6.819/6.869 Advances in Computer Vision

Aditya Khosla

Image by
kirkh.deviantart.com

Today's class

- Part 1: What is deep learning?

- Part 2: Supervised Deep Learning
  - Neural networks
  - Convolutional Neural Networks (CNNs)

- Part 3: Unsupervised Deep Learning
  - Overview of some approaches

Slide credit

- Many slides are taken/adapted from

Fei-Fei Li

Andrew Ng

Part 1:

What is deep learning?

Typical goal of machine learning

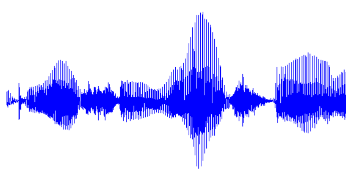
input | **output**

images/video

ML

Label: "Motorcycle"
Suggest tags
Image search
...

audio



ML

Speech recognition
Music classification
Speaker identification
...

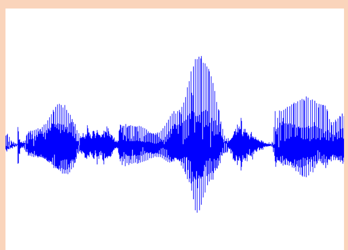
text

ML

Web search
Anti-spam
Machine translation
...

Typical goal of machine learning

Feature engineering: most time consuming!

input			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 ...

Our goal in object classification

→ ML → "motorcycle"

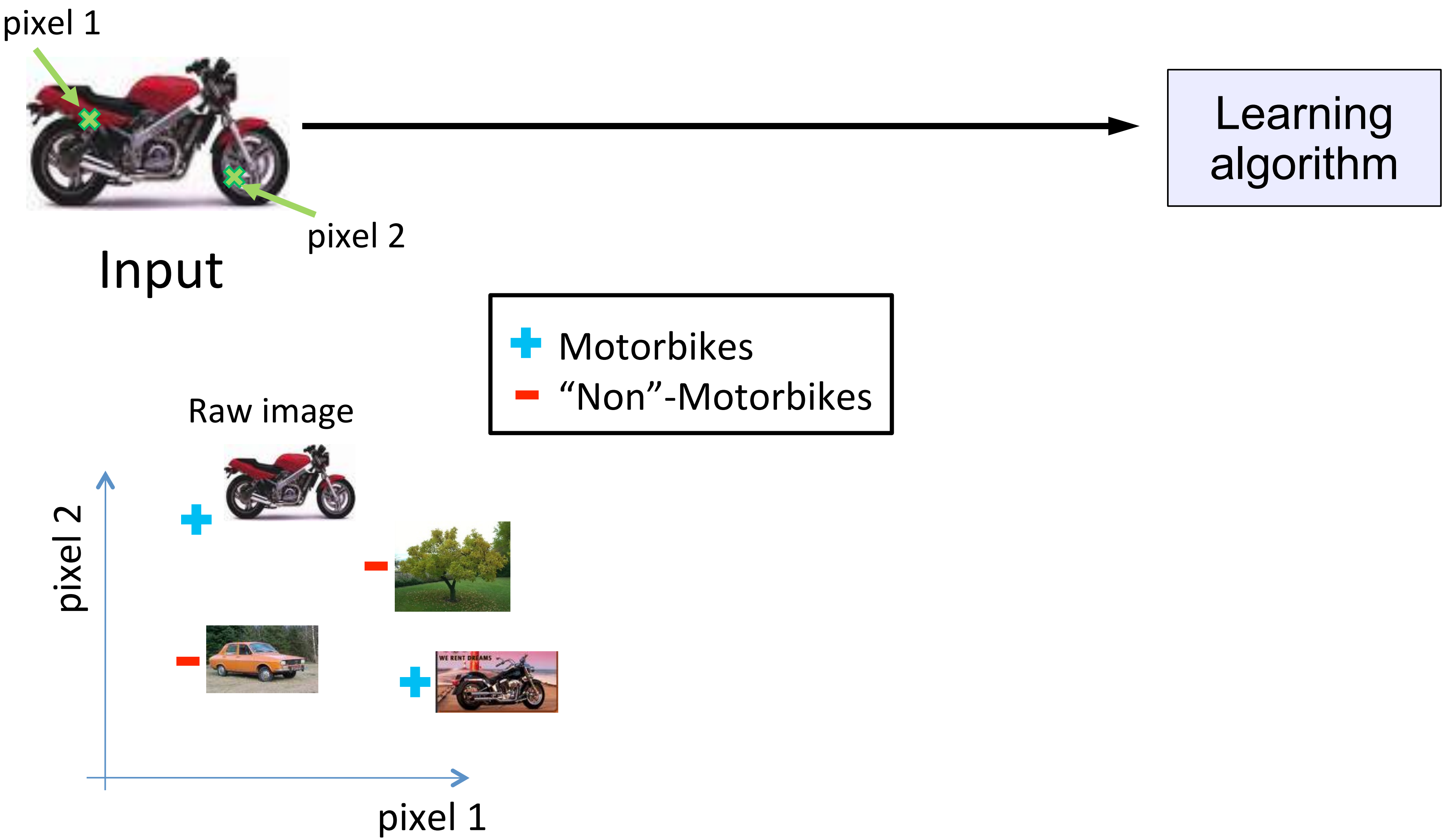
Why is this hard?

You see this:

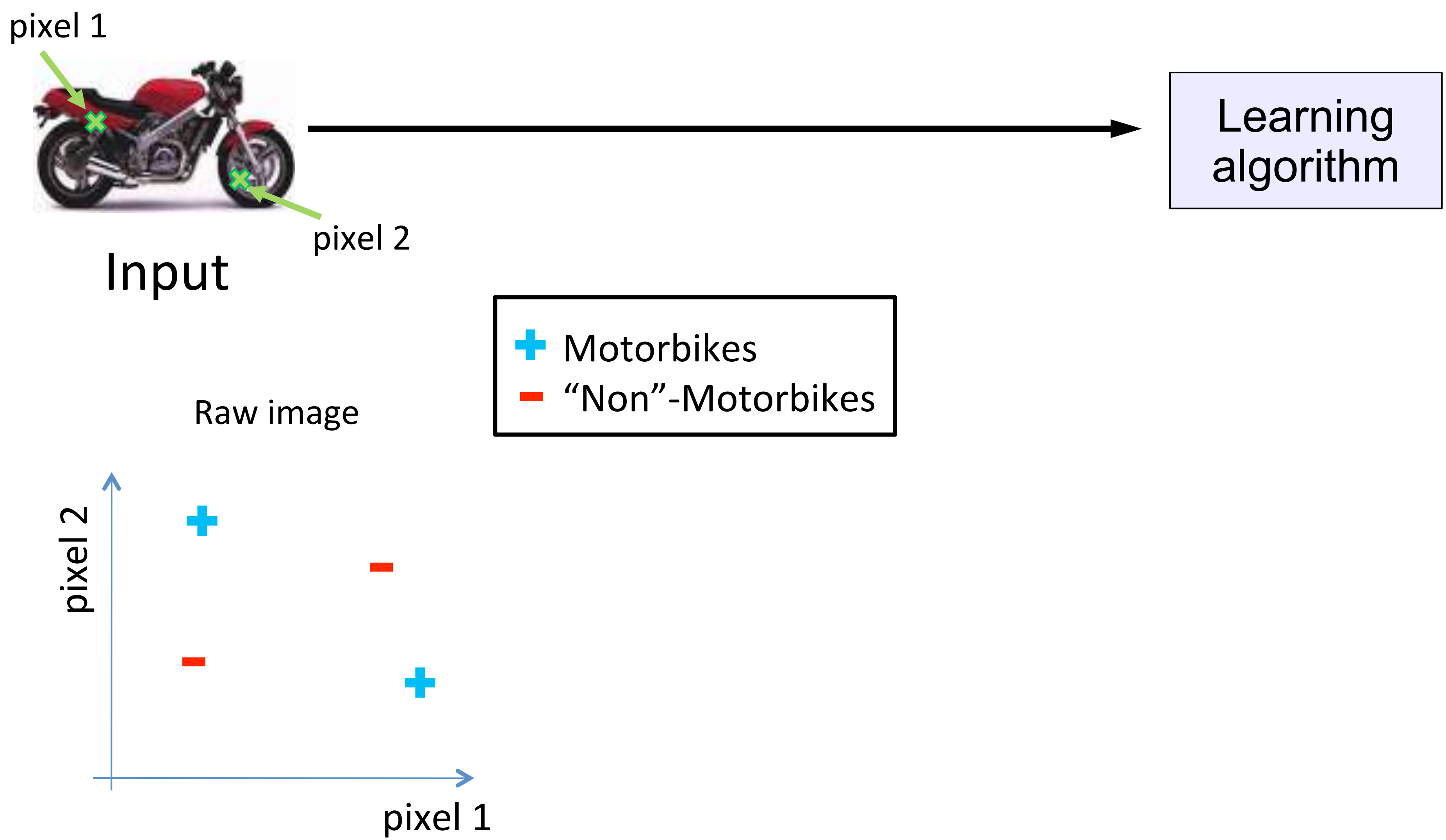
But the camera sees this:

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

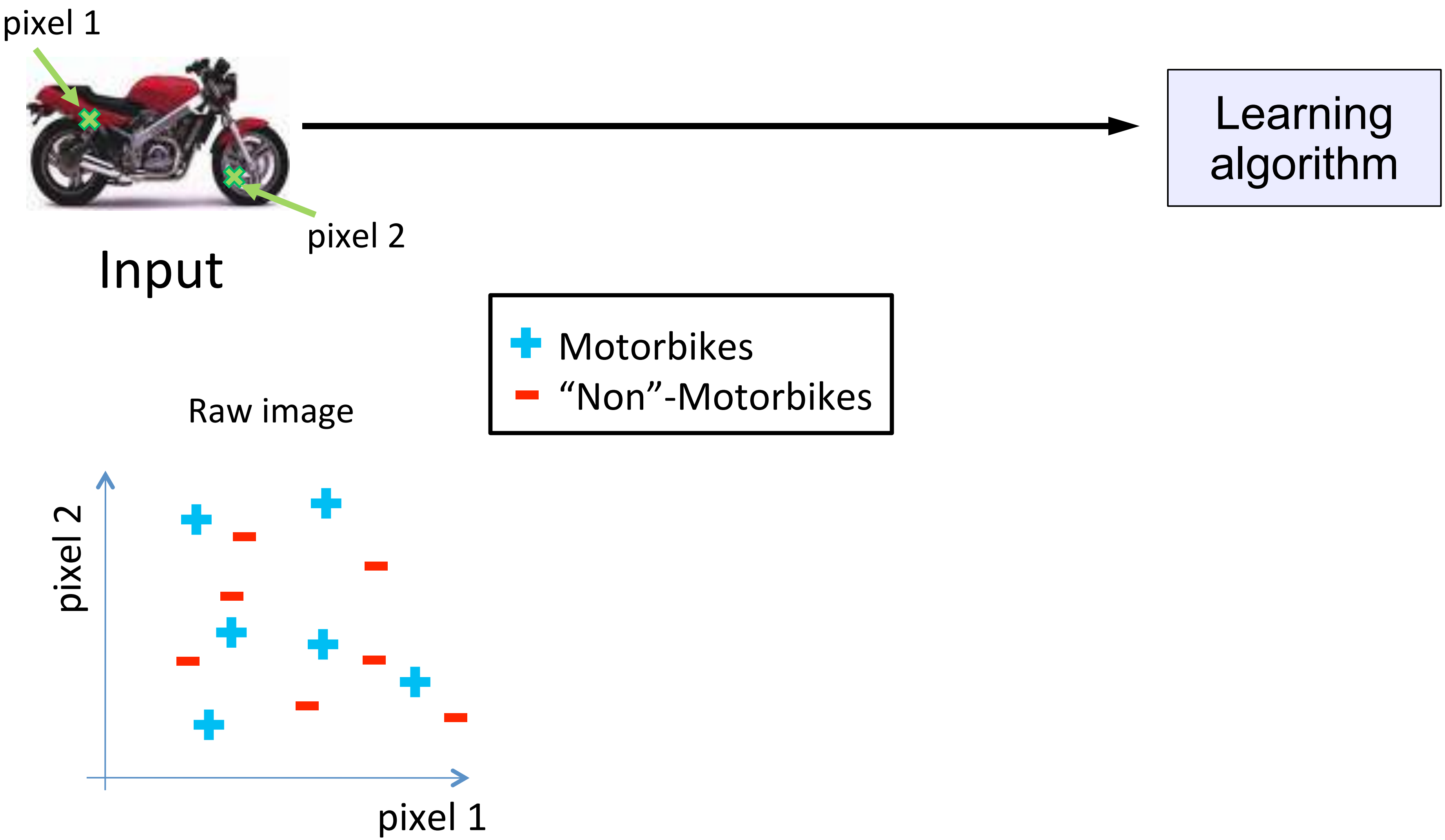
Pixel-based representation



Pixel-based representation



Pixel-based representation



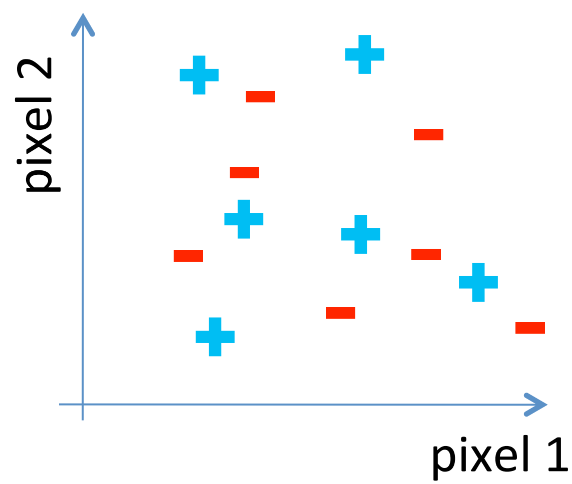
What we want

Input → Feature representation → Learning algorithm

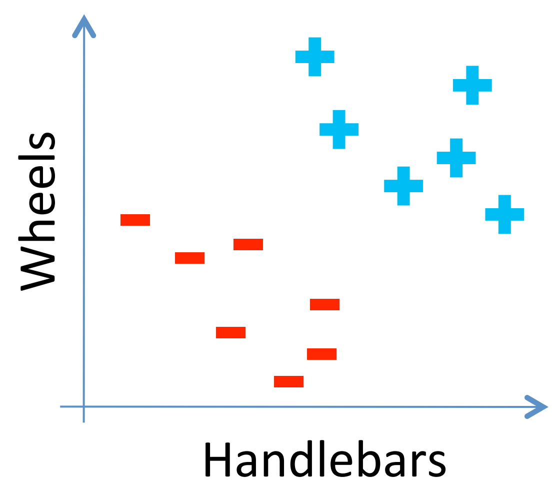
E.g., Does it have Handlebars? Wheels?

+ Motorbikes
– “Non”-Motorbikes

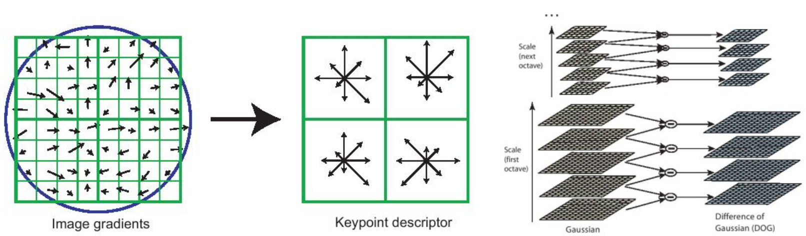
Raw image



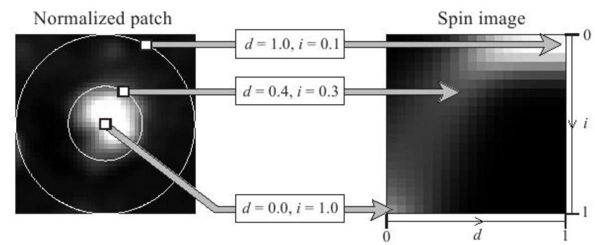
Features



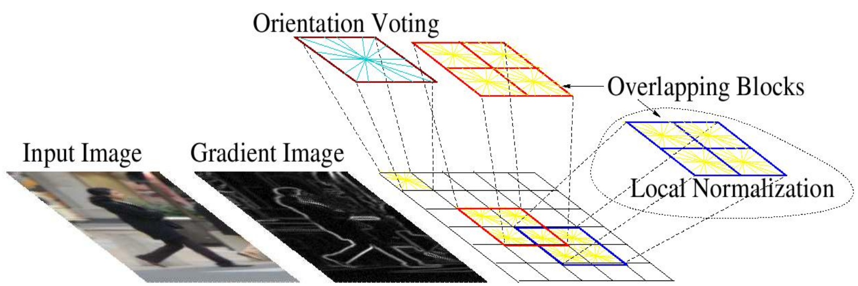
Some feature representations



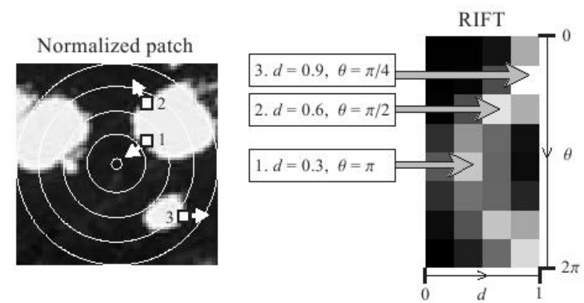
SIFT



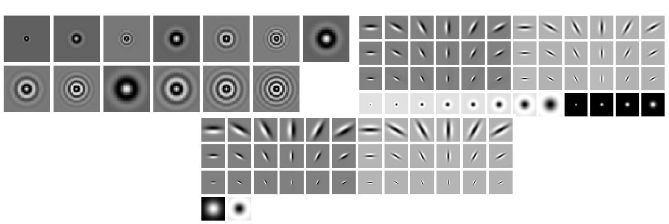
Spin image



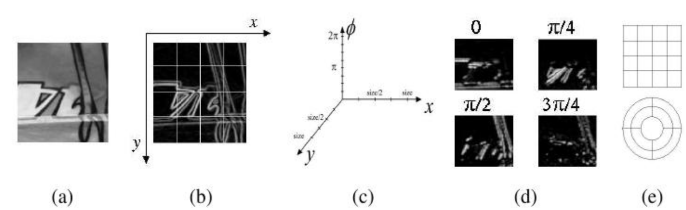
HoG



RIFT

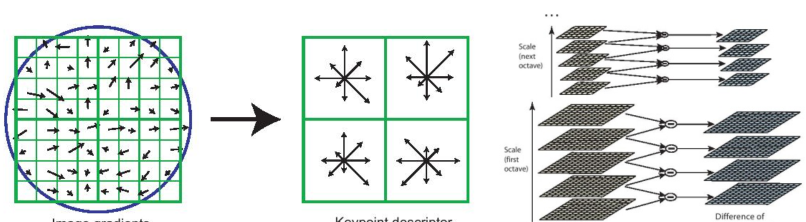


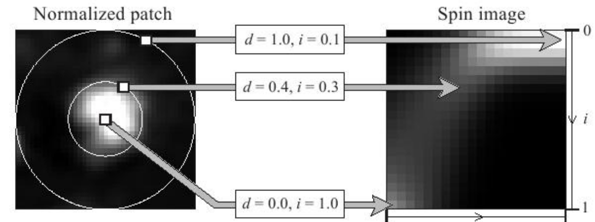
Textons



GLOH

Some feature representations

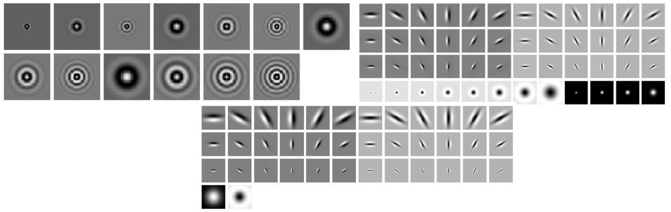




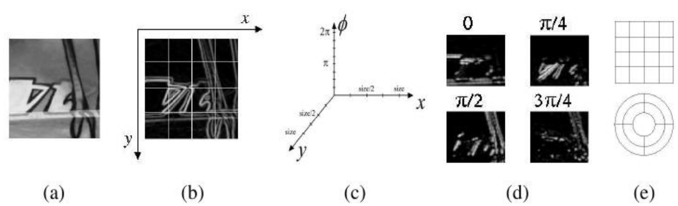
Coming up with features is often difficult, time-consuming, and requires expert knowledge.

HoG

RIFT

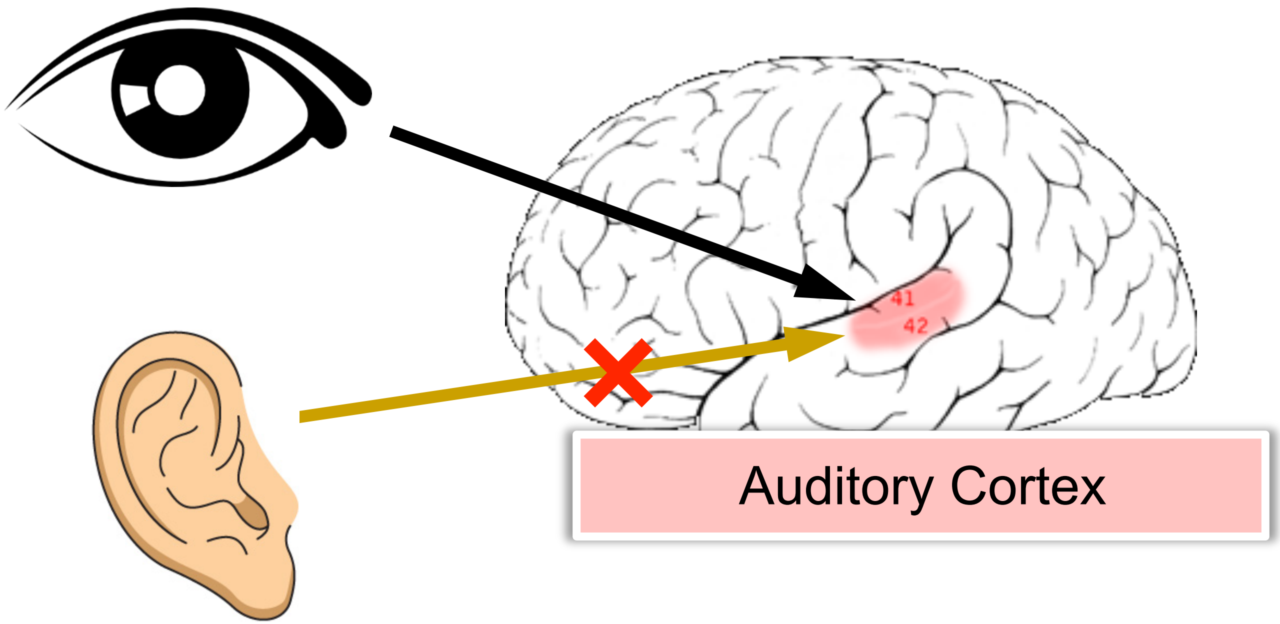


Textons



GLOH

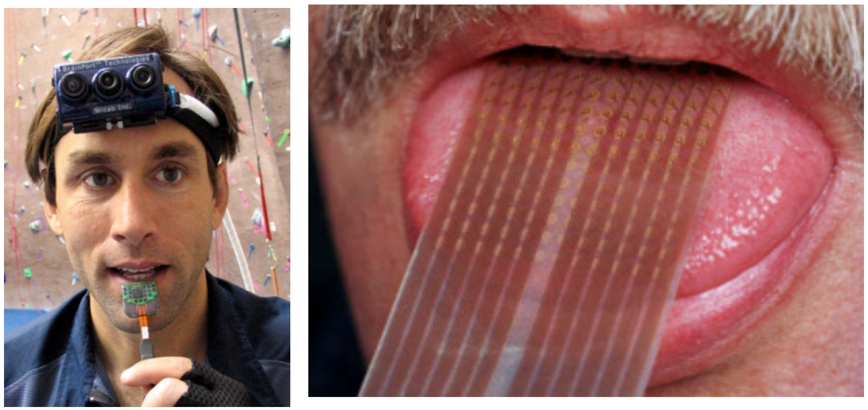
The brain:
a potential motivation for deep learning



Auditory cortex learns to see!

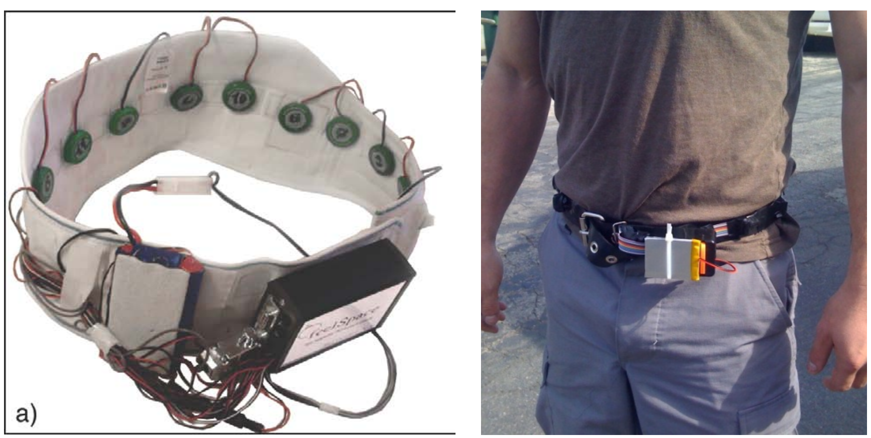
[Roe et al., 1992]

The brain adapts!

