Visit to
Neural Network to
Deep Learning

R. K. Agrawal
School of Computer and Systems Sciences
Jawaharlal Nehru University
NewDelhi-110067

Outline

- Typical goal of machine Learning
- Neural Network
- Deep learning
- Some common deep learning algorithms

*Many of slides adapted from Andrew Ng and G . Hinton

Typical goal of machine learning

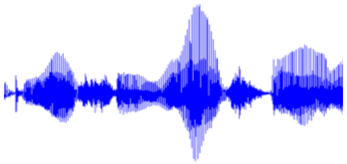
input | **output**

images/video

ML

Label: “Motorcycle”
Suggest tags
Image search
…

audio



ML

Speech recognition
Music classification
Speaker identification
…

text

ML

Web search
Anti-spam
Machine translation
…

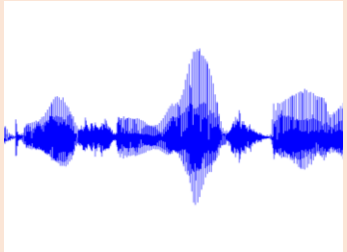
Typical goal of machine learning

Feature engineering: most time consuming!

input

images/video

audio



text

ML

ML

ML

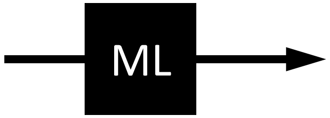
tput

Label: “Motorcycle”
Suggest tags
Image search
…

Speech recognition
Music classification
Speaker identification
…

Web search
Anti-spam
Machine translation
…

Our goal in object classification



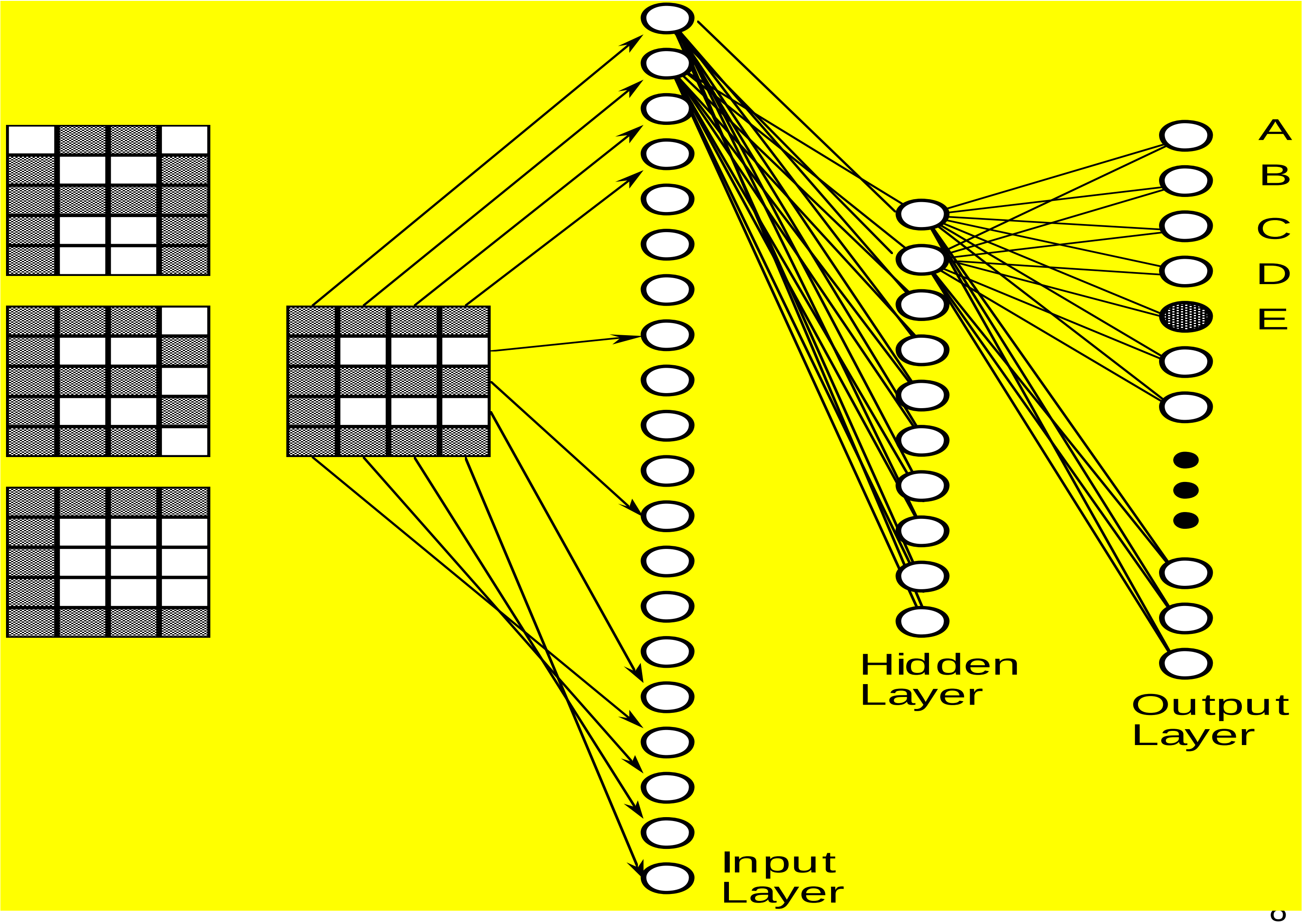
“motorcycle”

Face Recognition




Fingerprint recognition

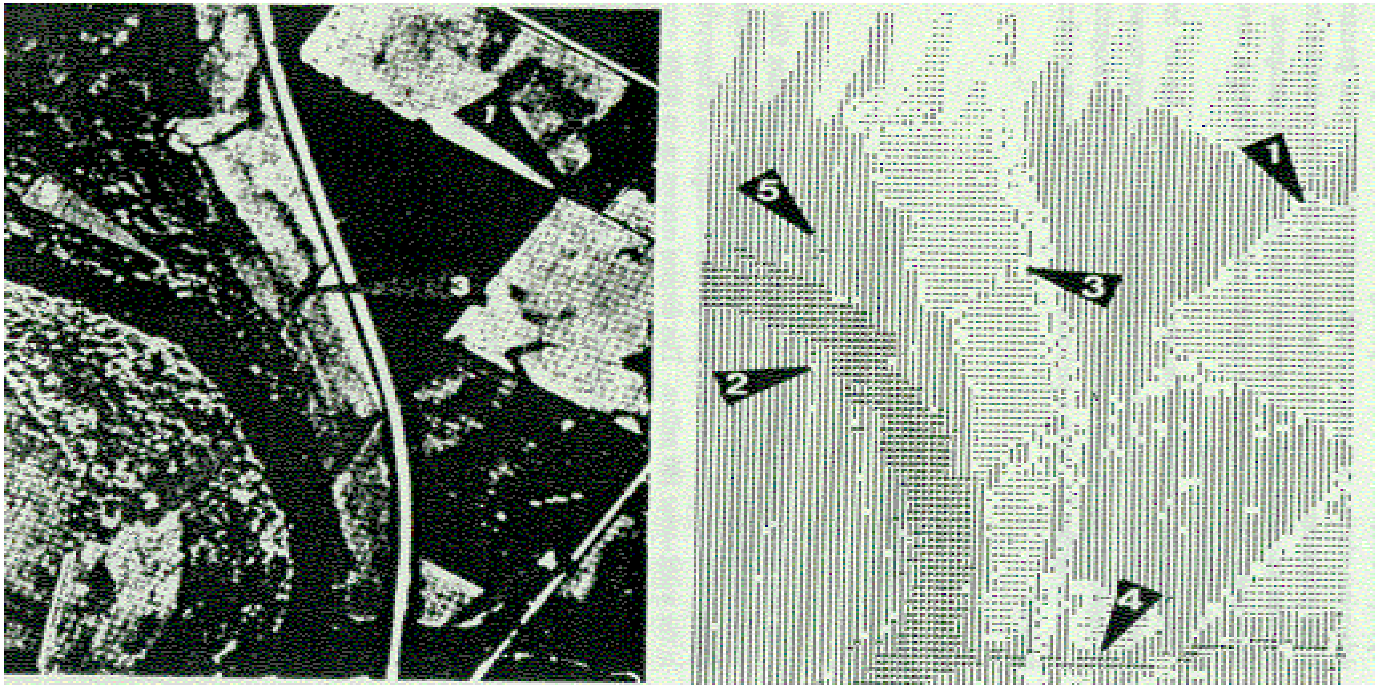
Optical Character Recognition



Detection of Oil Slicks

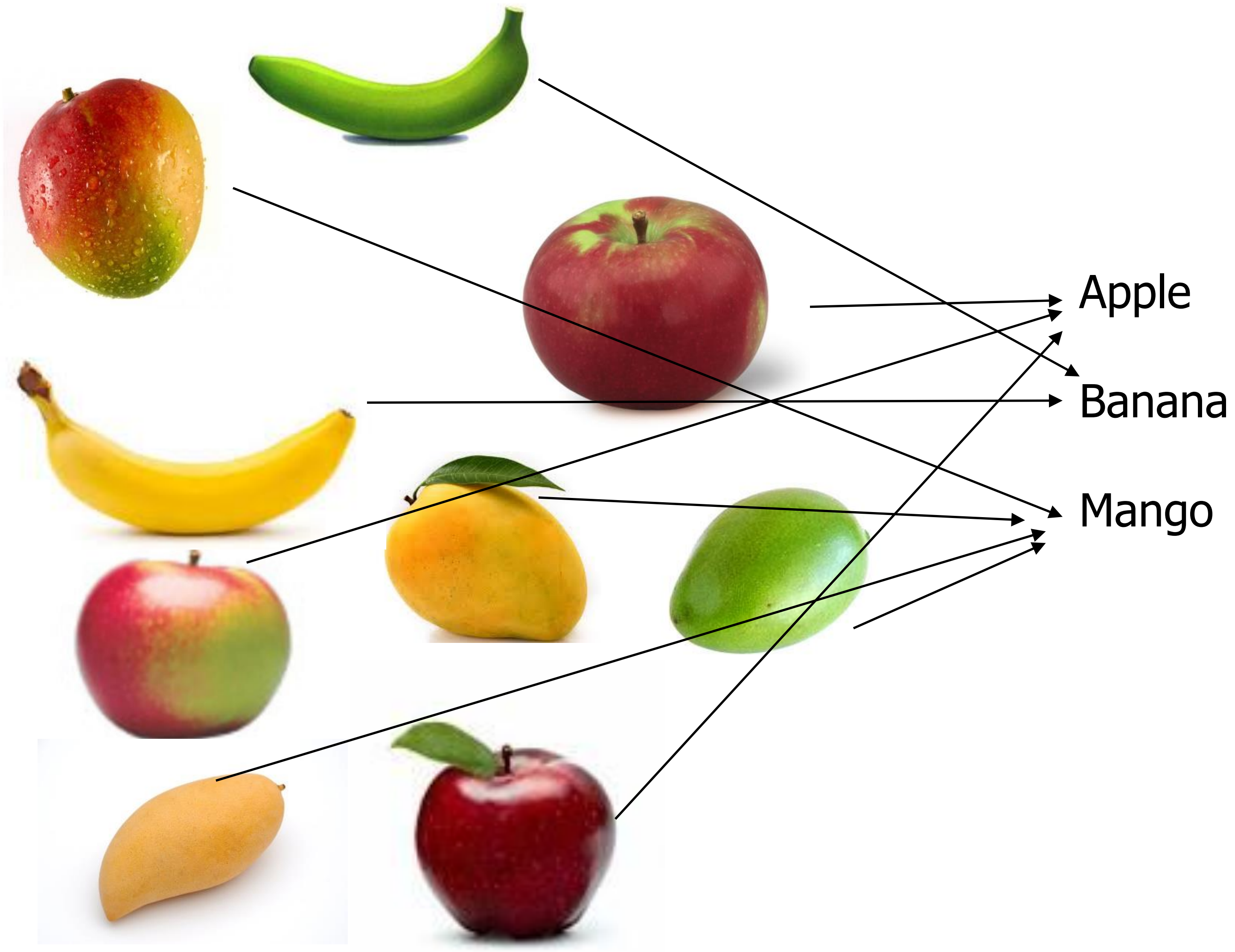
- Given radar satellite images of coastal waters

  Problem: **Detect Oil Slicks**

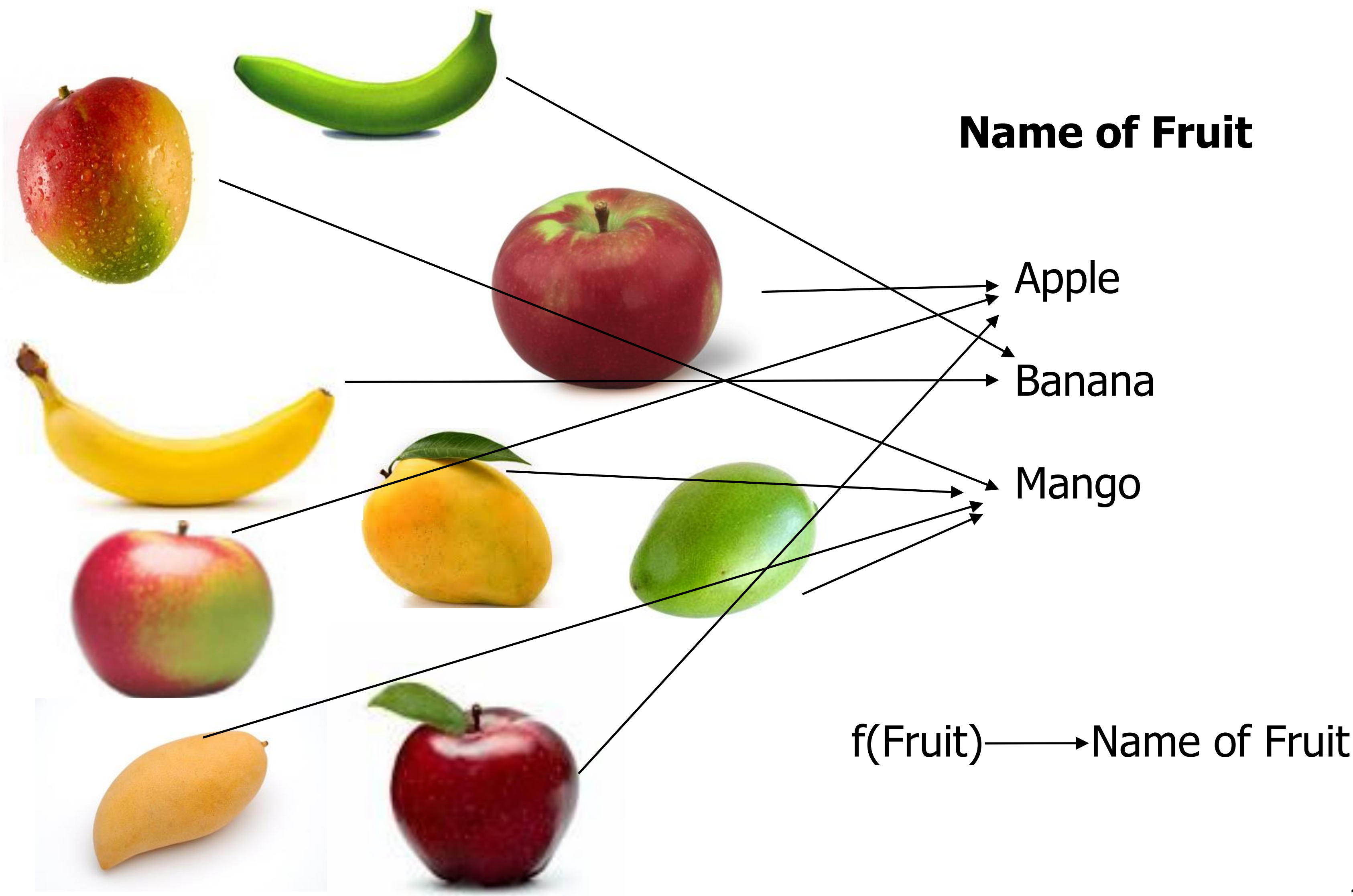


2

A B


Apple
Banana
Mango

Fruit

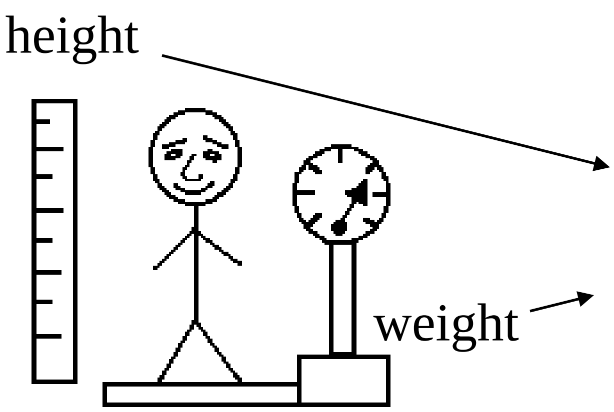


Classification

f(**Feature_vec**) $\longrightarrow$ **Fruit_type**

Feature Vector		**Fruit_type**
Color	**Shape**	
Red	Elliptical	Apple
Yellow	Elongated	Banana
Yellow	Elliptical	Mango
Green	Elliptical	Mango
Green	Elongated	Banana

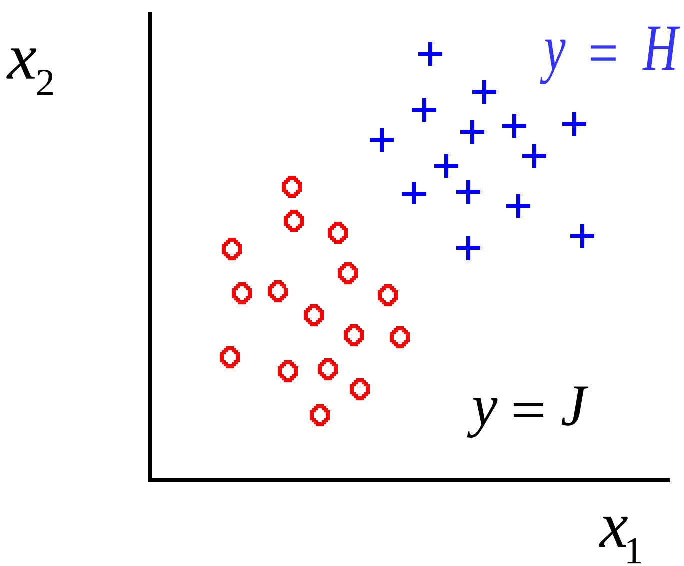
Classification



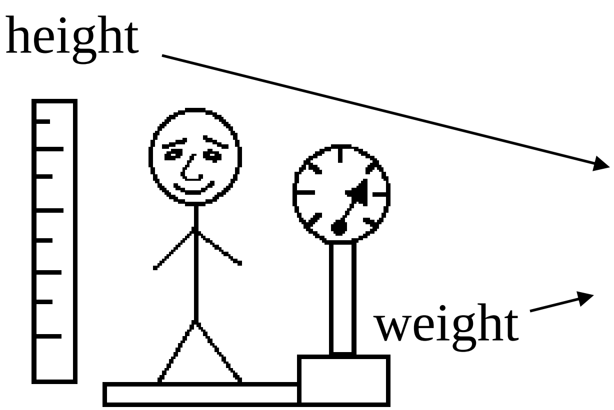
$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \mathbf{x}$$

$$X = \Re^2$$

Training examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$



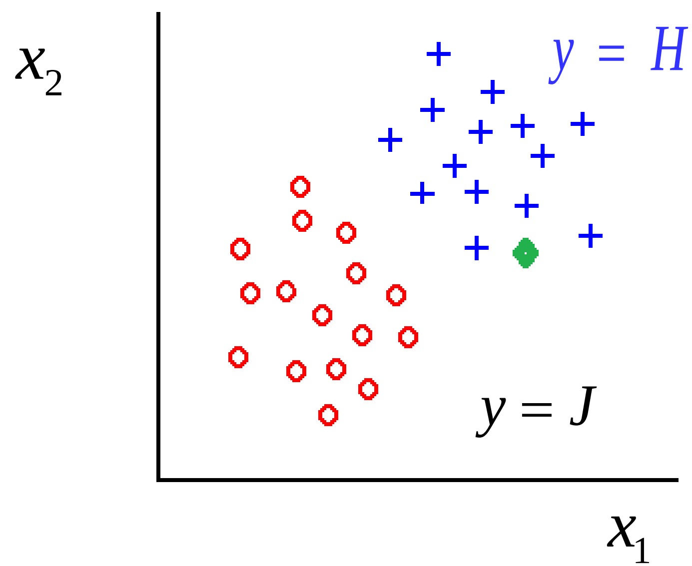
Classification



$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \mathbf{x}$$

$$X = \Re^2$$

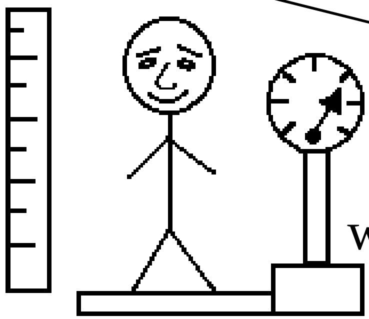
Training examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$



