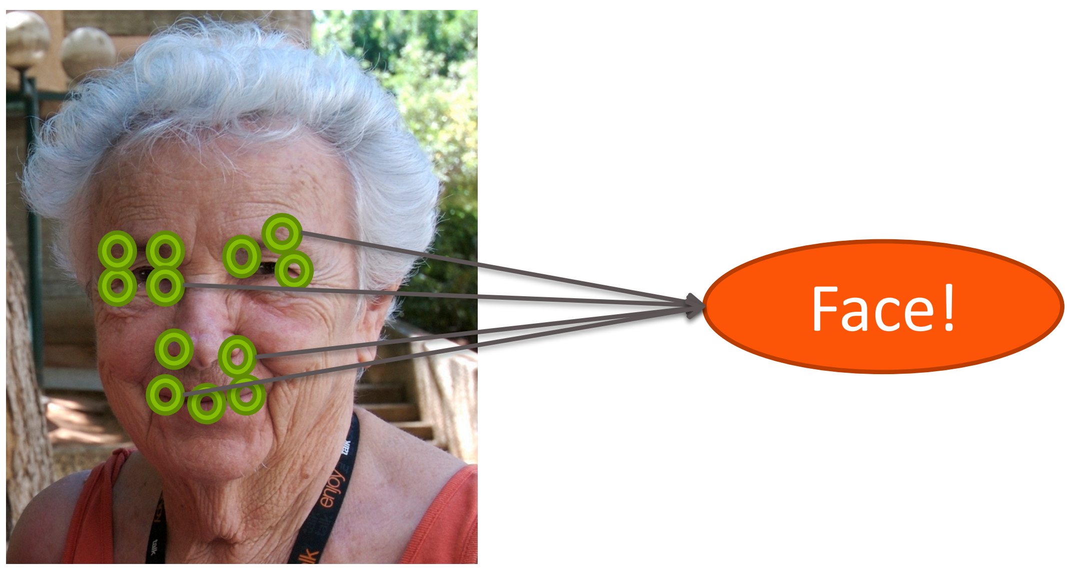
−(in reality, features are more low-level)

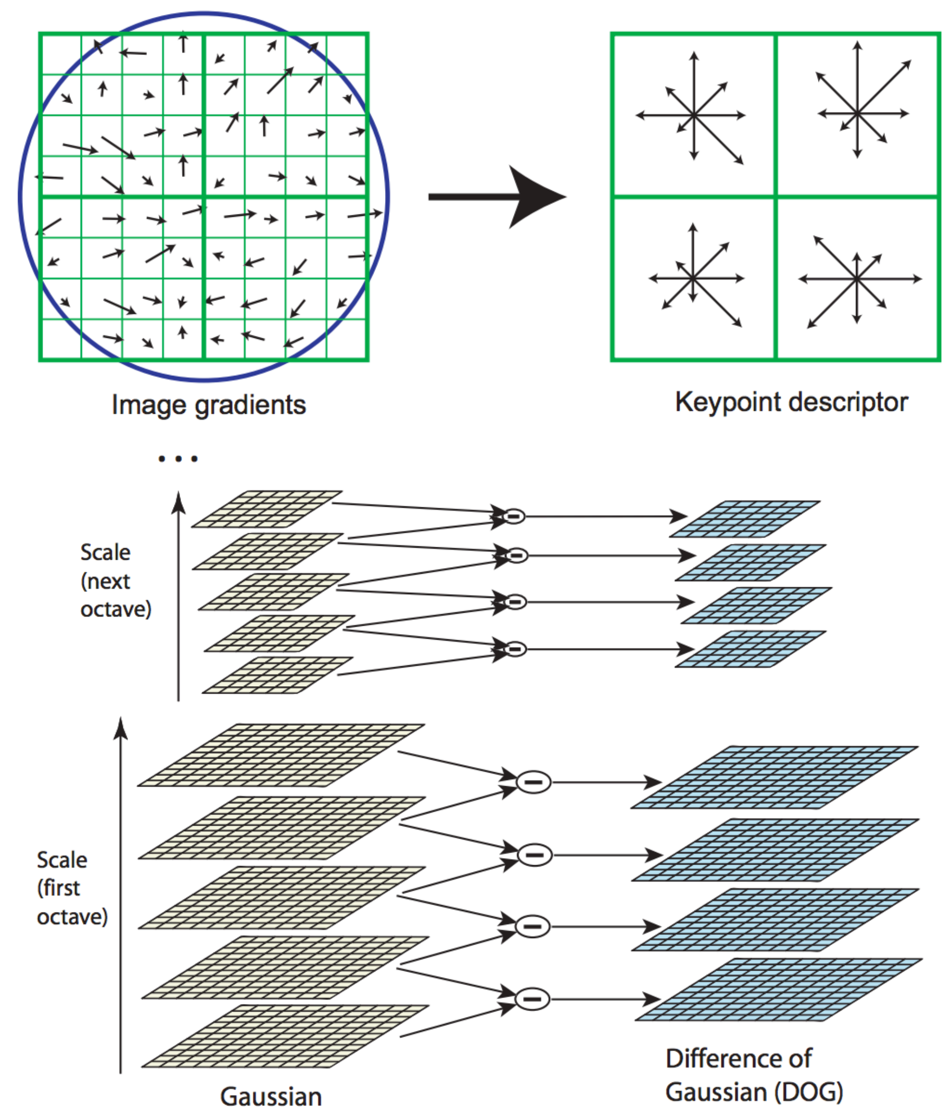


Typical local detectors look for locally “interesting points” in image

Image features: collections of locally interesting points

−Combined to build classifiers





Many hand created features exist
for finding interest points…

SIFT [Lowe ‘99]

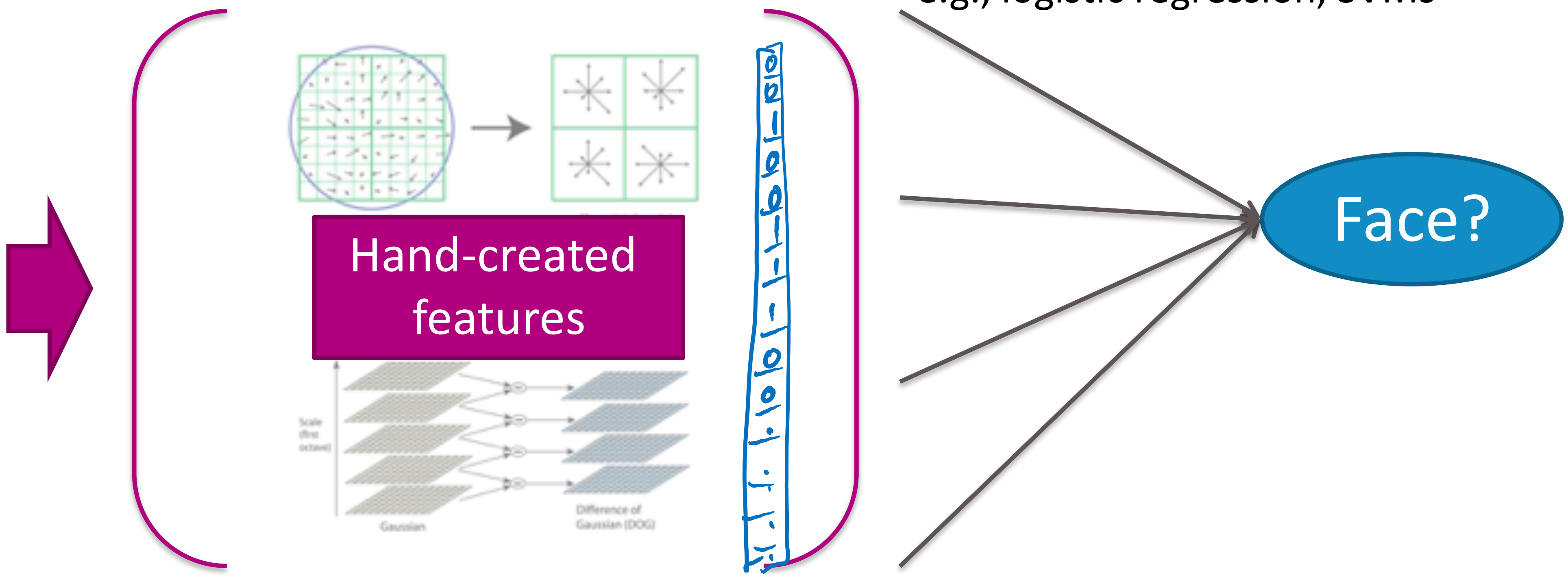
Classical image classification

Input

Extract features

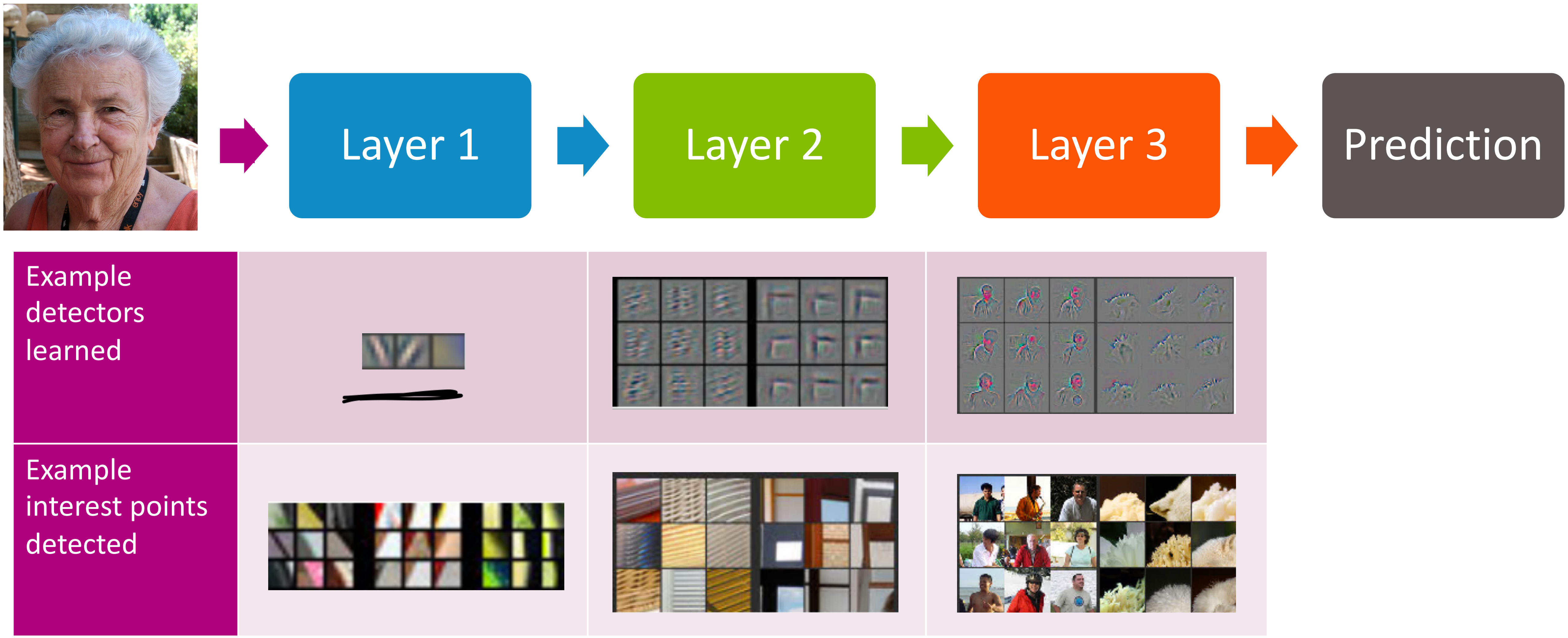
Use simple classifier

e.g., logistic regression, SVMs



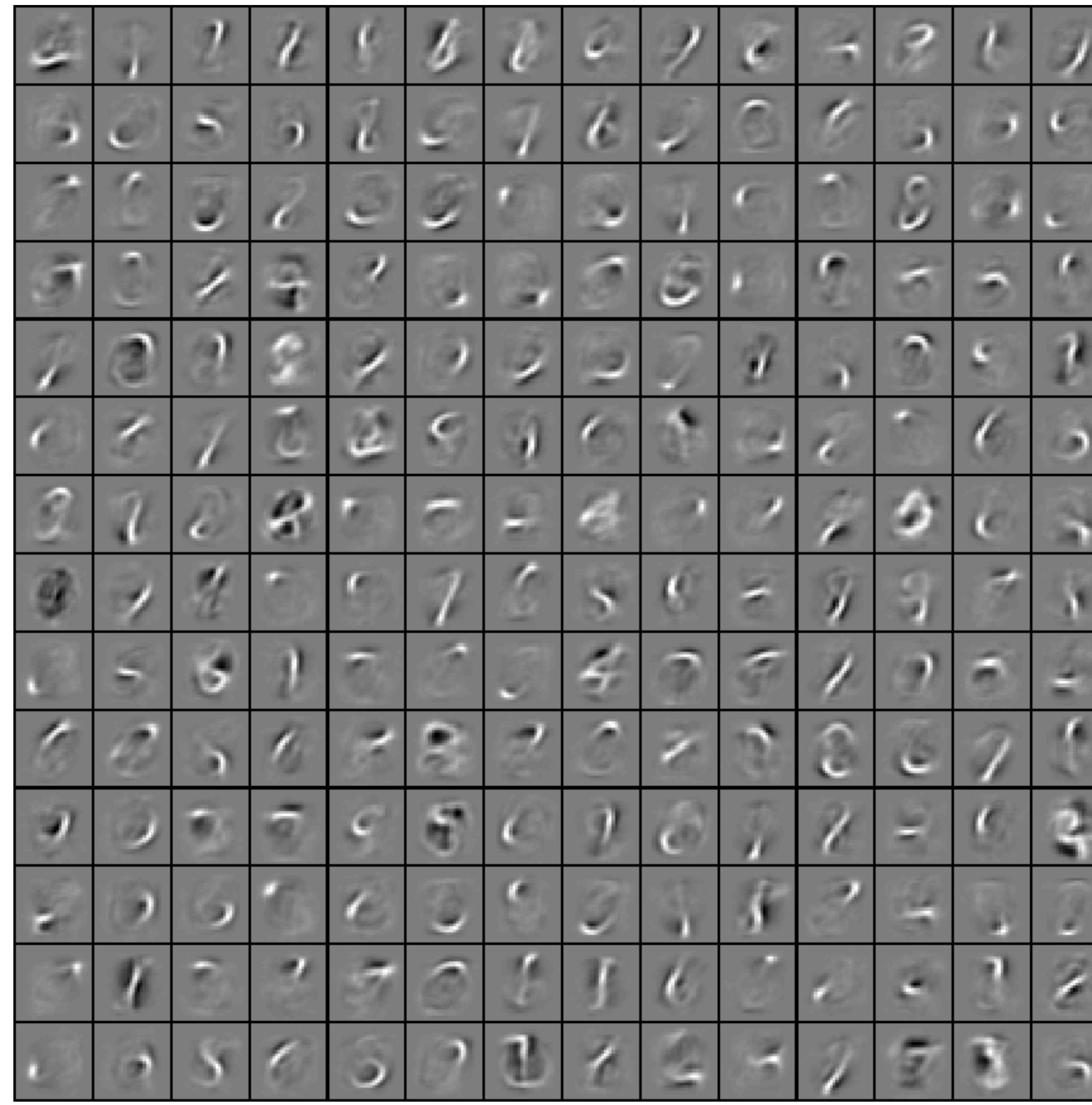
- Critically relies on having good features manually chosen

Instead, neural network (implicitly) discovers those features from data



[Zeiler & Fergus '13]