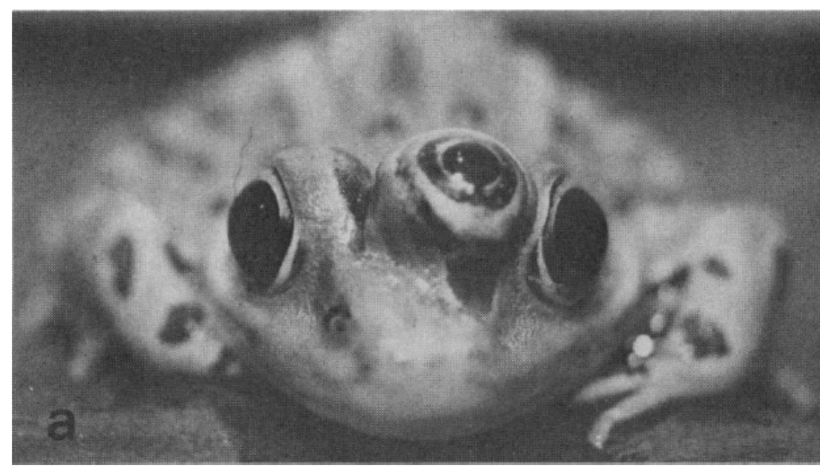


Seeing with your tongue

Human echolocation (sonar)



Haptic belt: Direction sense



Implanting a 3rd eye

[BrainPort; Welsh & Blasch, 1997; Nagel et al., 2005; Constantine-Paton & Law, 2009]

Basic idea of deep learning

- Also referred to as representation learning

- Is there some way to extract **meaningful features** from data in a *supervised* or *unsupervised* manner?

- Then, throw in some hierarchical 'stuff' to make it 'deep'

Part 2:

Supervised deep learning

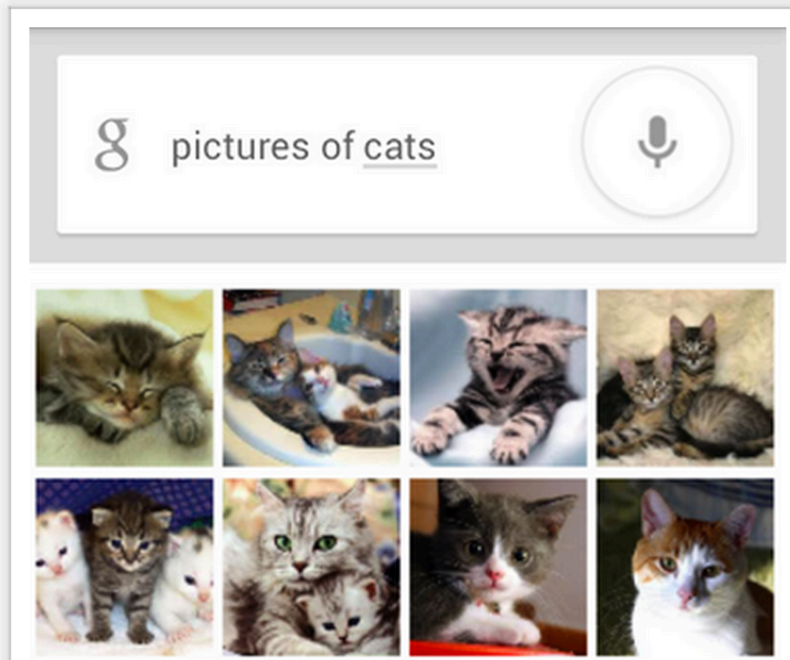
Speech recognition on Android

AUG
6

Speech Recognition and Deep Learning

Posted by Vincent Vanhoucke, Research Scientist, Speech Team

The New York Times recently published an article about Google's large scale deep learning project, which learns to discover patterns in large datasets, including... cats on YouTube!

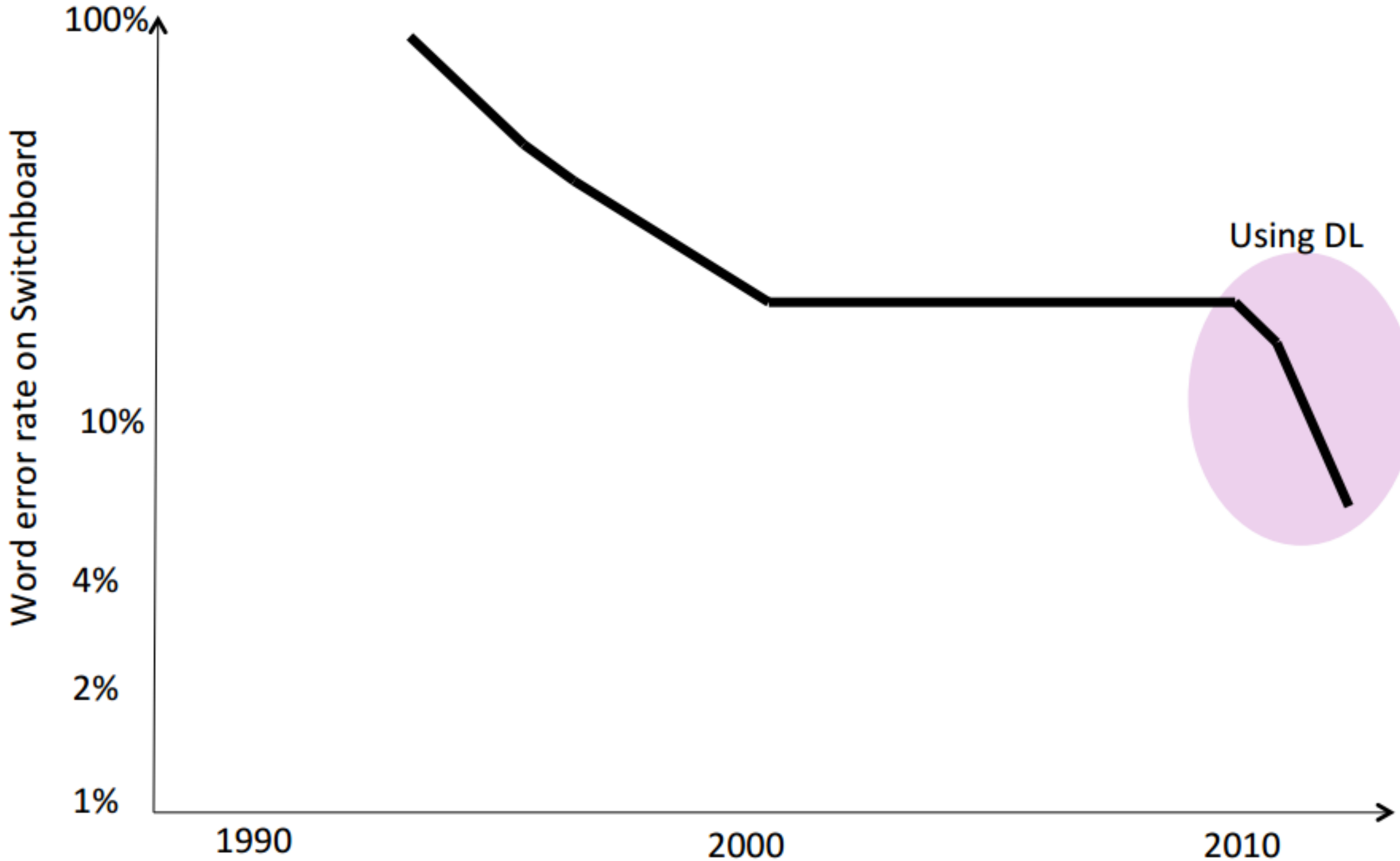


What's the point of building a gigantic cat detector you might ask? When you combine large amounts of data, large-scale distributed computing and powerful machine learning algorithms, you can apply the technology to address a large variety of practical problems.

With the launch of the latest Android platform release, Jelly Bean, we've taken a significant step towards making that technology useful: when you speak to your Android phone, chances are, you are talking to a neural network trained to recognize your speech.

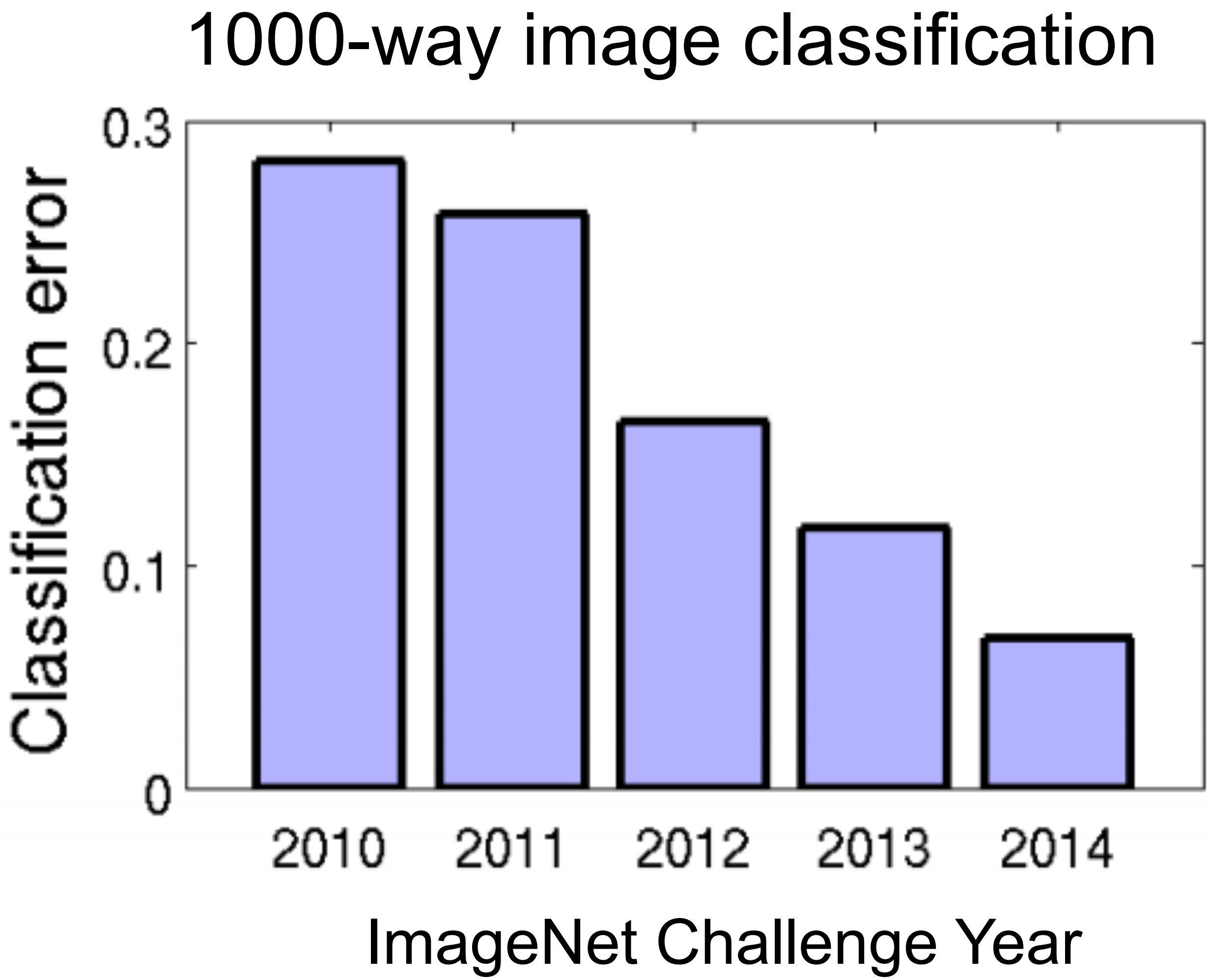
Using neural networks for speech recognition is nothing new: the first proofs of concept were developed in the late