$D_i =$ distance($\mathbf{x}$, $\mathbf{x}_i$), $i=1,2,\ldots,l$

Classification

height



$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \mathbf{x}$$

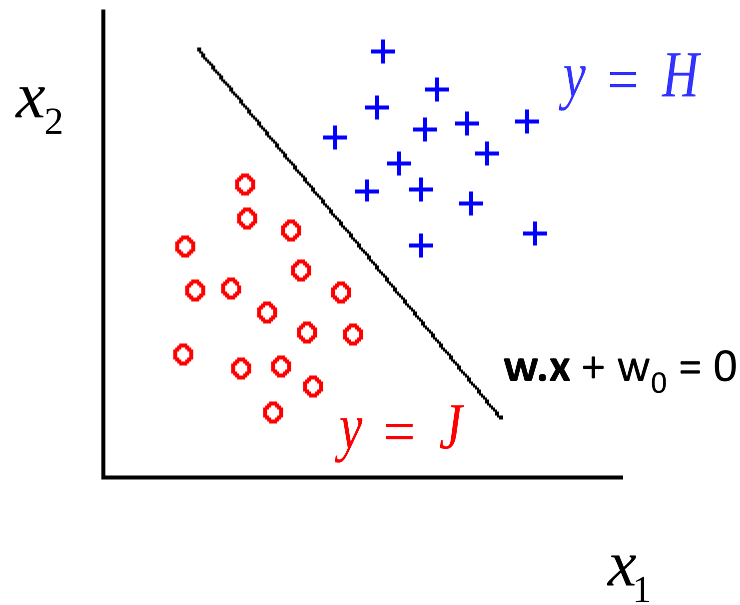
weight

$$X = \Re^2$$

Training examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$

Linear classifier:

$$f(\mathbf{x}) = \begin{cases} H & if \quad (\mathbf{w} \cdot \mathbf{x}) + w_0 \geq 0 \\ J & if \quad (\mathbf{w} \cdot \mathbf{x}) + w_0 < 0 \end{cases}$$



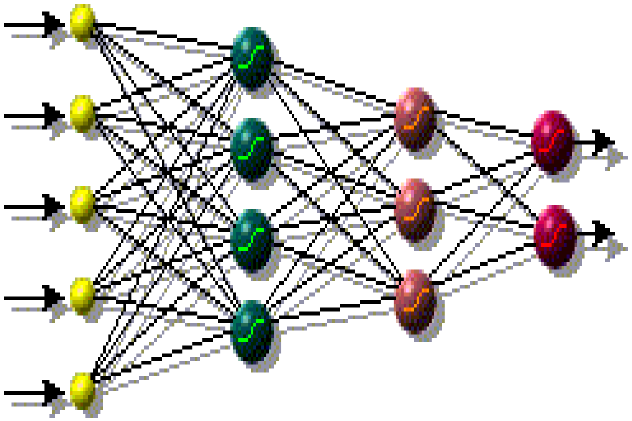
Classification: Definition

- Given a collection of records (*training set*)
  - Each record contains a set of *attributes*, one of the attributes is the *class* label.

- Find a *model* for class attribute as a function $q : X \to Y$ of the values of other attributes.

- Goal: <u>previously unseen</u> records should be assigned a class as accurately as possible.

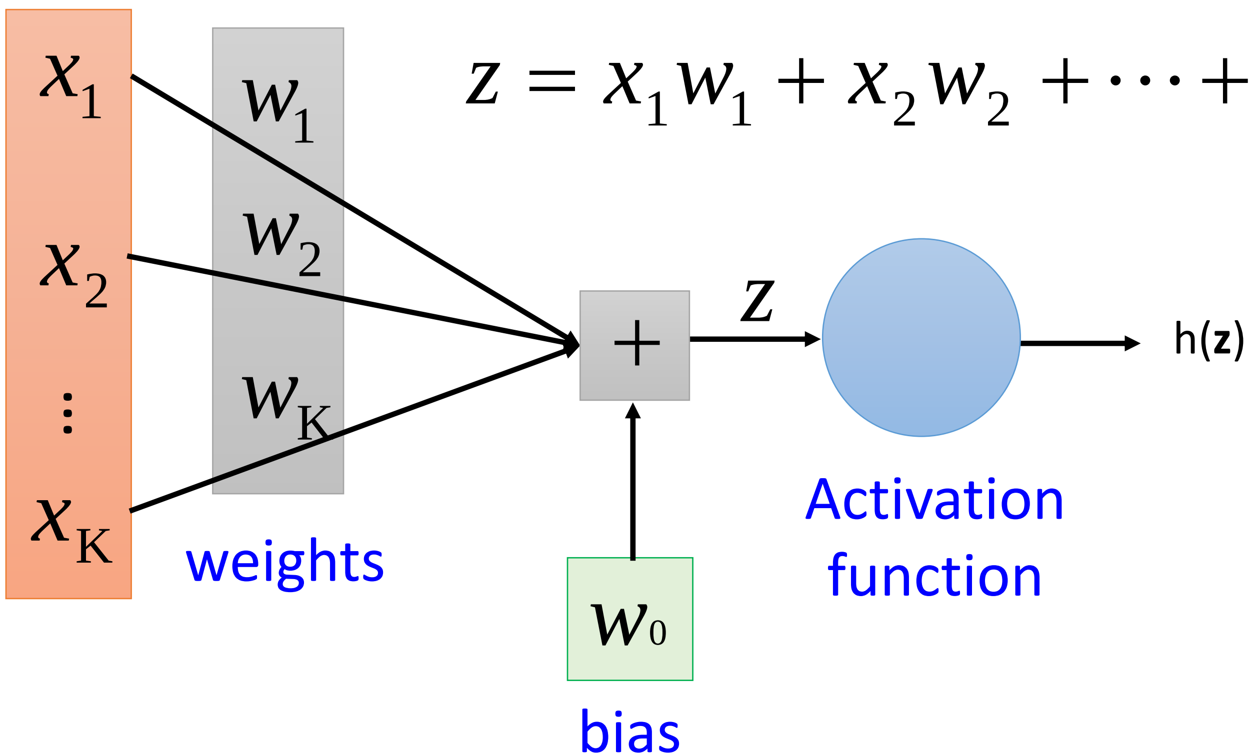
Neural Network



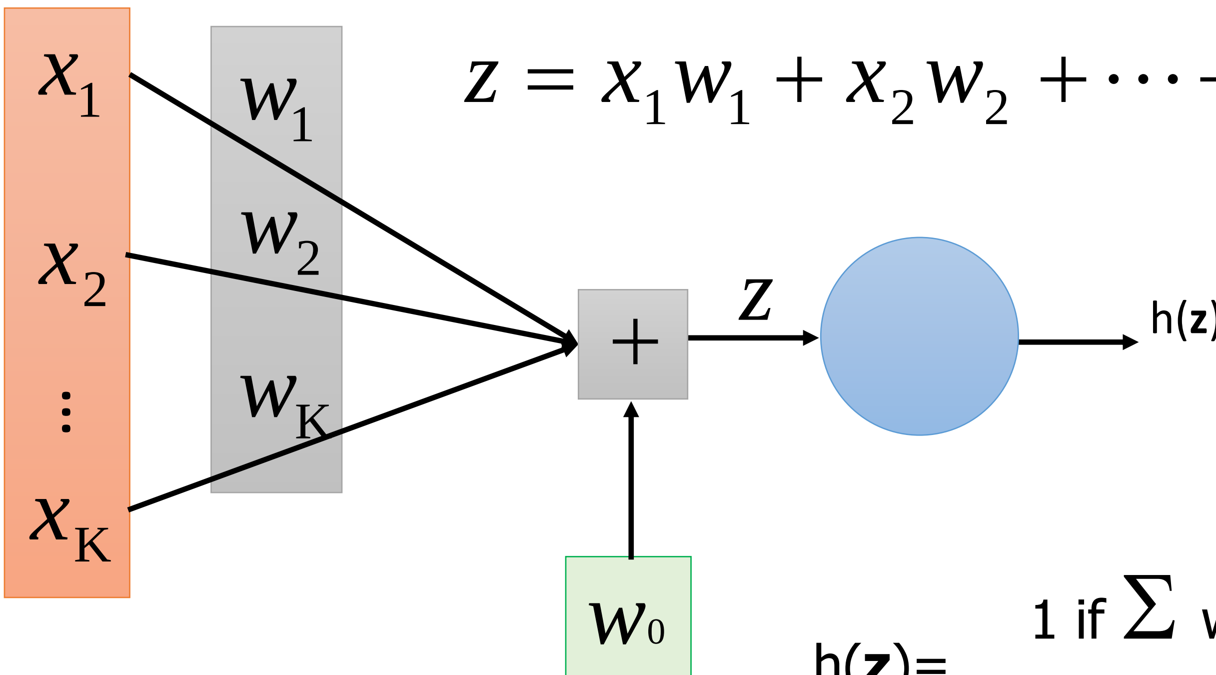
Elements of Neural Network

<u>Neuron</u> $f: R^K \rightarrow R$

$$z = x_1 w_1 + x_2 w_2 + \cdots + x_K w_K + w_0$$



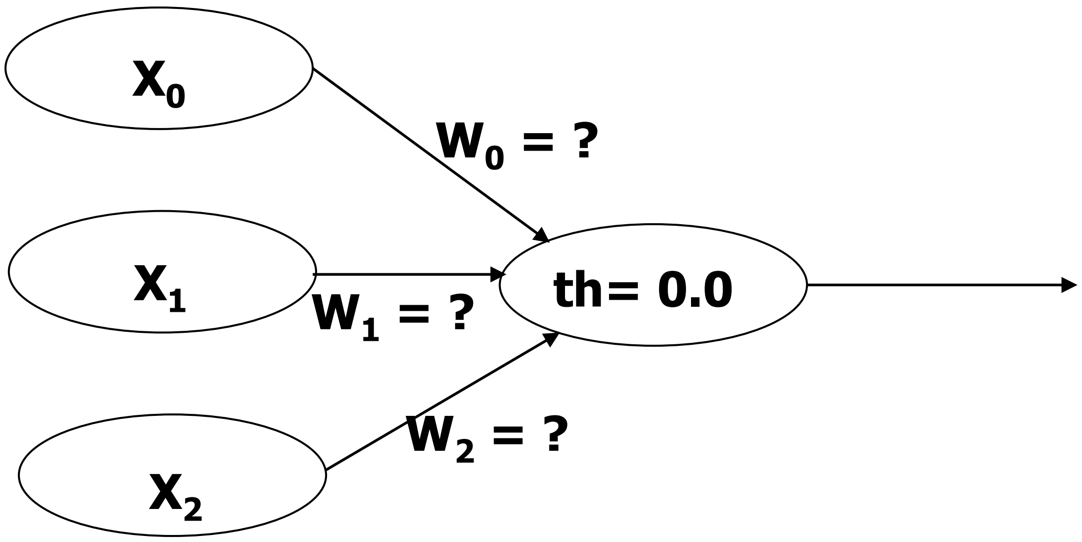
Single Perceptron



$$z = x_1 w_1 + x_2 w_2 + \cdots + x_K w_K + w_0$$

$$h(\mathbf{z}) = \begin{cases} 1 \text{ if } \sum w_i x_i > 0 \\ 0 \quad \text{otherwise} \end{cases}$$

Training Perceptrons

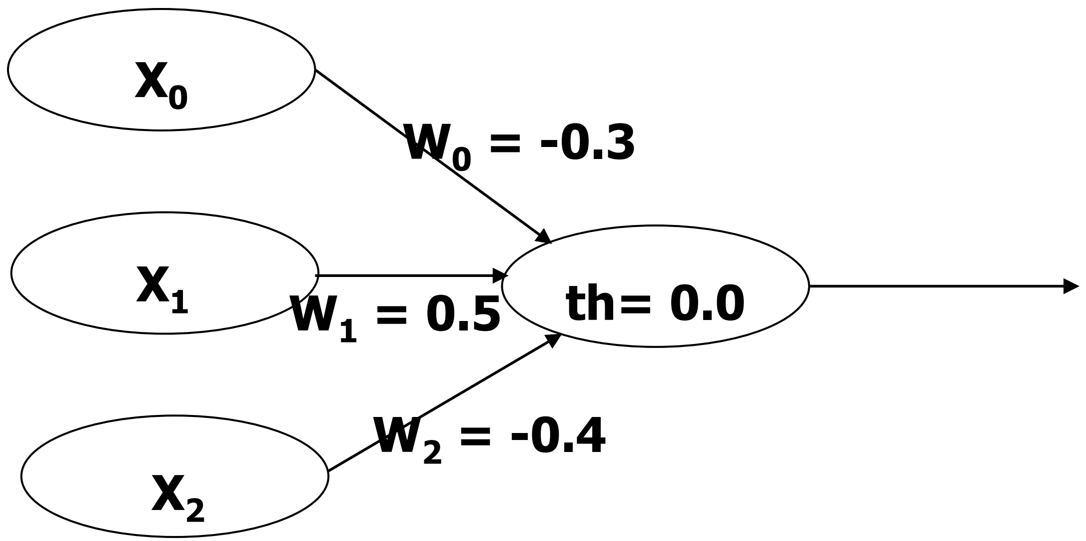


For AND

X_1	X_2	y
0	0	0
0	1	0
1	0	0
1	1	1

- Initialize with random weight values

Training Perceptrons



For AND

X_1	X_2	y
0	0	0
0	1	0
1	0	0
1	1	1

X_0	X_1	X_2	Summation	Output
1	0	0	(-1*0.3) + (0*0.5) + (0*-0.4) = -0.3	0
1	0	1	(-1*0.3) + (0*0.5) + (1*-0.4) = -0.7	0
1	1	0	(-1*0.3) + (1*0.5) + (0*-0.4) = 0.2	1
1	1	1	(-1*0.3) + (1*0.5) + (1*-0.4) = -0.2	0

Gradient Descent Learning Rule

- Train the w_i's such that they minimize the squared error
  - $E[w_1,\ldots,w_n] = \frac{1}{2} \sum_{d \in D} (y_d - h_d)^2$

  where D is the set of training examples

Gradient Descent

Gradient:
$\nabla E[w]=[\partial E/\partial w_0,\ldots\ \partial E/\partial w_n]$

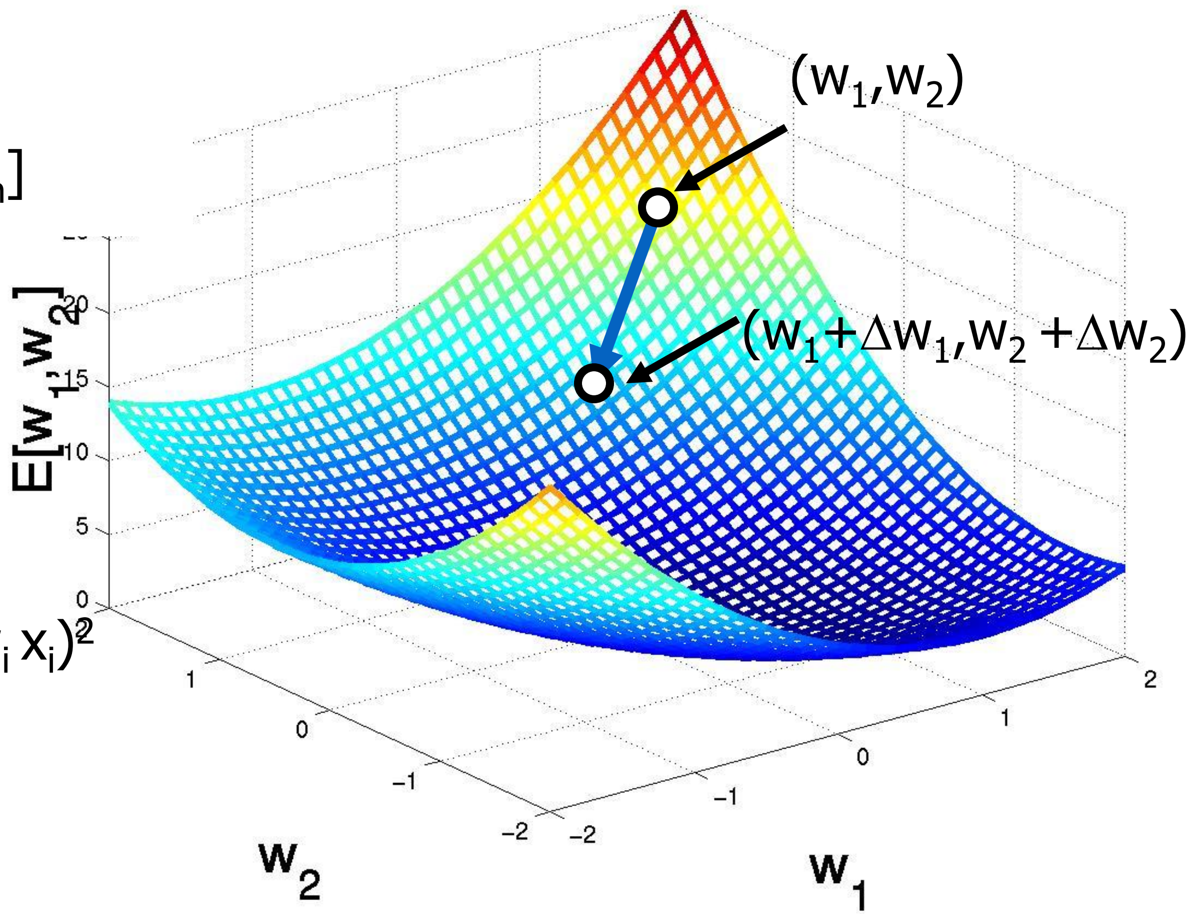
$\Delta w = -\ \nabla E[w]$

$\Delta w_i = -\ \partial E/\partial w_i$

$= -\ \partial/\partial w_i\ 1/2 \sum_d (y_d - h_d)^2$

$= -\ \partial/\partial w_i\ 1/2 \sum_d (y_d - \sum_i w_i\, x_i)^2$

$= \sum_d (y_d - h_d)(x_i)$



Gradient Descent

Gradient-Descent(*training_examples*, η)

Each training example is a pair of the form $<(x_1,\ldots x_n),t>$ where $(x_1,\ldots,x_n)$ is the vector of input values, and t is the target output value

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero
  - For each $<(x_1,\ldots x_n),t>$ in *training_examples* Do
    - Input the instance $(x_1,\ldots,x_n)$ to the linear unit and compute the output o
    - For each linear unit weight w_i Do
      - $\Delta w_i = \Delta w_i + \sum_d (y_d - h_d)\, x_i$

Weight Updation

- W_0= -0.3 + [(0-0)1+(0-0)1+(0-1)1+(1-0)1]= -0.3
- W_1= 0.5 + [(0-0)0+(0-0)0+(0-1)1+(1-0)1]= 0.5
- W_2= -0.4 + [(0-0)0+(0-0)1+(0-1)0+(1-0)1]= 0.6

X_0	X_1	X_2	Summation	Output
1	0	0	(-1*0.3) + (0*0.5) + (0*0.6) = -0.3	0
1	0	1	(-1*0.3) + (0*0.5) + (1*0.6) = 0.3	1
1	1	0	(-1*0.3) + (1*0.5) + (0*0.6) = 0.2	1
1	1	1	(-1*0.3) + (1*0.5) + (1*0.6) = 0.8	1

Weight Updation

- W_0= -0.3 + [(0-0)1+(0-1)1+(0-1)1+(1-1)1]= -2.3
- W_1= 0.5 + [(0-0)0+(0-1)0+(0-1)1+(1-1)1]= -0.5
- W_2= 0.6 + [(0-0)0+(0-1)1+(0-1)0+(1-1)1]= -0.4

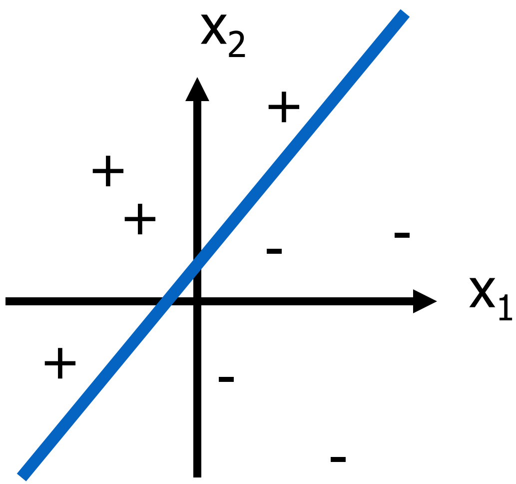
X_0	X_1	X_2	Summation	Output
1	0	0	(-1*2.3) + (-0*0.5) + (-0*0.4) = -2.3	0
1	0	1	(-1*2.3) + (-0*0.5) + (-1*0.4) = -2.7	0
1	1	0	(-1*2.3) + (-1*0.5) + (-0*0.4) = -2.8	0
1	1	1	(-1*2.3) + (-1*0.5) + (-1*0.4) = -3.2	0