- Each layer learns increasingly complex features, as we go higher in the layers



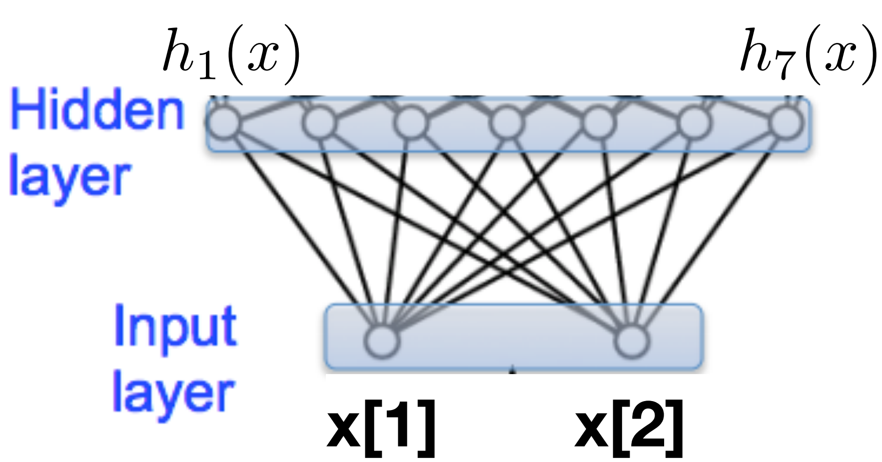
Convolutional neural networks

The challenge of applying regular neural networks (multilayer perceptrons) to images

- Interesting images are very high-dimensional
- And images have particular structures
  - Invariance to shift, scale, rotation

Convolutional Neural Networks (CNN)

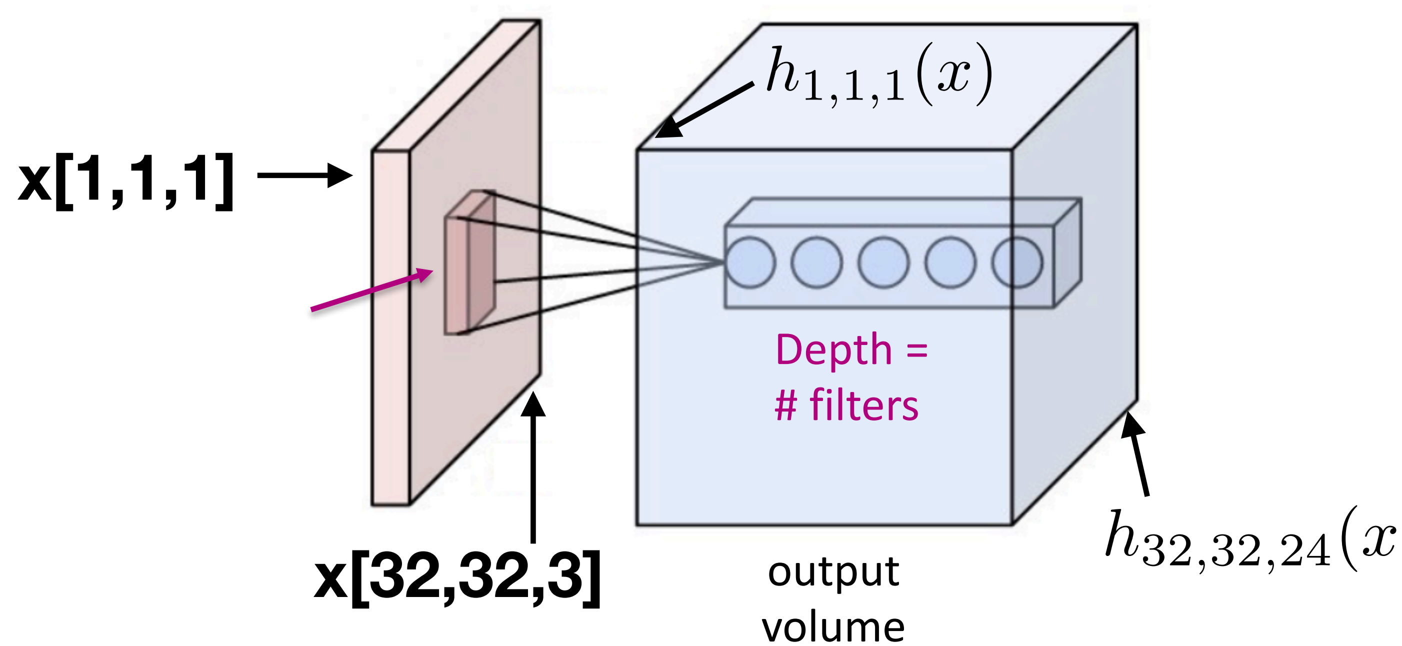
Main building block of NN: fully connected layer



$$h_i(x) = g(\, w_{i0} + w_{i1}x[1] + w_{i2}x[2] \,)$$

Activation function of choice

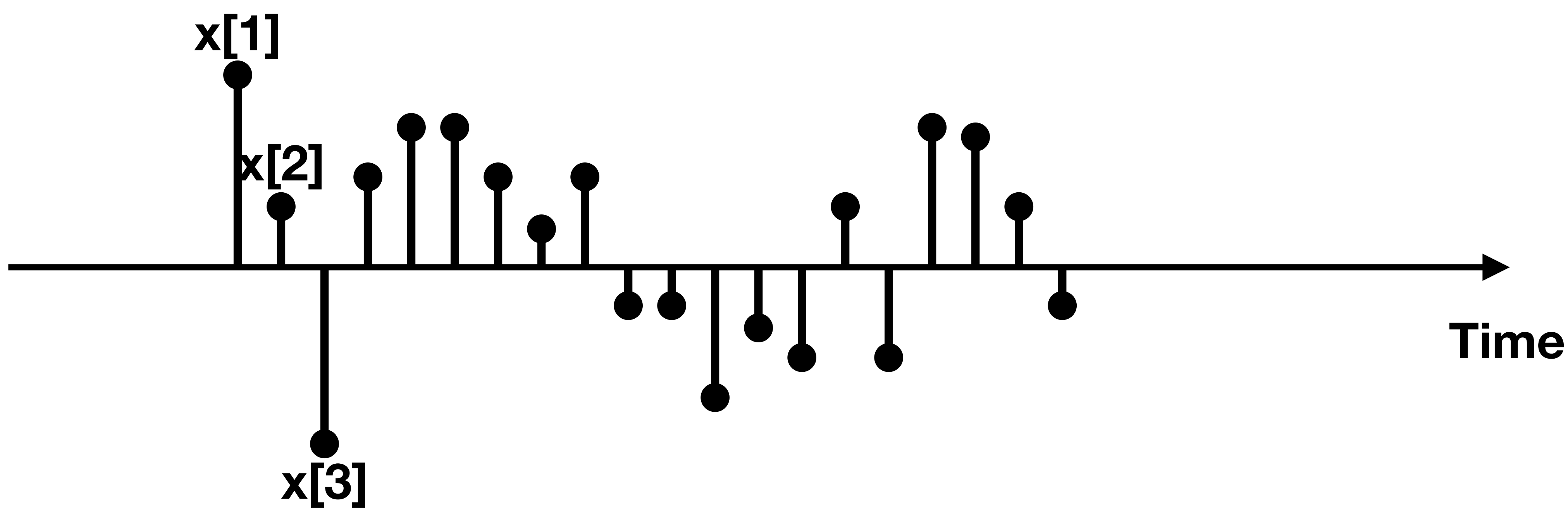
Main building block of CNN: convolutional layer



Both input and output are 3-dimensional arrays called tensors

Convolution

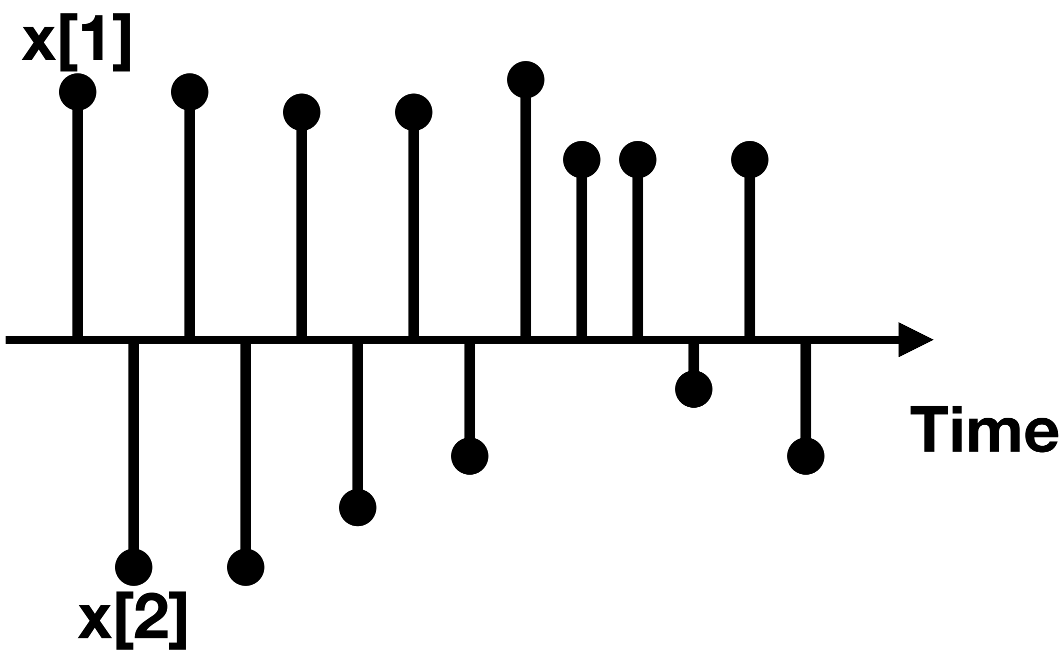
- Consider a one-dimensional signal (a sequence or a vector)
  - For example, speech recognition

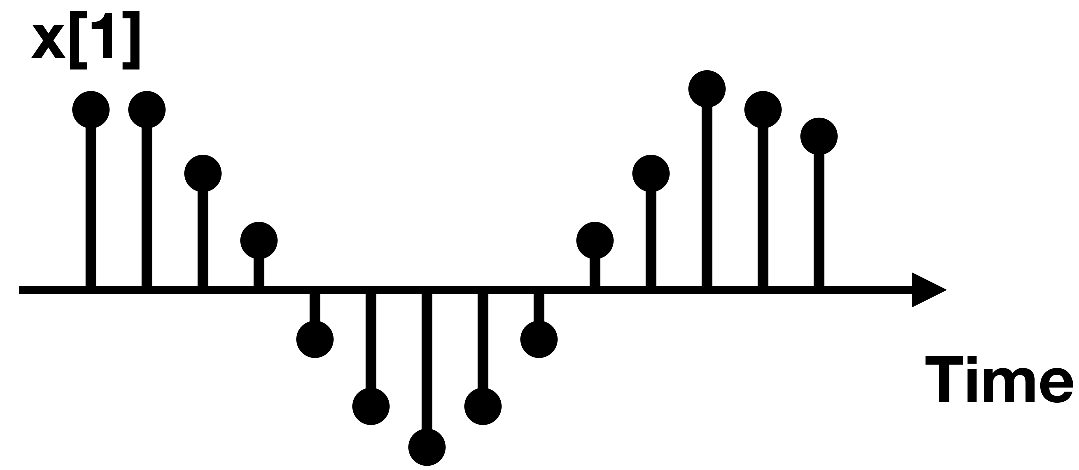


- A popular method for extracting features from a sequential data is convolution

Convolution

- Consider the task of classifying whether the signal ***x*** is high pitch (high frequency) or low pitch (low frequency)

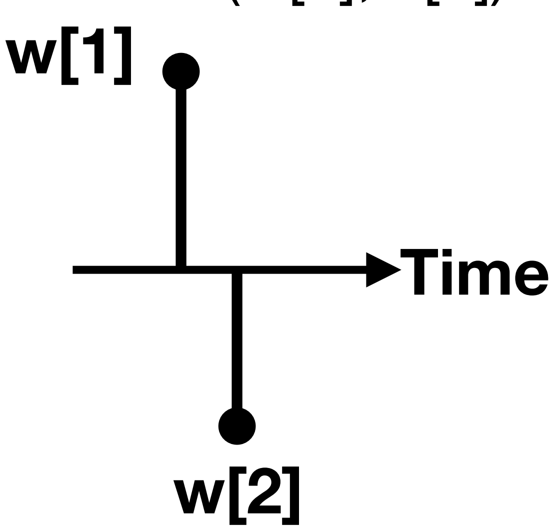




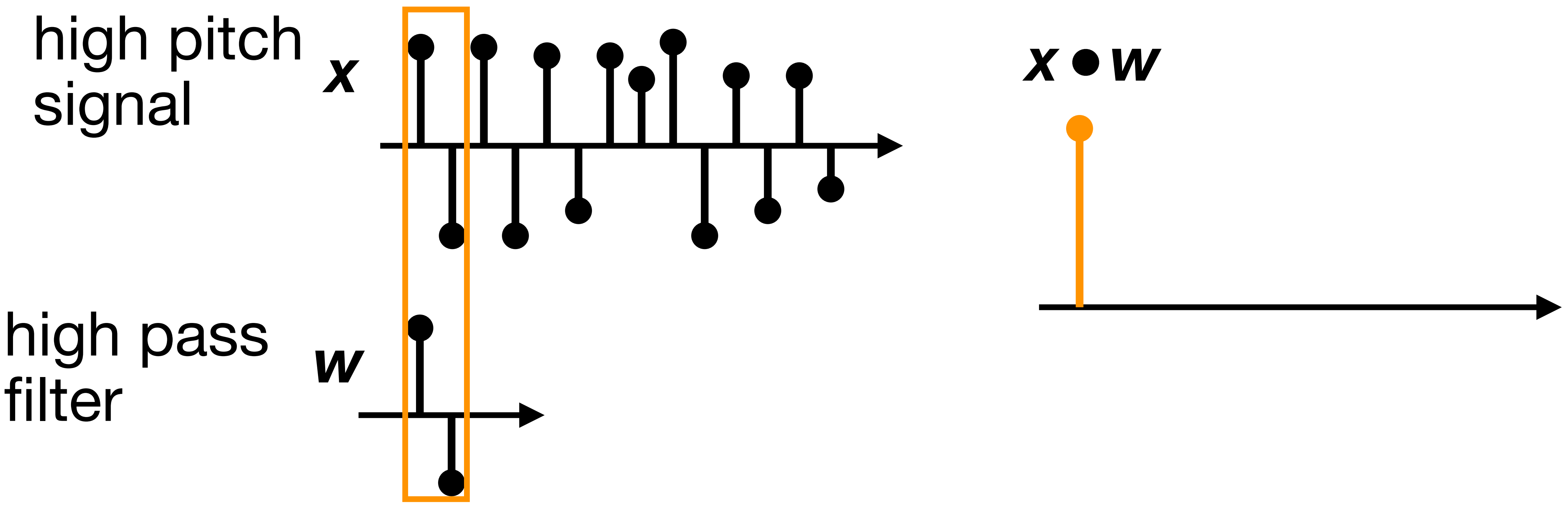
- We use a filter ***w*** and convolve ***x*** and ***w*** to get $\boldsymbol{x} \bullet \boldsymbol{w}$