Impact on speech recognition

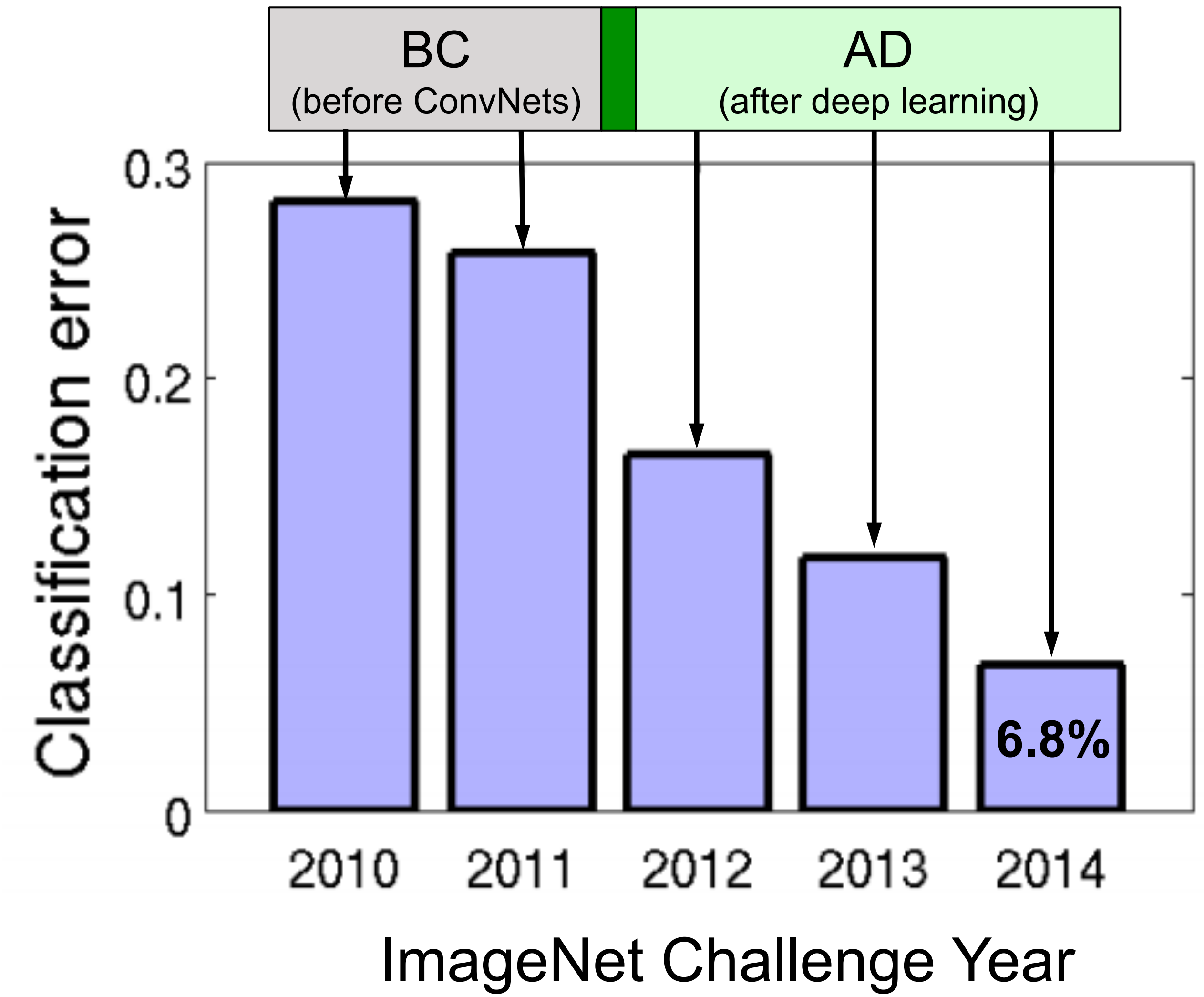


Application to Google Streetview

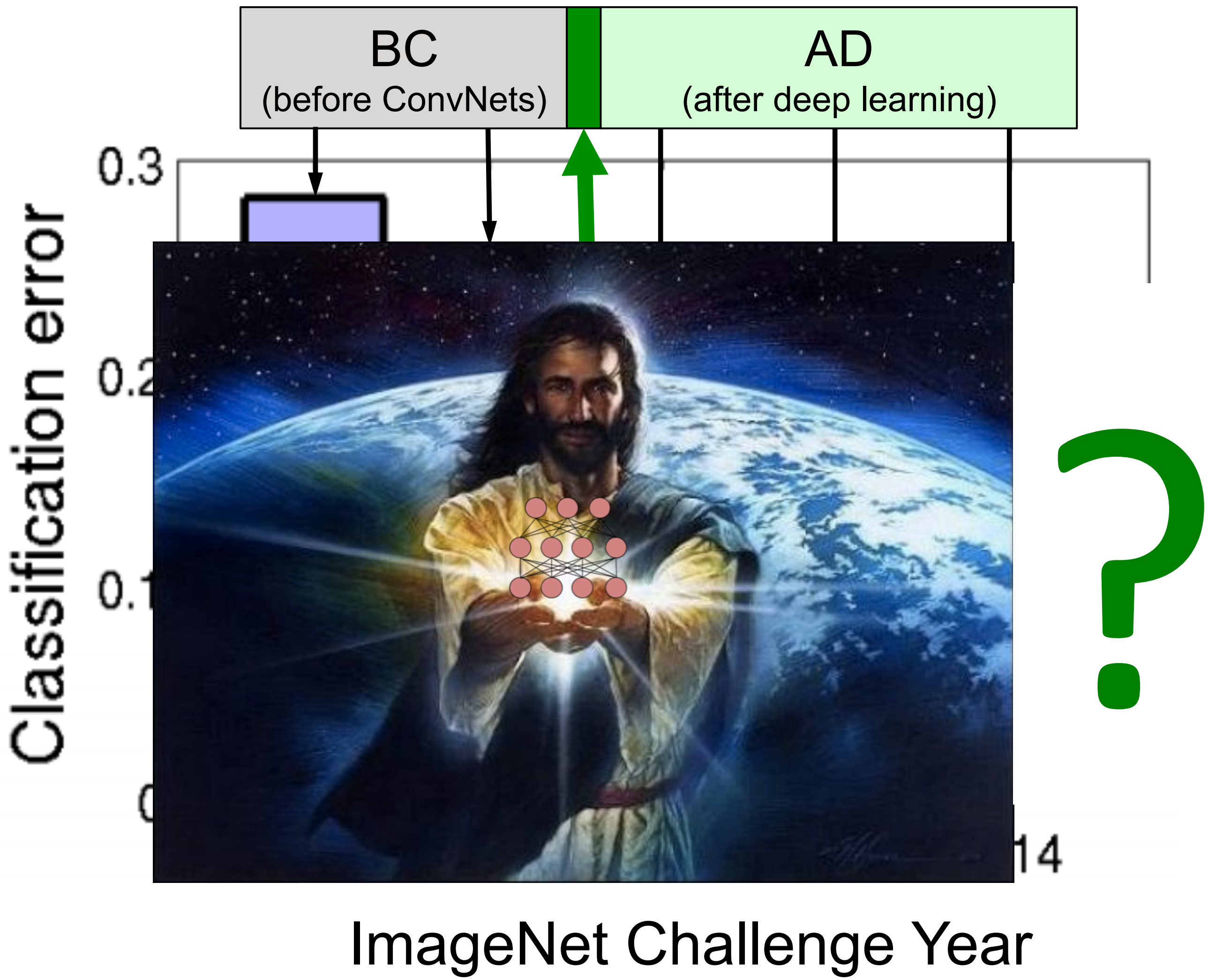
Object recognition



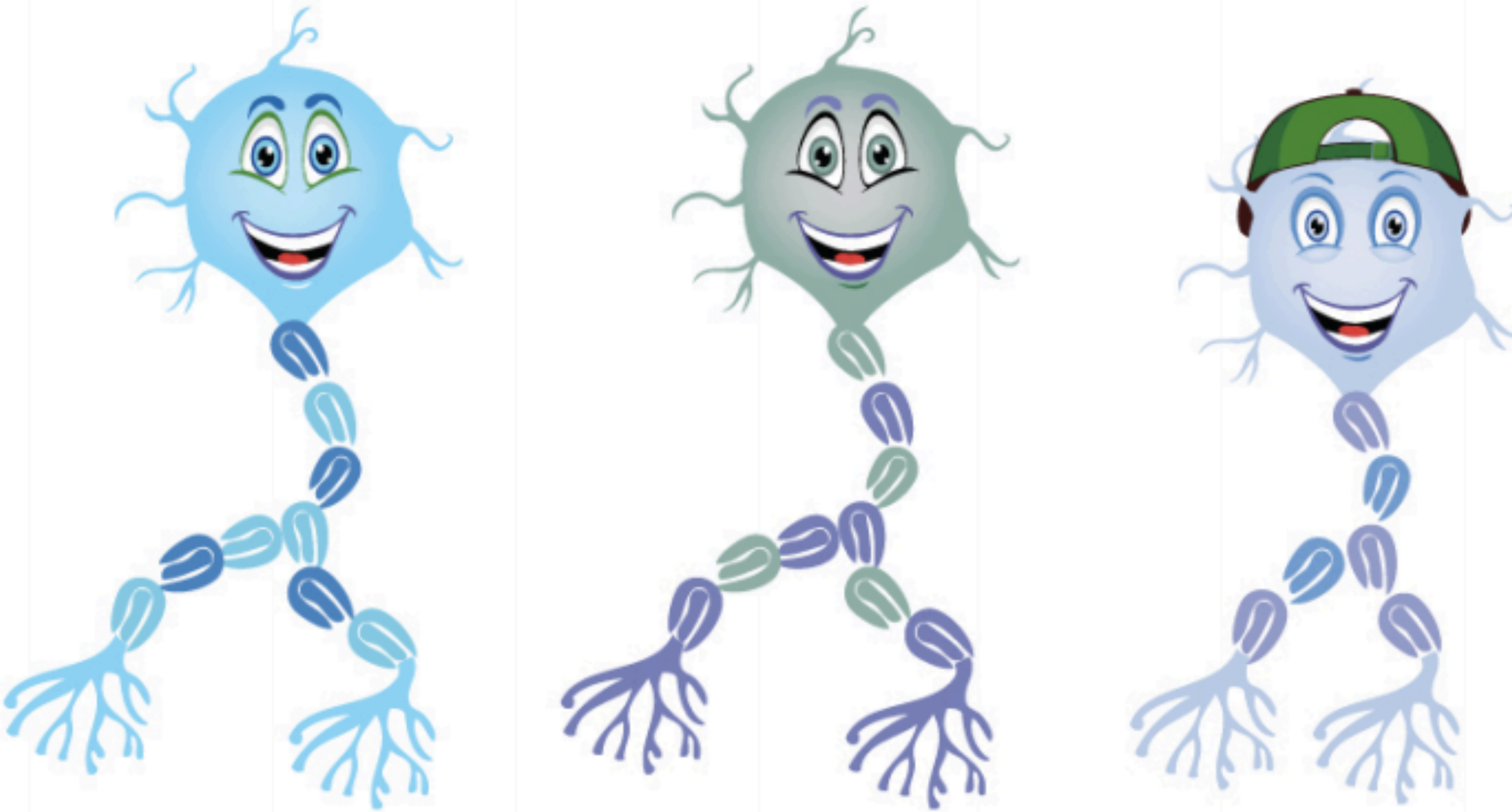
Object recognition



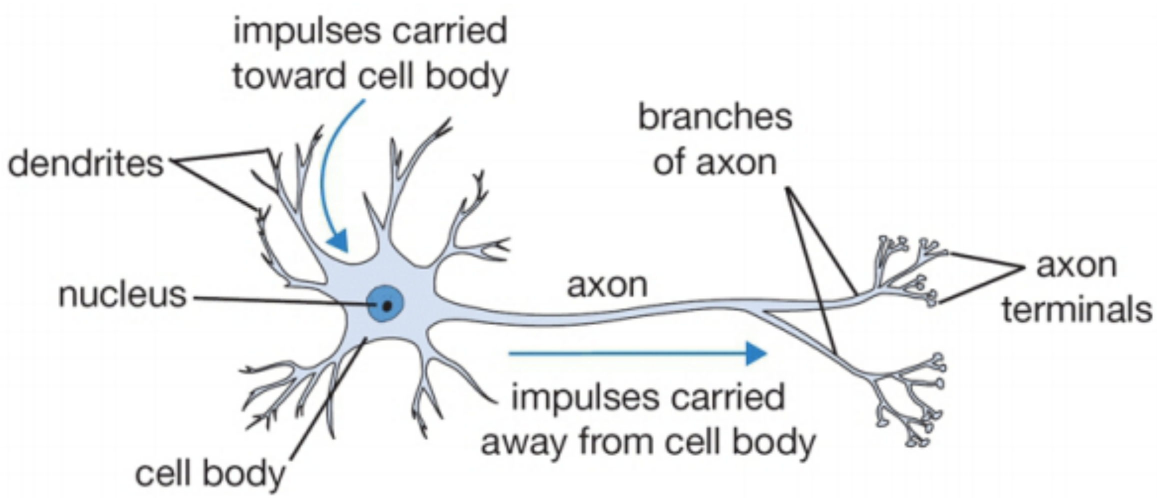
Object recognition



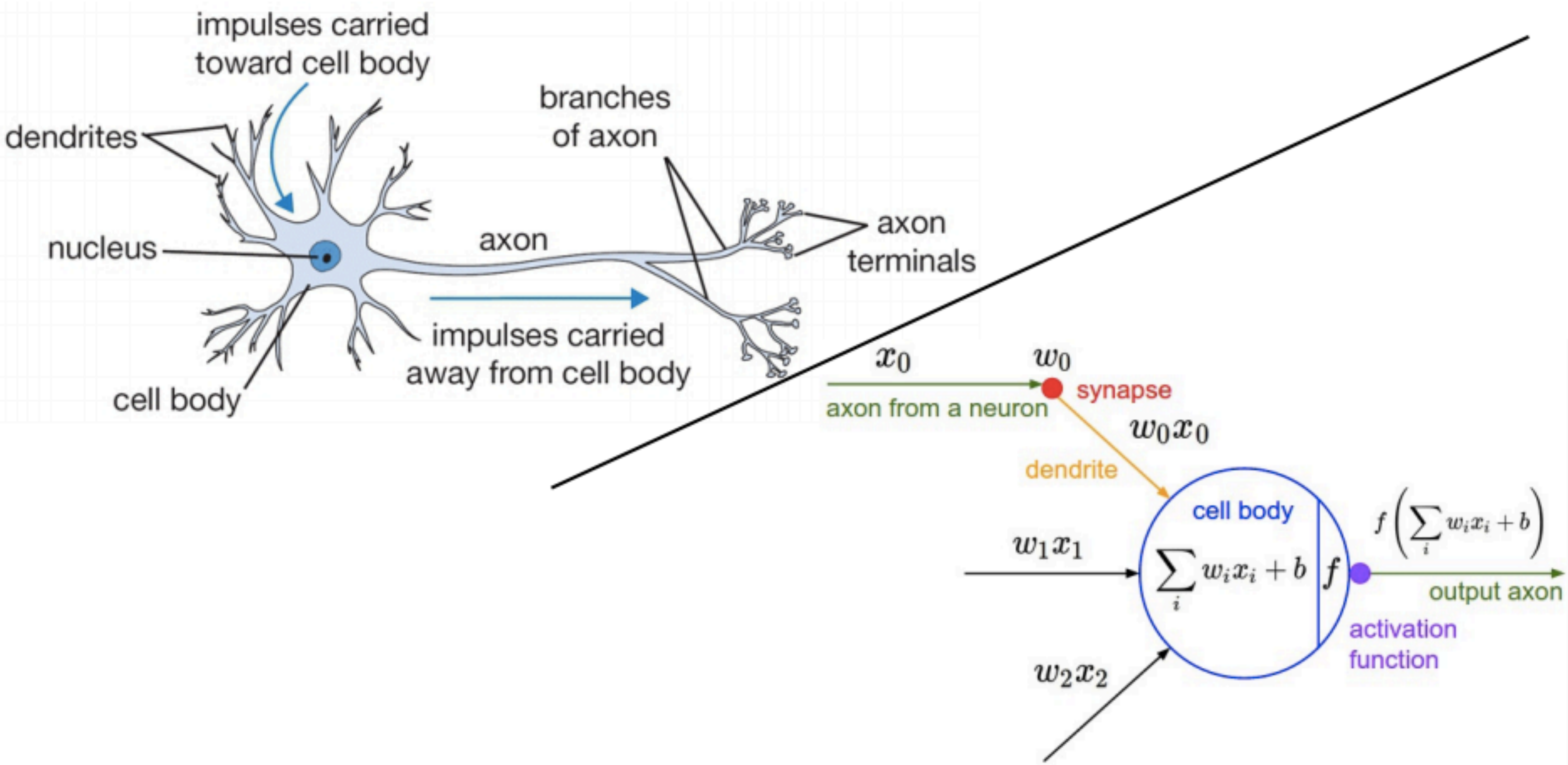
Neural networks



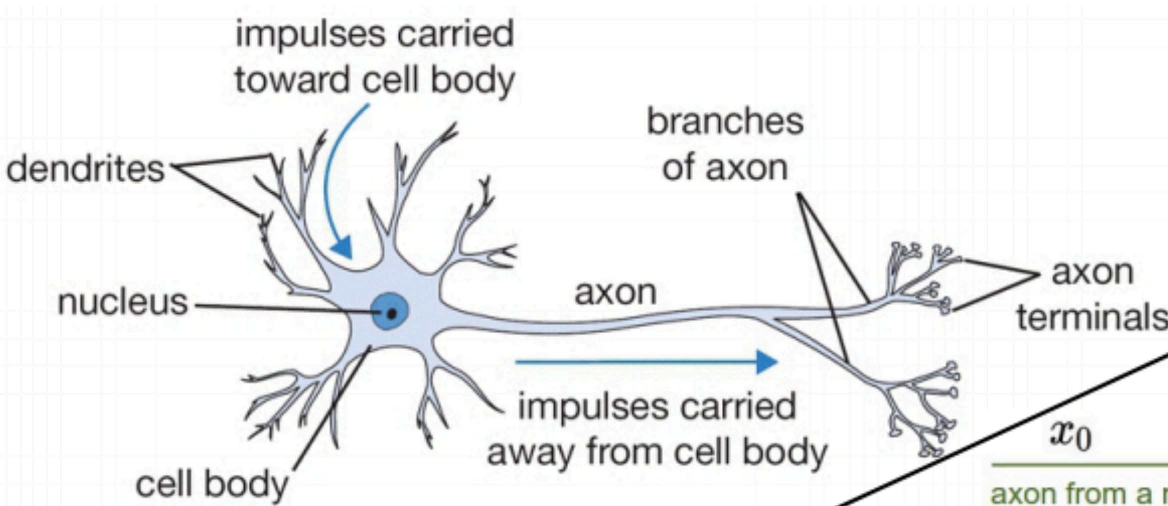
Neural networks

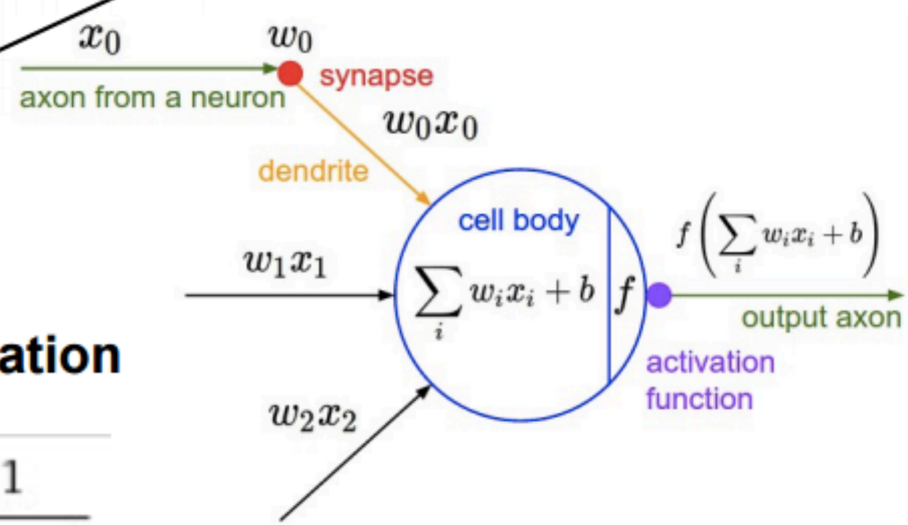


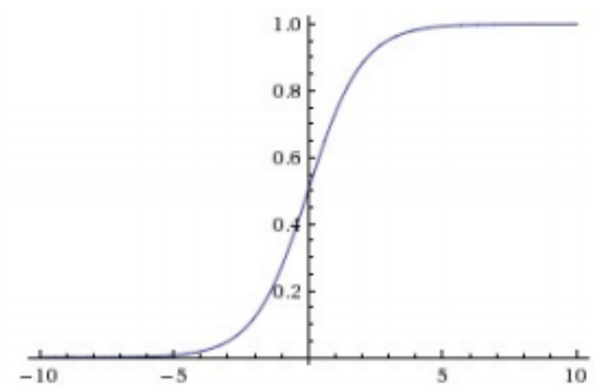
Neural networks



Neural networks





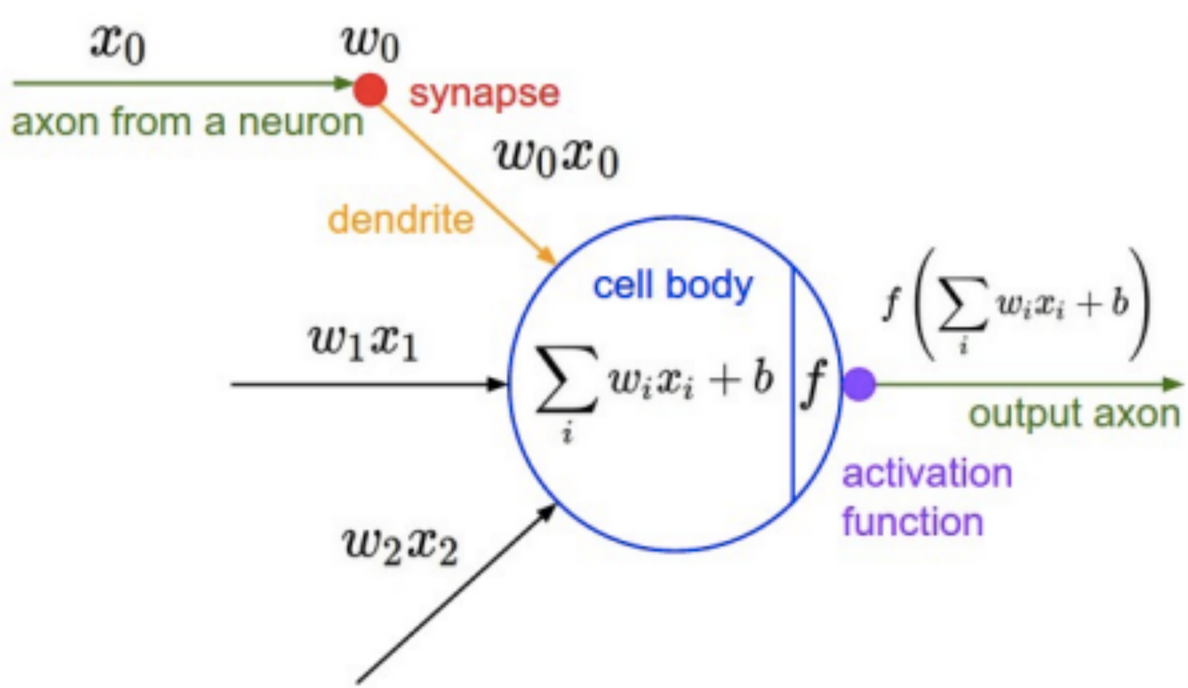


sigmoid activation function

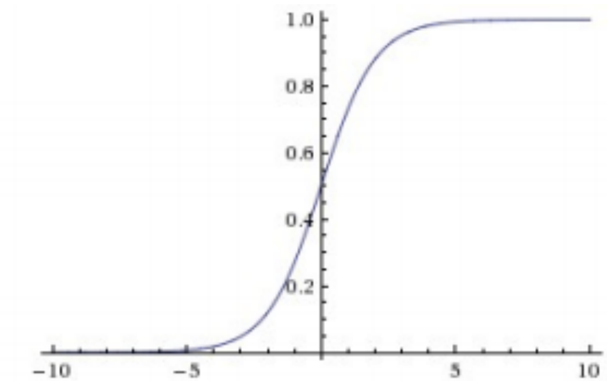
$$\frac{1}{1+e^{-x}}$$

A single neuron can be used as a binary linear classifier

e.g., logistic regression

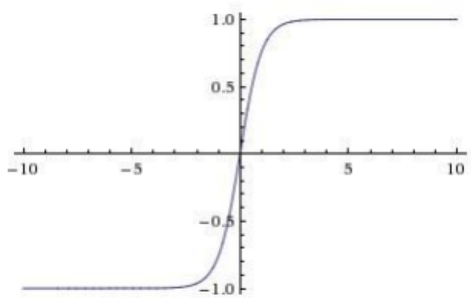


Activation functions

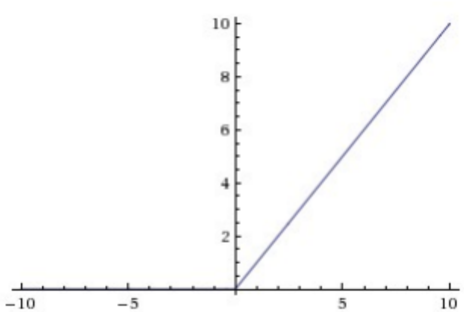


sigmoid activation function

$$\frac{1}{1+e^{-x}}$$



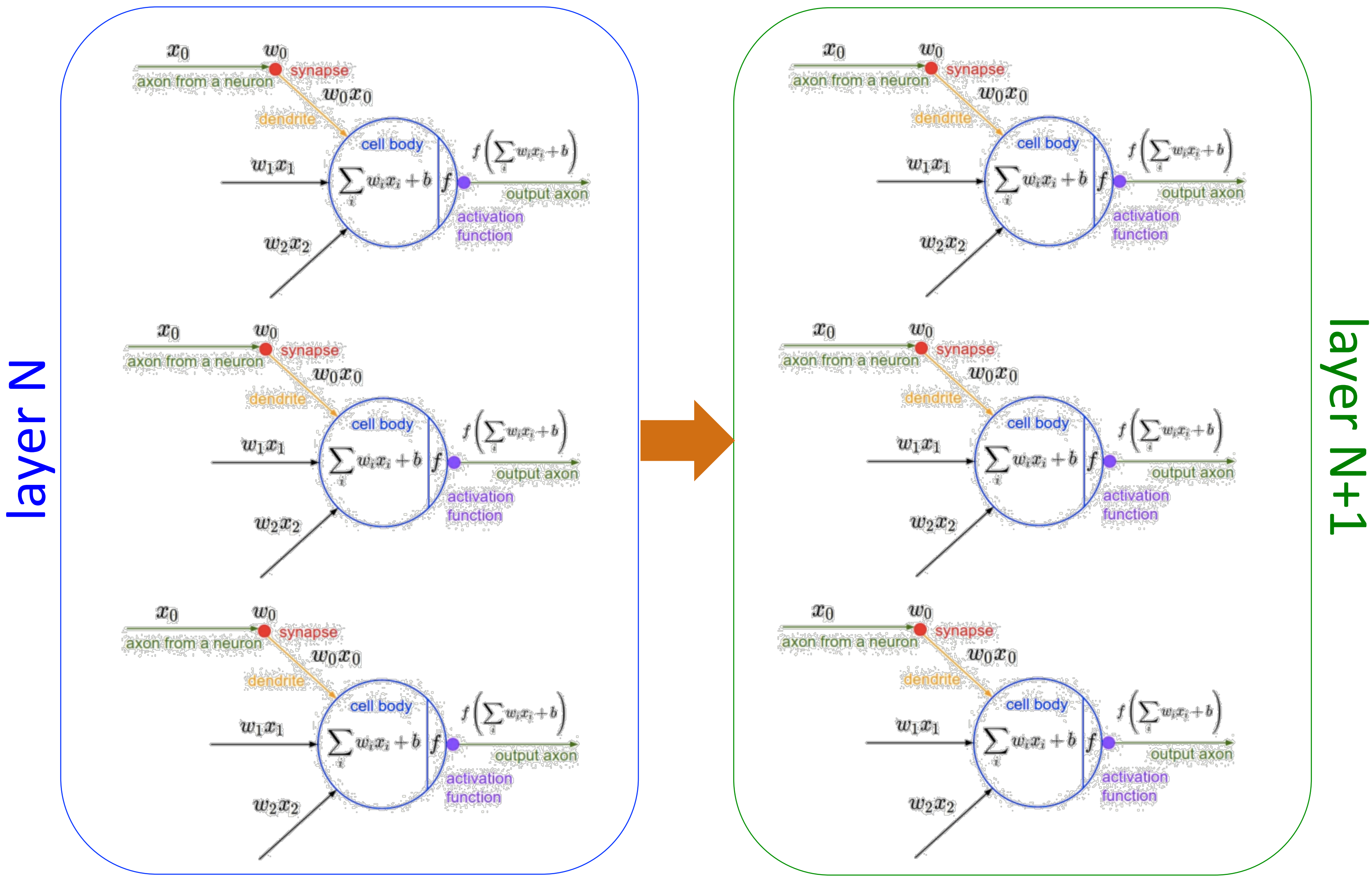
tanh(x)



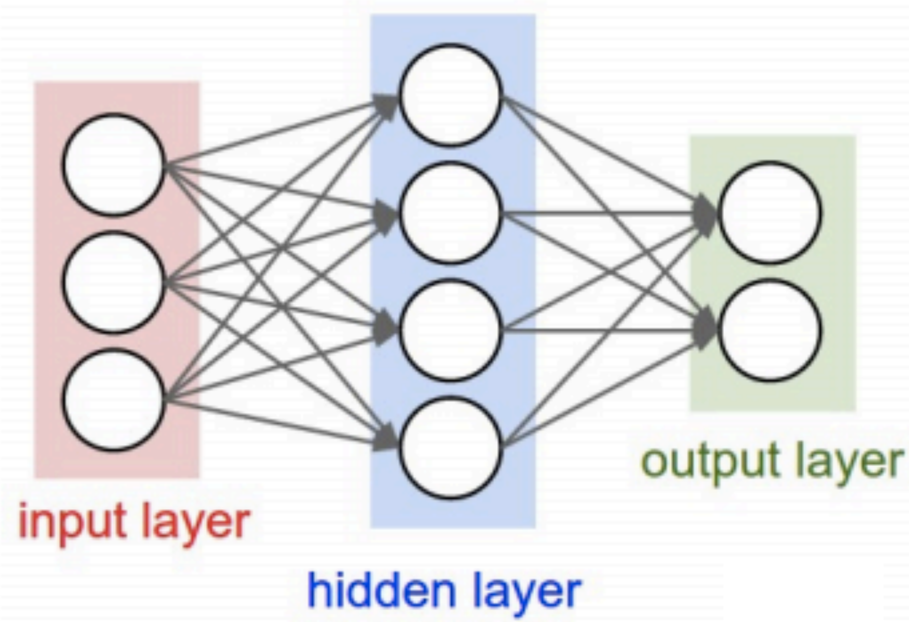
ReLU

f(x) = max(0,x)

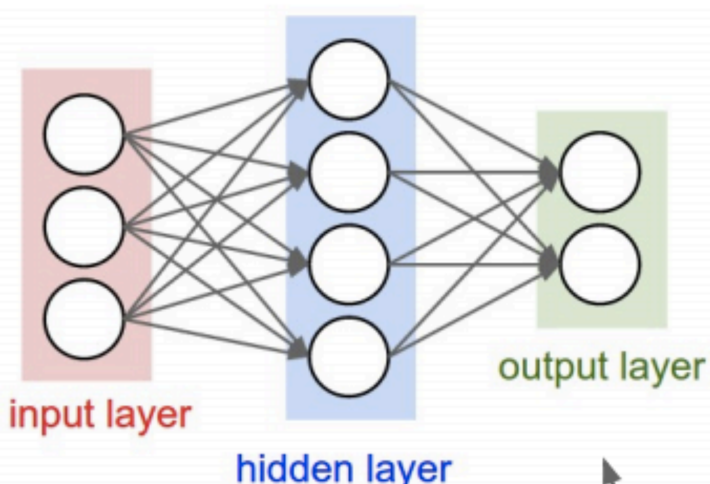
From neurons to neural network

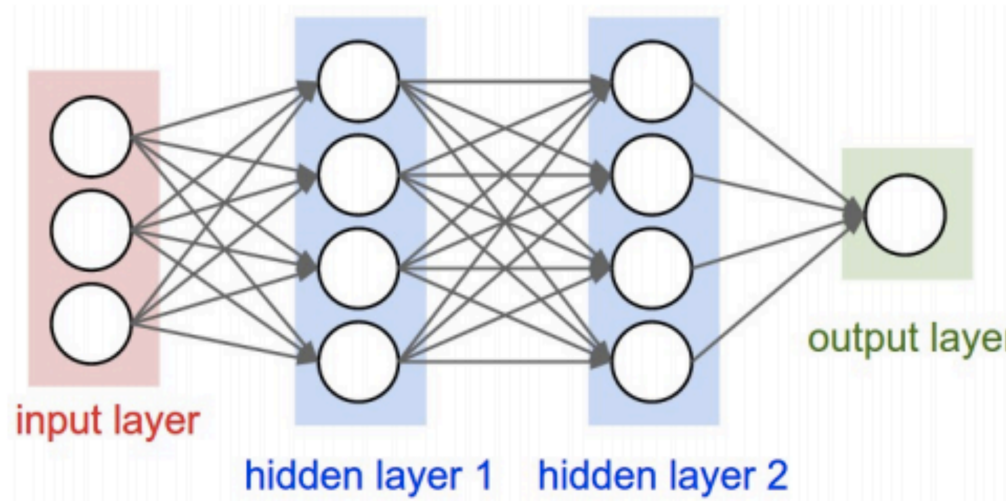


Neural networks: architectures



Neural networks: architectures



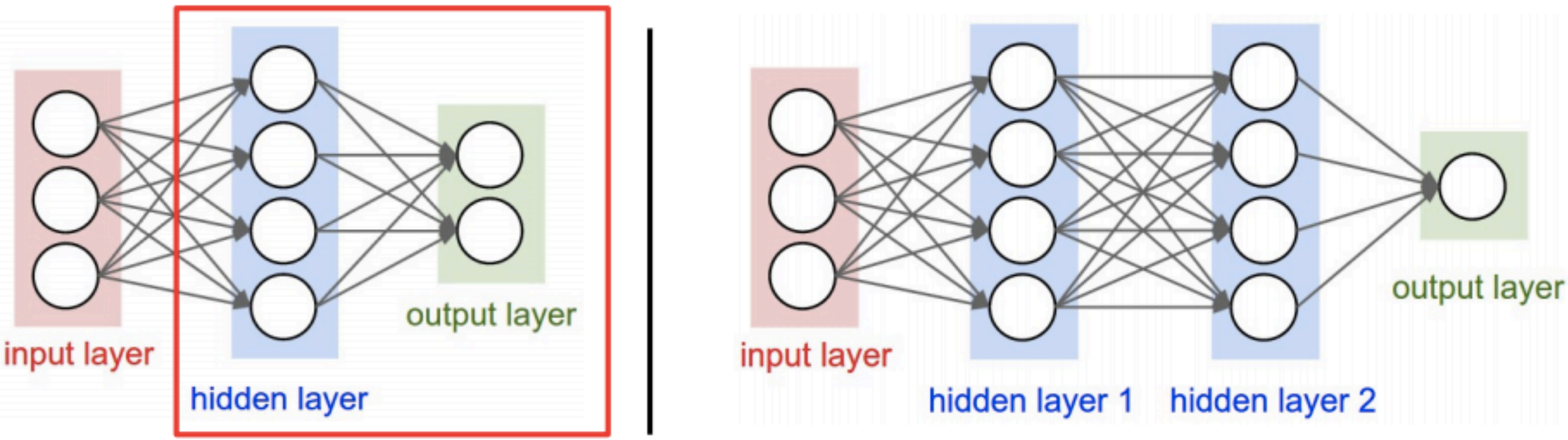


"3-layer neural net," or
"2-hidden-layer neural net"

"2-layer neural net," or
"1-hidden-layer neural net"

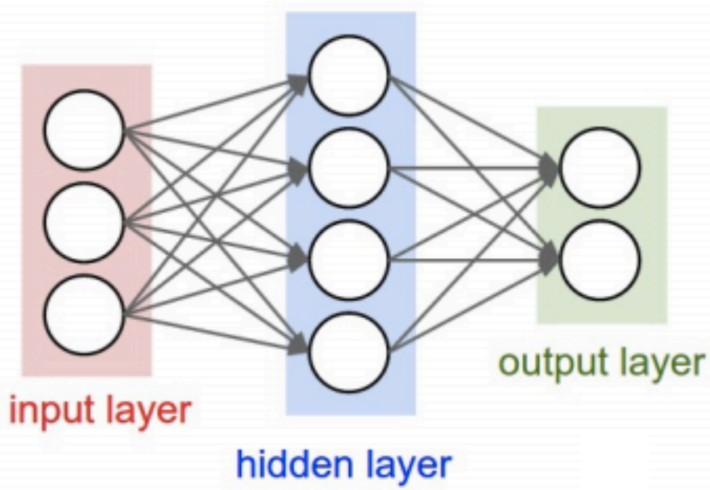
"Fully-connected" layers

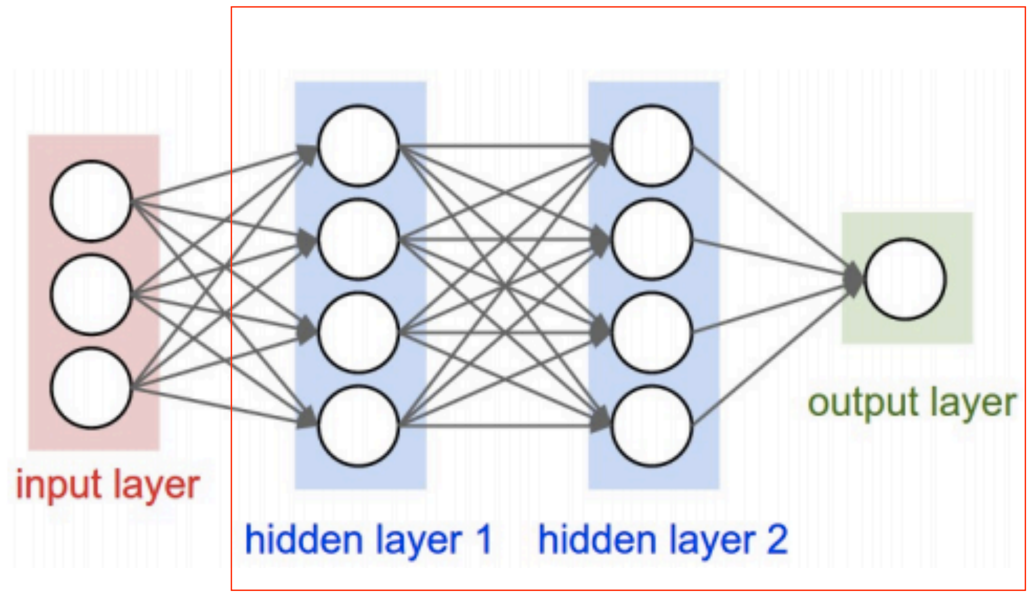
Neural network: architectures



Number of Neurons: ?
Number of Weights: ?
Number of Parameters: ?