Weight Updation

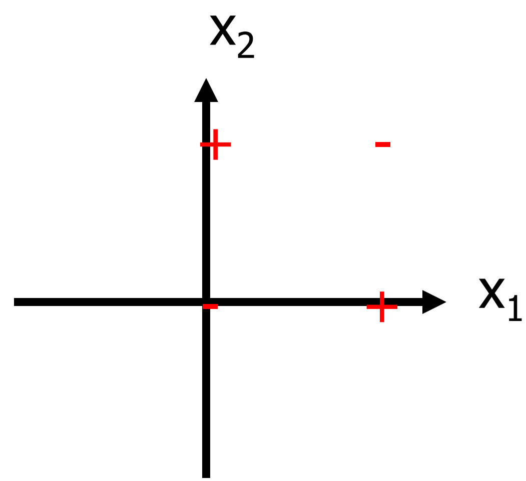
- W_0= -3.3 + [(0-0)1+(0-0)1+(0-0)1+(1-0)1]= -2.3
- W_1= 0.5 + [(0-0)0+(0-0)0+(0-0)1+(1-0)1]= 1.5
- W_2= 0.6 + [(0-0)0+(0-0)1+(0-0)0+(1-0)1]= 1.6

X_0	X_1	X_2	Summation	Output
1	0	0	(-1*2.3) + (0*1.5) + (0*1.6) = -2.3	0
1	0	1	(-1*2.3) + (0*1.5) + (1*1.6) = -0.7	0
1	1	0	(-1*2.3) + (1*1.5) + (0*1.6) = -0.8	0
1	1	1	(-1*2.3) + (1*1.5) + (1*1.6) = 0.8	1

Decision Surface of a Perceptron



Linearly separable

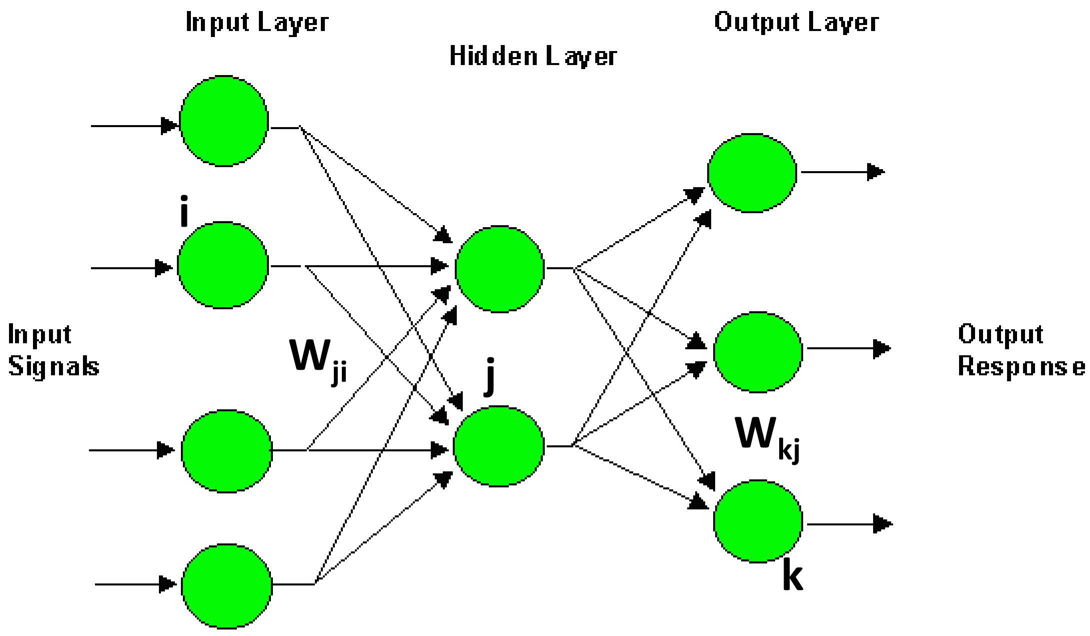


Non-Linearly separable

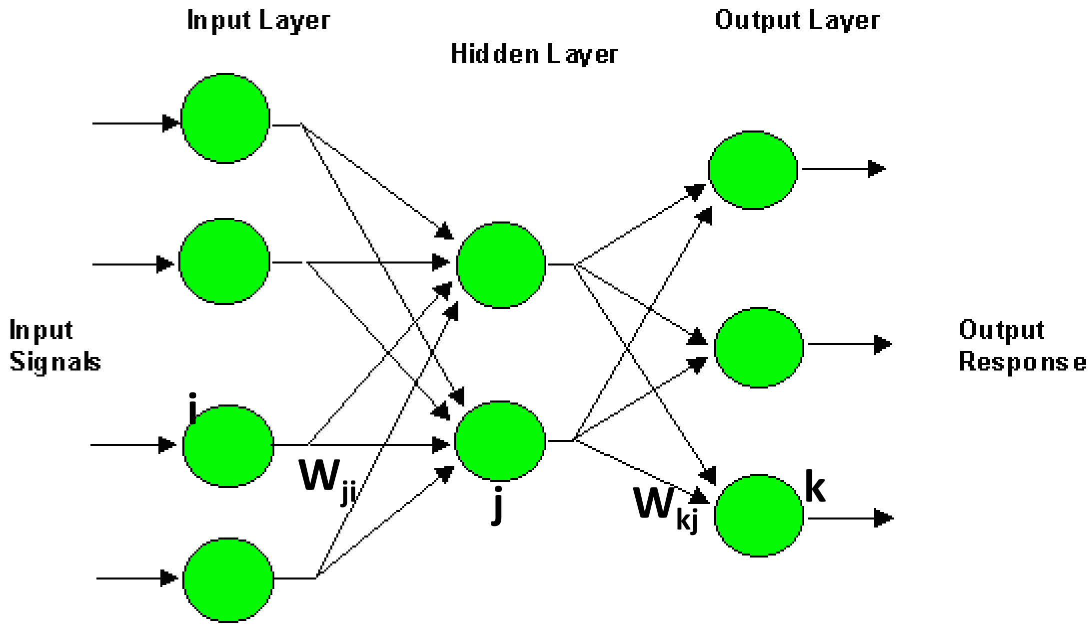
- But functions that are not linearly separable (e.g. XOR)
  XOR can solved as:

$XOR(x_1, x_2) = AND(\ OR(x_1, x_2), NAND(x_1, x_2))$

Multilayer Perceptron (MLP)



Multilayer Perceptron (MLP)



$$\mathbf{net}_j = \sum_{i=1}^{d} \mathbf{x}_i \mathbf{w}_{ji} + \mathbf{w}_{j0} = \sum_{i=0}^{d} \mathbf{x}_i \mathbf{w}_{ji} \equiv \mathbf{w}_j^t . \mathbf{x},$$

$$y_j = f(net_j)$$

$$net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} = w_k^t . y, \quad z_k = f(net_k)$$

Multilayer Perceptron (MLP)

$$J(w) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}\|t - z\|^2$$

$$\Delta w = -\eta \frac{\partial J}{\partial w}$$

$$\frac{\partial J}{\partial w_{ki}} = \frac{\partial J}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{ki}} = -\delta_k \frac{\partial net_k}{\partial w_{ki}}$$

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k} \cdot \frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k)$$

$$\frac{\partial net_k}{\partial w_{kj}} = y_j$$

$$\Delta w_{kj} = \eta \delta_k y_j = \eta (t_k - z_k) f'(net_k) y_j$$

Multilayer Perceptron (MLP)

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \cdot \frac{\partial y_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}}$$

$$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right] = -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j}$$

$$= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k} \cdot \frac{\partial net_k}{\partial y_j} = -\sum_{k=1}^{c}(t_k - z_k) f'(net_k) w_{kj}$$

$$\delta_j \equiv f'(net_j)\sum_{k=1}^{c} w_{kj}\delta_k$$

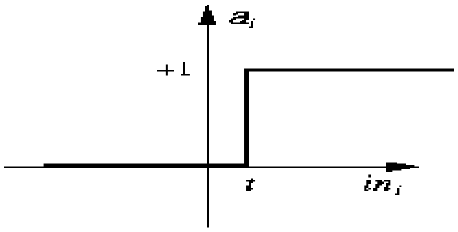
$$\Delta w_{ji} = \eta x_i \delta_j = \eta \underbrace{\left[\Sigma w_{kj}\delta_k\right] f'(net_j)}_{\delta_j} x_i$$

Types of Layers

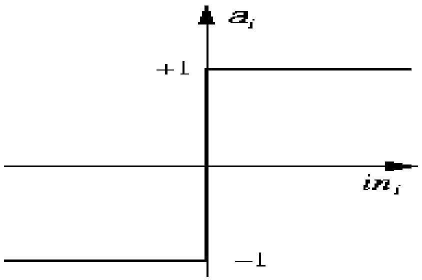
- The input layer.
  - Introduces input values into the network.
  - No activation function or other processing.
- The hidden layer(s).
  - Perform classification of features
  - Two hidden layers are sufficient to solve any problem
  - Features imply more layers may be better
- The output layer.
  - Functionally just like the hidden layers
  - Outputs are passed on to the world outside the neural network.

Activation functions

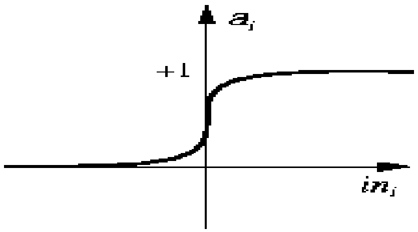
- Transforms neuron's input into output.



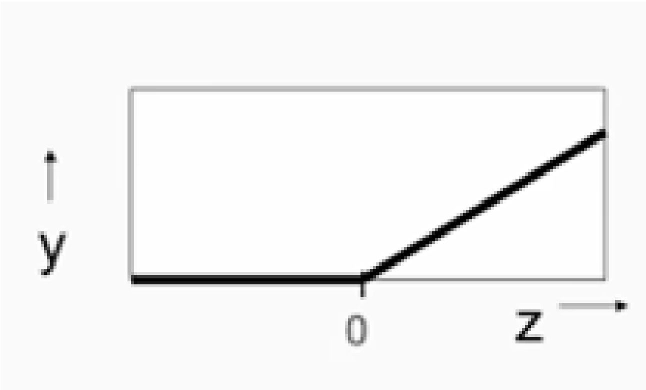
(a) Step function



(b) Sign function



(c) Sigmoid function



Rectified Linear Unit

Backpropagation Algorithm

Initialize **w** to some small random value

Do

- For each training example <$(x_1,\ldots x_n),t$> Do
  - compute the network outputs o_k
  - For each output unit k, compute $\delta_k=o_k(1-o_k)(t_k-o_k)$
  - For each hidden unit j, $\delta_j=o_j(1-o_j) \sum_k w_{kj}\, \delta_k$
  - Compute $w_{ji}=w_{ji}+\Delta w_{ji}$ where $\Delta w_{ji}= \eta\, \delta_j\, x_i$
  - Compute $w_{kj}=w_{kj}+\Delta w_{kj}$ where $\Delta w_{kj}= \eta \delta_k y_j$

Until the termination condition is met.

Return **w**

Universal Function Approximator

A one hidden layer FFNN with sufficiently large number of hidden nodes can approximate any function (Hornik, 1991)

Handwritten Character Recognition

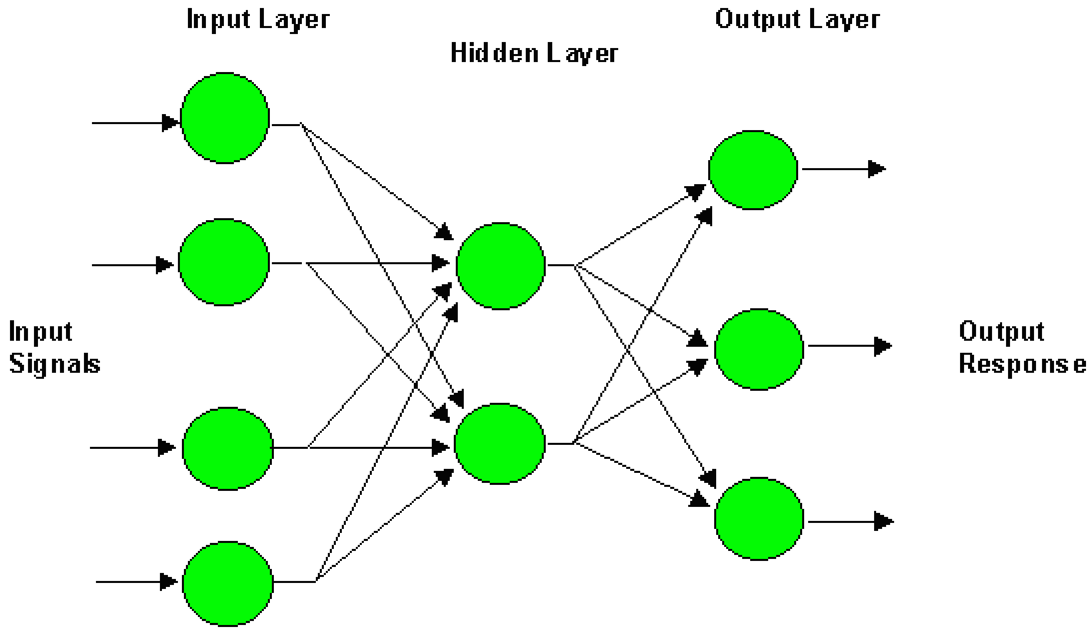


Image size= 100 x 100
No. of nodes at hidden layer= 10^6

No. of Classes =26
No. of Weights to be learned= 10^{10}

Shallow vs Deep

- Functions that can be compactly represented by a depth k architecture with fewer computational elements might require an larger number of computational elements to be represented by a depth k − 1 architecture.

Consequences are:

- Computational: We don't need exponentially many elements in the layers
- Statistical: poor generalization may be expected when using an insufficiently deep architecture for representing some functions

10 BREAKTHROUGH TECHNOLOGIES 2013

MIT Technology Review

Introduction | The 10 Technologies | Past Years

Deep Learning

With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart. →

Temporary Social Media

Messages that quickly self-destruct could enhance the privacy of online communications and make people freer to be spontaneous. →

Prenatal DNA Sequencing

Reading the DNA of fetuses will be the next frontier of the genomic revolution. But do you really want to know about the genetic problems or musical aptitude of your unborn child? →

Additive Manufacturing

Skeptical about 3-D printing? GE, the world's largest manufacturer, is on the verge of using the technology to make jet parts. →

Baxter: The Blue-Collar Robot

Rodney Brooks's newest creation is easy to interact with, but the complex innovations behind the robot show just how hard it is to get along with people. →

Memory Implants

A maverick neuroscientist believes he has deciphered the code by which the brain forms long-term memories. Next: testing a prosthetic implant for people suffering from long-term memory loss. →

Smart Watches

The designers of the Pebble watch realized that a mobile phone is more useful if you don't have to take it out of your pocket. →

Ultra-Efficient Solar Power

Doubling the efficiency of a solar cell would completely change the economics of renewable energy. Nanotechnology just might make it possible. →

Big Data from Cheap Phones

Collecting and analyzing information from simple cell phones can provide surprising insights into how people move about and behave – and even help us understand the spread of diseases. →

Supergrids

A new high-power circuit breaker could finally make highly efficient DC power grids practical. →

Deep Learning

- Multilayer neural networks have been around for 25 years. What's actually new?

- We had good algorithms for learning the weights in networks with 1 or 2 hidden layer(s)

- But these algorithms are not good at learning the weights for networks with more hidden layers

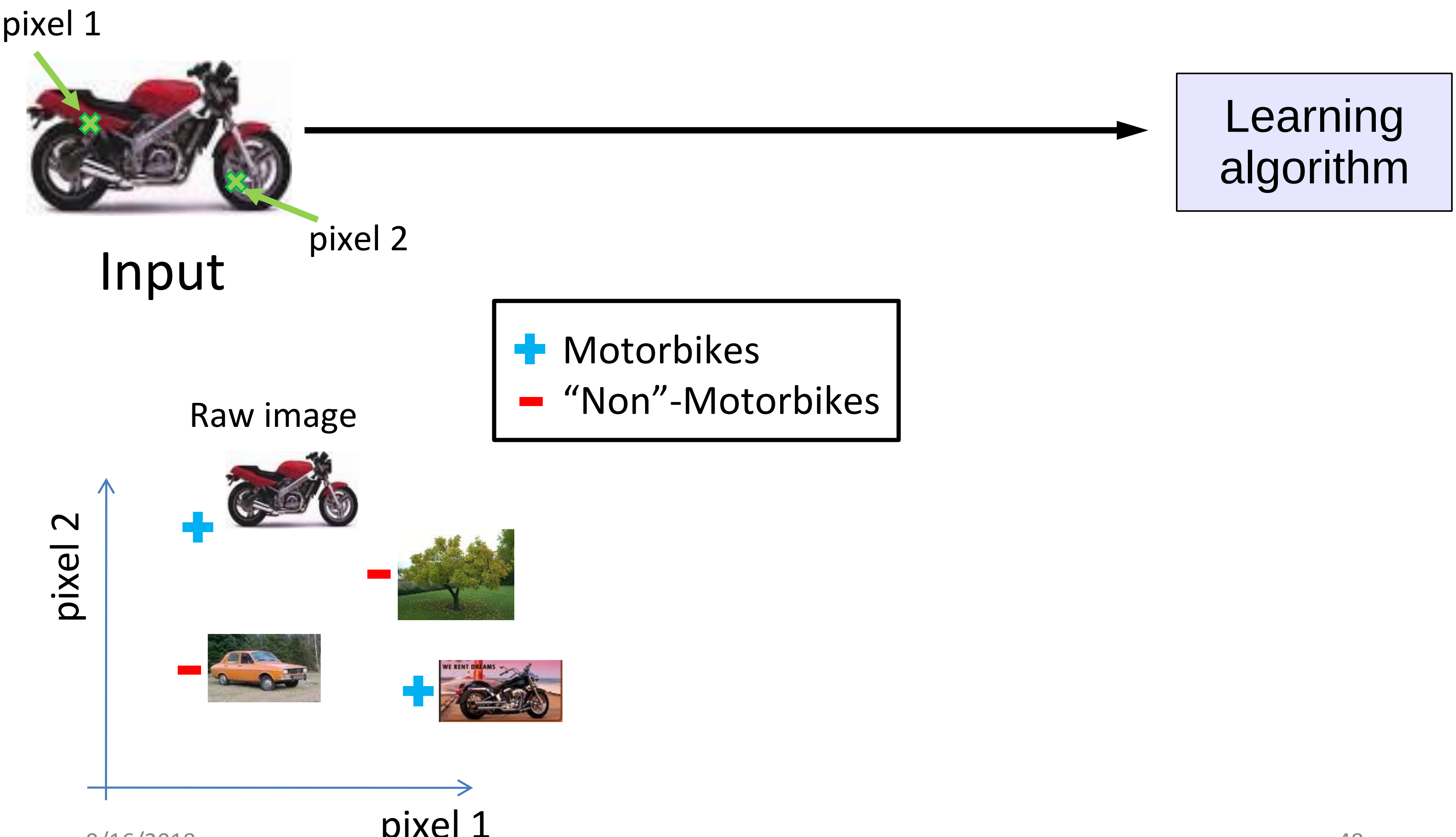
Why is this hard?

