Neural Networks

Marcello Pelillo

University of Venice, Italy

Artificial Intelligence

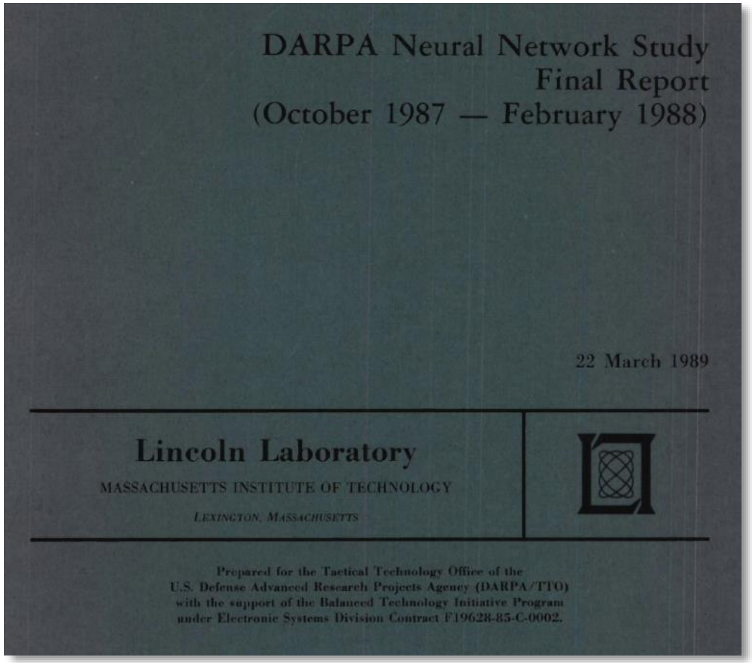
a.y. 2017/18

Ca' Foscari University of Venice

European Centre for Living Technology

DARPA Neural Network Study (1989)

“Over the history of computing science, two advances have matured: High speed numerical processing and knowledge processing (Artificial Intelligence). Neural networks seem to offer the next necessary ingredient for intelligent machines – namely, knowledge formation and organization.”



DARPA Neural Network Study
Final Report
(October 1987 — February 1988)

22 March 1989

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON, MASSACHUSETTS

Prepared for the Tactical Technology Office of the
U.S. Defense Advanced Research Projects Agency (DARPA/TTO)
with the support of the Balanced Technology Initiative Program
under Electronic Systems Division Contract F19628-85-C-0002.

DARPA Neural Network Study (1989)

> Two key features which, it is widely believed, distinguish neural networks from any other sort of computing developed thus far:
>
> **Neural networks are adaptive, or trainable.** Neural networks are not so much programmed as they are trained with data − thus many believe that the use of neural networks can relieve today's computer programmers of a significant portion of their present programming load. Moreover, neural networks are said to improve with experience − the more data they are fed, the more accurate or complete their response.
>
> **Neural networks are naturally massively parallel.** This suggests they should be able to make decisions at high-speed and be fault tolerant.

History

Early work (1940-1960)

McCulloch & Pitts	(Boolean logic)
Rosenblatt	(Learning)
Hebb	(Learning)

Transition (1960-1980)

Widrow – Hoff	(LMS rule)
Anderson	(Associative memories)
Amari	

Resurgence (1980-1990's)

Hopfield	(Ass. mem. / Optimization)
Rumelhart et al.	(Back-prop)
Kohonen	(Self-organizing maps)
Hinton , Sejnowski	(Boltzmann machine)

New resurgence (2012 -)

- CNNs, Deep learning, GAN's ….

A Few Figures

The human cerebral cortex is composed of about

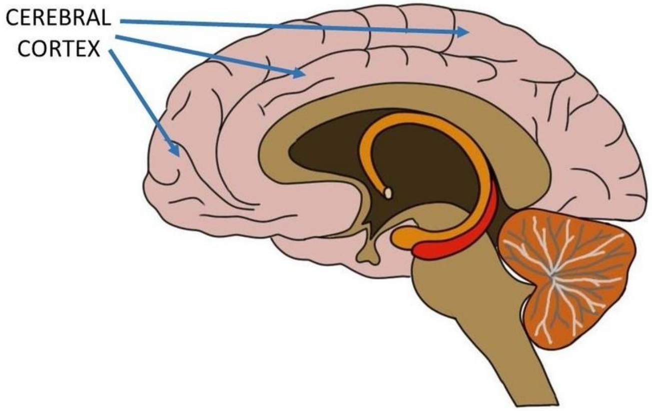
100 billion (10^{11}) neurons

of many different types.

Each neuron is connected to other 1000 / 10000 neurons, wich yields

$10^{14}/10^{15}$ connections

The cortex covers about 0.15 m^2
and is 2-5 mm thick

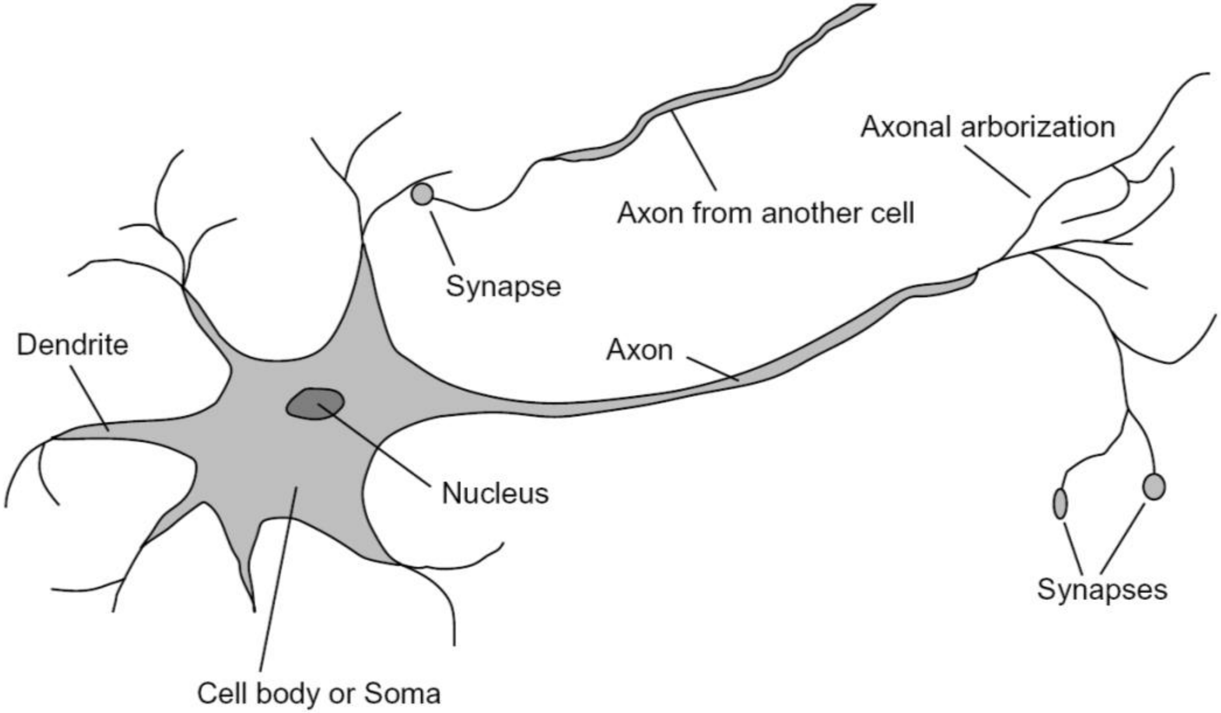


The Neuron

Cell Body (Soma): 5-10 microns in diameter

Axon: Output mechanism for a neuron; one axon/cell, but thousands of branches and cells possible for a single axon

Dendrites: Receive incoming signals from other nerve axons via synapse



Neural Dynamics

The transmission of signal in the cerebral cortex is a complex process:

electrical → chemical → electrical

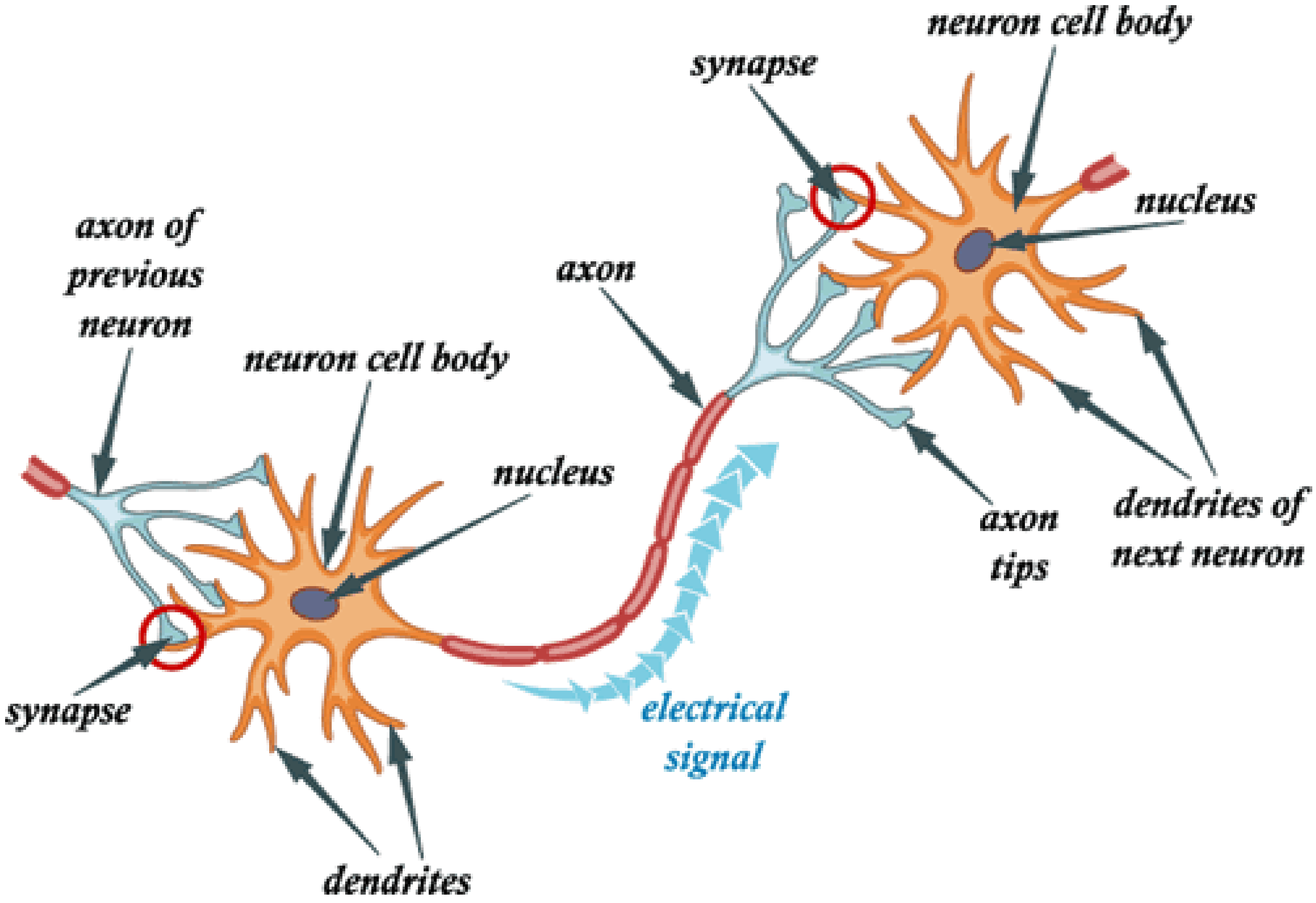
Simplifying :

1) The cellular body performs a “weighted sum” of the incoming signals

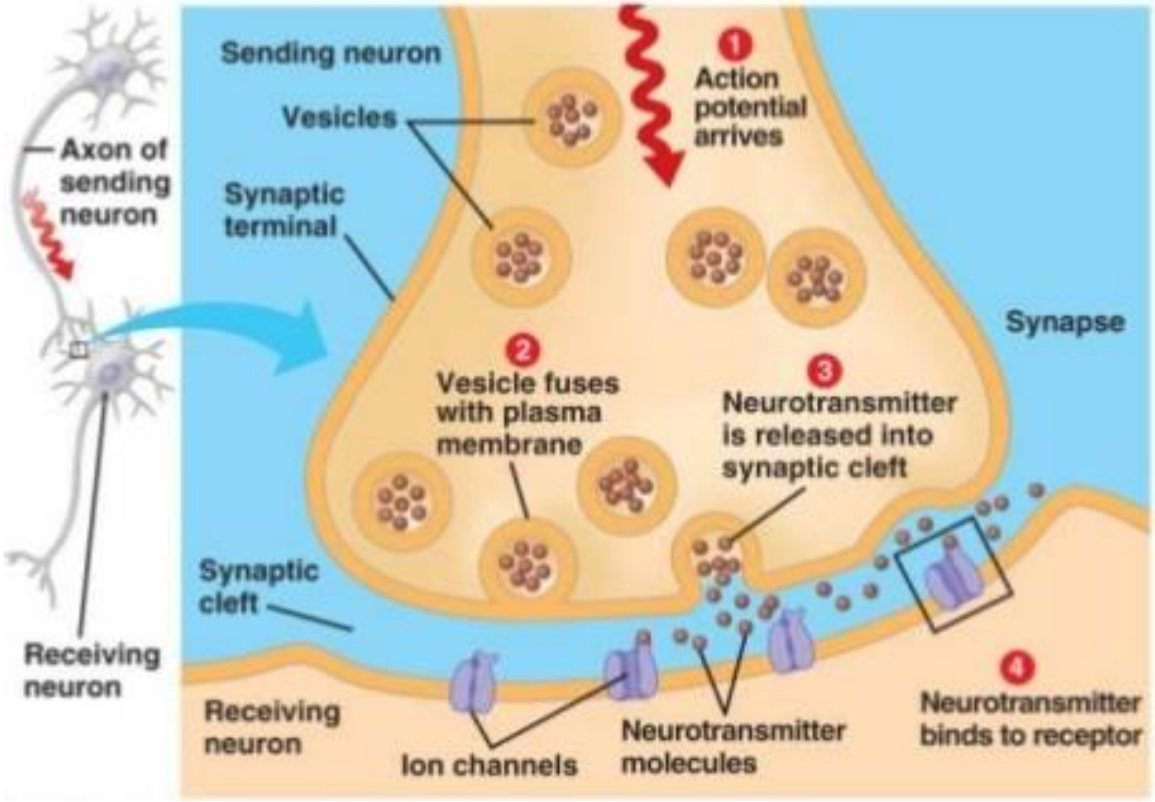
2) If the result exceeds a certain threshold value, then it produces an “action potential” which is sent down the axon (cell has “fired”), otherwise it remains in a rest state

3) When the electrical signal reaches the synapse, it allows the “neuro-transmitter” to be released and these combine with the “receptors” in the post-synaptic membrane

4) The post-synaptic receptors provoke the diffusion of an electrical signal in the post-synaptic neuron


neuron cell body
synapse
nucleus
axon of previous neuron
axon
neuron cell body
nucleus
dendrites of next neuron
axon tips
synapse
electrical signal
dendrites

Synapses


Sending neuron
Vesicles
Axon of sending neuron
Synaptic terminal
1 Action potential arrives
Synapse
2 Vesicle fuses with plasma membrane
3 Neurotransmitter is released into synaptic cleft
Synaptic cleft
Receiving neuron
Receiving neuron
Ion channels
Neurotransmitter molecules
4 Neurotransmitter binds to receptor

SYNAPSE is the relay point where information is conveyed by chemical transmitters from neuron to neuron. A synapse consists of two parts: the knowblike tip of an axon terminal and the receptor region on the surface of another neuron. The membranes are separated by a synaptic cleft some 200 nanometers across. Molecules of chemical transmitter, stored in vesicles in the axon terminal, are released into the cleft by arriving nerve impulses. Transmitter changes electrical state of the receiving neuron, making it either more likely or less likely to fire an impulse.

Synaptic Efficacy

It’s the amount of current that enters into the post-synaptic neuron, compared to the action potential of the pre-synaptic neuron.

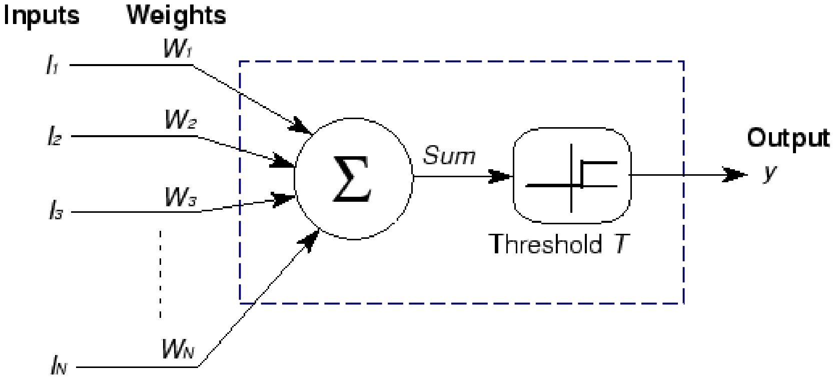
Learning takes place by modifying the synaptic efficacy.

Two types of synapses:

- **Excitatory** : favor the generation of action potential in the post-synaptic neuron
- **Inhibitory** : hinder the generation of action potential

The McCulloch and Pitts Model (1943)

The McCulloch-Pitts (MP) Neuron is modeled as a binary threshold unit

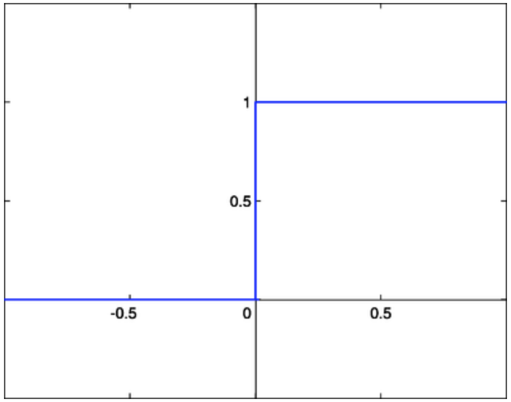


The unit “fires” if the **net input** $\sum_j w_j I_j$ reaches (or exceeds) the unit’s threshold *T*:

$$y = g\left(\sum_j w_j I_j - T\right)$$

If neuron is firing, then its output *y* is 1, otherwise it is 0.

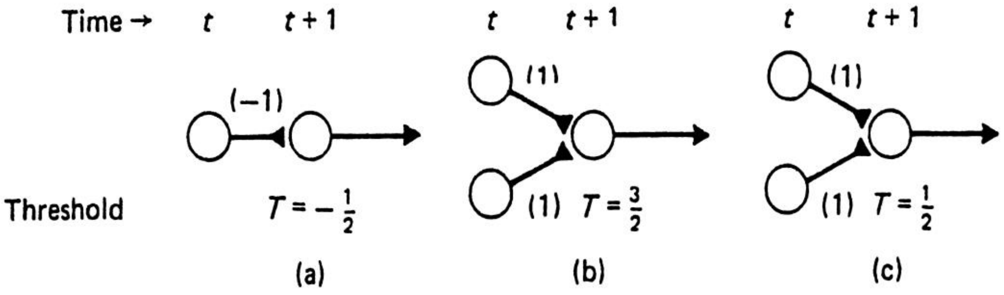
g is the unit step function: $g(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$



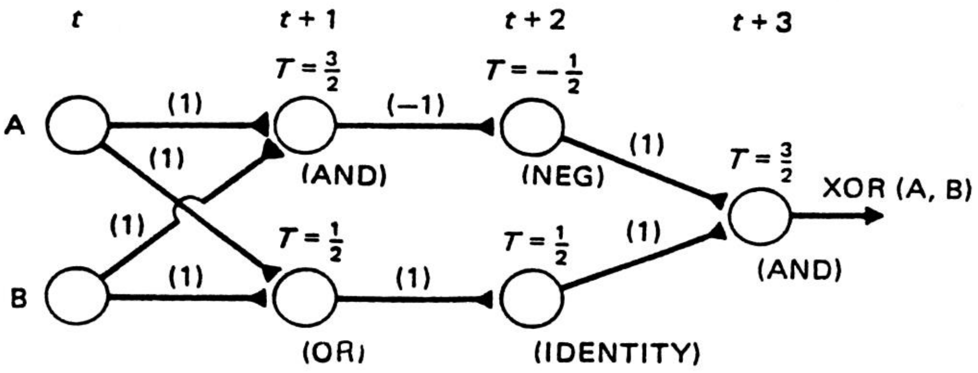
Weights w_{ij} represent the strength of the synapse between neuron *j* and neuron *i*

Properties of McCulloch-Pitts Networks

By properly combining MP neurons one can simulate the behavior of any Boolean circuit.



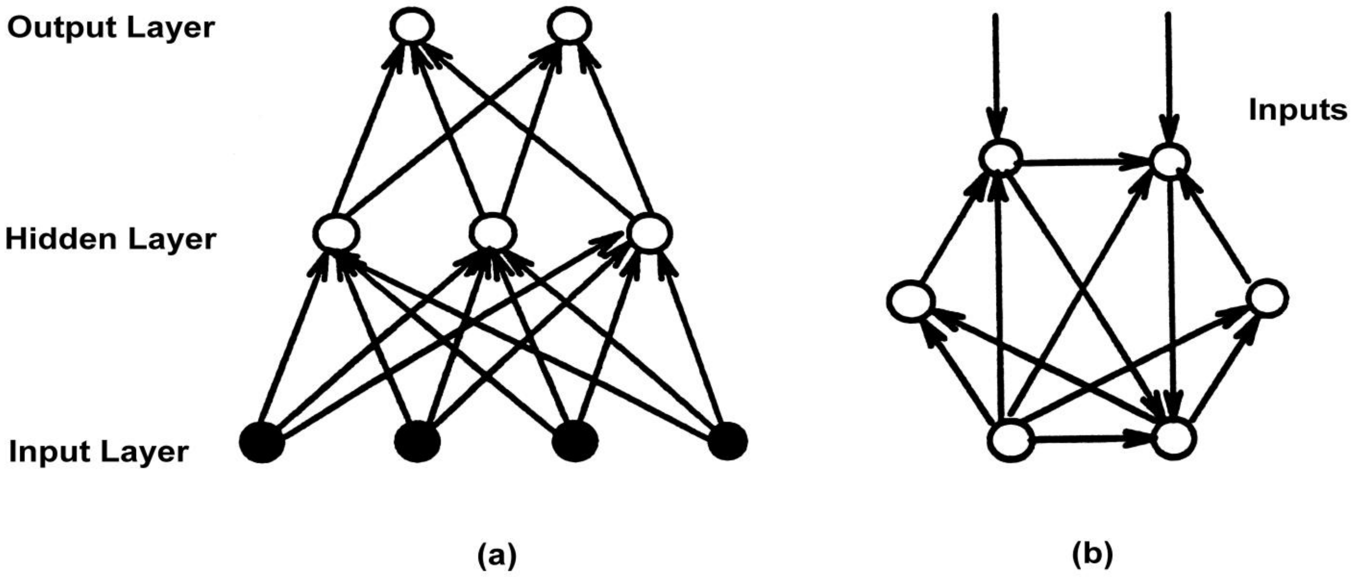
Three elementary logical operations (a) **negation**, (b) **and**, (c) **or**. In each diagram the states of the neurons on the left are at time *t* and those on the right at time *t* +1.



Network Topologies and Architectures

- Feedforward only *vs.* Feedback loop (Recurrent networks)
- Fully connected *vs.* sparsely connected
- Single layer *vs.* multilayer

Multilayer perceptrons, Hopfield networks,
Boltzman machines, Kohonen networks, …



Classification Problems

Given :

1) some “features”: $f_1, f_2, \ldots, f_n$

2) some “classes”: $c_1, \ldots, c_m$

Problem :

To classify an “object” according to its features

Example #1

To classify an “object” as :

c_1 = “ watermelon ”

c_2 = “ apple ”

c_3 = “ orange ”

According to the following features :

f_1 = “ weight ”

f_2 = “ color ”

f_3 = “ size ”

Example :

weight = 80 g

color = green

size = 10 cm^3

→ “apple”

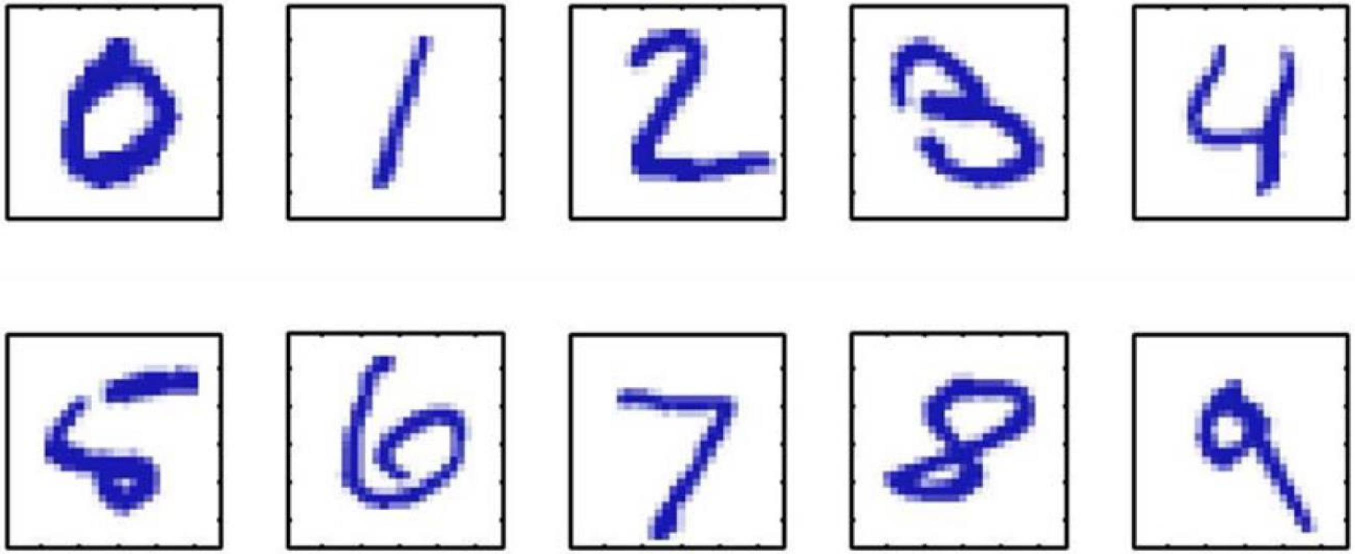
Example #2

Problem: Establish whether a patient got the flu

- Classes : { “ flu ” , “ non-flu ” }
- (Potential) Features :

f_1	:	Body temperature	
f_2	:	Headache ?	(yes / no)
f_3	:	Throat is red ?	(yes / no / medium)
f_4	:		

Example #3
Hand-written digit recognition



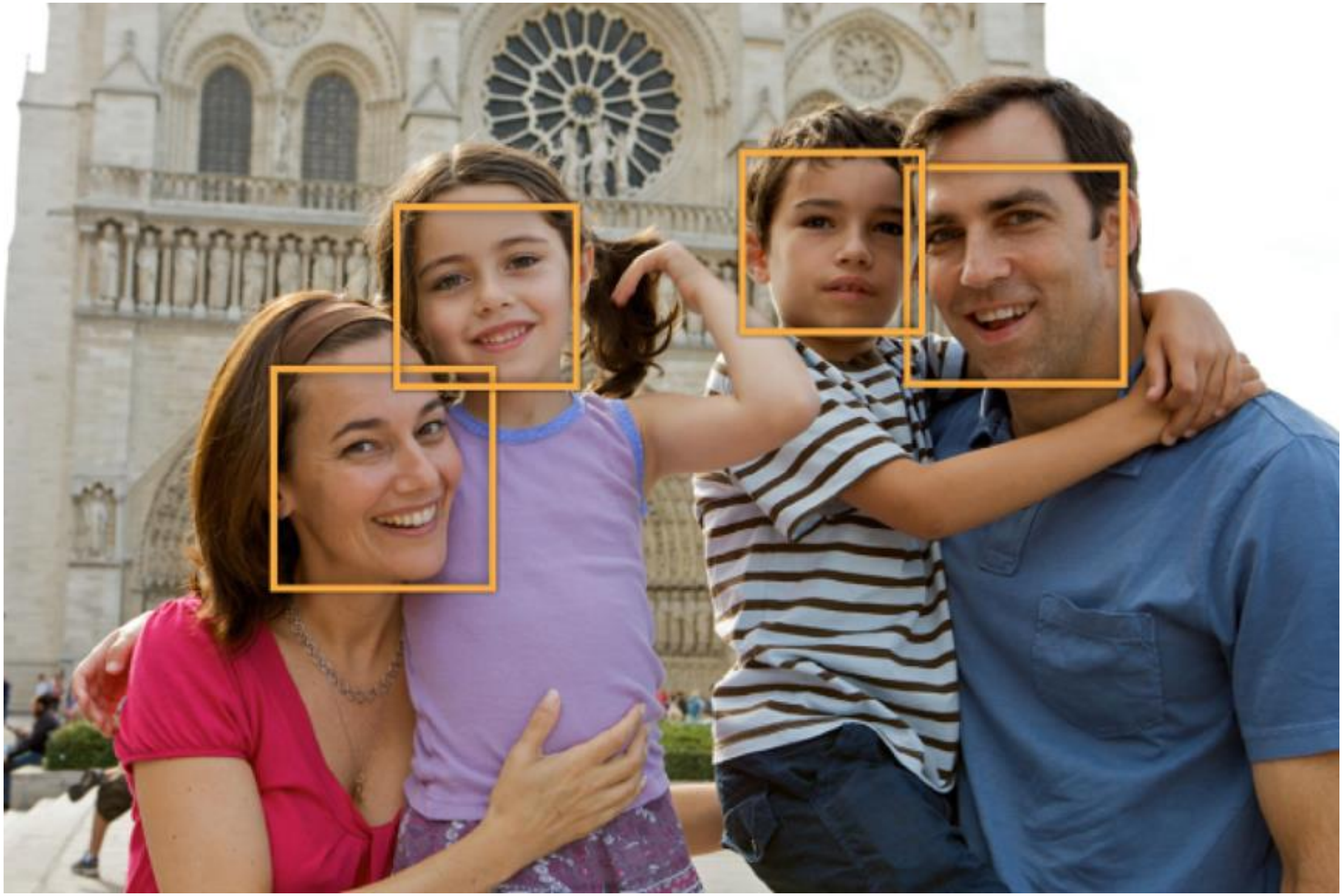
Images are 28 x 28 pixels

Represent input image as a vector $\mathbf{x} \in \mathbb{R}^{784}$

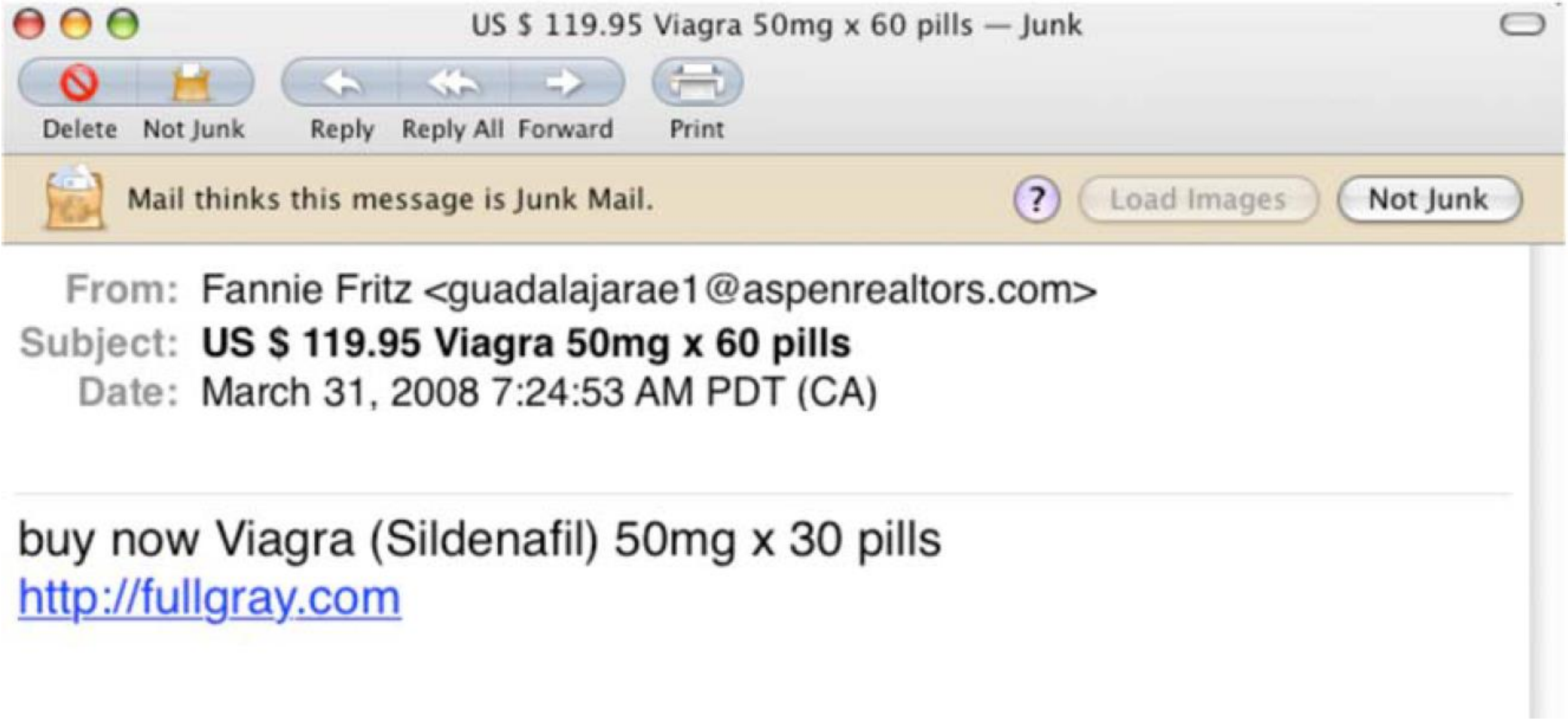
Learn a classifier $f(\mathbf{x})$ such that,

$$f : \mathbf{x} \rightarrow \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

Example #4:
Face Detection



Example #5:
Spam Detection



Mail thinks this message is Junk Mail.