Neural network: architectures

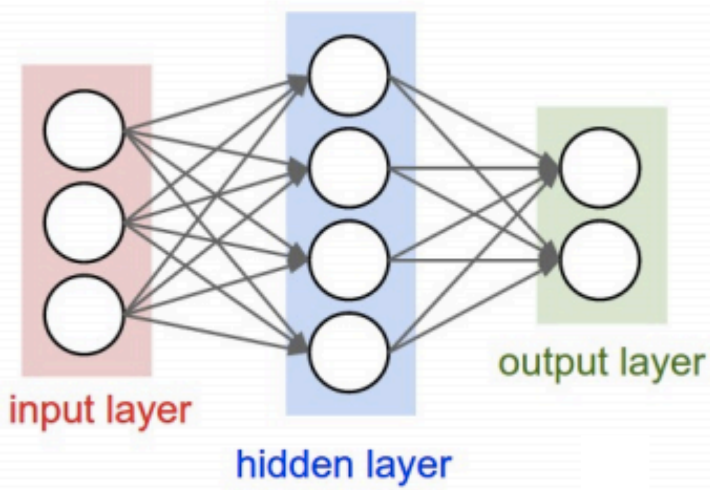


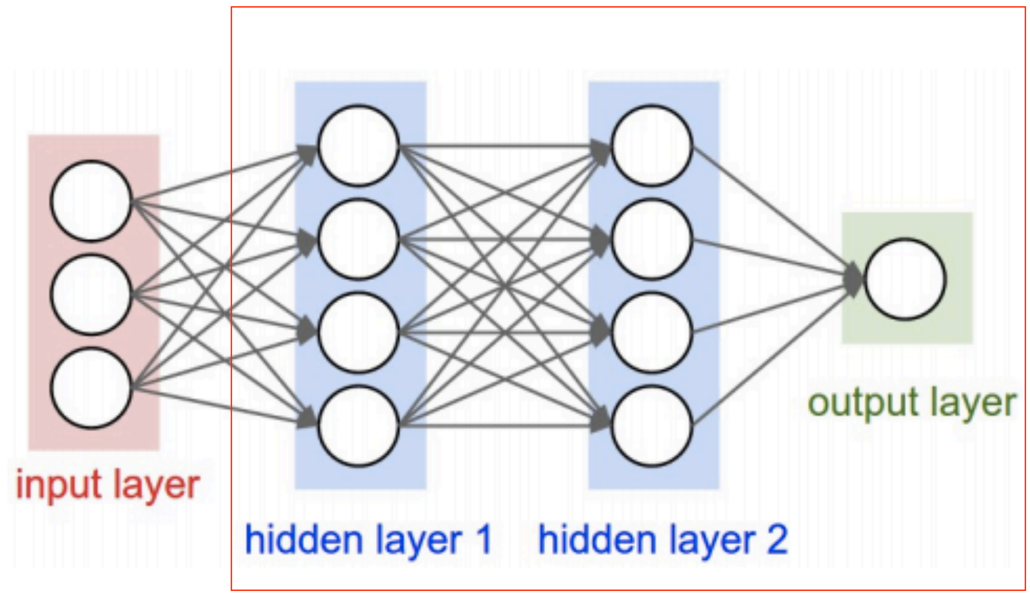


Number of Neurons: 4+2 = 6
Number of Weights: [4x3 + 2x4] = 20
Number of Parameters: 20 + 6 = 26 (biases!)

Number of Neurons: ?
Number of Weights: ?
Number of Parameters: ?

Neural network: architectures

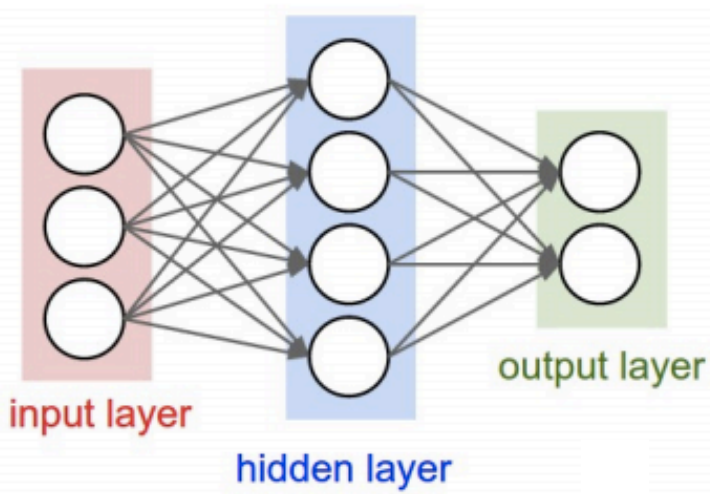


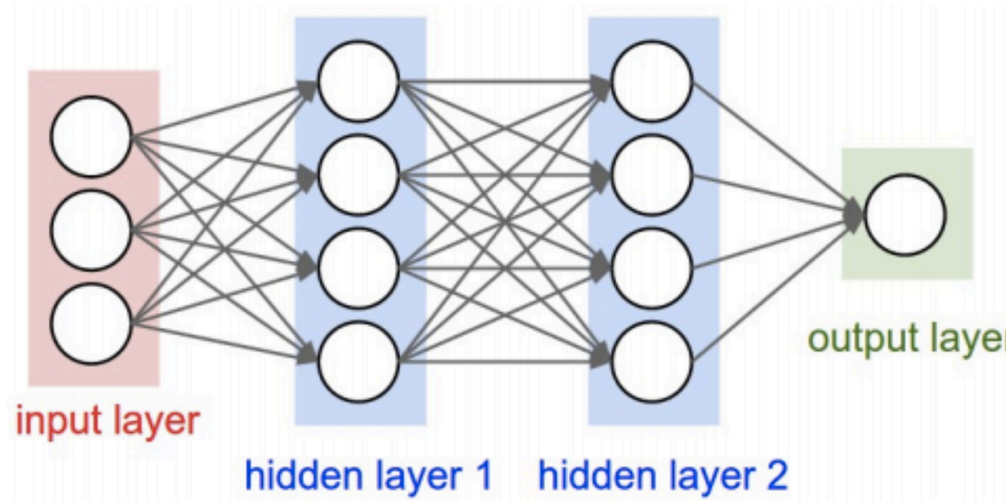


Number of Neurons: 4+2 = 6
Number of Weights: [4x3 + 2x4] = 20
Number of Parameters: 20 + 6 = 26 (biases!)

Number of Neurons: 4 + 4 + 1 = 9
Number of Weights: [4x3+4x4+1x4]=32
Number of Parameters: 32+9 = 41

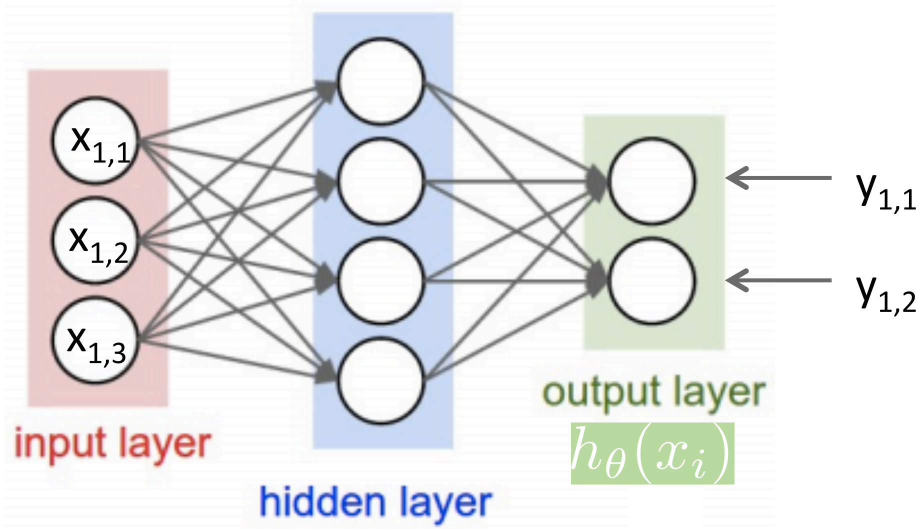
Neural network: architectures





Modern CNNs: ~10 million neurons
Human visual cortex: ~5 billion neurons

Training a neural network



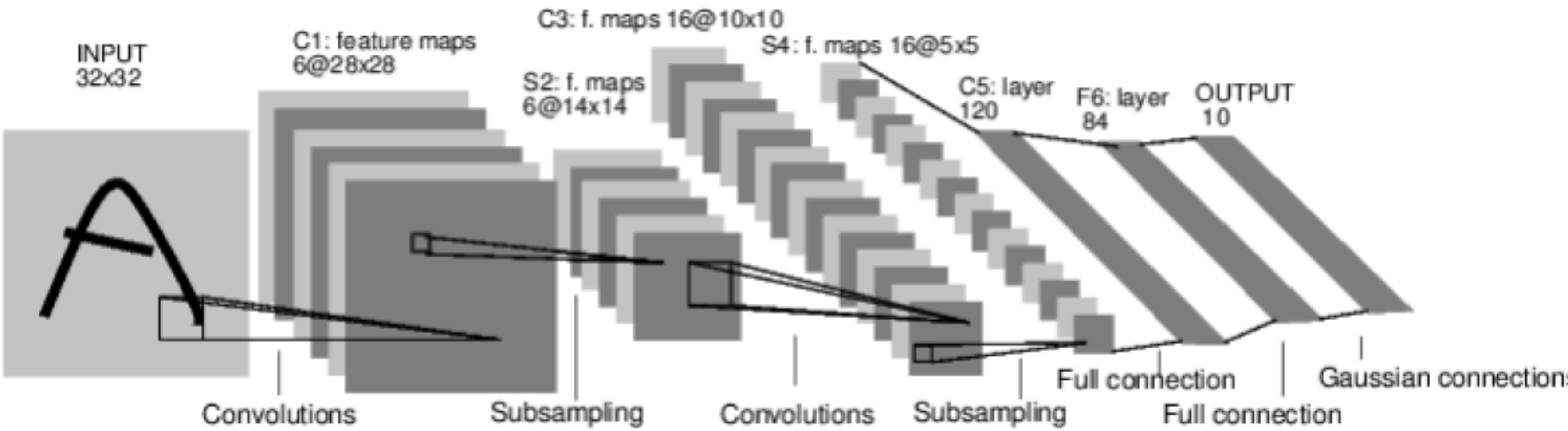
Given training set (x_1, y_1), (x_2, y_2), (x_3, y_3), ….

Adjust parameters θ (for every node) to make:

$$h_\theta(x_i) \approx y_i$$

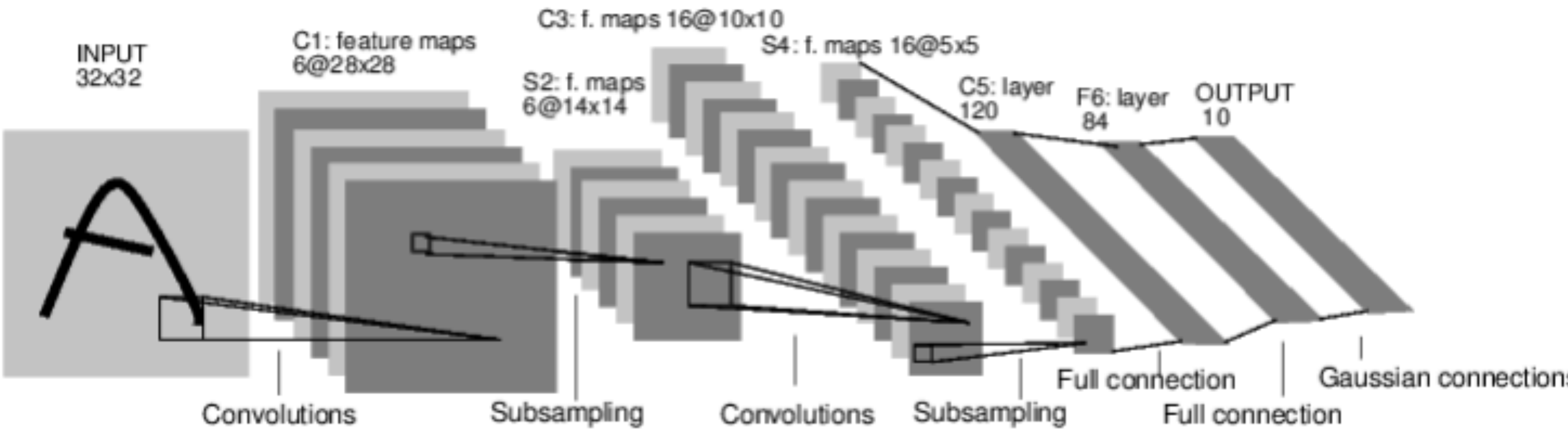
(Use gradient descent - "Backpropagation" algorithm)

Convolutional neural networks



[LeNet-5, LeCun 1980]

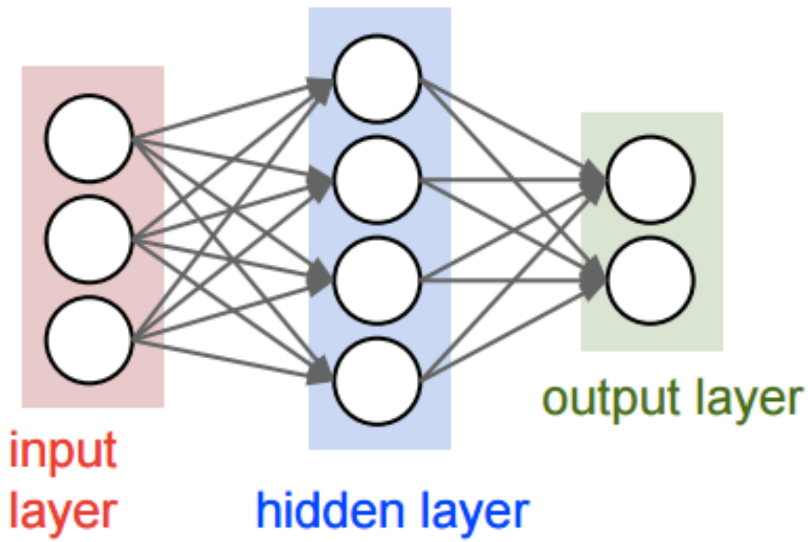
Convolutional neural networks
aka ConvNets
aka CNNs
aka Computer vision Savior



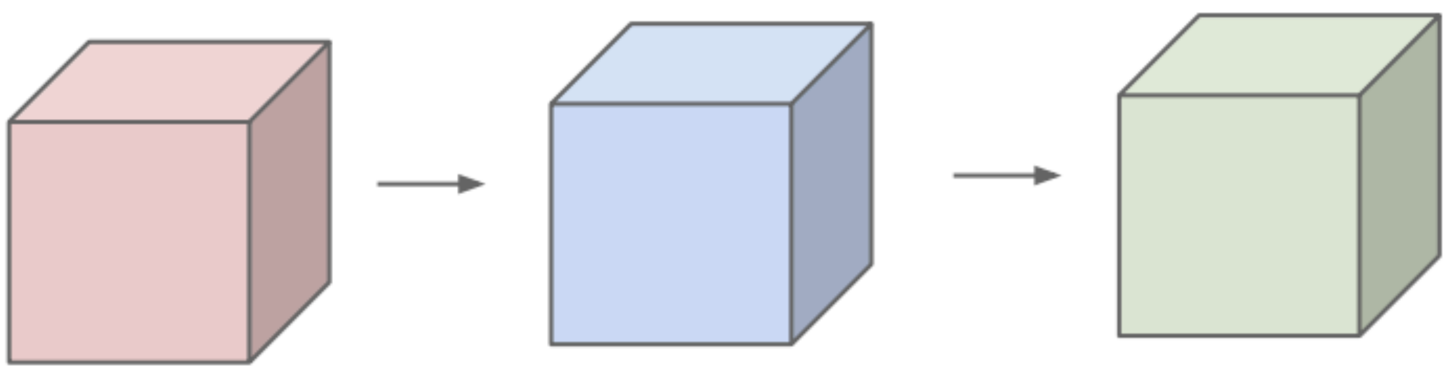
[LeNet-5, LeCun 1980]

ConvNets: architecture

before:

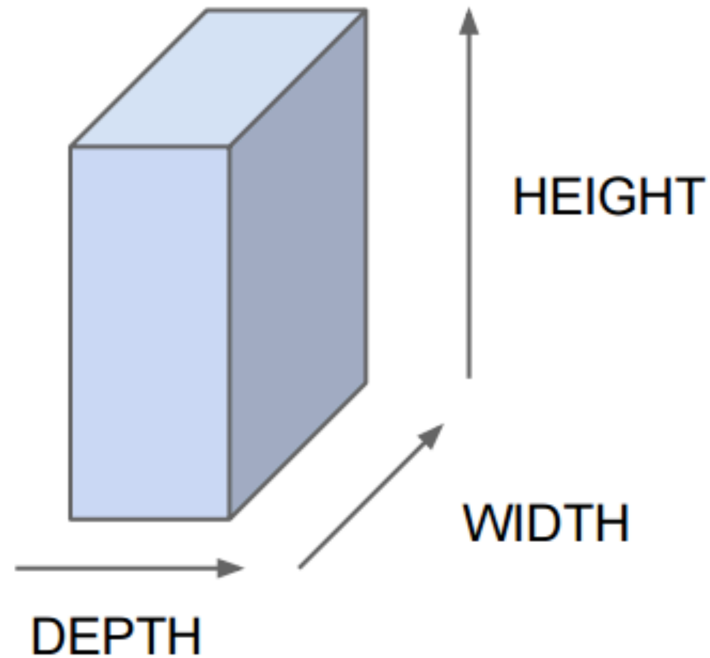


now:



ConvNets: architecture

All Neural Net activations arranged in **3 dimensions:**

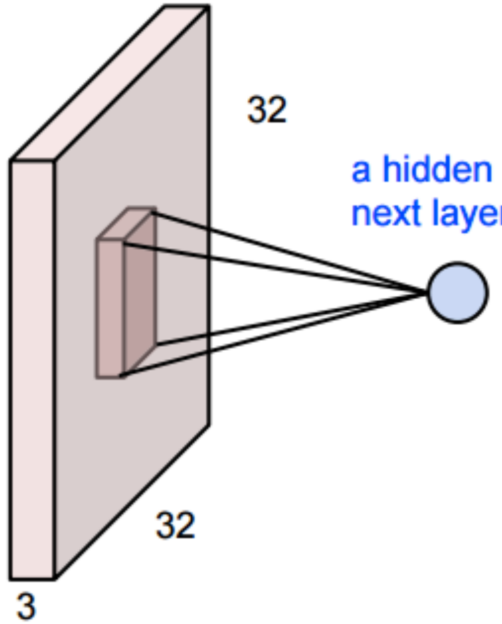


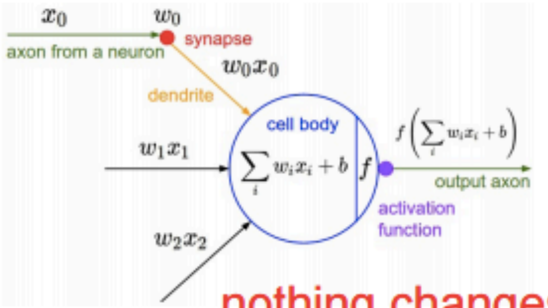
For example, a CIFAR-10 image is a 32x32x3 volume
32 width, 32 height, 3 depth (RGB channels)

ConvNets: architecture

Convolutional Neural Networks
are just Neural Networks BUT:

1. **Local connectivity**



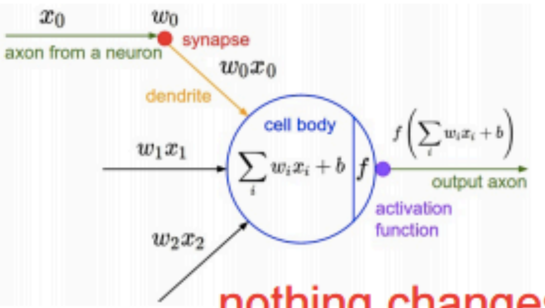


nothing changes

ConvNets: architecture

Convolutional Neural Networks
are just Neural Networks BUT:

1. Local connectivity



nothing changes

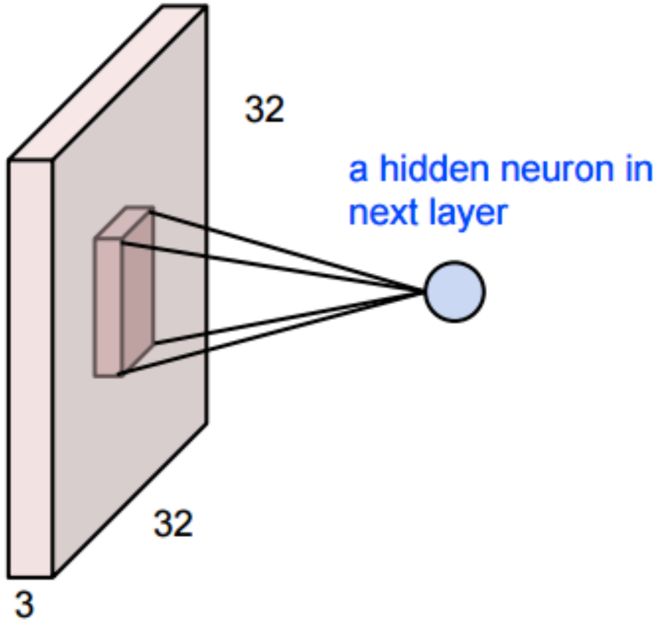


image: 32x32x3 volume
before: full connectivity: 32x32x3 weights

now: one neuron will connect to, e.g. 5x5x3 chunk and only have 5x5x3 weights.

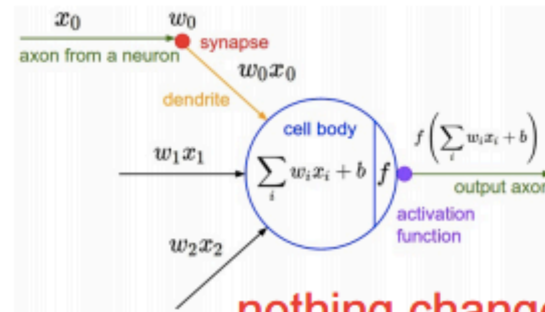
note that connectivity is:

- <u>local in space</u> (5x5 inside 32x32)
- <u>but full in depth</u> (all 3 depth channels)

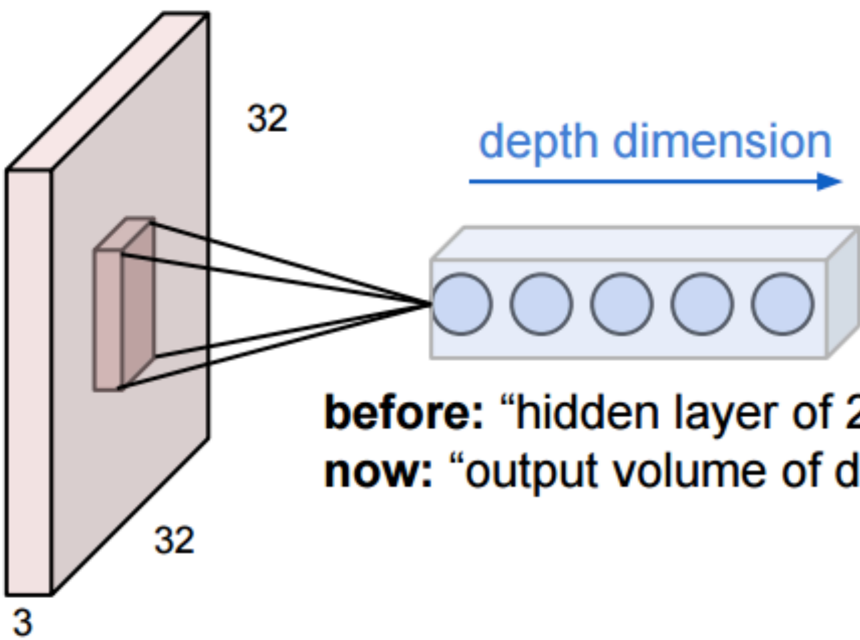
ConvNets: architecture

Convolutional Neural Networks are just Neural Networks BUT:

1. **Local connectivity**



nothing changes



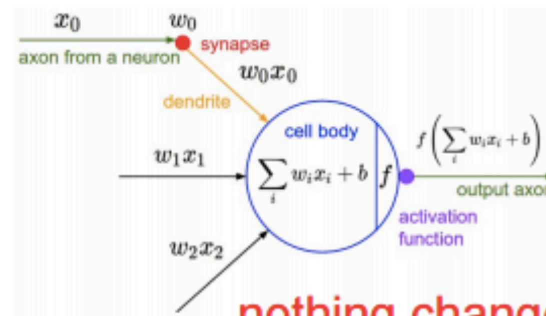
before: "hidden layer of 200 neurons"
now: "output volume of depth 200"

Multiple neurons all looking at the same region of the input volume, stacked along depth.