You see this:

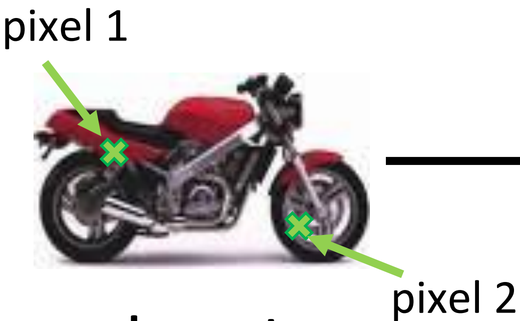
But the camera sees this:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

Pixel-based representation



Pixel-based representation

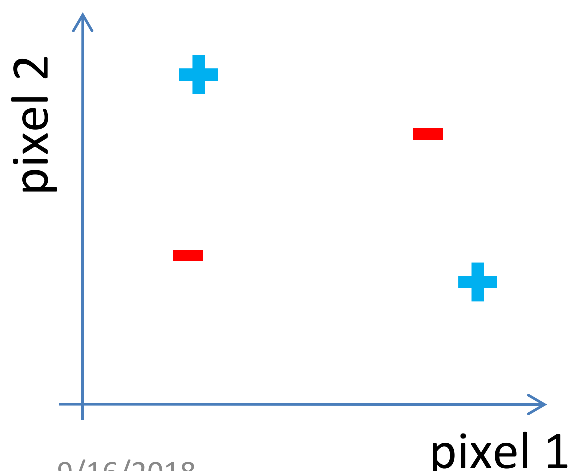


Input

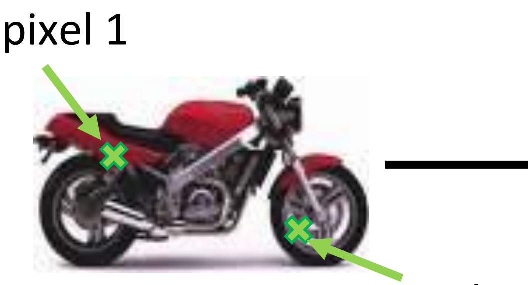
Learning algorithm

✚ Motorbikes
▬ “Non”-Motorbikes

Raw image



Pixel-based representation



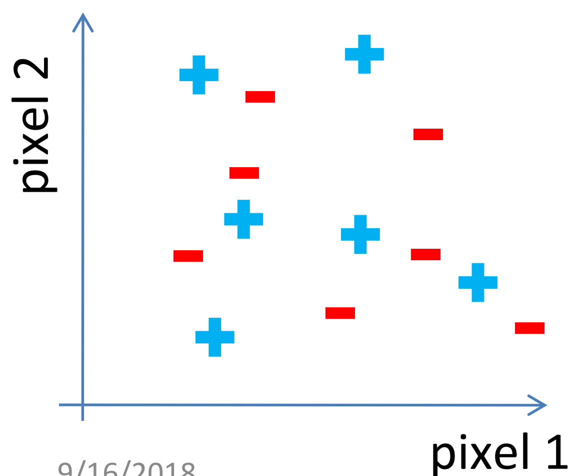
Input

Learning algorithm

Motorbikes

"Non"-Motorbikes

Raw image



What we want

Input

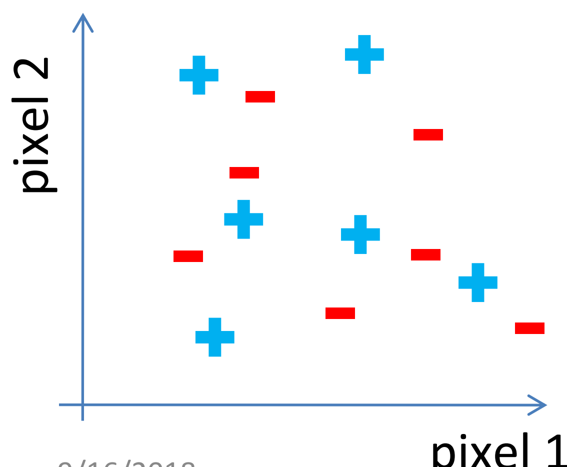
Feature representation

Learning algorithm

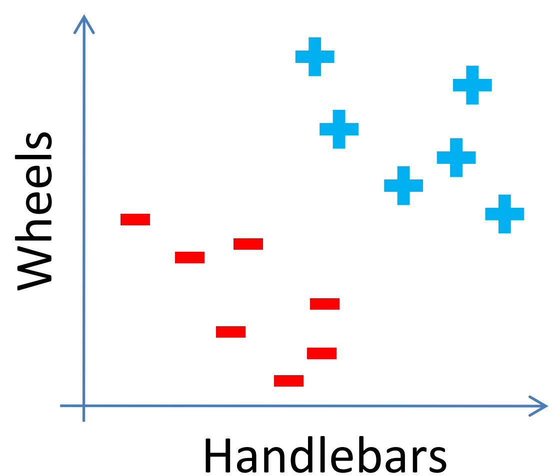
E.g., Does it have Handlebars? Wheels?

+ Motorbikes
– “Non”-Motorbikes

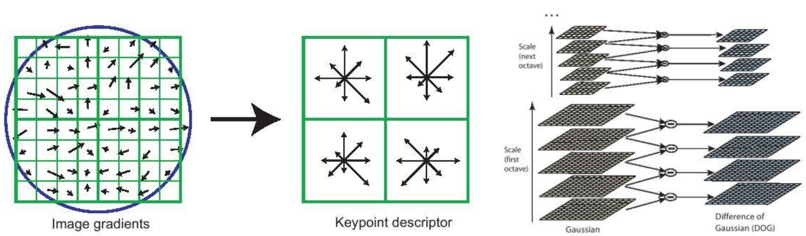
Raw image



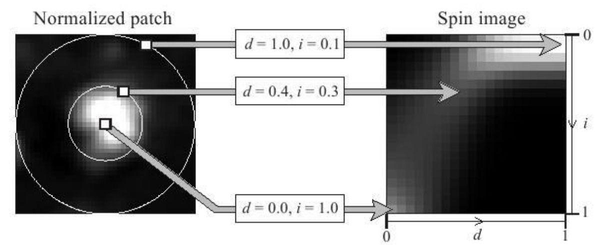
Features



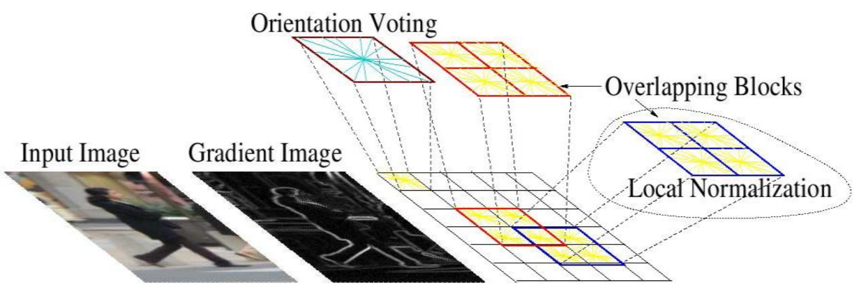
Some feature representations



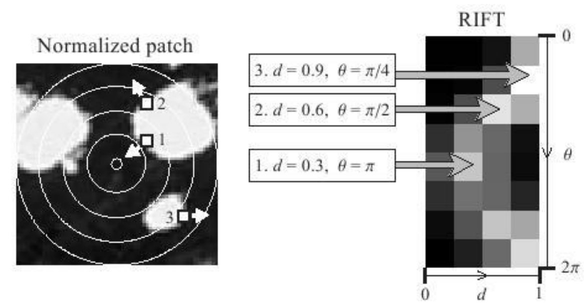
SIFT



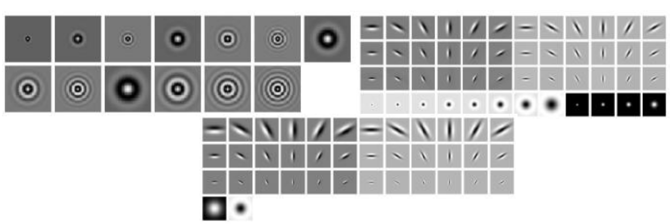
Spin image



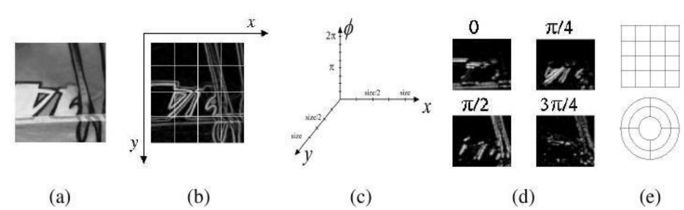
HoG



RIFT

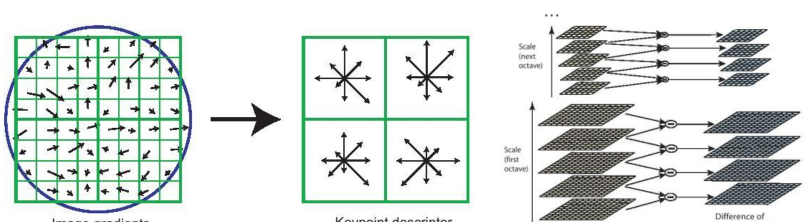


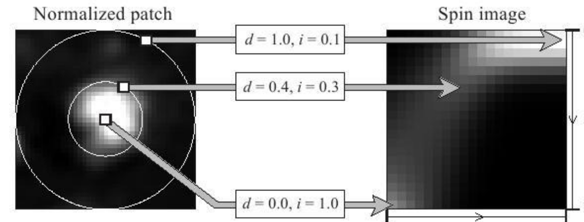
Textons



GLOH

Some feature representations

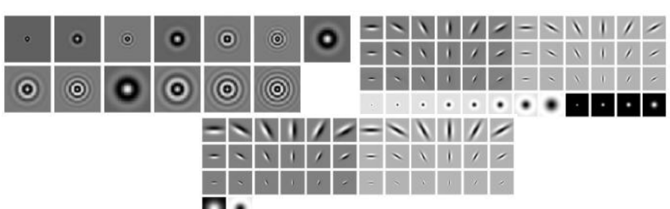


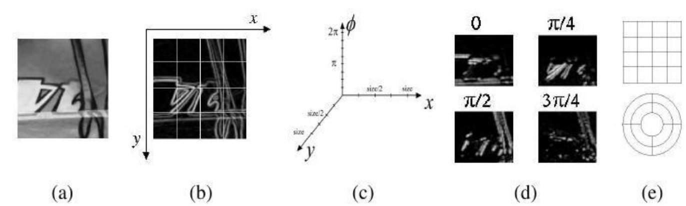


Coming up with features is often difficult, time-consuming, and requires expert knowledge.

HoG

RIFT

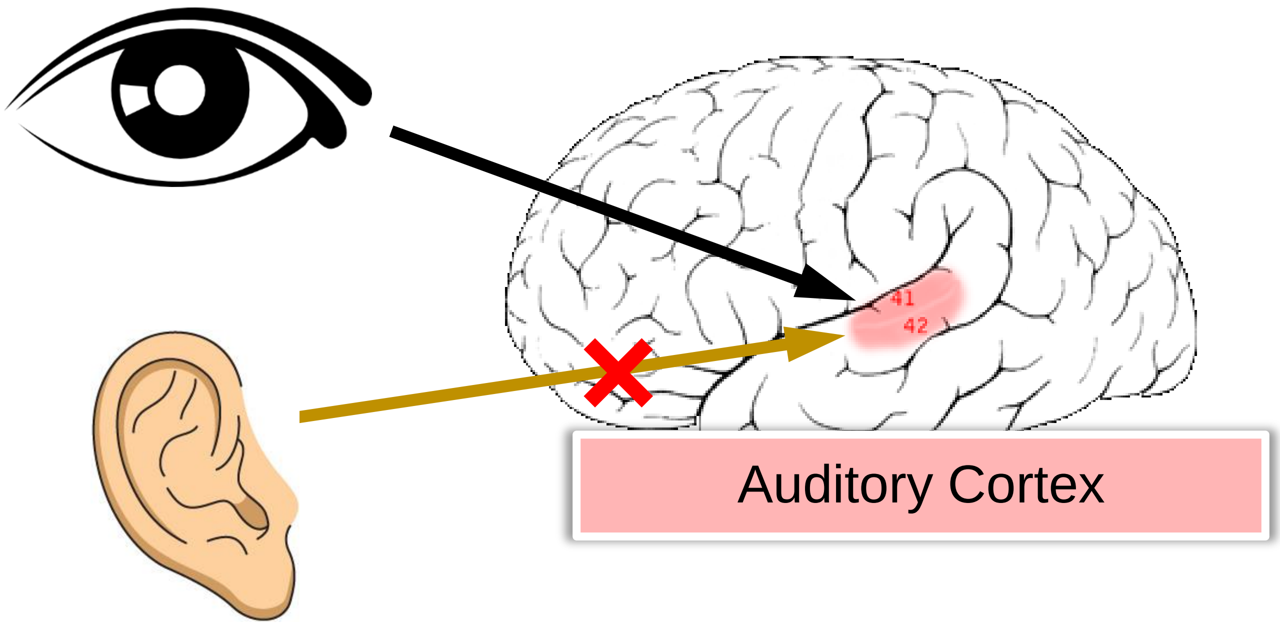




Textons

GLOH

The brain: potential motivation for deep learning

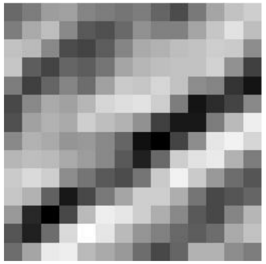


Auditory cortex learns to see!

[Roe et al., 1992]

Feature learning problem

- Given a 14x14 image patch x, can represent it using 196 real numbers.



$$\begin{bmatrix} 255 \\ 98 \\ 93 \\ 87 \\ 89 \\ 91 \\ 48 \\ \dots \end{bmatrix}$$

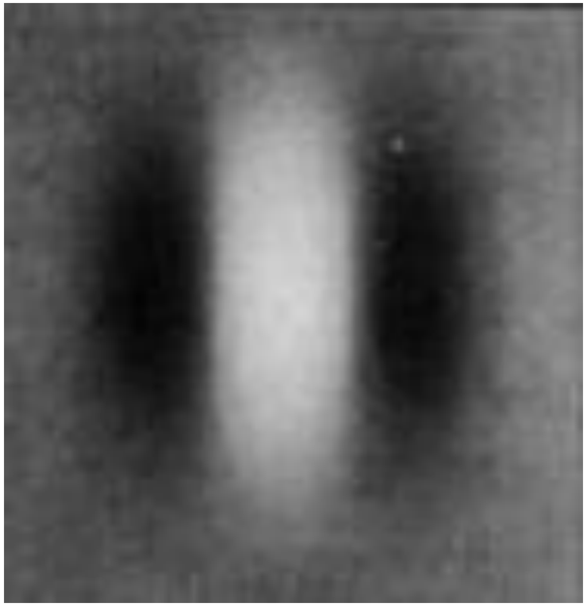
- Problem: Can we find a learn a better feature vector to represent this?

First stage of visual processing: V1

V1 is the first stage of visual processing in the brain.

Neurons in V1 typically modeled as edge detectors:

Neuron #1 of visual cortex (model)



Neuron #2 of visual cortex (model)

Learning sensor representations

Sparse coding (Olshausen & Field,1996)

Input: Images $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$ (each in $R^{n \times n}$)

Learn: Dictionary of bases $\phi_1, \phi_2, \ldots, \phi_k$ (also $R^{n \times n}$), so that each input x can be approximately decomposed as:

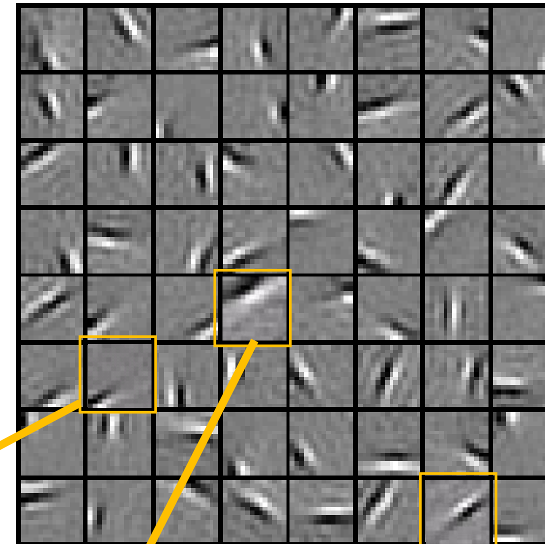
$$x \approx \sum_{j=1}^{k} a_j \phi_j$$

s.t. a_j's are mostly zero ("sparse")

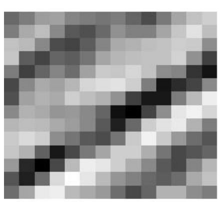
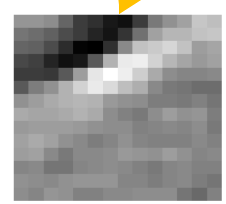
Sparse coding illustration

Natural Images

Learned bases ($\phi_1, \ldots, \phi_{64}$): "Edges"



Test example

 $\approx 0.8 *$  $+ 0.3 *$  $+ 0.5 *$ 

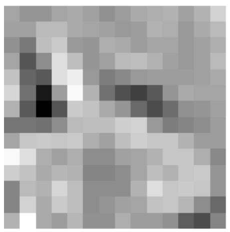
$$x \approx 0.8 * \phi_{36} + 0.3 * \phi_{42} + 0.5 * \phi_{63}$$

$[a_1, \ldots, a_{64}] = [0, 0, \ldots, 0, \mathbf{0.8}, 0, \ldots, 0, \mathbf{0.3}, 0, \ldots, 0, \mathbf{0.5}, 0]$

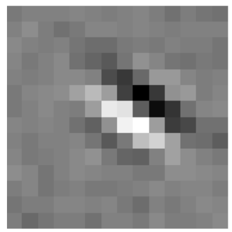
(feature representation)

Sparse coding illustration

Represent as: [a_{15}=0.6, a_{28}=0.8, a_{37} = 0.4]

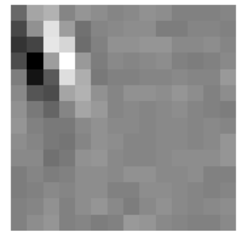


0.6 *



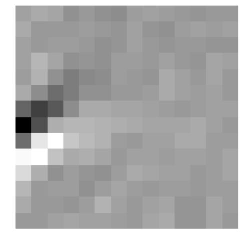
ϕ_{15}

+ 0.8 *



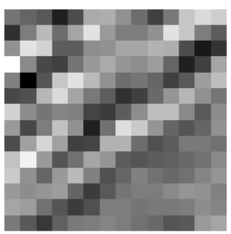
ϕ_{28}

+ 0.4 *

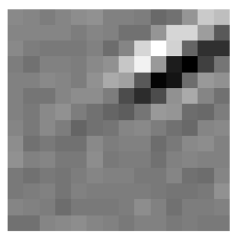


ϕ_{37}

Represent as: [a_{5}=1.3, a_{18}=0.9, a_{29} = 0.3]

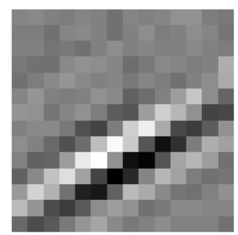


1.3 *



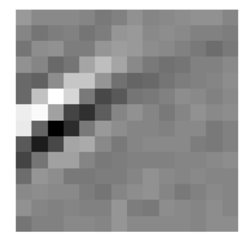
ϕ_{5}

+ 0.9 *



ϕ_{18}

+ 0.3 *



ϕ_{29}

- **Method “invents” edge detection**
- Automatically learns to represent an image in terms of the edges that appear in it. Gives a more succinct, higher-level representation than the raw pixels.
- Quantitatively similar to primary visual cortex (area V1) in brain.