From: Fannie Fritz <guadalajarae1@aspenrealtors.com>
Subject: **US $ 119.95 Viagra 50mg x 60 pills**
Date: March 31, 2008 7:24:53 AM PDT (CA)

buy now Viagra (Sildenafil) 50mg x 30 pills
http://fullgray.com

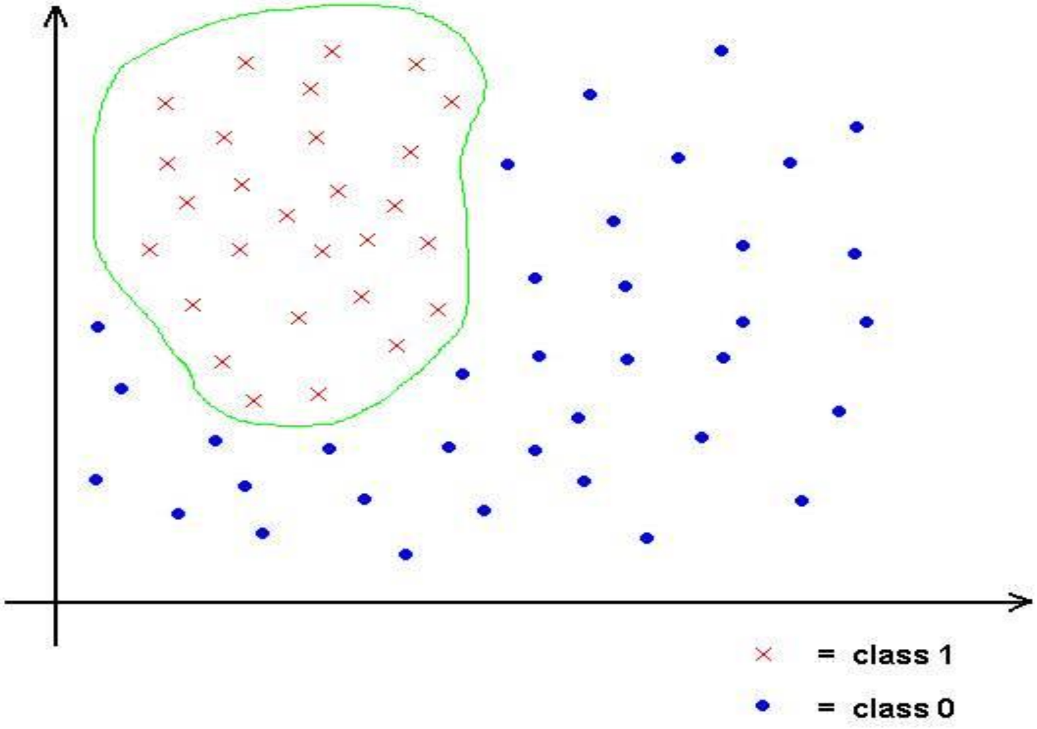
Geometric Interpretation

Example:

Classes = { 0 , 1 }

Features = x , y : both taking value in [0 , +∞ [

Idea: Objects are represented as “point” in a geometric space



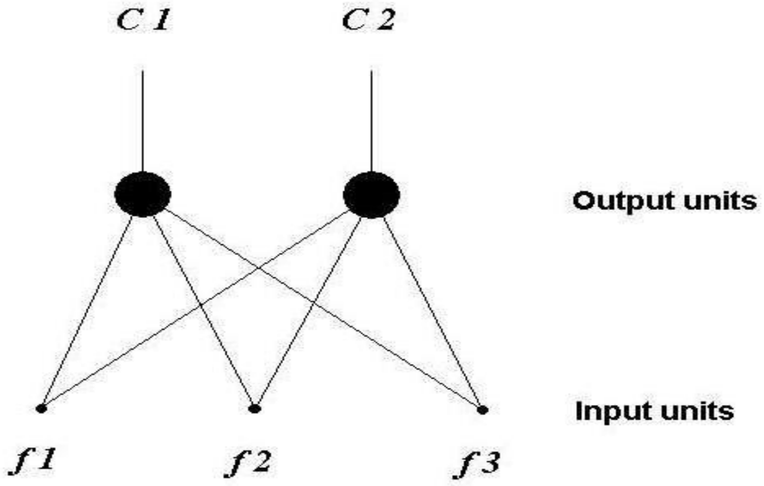
Neural Networks for Classification

A neural network can be used as a classification device .

Input ≡ features values

Output ≡ class labels

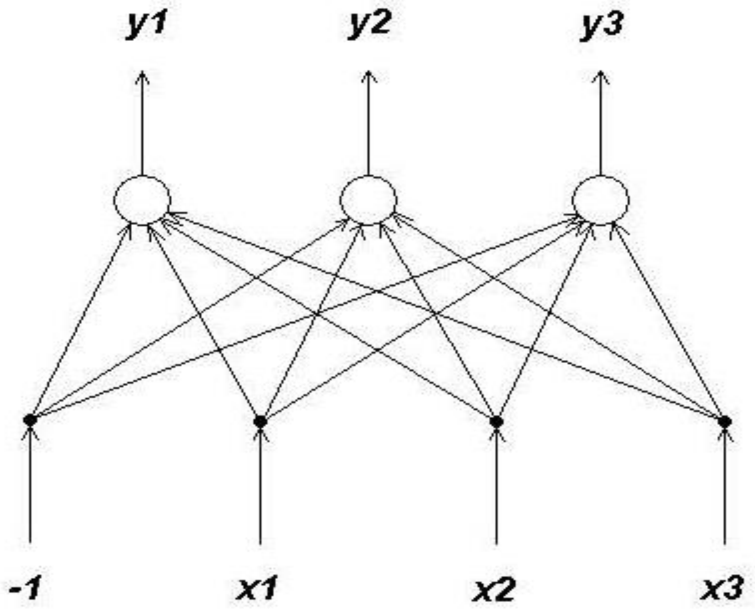
Example : 3 features , 2 classes



Thresholds

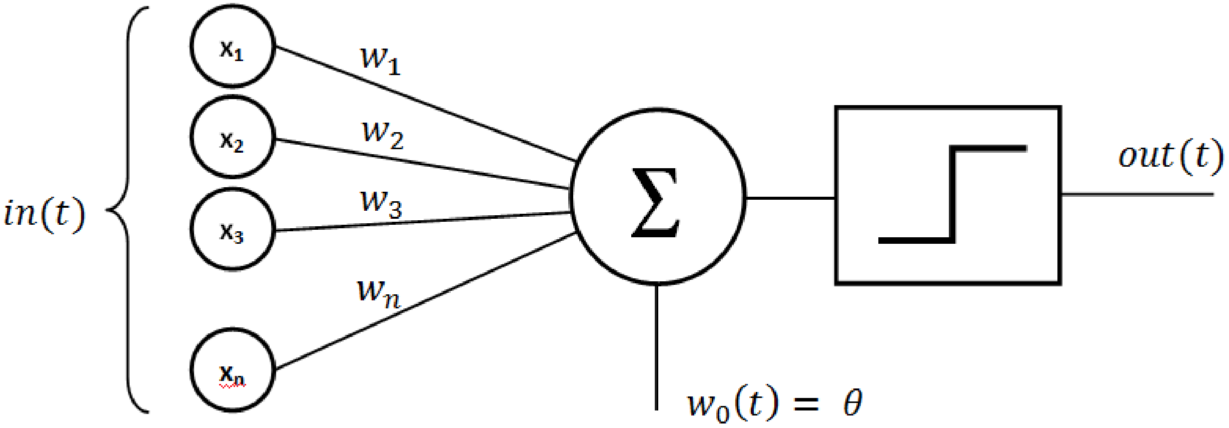
We can get rid of the thresholds associated to neurons by adding an extra unit **permanently clamped at -1**.

In so doing, thresholds become weights and can be adaptively adjusted during learning.



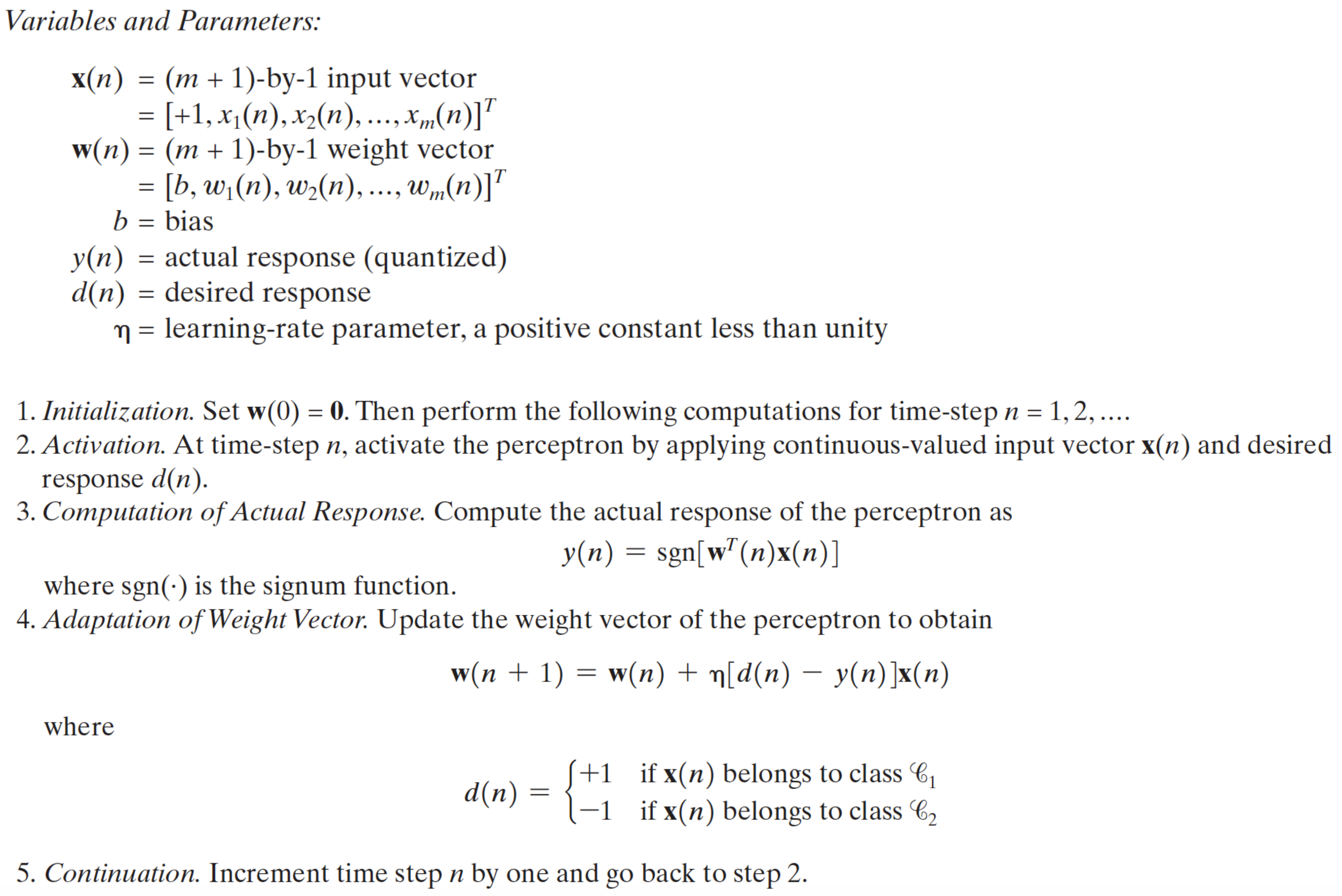
The Perceptron

A network consisting of one layer of M&P neurons connected in a feedforward way (i.e. no lateral or feedback connections).



- Discrete output (+1 / -1)
- Capable of “learning” from examples (Rosenblatt)
- They suffer from serious computational limitations

The Perceptron Learning Algorithm



Variables and Parameters:

$$
\begin{aligned}
\mathbf{x}(n) &= (m+1)\text{-by-1 input vector} \\
&= [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T \\
\mathbf{w}(n) &= (m+1)\text{-by-1 weight vector} \\
&= [b, w_1(n), w_2(n), \ldots, w_m(n)]^T \\
b &= \text{bias} \\
y(n) &= \text{actual response (quantized)} \\
d(n) &= \text{desired response} \\
\eta &= \text{learning-rate parameter, a positive constant less than unity}
\end{aligned}
$$

1. *Initialization.* Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time-step $n = 1, 2, \ldots$.
2. *Activation.* At time-step n, activate the perceptron by applying continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$.
3. *Computation of Actual Response.* Compute the actual response of the perceptron as

   $$y(n) = \operatorname{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$

   where $\operatorname{sgn}(\cdot)$ is the signum function.
4. *Adaptation of Weight Vector.* Update the weight vector of the perceptron to obtain

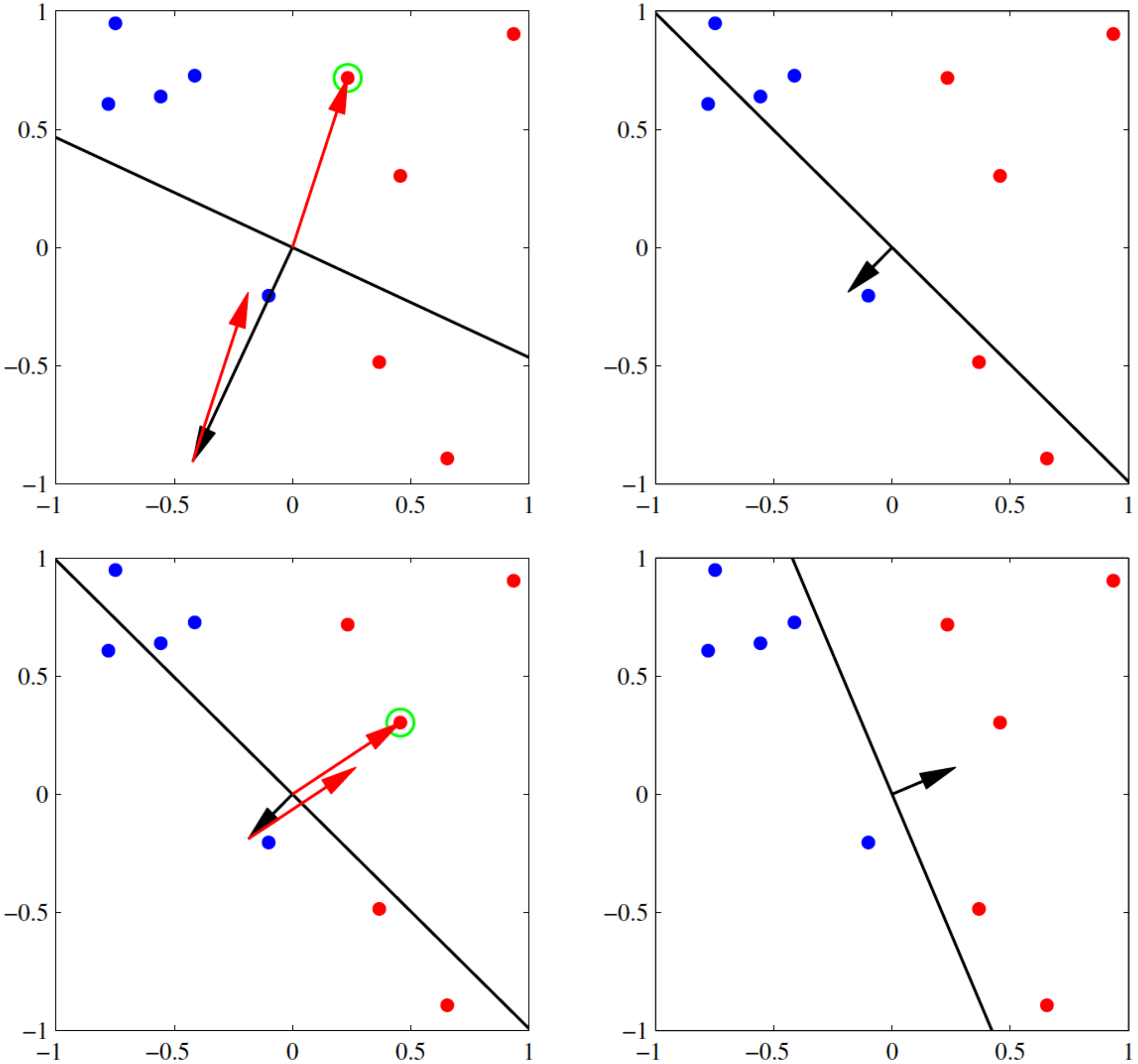
   $$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$

   where

   $$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{cases}$$

5. *Continuation.* Increment time step n by one and go back to step 2.

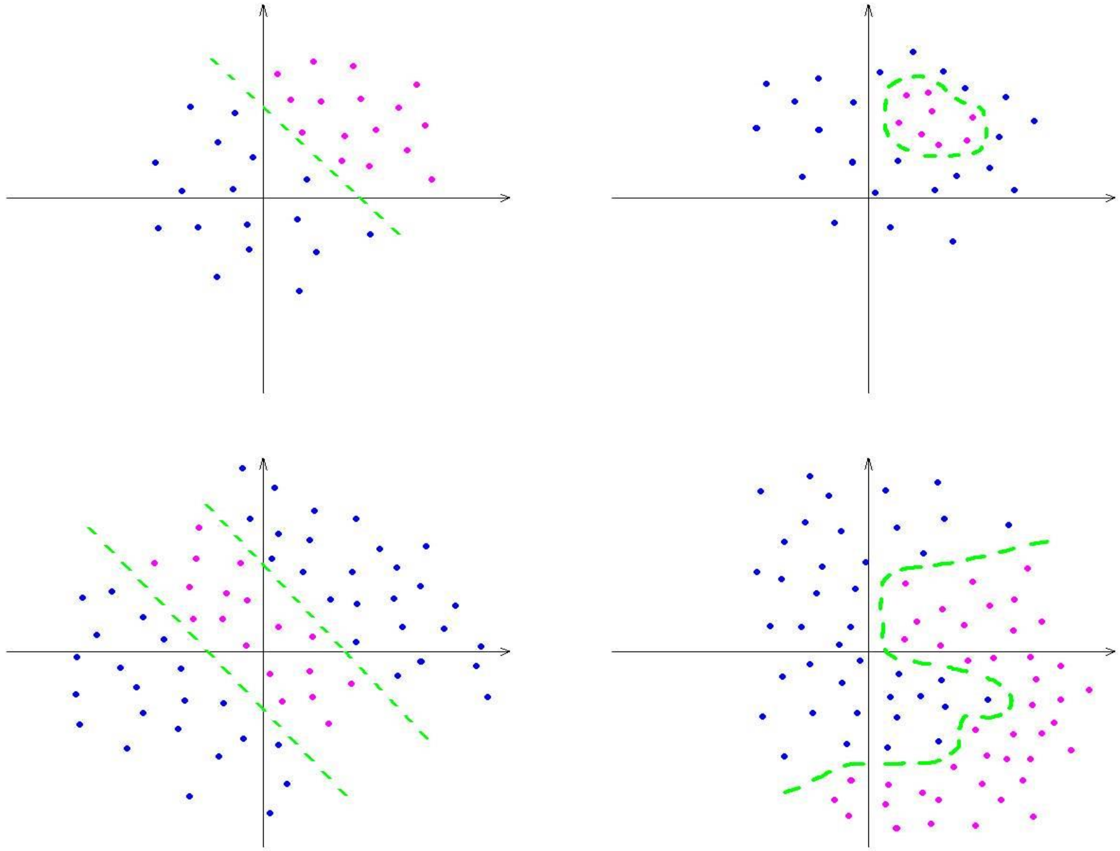
The Perceptron Learning Algorithm



Decision Regions

It’s an area wherein all examples of one class fall.

Examples:

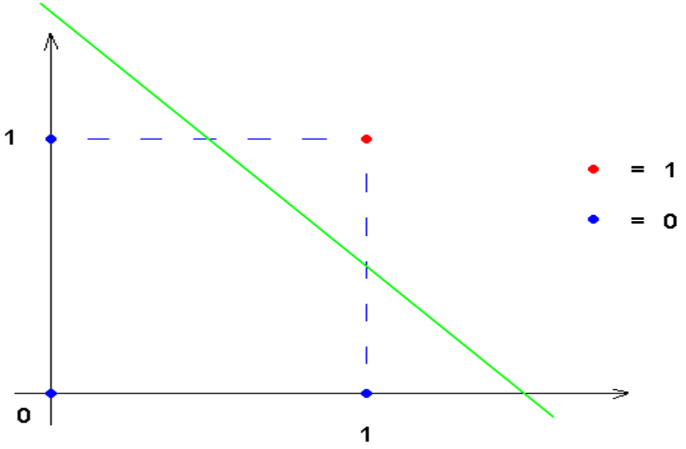


Linear Separability

A classification problem is said to be **linearly separable** if the decision regions can be separated by a hyperplane.

Example: AND

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

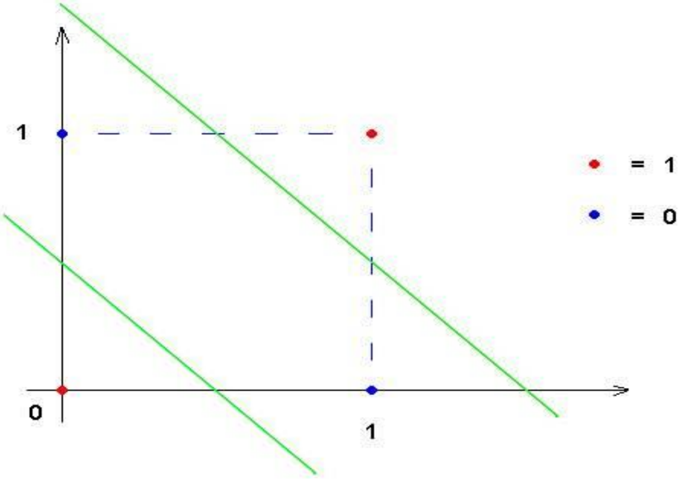


Limitations of Perceptrons

It has been shown that perceptrons can only solve linearly separable problems.

Example: XOR (exclusive OR)

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0



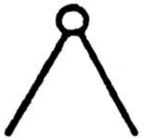
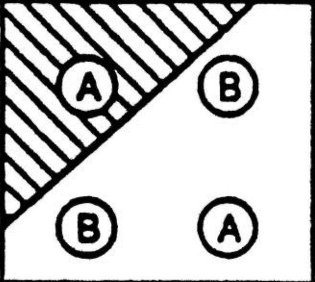
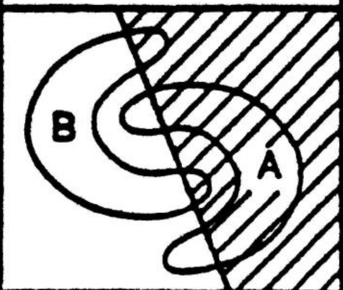
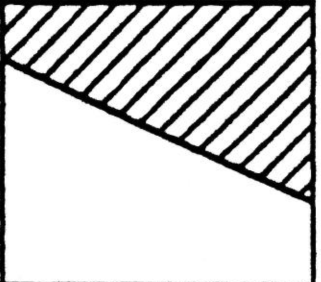
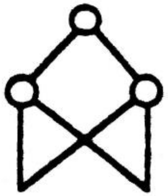
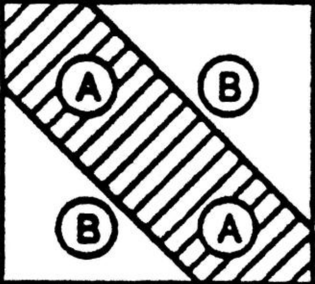
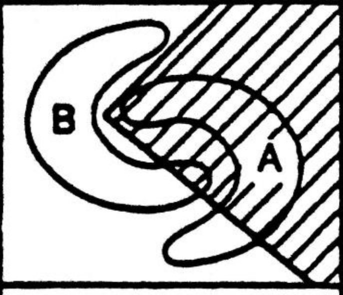
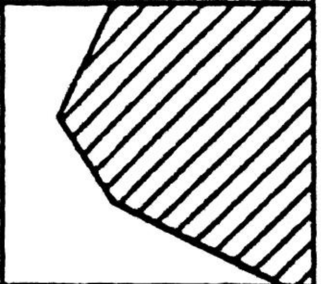
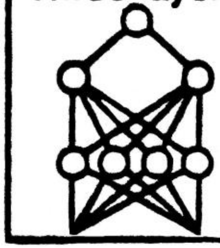
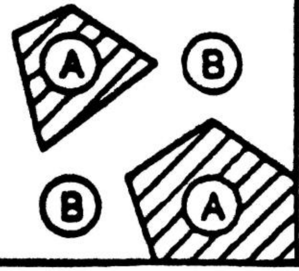
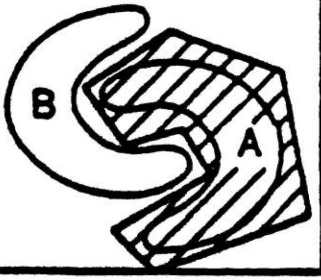
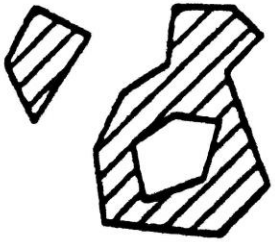
The Perceptron Convergence Theorem

Theorem (Rosenblatt, 1960)

If the training set is **linearly separable**, the perceptron learning algorithm **always converges** to a consistent hypothesis after a **finite** number of epochs, for any $\eta > 0$.

If the training set is **not** linearly separable, after a certain number of epochs the weights start oscillating.

A View of the Role of Units

Structure	Type of Decision Regions	Exclusive-OR Problem	Classes with Meshed Regions	Most General Region Shapes
Single-layer 	Half plane bounded by hyperplane			
Two-layers 	Convex open or closed regions			
Three-layers 	Arbitrary (Complexity limited by number of nodes)			

Multi–Layer Feedforward Networks

- Limitation of simple perceptron: can implement only linearly separable functions

- Add “ hidden” layers between the input and output layer. A network with just one hidden layer can represent any Boolean functions including XOR

- Power of multilayer networks was known long ago, but algorithms for training or learning, e.g. back-propagation method, became available only recently (invented several times, popularized in 1986)

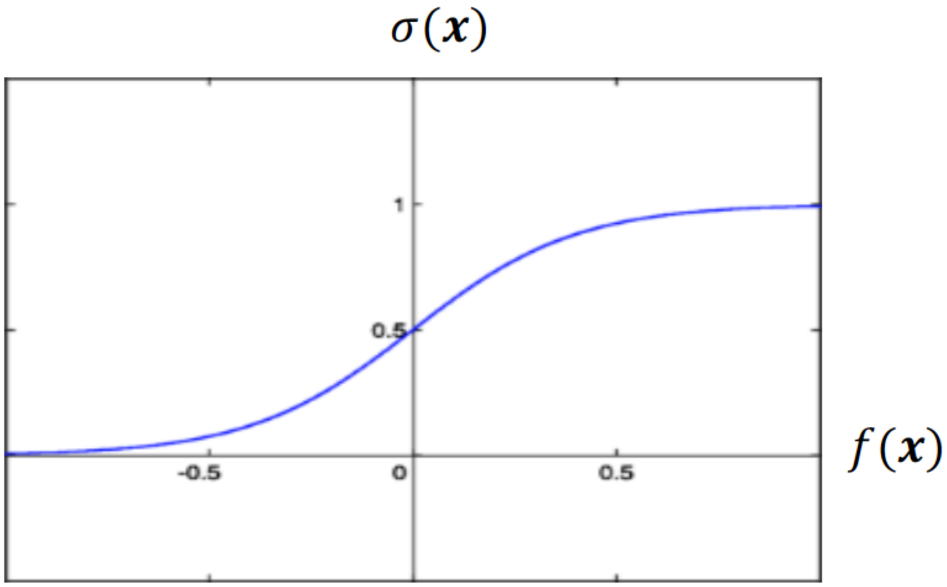
- **Universal approximation power:** Two-layer network can approximate any smooth function (Cybenko, 1989; Funahashi, 1989; Hornik, et al.., 1989)

- Static (no feedback)

Continuous-Valued Units

Sigmoid (or logistic)

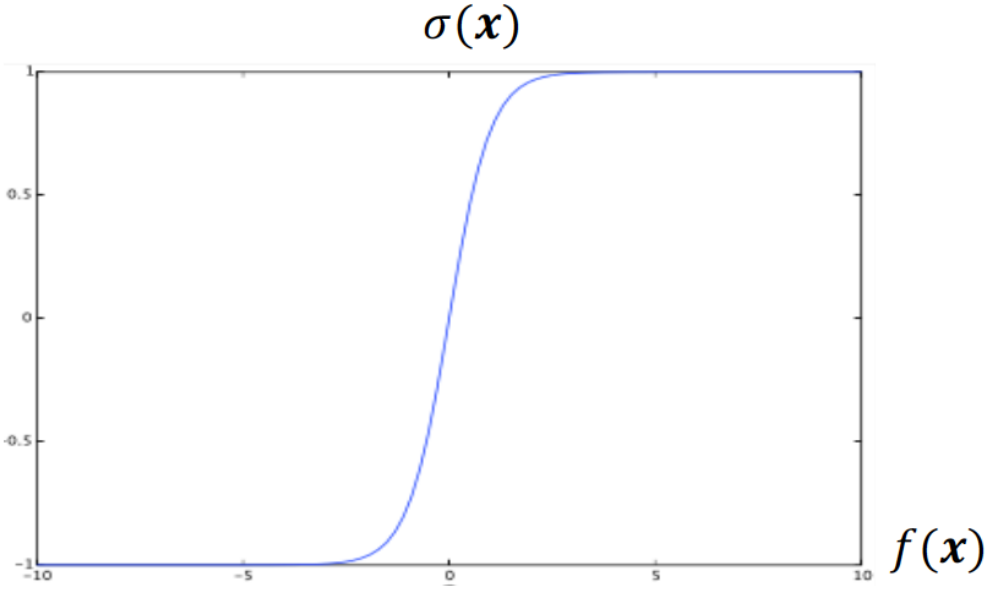
$$\sigma(\boldsymbol{x}) = \frac{1}{1 + e^{-f(\boldsymbol{x})}} \in (0,1)$$



Continuous-Valued Units

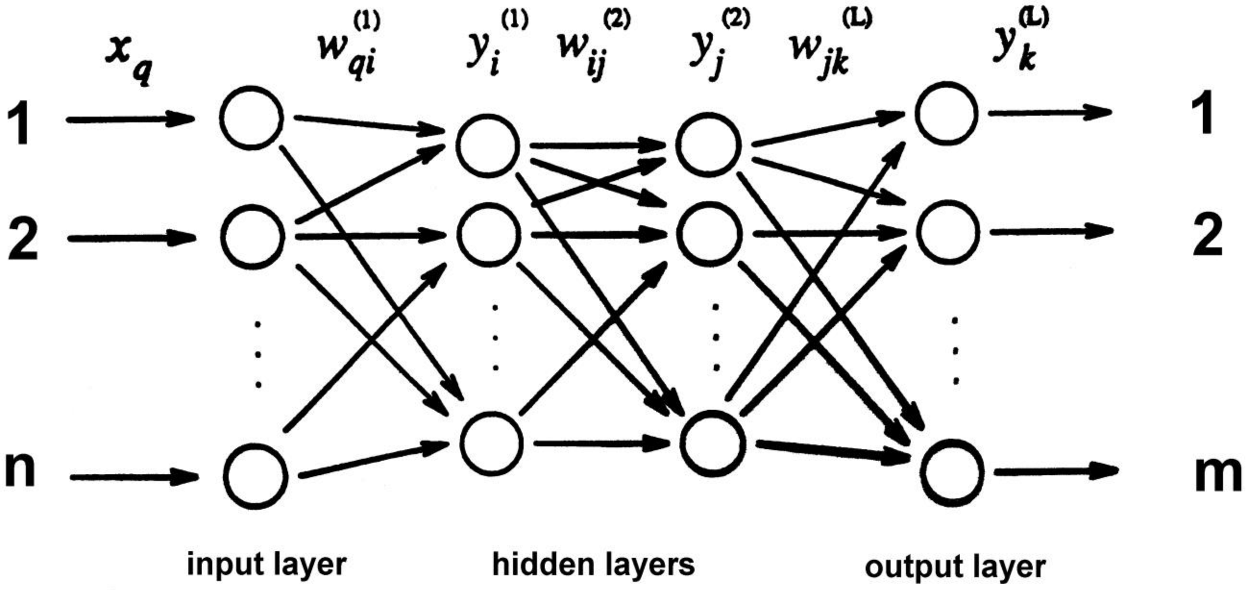
Hyperbolic tangent

$$\sigma(\boldsymbol{x}) = \frac{e^{f(\boldsymbol{x})} - e^{-f(\boldsymbol{x})}}{e^{f(\boldsymbol{x})} + e^{-f(\boldsymbol{x})}} \in (-1,1)$$



Back-propagation Learning Algorithm

- An algorithm for learning the weights in a feed-forward network, given a training set of input-output pairs
- The algorithm is based on gradient descent method.