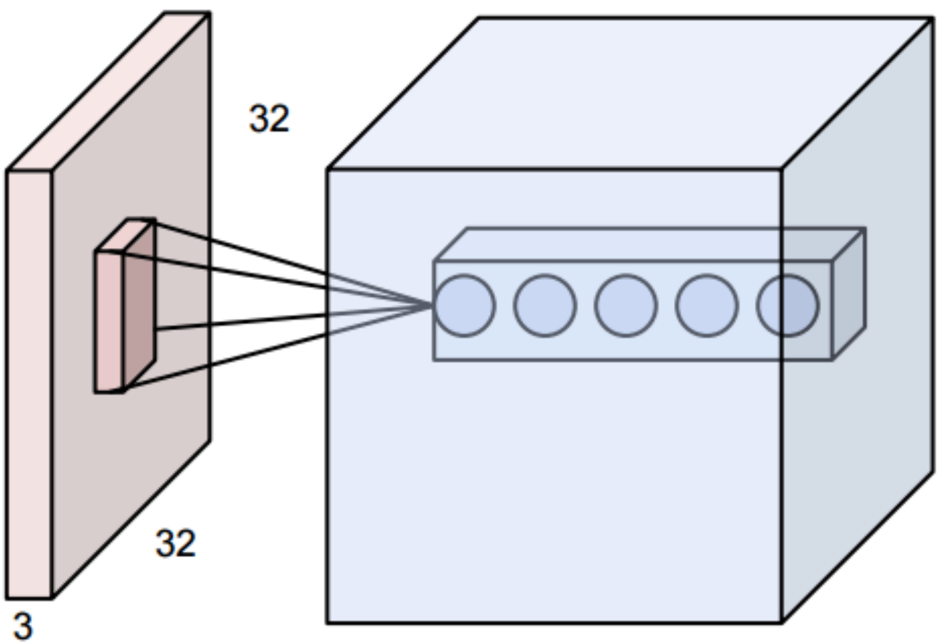
ConvNets: architecture

Convolutional Neural Networks are just Neural Networks BUT:

1. **Local connectivity**



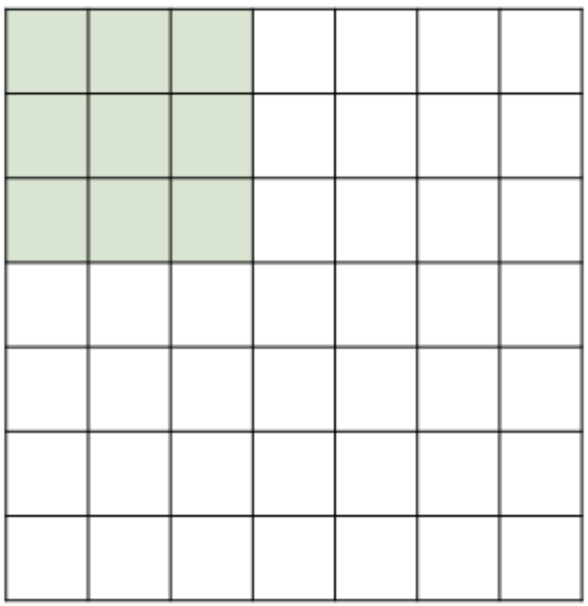
nothing changes



These form a single [1 x 1 x depth] “depth column” in the output volume

ConvNets: architecture

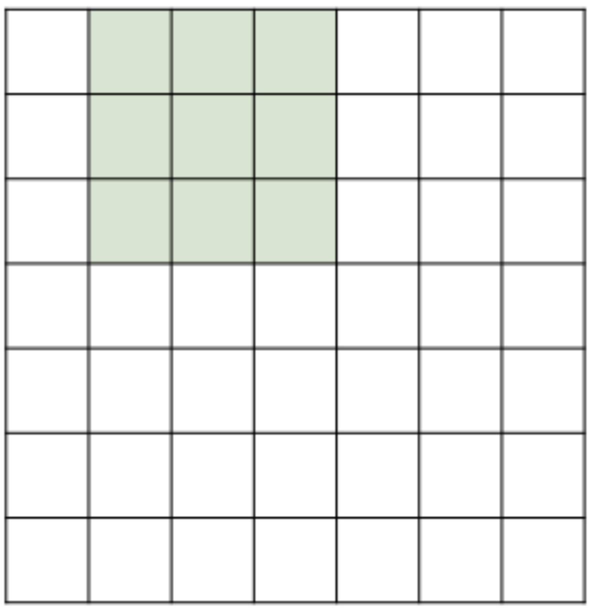
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1

ConvNets: architecture

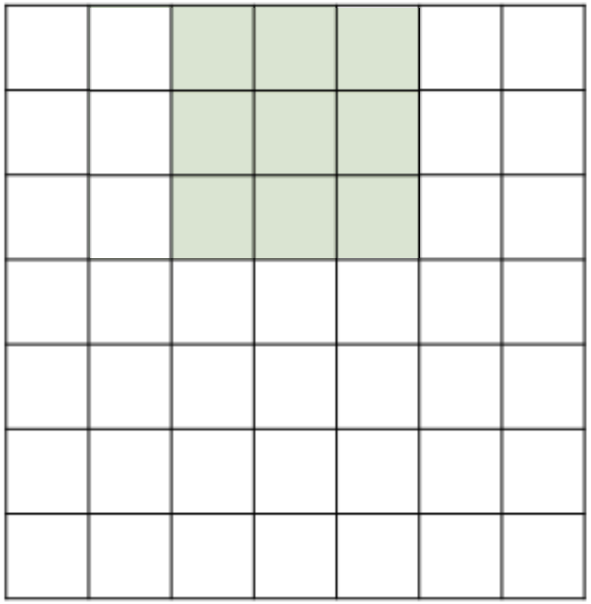
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1

ConvNets: architecture

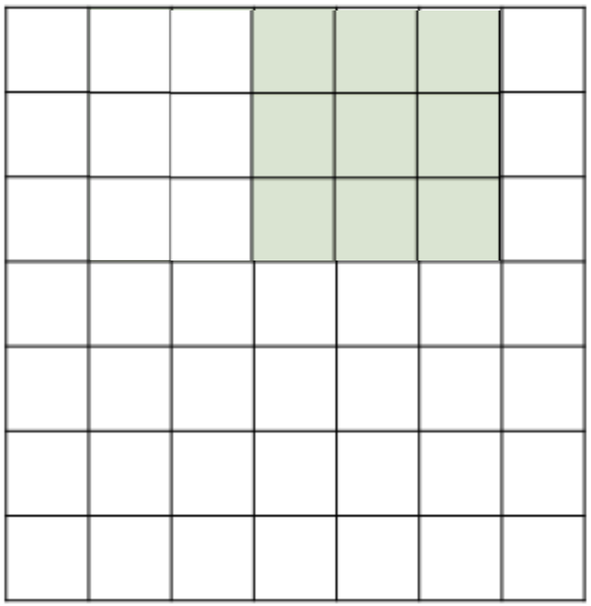
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1

ConvNets: architecture

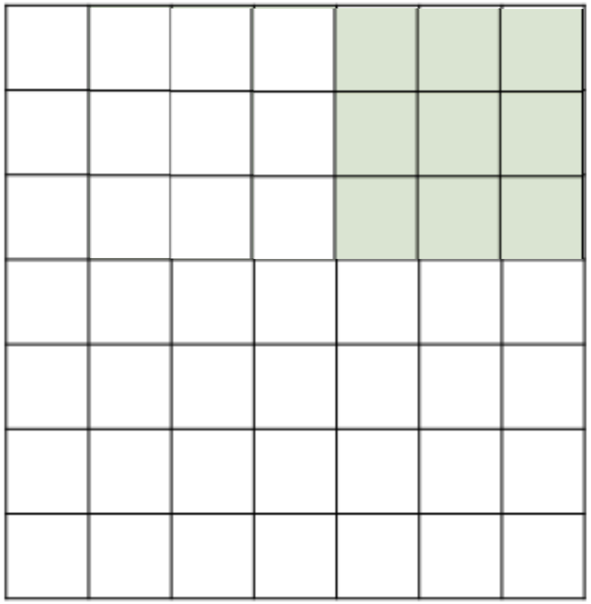
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1

ConvNets: architecture

Replicate this column of hidden neurons across space, with some **stride.**

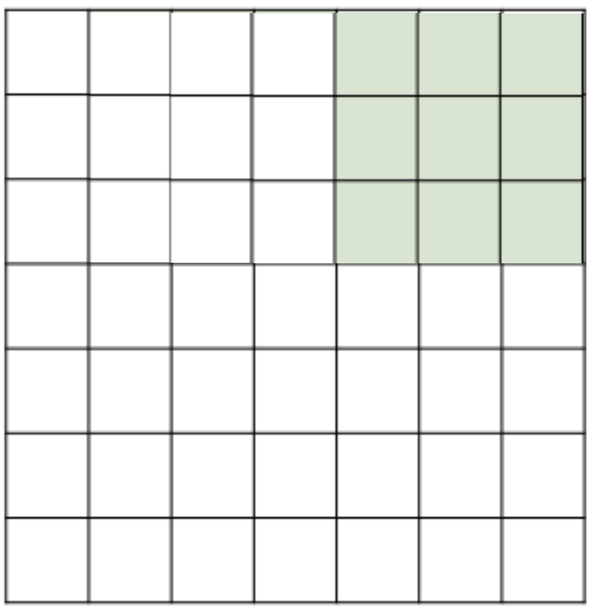


7x7 input
assume 3x3 connectivity, stride 1

--> 5x5 output

ConvNets: architecture

Replicate this column of hidden neurons across space, with some **stride.**



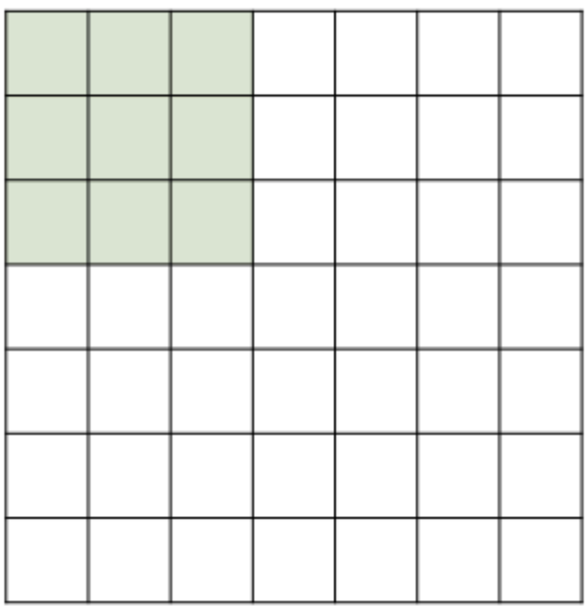
7x7 input
assume 3x3 connectivity, stride 1

--> 5x5 output

What about stride 2?

ConvNets: architecture

Replicate this column of hidden neurons across space, with some **stride.**



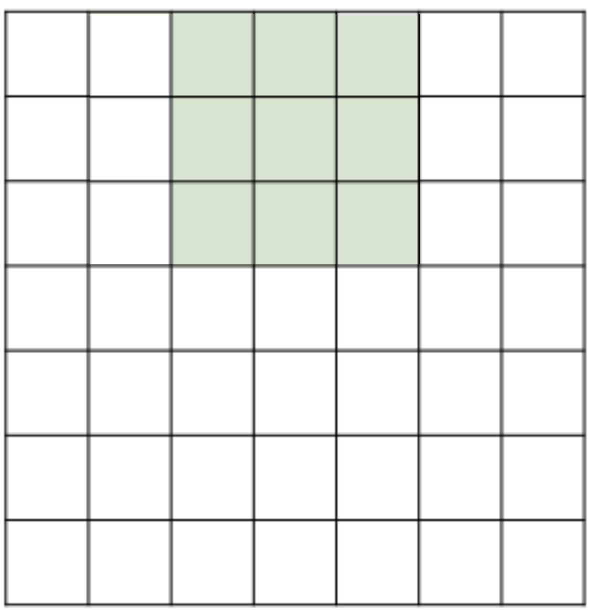
7x7 input
assume 3x3 connectivity, stride 1

--> 5x5 output

What about stride 2?

ConvNets: architecture

Replicate this column of hidden neurons across space, with some **stride.**



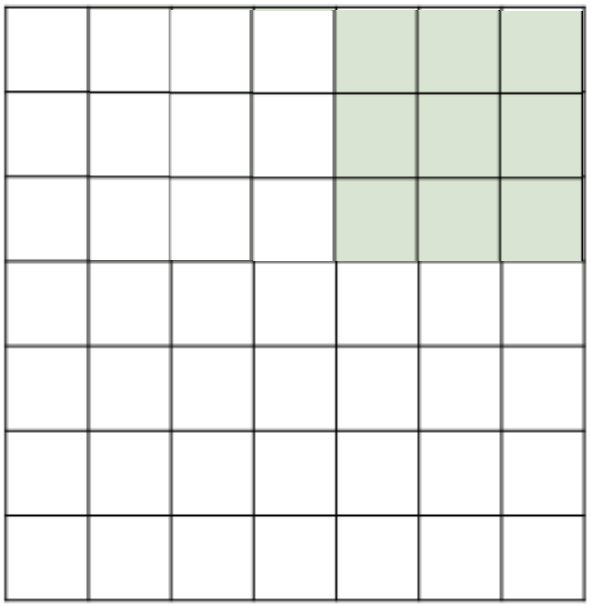
7x7 input
assume 3x3 connectivity, stride 1

--> 5x5 output

What about stride 2?

ConvNets: architecture

Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1

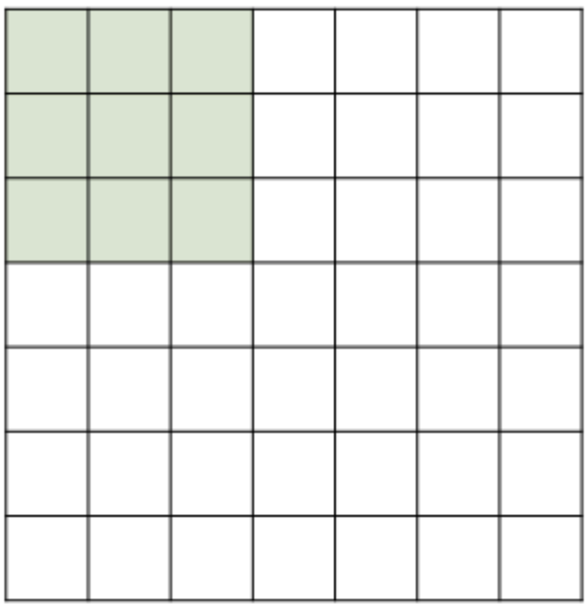
--> 5x5 output

7x7 input
assume 3x3 connectivity, stride 2

--> 3x3 output

ConvNets: architecture

Replicate this column of hidden neurons across space, with some **stride.**



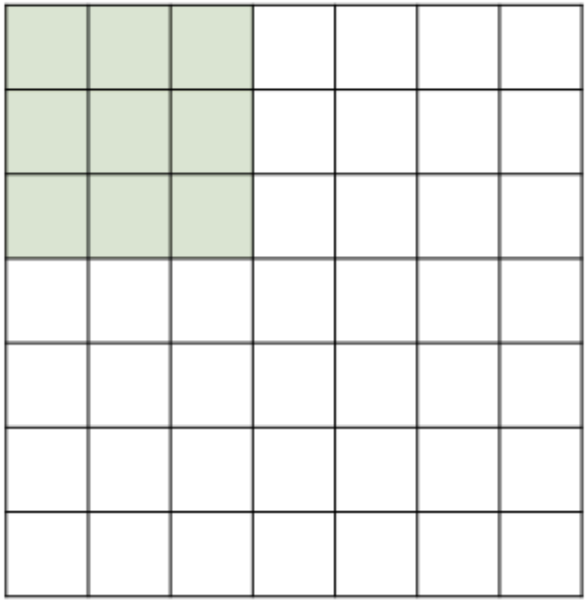
7x7 input
assume 3x3 connectivity, stride 1

--> 5x5 output

What about stride 3?

ConvNets: architecture

Replicate this column of hidden neurons across space, with some **stride.**

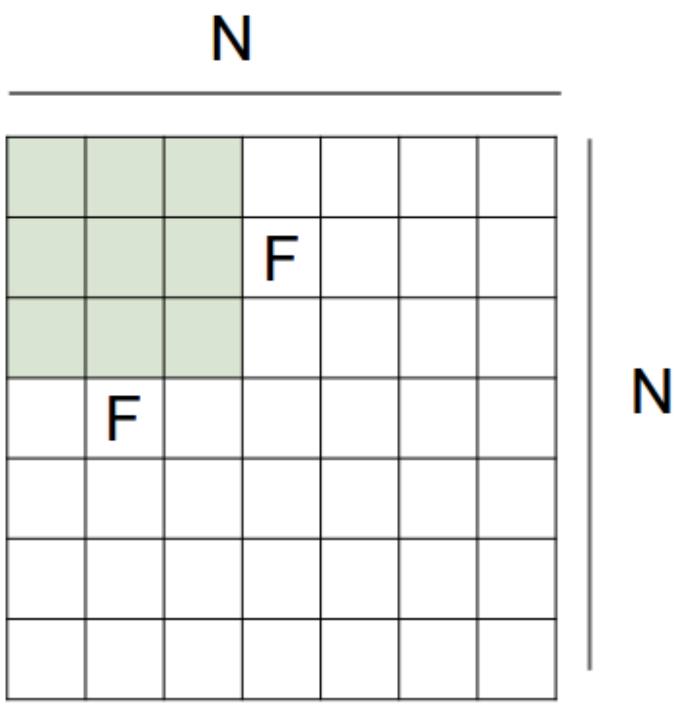


7x7 input
assume 3x3 connectivity, stride 1

--> 5x5 output

What about stride 3? **CANNOT**

ConvNets: architecture

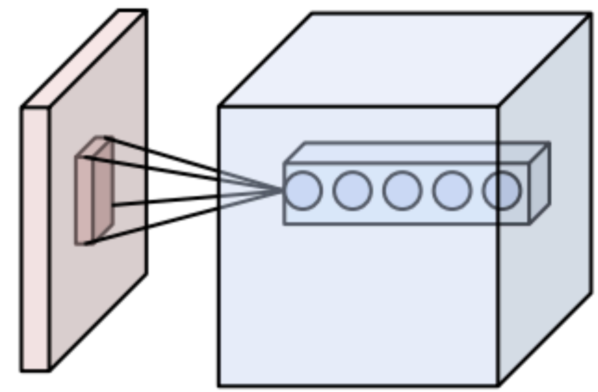


Output size:
(N - F) / stride + 1

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

ConvNets: architecture

Examples time:

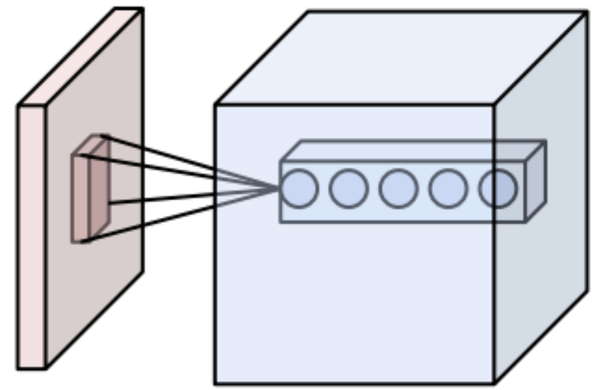


Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 1**
Number of neurons: **5**

Output volume: ?

ConvNets: architecture

Examples time:

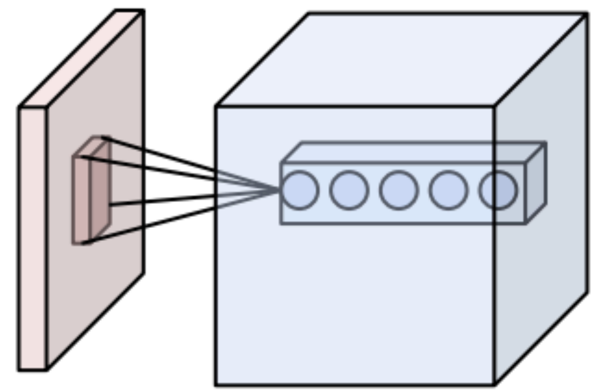


Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 1**
Number of neurons: **5**

Output volume: (32 - 5) / 1 + 1 = 28, so: **28x28x5**
How many weights for each of the 28x28x5 neurons?

ConvNets: architecture

Examples time:

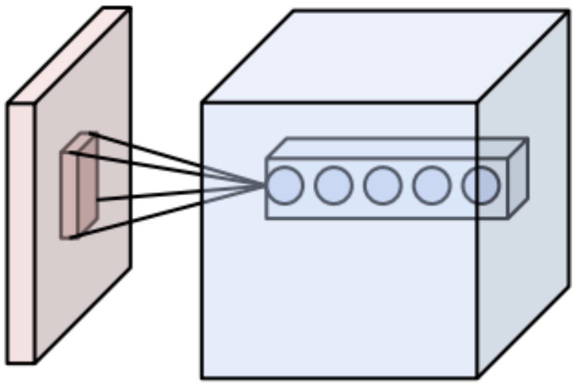


Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 1**
Number of neurons: **5**

Output volume: (32 - 5) / 1 + 1 = 28, so: **28x28x5**
How many weights for each of the 28x28x5 neurons? **5x5x3 = 75**

ConvNets: architecture

Examples time:

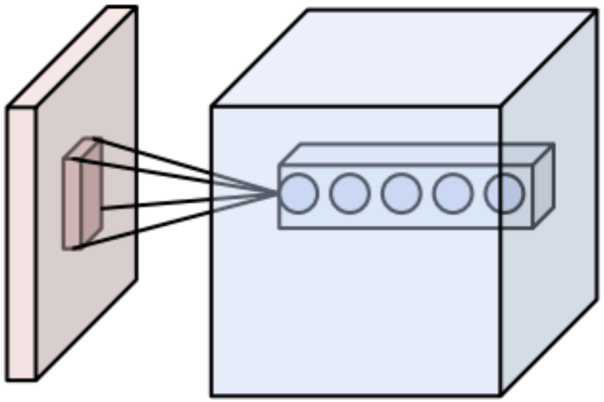


Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 2**
Number of neurons: **5**

Output volume: ?