Feature detectors

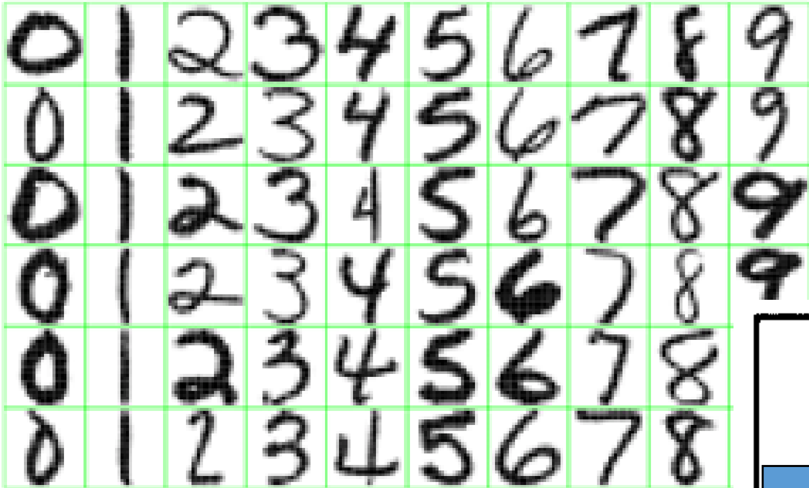
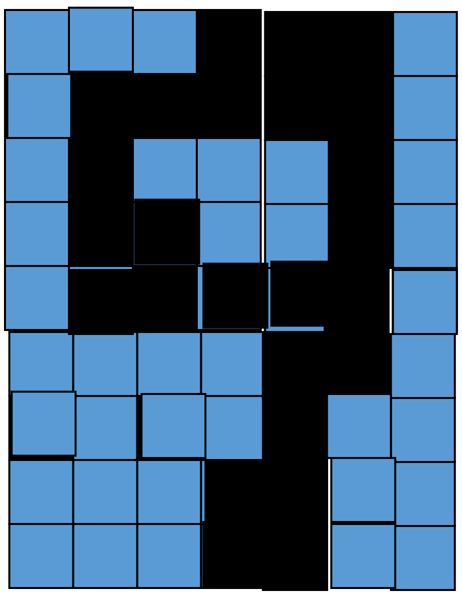
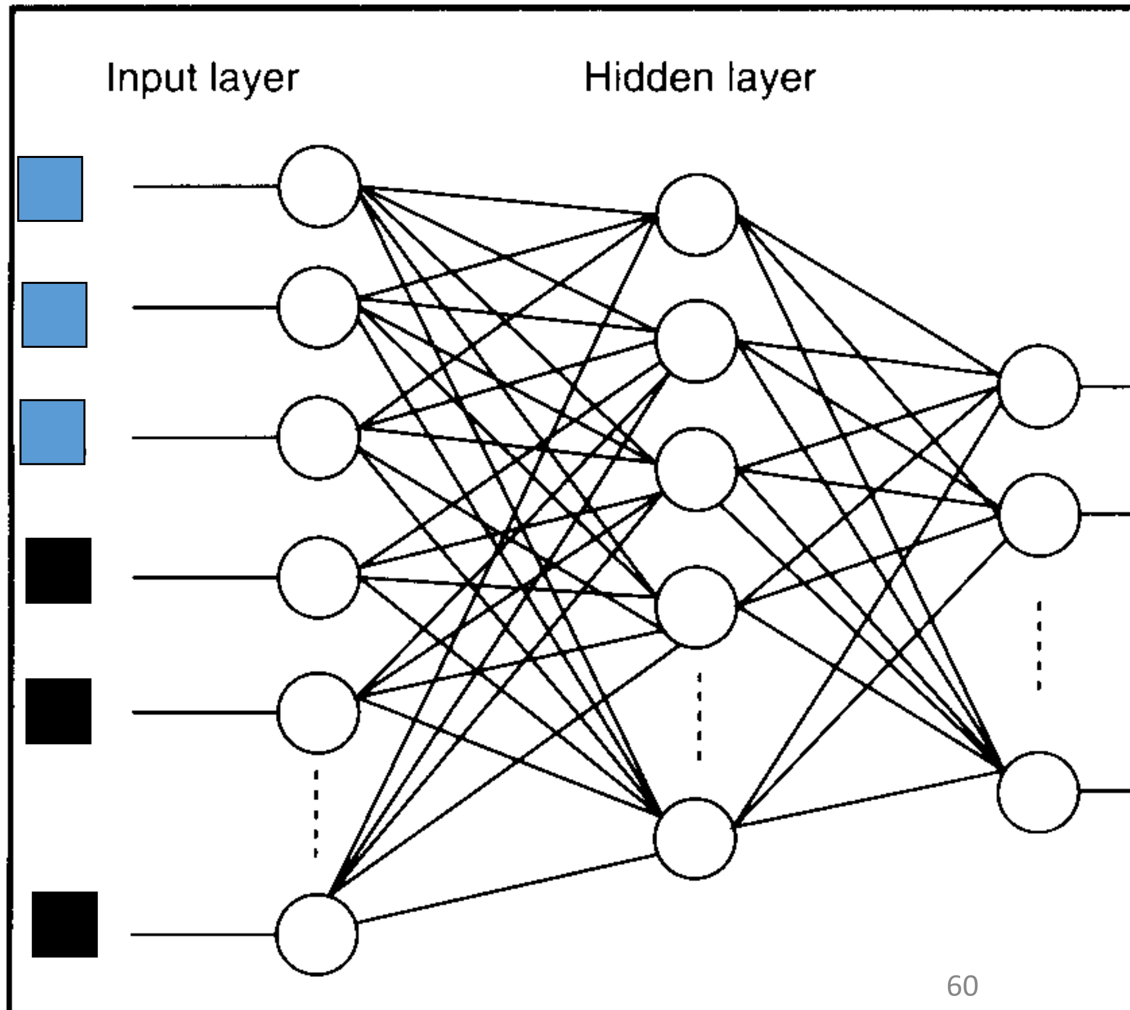


Figure 1.2: *Examples of handwritten digits from postal envelopes.*





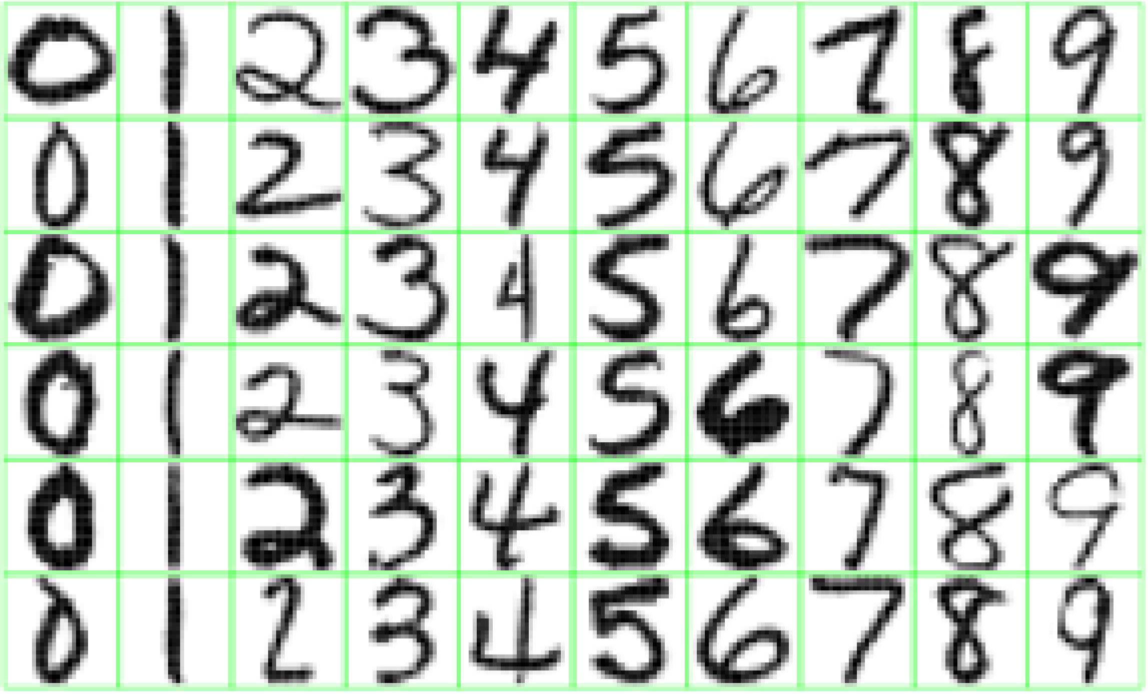


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

What features might you expect a good NN to learn, when trained with data like this?

1

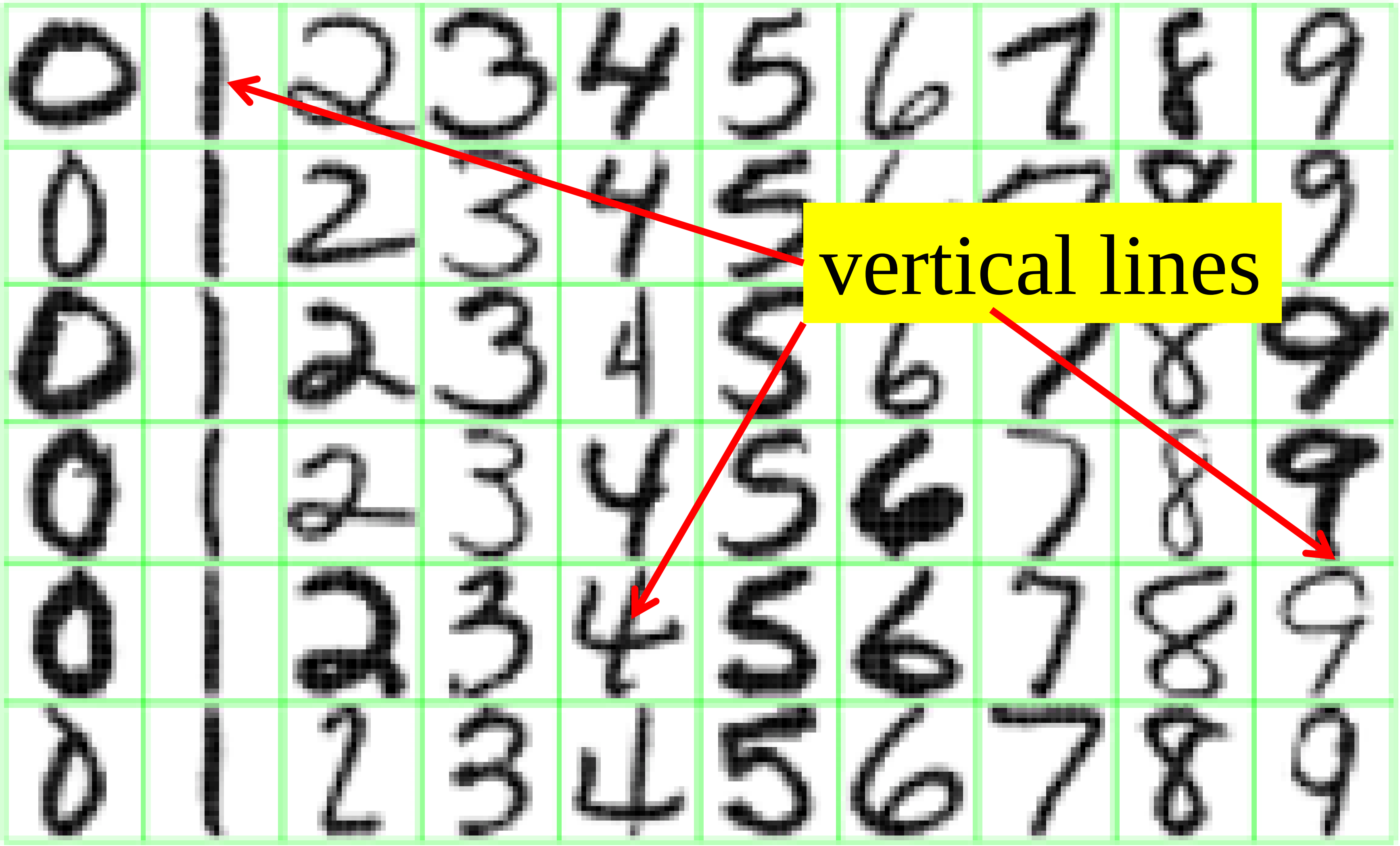
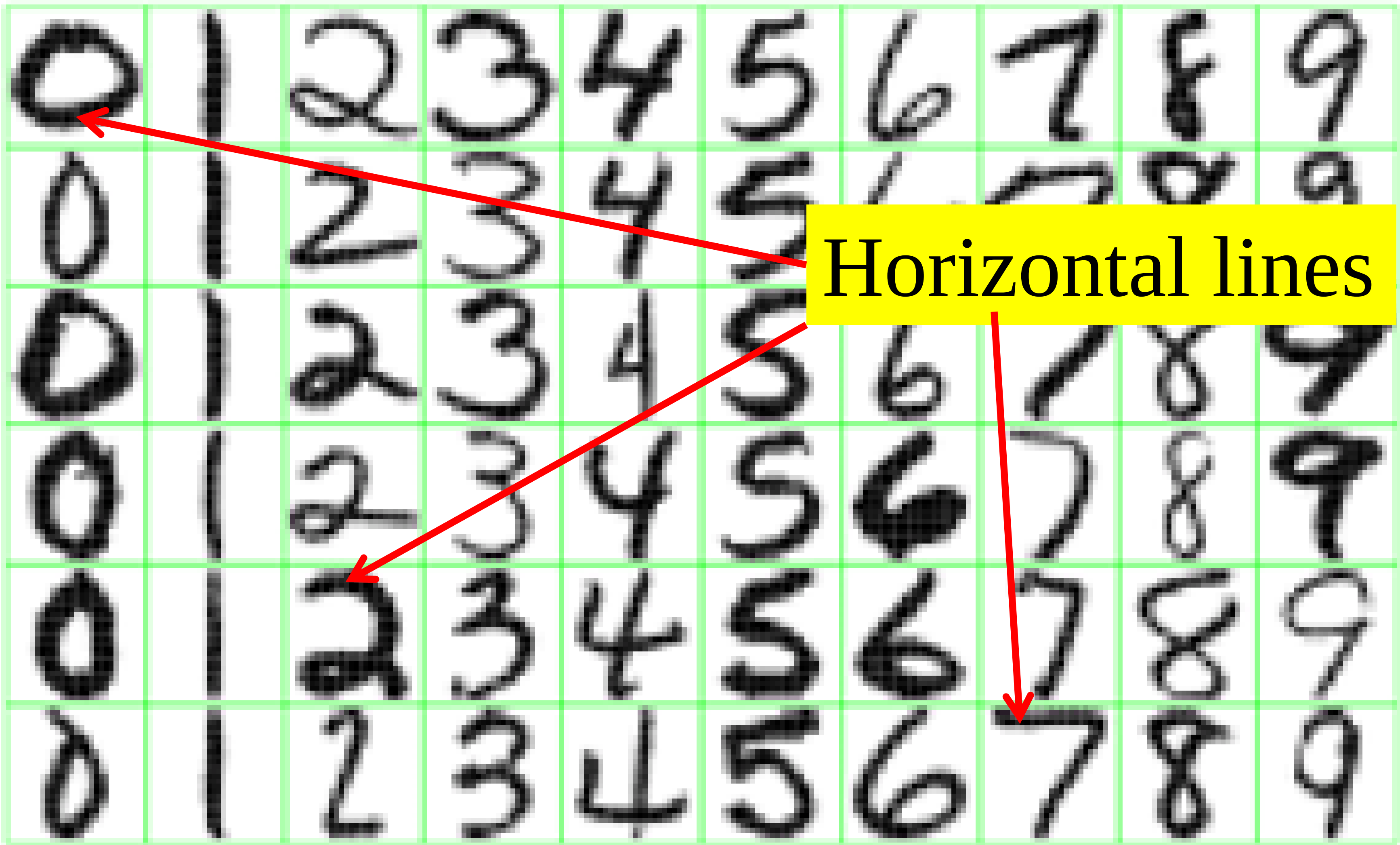


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*



1

Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

1

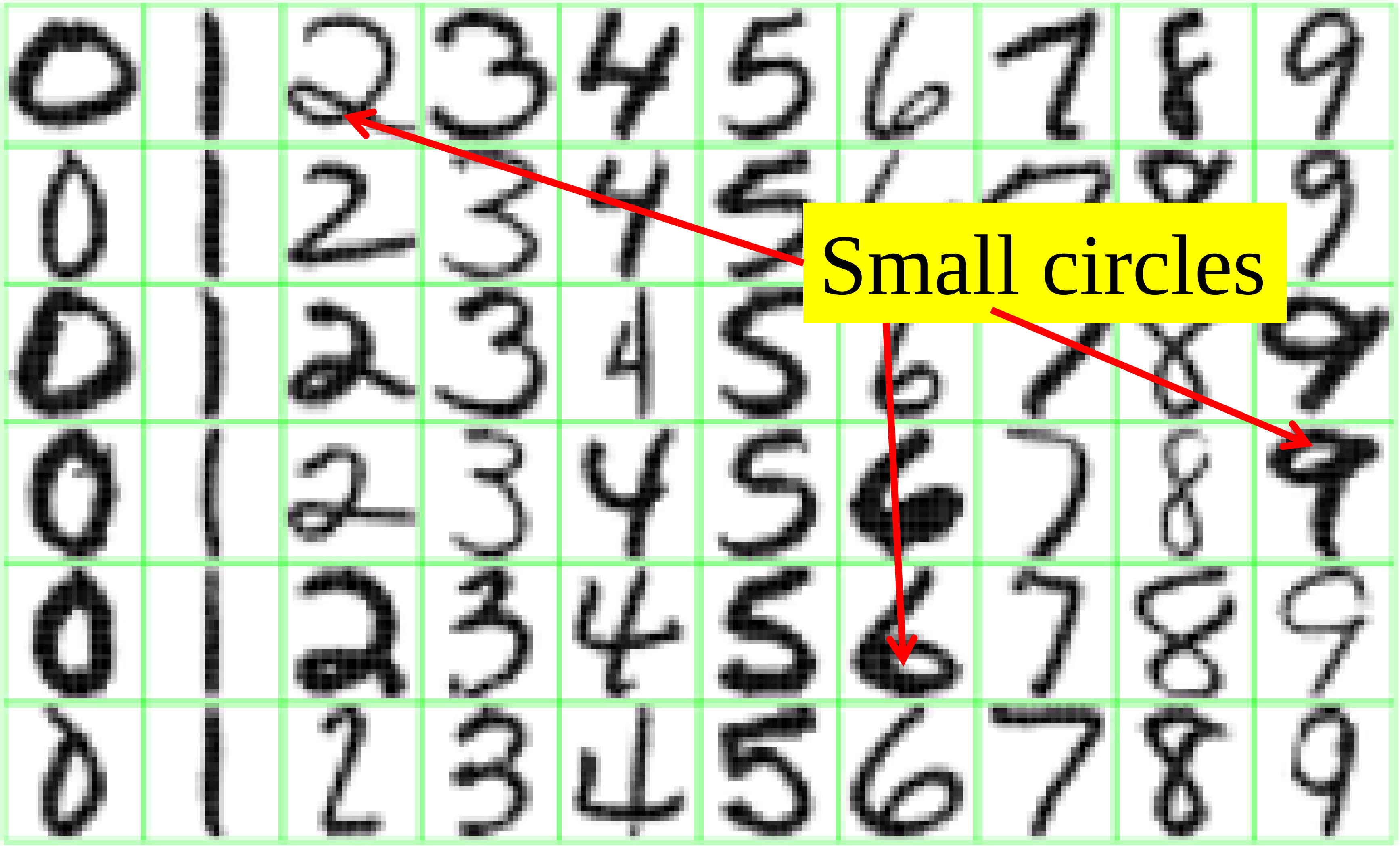
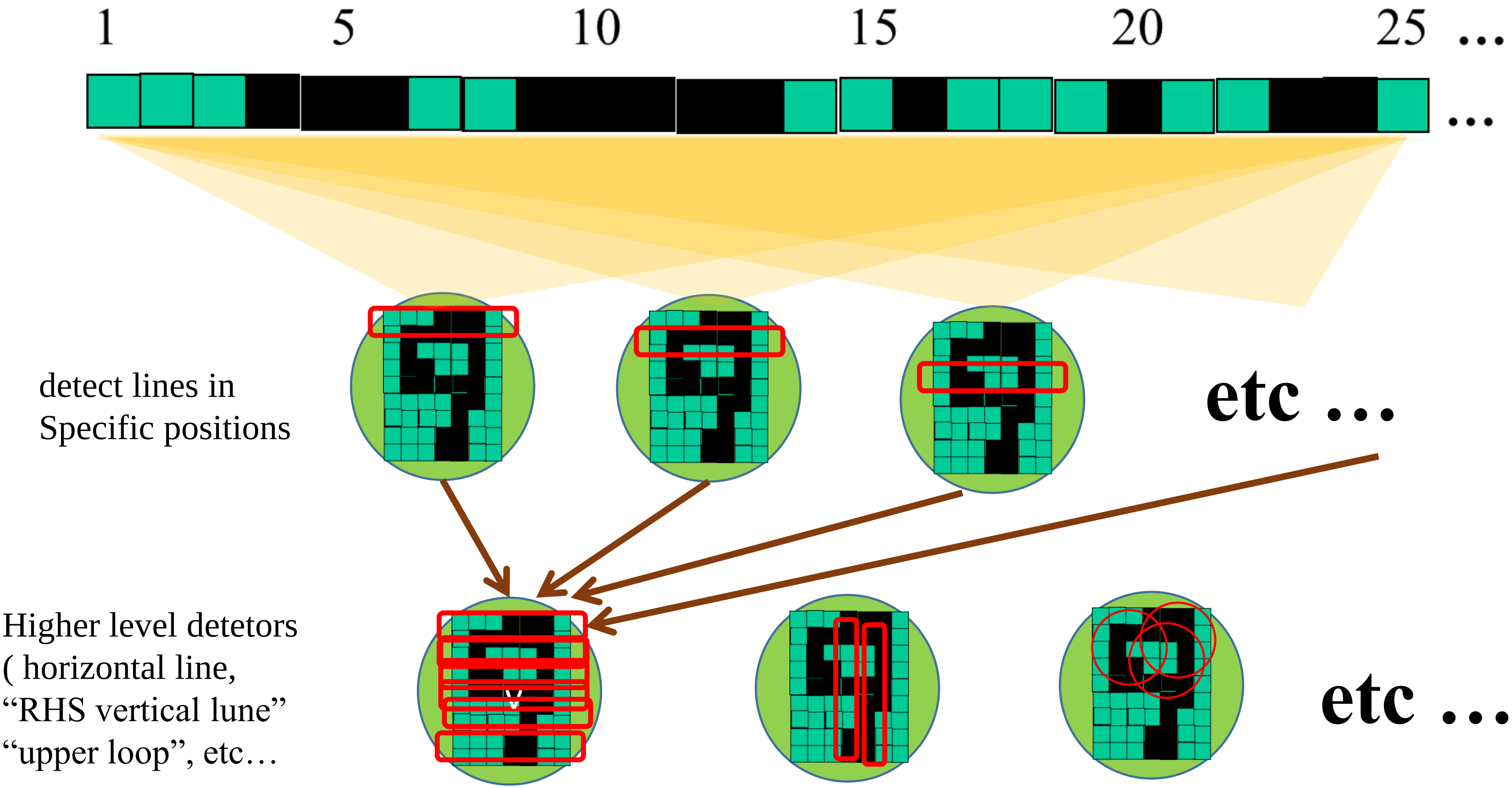