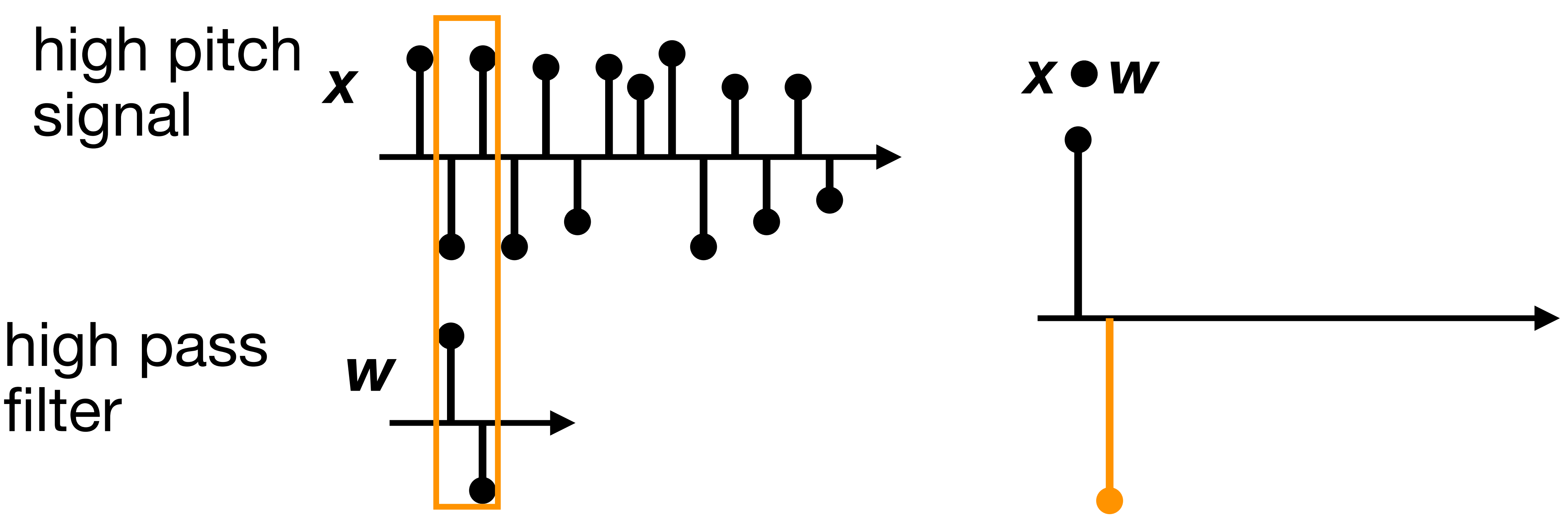
Example of length 2 filter

w=(w[1],w[2])

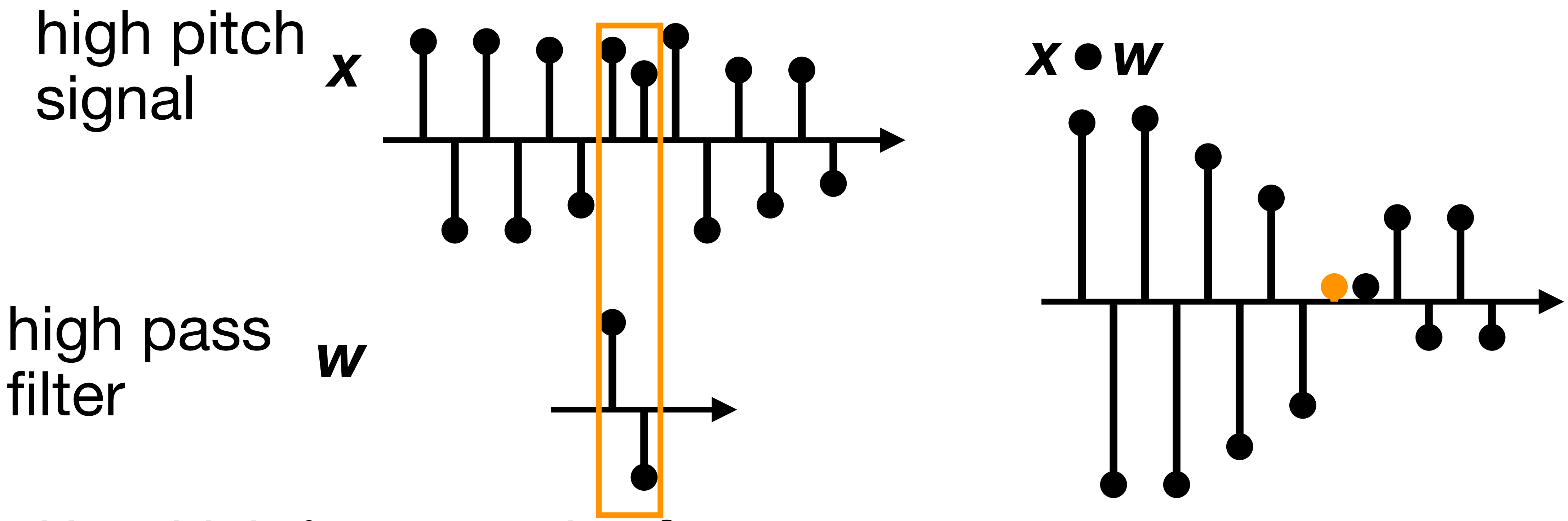


- Convolving high pass filter with a high pitch signal
- Slide the filter from left to right and compute the inner-product (entrywise product and sum)





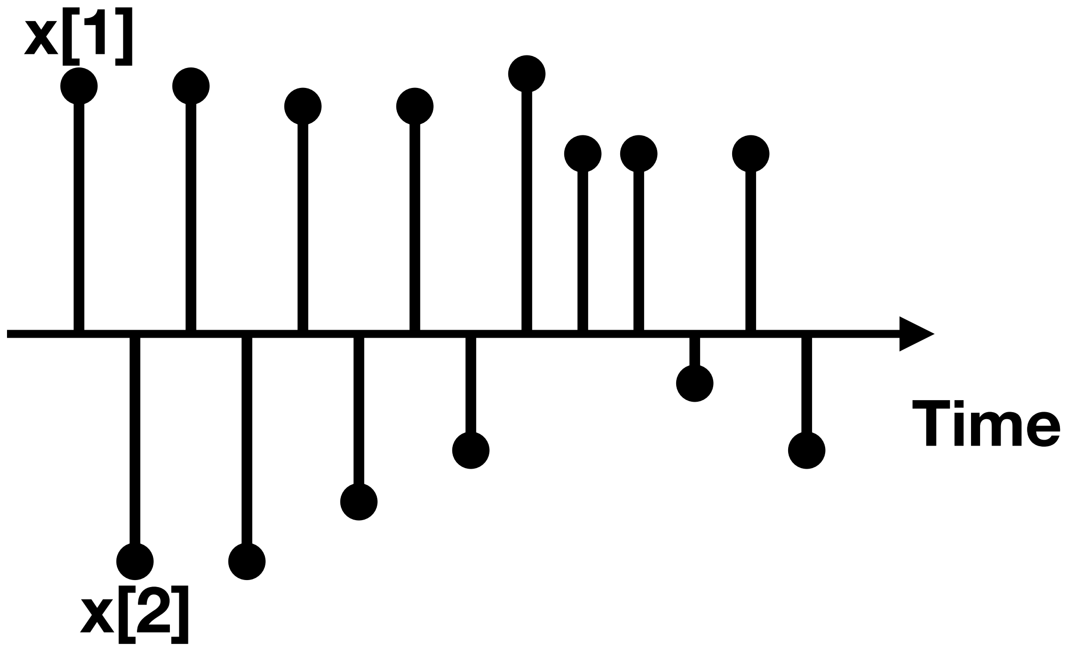
- Convolution

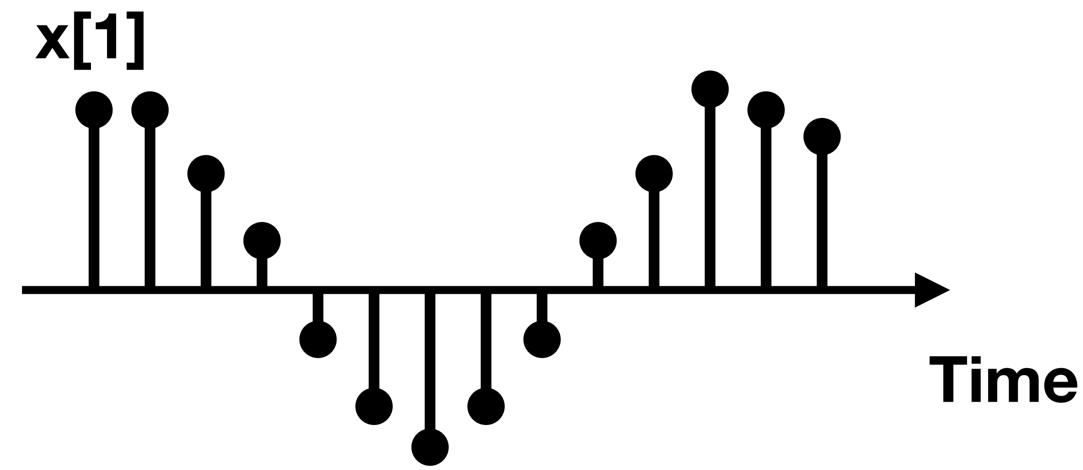


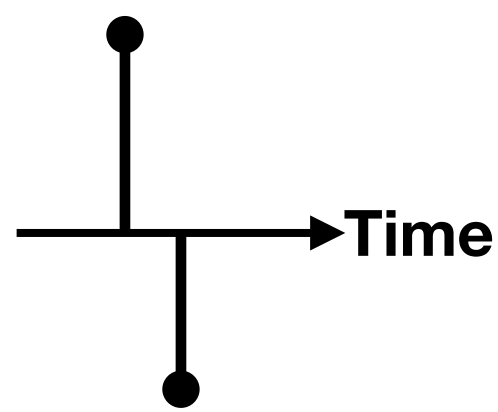
- How high frequency is ***x***?
- **Pooling** operation aggregates the data
  - max-pooling: $\boldsymbol{\max(x \bullet w)}$
  - Average pooling: (1/N)($|(x \bullet w)[1]| + \ldots + |(x \bullet w)[N]|$)

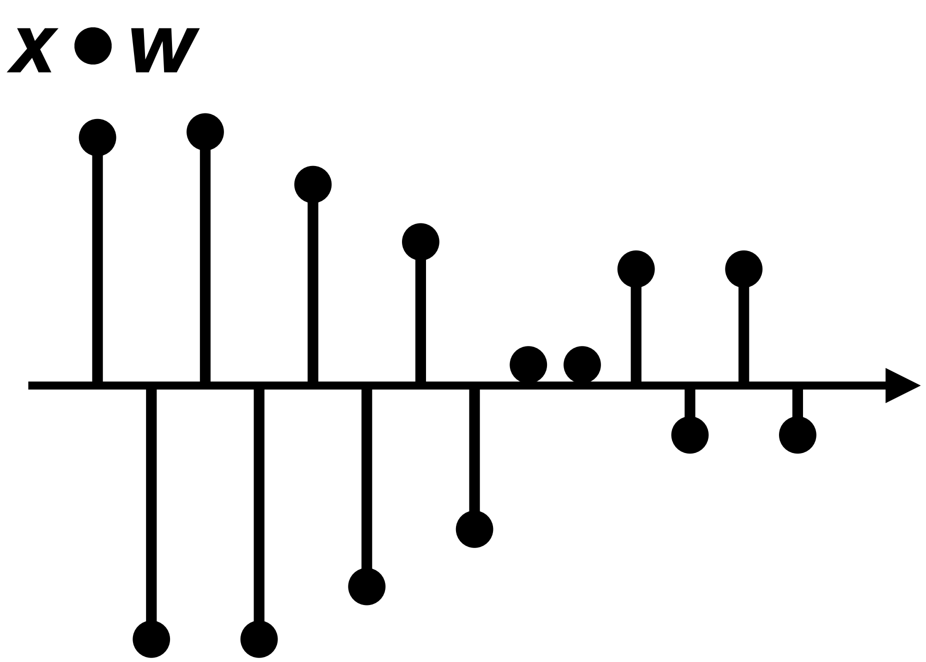
- Convolved and Pooled value will be large for high-frequency data

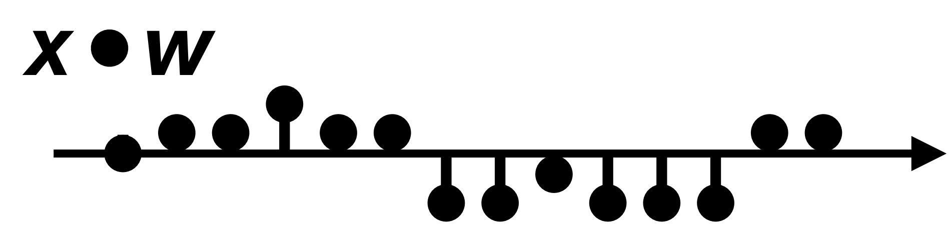
high-freq. signal vs. low-freq. signal







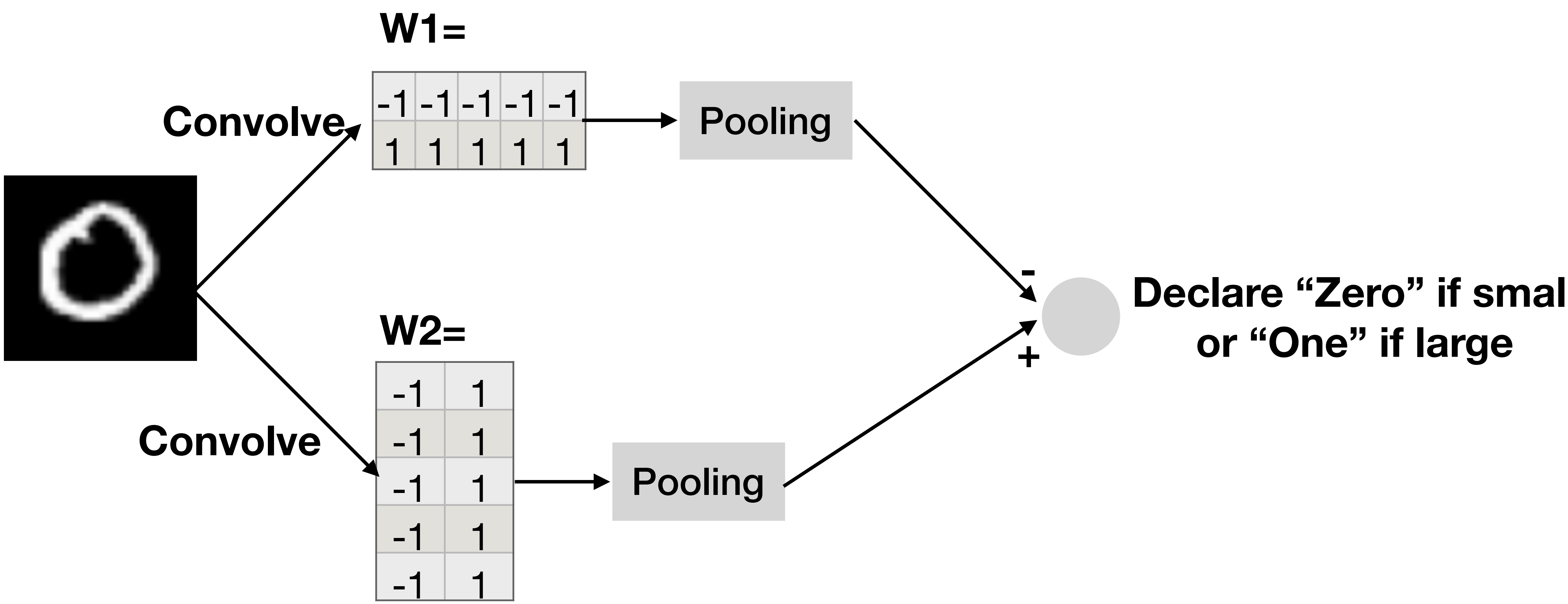




Two-dimensional convolution

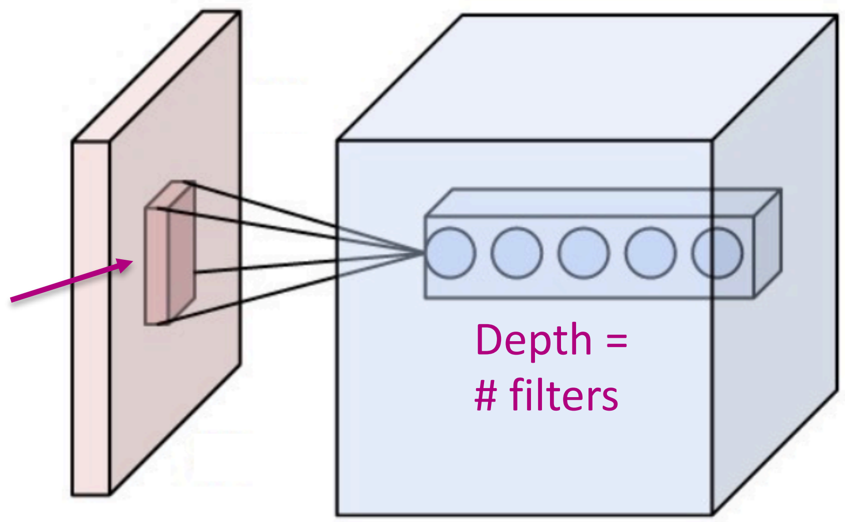
- Consider a task of classifying 0's and 1's