ConvNets: architecture

Examples time:



Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 2**
Number of neurons: **5**

Output volume: ? Cannot: (32-5)/2 + 1 = 14.5 :\

ConvNets: architecture

Input volume of size [W1 x H1 x D1]
using K neurons with receptive fields F x F and applying them at strides of S gives
Output volume: [W2, H2, D2]

$\boxed{W2 = (W1-F)/S+1}$ $\boxed{H2 = (H1-F)/S+1}$ $\boxed{D2 = K}$

ConvNets: architecture

There's one more problem…

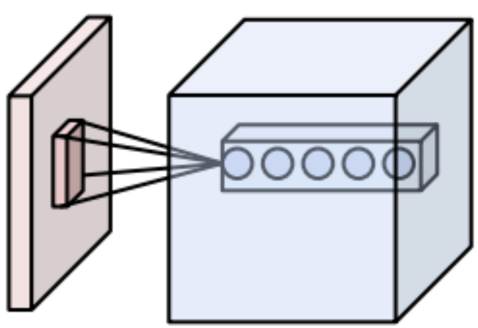
Assume input **[32 x 32 x3]**

30 neurons with receptive fields **5x5**, applied at **stride 1/pad1:**

=> Output volume: [32 x 32 x 30]   (32*32*30 = 30720 neurons)

Each neuron has 5*5*3 (=75) weights

=> Number of weights in such layer: 30720 * 75 ~= **3 million :**



ConvNets: architecture

There's one more problem…

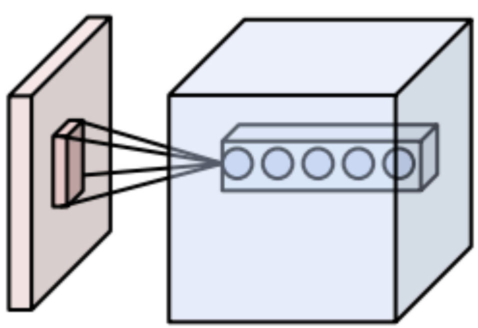
Assume input **[32 x 32 x3]**

30 neurons with receptive fields **5x5**, applied at **stride 1/pad1:**

=> Output volume: [32 x 32 x 30] (32*32*30 = 30720 neurons)

Each neuron has 5*5*3 (=75) weights

=> Number of weights in such layer: 30720 * 75 ~= **3 million :**



Example trained weights

ConvNets: architecture

There's one more problem…

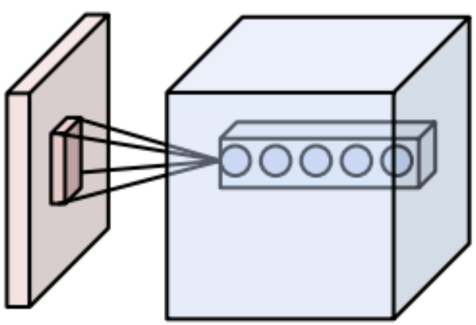
Assume input **[32 x 32 x3]**

30 neurons with receptive fields **5x5**, applied at **stride 1/pad1:**

=> Output volume: [32 x 32 x 30]   (32*32*30 = 30720 neurons)

Each neuron has 5*5*3 (=75) weights

=> Number of weights in such layer: 30720 * 75 ~= **3 million :**



Example trained weights

IDEA: lets not learn the same thing across all spatial locations

ConvNets: architecture

Our first ConvNet layer had size **[32 x 32 x3]**
If we had **30** neurons with receptive fields **5x5**, **stride 1, pad 1**
Output volume: [32 x 32 x 30] (32*32*30 = 30720 neurons)
Each neuron has 5*5*3 (=75) weights

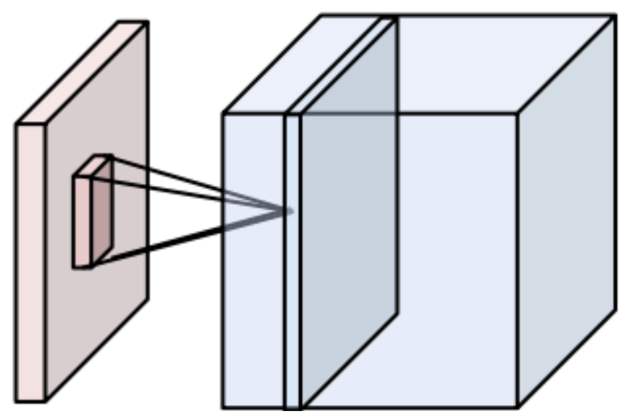
Before:
#weights in such layer: (32*32*30) * 75 = **3 million :**

Now: (paramater sharing)
#weights in the layer: 30 * 75 = **2250**.

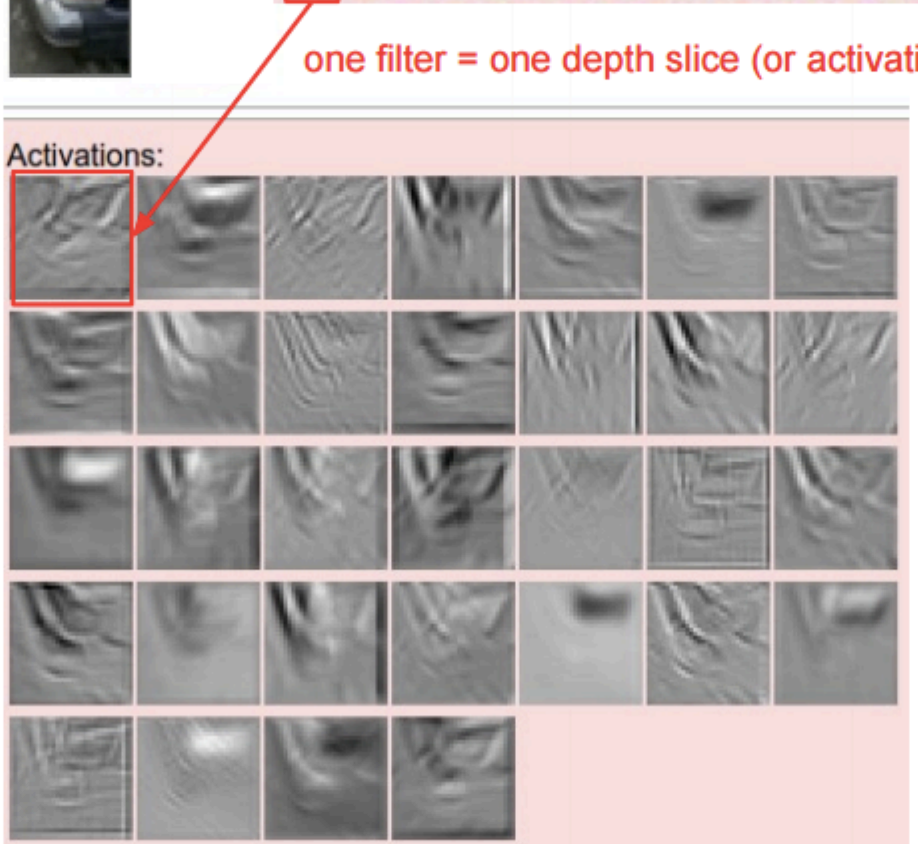
ConvNets: architecture

These layers are called **Convolutional Layers**

1. Connect neurons only to local receptive fields
2. Use the same neuron weight parameters for neurons in each "depth slice" (i.e. across spatial positions)



ConvNets: architecture



5x5 filters

Can call the neurons “filters”

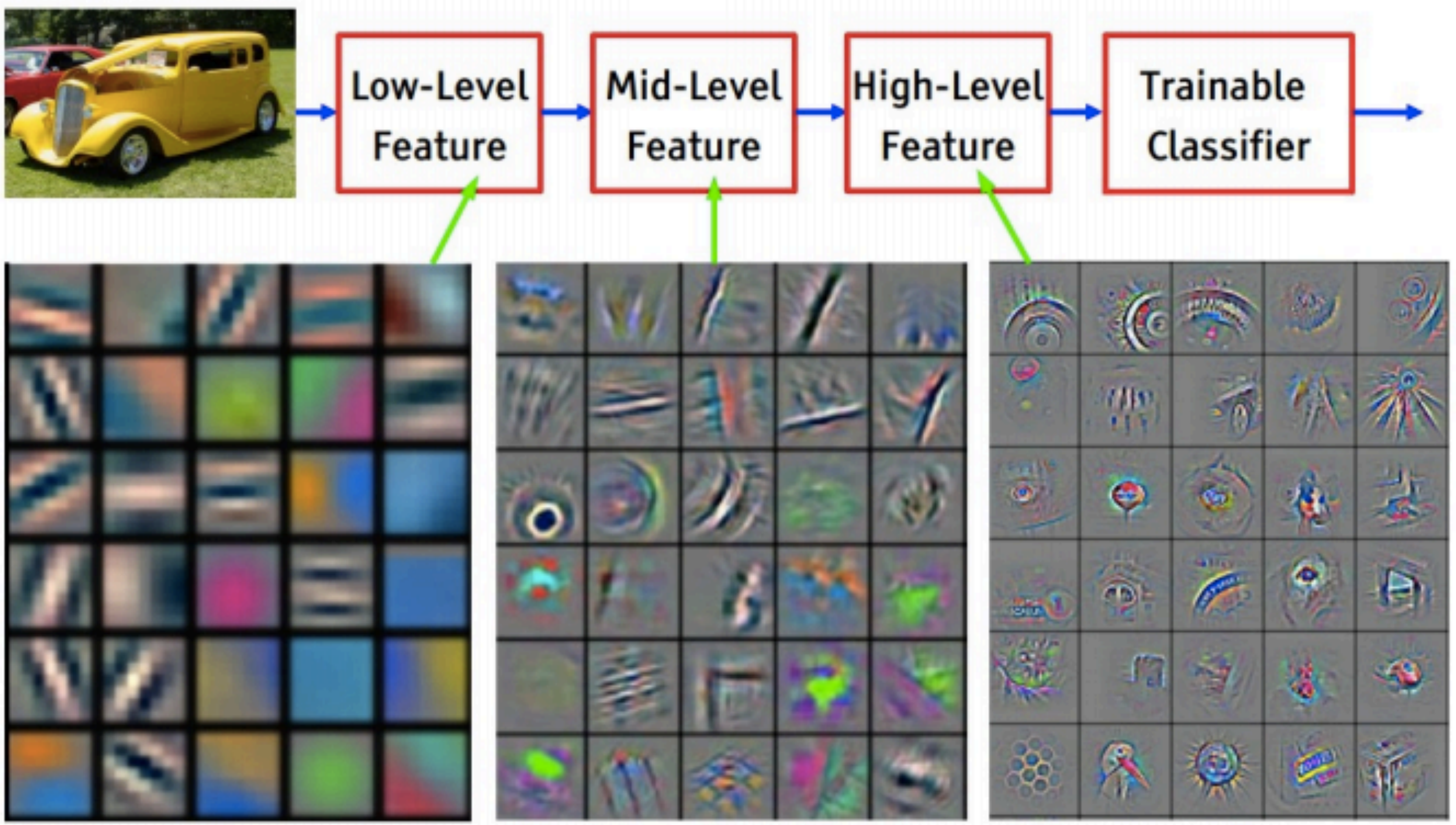
We call the layer convolutional because it is related to convolution of two signals (kind of):

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1,y-n_2]$$

elementwise multiplication and sum a filter and the signal (image)

Fast-forward to today

[From recent Yann LeCun slides]

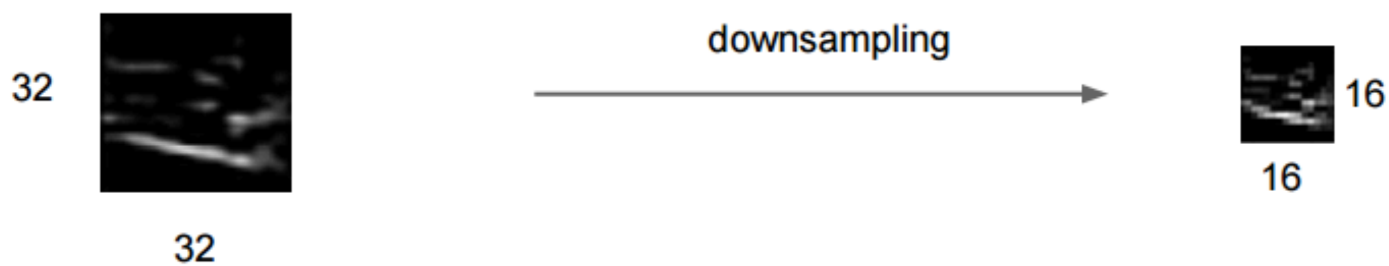


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

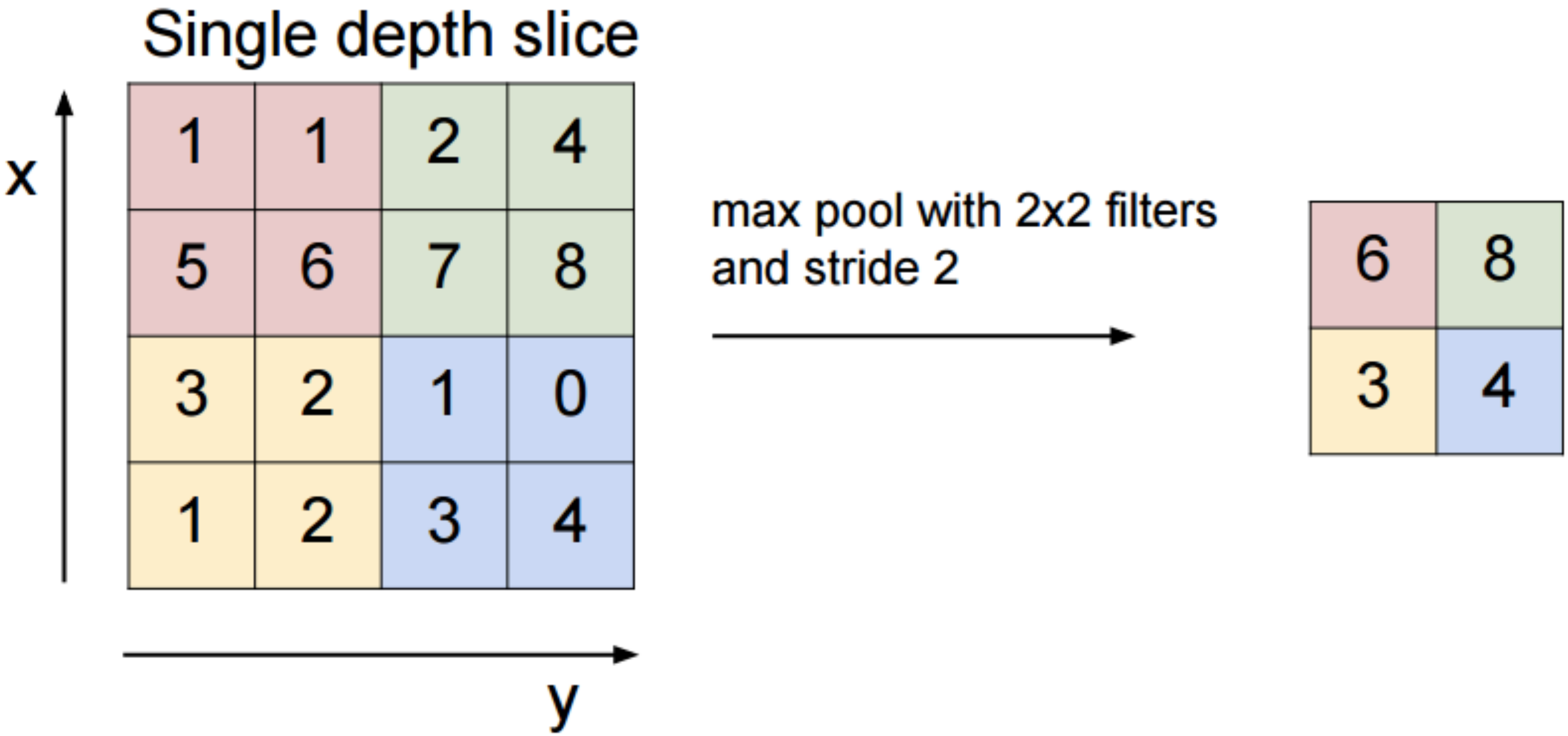
ConvNets: Max pooling

In ConvNet architectures, **Conv** layers are often followed by **Pool** layers

- convenience layer: makes the representations smaller and more manageable without losing too much information. Computes MAX operation (most common)



ConvNets: Max pooling



ConvNets: Max pooling

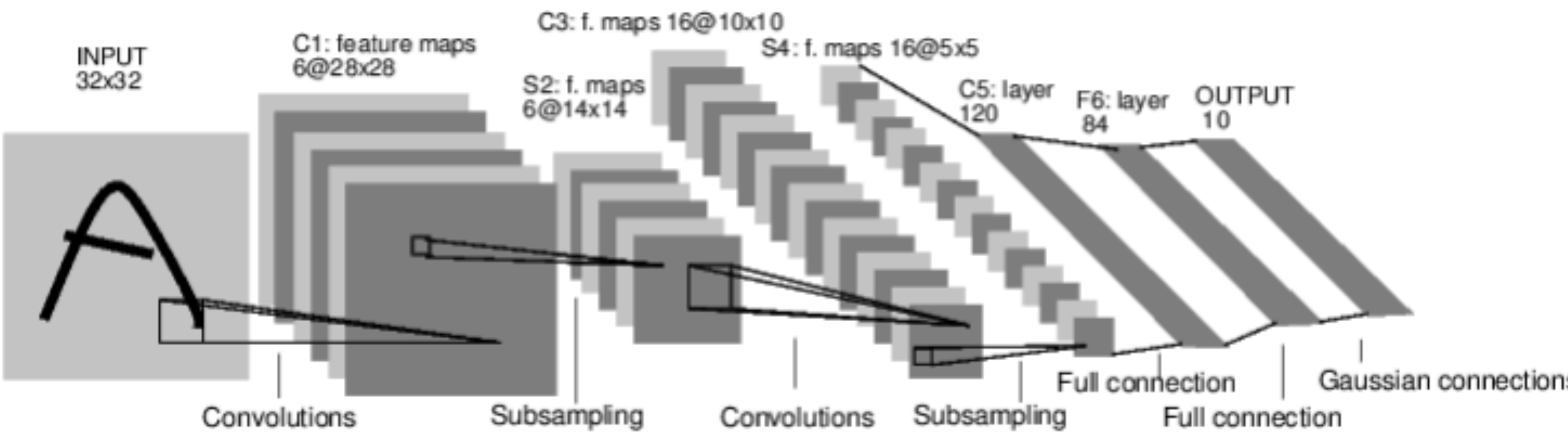
In ConvNet architectures, **Conv** layers are often followed by **Pool** layers

- convenience layer: makes the representations smaller and more manageable without losing too much information. Computes MAX operation (most common)

Input volume of size [W1 x H1 x D1]
Pooling unit receptive fields F x F and applying them at strides of S gives
Output volume: [W2, H2, D1]
W2 = (W1-F)/S+1, H2 = (H1-F)/S+1

Note: pooling happens independently across each slice, preserving number of slices
E.g. a pooling "neuron" of size 2x2 will perform MAX operation over 4 numbers.

ConvNets: summary

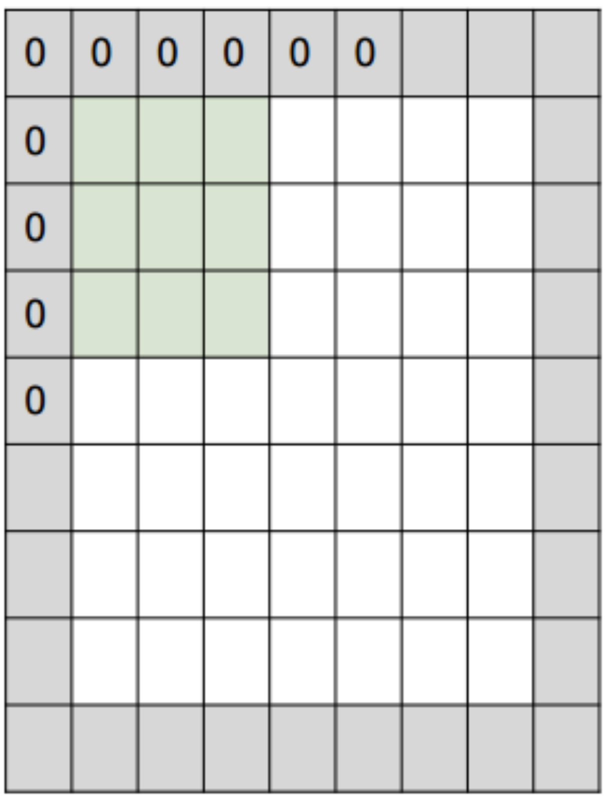


[LeNet-5, LeCun 1980]

ConvNets: tips & tricks

In practice: Common to zero pad the border

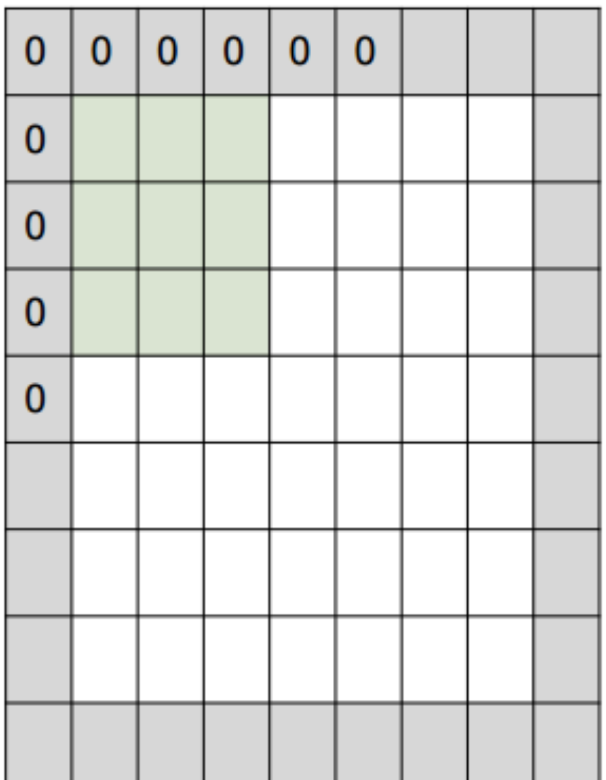
(in each channel)



e.g. input 7x7
neuron with receptive field 3x3, stride 1
pad with 1 pixel border => what is the output?

ConvNets: tips & tricks

In practice: Common to zero pad the border



(in each channel)

e.g. input 7x7
neuron with receptive field 3x3, stride 1
pad with 1 pixel border => what is the output?

7x7 => preserved size!

in general, common to see stride 1, size F, and zero-padding with (F-1)/2.
(Will preserve input size spatially)

ConvNets: tips & tricks

- start with image that has power-of-2 size
- for **conv layers**, use stride 1 filter size 3x3 pad input with a border of zeros (1 spatially)

This makes it so that: [W1,H1,**D1**] -> [W1,H1,**D2**] (i.e. spatial size exactly preserved)

- for **pool layers**, use pool size 2x2 (more = worse)

Part 3:

Unsupervised deep learning

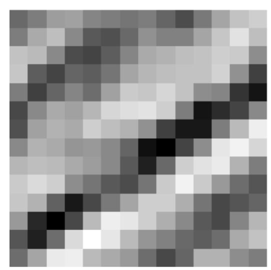
Methods of Unsupervised Deep Learning

- Autoencoders
- Deep belief networks (DBNs)
  - Restricted Boltzmann Machines (RBMs)
  - Deep Boltzmann Machines (DBMs)
- Sparse coding

…

Feature learning problem

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



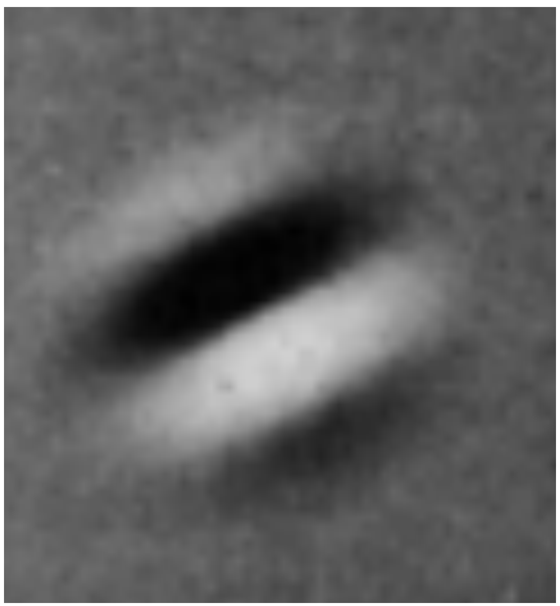
$$\begin{pmatrix} 255 \\ 98 \\ 93 \\ 87 \\ 89 \\ 91 \\ 48 \\ \ldots \end{pmatrix}$$

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

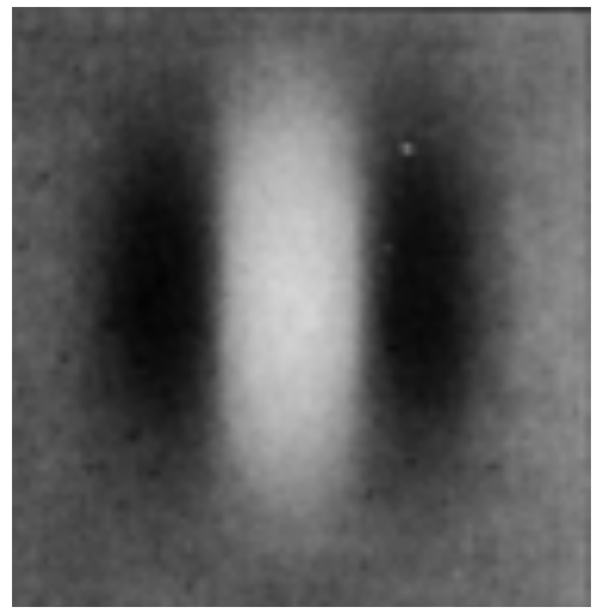
First stage of visual processing: V1

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 $R^{n \times n}$)

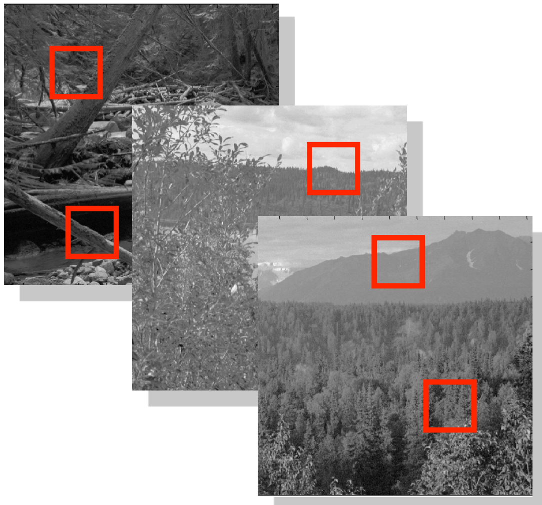
Learn: Dictionary of bases $\phi_1, \phi_2, \ldots, \phi_k$ (also $R^{n \times n}$), so that each input x can be approximately decomposed as:

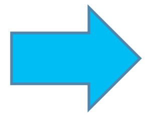
$$x \approx \sum_{j=1}^{k} a_j \phi_j$$

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

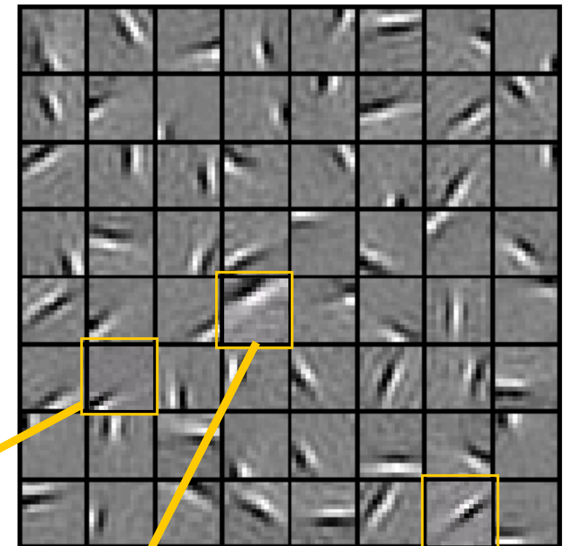
Sparse coding illustration

Natural Images

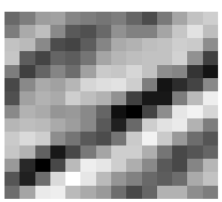
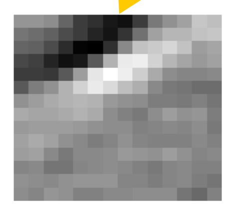
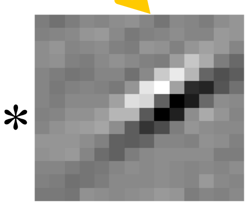




Learned bases ($\phi_1, \ldots, \phi_{64}$): "Edges"



Test example

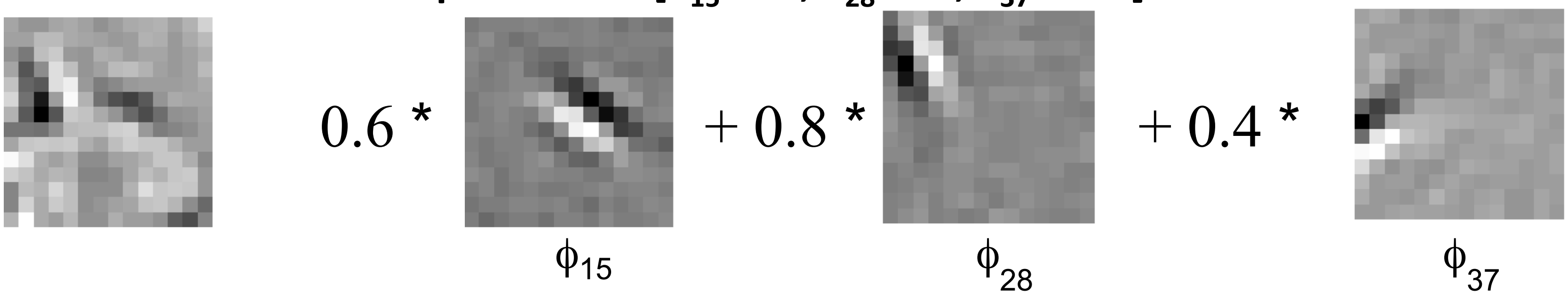
 $\approx 0.8 *$  $+ 0.3 *$  $+ 0.5 *$ 

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

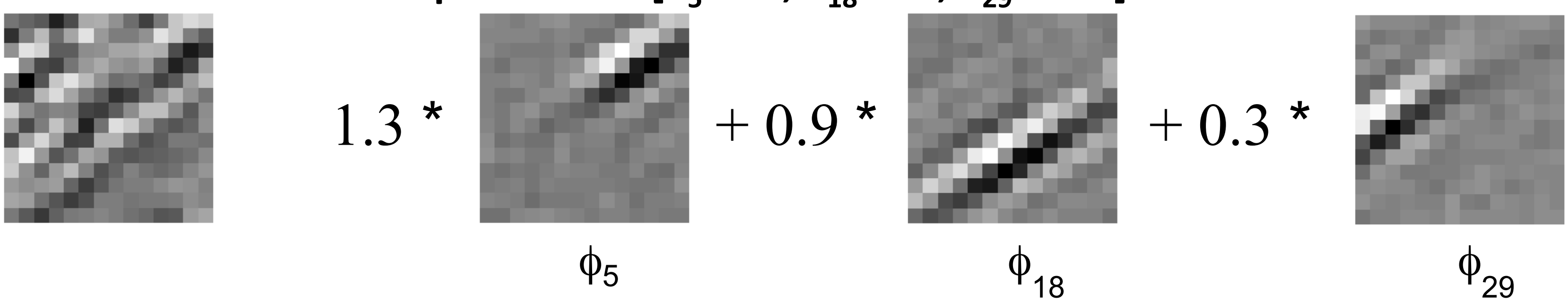
$[a_1, \ldots, a_{64}] = [0, 0, \ldots, 0, \mathbf{0.8}, 0, \ldots, 0, \mathbf{0.3}, 0, \ldots, 0, \mathbf{0.5}, 0]$
(feature representation)

Sparse coding illustration

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



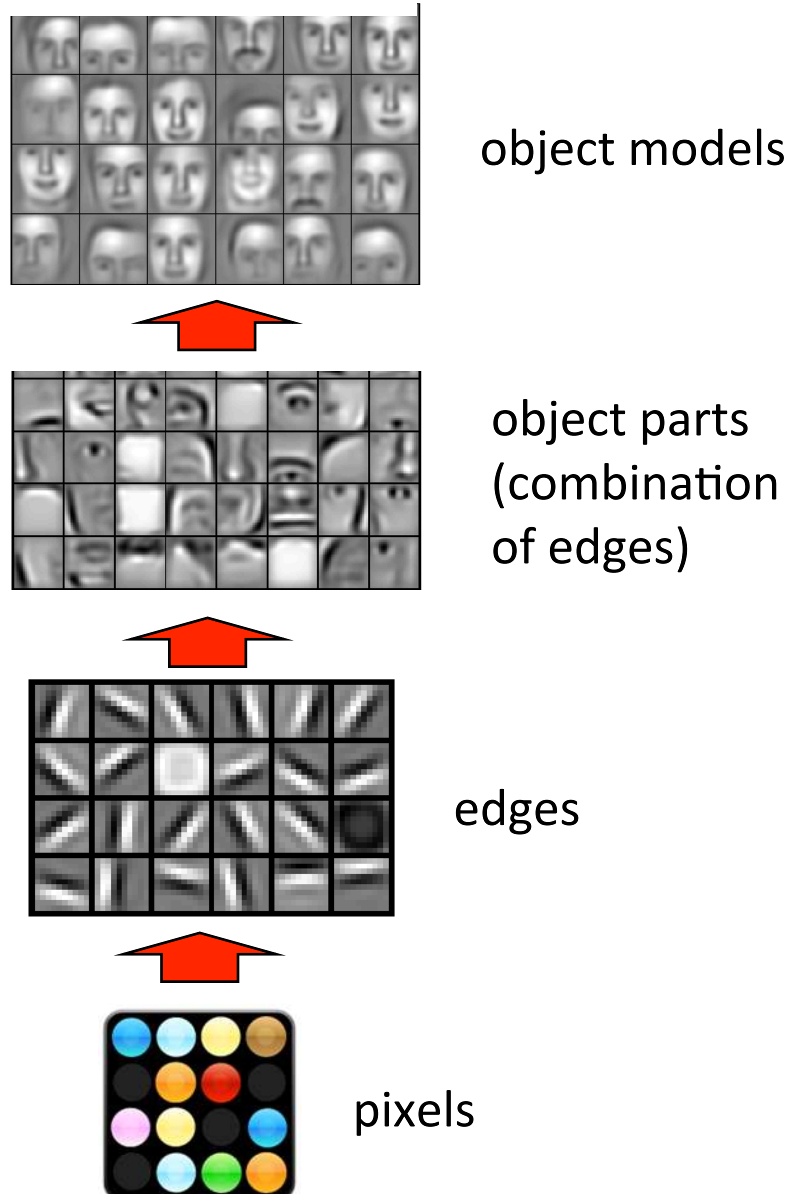
Represent as: [a_{5}=1.3, a_{18}=0.9, a_{29} = 0.3]



- **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.

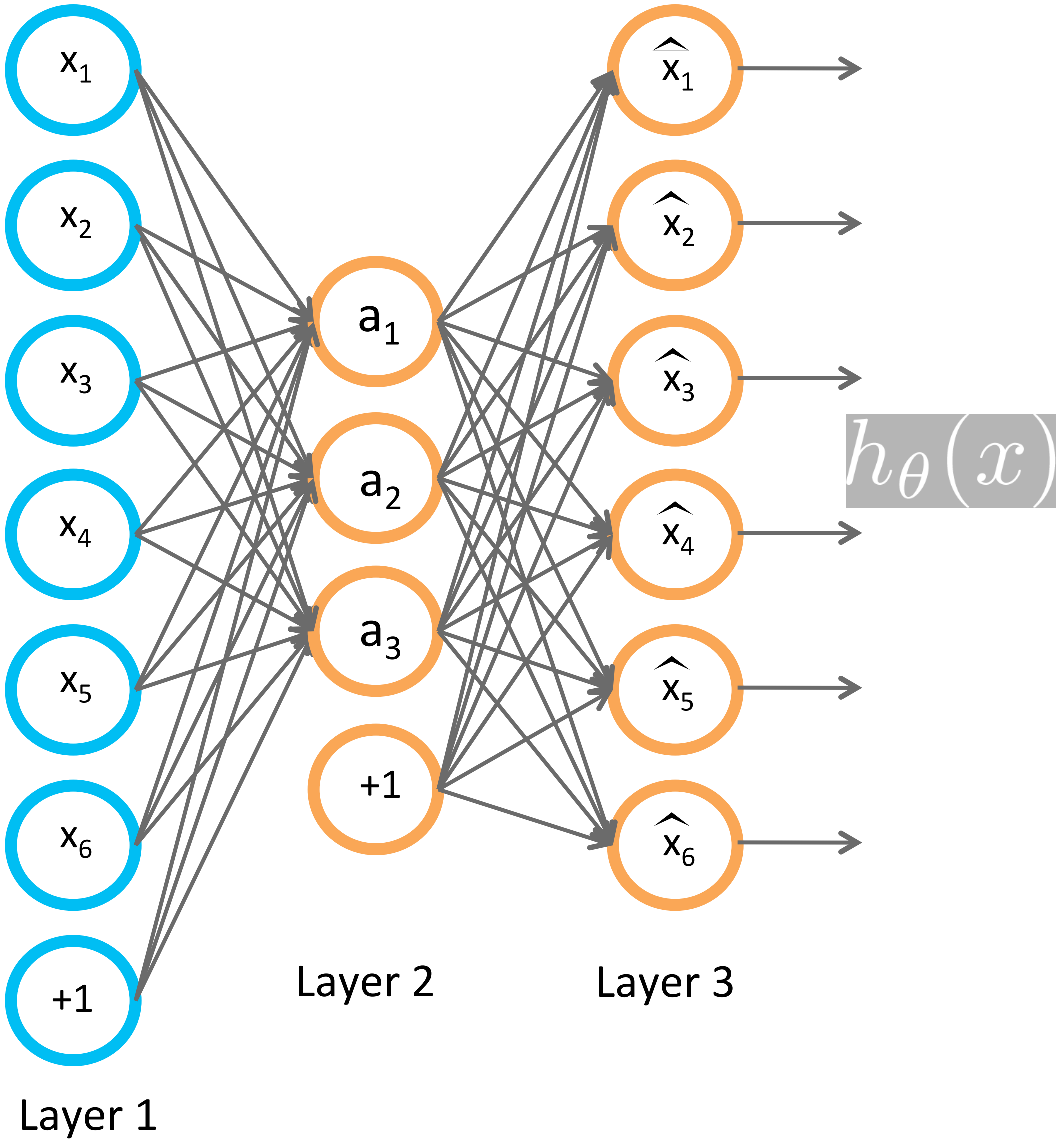
Going deep

Training set: Aligned images of faces.



[Honglak Lee]

Autoencoder



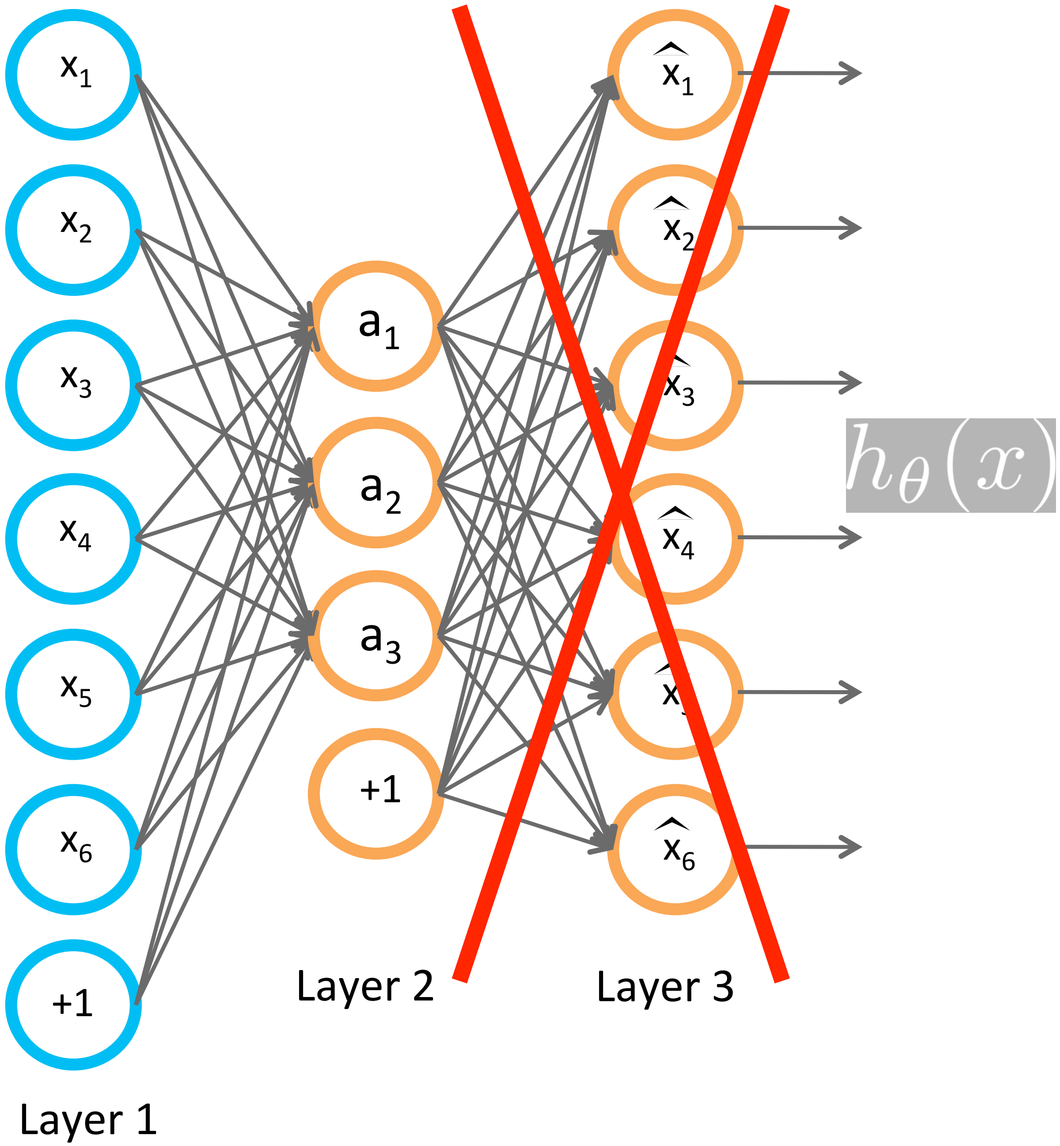
Network is trained to output the input (learn identify function).

$$h_\theta(x) \approx x$$

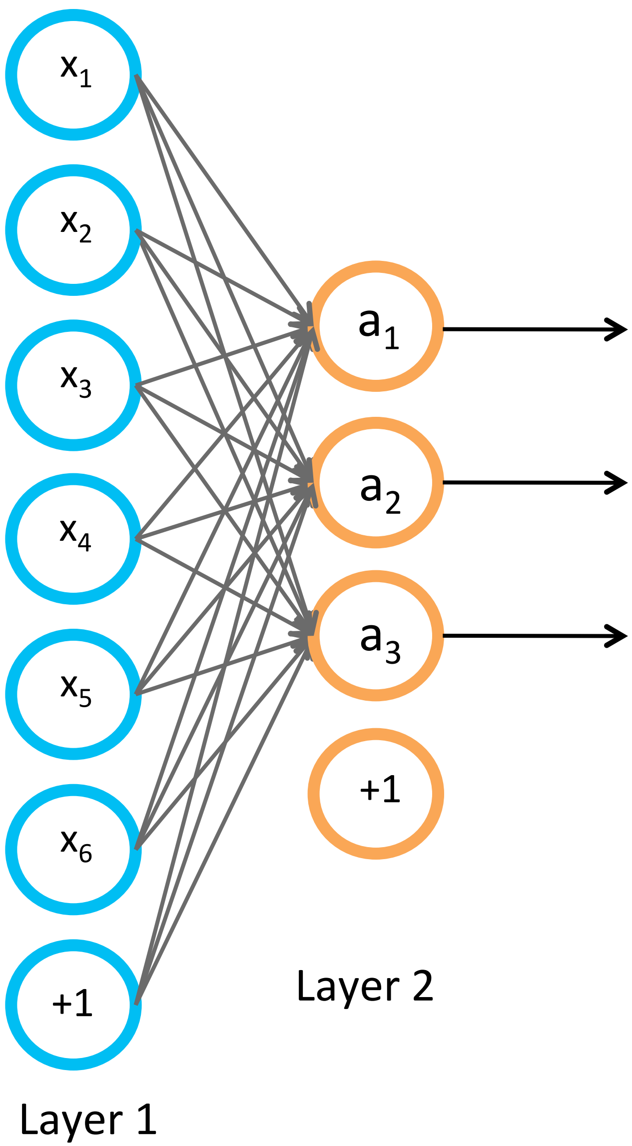
Trivial solution unless:
- Constrain number of units in Layer 2 (learn compressed representation), or

- Constrain Layer 2 to be **sparse**.

Autoencoder



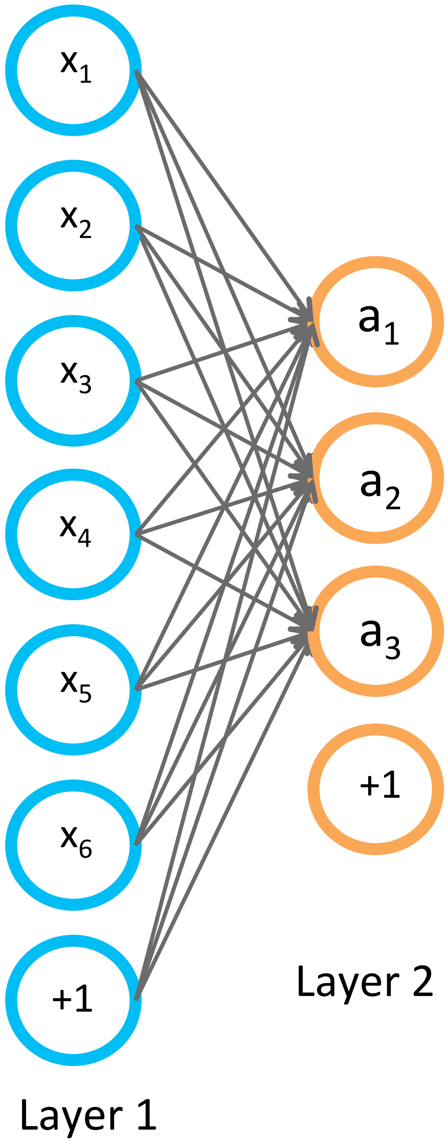
Autoencoder



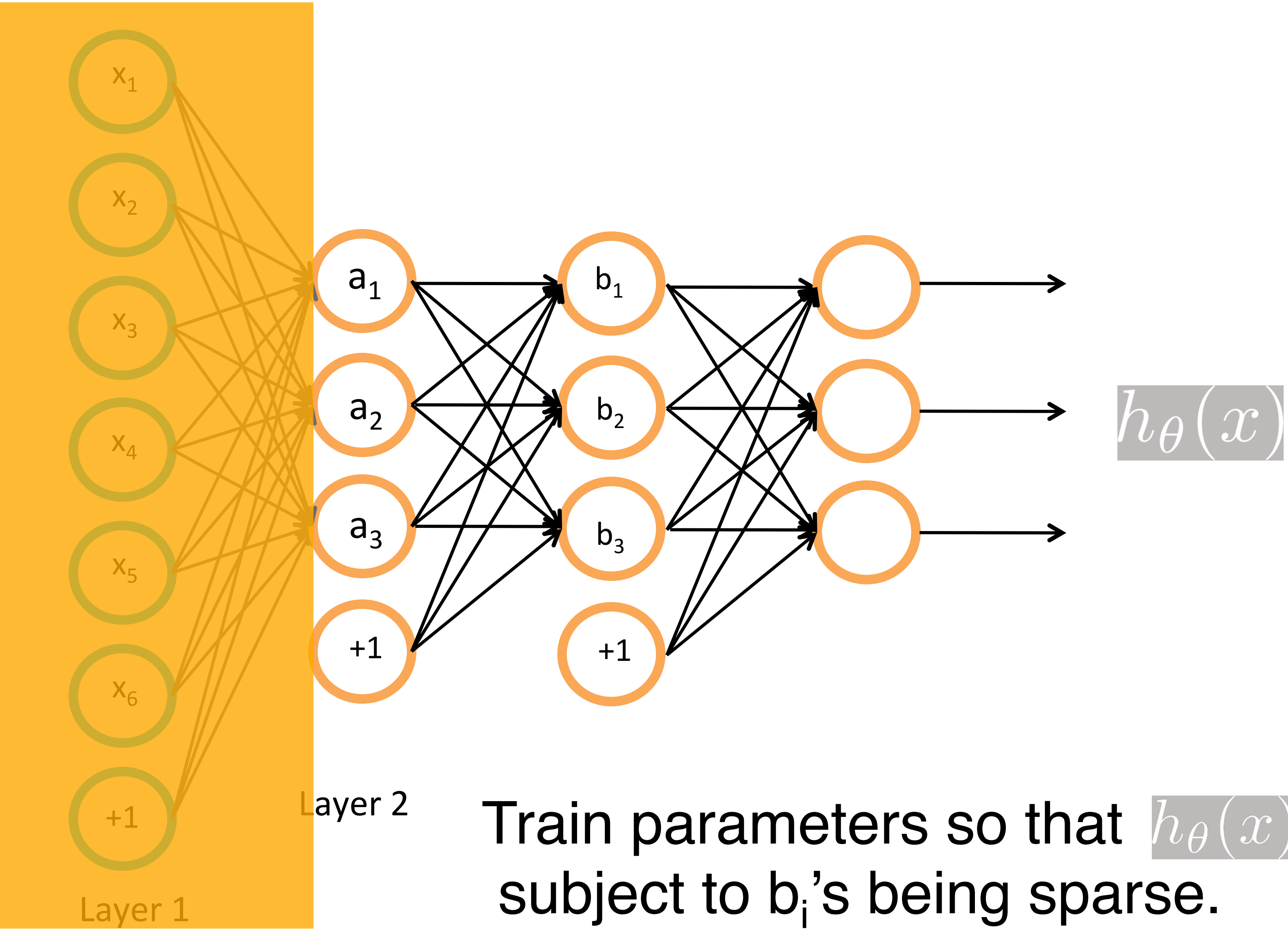
$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

New representation for input.