Supervised Learning

Supervised learning algorithms require the presence of a “teacher” who provides the right answers to the input questions.

Technically, this means that we need a **training set** of the form

$$L = \left\{ \left(\mathrm{x}^1, \mathrm{y}^1\right), \quad ..... \quad \left(\mathrm{x}^p, \mathrm{y}^p\right) \right\}$$

where :

$\mathrm{x}^m \quad \left(m = 1 \ldots p\right)$ is the network input vector

$\mathrm{y}^m \quad \left(m = 1 \ldots p\right)$ is the **desired** network output
vector

Supervised Learning

The learning (or training) phase consists of determining a configuration of weights in such a way that the network output be as close as possible to the desired output, for all the examples in the training set.

Formally, this amounts to minimizing an **error function** such as (not only possible one):

$$E = \frac{1}{2} \underset{m}{\text{å}} \; \underset{k}{\text{å}} \left(y_k^m - O_k^m \right)^2$$

where O_k^μ is the output provided by the output unit k when the network is given example μ as input.

Back-Propagation

To minimize the error function *E* we can use the classic **gradient-descent** algorithm:

$$w_{ji} \leftarrow w_{ji} - \eta \frac{\partial E_k}{\partial w_{ji}}$$

η = “learning rate”

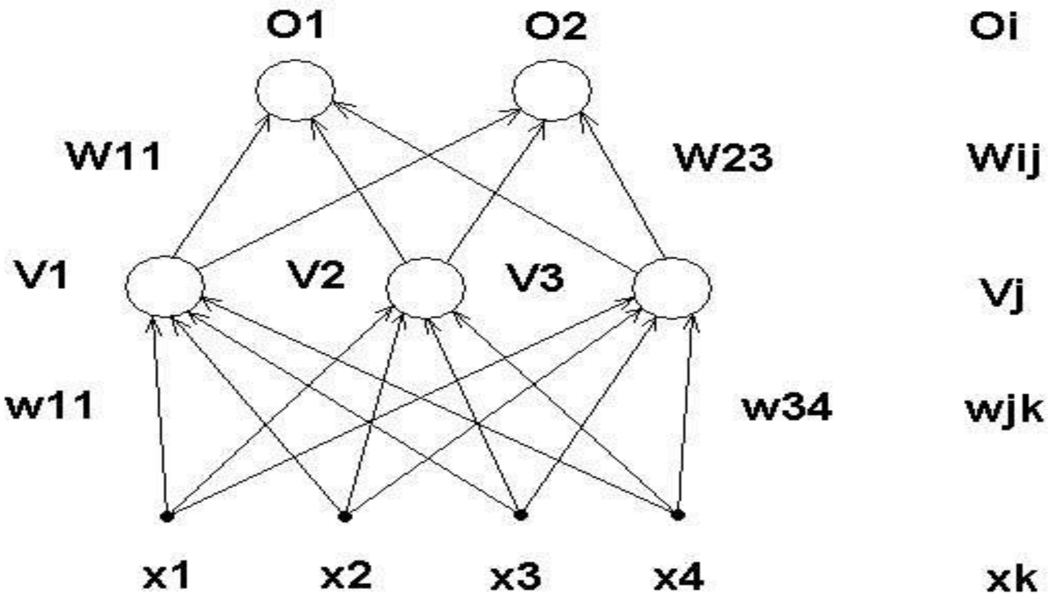
To compute the partial derivates we use the **error back propagation** algorithm.

It consists of two stages:

Forward pass : the input to the network is propagated layer after layer in forward direction

Backward pass : the “error” made by the network is propagated backward, and weights are updated properly

Notations



Given pattern *μ*, hidden unit *j* receives a net input

$$h_j^\mu = \sum_k w_{jk} \, x_k^\mu$$

and produces as output :

$$V_j^\mu = g\left(h_j^\mu\right) = g\left(\sum_k w_{jk} \, x_k^\mu\right)$$

Back-Prop:
Updating Hidden-to-Output Weights

$$E = \frac{1}{2} \underset{m}{\text{å}} \; \underset{k}{\text{å}} \left(y_k^m - O_k^m \right)^2$$

$$\text{D}W_{ij} = - h \frac{\partial E}{\partial W_{ij}}$$

$$= - h \frac{\partial}{\partial W_{ij}} \left[\frac{1}{2} \sum_m \sum_k \left(y_k^m - O_k^m \right)^2 \right]$$

$$= h \sum_m \sum_k \left(y_k^m - O_k^m \right) \frac{\partial O_k^m}{\partial W_{ij}}$$

$$= h \sum_m \left(y_i^m - O_i^m \right) \frac{\partial O_i^m}{\partial W_{ij}}$$

$$= h \sum_m \left(y_i^m - O_i^m \right) g'\left(h_i^m \right) V_j^m$$

$$= h \sum_m d_i^m V_j^m$$

where : $\delta_i^\mu = \left(y_i^\mu - O_i^\mu \right) g'\left(h_i^\mu \right)$

Back-Prop:
Updating Input-to-Hidden Weights (1)

$$E = \frac{1}{2} \underset{m}{\text{å}} \; \underset{k}{\text{å}} \left(y_k^m \text{- } O_k^m \right)^2$$

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}}$$

$$= \eta \sum_{\mu} \sum_{i} \left(y_i^\mu - O_i^\mu \right) \frac{\partial O_i^\mu}{\partial w_{jk}}$$

$$= \eta \sum_{\mu} \sum_{i} \left(y_i^\mu - O_i^\mu \right) g'\left(h_i^\mu \right) \frac{\partial h_i^\mu}{\partial w_{jk}}$$

$$\frac{\partial h_i^\mu}{\partial w_{jk}} = \sum_{l} W_{il} \frac{\partial V_l^\mu}{\partial w_{jk}}$$

$$= W_{ij} \frac{\partial V_j^\mu}{\partial w_{jk}}$$

$$= W_{ij} \frac{\partial g\left(h_j^\mu \right)}{\partial w_{jk}}$$

$$= W_{ij} \; g'\left(h_j^\mu \right) \frac{\partial h_j^\mu}{\partial w_{jk}}$$

Back-Prop:
Updating Input-to-Hidden Weights (2)

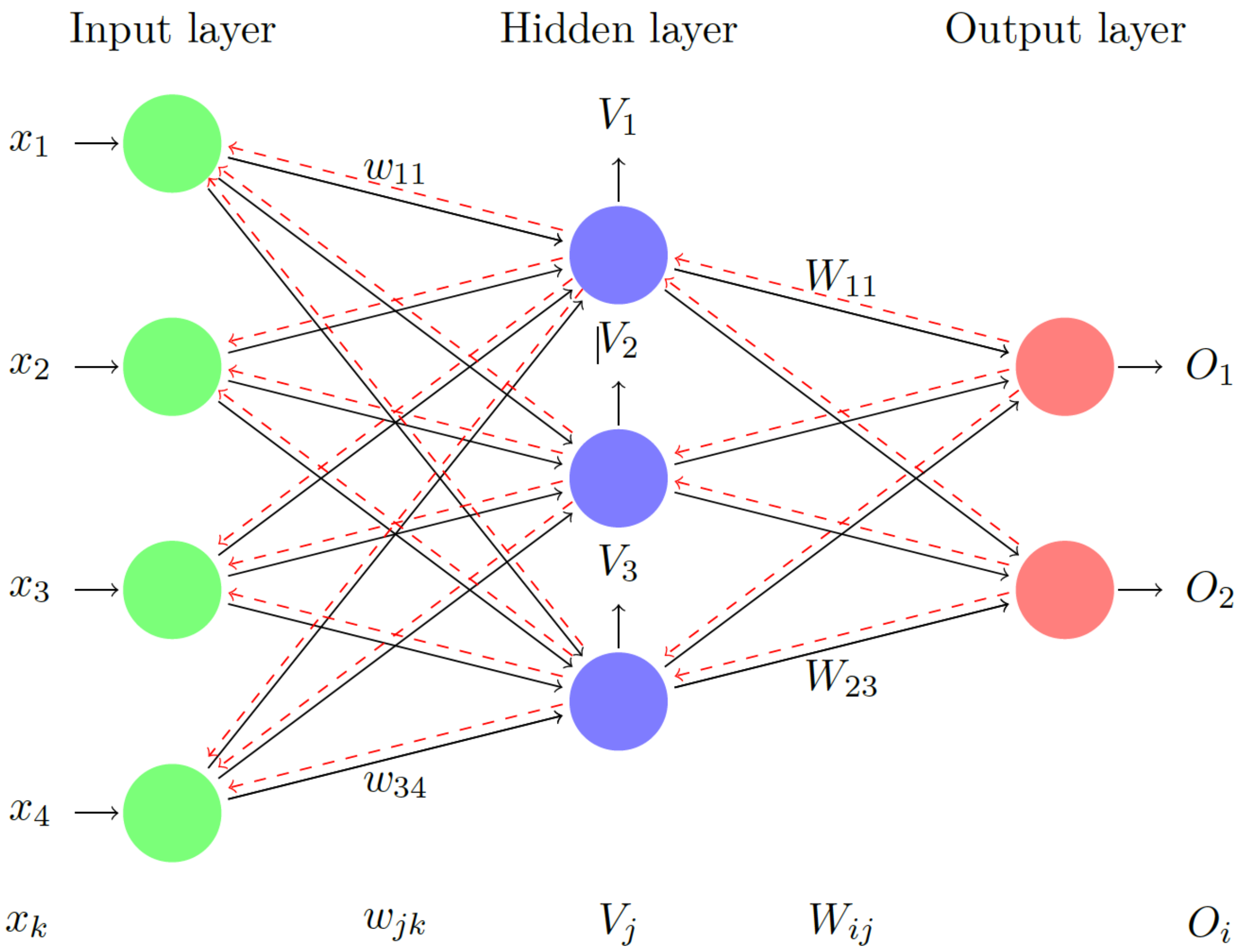
$$\frac{\partial h_j^\mu}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} \sum_m w_{jm} x_m^\mu$$
$$= x_k^\mu$$

Hence, we get:

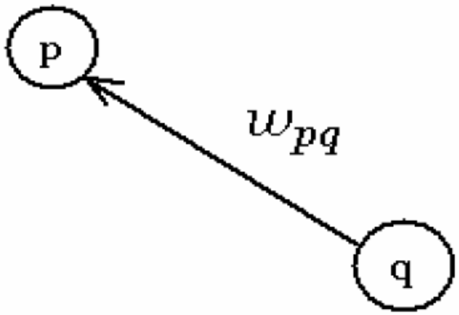
$$\Delta w_{jk} = \eta \sum_{\mu,i} \left(y_i^\mu - O_i^\mu\right) g'\left(h_i^\mu\right) W_{ij}\ g'\left(h_j^\mu\right) x_k^\mu$$
$$= \eta \sum_{\mu,i} \delta_i^\mu\ W_{ij}\ g'\left(h_j^\mu\right) x_k^\mu$$
$$= \eta \sum_{\mu} \hat{\delta}_j^\mu\ x_k^\mu$$

where : $\hat{\delta}_j^\mu = g'\left(h_j^\mu\right) \sum_i \delta_i^\mu\ W_{ij}$

Error Back-Propagation



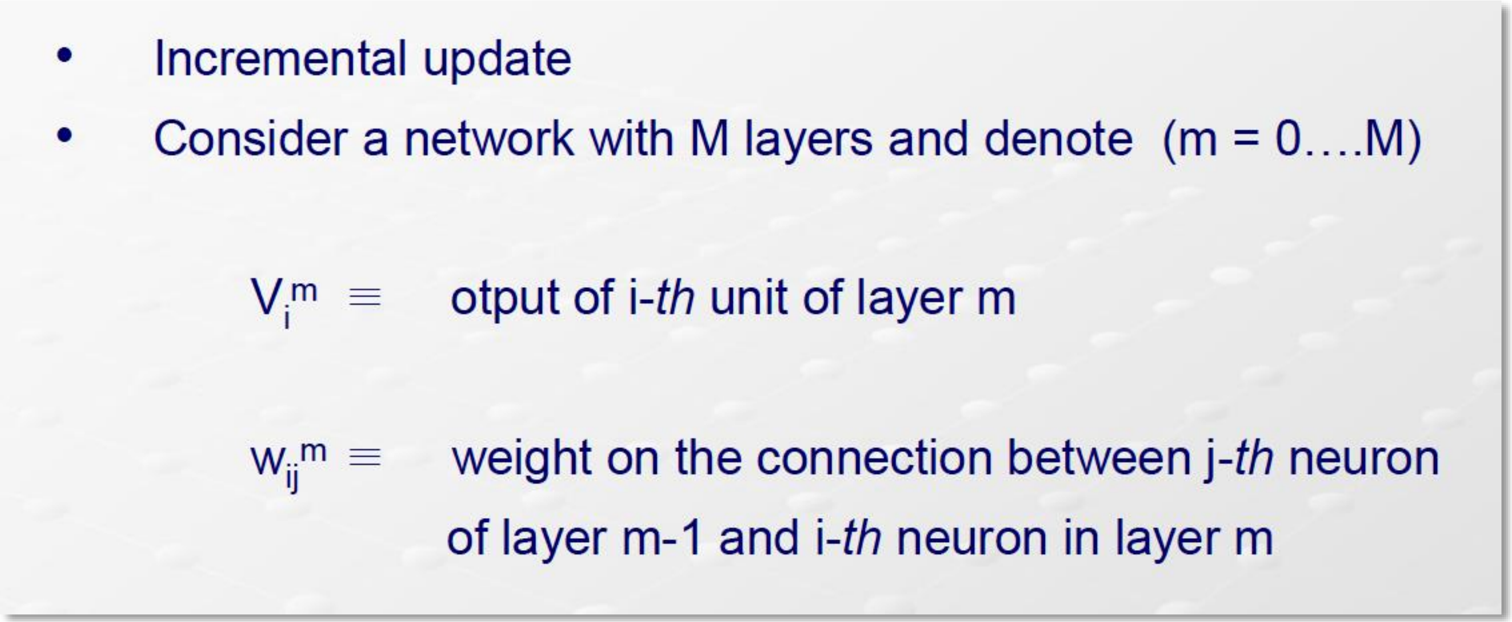
Locality of Back-Prop



$$\Delta\omega_{pq} = \eta \sum_{\mu} \delta_p^{\mu} V_q^{\mu}$$ off - line

$$\Delta\omega_{pq} = \eta\, \delta_p^{\mu} V_q^{\mu}$$ on - line

The Back-Propagation Algorithm



- Incremental update
- Consider a network with M layers and denote (m = 0….M)

$V_i^m \equiv$ otput of i-*th* unit of layer m

$w_{ij}^m \equiv$ weight on the connection between j-*th* neuron of layer m-1 and i-*th* neuron in layer m

The Back-Propagation Algorithm

1. Initialize the weight to (small) random values
2. Choose a pattern $\overline{x}^{\mu}$ and apply it to the input layer (m=0)

$$V_k^0 = x_k^{\mu} \qquad \forall\, k$$

3. Propagate the signal forward:

$$V_i^m = g\left(h_i^m\right) = g\left(\sum_j w_{ij} V_j^{m-1}\right)$$

4. Compute the δ's for the output layer:

$$\delta_i^M = g'\left(h_i^M\right)\left(y_i^M - V_i^M\right)$$

5. Compute the δ's for all preceding layers:

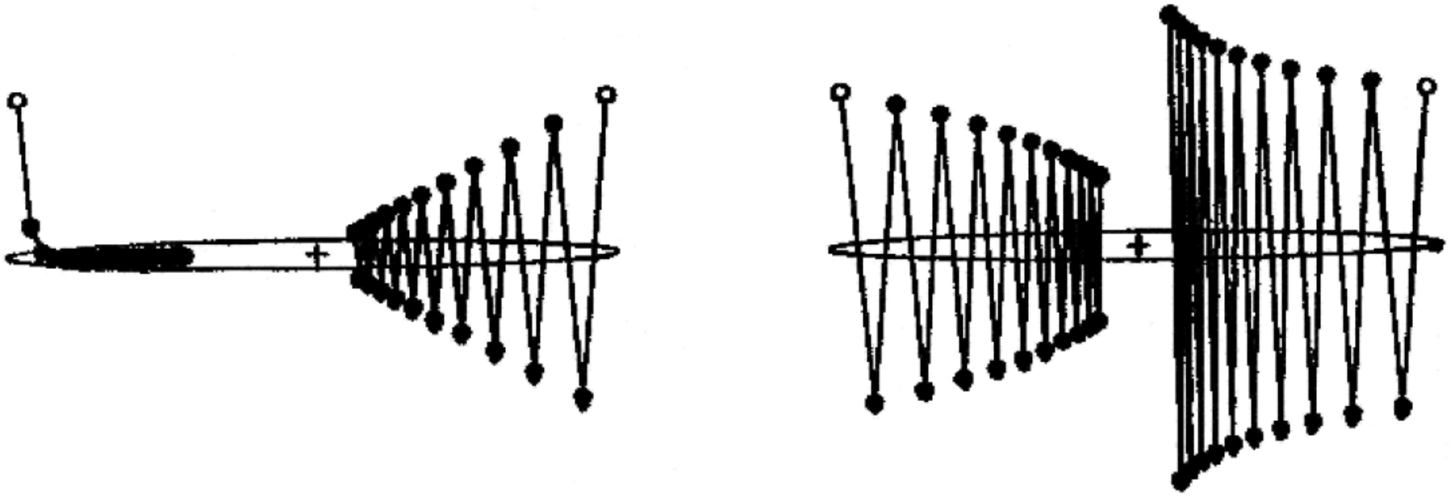
$$\delta_i^{m-1} = g'\left(h_i^{m-1}\right)\sum_j w_{ji}^m \delta_j^m$$

6. Update connection weights:

$$w_{ij}^{NEW} = w_{ij}^{OLD} + \Delta w_{ij} \qquad where \qquad \Delta w_{ij} = \eta\, \delta_i^m V_j^{m-1}$$

7. Go back to step 2 until convergence

The Role of the Learning Rate



Gradient descent on a simple quadratic surface (the left and right parts are copies of the same surface). Four trajectories are shown, each for 20 steps from the open circle. The minimum is at the + and the ellipse shows a constant error contour. The only significant difference between the trajectories is the value of *η*, which was 0.02, 0.0476, 0.049, and 0.0505 from left to right.

The Momentum Term

Gradient descent may:

- Converge too slowly if η is too small
- Oscillate if η is too large

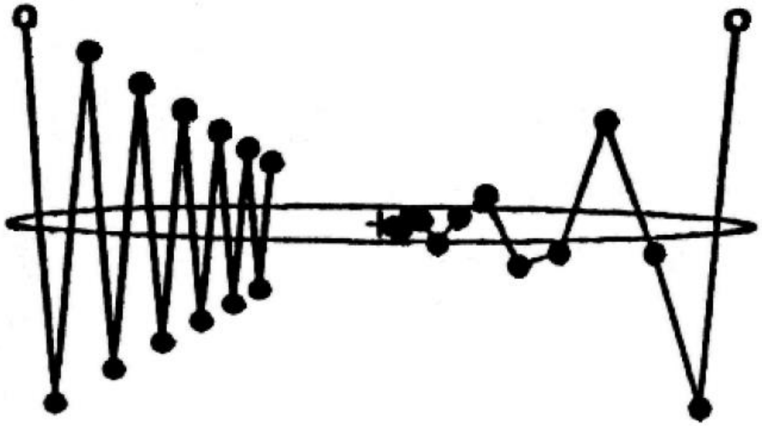
Simple remedy:

$$\Delta\omega_{pq}(t+1) = -\eta\frac{\partial E}{\partial w_{pq}} + \underbrace{\alpha\,\Delta w_{pq}(t)}_{momentum}$$

The momentum term allows us to use large values of η thereby avoiding oscillatory phenomena

Typical choice: $\alpha = 0.9$, $\eta = 0.5$

The Momentum Term



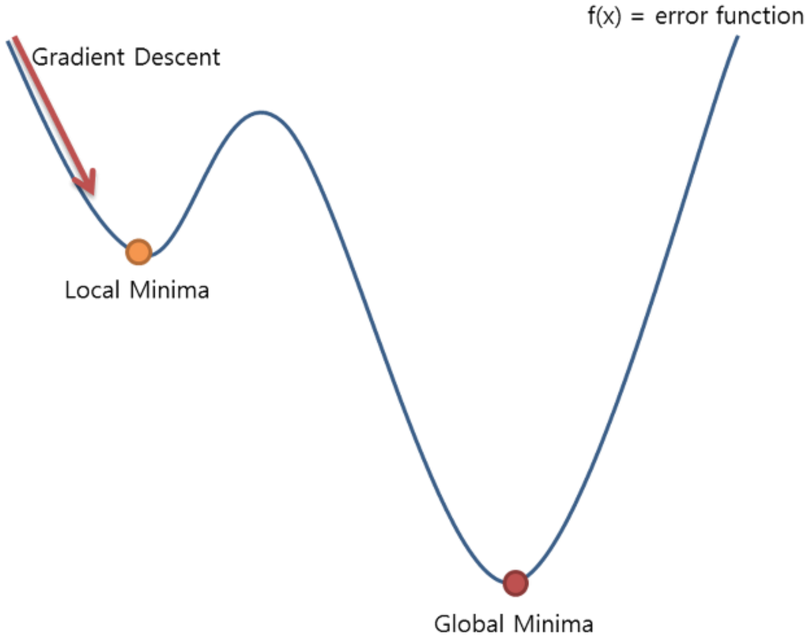
Gradient descent on the simple quadratic surface. Both trajectories are for 12 steps with $\eta = 0.0476$, the best value in the absence of momentum. On the left there is no momentum ($\alpha = 0$), while $\alpha = 0.5$ on the right.

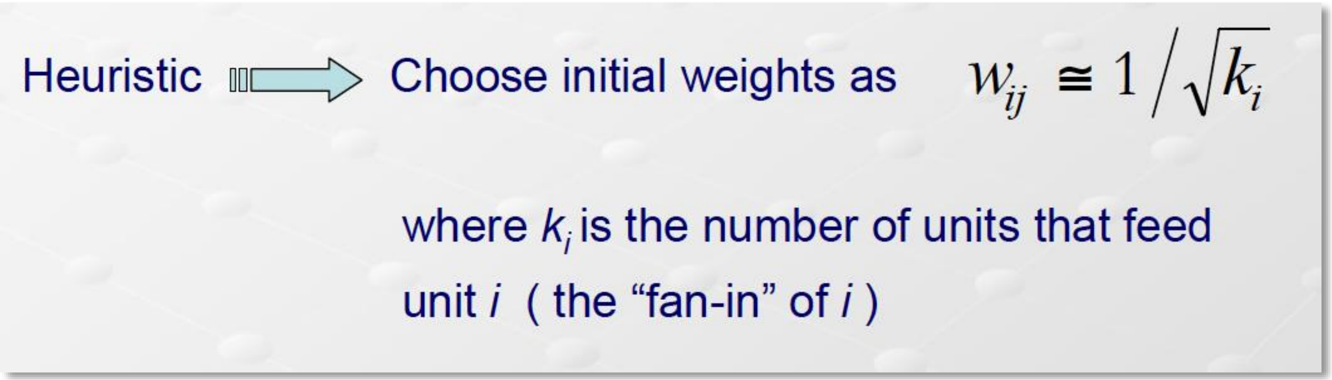
The Problem of Local Minima

Back-prop **cannot avoid local minima**.

Choice of initial weights is important.

If they are too large the nonlinearities tend to saturate since the beginning of the learning process.

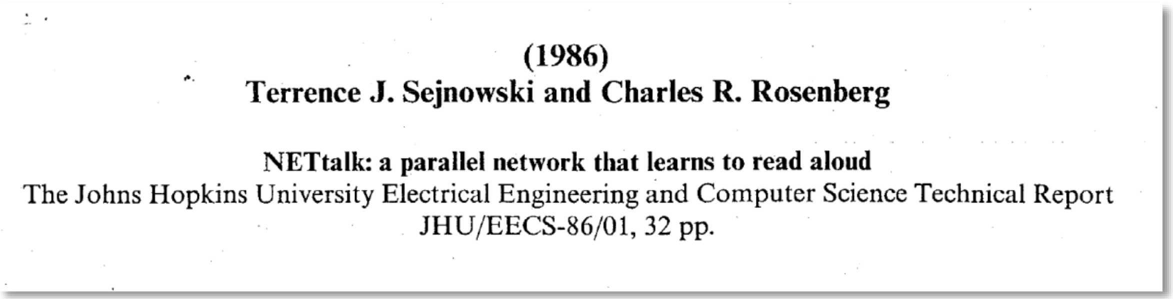




Heuristic ⇨ Choose initial weights as $w_{ij} \cong 1/\sqrt{k_i}$

where k_i is the number of units that feed unit *i* (the “fan-in” of *i*)

NETtalk



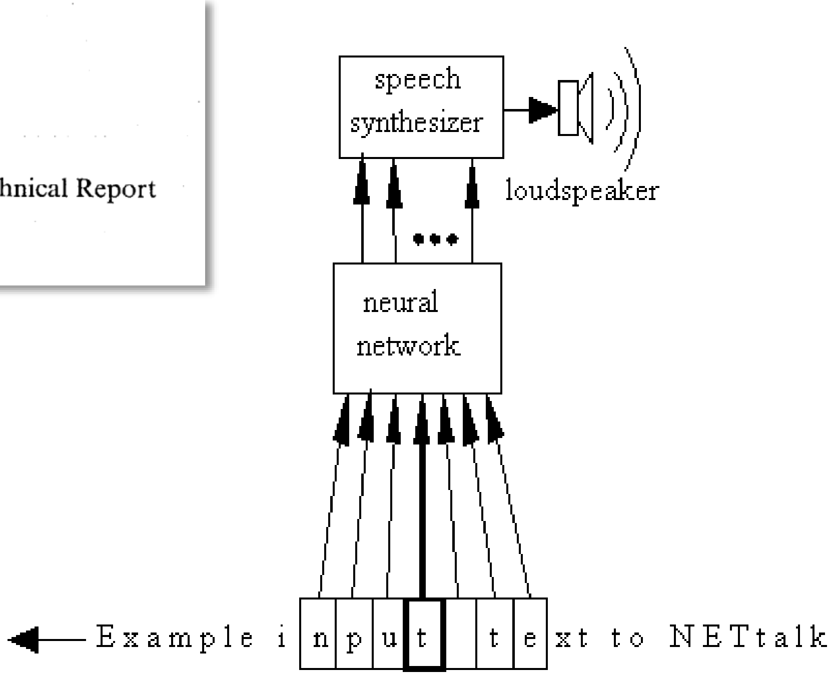
(1986)

Terrence J. Sejnowski and Charles R. Rosenberg

NETtalk: a parallel network that learns to read aloud

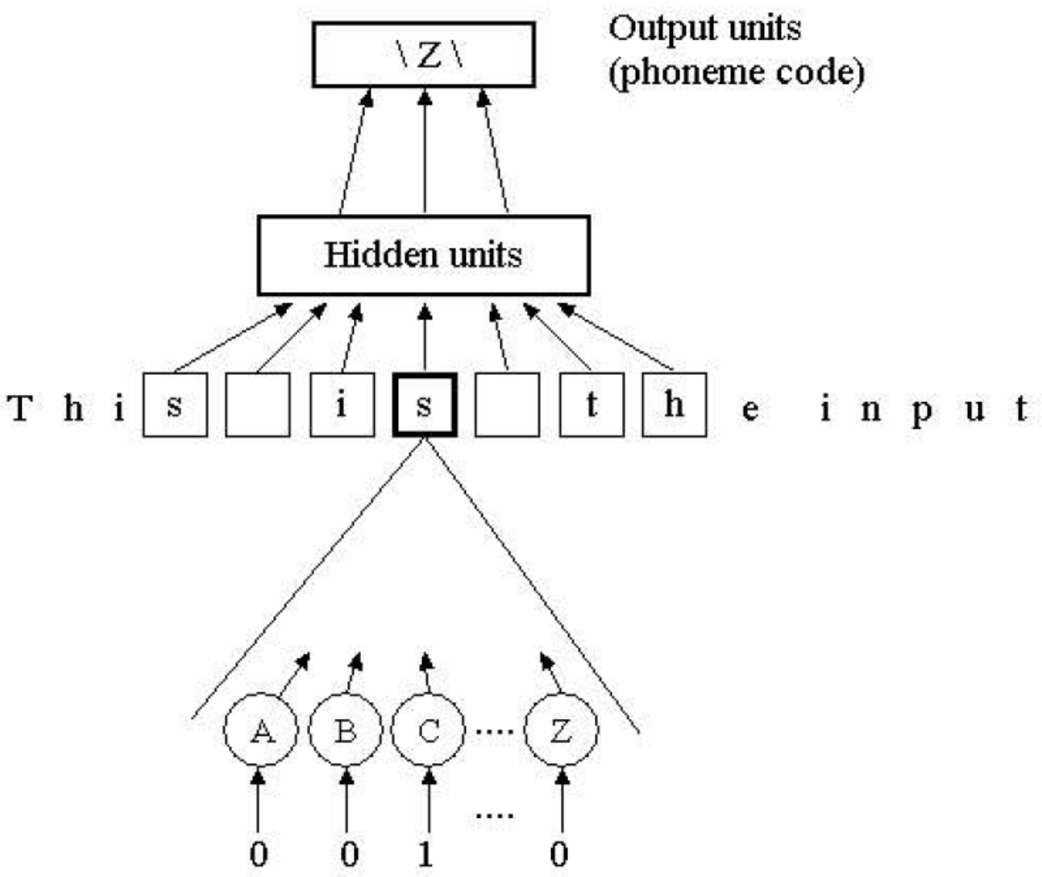
The Johns Hopkins University Electrical Engineering and Computer Science Technical Report

JHU/EECS-86/01, 32 pp.



NETtalk neural network speech synthesizer. The NETtalk backpropagation network is trained by a rule-based expert system element of the DECtalk commercial speech synthesis system. NETtalk is then used to replace that element. The result is a new speech synthesis system that has approximately the same overall performance as the original. In other words, the NETtalk neural network becomes functionally equivalent to an expert system with hundreds of rules. The question then becomes: how are these rules represented within the NETtalk neural network? The answer is: nobody really knows.