- One manual way is to use some 2-d filters

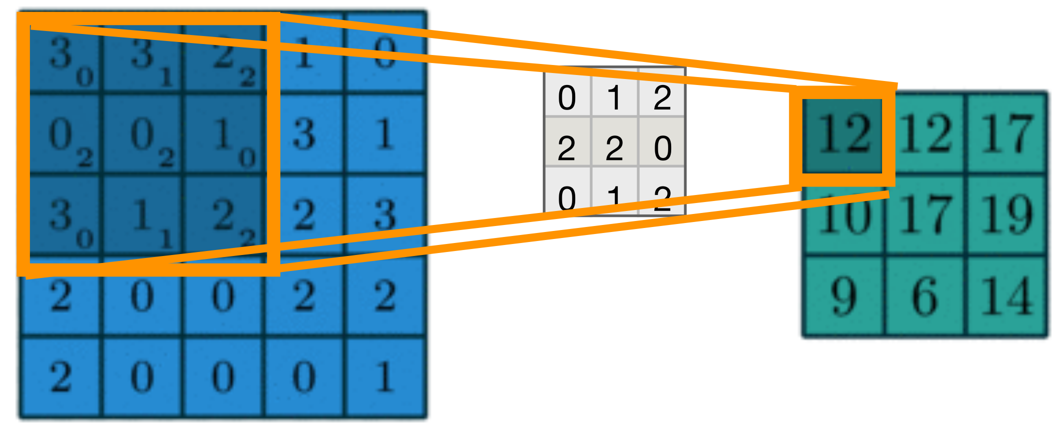


Example of convolutional layer with 3x3 filter (9 parameters)

To understand the convolution of 3-dimensional arrays (tensors),
let's consider the convolution of 2-dimensional arrays (matrices)



Input image **Output image**



In this example, we consider a 3x3 convolution

Sub-region in input image | **Filter represented by a matrix *W*** | **Output Image (pixel)**

$$\begin{bmatrix} 3 & 3 & 2 \\ 0 & 0 & 1 \\ 3 & 1 & 2 \end{bmatrix} \bullet \; W = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{bmatrix} \rightarrow \boxed{12}$$

$= \text{inner product of two matrices of the same size}$

$$\begin{bmatrix} 0 & 0 & 1 \\ 3 & 1 & 2 \\ 2 & 0 & 0 \end{bmatrix} \bullet \begin{bmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{bmatrix} \rightarrow \boxed{}$$

Key aspect of convolutional layer is that it applies the same filter (with the same weights) to all sub-regions [also known as weight sharing] → **This gives efficiency + shift invariance**

Convolution

- What is the output image?

W = $\begin{bmatrix} 0 & 1 \\ 2 & 0 \end{bmatrix}$ Filter (2x2)

X = $\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ Input Image (3x3)

Option A $\begin{bmatrix} 0 & 2 & 2 & 0 \\ 1 & 1 & 2 & 2 \\ 0 & 3 & 3 & 2 \\ 1 & 1 & 1 & 0 \end{bmatrix}$

Option B $\begin{bmatrix} 3 & 3 \\ 3 & 4 \end{bmatrix}$

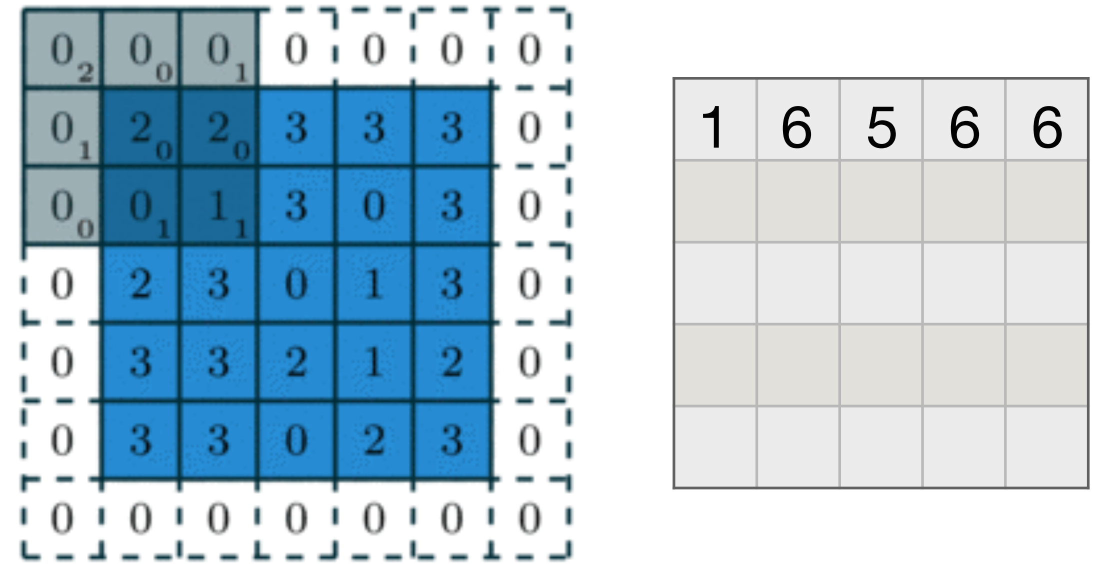
Option C $\begin{bmatrix} 1 & 2 \\ 3 & 3 \end{bmatrix}$

Option D $\begin{bmatrix} 0 & 7 \\ 14 & 0 \end{bmatrix}$

In practice,

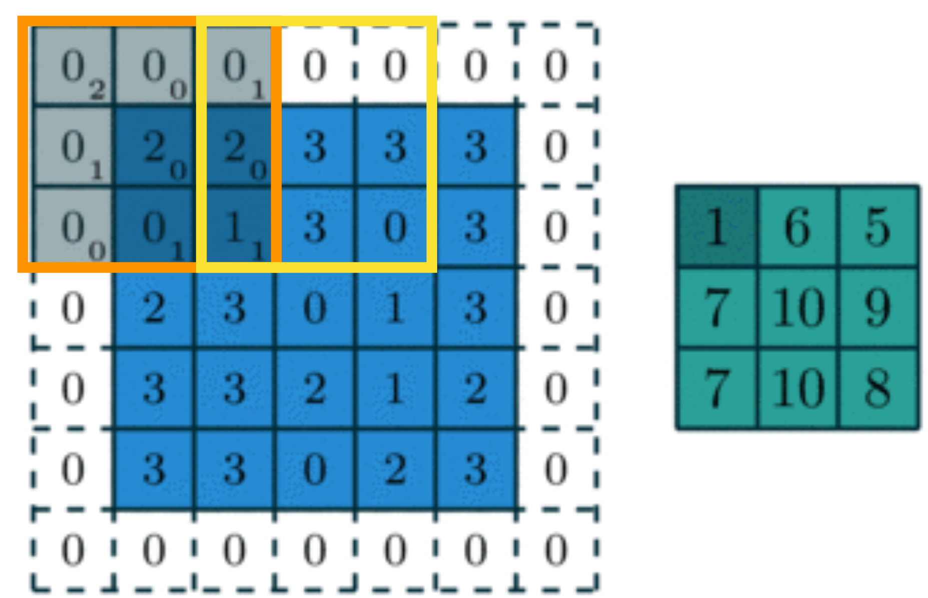
- We commonly use convolution with **zero-padding** and **stride**

- **Zero-padding:**
- Pad zeros around the boundary to preserve information and avoid boundary effect



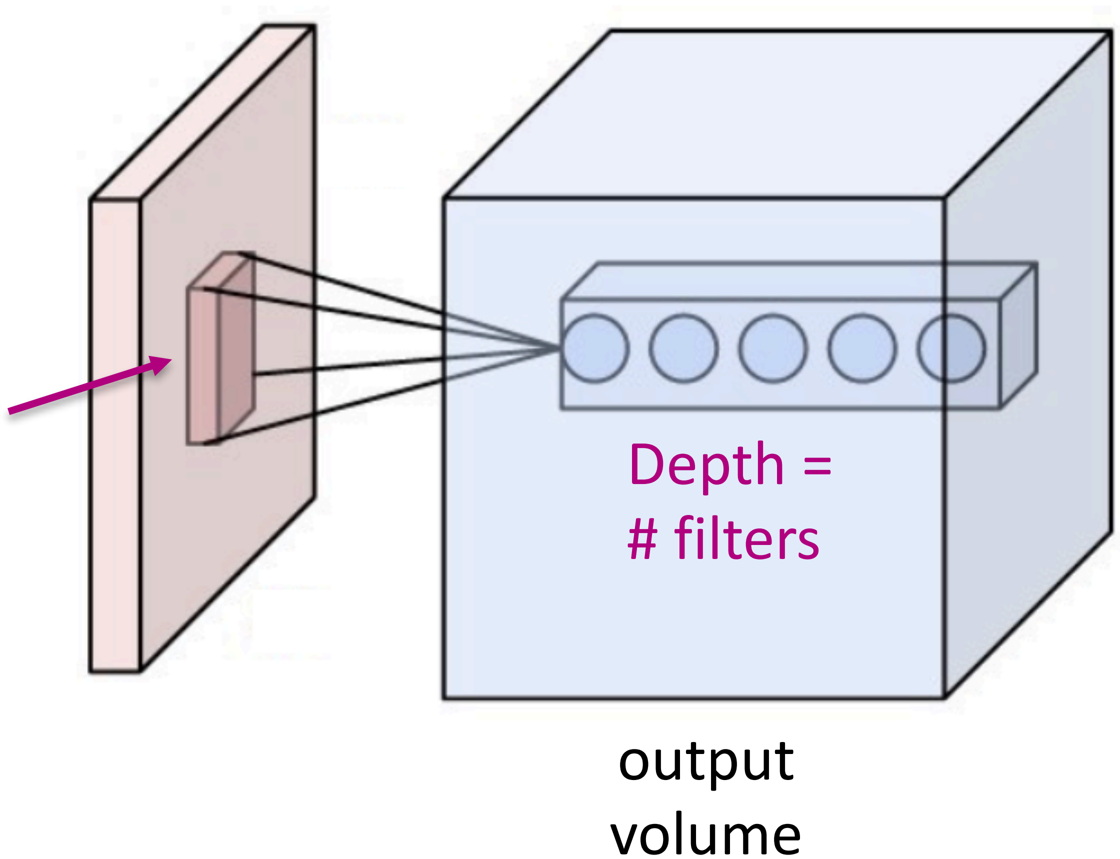
If we have 7x7 filter, how many zeros do we need to pad on one row on one side?

- **Stride:**
- skip patches periodically to reduce redundancy and increase efficiency, and capture different resolutions



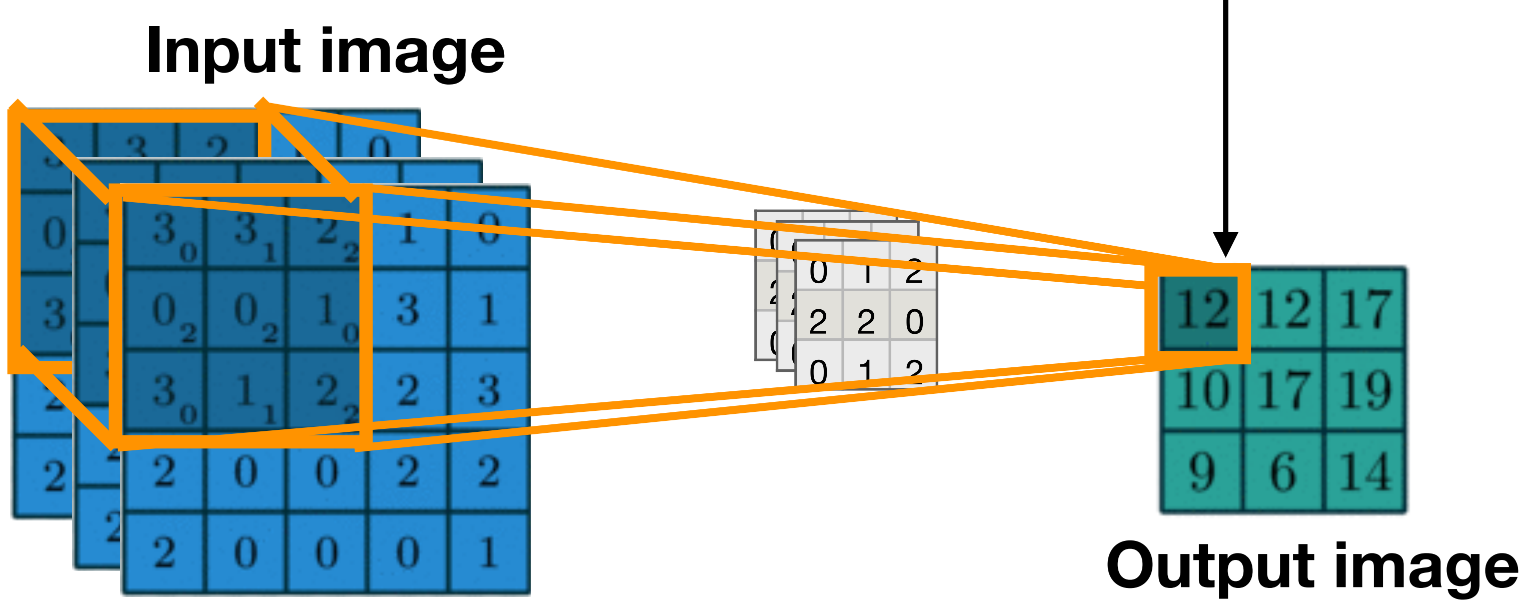
This is an example with 3x3 filter and **stride 2**

Component in CNN: Convolutional layer



In image processing, convolution is typically an operation over three-dimensional arrays

each scalar output = inner product of two tensors of the same size (3x3x3)