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

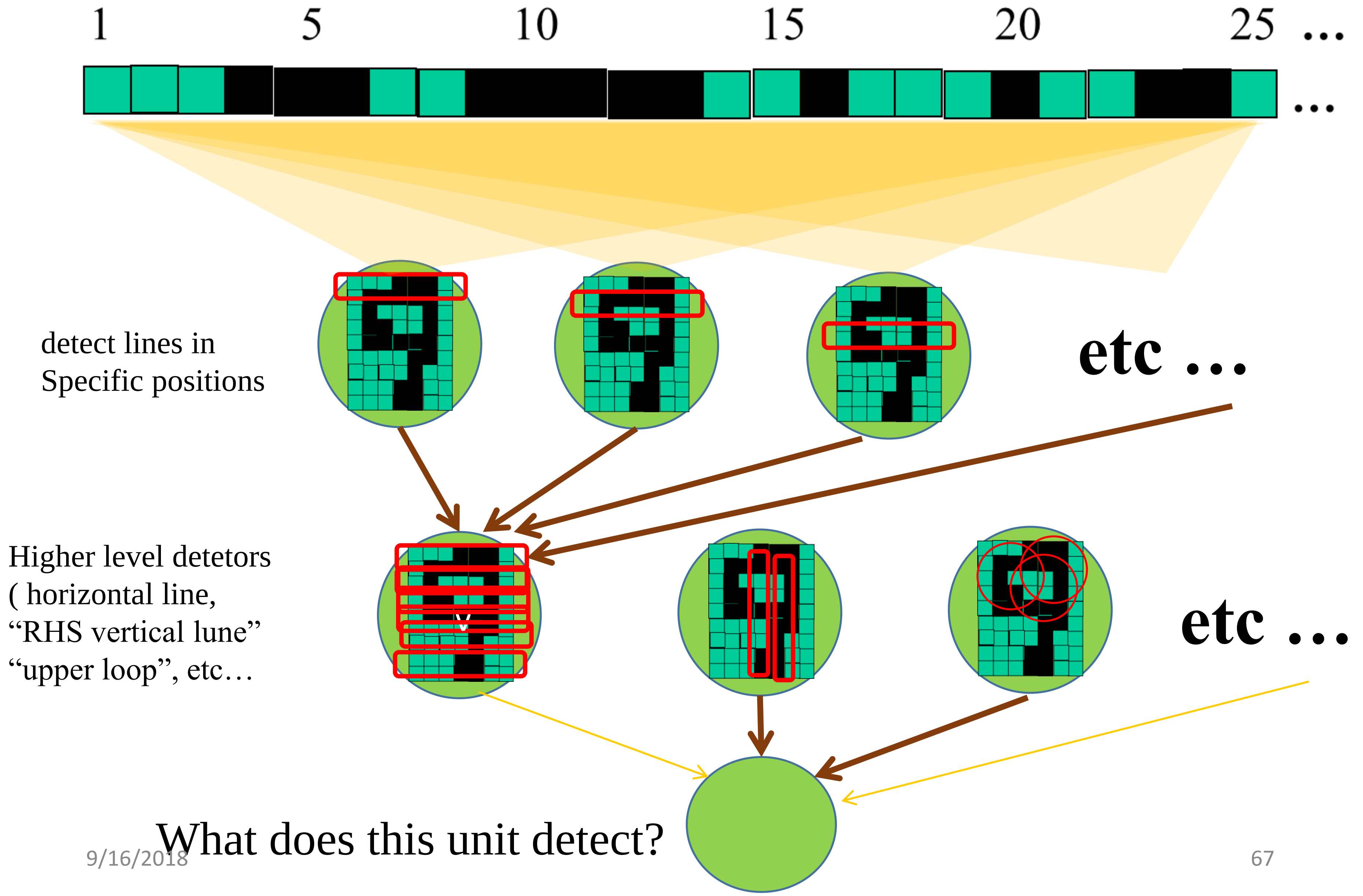
1

But what about position invariance ???
our example unit detectors were tied to
specific parts of the image

successive layers can learn higher-level features …

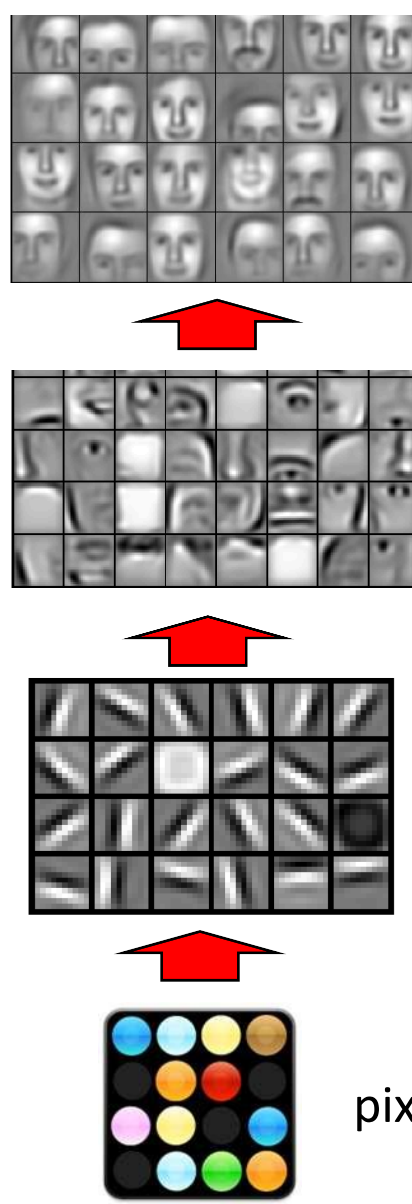


successive layers can learn higher-level features …



Going deep

Training set: Aligned images of faces.



object models

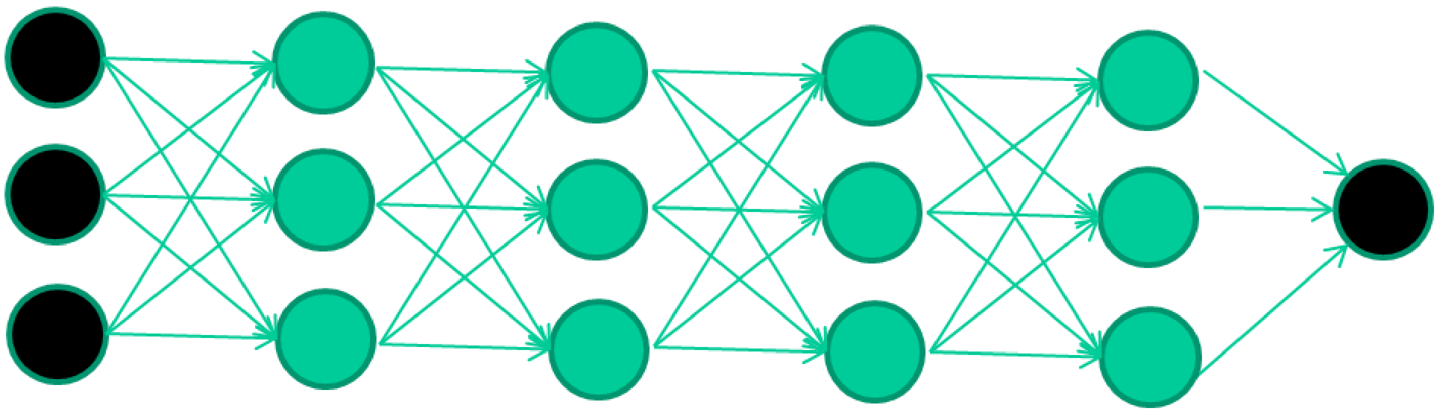
object parts (combination of edges)

edges

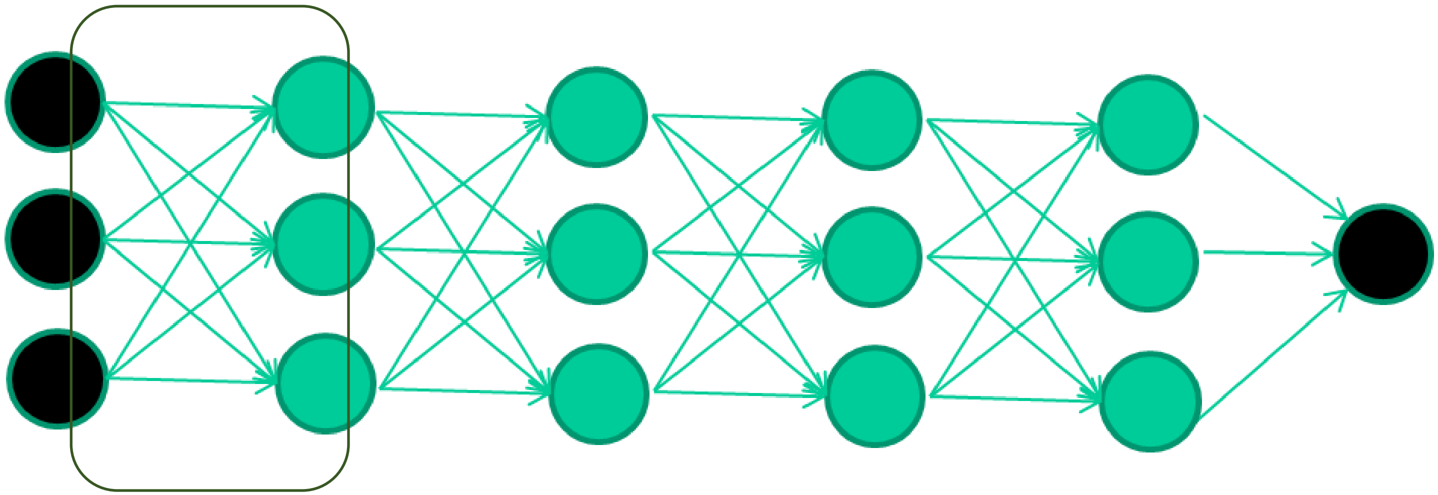
pixels

[Honglak Lee]