NETtalk


\ Z \
Output units
(phoneme code)
Hidden units
T h i s
i s
t h e i n p u t
A
B
C
....
Z
....
0
0
1
0

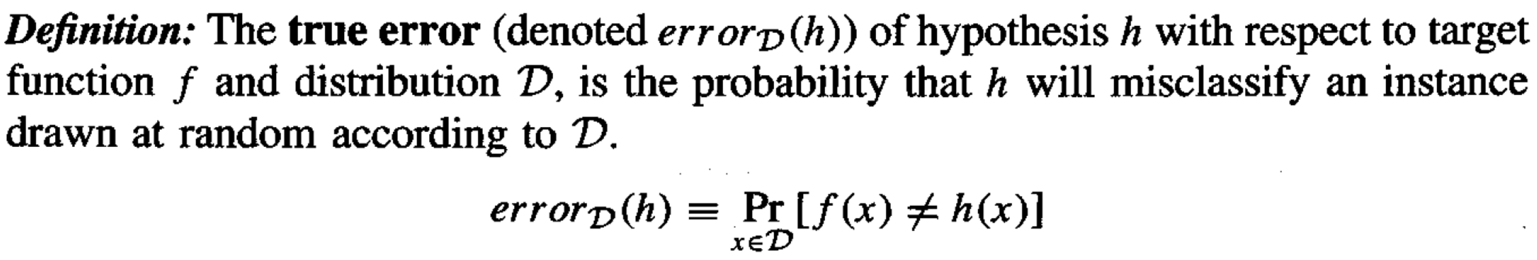
NETtalk

- A network to pronounce English text
- 7 x 29 (=203) input units
- 1 hidden layer with 80 units
- 26 output units encoding phonemes
- Trained by 1024 words in context
- Produce intelligible speech after 10 training epochs
- Functionally equivalent to DEC-talk
- Rule-based DEC-talk was the result of a decade effort by many linguists
- NETtalk learns from examples and, require no linguistic knowledge

Theoretical / Practical Questions

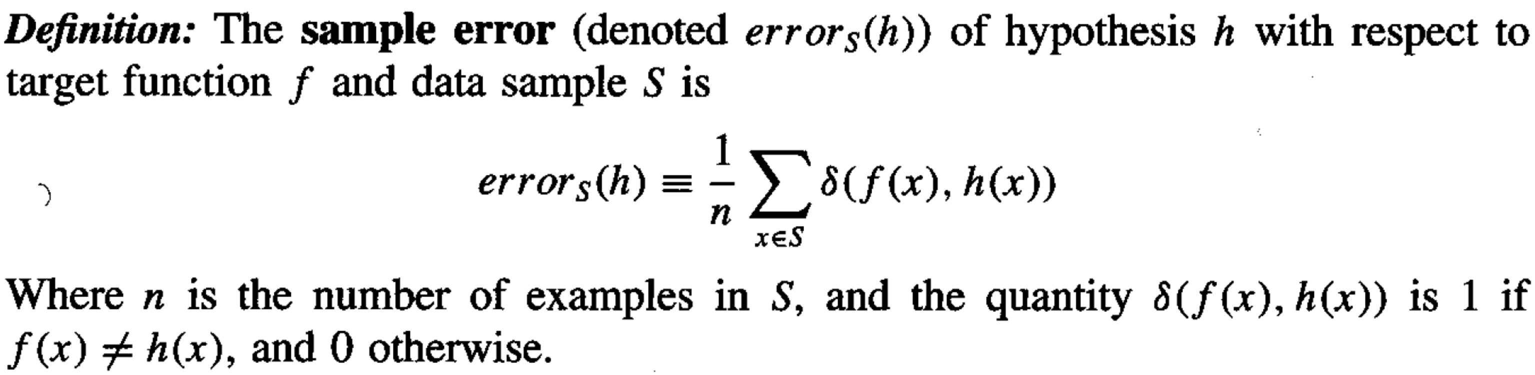
- How many layers are needed for a given task?
- How many units per layer?
- To what extent does representation matter?
- What do we mean by generalization?
- What can we expect a network to generalize?
  - Generalization: performance of the network on data not included in the training set
  - Size of the training set: how large a training set should be for "good" generalization?
  - Size of the network: too many weights in a network result in poor generalization

True *vs* Sample Error



Definition: The **true error** (denoted $error_{\mathcal{D}}(h)$) of hypothesis h with respect to target function f and distribution $\mathcal{D}$, is the probability that h will misclassify an instance drawn at random according to $\mathcal{D}$.

$$error_{\mathcal{D}}(h) \equiv \Pr_{x \in \mathcal{D}}[f(x) \neq h(x)]$$



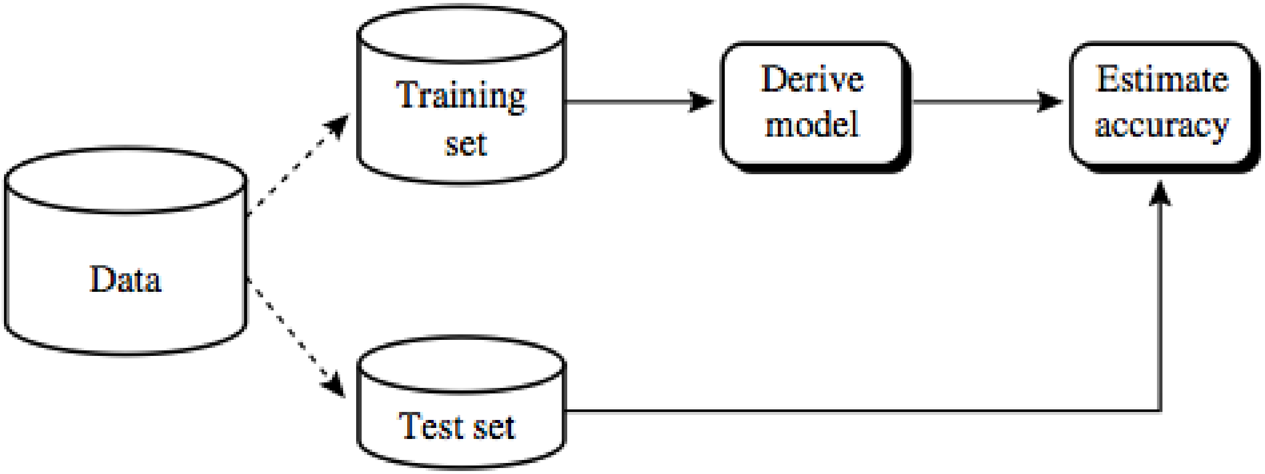
Definition: The **sample error** (denoted $error_S(h)$) of hypothesis h with respect to target function f and data sample S is

$$error_S(h) \equiv \frac{1}{n} \sum_{x \in S} \delta(f(x), h(x))$$

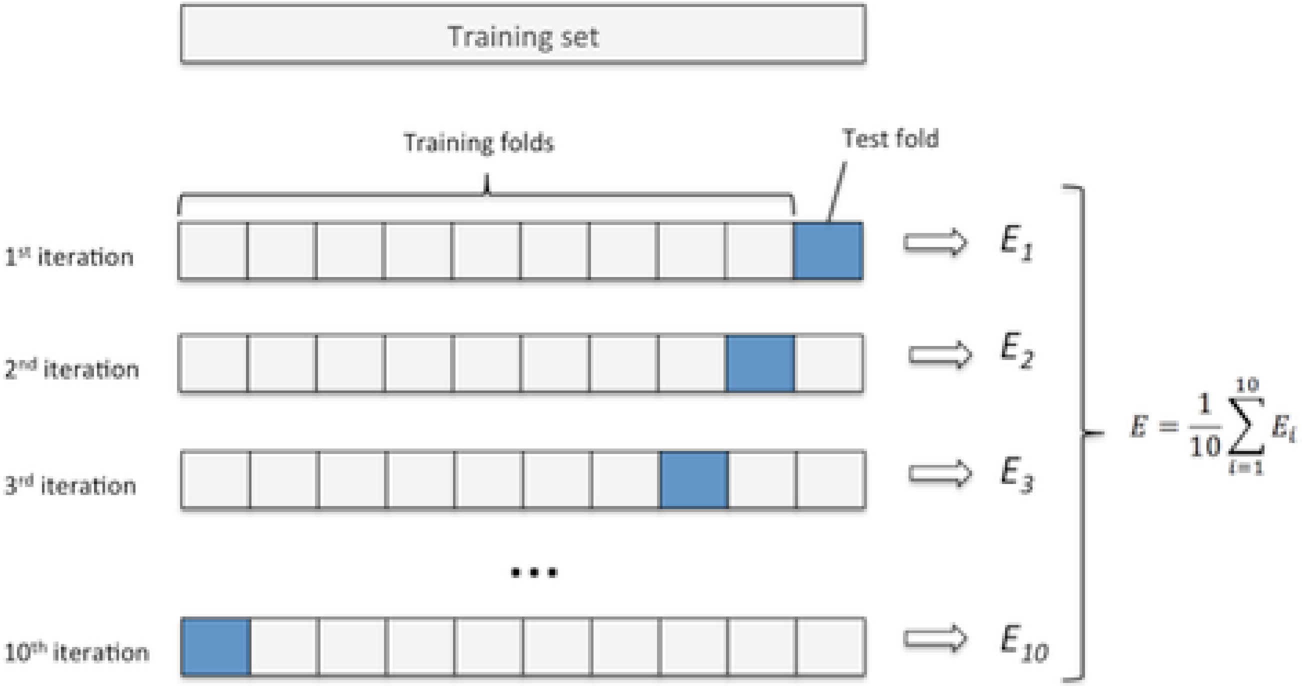
Where n is the number of examples in S, and the quantity $\delta(f(x), h(x))$ is 1 if $f(x) \neq h(x)$, and 0 otherwise.

The **true error** is unknown (and will remain so forever…).
On which sample should I compute the **sample error**?

Training *vs* Test Set

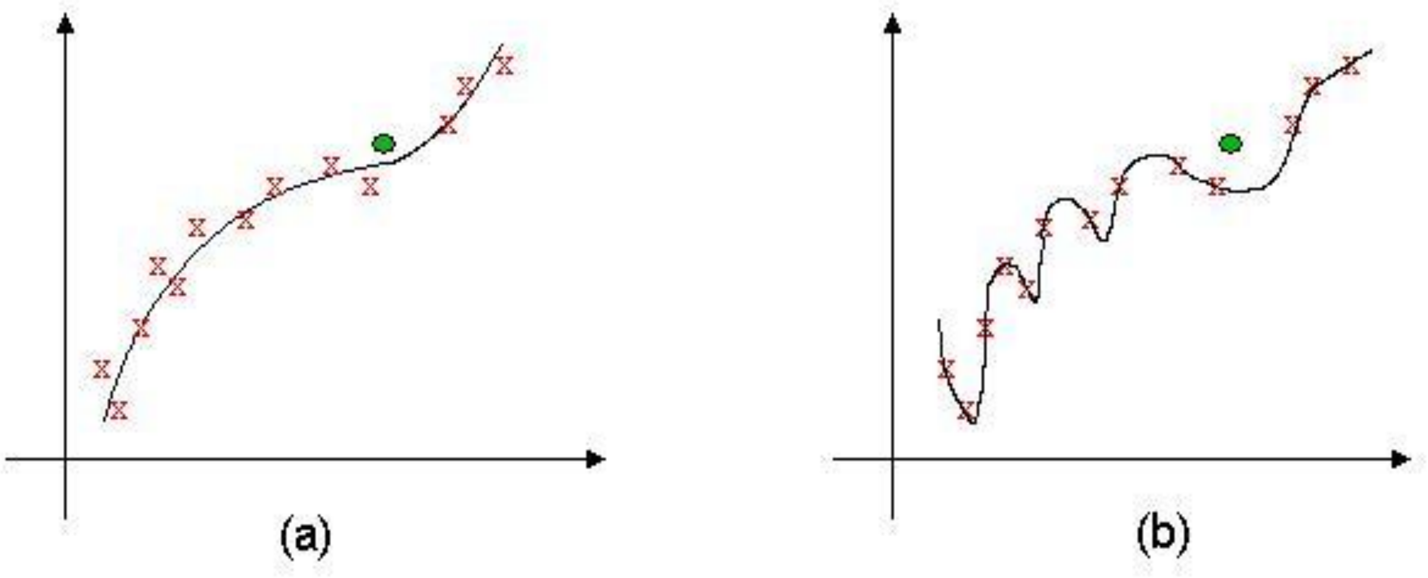


Cross-validation



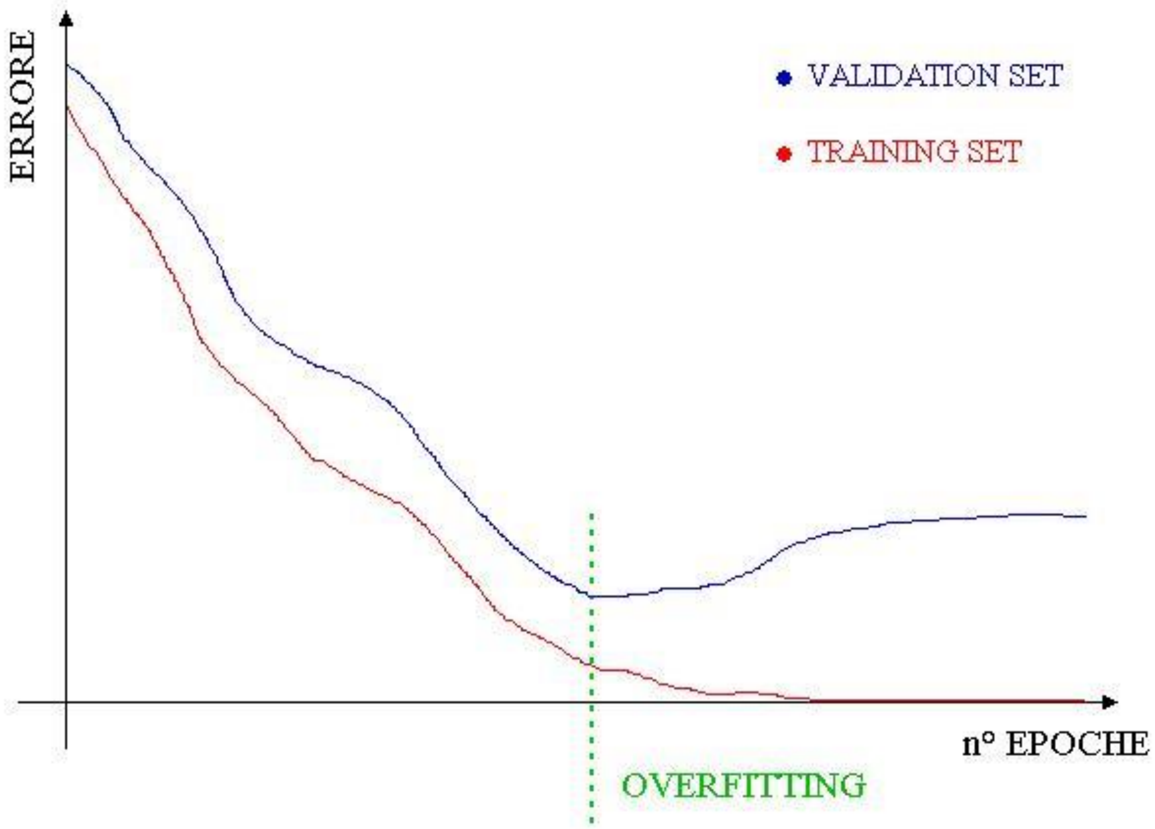
Leave-one-out: using as many test folds as there are examples (size of test fold = 1)

Overfitting



(a) A good fit to noisy data.(b) Overfitting of the same data: the fit is perfect on the “training set” (x’ s), but is likely to be poor on “test set” represented by the circle.

Early Stopping



Size Matters

- The size (i.e. the number of hidden units) of an artificial neural network affects both its functional capabilities and its generalization performance

- Small networks could not be able to realize the desired input / output mapping

- Large networks lead to poor generalization performance

The Pruning Approach

Train an over-dimensioned net and then remove redundant nodes and / or connections:

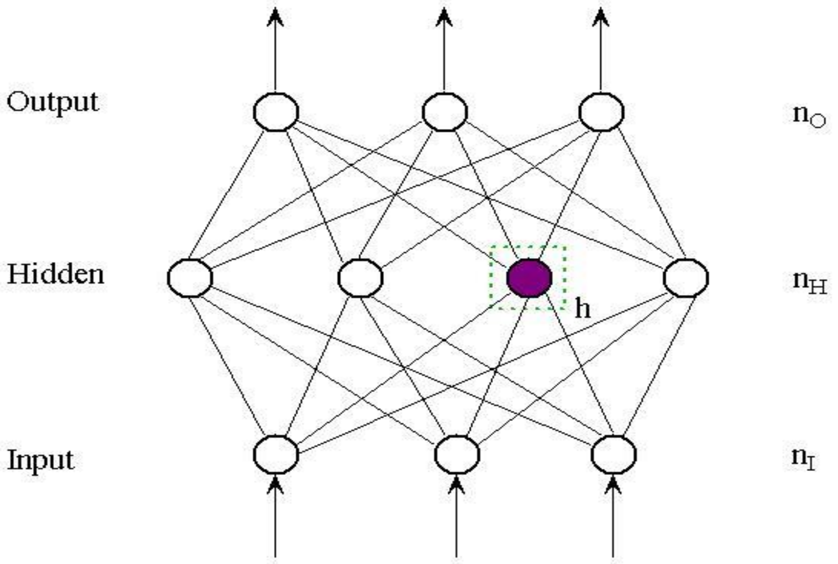
- Sietsma & Dow (1988, 1991)
- Mozer & Smolensky (1989)
- Burkitt (1991)

Adavantages:

- arbitrarily complex decision regions
- faster training
- independence of the training algorithm

An Iterative Pruning Algorithm

Consider (for simplicity) a net with one hidden layer:



Suppose that unit *h* is to be removed:

IDEA: Remove unit *h* (and its in/out connections) and adjust the remaining weights so that the I/O behavior is the same

G. Castellano, A. M. Fanelli, and M. Pelillo, An iterative pruning algorithm for feedforward neural networks, *IEEE Transactions on Neural Networks* 8(3):519-531, 1997.

An Iterative Pruning Algorithm

This is equivalent to solving the system:

$$\underbrace{\overset{n_h}{\underset{j=1}{\text{å}}} w_{ij}\, y_j^{(m)}}_{\textit{before}} = \underbrace{\overset{n_h}{\underset{\substack{j=1\\ j^1\, h}}{\text{å}}} \left(w_{ij} + d_{ij}\right) y_j^{(m)}}_{\textit{after}} \qquad i = 1 \square\ n_O\,,\ m = 1 \square\ P$$

which is equivalent to the following linear system (in the unknown δ’s):

$$\sum_{j \neq h} \delta_{ij}\, y_j^{(\mu)} = w_{ih}\, y_h^{(\mu)} \qquad i = 1 \square\ n_O\,,\ m = 1 \square\ P$$

An Iterative Pruning Algorithm

In a more compact notation:

$$Ax = b$$

where A Î Â$^{Pn_o \text{´} n_o(n_h - 1)}$

But solution does not always exists.

Least-square solution :

$$\min_{x} \| Ax - b \|$$

Detecting Excessive Units

- Residual-reducing methods for LLSPs start with an initial solution x_0 and produces a sequences of points $\{x_k\}$ so that the residuals

  $$\| Ax_k - b \| = r_k$$

  decrease: $r_k \leq r_{k-1}$

- Starting point: $x_0 = 0 \quad \left(\Rightarrow r_0 = \| b \| \right)$

- Excessive units can be detected so that $\| b \|$ is minimum

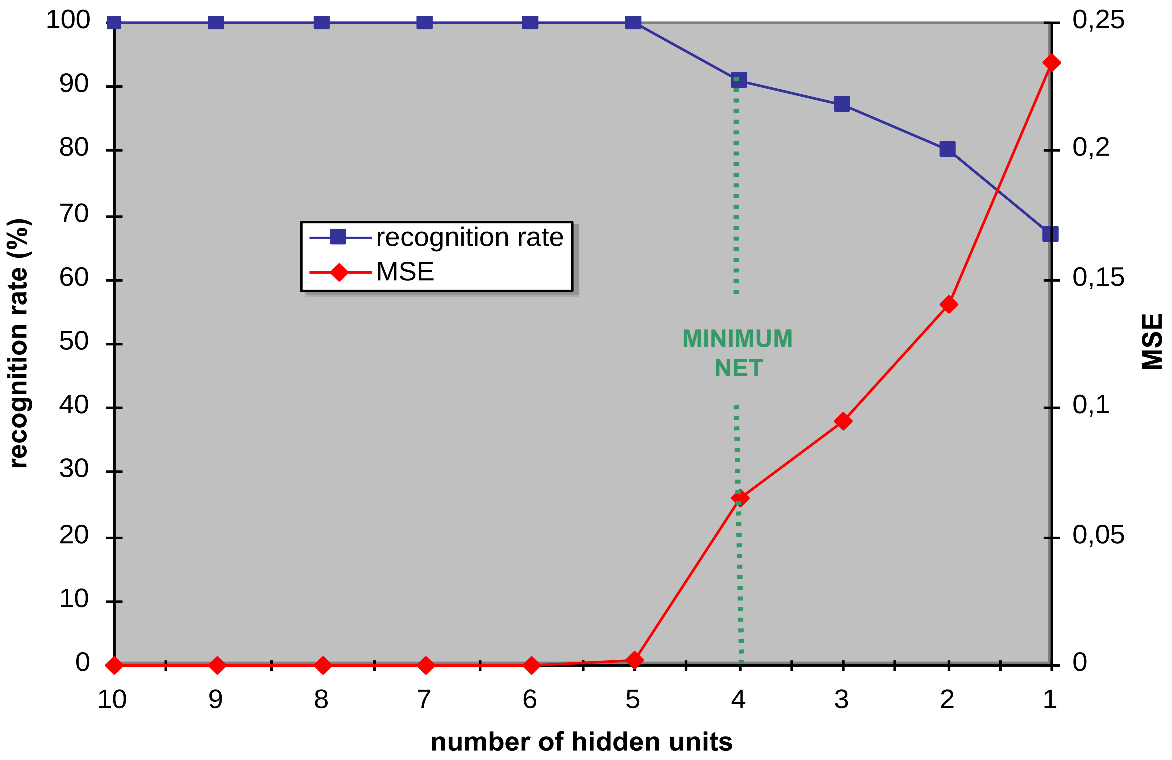
The Pruning Algorithm

```
1)  Start with an over-sized trained network

2)  Repeat

        2.1) find the hidden unit h for which $\|b\|$ is minimum
        2.2) solve the corresponding system
        2.3) remove unit h

    Until  Perf(pruned) – Perf(original) < epsilon

3)  Reject the last reduced network
```

Example: 4-bit parity

Ten initial 4-10-1 networks

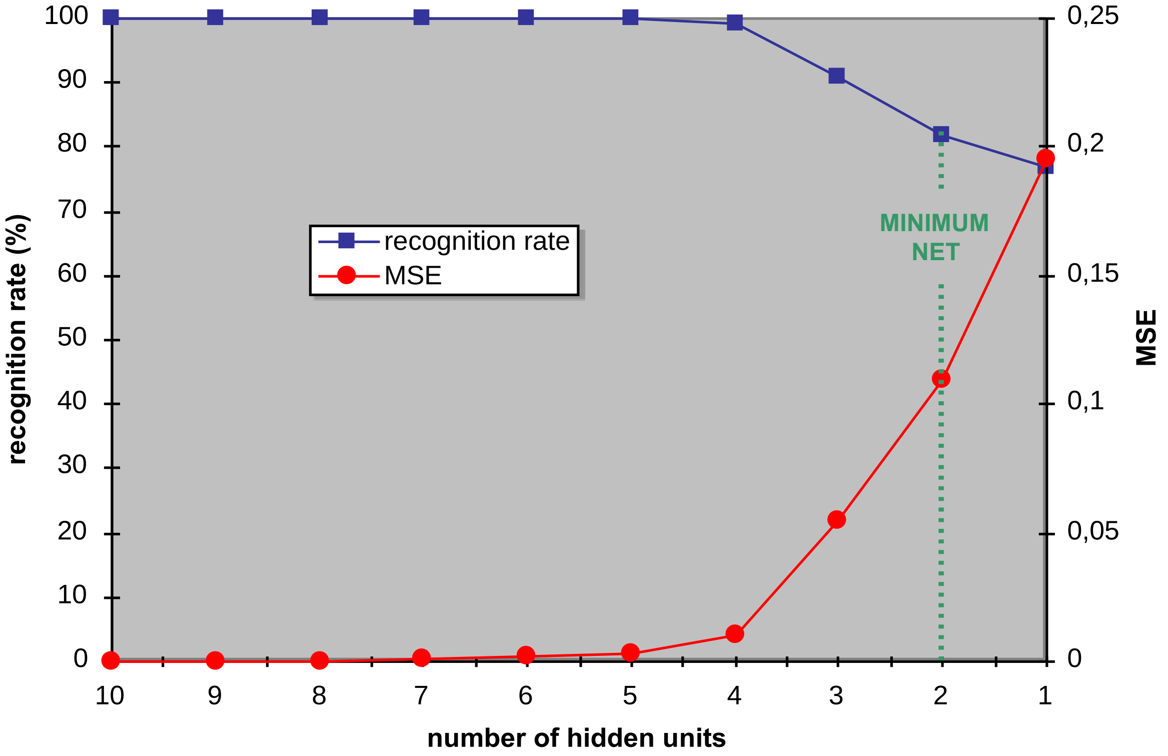
Pruned nets { nine 4-5-1, one 4-4-1 } 5 hidden nodes (average)



Example: 4-bit simmetry

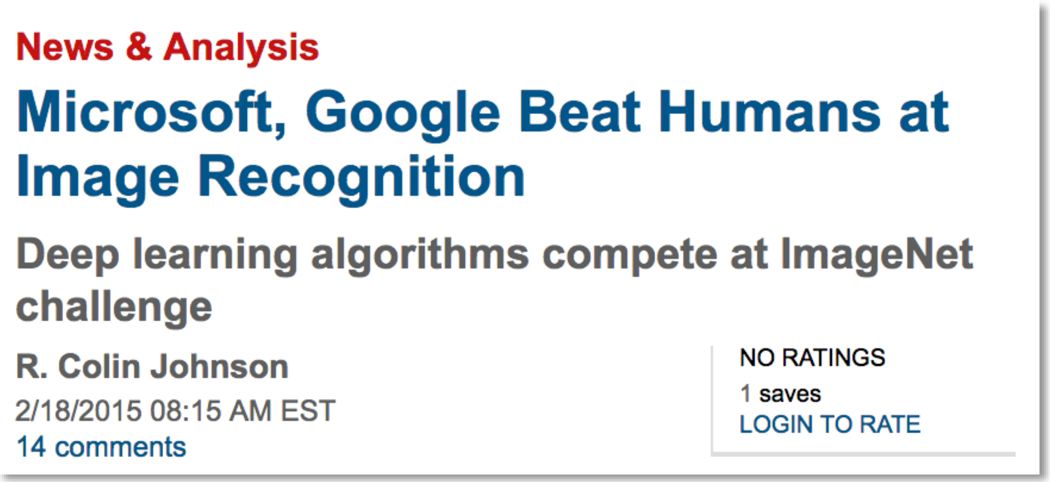
Ten initial 4-10-1 networks

Pruned nets → 4.6 hidden nodes (average)



Deep Neural Networks

The Age of "Deep Learning"



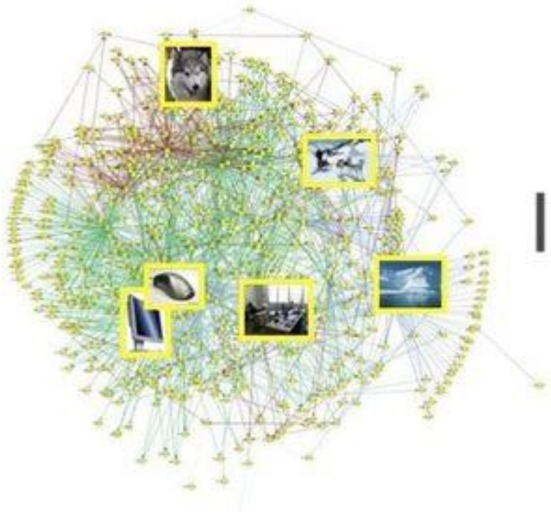
News & Analysis

Microsoft, Google Beat Humans at Image Recognition

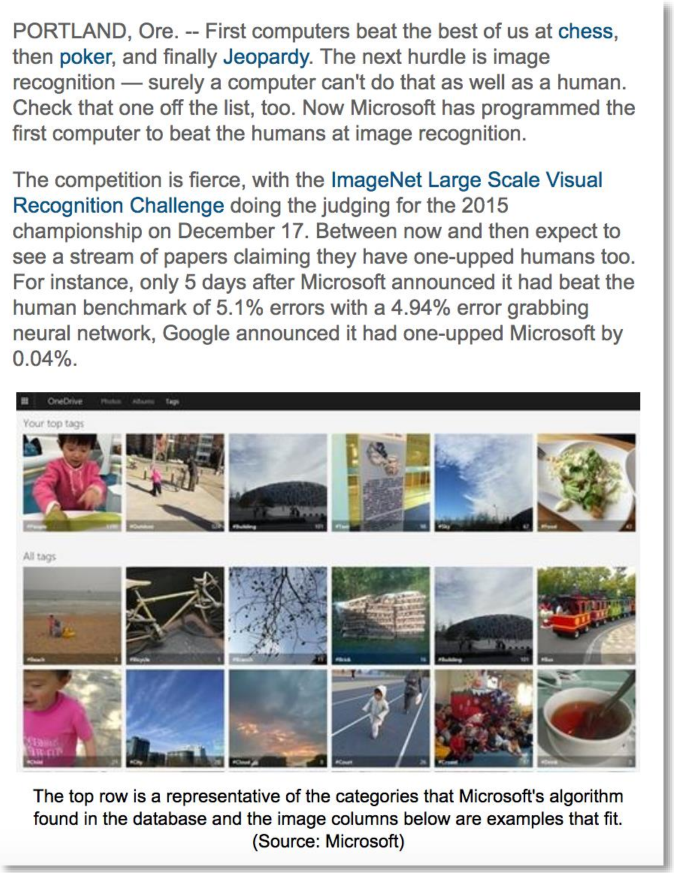
Deep learning algorithms compete at ImageNet challenge

R. Colin Johnson
2/18/2015 08:15 AM EST
14 comments

NO RATINGS
1 saves
LOGIN TO RATE



IMAGENET



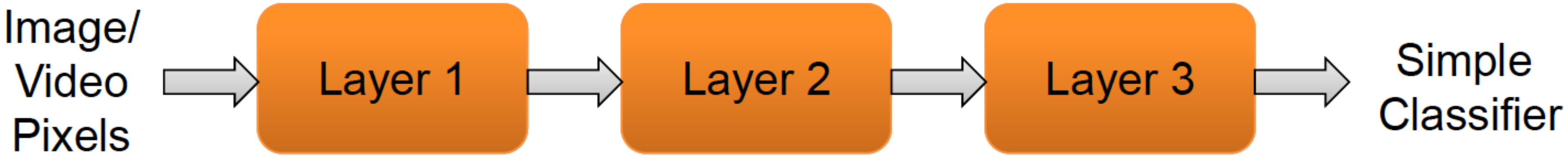
PORTLAND, Ore. -- First computers beat the best of us at chess, then poker, and finally Jeopardy. The next hurdle is image recognition — surely a computer can't do that as well as a human. Check that one off the list, too. Now Microsoft has programmed the first computer to beat the humans at image recognition.

The competition is fierce, with the ImageNet Large Scale Visual Recognition Challenge doing the judging for the 2015 championship on December 17. Between now and then expect to see a stream of papers claiming they have one-upped humans too. For instance, only 5 days after Microsoft announced it had beat the human benchmark of 5.1% errors with a 4.94% error grabbing neural network, Google announced it had one-upped Microsoft by 0.04%.