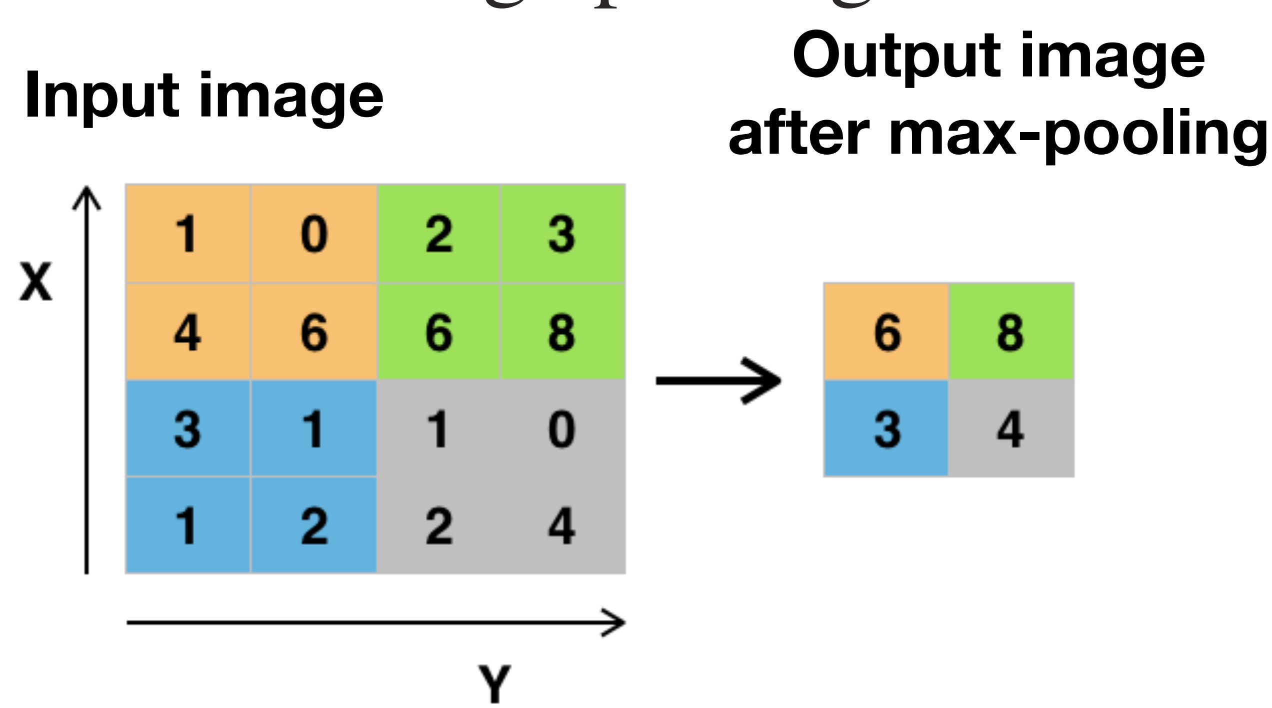


Component in CNN: Pooling layer

- Downsampling the spatial dimensions

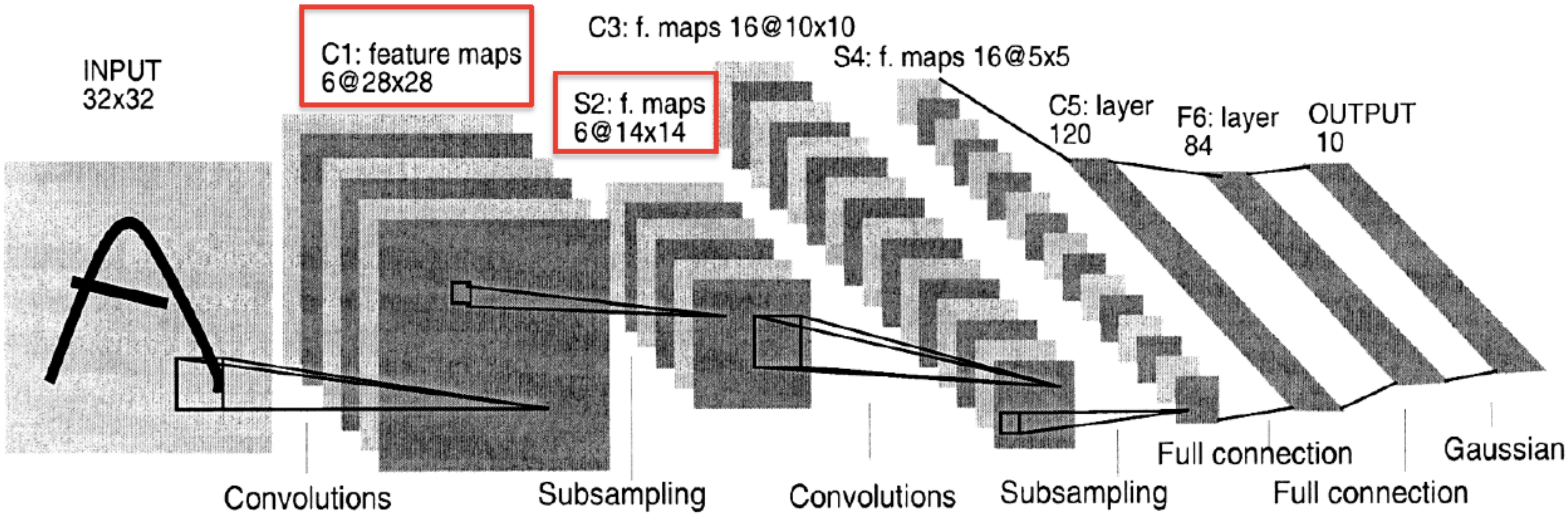
- Common to insert between successive **conv** layers

- Typically, **max pooling of size 2x2 with stride 2**
  - Applied separately to each depth slice
  - Tends to work better than average pooling



Performance of deep learning

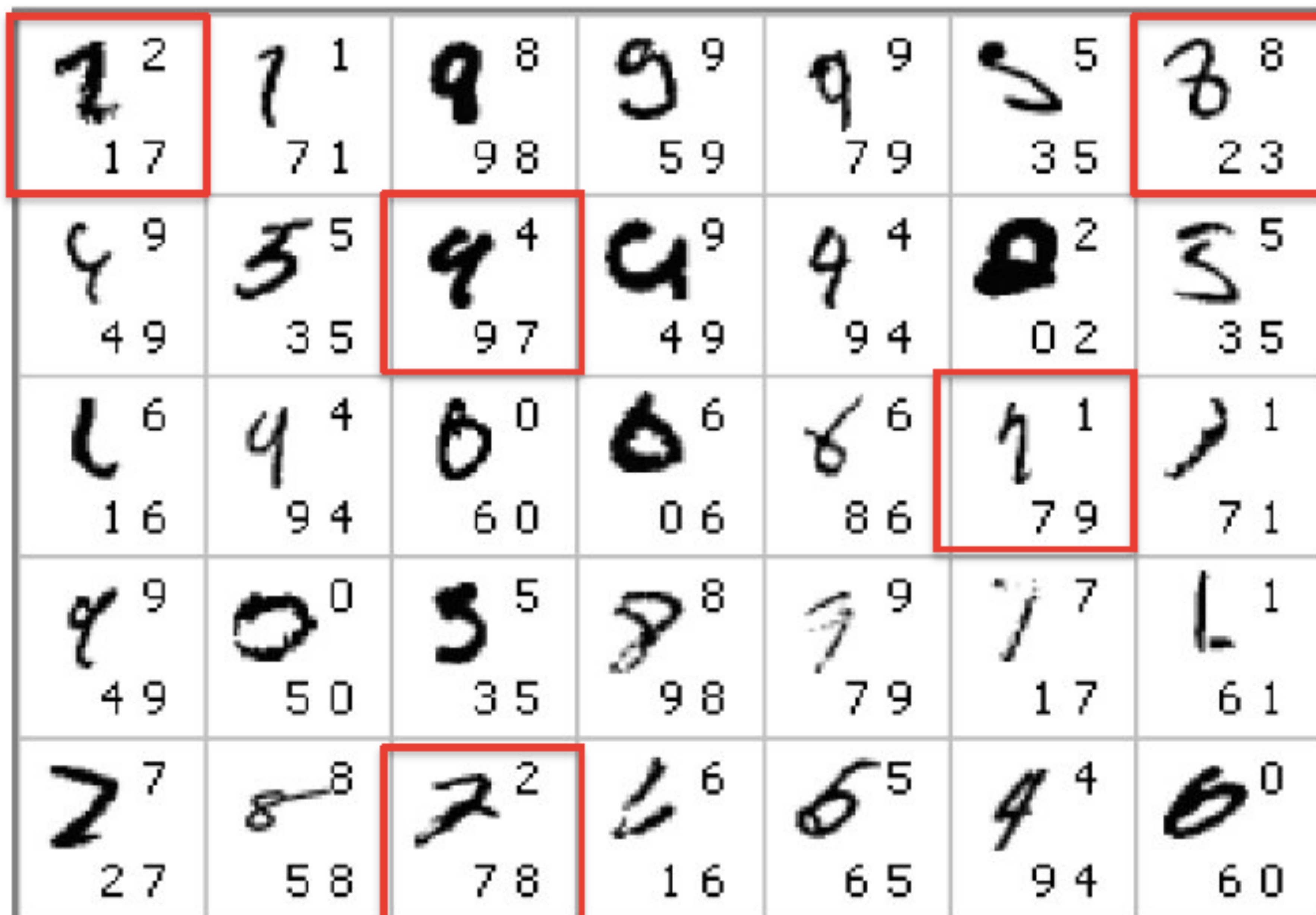
- LeNet, 1990's



- 82 error made by LeNet on MNIST

35 error made by Ciresan et al.
further, most of the time the true answer is in the top-2 prediction

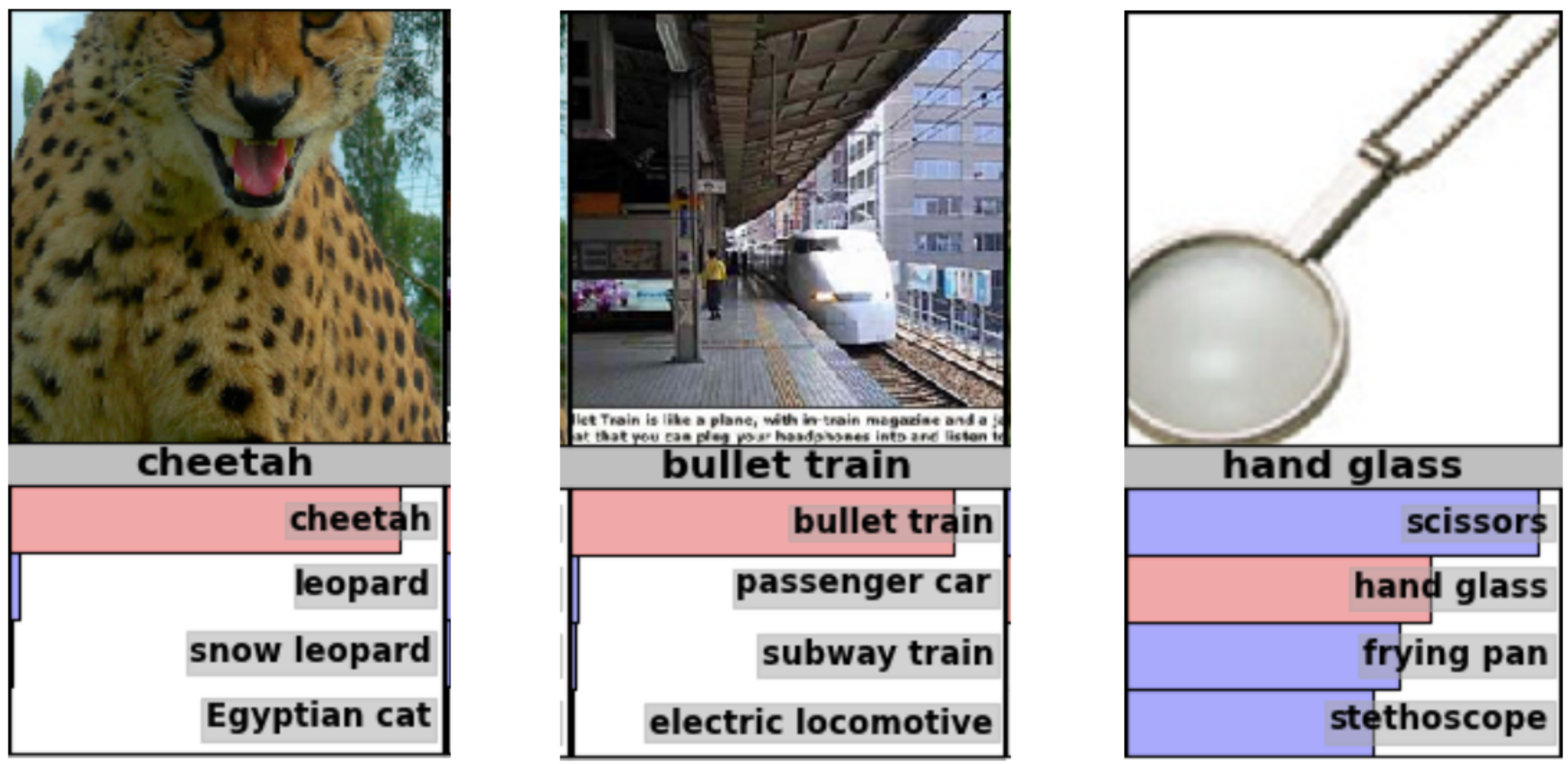
idea: train with transformed samples [data augmentation]



ImageNet 2012 competition: 1.2M training images

Challenging dataset:

High-dimensional data from previous 28 x28 grey-scale to now 256x256 color
10 classes to 1,000 classes
multiple objects
natural 3-d scene