New way to train multi-layer NNs…

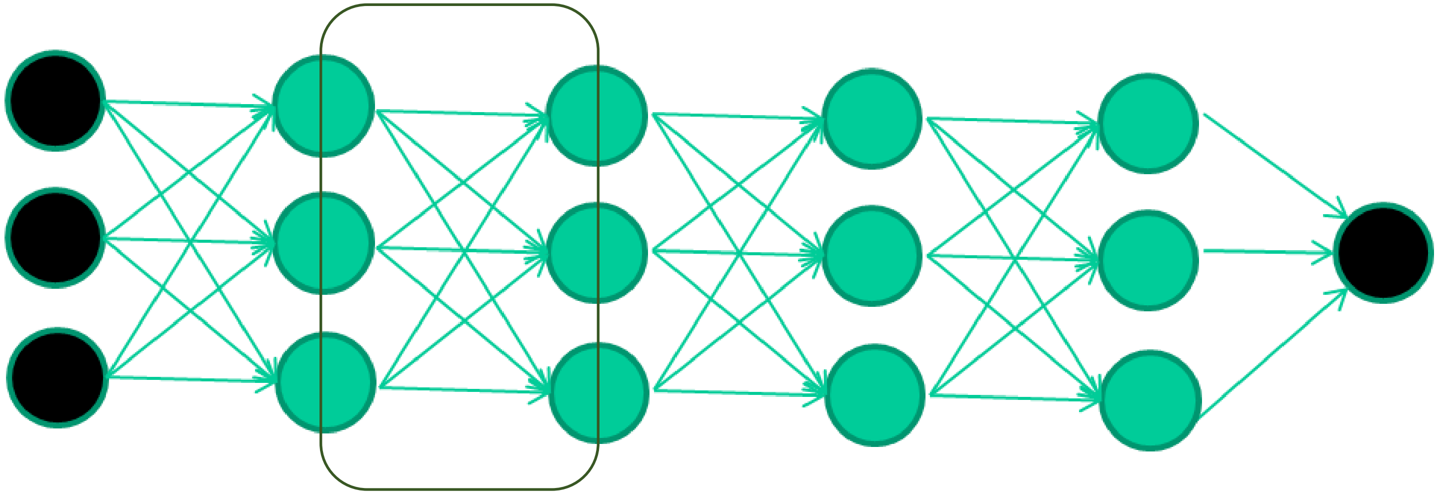


New way to train multi-layer NNs…



Train **this** layer first

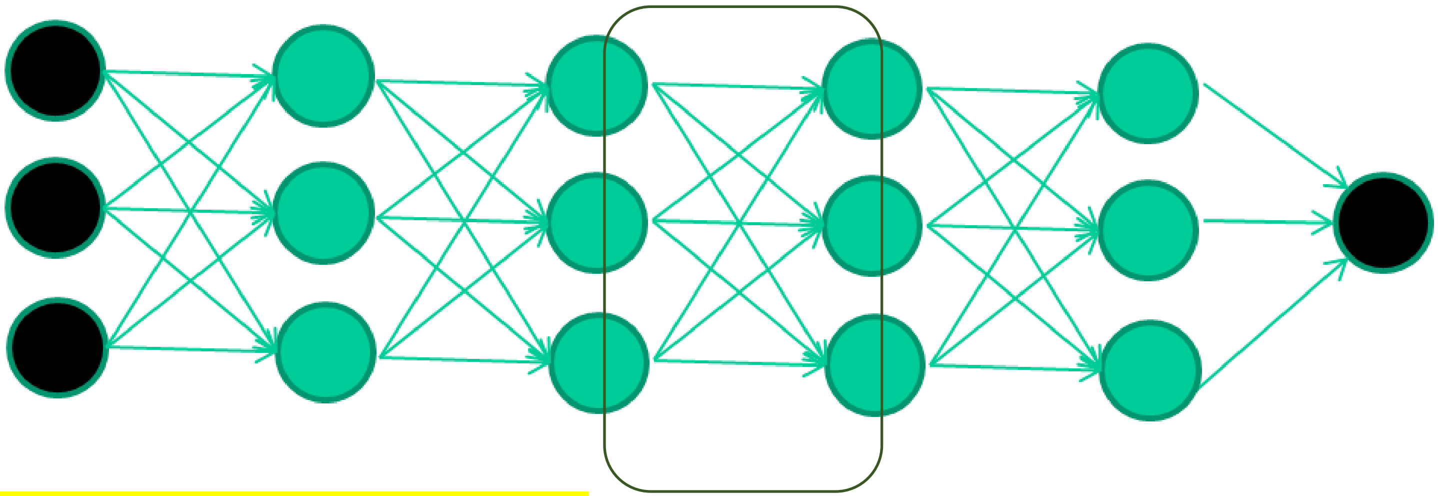
New way to train multi-layer NNs…



Train **this** layer first

then **this** layer

New way to train multi-layer NNs…

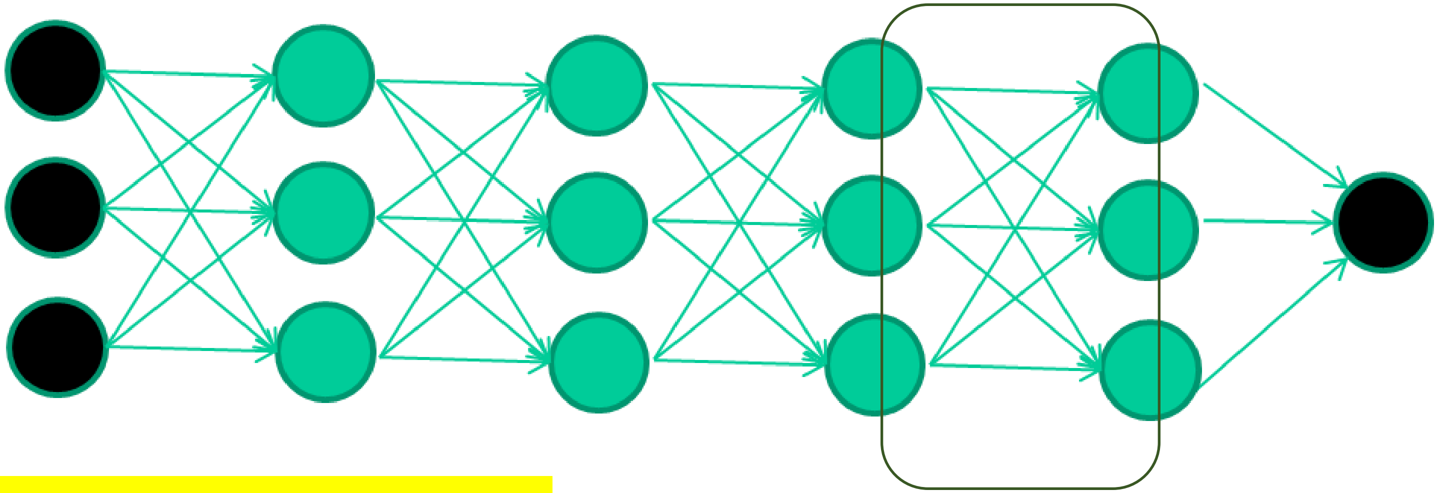


Train **this** layer first

then **this** layer

then **this** layer

New way to train multi-layer NNs…



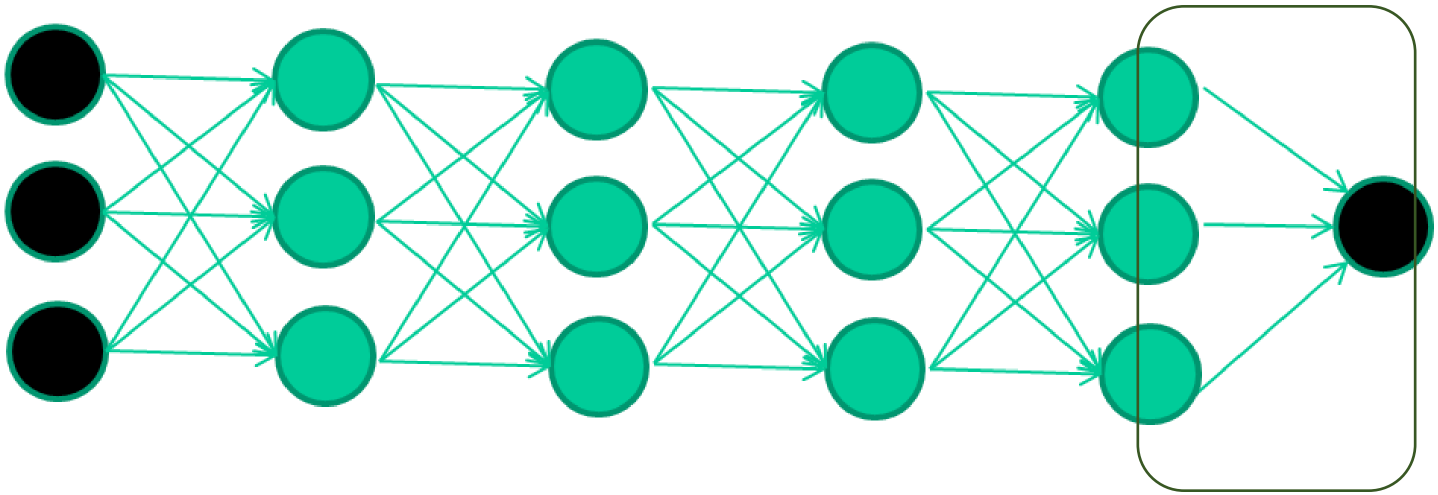
Train **this** layer first

then **this** layer

then **this** layer

then **this** layer

New way to train multi-layer NNs…



Train **this** layer first

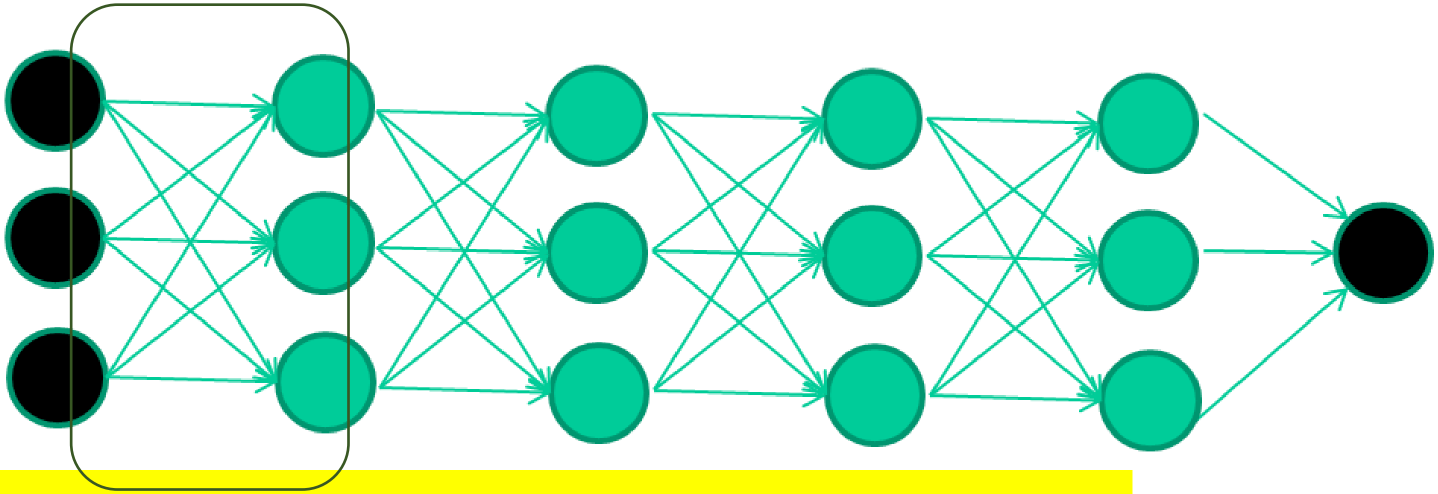
then **this** layer

then **this** layer

then **this** layer

finally **this** layer

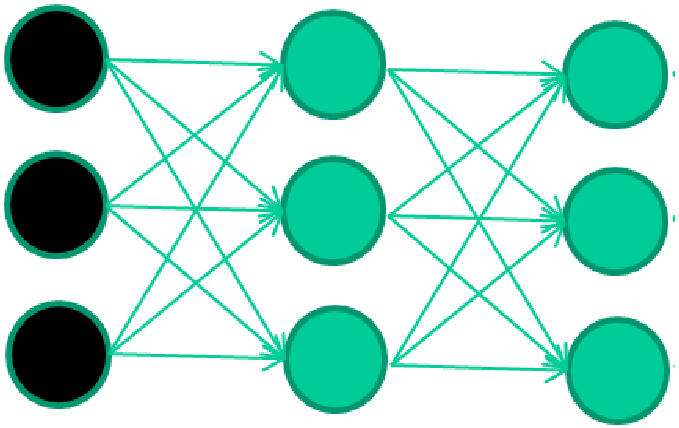
New way to train multi-layer NNs…



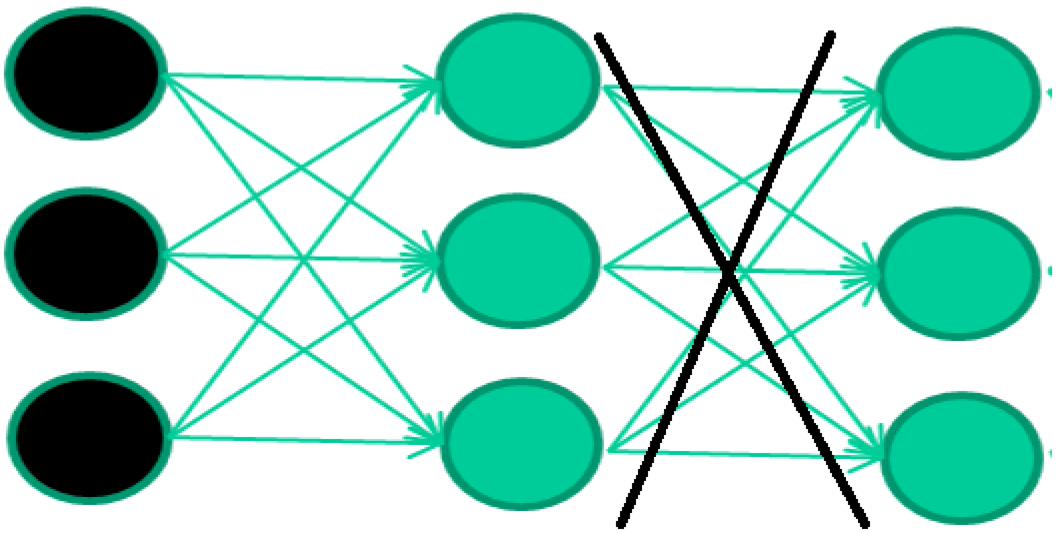
*EACH of the (non-output) layers is trained to be an **autoencoder***

Basically, it is forced to learn good features that describe what comes from the previous layer

Autoencoder

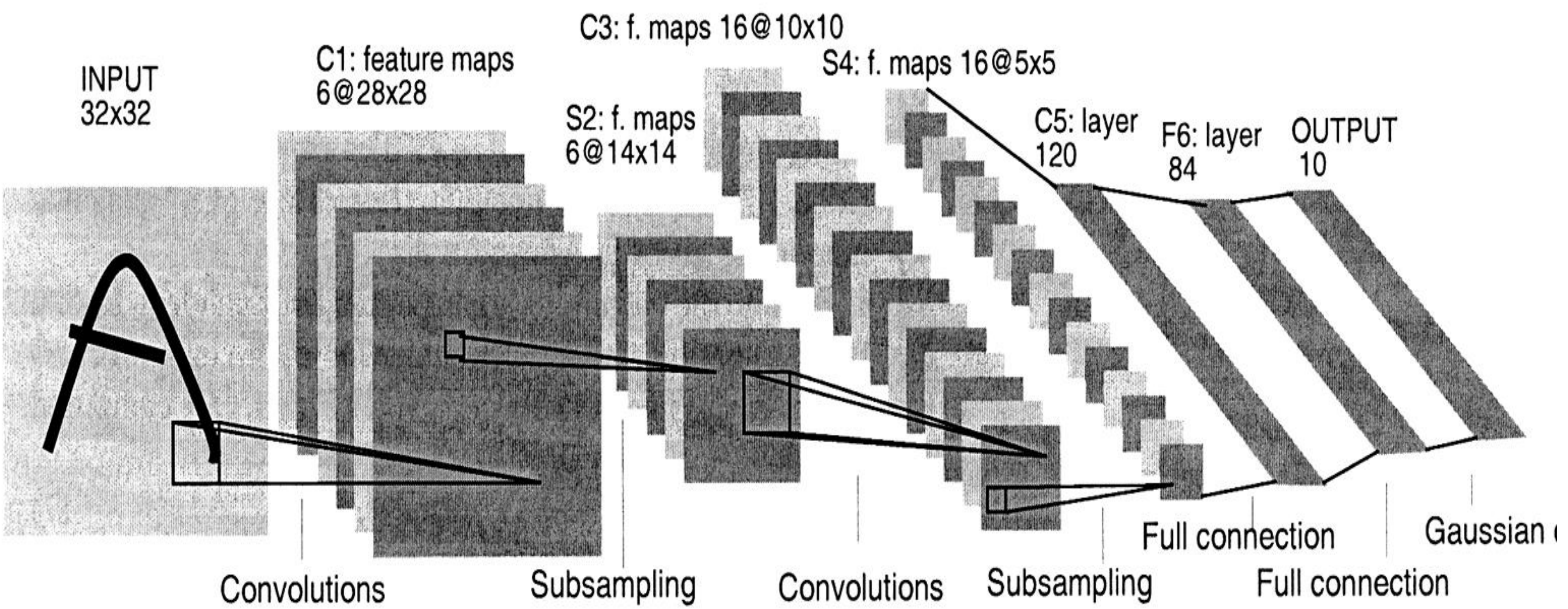


Autoencoder



Deep learning for Images

Convolutional Neural Network (CNN)



Convolution

These are the network parameters to be learned.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

Each filter detects a small pattern (3 x 3).

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Dot product

3 -1

6 x 6 image

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

If stride=2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3 -3

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

Convolution

-1	1	-1
-1	1	-1
-1	1	-1

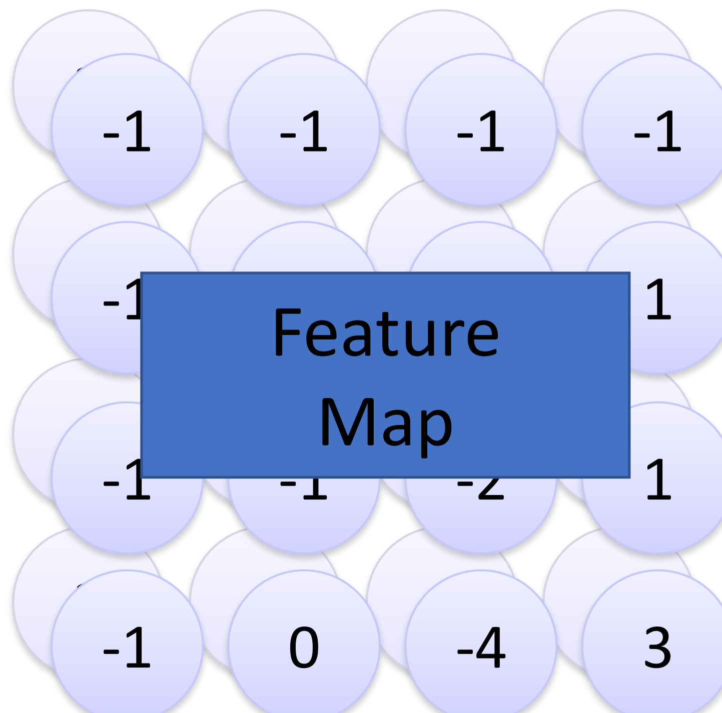
Filter 2

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Repeat this for each filter



Two 4 x 4 images
Forming 2 x 4 x 4 matrix

Color image: RGB 3 channels

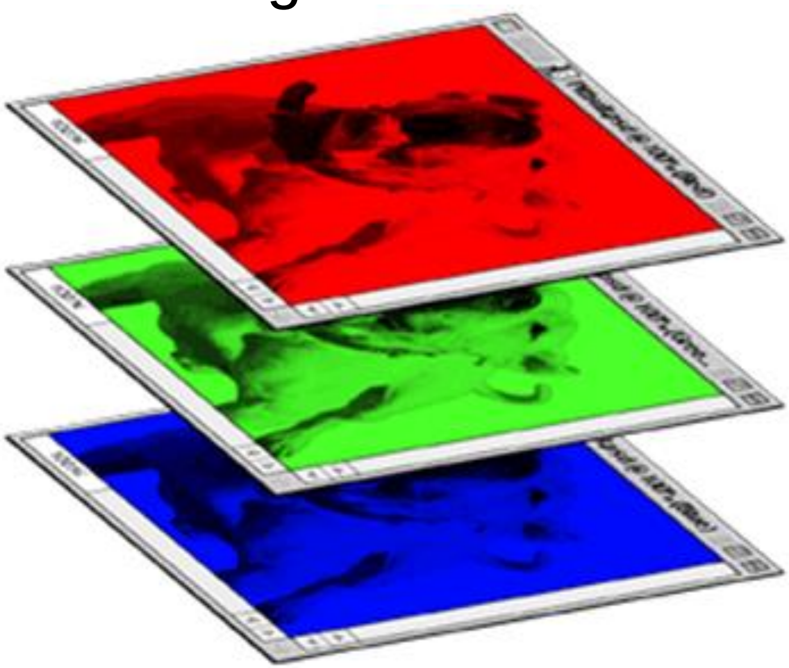
Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

Filter 2

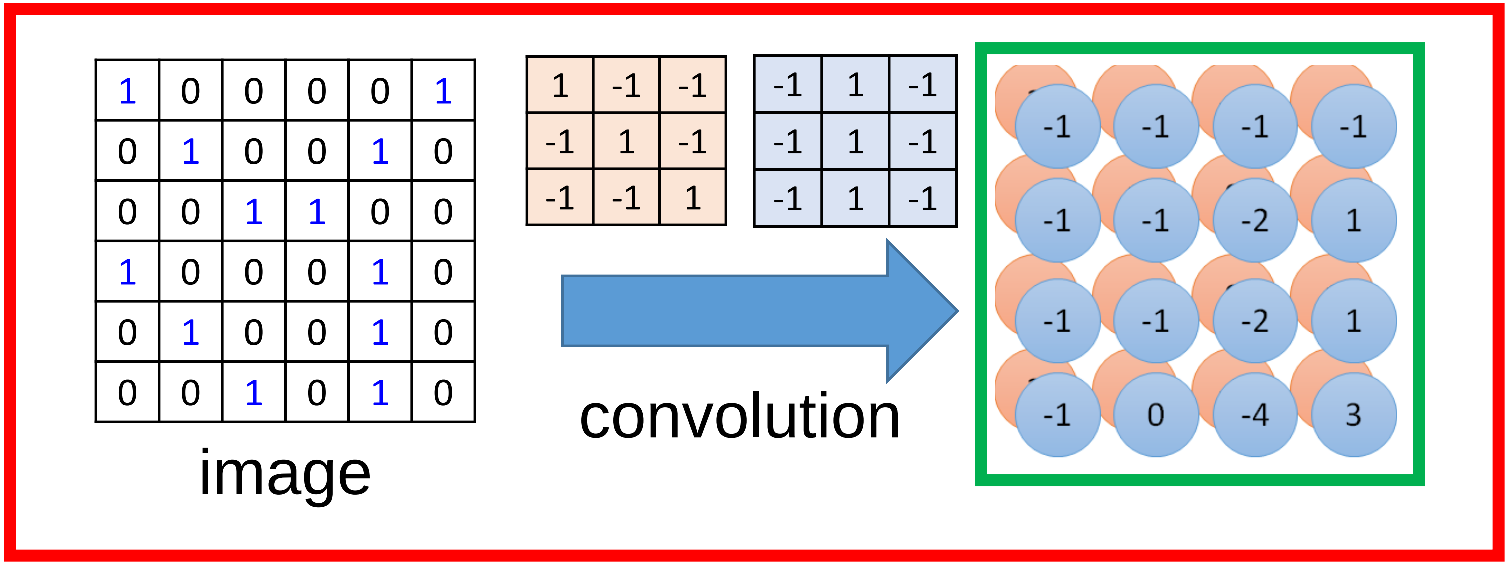
-1	1	-1
-1	1	-1
-1	1	-1

Color image

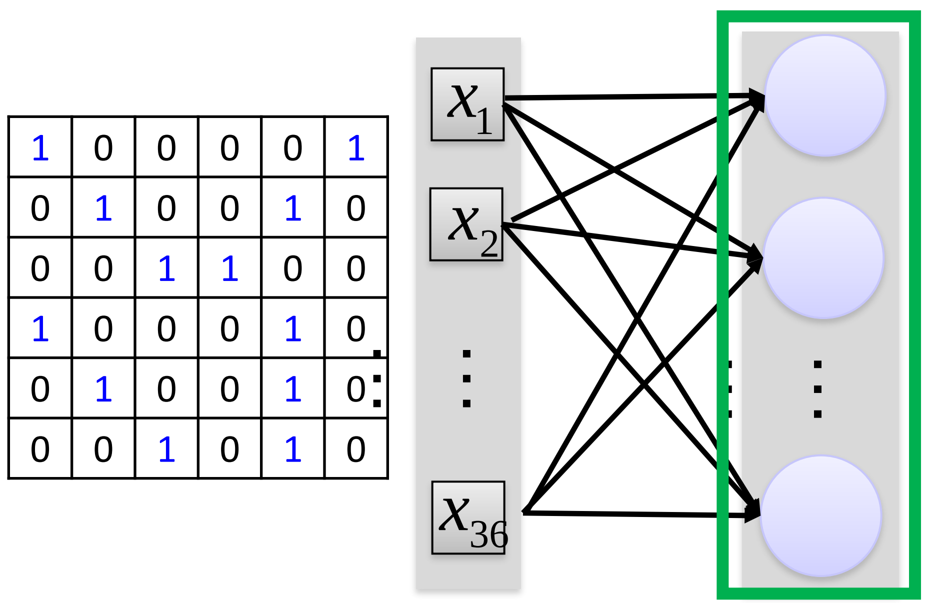


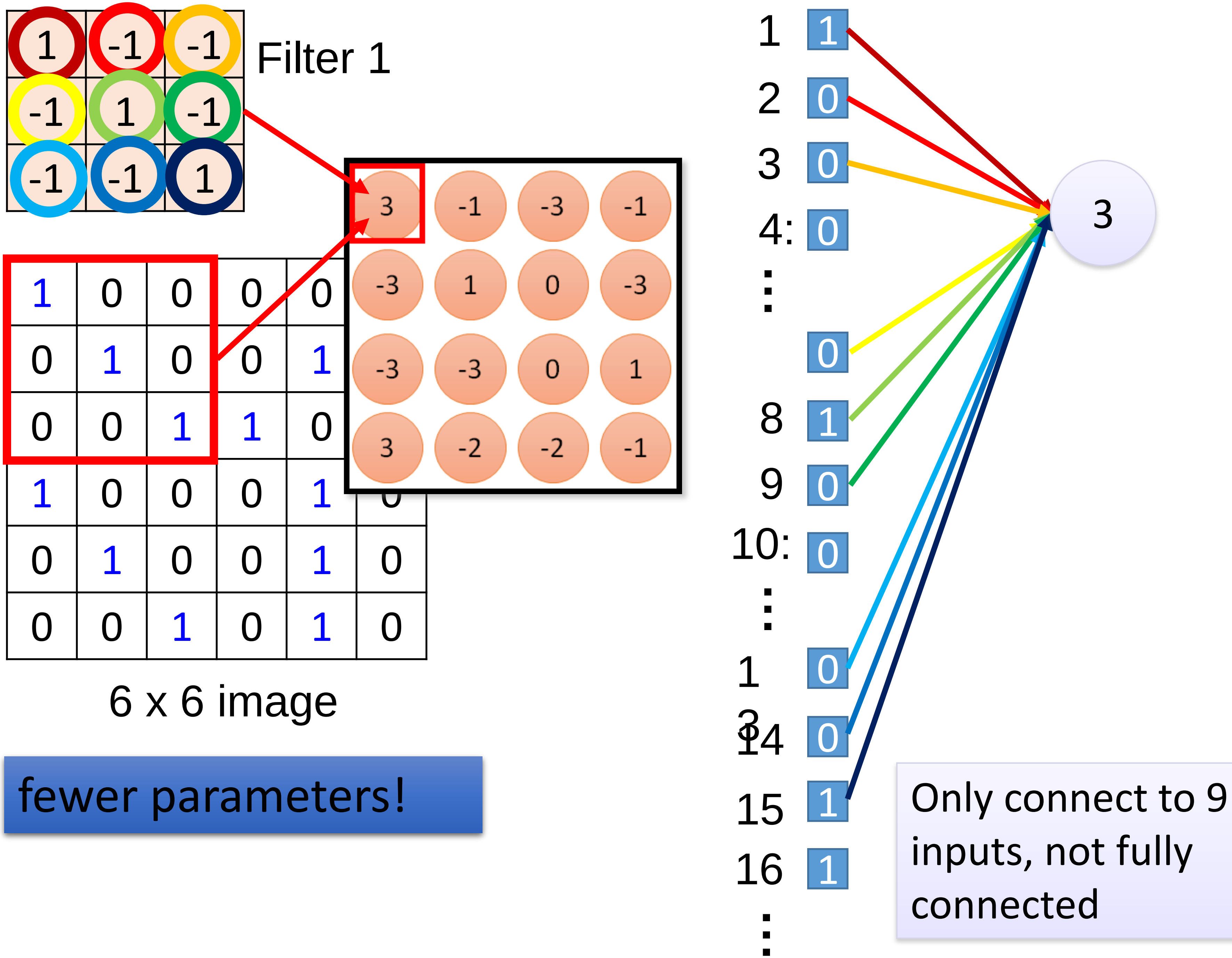
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Convolution v.s. Fully Connected