The top row is a representative of the categories that Microsoft's algorithm found in the database and the image columns below are examples that fit. (Source: Microsoft)

The Deep Learning "Philosophy"

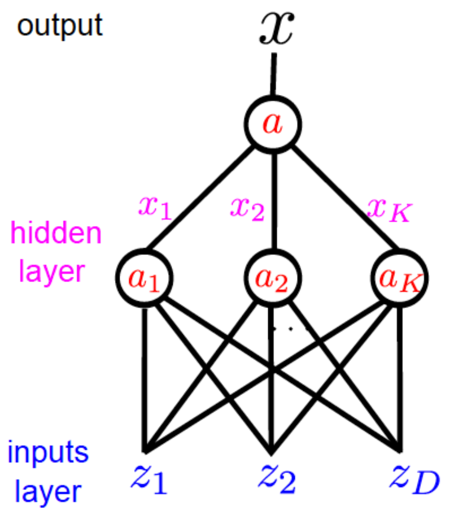
- Learn a feature hierarchy all the way from pixels to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly



Shallow *vs* Deep Networks

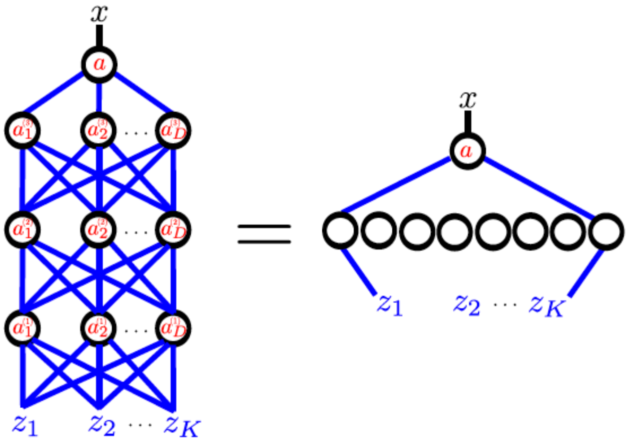
Shallow architectures are inefficient at representing deep functions

single layer neural network implements: $x = f_\theta(\mathbf{z})$



networks we met last lecture
with large enough single hidden layer
can implement **any** function
'universal approximator'

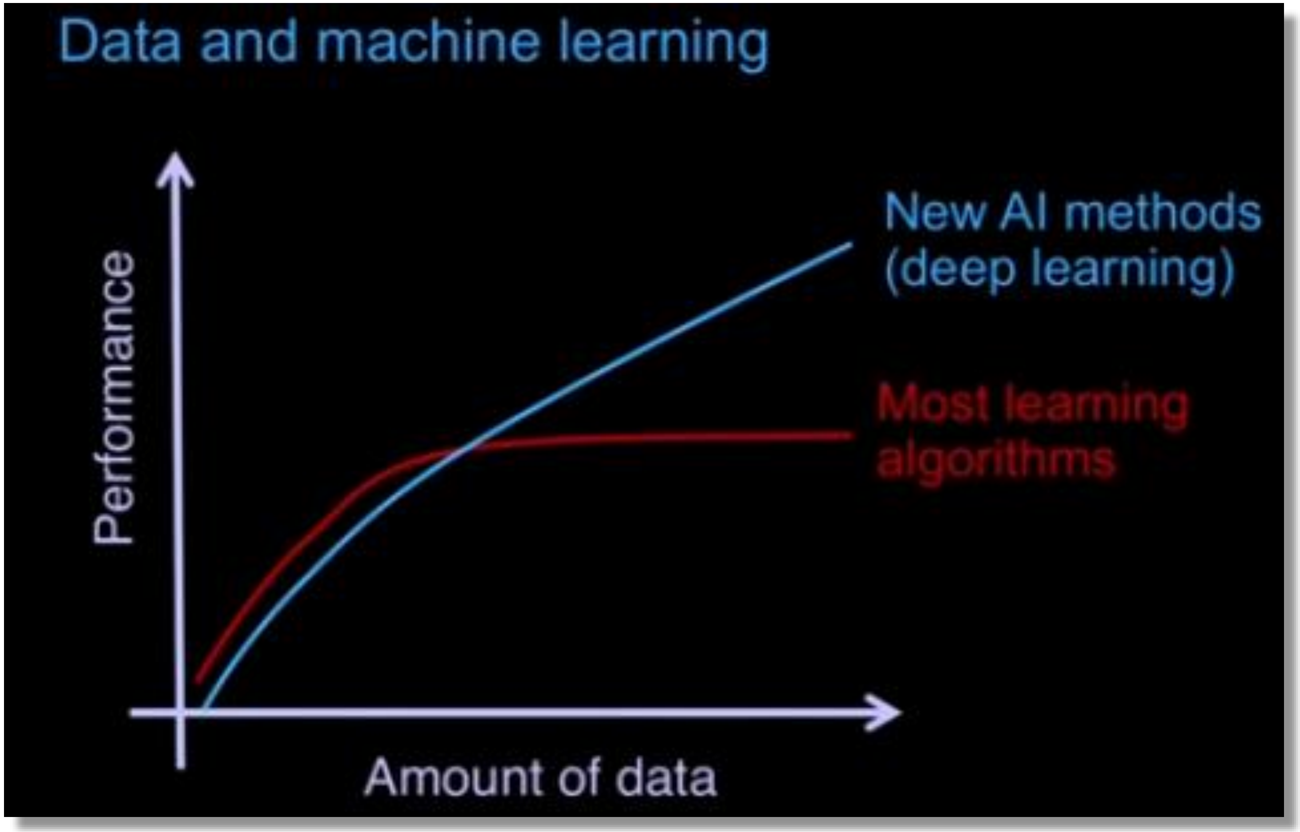
shallow networks can be computationally inefficient



however, if the function is 'deep'
a very large hidden layer may
be required

From. R. E. Turner

Performance Improves with More Data



Old Idea… Why Now?

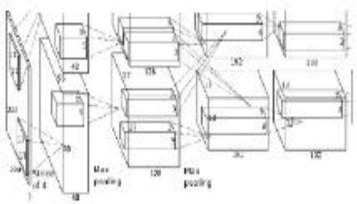
1. We have more data - from Lena to ImageNet.

1. We have more computing power, GPUs are really good at this.

1. Last but not least, we have new ideas

Big Data: ImageNet

+



Deep Convolutional Neural Network

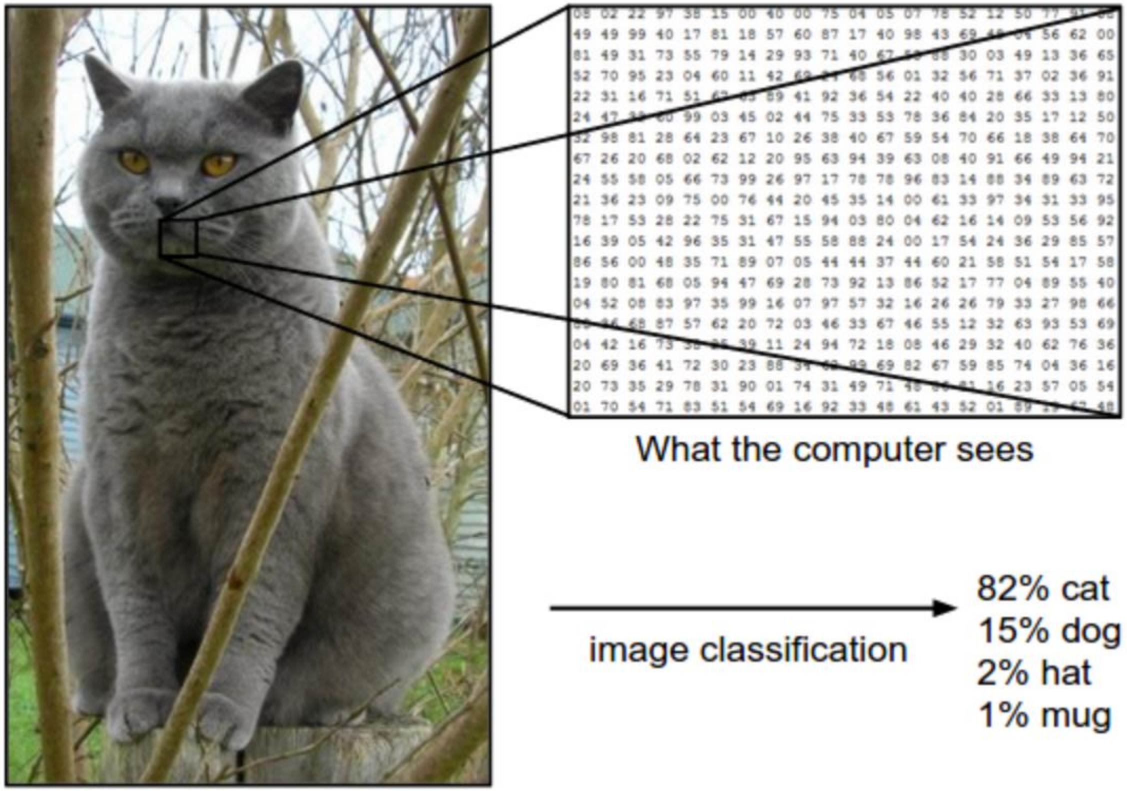
+

Backprop on GPU

=

Learned Weights

Image Classification



08	02	22	97	38	15	00	40	00	75	04	05	07	78	52	12	50	77	91	08
49	49	99	40	17	81	18	57	60	87	17	40	98	43	69	48	04	56	62	00
81	49	31	73	55	79	14	29	93	71	40	67	53	88	30	03	49	13	36	65
52	70	95	23	04	60	11	42	69	24	68	56	01	32	56	71	37	02	36	91
22	31	16	71	51	67	63	89	41	92	36	54	22	40	40	28	66	33	13	80
24	47	32	60	99	03	45	02	44	75	33	53	78	36	84	20	35	17	12	50
32	98	81	28	64	23	67	10	26	38	40	67	59	54	70	66	18	38	64	70
67	26	20	68	02	62	12	20	95	63	94	39	63	08	40	91	66	49	94	21
24	55	58	05	66	73	99	26	97	17	78	78	96	83	14	88	34	89	63	72
21	36	23	09	75	00	76	44	20	45	35	14	00	61	33	97	34	31	33	95
78	17	53	28	22	75	31	67	15	94	03	80	04	62	16	14	09	53	56	92
16	39	05	42	96	35	31	47	55	58	88	24	00	17	54	24	36	29	85	57
86	56	00	48	35	71	89	07	05	44	44	37	44	60	21	58	51	54	17	58
19	80	81	68	05	94	47	69	28	73	92	13	86	52	17	77	04	89	55	40
04	52	08	83	97	35	99	16	07	97	57	32	16	26	26	79	33	27	98	66
88	36	68	87	57	62	20	72	03	46	33	67	46	55	12	32	63	93	53	69
04	42	16	73	38	25	39	11	24	94	72	18	08	46	29	32	40	62	76	36
20	69	36	41	72	30	23	88	34	62	99	69	82	67	59	85	74	04	36	16
20	73	35	29	78	31	90	01	74	31	49	71	48	86	81	16	23	57	05	54
01	70	54	71	83	51	54	69	16	92	33	48	61	43	52	01	89	19	67	48

What the computer sees

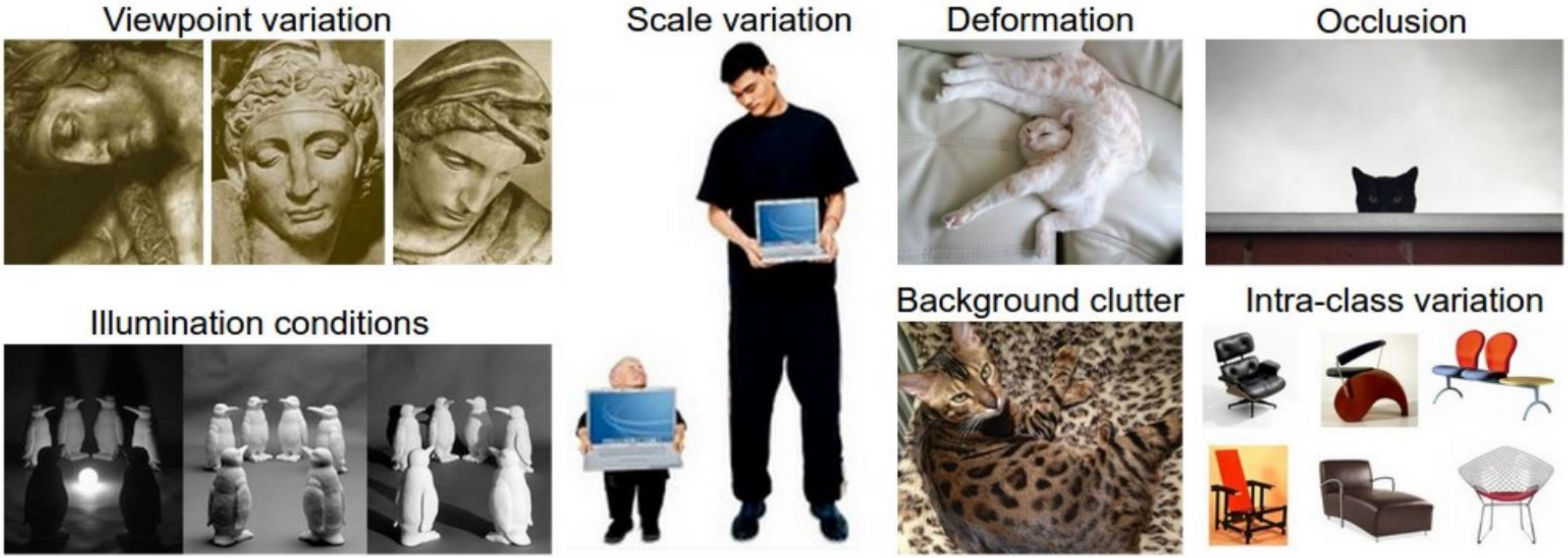
image classification

82% cat
15% dog
2% hat
1% mug

Predict a single label (or a distribution over labels as shown here to indicate our confidence) for a given image. Images are 3-dimensional arrays of integers from 0 to 255, of size Width x Height x 3. The 3 represents the three color channels Red, Green, Blue.

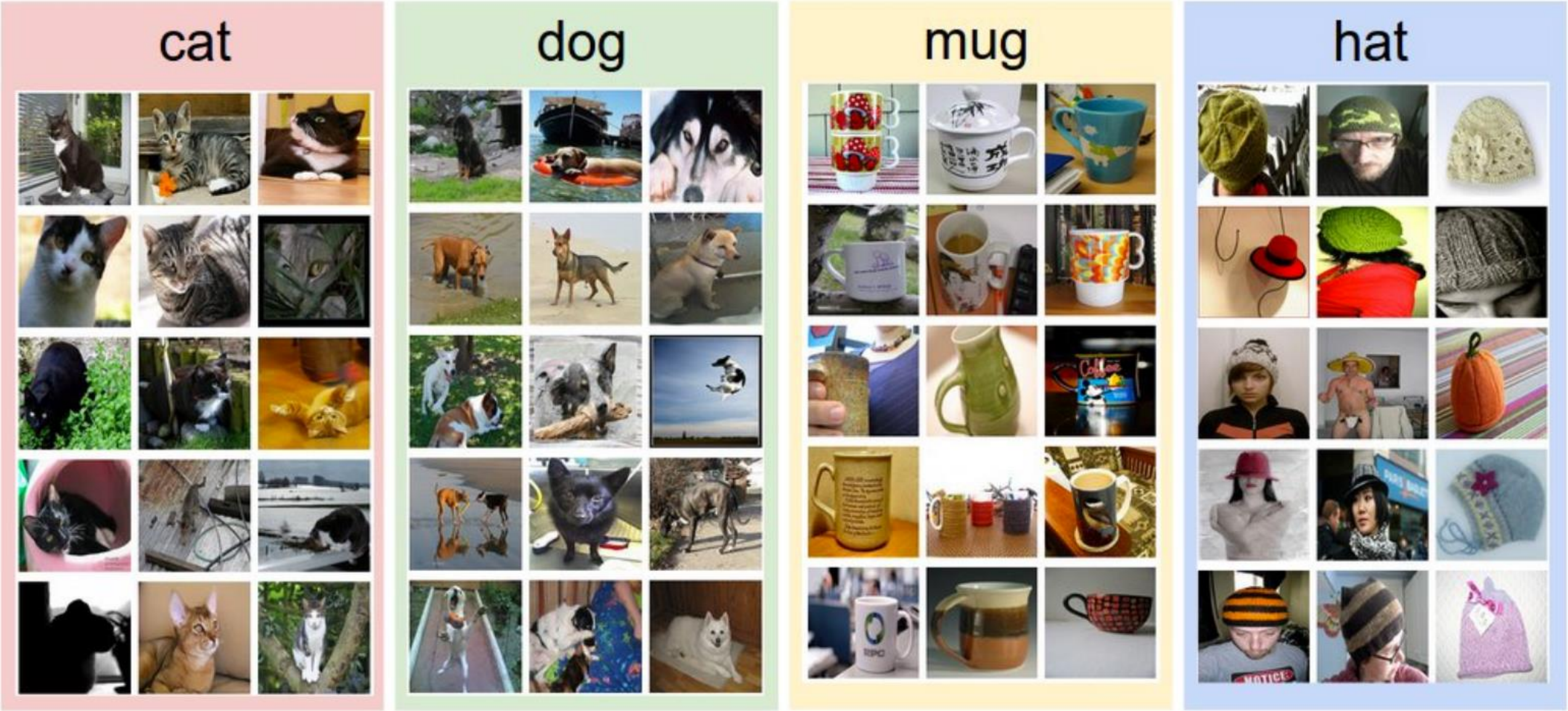
From: A. Karpathy

Challenges



From: A. Karpathy

The Data-Driven Approach



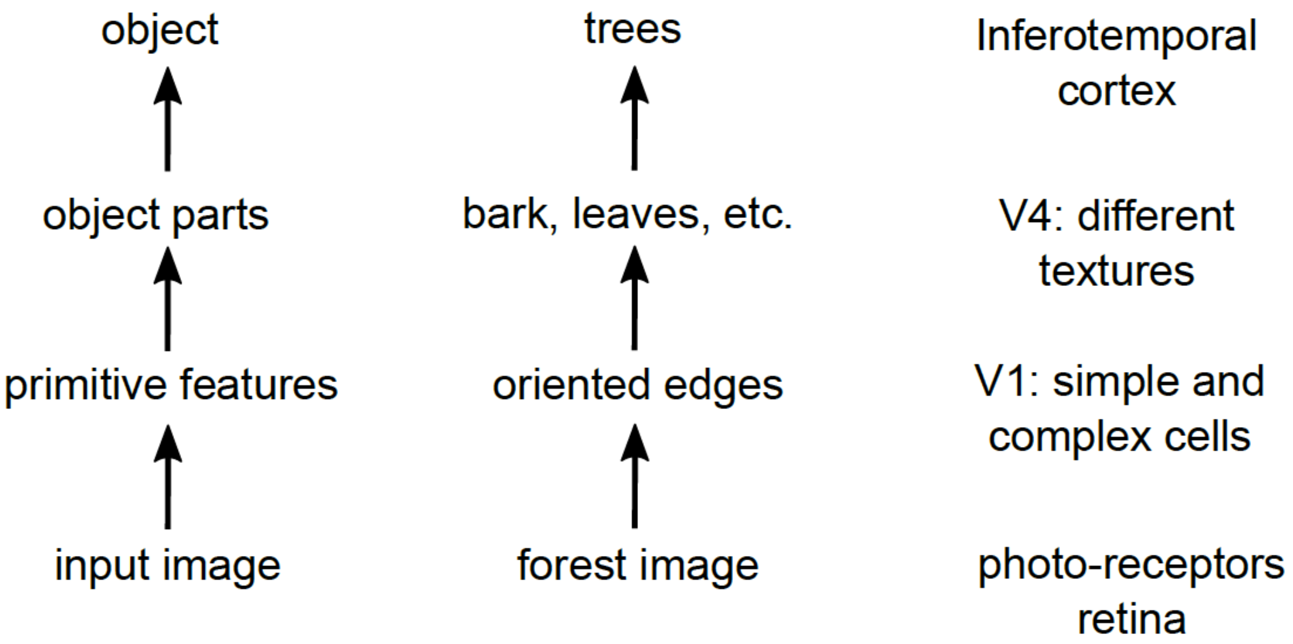
An example training set for four visual categories.

In practice we may have thousands of categories and hundreds of thousands of images for each category.

From: A. Karpathy

Inspiration from Biology

Biological vision is hierachically organized



object

object parts

primitive features

input image

trees

bark, leaves, etc.

oriented edges

forest image

Inferotemporal cortex

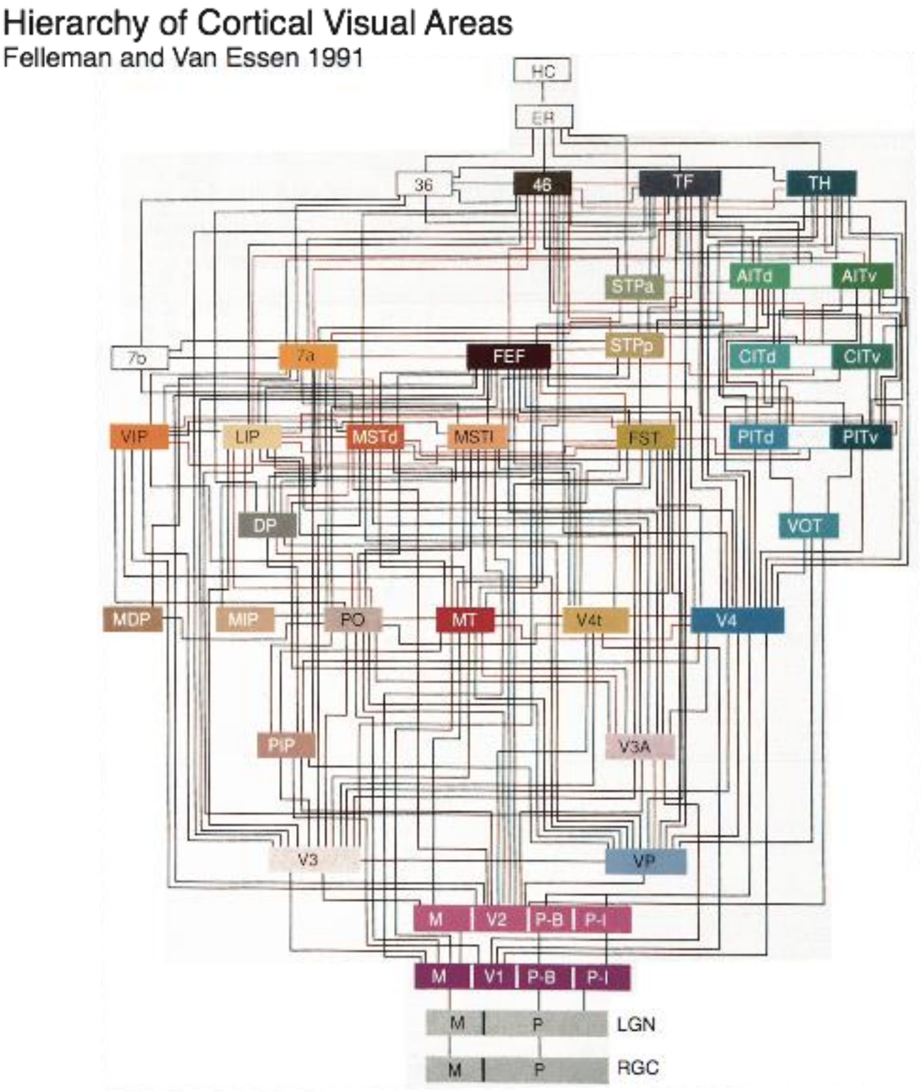
V4: different textures

V1: simple and complex cells

photo-receptors retina

From. R. E. Turner

Hierarchy of Visual Areas



Hierarchy of Cortical Visual Areas

Felleman and Van Essen 1991

From. D. Zoccolan

The Retina

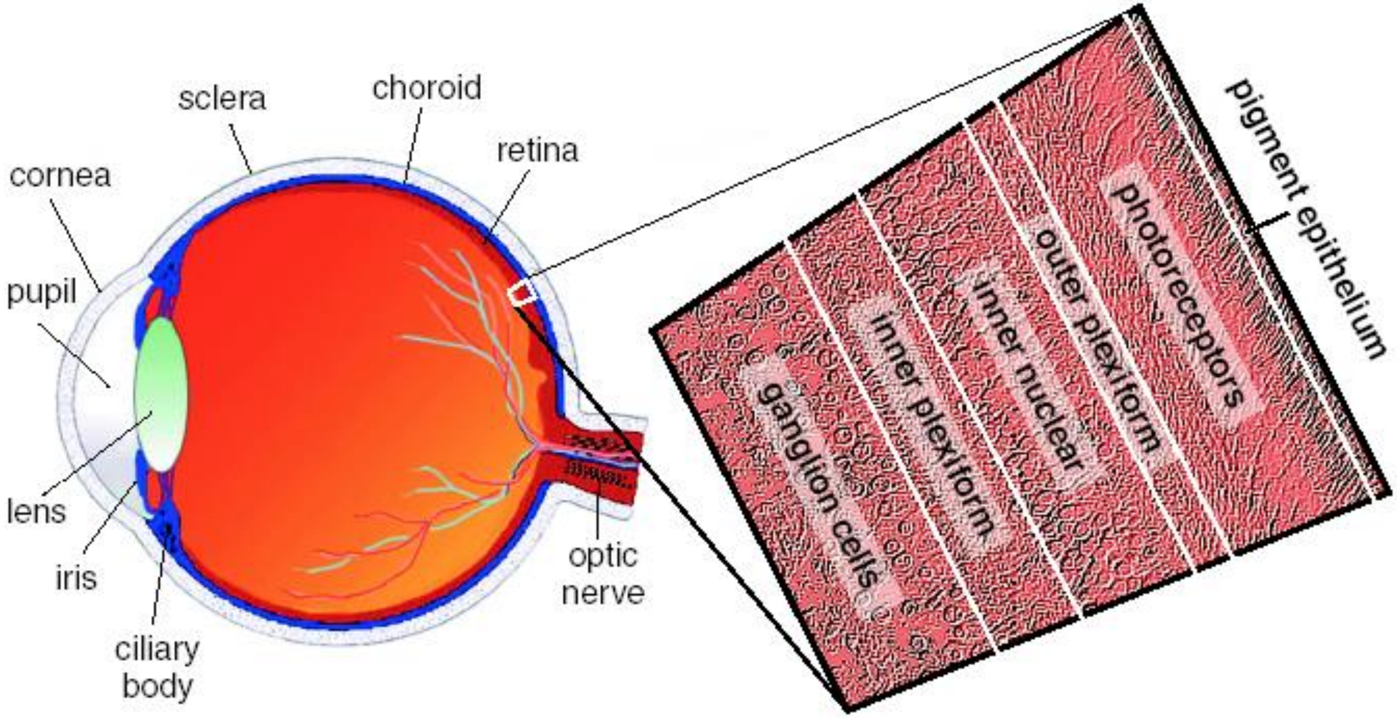


Figure 2. Diagram of a human eye shows its various structures *(left)*. A thin piece of retina is enlarged in a photomicrograph *(right)*, revealing its layers. The photoreceptors lie against a dark row of cells called the pigment epithelium. (Drawing by the author. Except where noted, photographs by Nicolas Cuenca and the author.)

The Retina

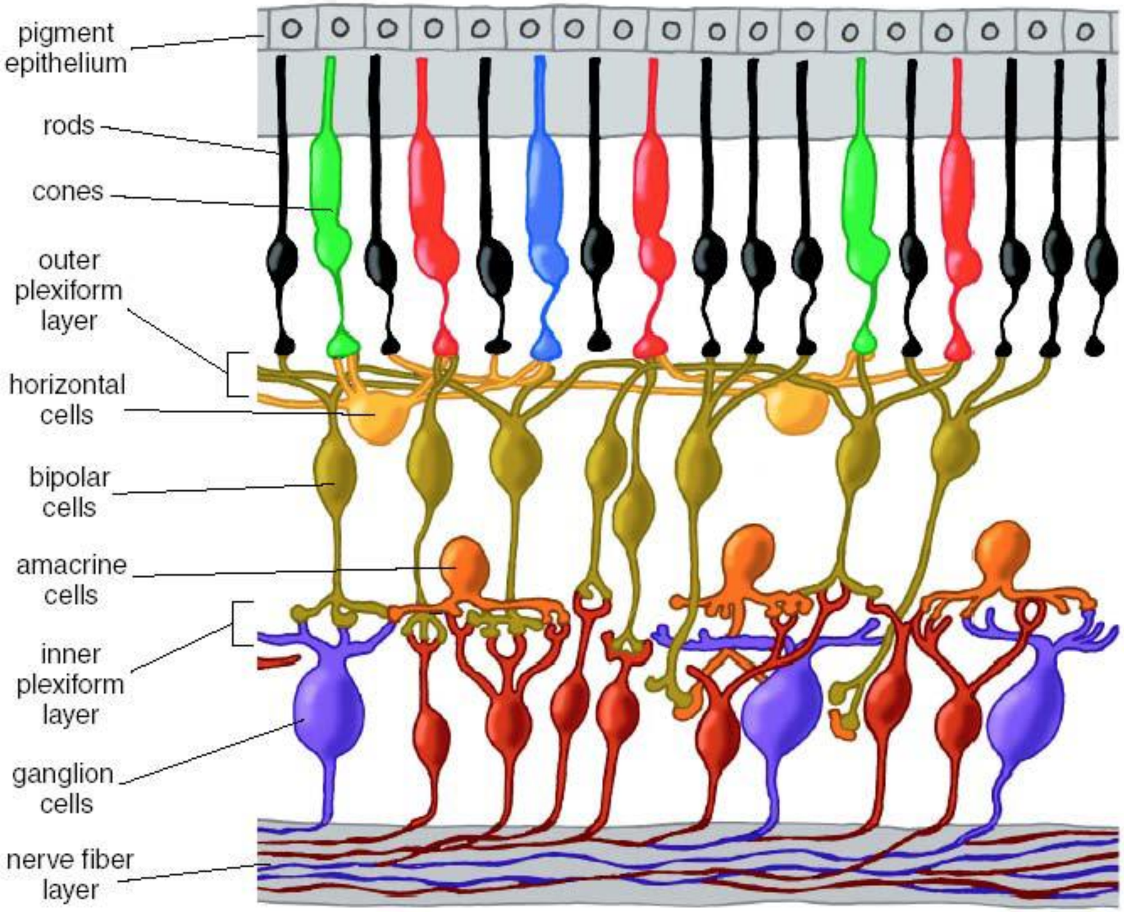
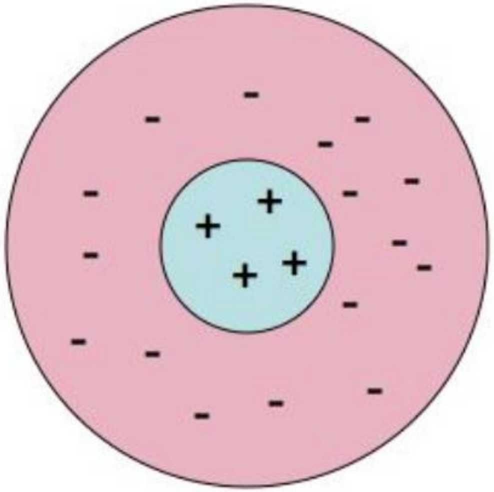


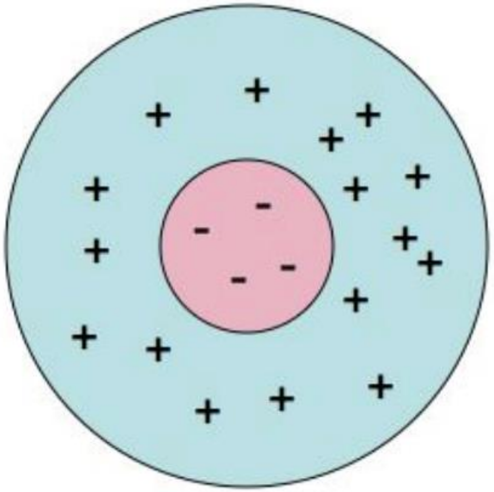
Figure 3. Cells in the retina are arrayed in discrete layers. The photoreceptors are at the top of this rendering, close to the pigment epithelium. The bodies of horizontal cells and bipolar cells compose the inner nuclear layer. Amacrine cells lie close to ganglion cells near the surface of the retina. Axon-to-dendrite neural connections make up the plexiform layers separating rows of cell bodies.

Receptive Fields

“The region of the visual field in which light stimuli evoke responses of a given neuron.”