Autoencoder

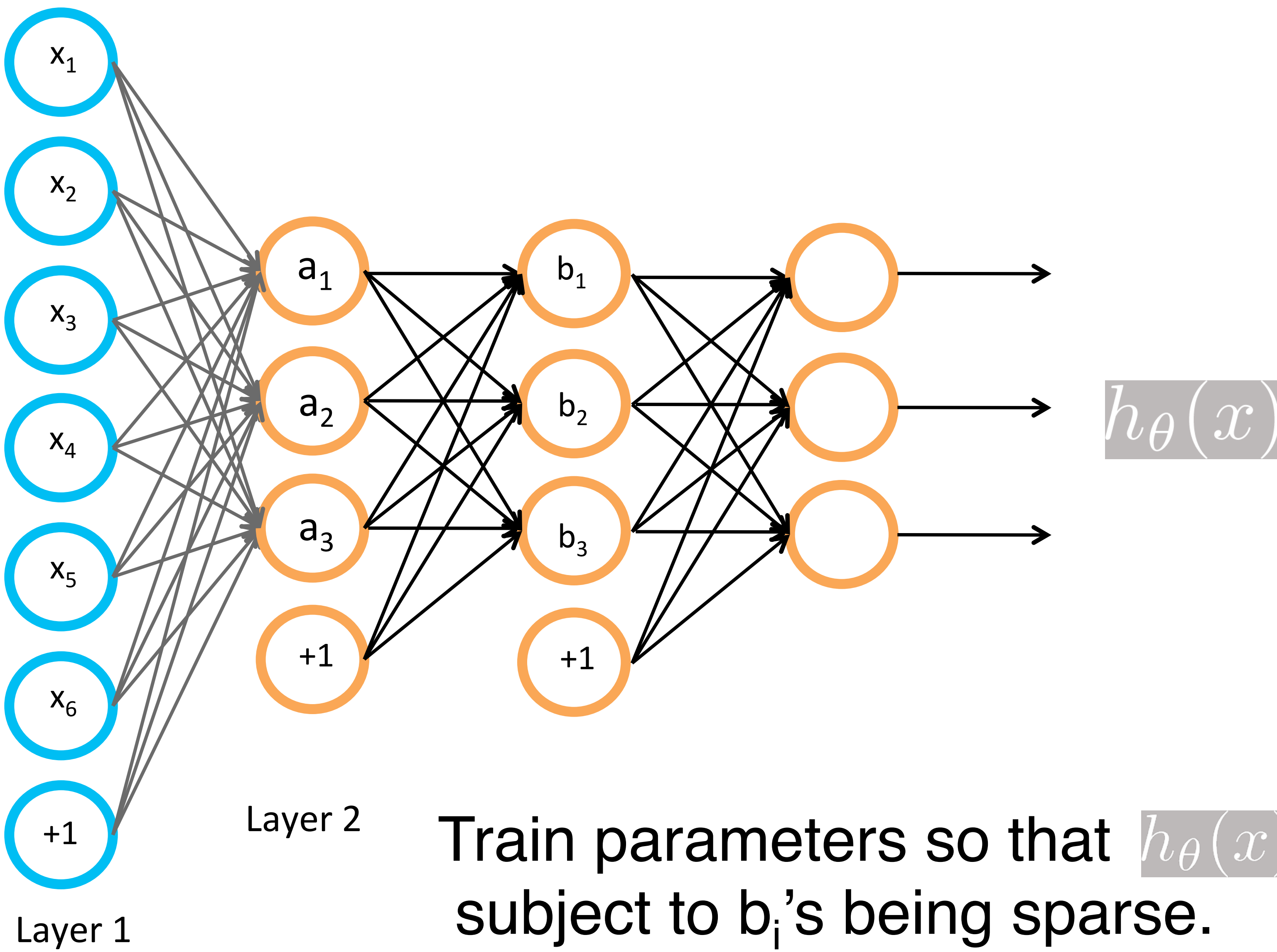


Autoencoder



Train parameters so that $h_\theta(x) \approx a$, subject to b_i's being sparse.

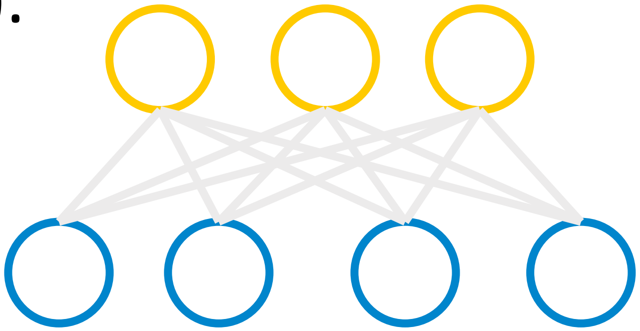
Autoencoder



Train parameters so that $h_\theta(x) \approx a$, subject to b_i's being sparse.

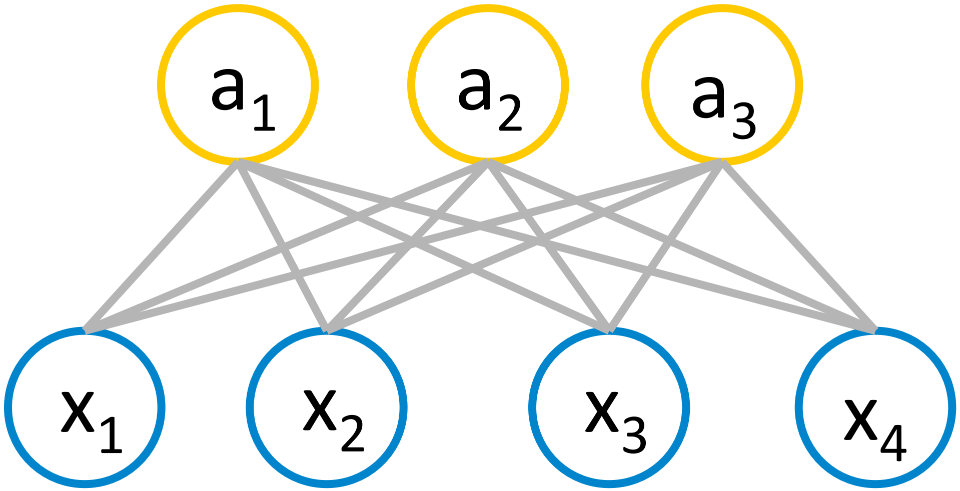
Deep Belief Net

- Deep Belief Net (DBN) is another algorithm for learning a feature hierarchy.
- Building block: 2-layer graphical model (Restricted Boltzmann Machine).



- Can then learn additional layers one at a time.

Restricted Boltzmann Machine (RBM)



Layer 2. $[a_1, a_2, a_3]$
(binary-valued)

Input $[x_1, x_2, x_3, x_4]$

MRF with joint distribution:

$$P(x, a) \propto \exp\left(-\sum_{i,j} x_i a_j W_{ij}\right)$$

Use Gibbs sampling for inference.

Given observed inputs x, want maximum likelihood estimation:

$$\max_W P(x) = \max_W \sum_a P(x, a)$$

Deep Belief Network

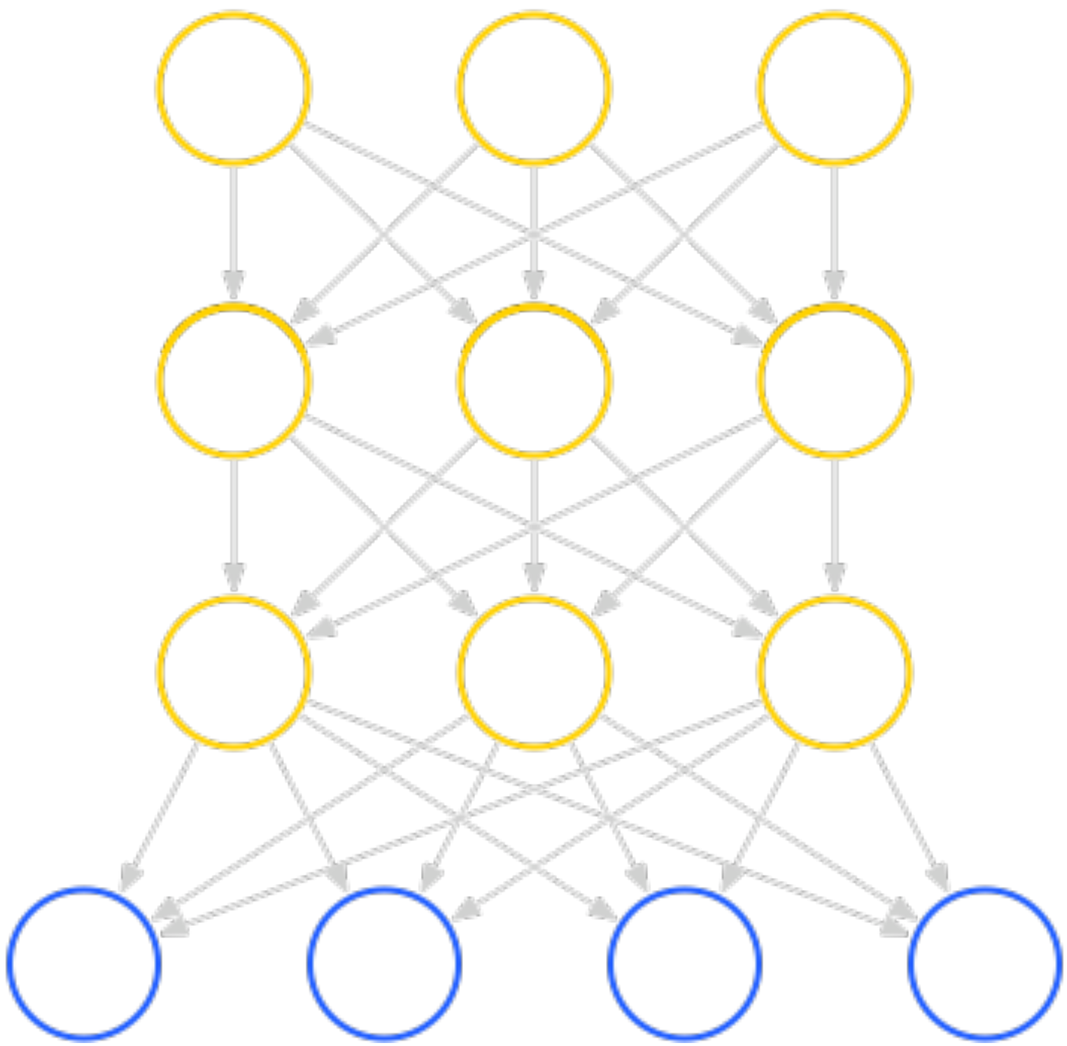
Similar to a sparse autoencoder in many ways. Stack RBMs on top of each other to get DBN.

Layer 3. $[b_1, b_2, b_3]$

Layer 2. $[a_1, a_2, a_3]$

Input $[x_1, x_2, x_3, x_4]$

Deep Belief Network



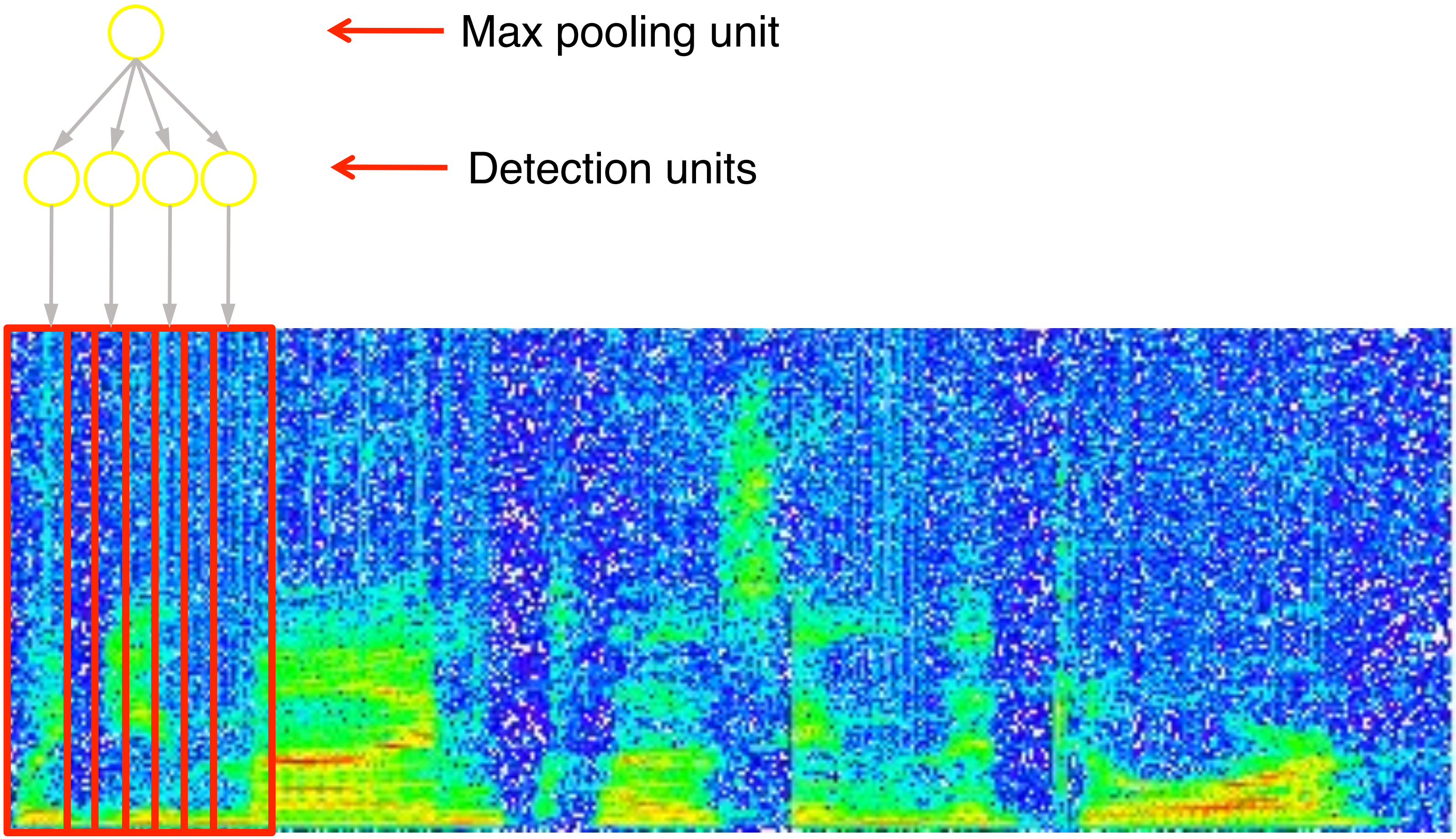
Layer 4. $[c_1, c_2, c_3]$

Layer 3. $[b_1, b_2, b_3]$

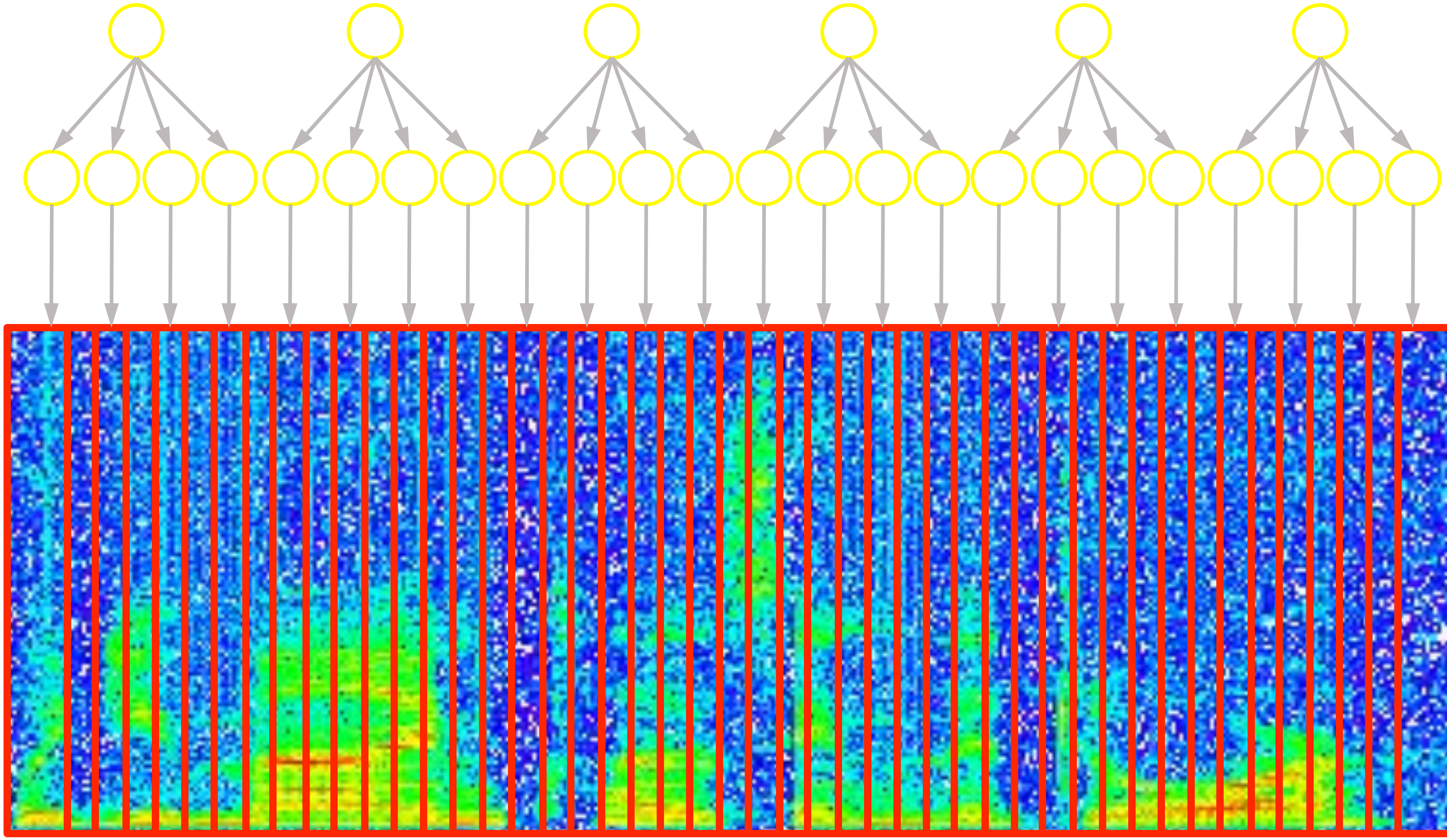
Layer 2. $[a_1, a_2, a_3]$

Input $[x_1, x_2, x_3, x_4]$

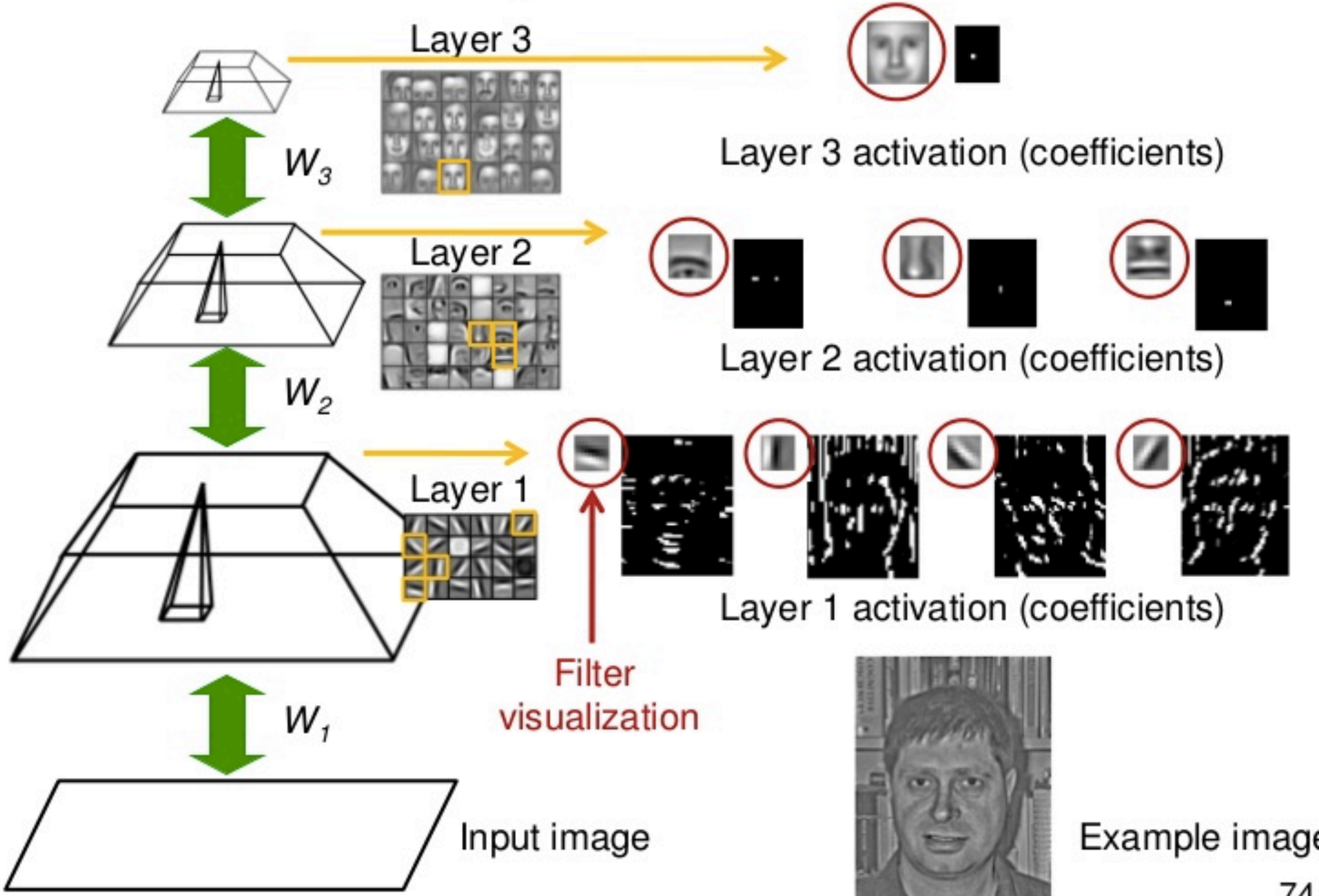
Convolutional DBN for audio



Convolutional DBN for audio



Convolutional deep belief networks illustration



Benefits of unsupervised feature learning

Task: video activity recognition

Method	Accuracy
Hessian + ESURF [Williems et al 2008]	38%
Harris3D + HOG/HOF [Laptev et al 2003, 2004]	45%
Cuboids + HOG/HOF [Dollar et al 2005, Laptev 2004]	46%
Hessian + HOG/HOF [Laptev 2004, Williems et al 2008]	46%
Dense + HOG / HOF [Laptev 2004]	47%
Cuboids + HOG3D [Klaser 2008, Dollar et al 2005]	46%
Unsupervised feature learning (our method)	**52%**

[Le, Zhou & Ng, 2011]

Audio

TIMIT Phone classification	Accuracy
Prior art (Clarkson et al.,1999)	79.6%
Feature learning	**80.3%**

TIMIT Speaker identification	Accuracy
Prior art (Reynolds, 1995)	99.7%
Feature learning	**100.0%**

Images

CIFAR Object classification	Accuracy
Prior art (Ciresan et al., 2011)	80.5%
Feature learning	**82.0%**

NORB Object classification	Accuracy
Prior art (Scherer et al., 2010)	94.4%
Feature learning	**95.0%**

Video

Hollywood2 Classification	Accuracy
Prior art (Laptev et al., 2004)	48%
Feature learning	**53%**

YouTube	Accuracy
Prior art (Liu et al., 2009)	71.2%
Feature learning	**75.8%**

KTH	Accuracy
Prior art (Wang et al., 2010)	92.1%
Feature learning	**93.9%**

UCF	Accuracy
Prior art (Wang et al., 2010)	85.6%
Feature learning	**86.5%**

Text/NLP

Paraphrase detection	Accuracy
Prior art (Das & Smith, 2009)	76.1%
Feature learning	**76.4%**

Sentiment (MR/MPQA data)	Accuracy
Prior art (Nakagawa et al., 2010)	77.3%
Feature learning	**77.7%**

ImageNet classification: 22,000 classes

…
smoothhound, smoothhound shark, Mustelus mustelus
American smooth dogfish, Mustelus canis
Florida smoothhound, Mustelus norrisi
whitetip shark, reef whitetip shark, Triaenodon obseus
Atlantic spiny dogfish, Squalus acanthias
Pacific spiny dogfish, Squalus suckleyi
hammerhead, hammerhead shark
smooth hammerhead, Sphyrna zygaena
smalleye hammerhead, Sphyrna tudes
shovelhead, bonnethead, bonnet shark, Sphyrna tiburo
angel shark, angelfish, Squatina squatina, monkfish
electric ray, crampfish, numbfish, torpedo
smalltooth sawfish, Pristis pectinatus
guitarfish
roughtail stingray, Dasyatis centroura
butterfly ray
eagle ray
spotted eagle ray, spotted ray, Aetobatus narinari
cownose ray, cow-nosed ray, Rhinoptera bonasus
manta, manta ray, devilfish
Atlantic manta, Manta birostris
devil ray, Mobula hypostoma
grey skate, gray skate, Raja batis
little skate, Raja erinacea
…

Stingray

Mantaray

ImageNet Classification: 14M images, 22k categories

0.005%	9.5%	?
Random guess	State-of-the-art (Weston, Bengio '11)	Feature learning From raw pixels

Le, et al., *Building high-level features using large-scale unsupervised learning*. ICML 2012

ImageNet Classification: 14M images, 22k categories

0.005%	9.5%	21.3%
Random guess	State-of-the-art (Weston, Bengio ‘11)	Feature learning From raw pixels

Le, et al., *Building high-level features using large-scale unsupervised learning*. ICML 2012