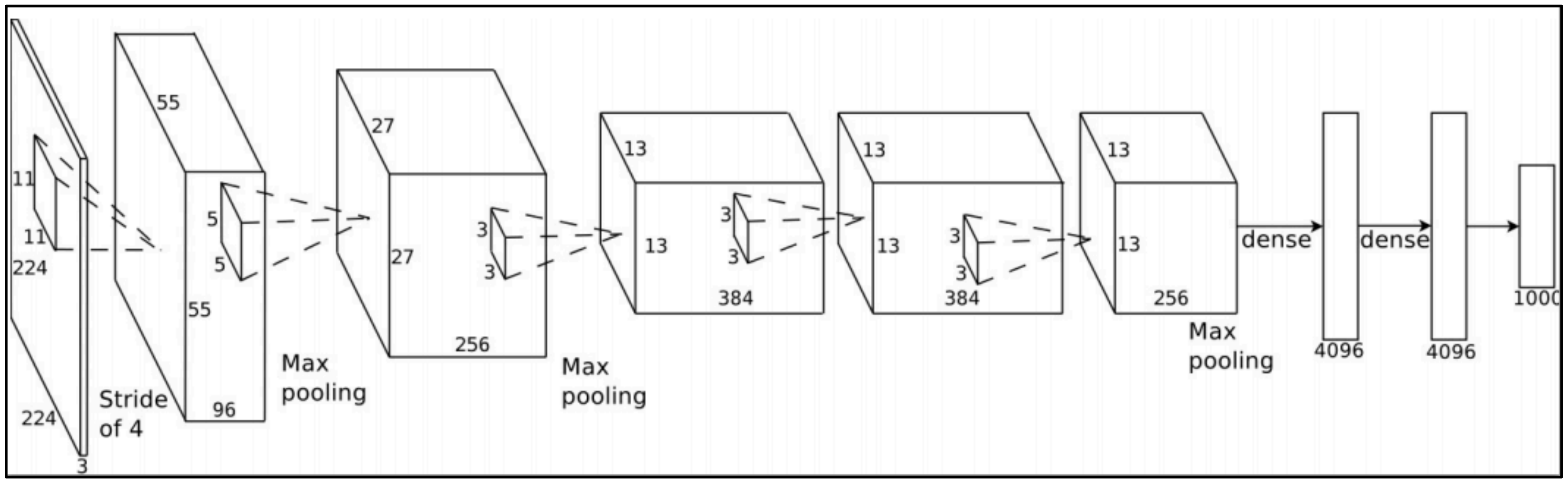


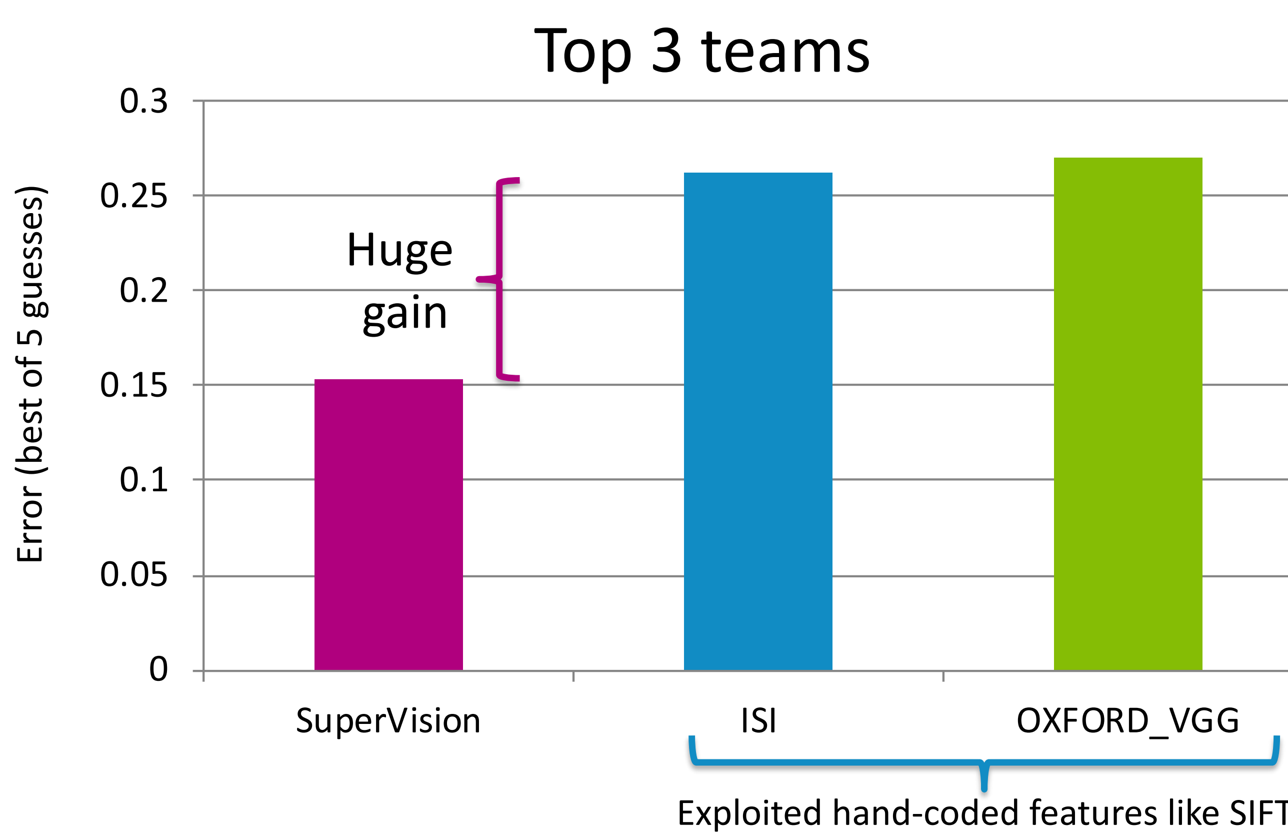
ImageNet 2012 competition:
1.2M training images, 1000 categories

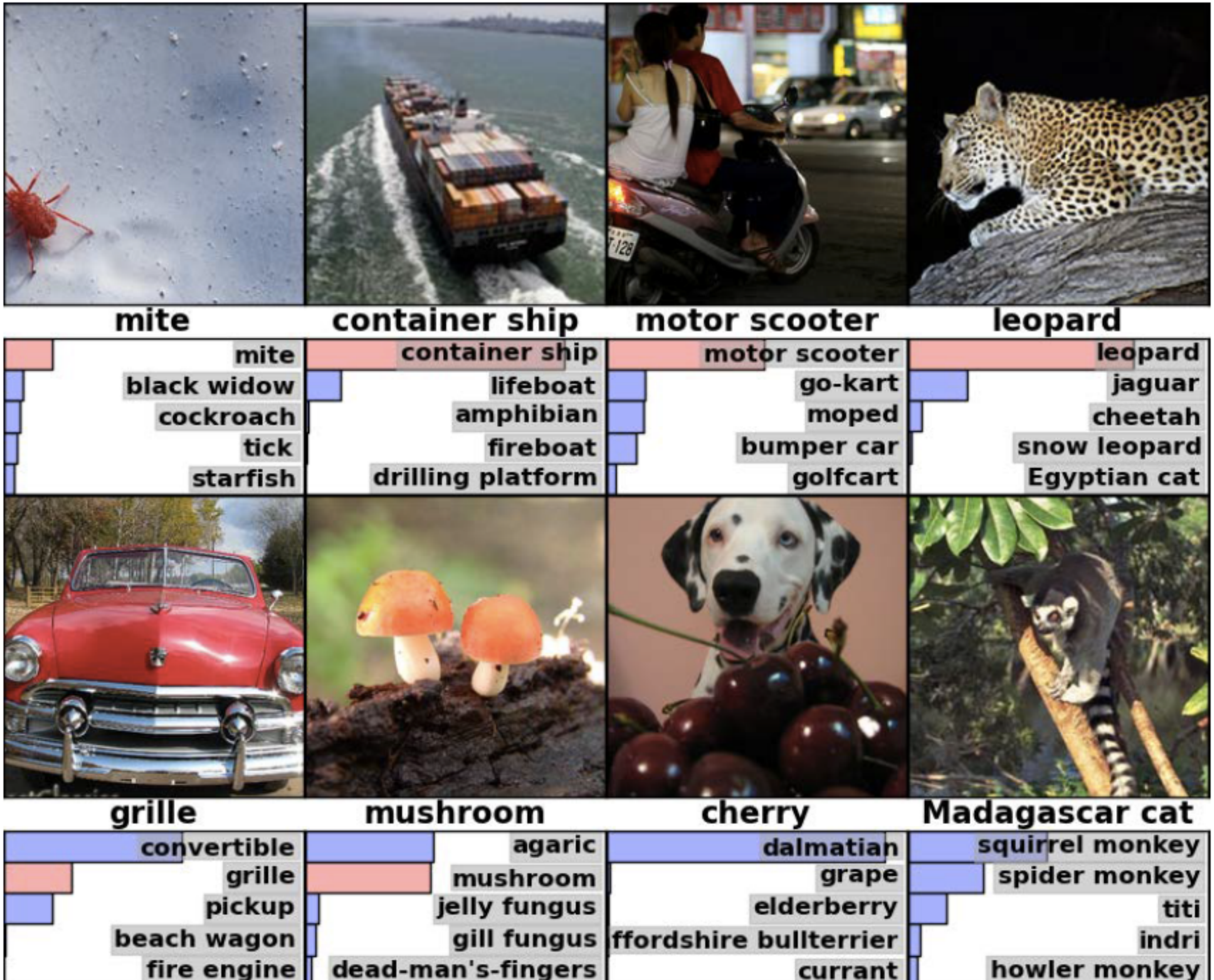
Winning entry: SuperVision

8 layers, 60M parameters [Krizhevsky et al. ’12]

AlexNet



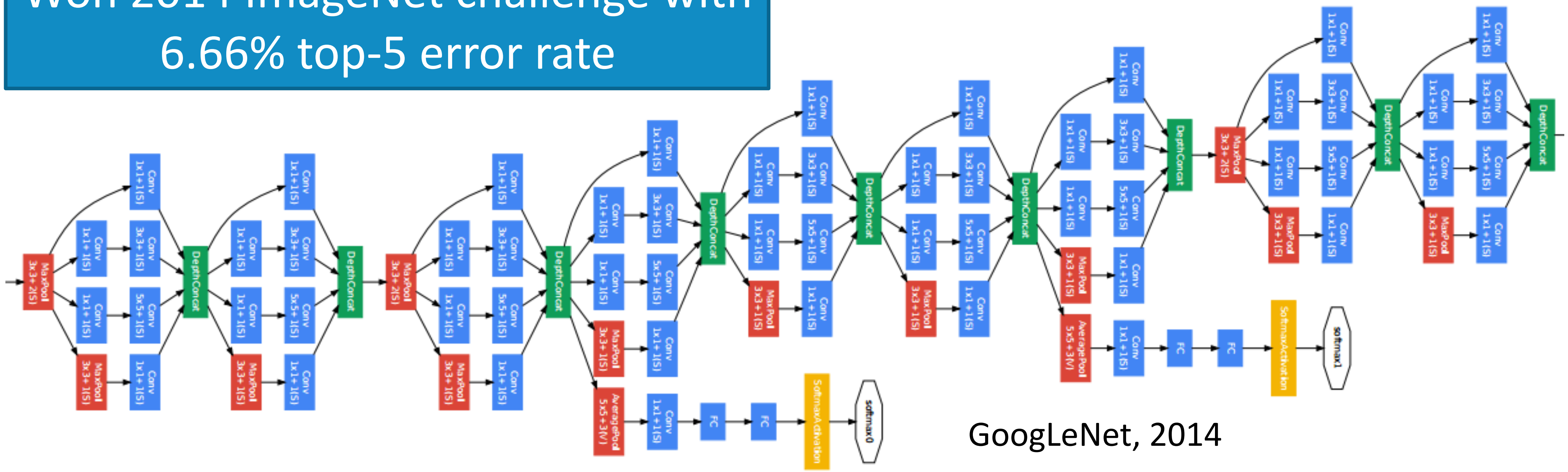




mite
mite
black widow
cockroach
tick
starfish
container ship
container ship
lifeboat
amphibian
fireboat
drilling platform
motor scooter
motor scooter
go-kart
moped
bumper car
golfcart
leopard
leopard
jaguar
cheetah
snow leopard
Egyptian cat
grille
convertible
grille
pickup
beach wagon
fire engine
mushroom
agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers
cherry
dalmatian
grape
elderberry
ffordshire bullterrier
currant
Madagascar cat
squirrel monkey
spider monkey
titi
indri
howler monkey

Going even deeper…

Won 2014 ImageNet challenge with 6.66% top-5 error rate

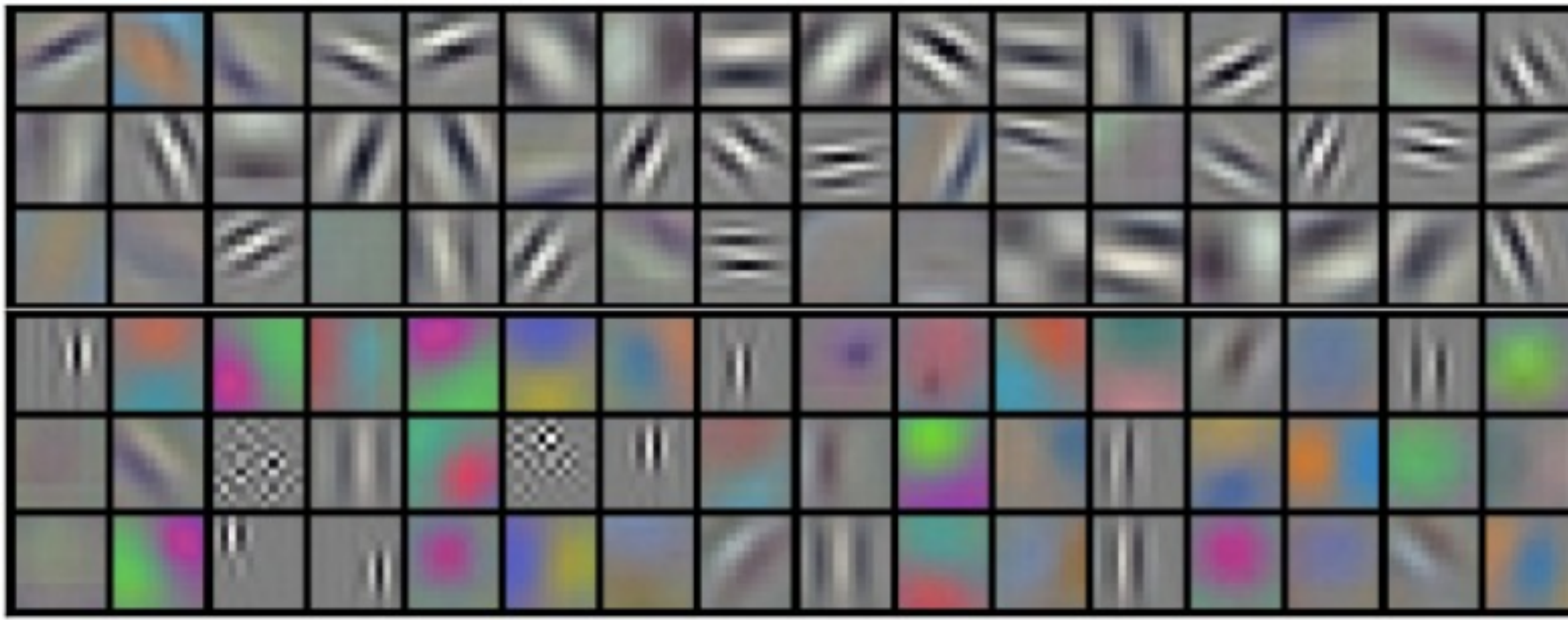


GoogLeNet, 2014

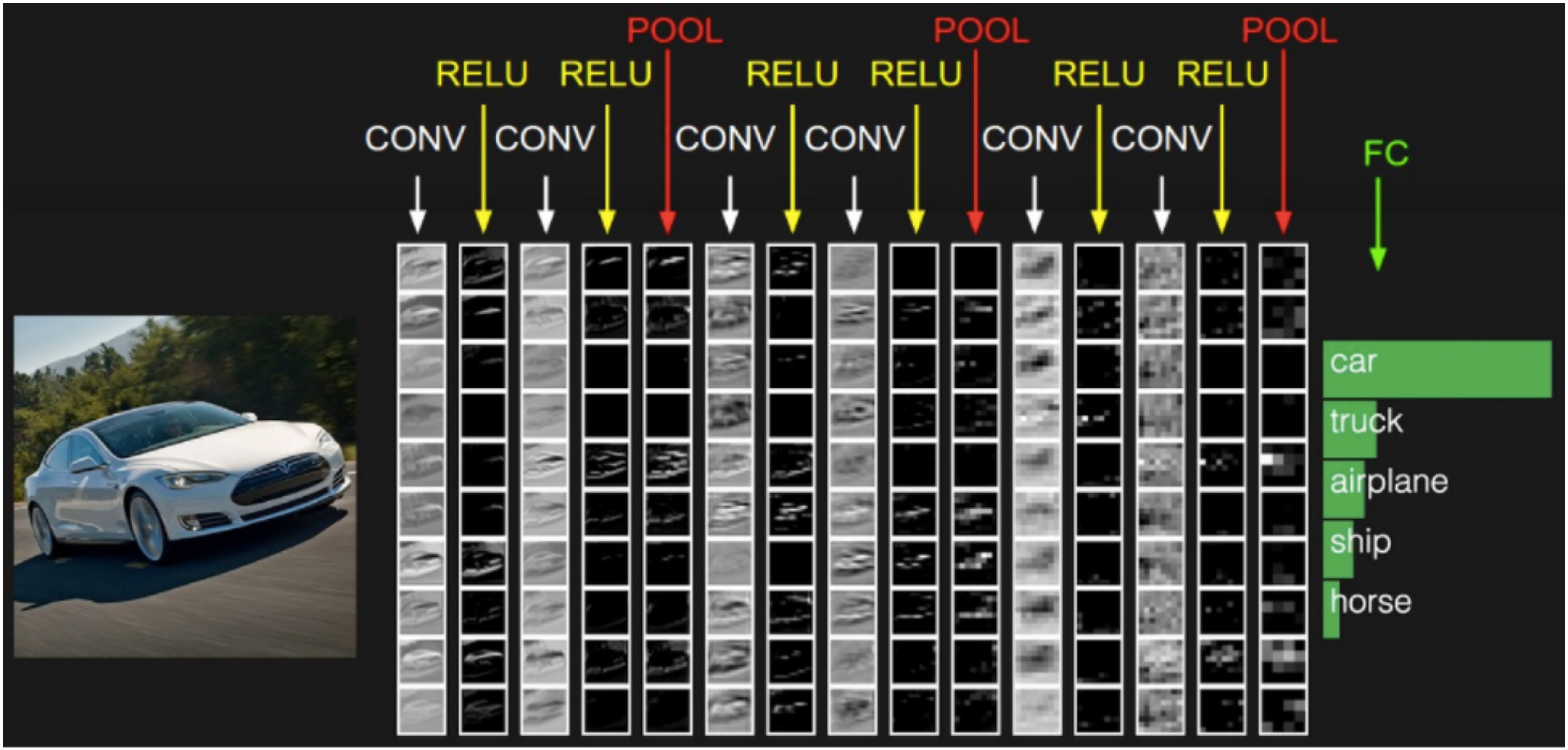
Huge CNN depth has proven helpful in recognition systems… Maybe because images contain hierarchical structure (faces contain eyes contain edges, etc.)