On-center, Off-surround



Off-center, On-surround

Cellular Recordings

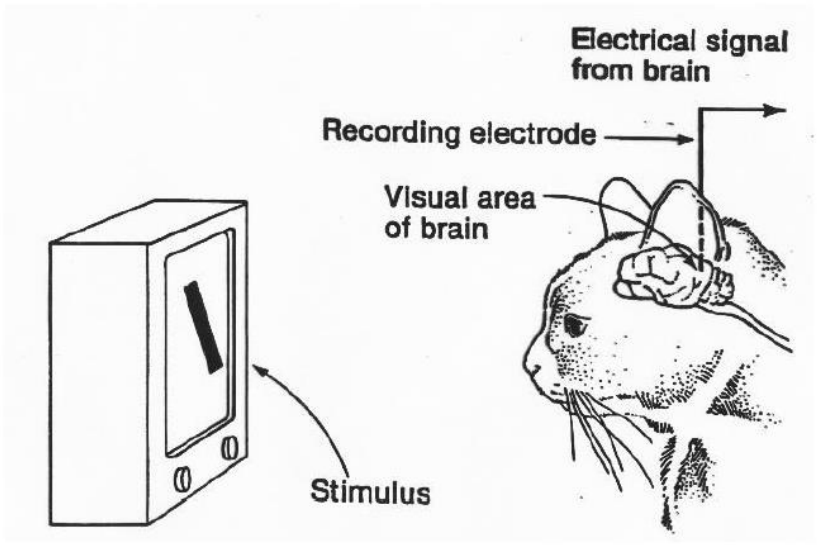
Kuffler, Hubel, Wiesel, …

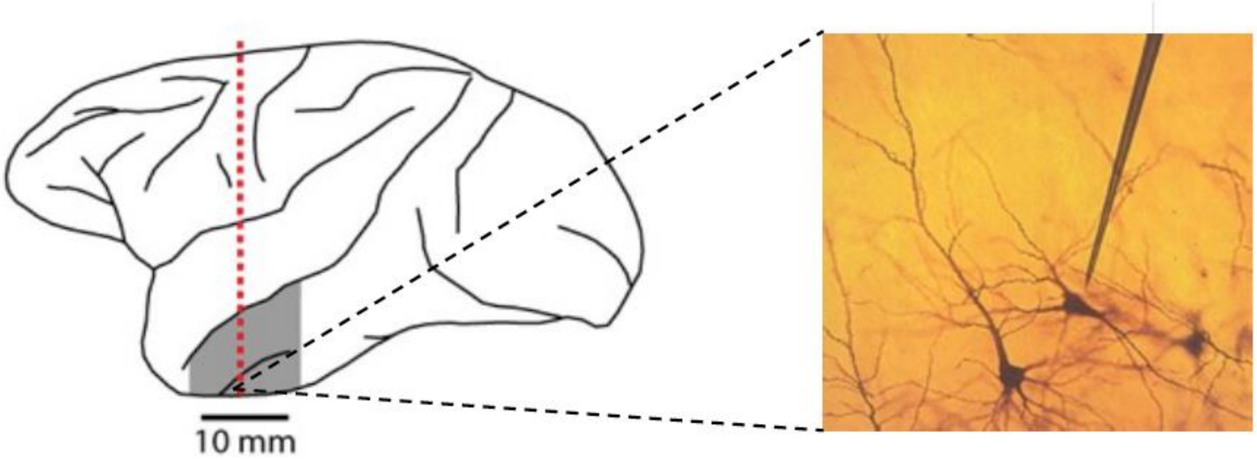
1953: *Discharge patterns and functional organization of mammalian retina*

1959: *Receptive fields of single neurones in the cat's striate cortex*

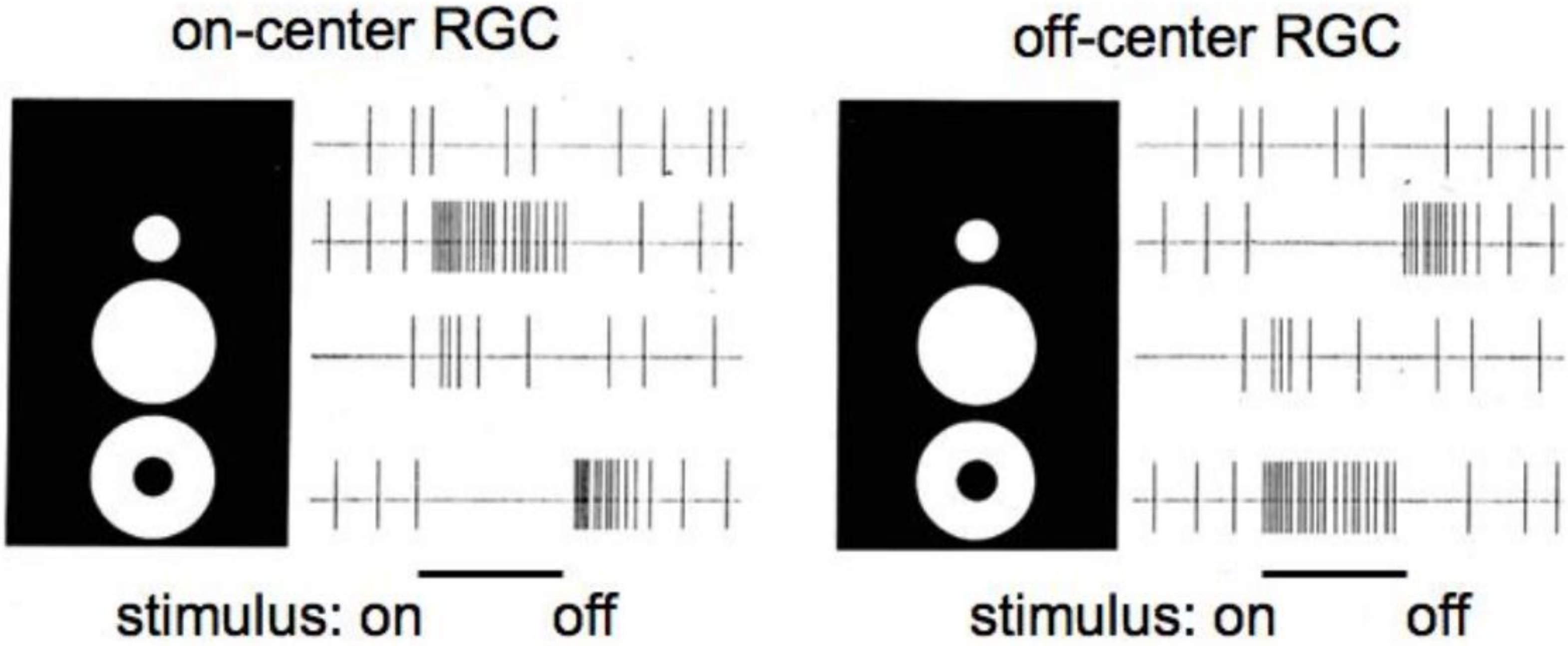
1962: *Receptive fields, binocular interaction and functional architecture in the cat's visual cortex*

1968 ..

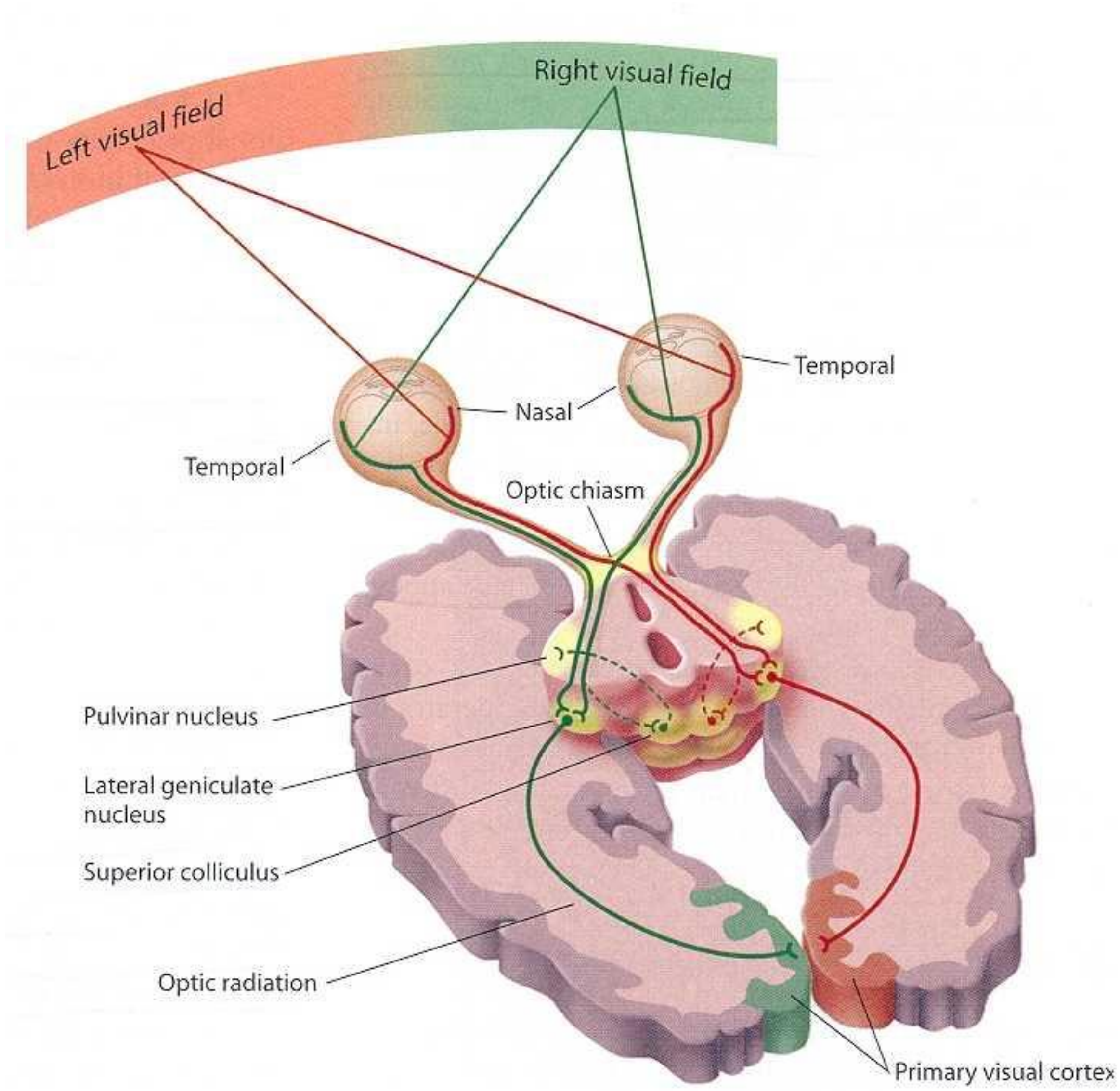




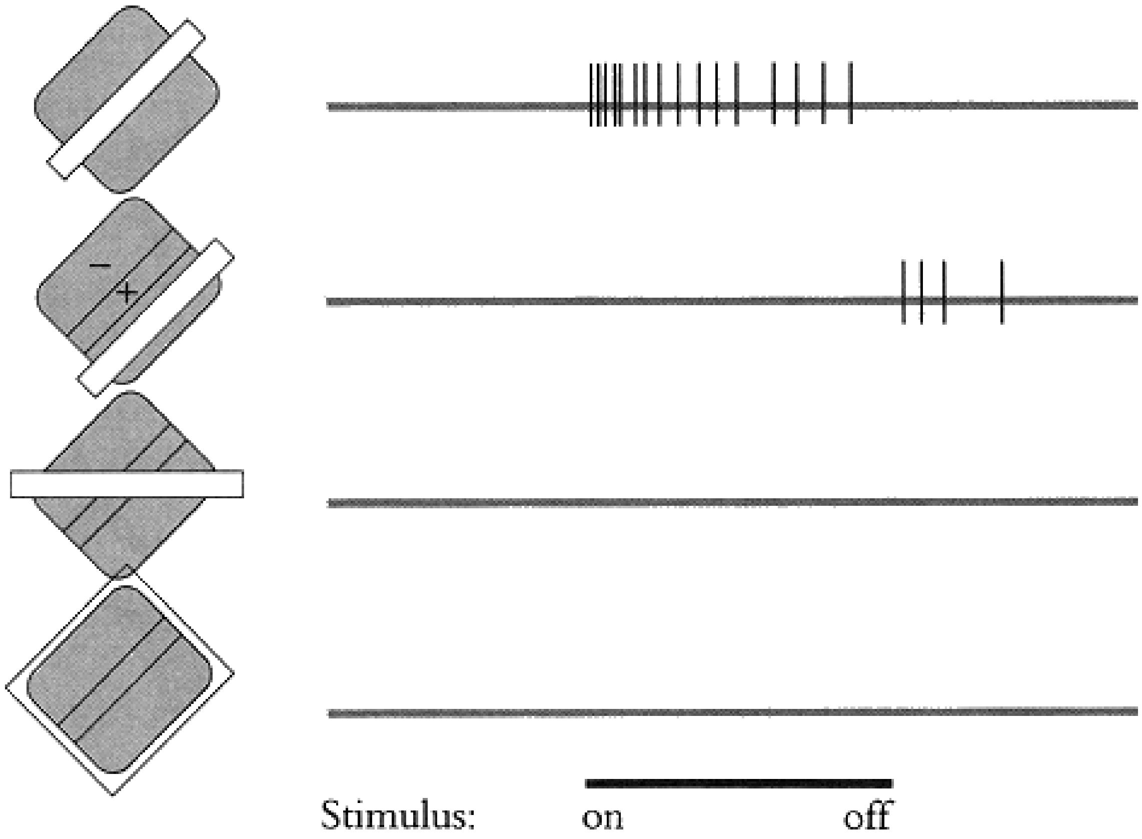
Retinal Ganglion Cell Response



Beyond the Retina

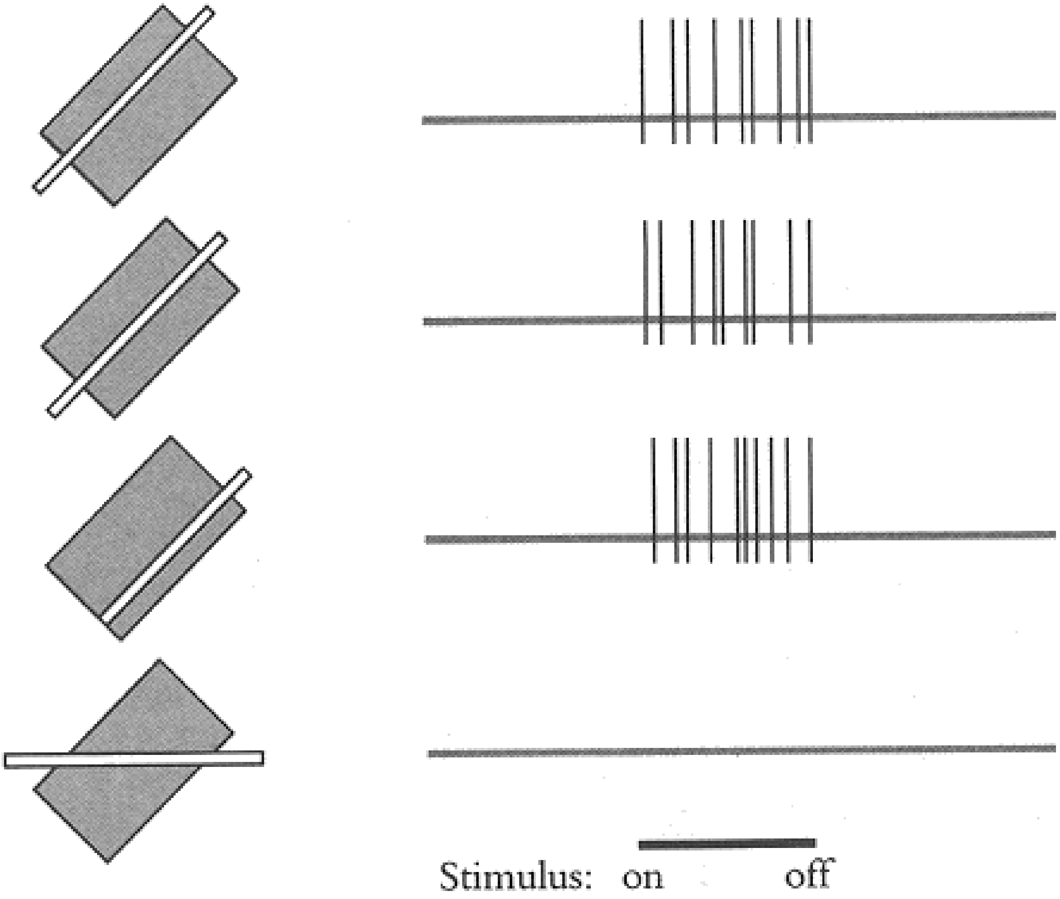


Simple Cells

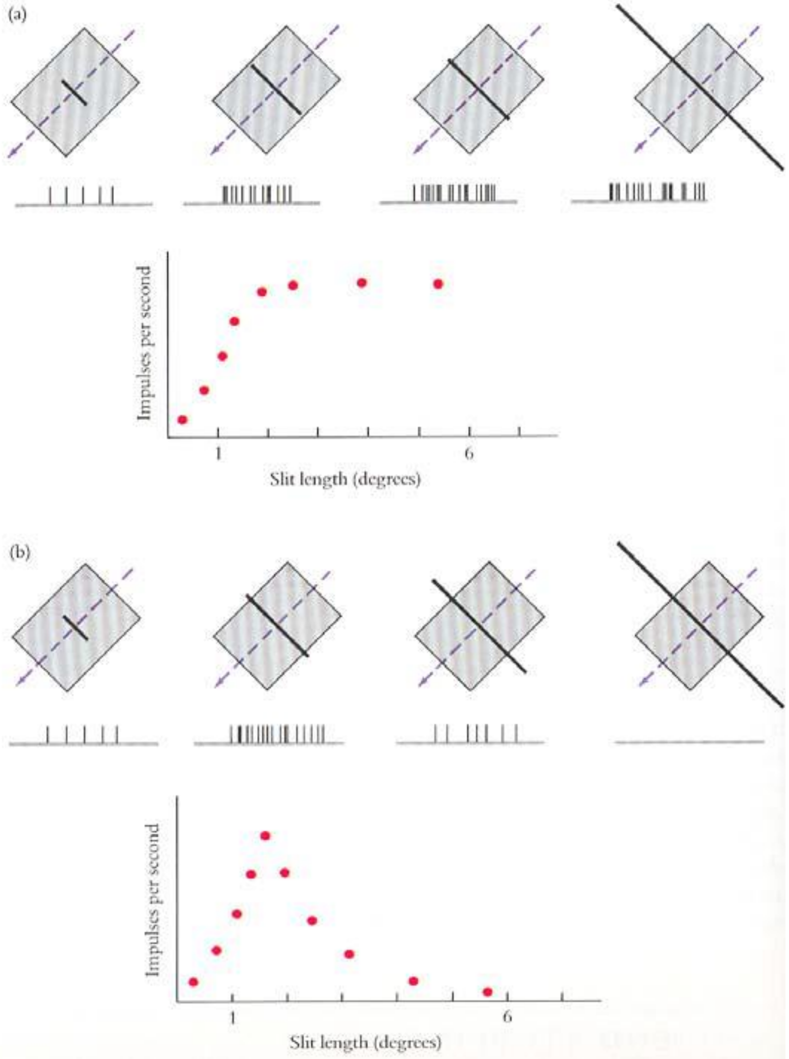


Orientation selectivity: Most V1 neurons are orientation selective meaning that they respond strongly to lines, bars, or edges of a particular orientation (e.g., vertical) but not to the orthogonal orientation (e.g., horizontal).

Complex Cells

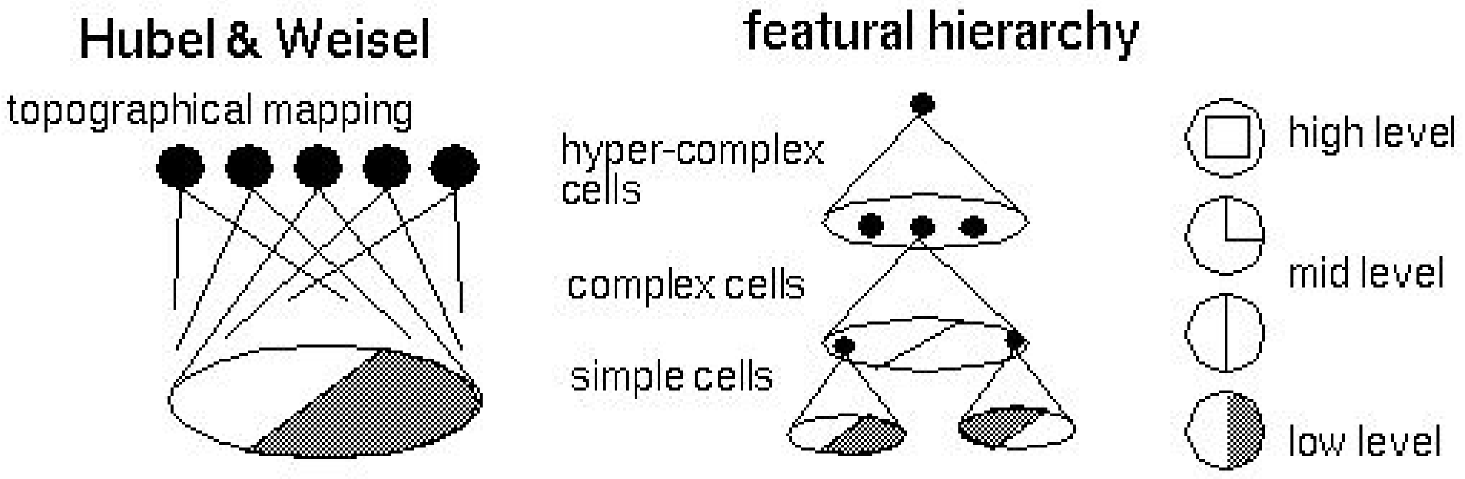


Hypercomplex Cells (end-stopping)

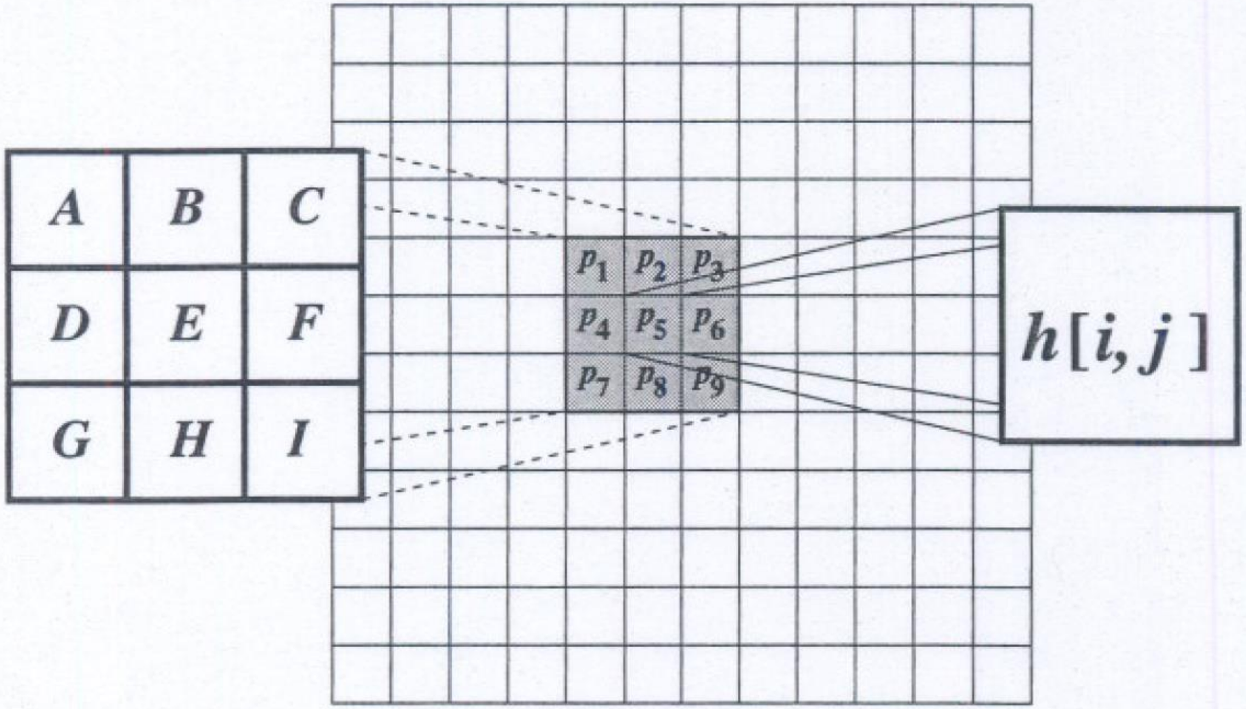


Top: An ordinary complex cell responds to various lengths of a slit of light. The duration of each record is 2 seconds. As indicated by the graph of response versus slit length, for this cell the response increases with length up to about 2 degrees, after which there is no change. *Bottom:* For this end-stopped cell, responses improve up to 2 degrees but then decline, so that a line 6 degrees or longer gives no response.

Take-Home Message:
Visual System as a Hierarchy of Feature Detectors

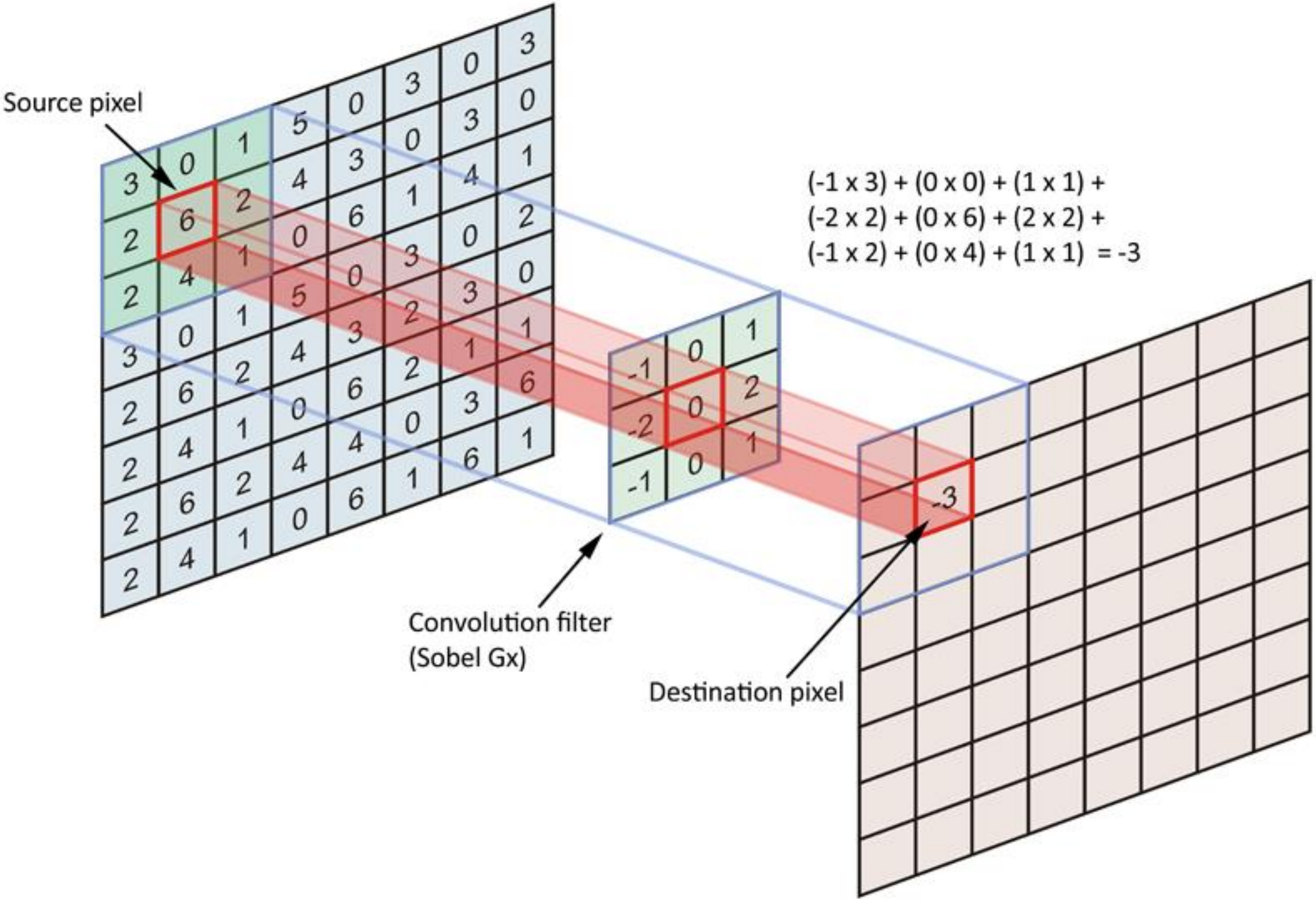


Convolution



$$h[i, j] = A\,p_1 + B\,p_2 + C\,p_3 + D\,p_4 + E\,p_5 + F\,p_6 + G\,p_7 + H\,p_8 + I\,p_9$$

Convolution



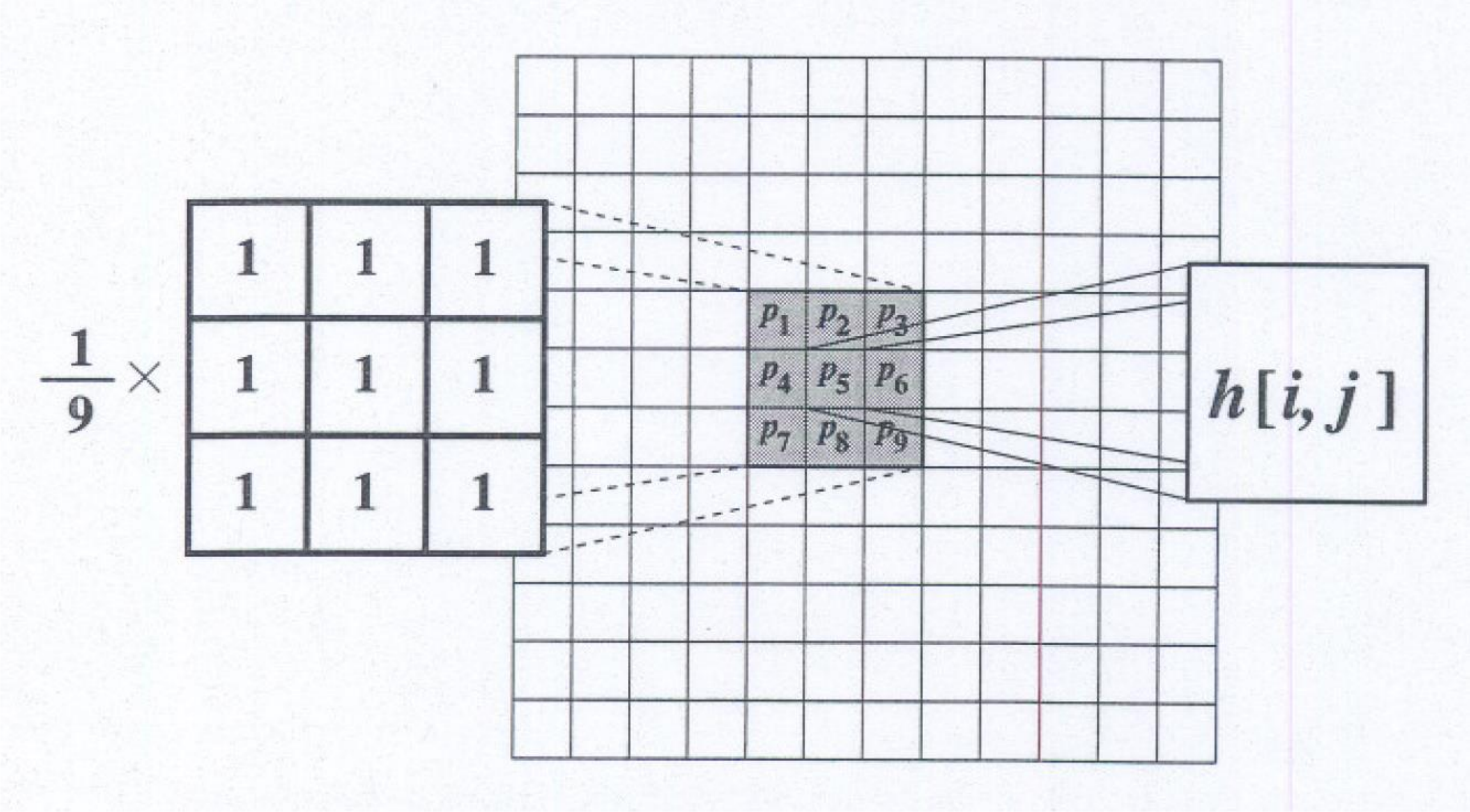
3	0	1	5	0	3	0	3
2	6	2	4	3	0	3	0
2	4	1	0	6	1	4	1
3	0	1	5	0	3	0	2
2	6	2	4	3	2	3	0
2	4	1	0	6	2	1	1
2	6	2	4	4	0	3	6
2	4	1	0	6	1	6	1