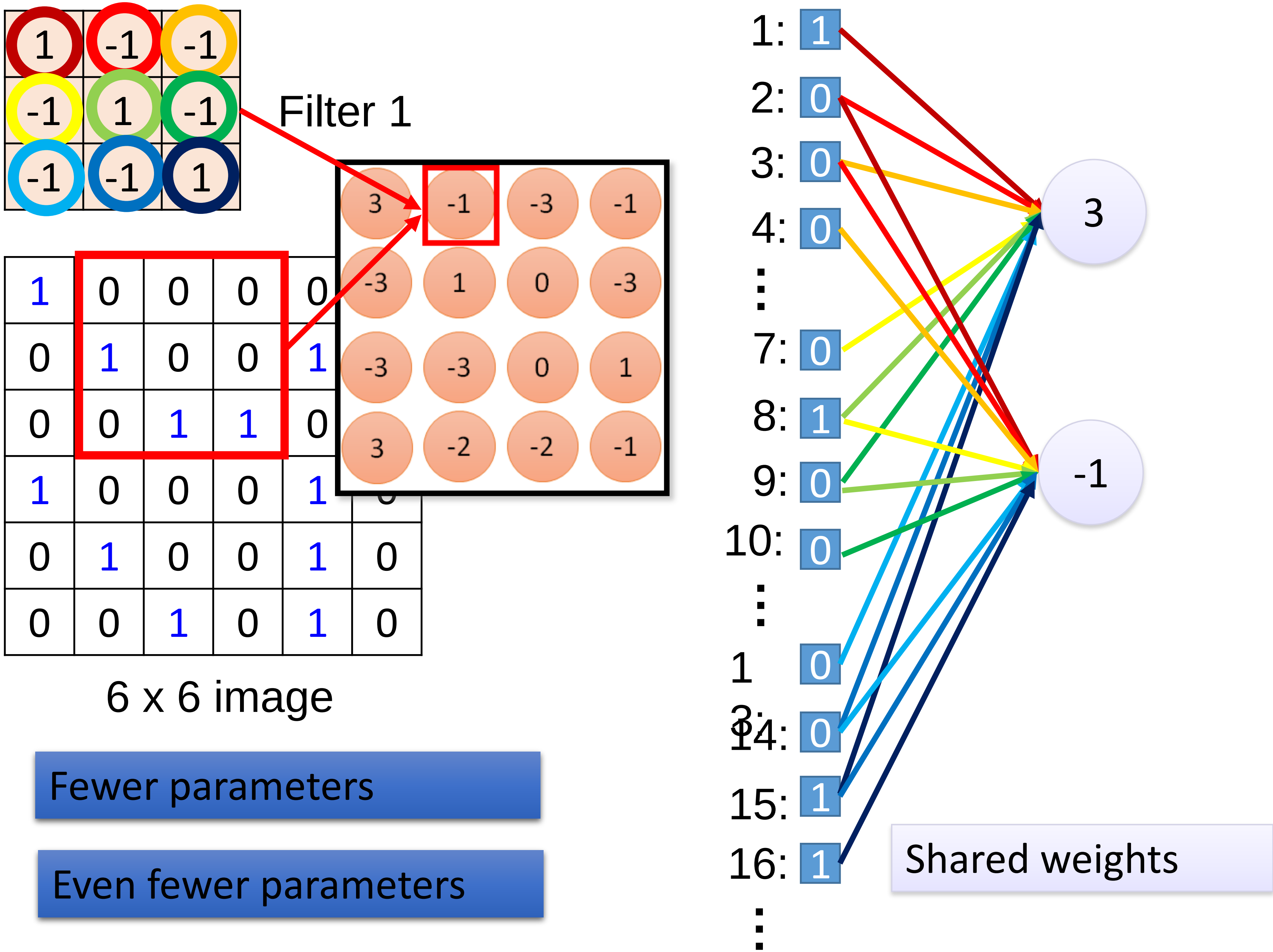


Fully-connected





Filter 1
1 -1 -1
-1 1 -1
-1 -1 1
3 -1 -3 -1
-3 1 0 -3
-3 -3 0 1
3 -2 -2 -1
1 0 0 0 0
0 1 0 0 1
0 0 1 1 0
1 0 0 0 1
0 1 0 0 1 0
0 0 1 0 1 0
6 x 6 image
fewer parameters!
1
2
3
4:
8
9
10:
13
14
15
16
3
Only connect to 9 inputs, not fully connected


1 -1 -1
-1 1 -1
-1 -1 1
Filter 1
3 -1 -3 -1
-3 1 0 -3
-3 -3 0 1
3 -2 -2 -1
1 0 0 0 0
0 1 0 0 1
0 0 1 1 0
1 0 0 0 1
0 1 0 0 1 0
0 0 1 0 1 0
6 x 6 image
Fewer parameters
Even fewer parameters
1: 1
2: 0
3: 0
4: 0
7: 0
8: 1
9: 0
10: 0
13: 0
14: 0
15: 1
16: 1
3
-1
Shared weights

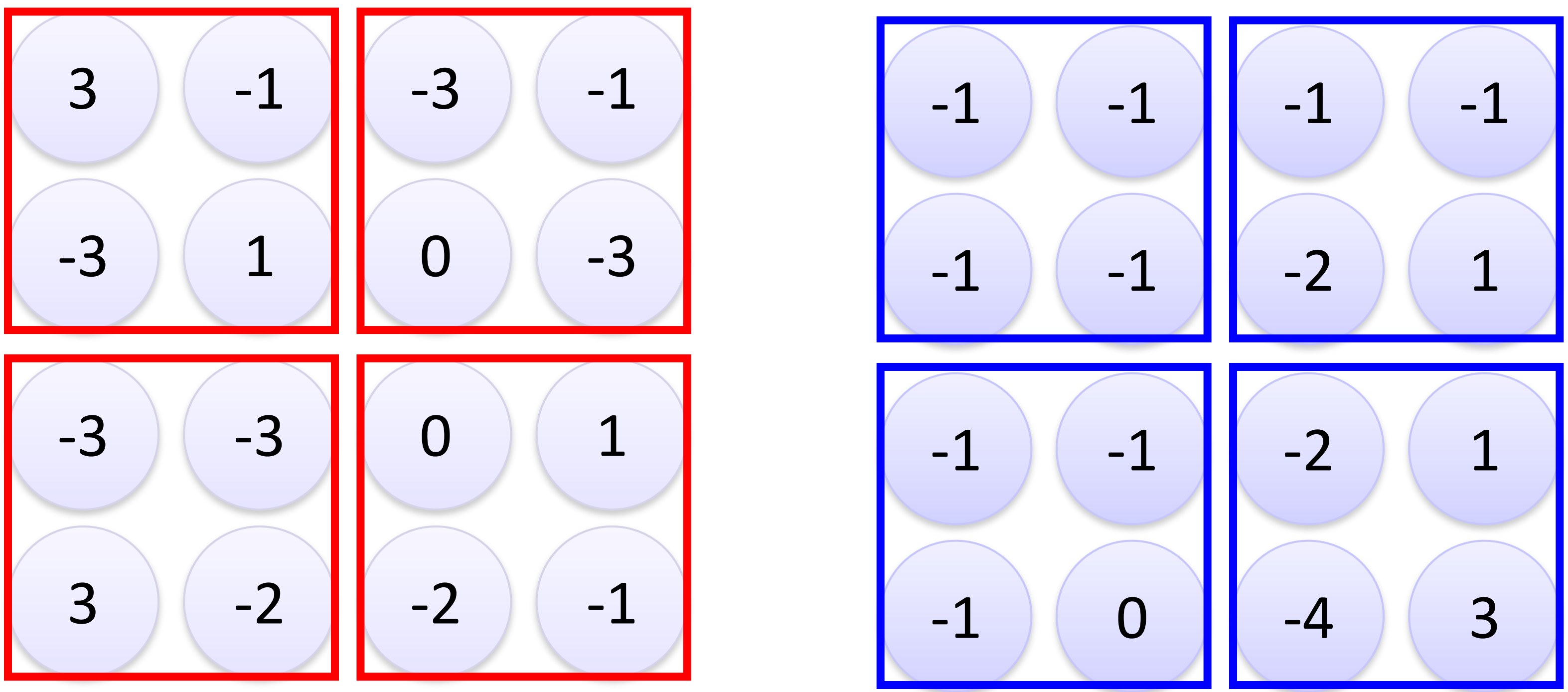
Max Pooling

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

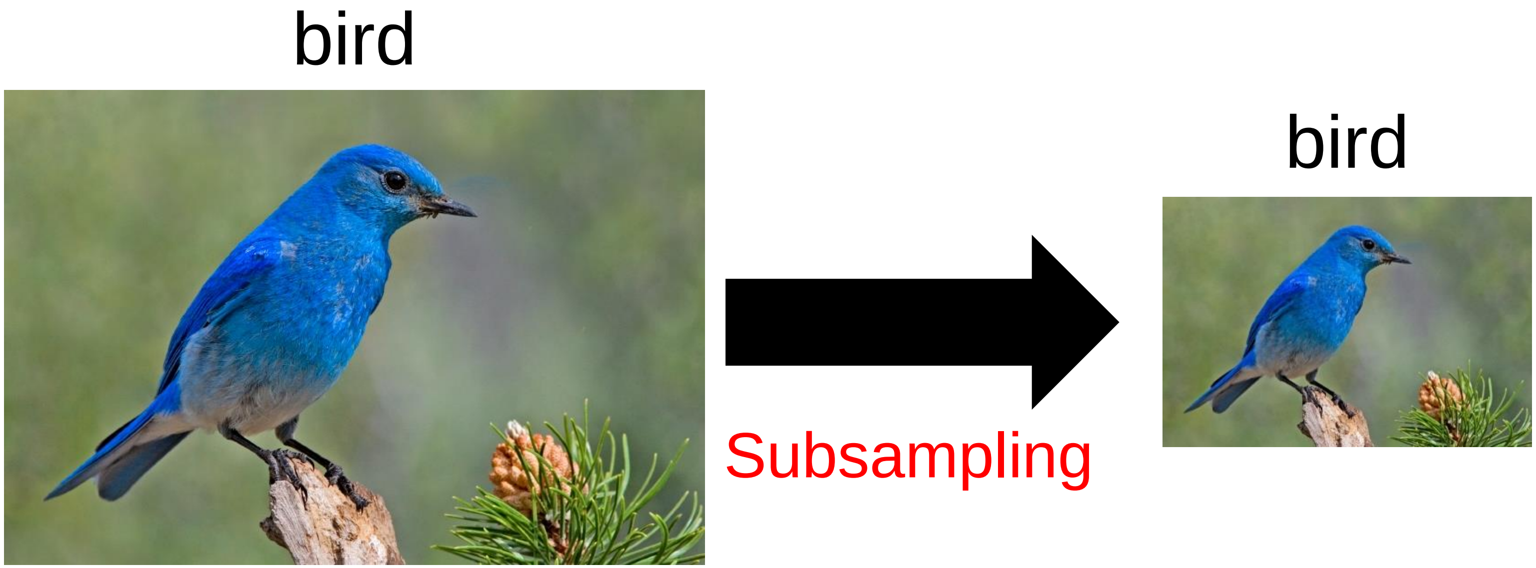
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2



Why Pooling?

- Subsampling pixels will not change the object



We can subsample the pixels to make image smaller

fewer parameters to characterize the image

A CNN compresses a fully connected network in two ways

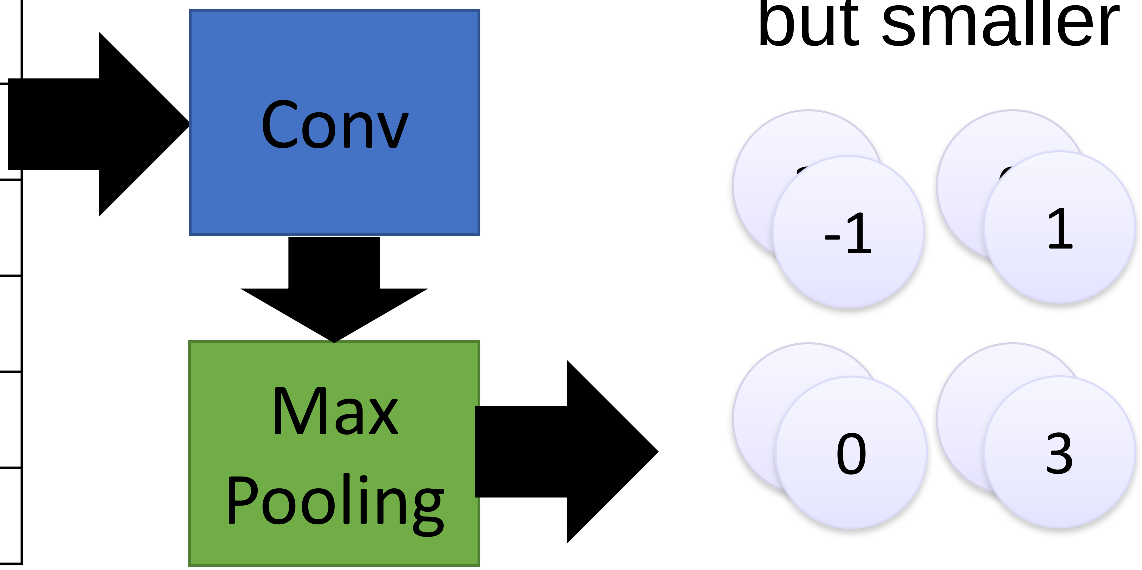
- Reducing number of connections
- Shared weights on the edges
- Max pooling further reduces the complexity

Max Pooling

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

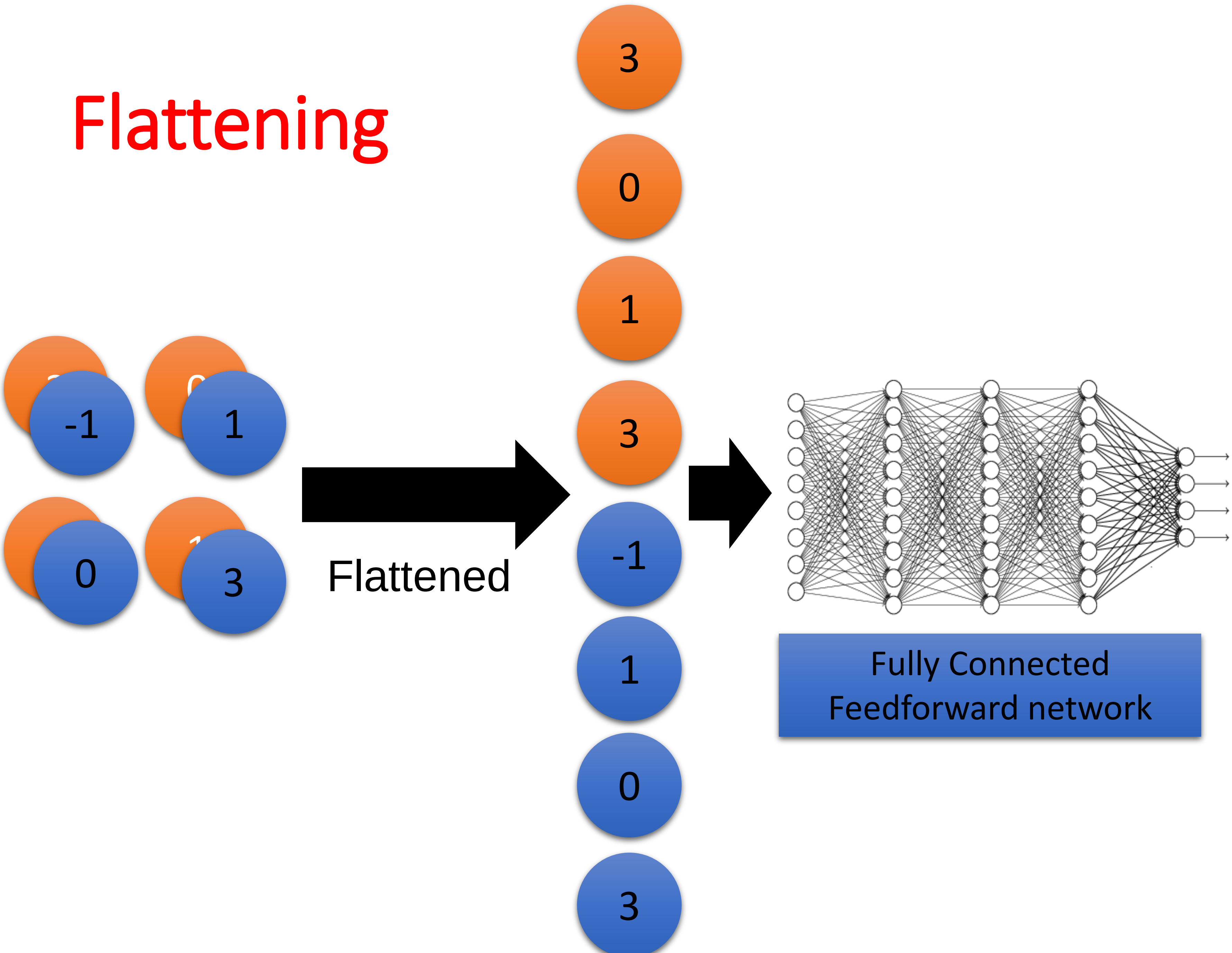
New image
but smaller



2 x 2 image

Each filter
is a channel

Flattening

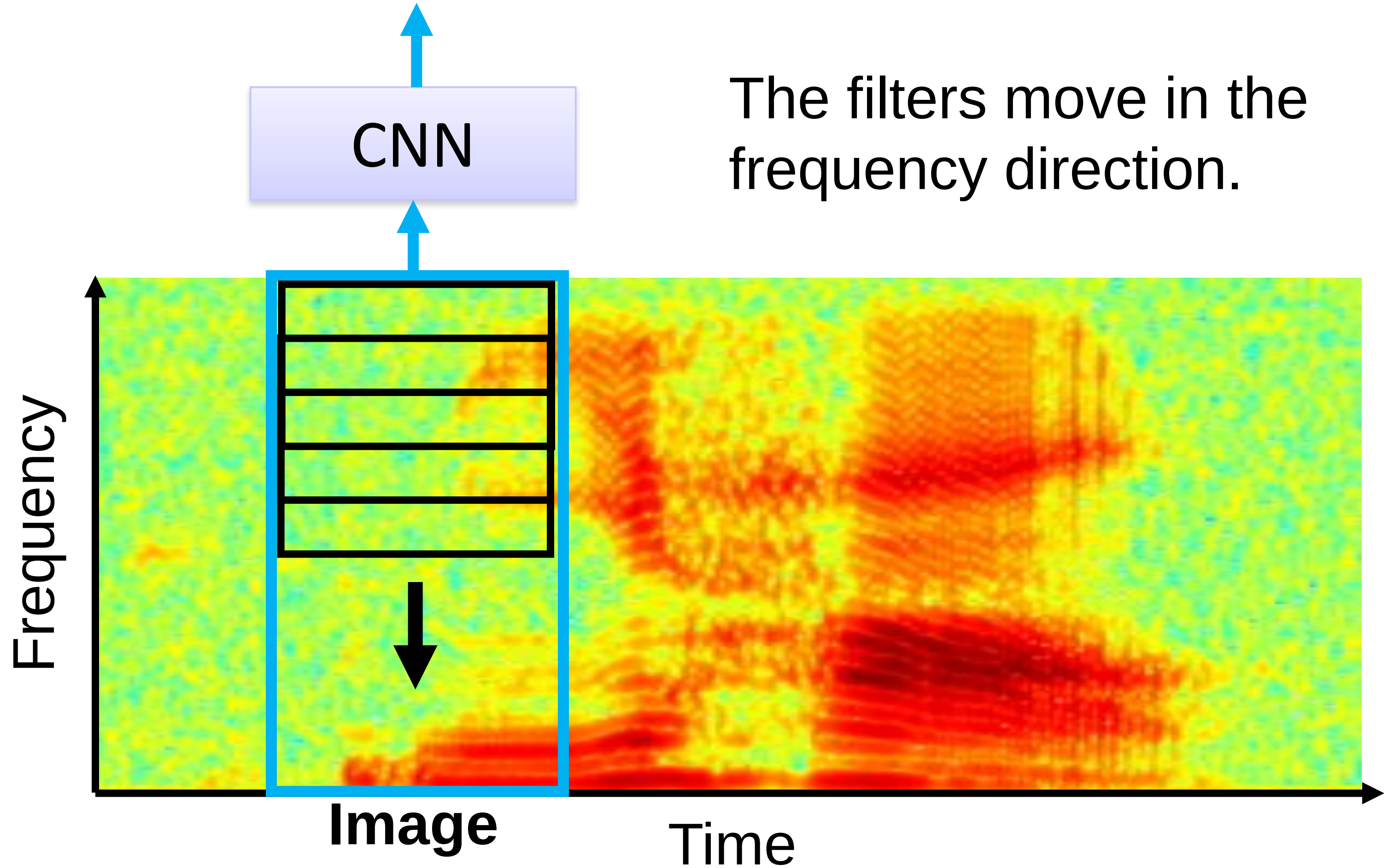

-1
1
0
3
Flattened
3
0
1
3
-1
1
0
3
Fully Connected
Feedforward network

Convolutional Neural Network (CNN)

- Compared to standard feedforward neural networks with similarly-sized layers,
  - CNNs have much fewer connections and parameters
  - and so they are easier to train,
  - while their theoretically-best performance is likely to be only slightly worse.

CNN in speech recognition

The filters move in the frequency direction.



Spectrogram

Challenges

- How to decide the number of hidden layers and nodes?
- Choosing suitable deep learning architecture for a given data
- Choosing suitable error function

References

- Neural Networks for Pattern Recognition”, Bishop, C.M., 1996
- Deep Belief Nets, 2007 NIPS tutorial , G . Hinton
- Slides adapted from Andrew NG and G . Hinton
- https://www.macs.hw.ac.uk/~dwcorne/
- https://cs.uwaterloo.ca/~mli/
- cs231n.stanford.edu/slides/2017/

Thanks