What happens when we train convolutional neural networks?

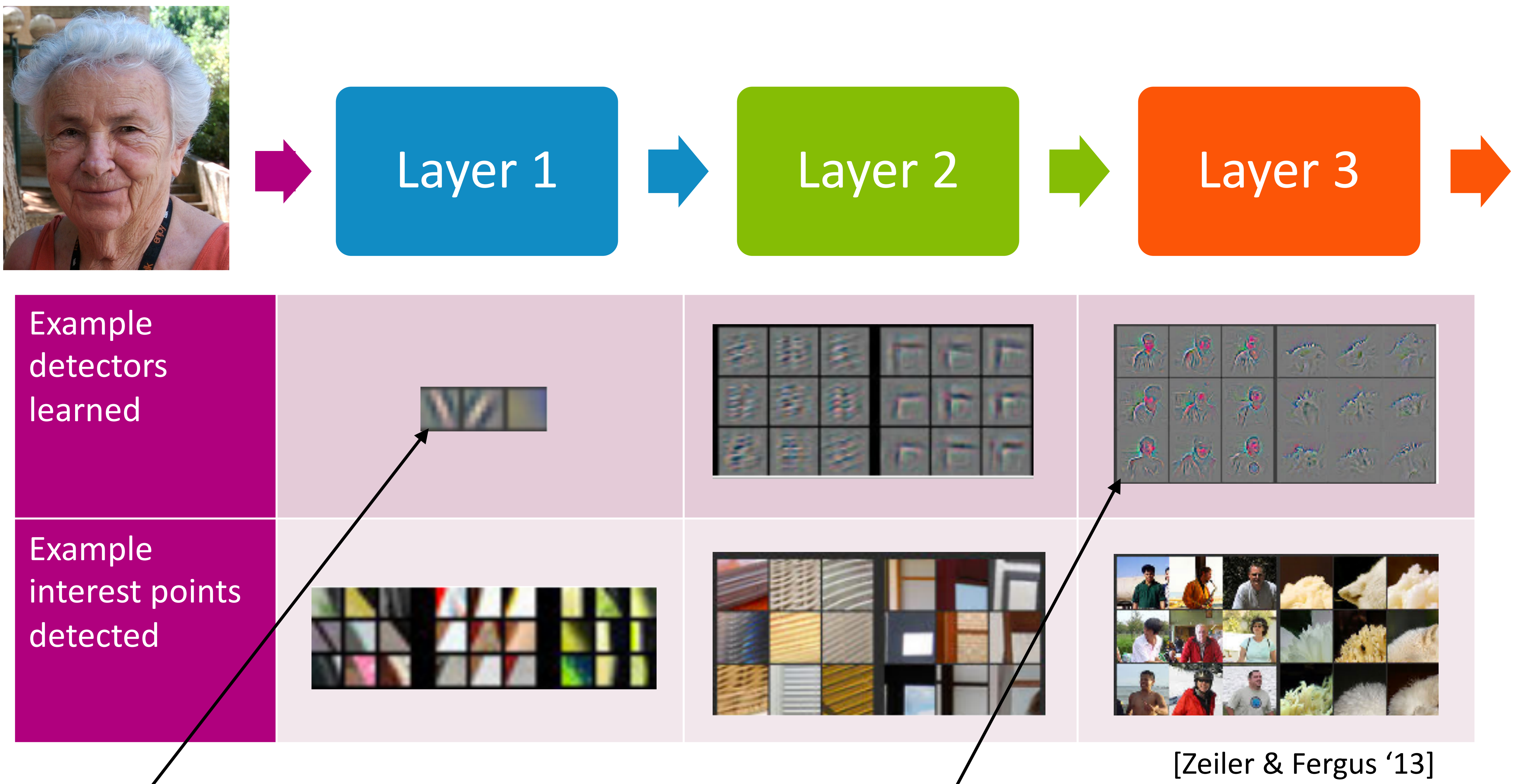
- First convolutional layer trained on natural images looks like the following



- Simple geometric patterns are “detected” or “matched” in the first layer


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

Returning to our example…
“Detectors” are the learned filters



[Zeiler & Fergus ‘13]

Filter *W* at layer 1 **Feature map at layer 3 shown on image space**

Performance of deep learning

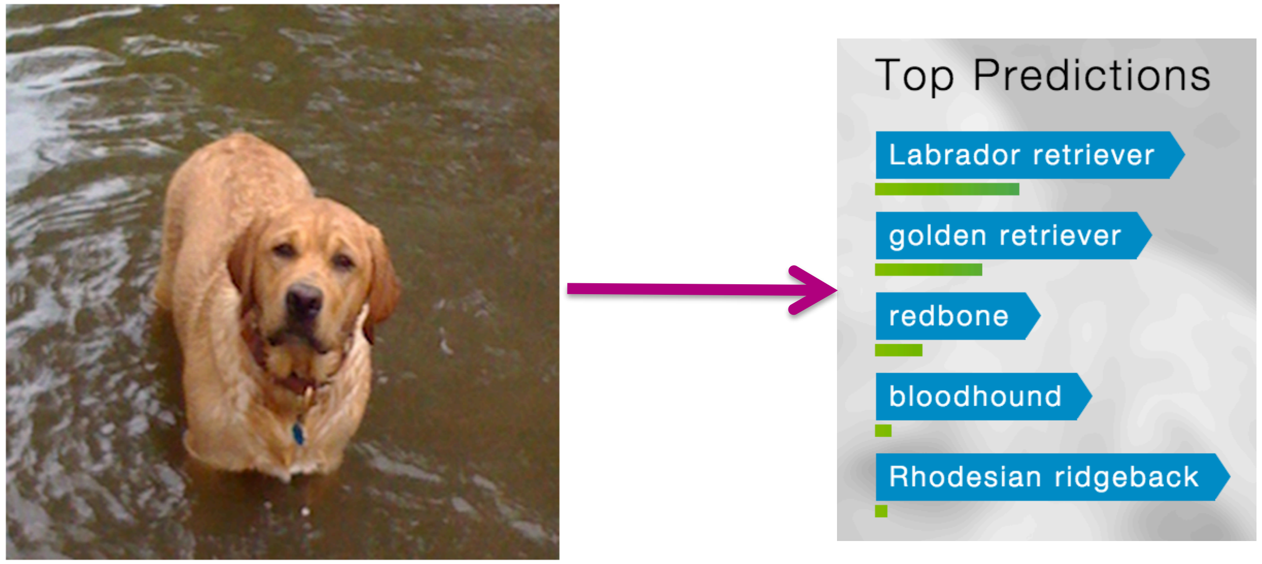
German traffic sign recognition benchmark

- 99.5% accuracy (IDSIA team)

House number recognition

- 97.8% accuracy per character [Goodfellow et al. ’13]

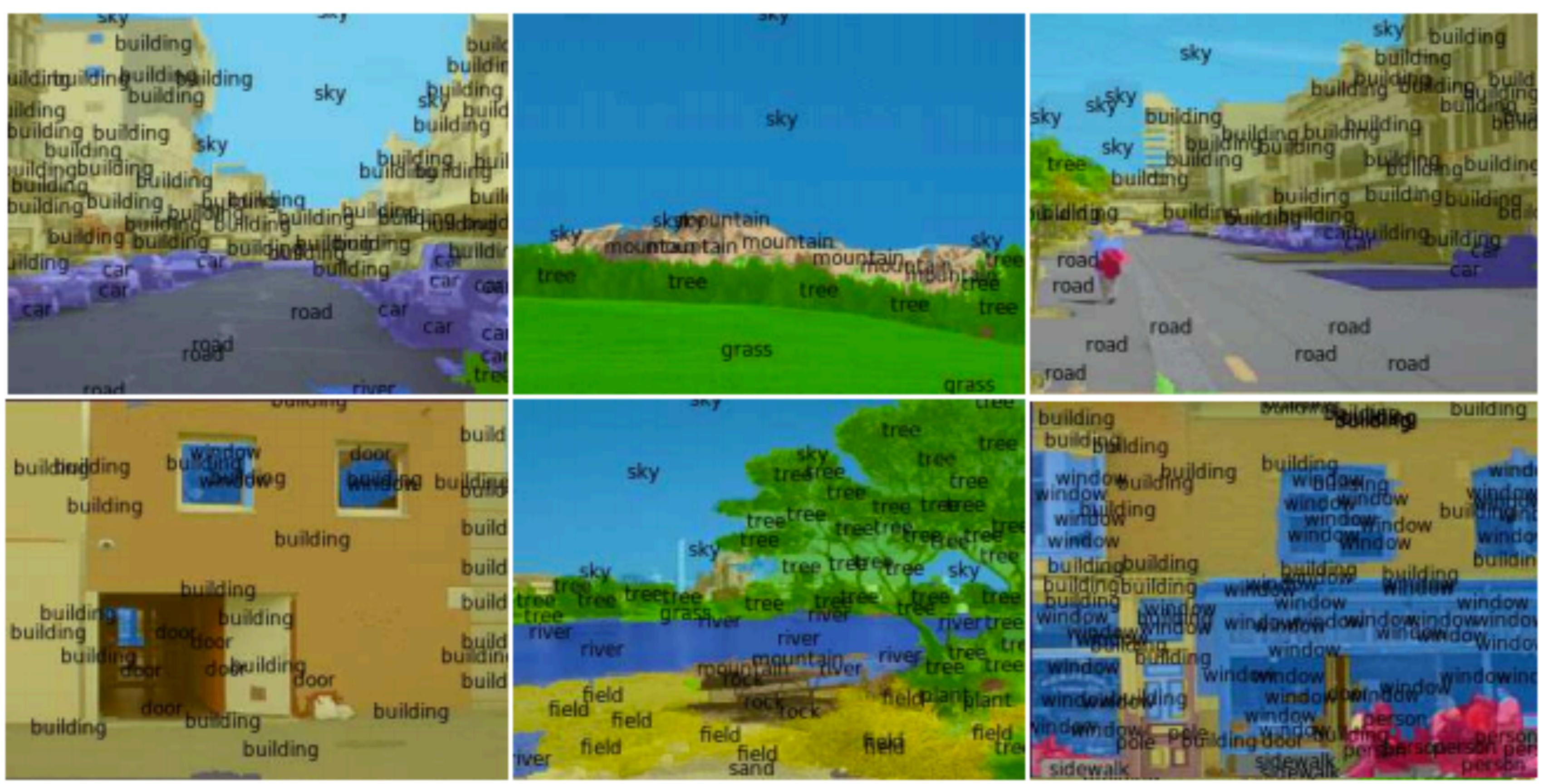
- Image classification



Input: **x**
Image pixels

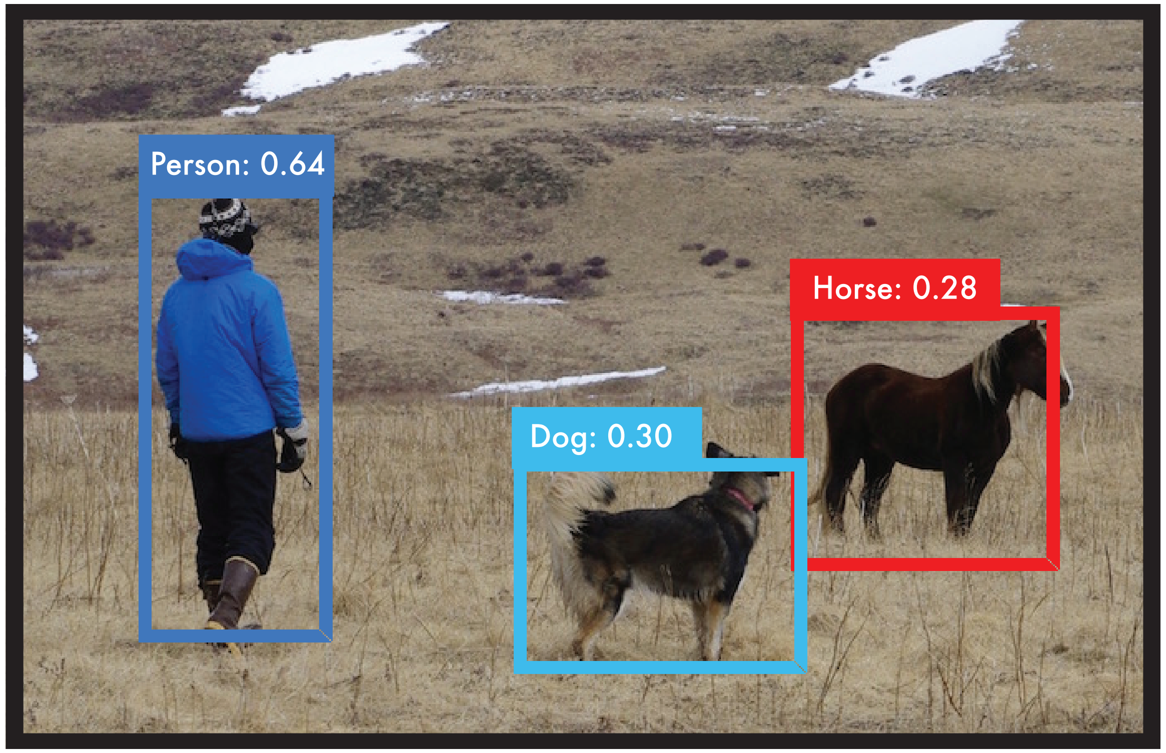
Output: y
Predicted object

- Scene parsing



[Farabet et al. '13]