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

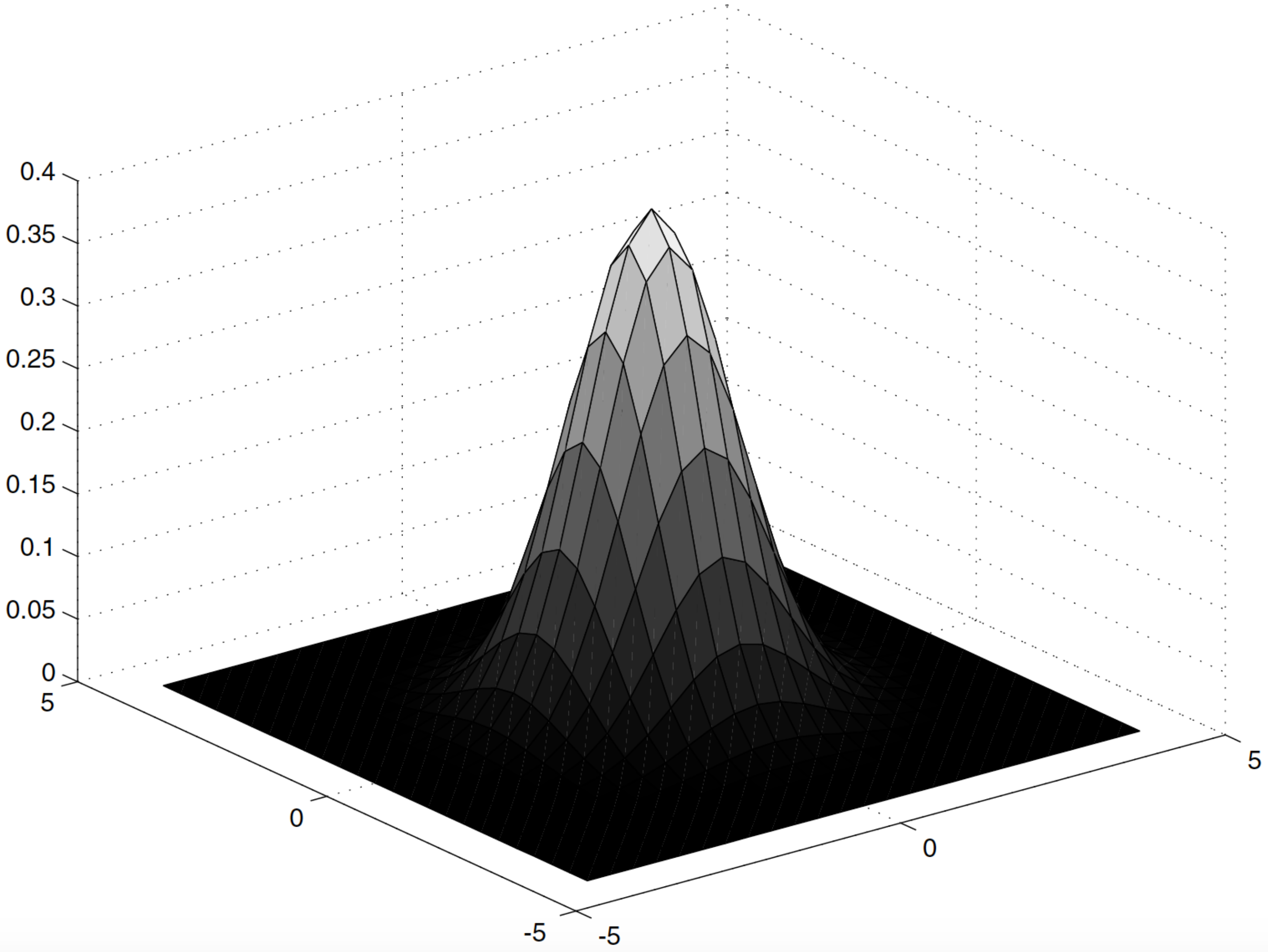
-3

(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3

Mean Filters



Gaussian Filters



Gaussian Filters

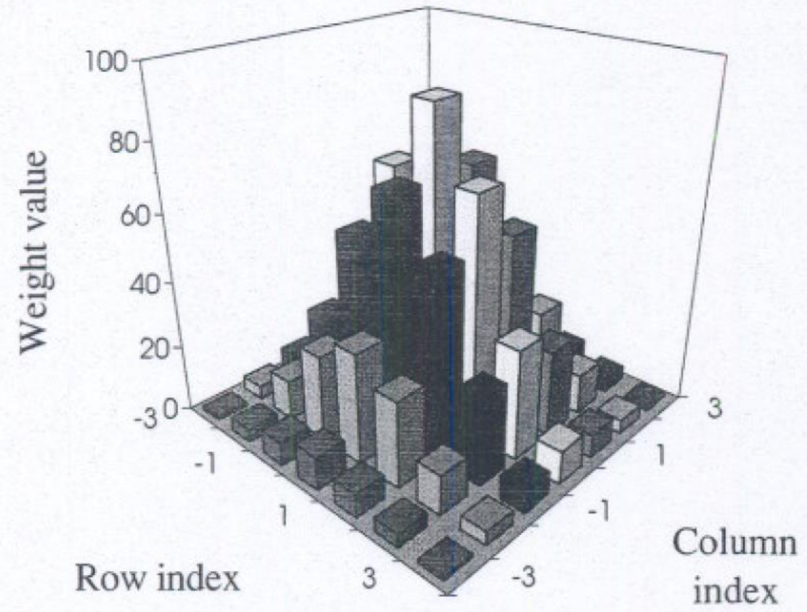
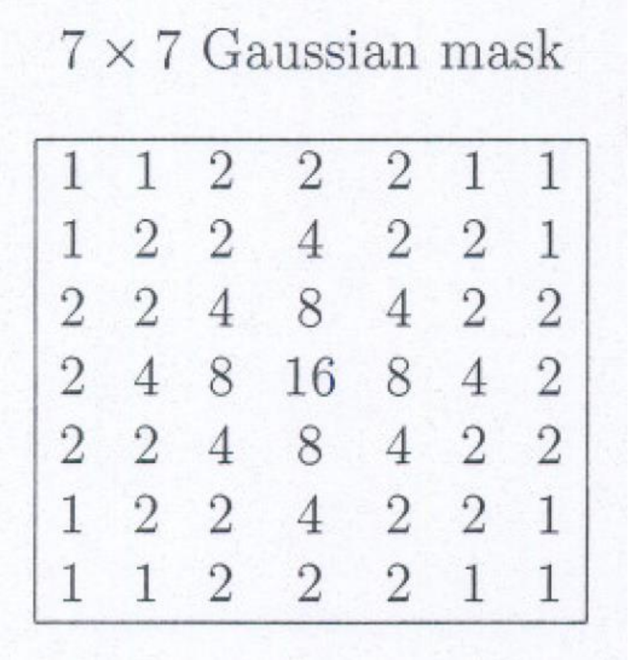


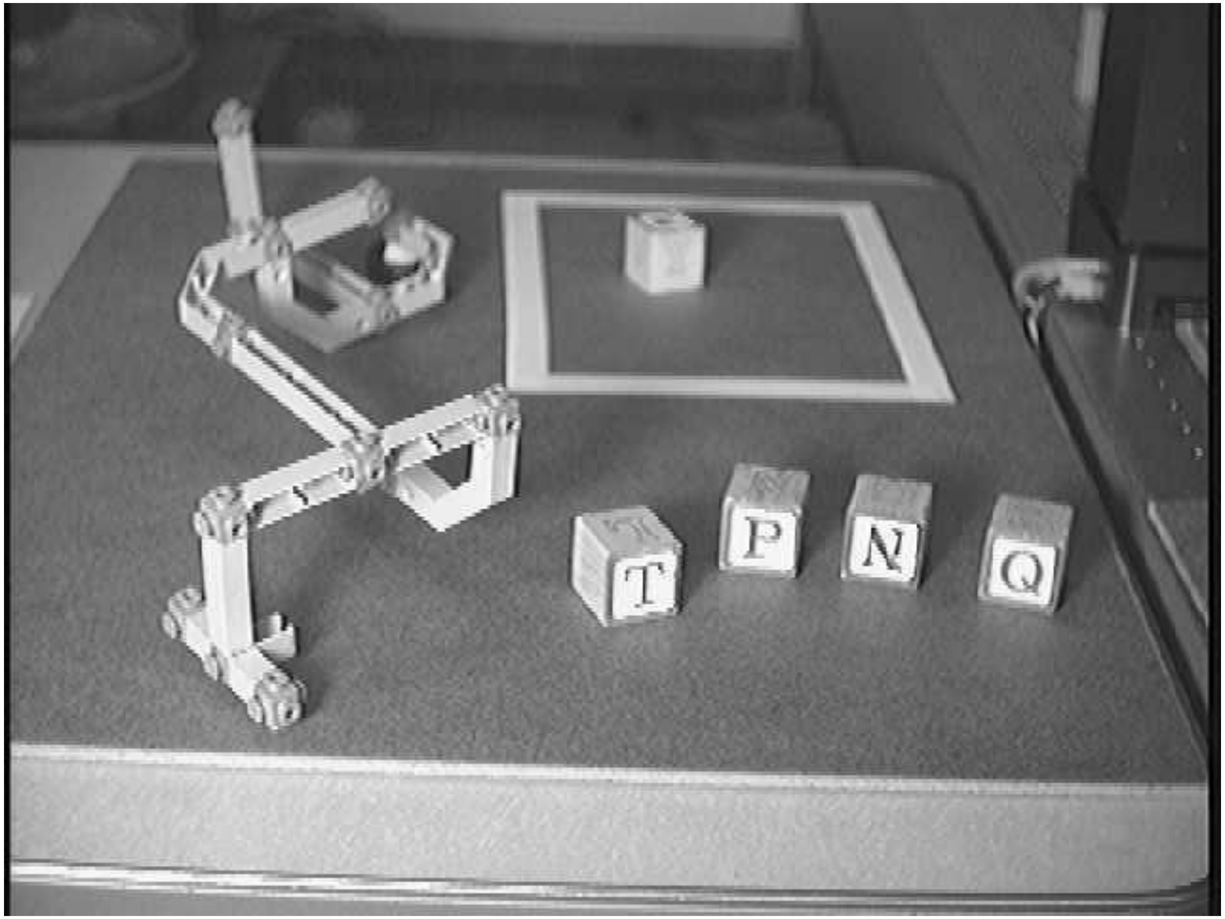
Figure 4.15: A 3-D plot of the 7 × 7 Gaussian mask.



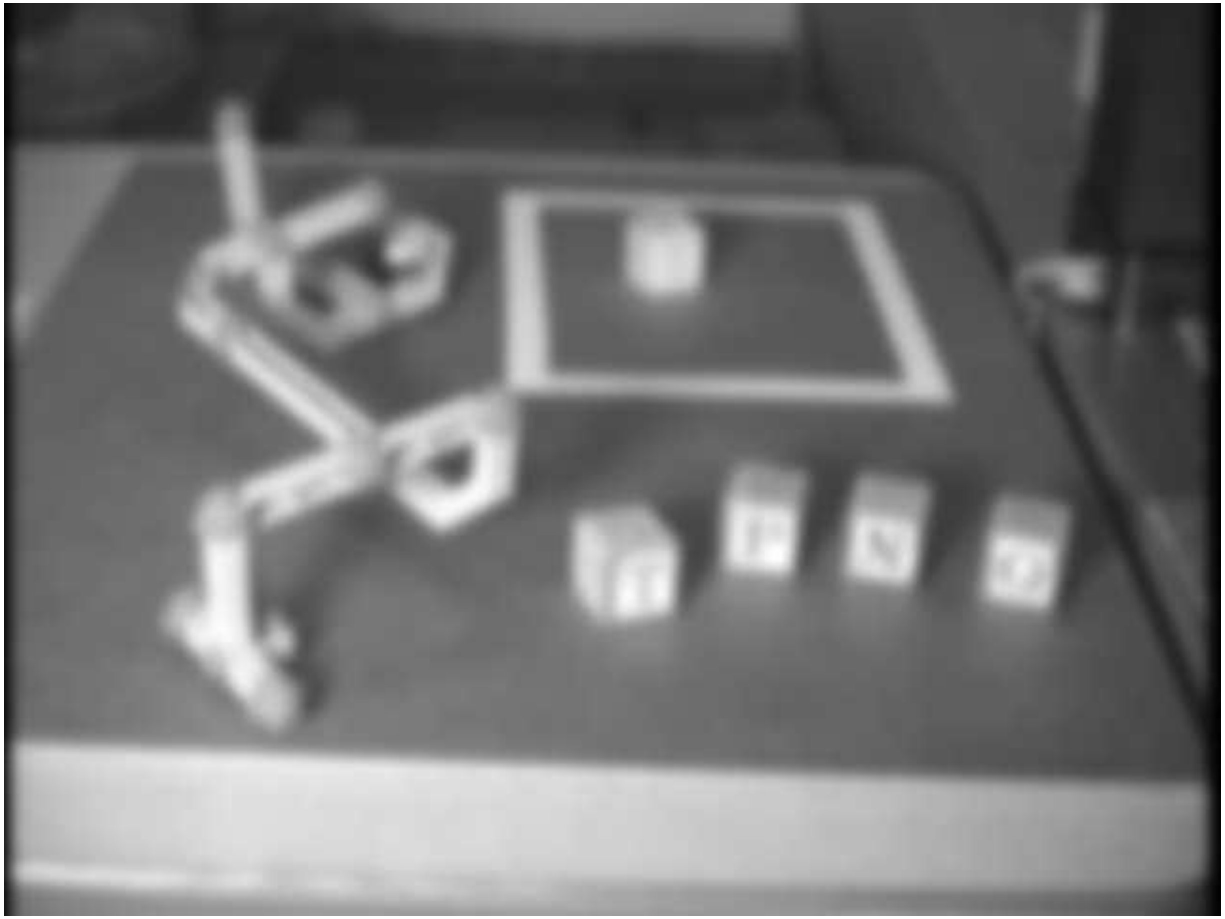
7 × 7 Gaussian mask

1	1	2	2	2	1	1
1	2	2	4	2	2	1
2	2	4	8	4	2	2
2	4	8	16	8	4	2
2	2	4	8	4	2	2
1	2	2	4	2	2	1
1	1	2	2	2	1	1

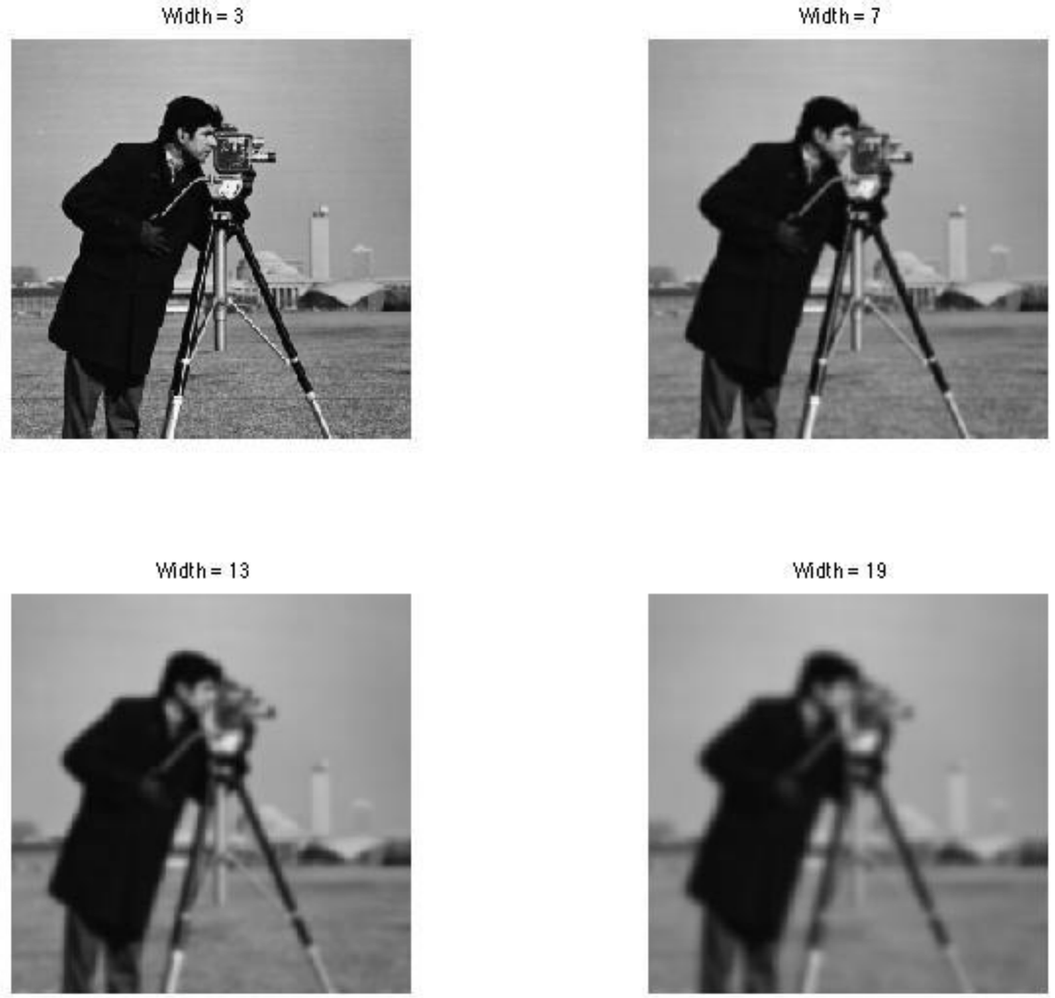
The Effect of Gaussian Filters



The Effect of Gaussian Filters



Kernel Width Affects Scale



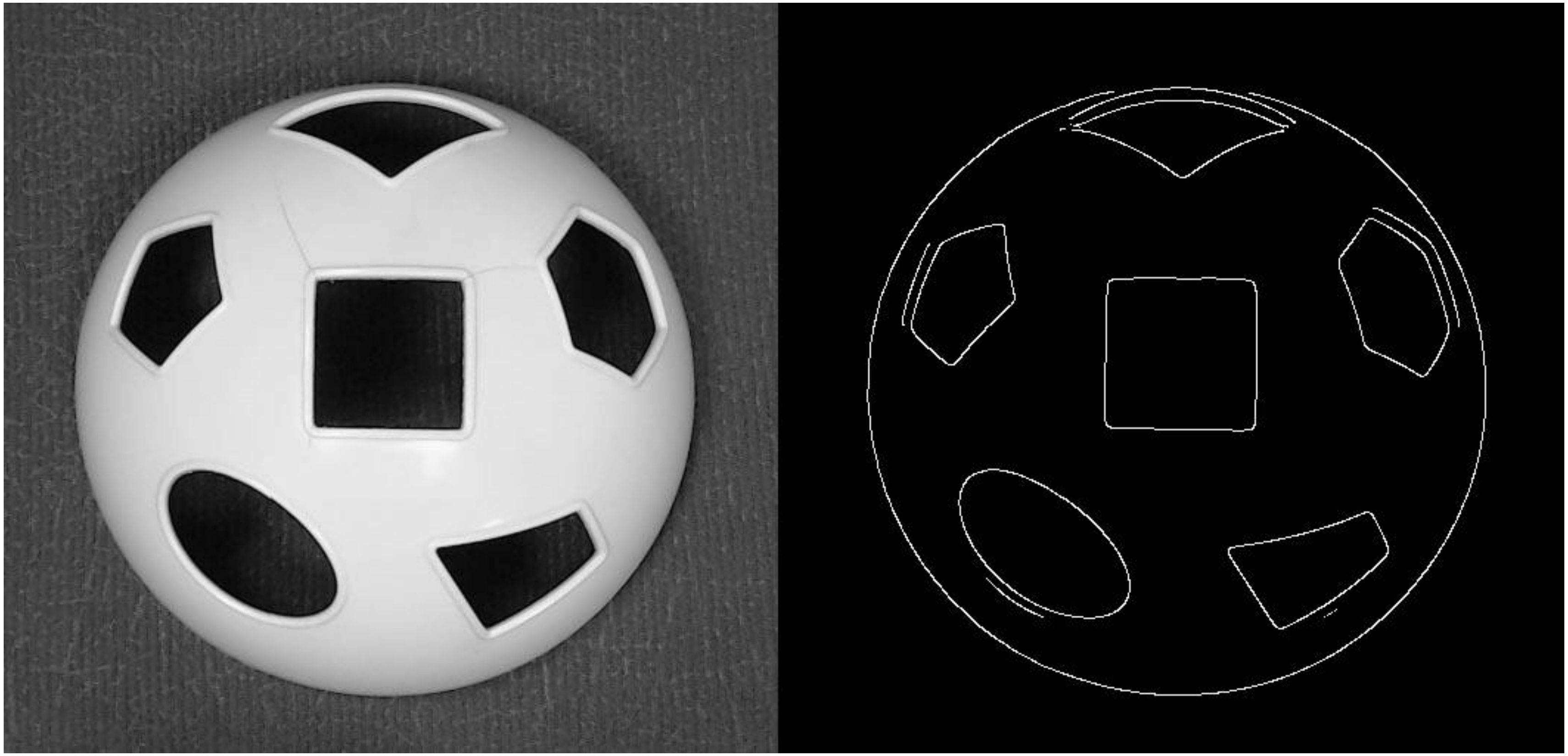
Width = 3

Width = 7

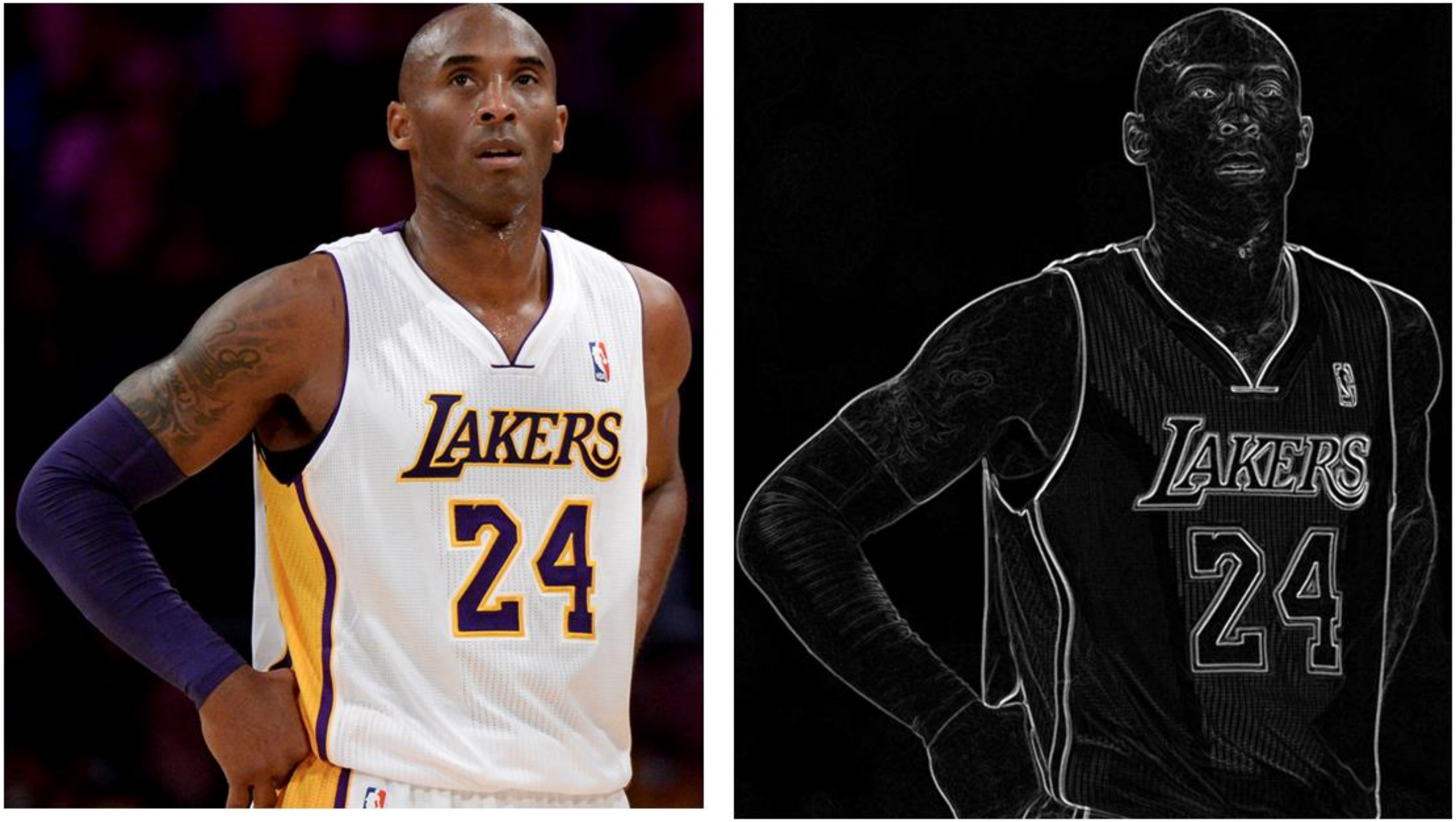
Width = 13

Width = 19

Edge detection



Edge detection



Using Convolution for Edge Detection

Roberts Operator

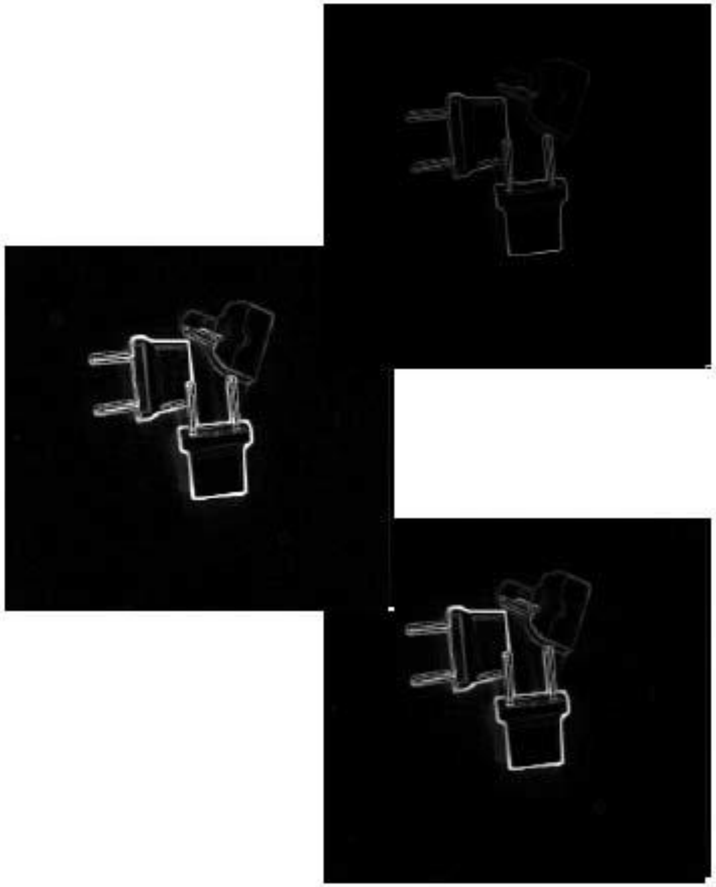
$$G_x \approx \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad G_y \approx \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Sobel Operator

$$G_x \approx \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y \approx \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

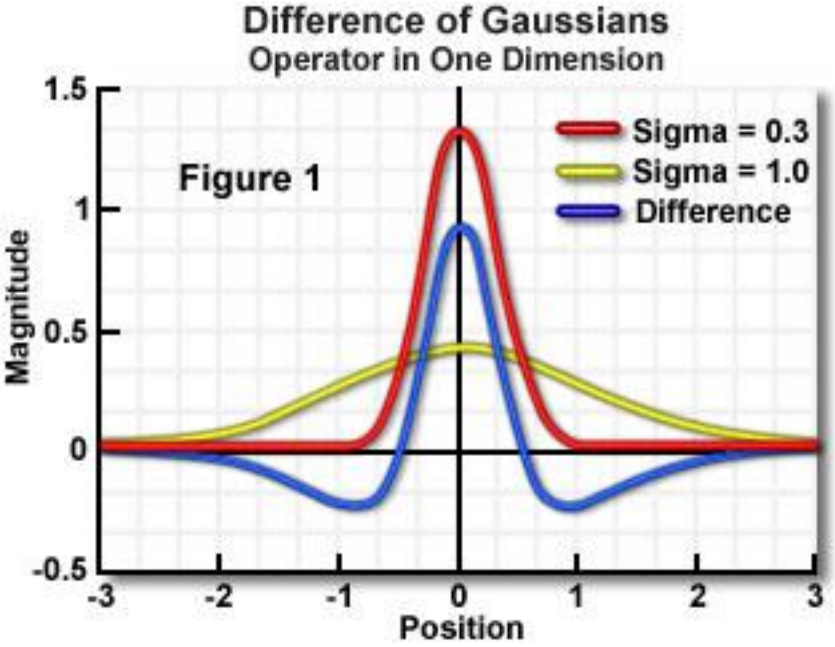
Prewitt Operator

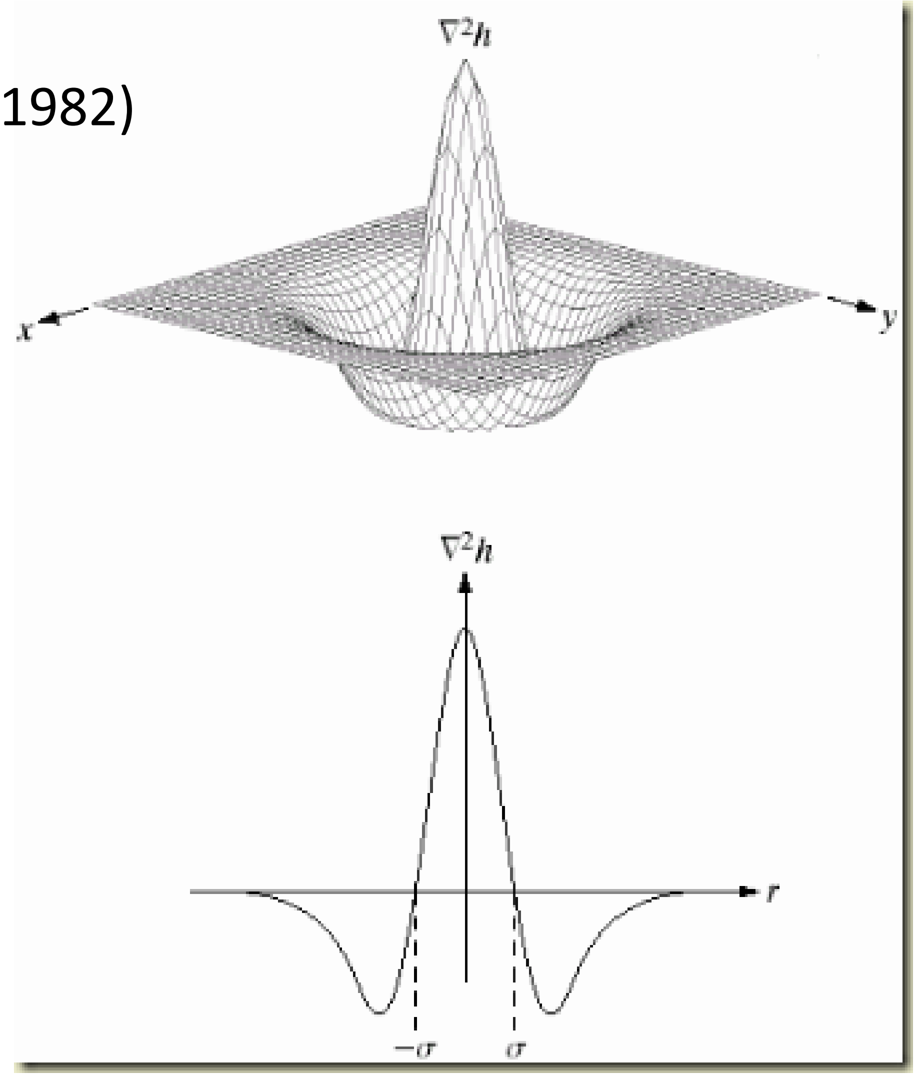
$$G_x \approx \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y \approx \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$



A Variety of Image Filters

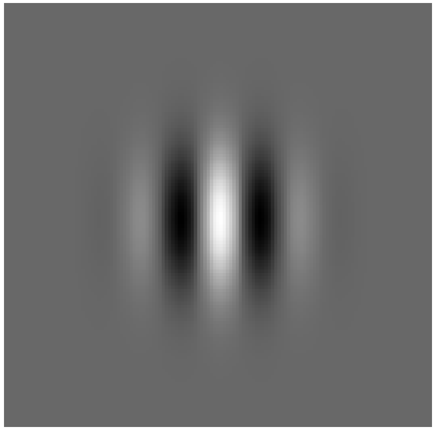
Laplacian of Gaussians (LoG) (Marr 1982)

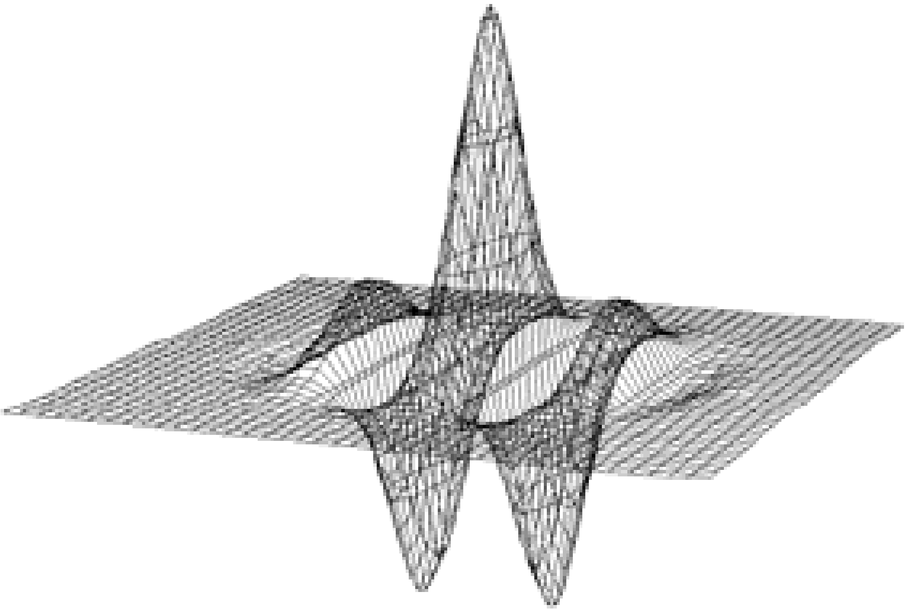




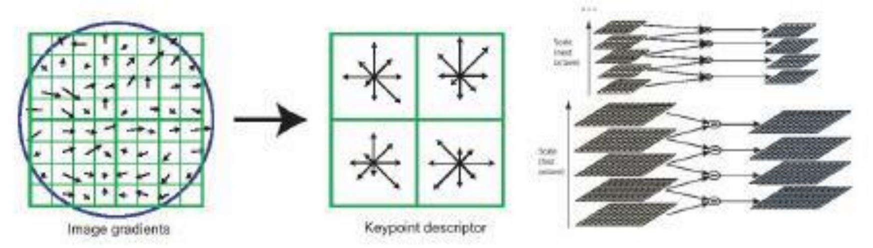
A Variety of Image Filters

Gabor filters (directional) (Daugman 1985)