- Object detection



Redmon et al. 2015
http://pjreddie.com/yolo/

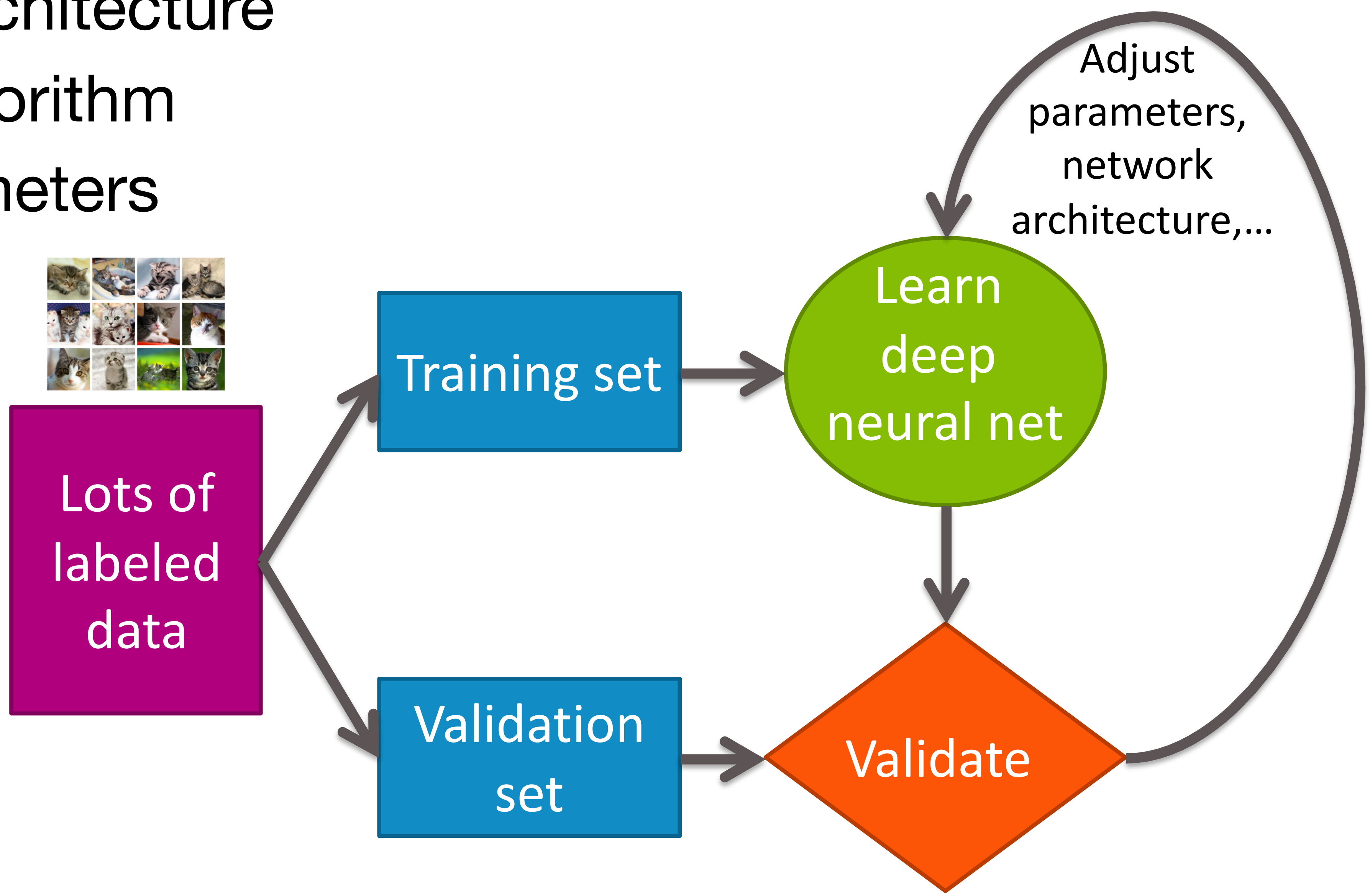
- Retrieving similar objects

Deep Learning practice

- Pros
  - Instead of manually engineering features, enable automated learning of features
  - Impressive performance gains in practice
    - Image processing
    - Natural language processing
    - Speech recognition
  - Making huge impacts in many applications in many fields

Deep Learning practice

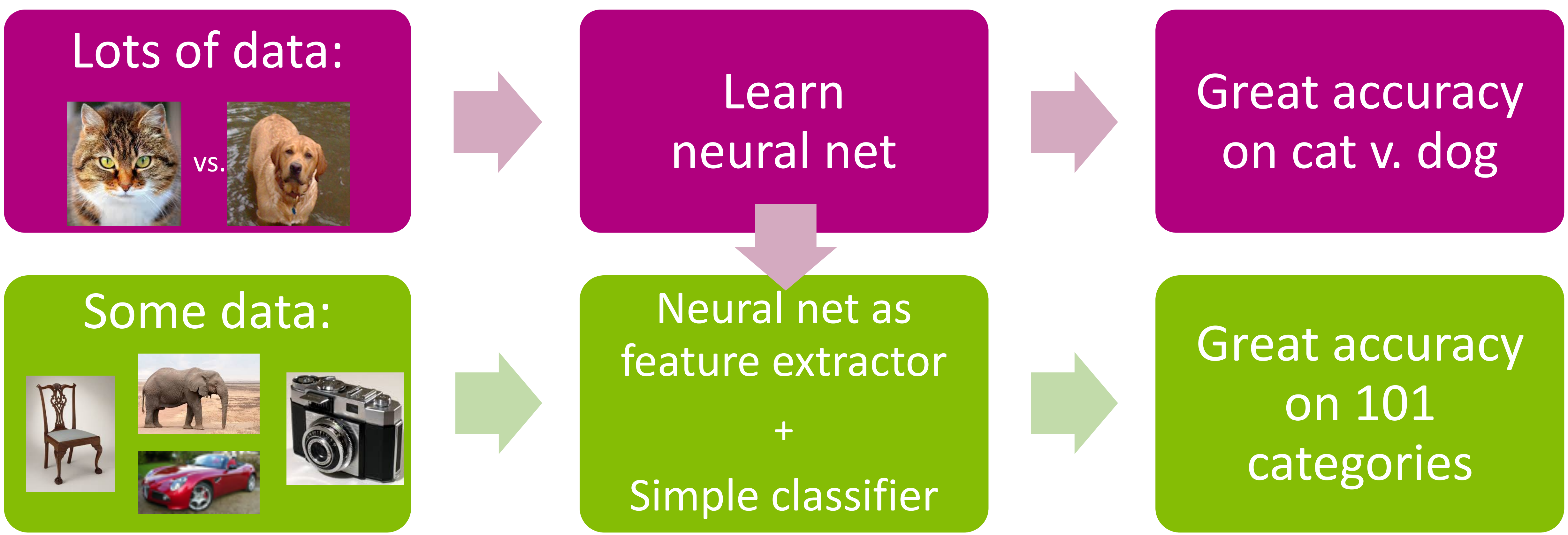
- Cons
  - Requires a lot of data
  - Computationally really expensive
  - Hard to tune hyper-parameters
    - Choice of architecture
    - Learning algorithm
    - Hyper-parameters



Transfer Learning

Transfer Learning

- Transfer Learning
  - Use data from one task to help learn on another task
  - Old idea, explored for deep learning by Donahue et al. ’14 & others
  -

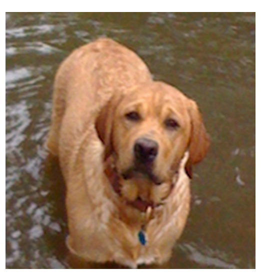


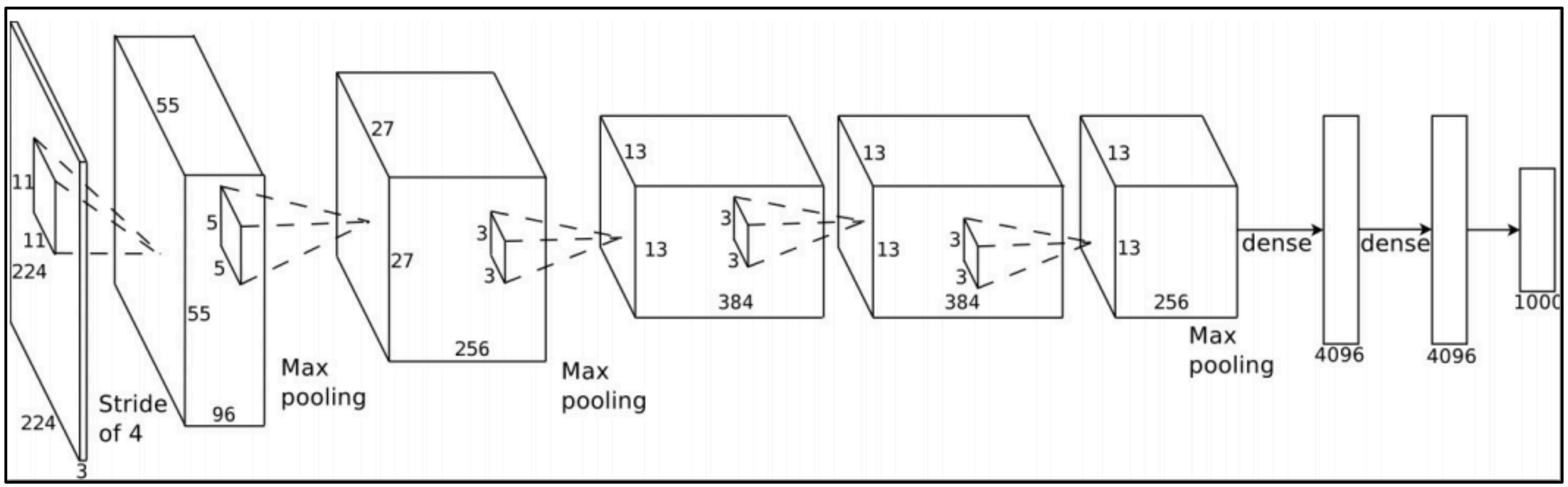
What is learned in a neural networks

- Initial layers are not too sensitive/specific to the task at training

Neural net trained for Task 1: cat vs. dog

vs.



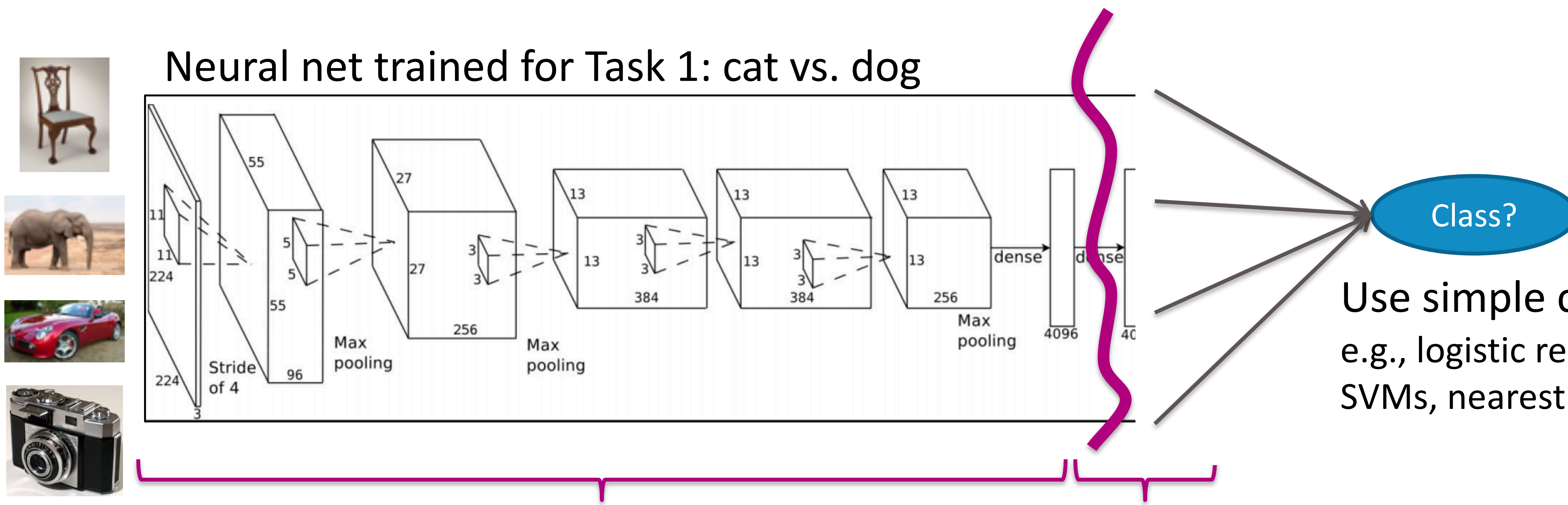


More generic
Can be used as feature extractor

Very specific
to Task 1
Should be ignored
for other tasks

Transfer learning

- For the second task of predicting 101 categories, (re)-train only the last layer of the neural network



Use simple classifier

e.g., logistic regression, SVMs, nearest neighbor,...

More generic
Can be used as feature extractor

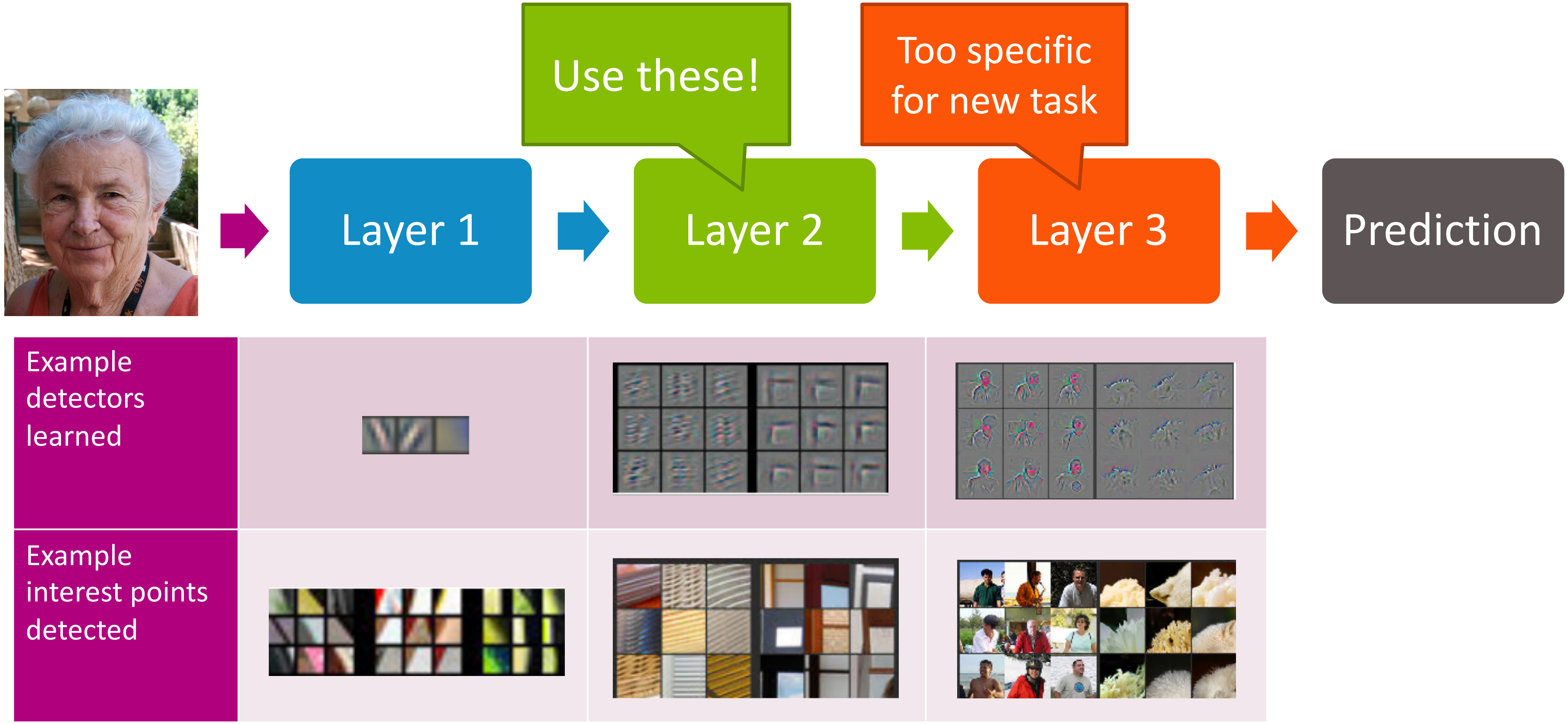
Very specific to Task 1
Should be ignored for other tasks

Keep weights fixed!

Re-train

Transfer learning

- Need to be careful about where you cut, as latter layers may be too task specific



[Zeiler & Fergus '13]