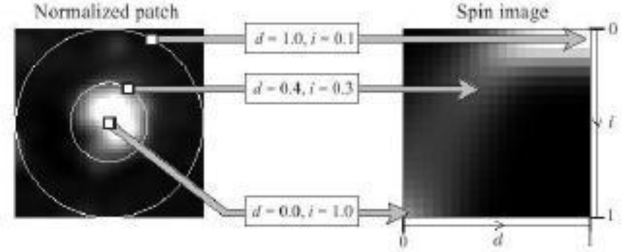




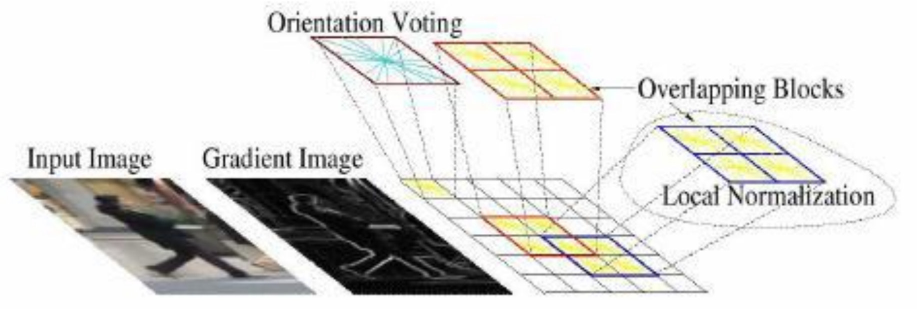
A Variety of Image Filters



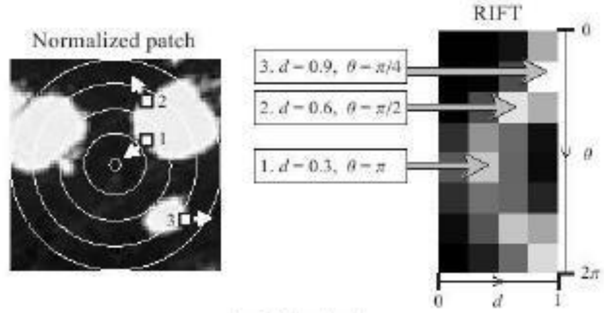
SIFT



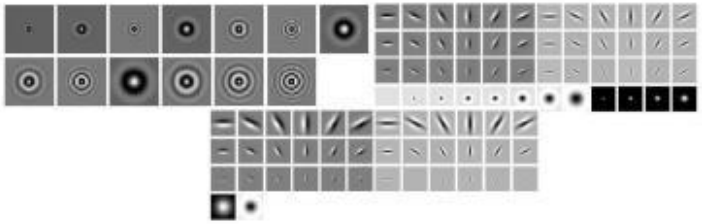
Spin image



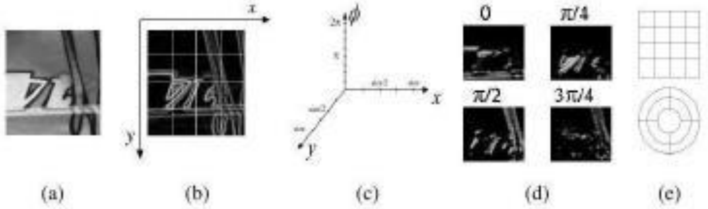
HoG



RIFT



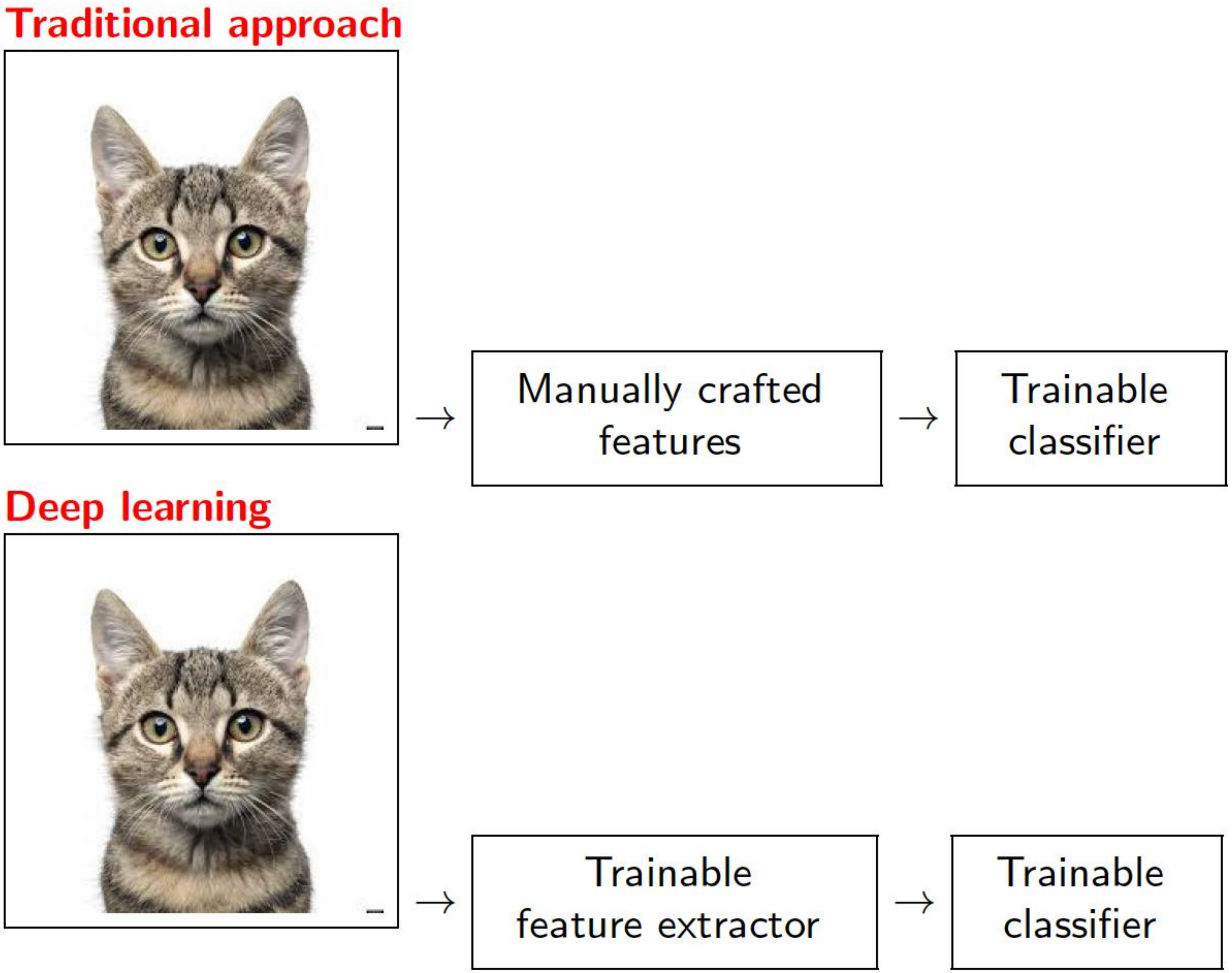
Textons



GLOH

From: M. Sebag

Traditional *vs* Deep Learning Approach



Traditional approach

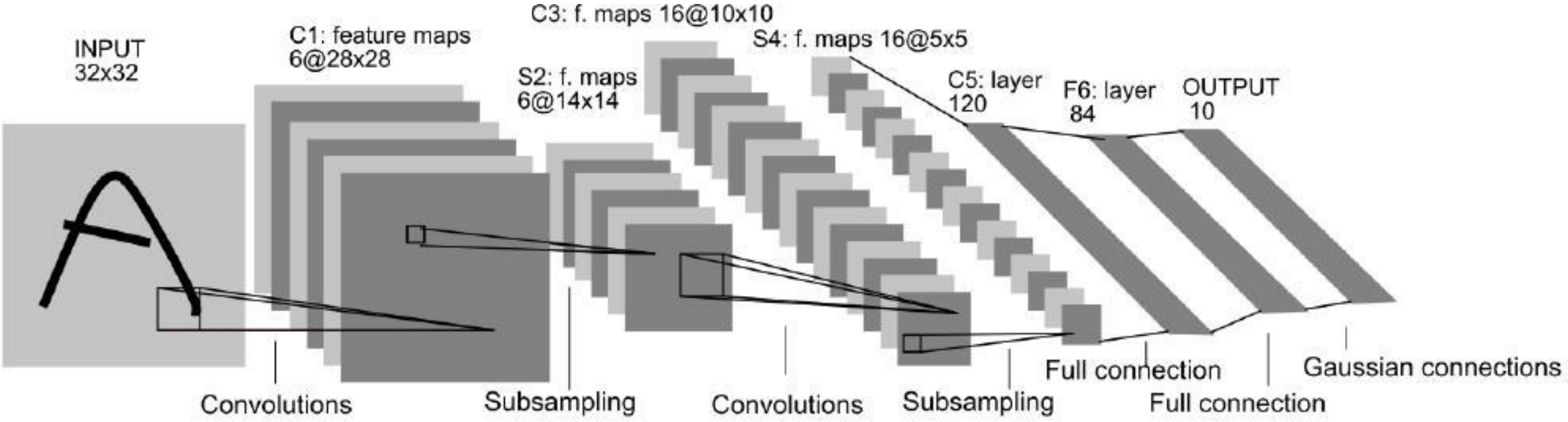
→ Manually crafted features → Trainable classifier

Deep learning

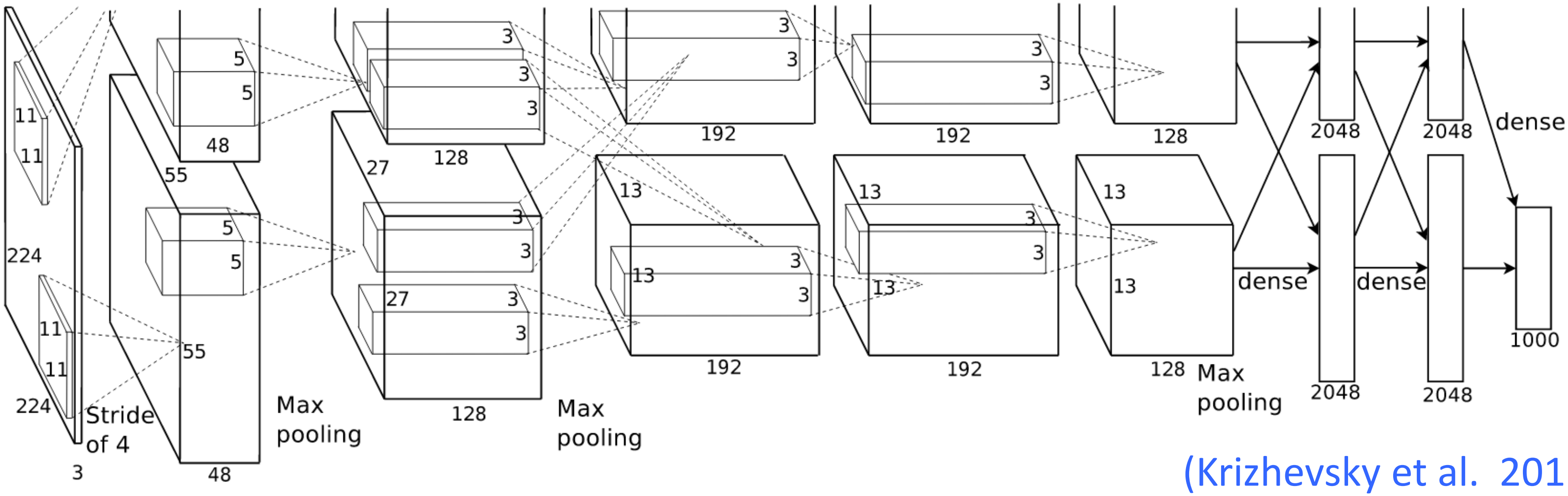
→ Trainable feature extractor → Trainable classifier

From: M. Sebag

Convolutional Neural Networks (CNNs)



(LeCun 1998)



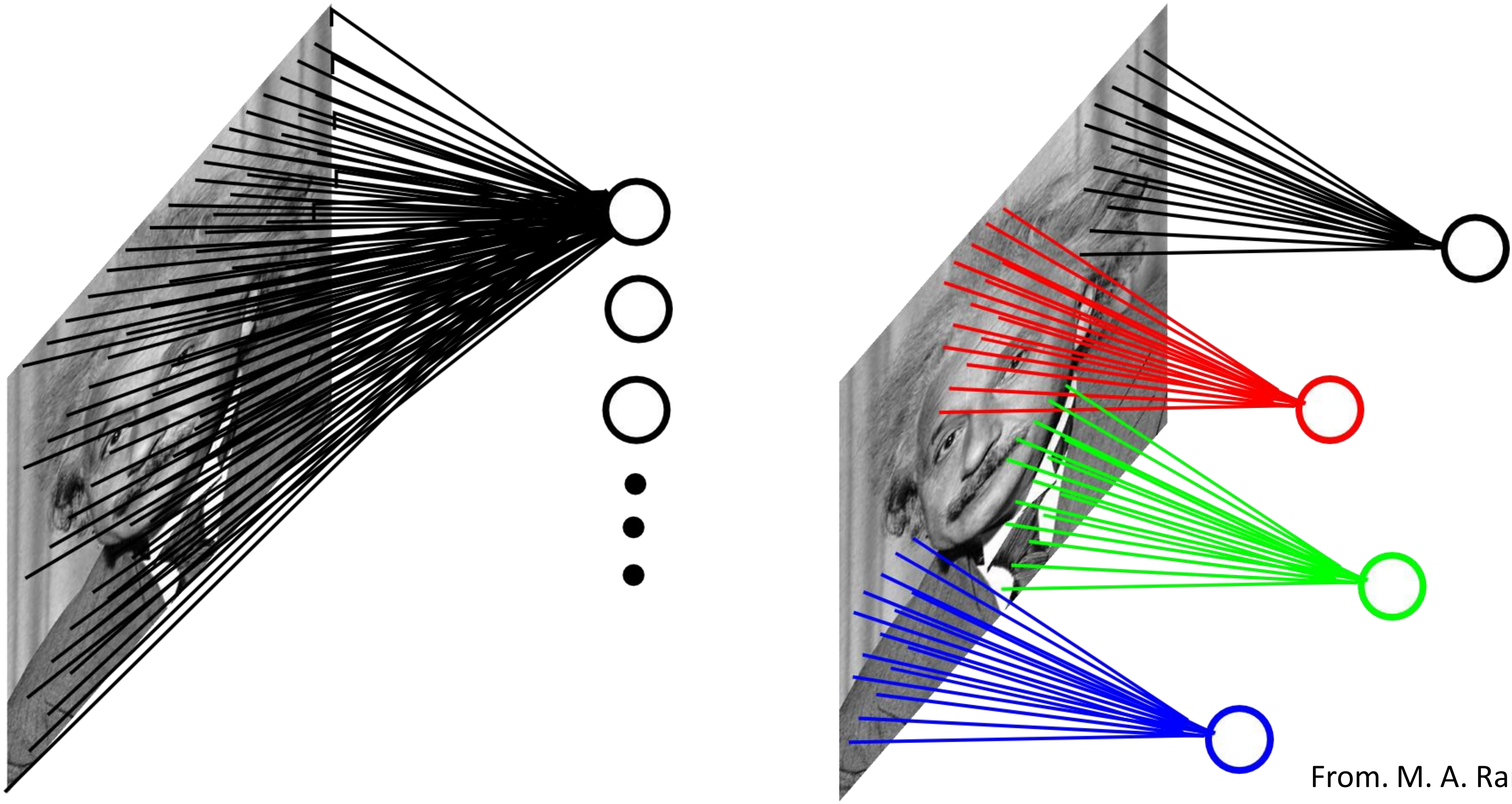
(Krizhevsky et al. 2012)

Fully- *vs* Locally-Connected Networks

Fully-connected: 400,000 hidden units = 16 billion parameters

Locally-connected: 400,000 hidden units 10 x 10 fields = 40 million parameters

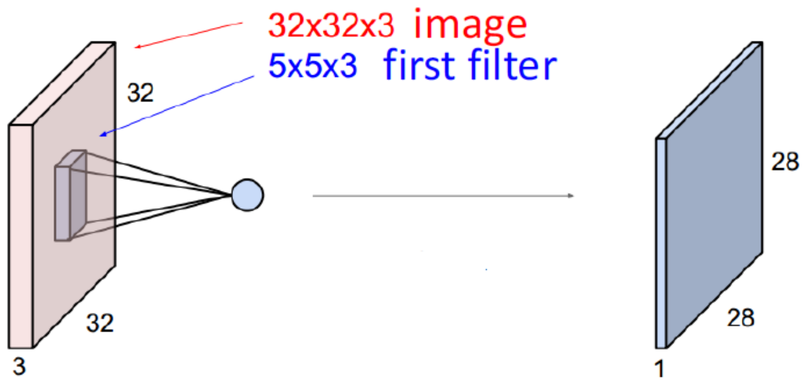
Local connections capture local dependencies

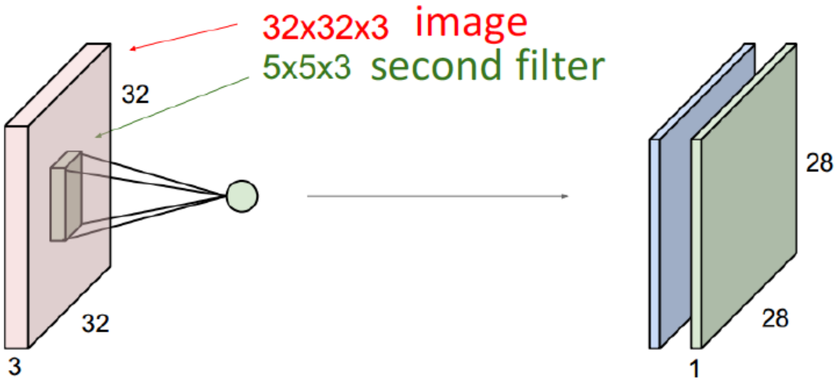


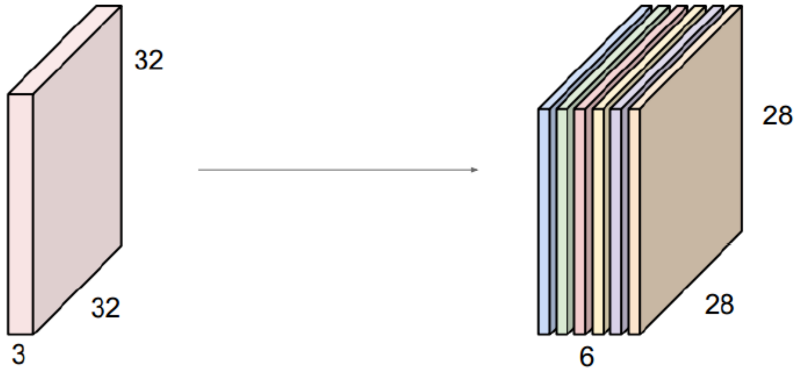
From. M. A. Ranzato

Using Several Trainable Filters

Normally, several filters are packed together and learnt automatically during training

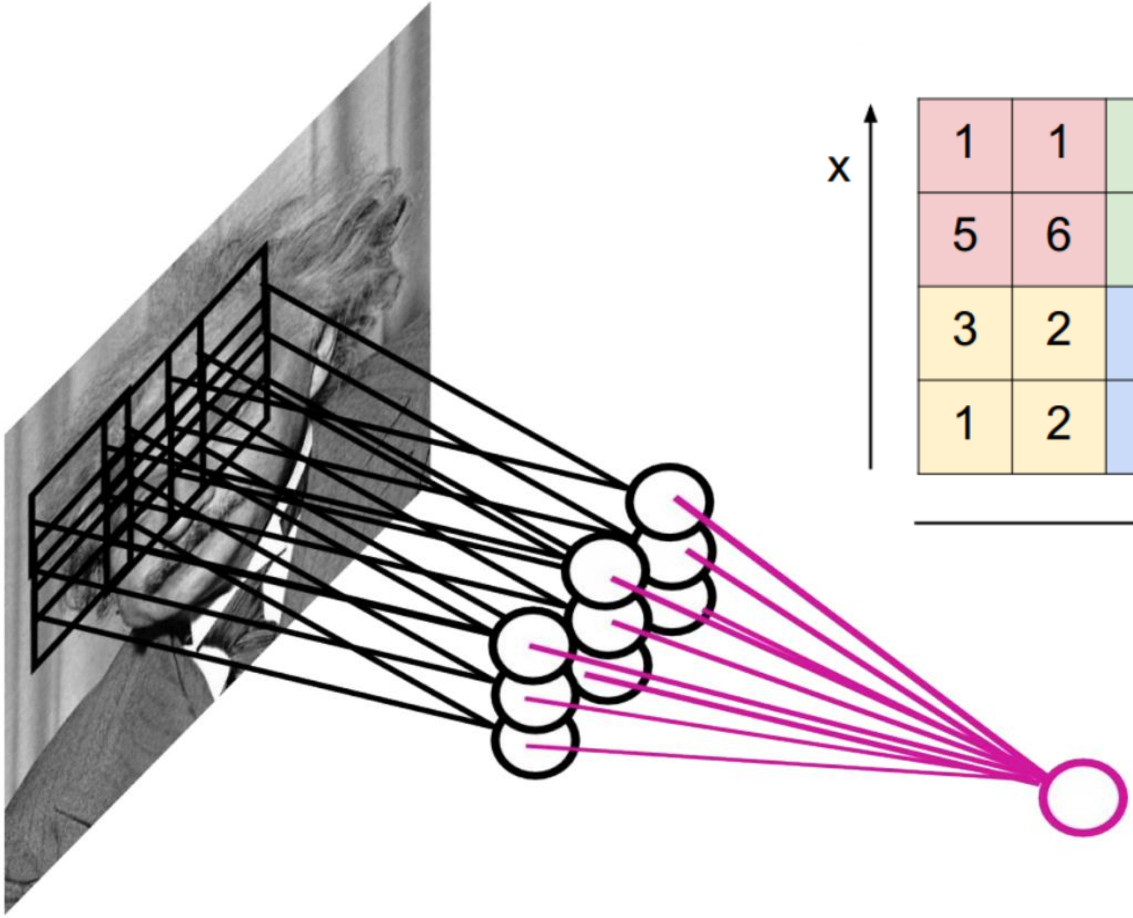


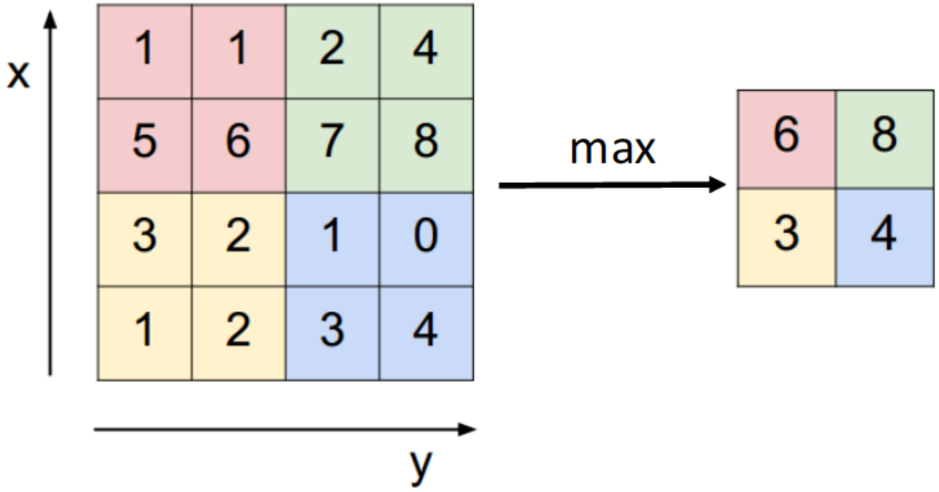




Pooling

Max pooling is a way to simplify the network architecture, by downsampling the number of neurons resulting from filtering operations.

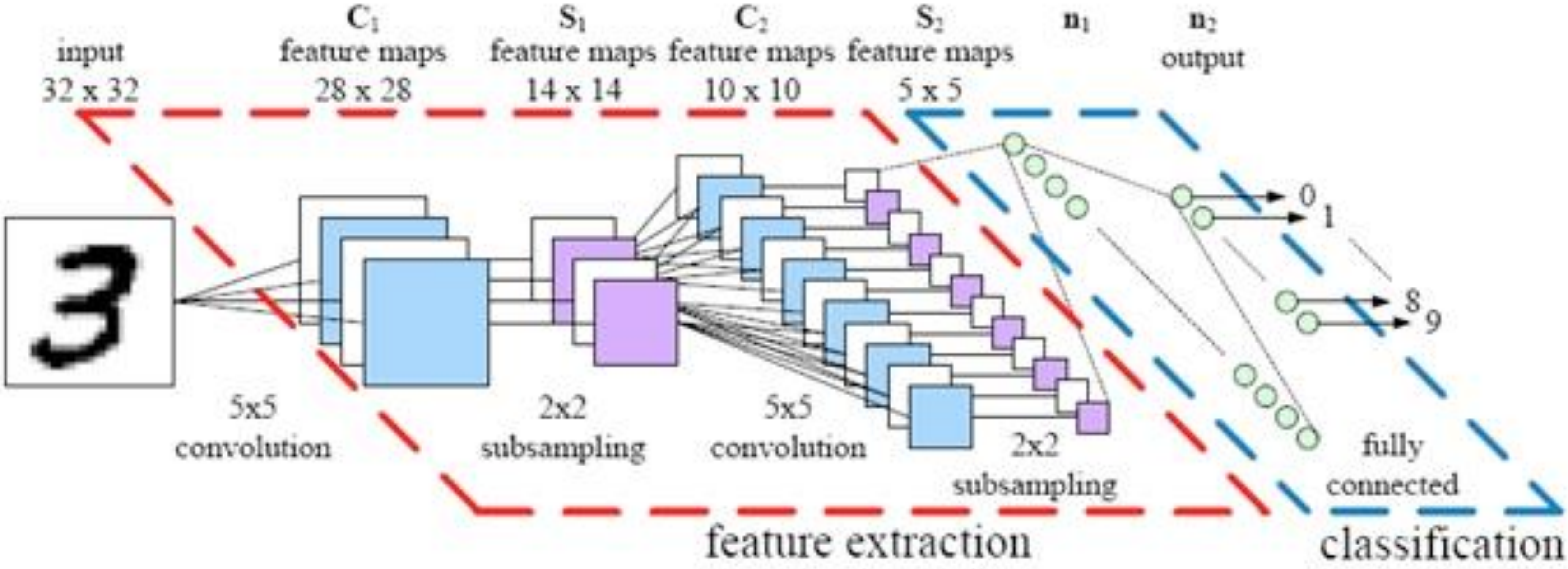




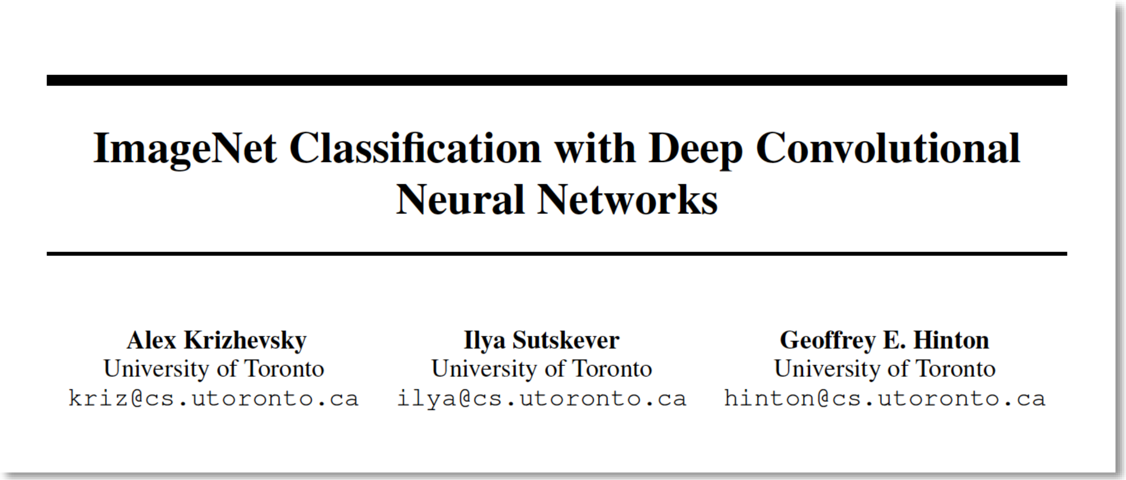
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

6	8
3	4

Combining Feature Extraction and Classification



AlexNet (2012)



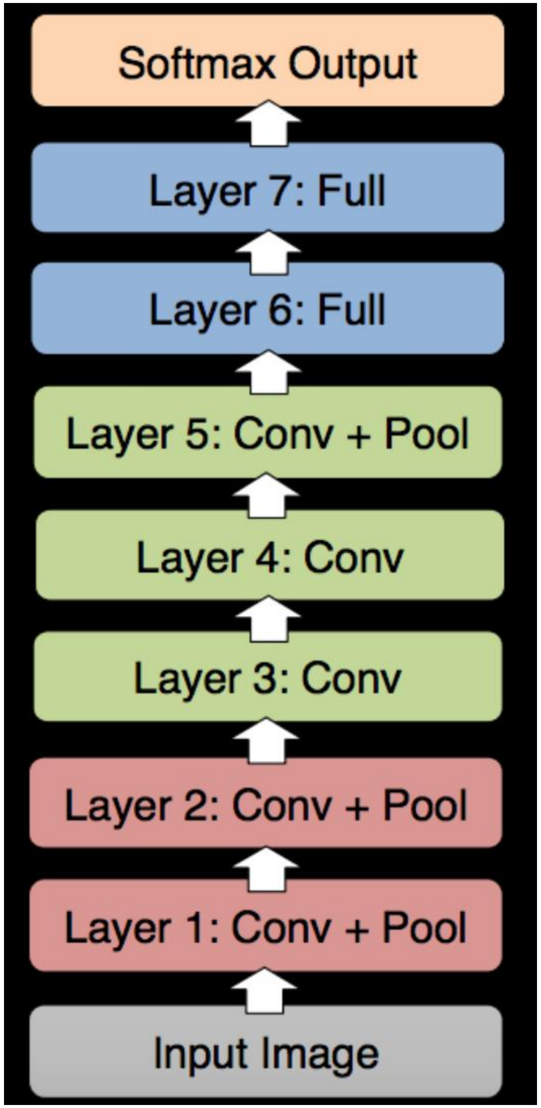
ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

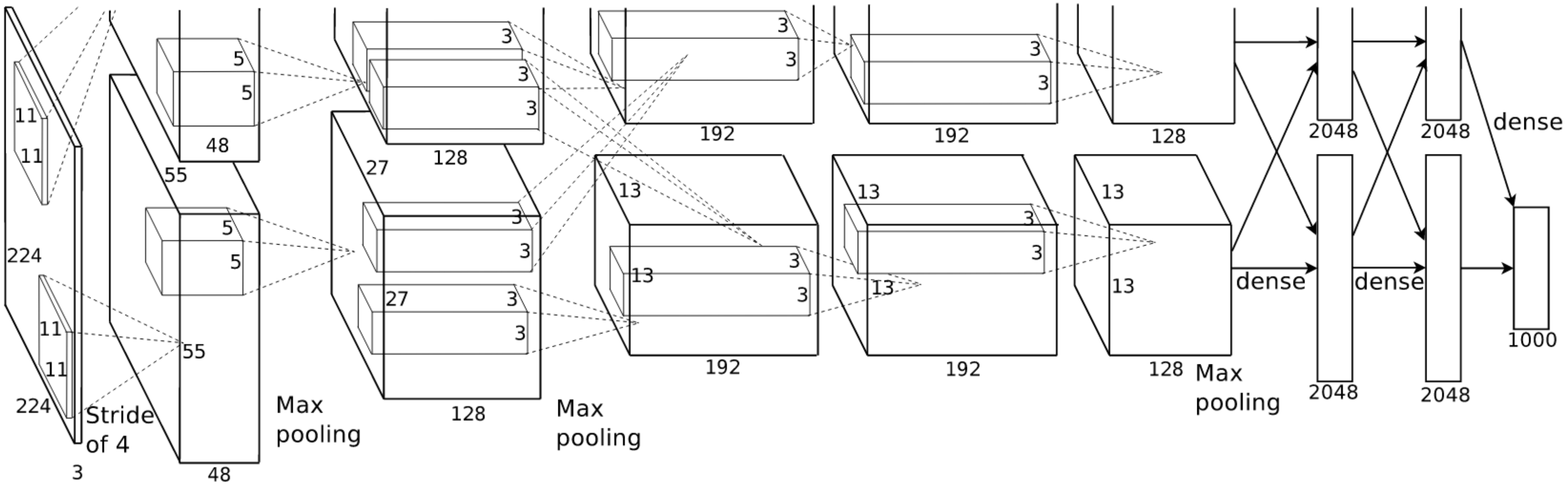
- 8 layers total
- Trained on Imagenet Dataset (1000 categories, 1.2M training images, 150k test images)



AlexNet Architecture

- 1st layer: 96 kernels (11 x 11 x 3)
- Normalized, pooled
- 2nd layer: 256 kernels (5 x 5 x 48)
- Normalized, pooled
- 3rd layer: 384 kernels (3 x 3 x 256)
- 4th layer: 384 kernels (3 x 3 x 192)
- 5th layer: 256 kernels (3 x 3 x 192)
- Followed by 2 fully connected layers, 4096 neurons each
- Followed by a 1000-way SoftMax layer

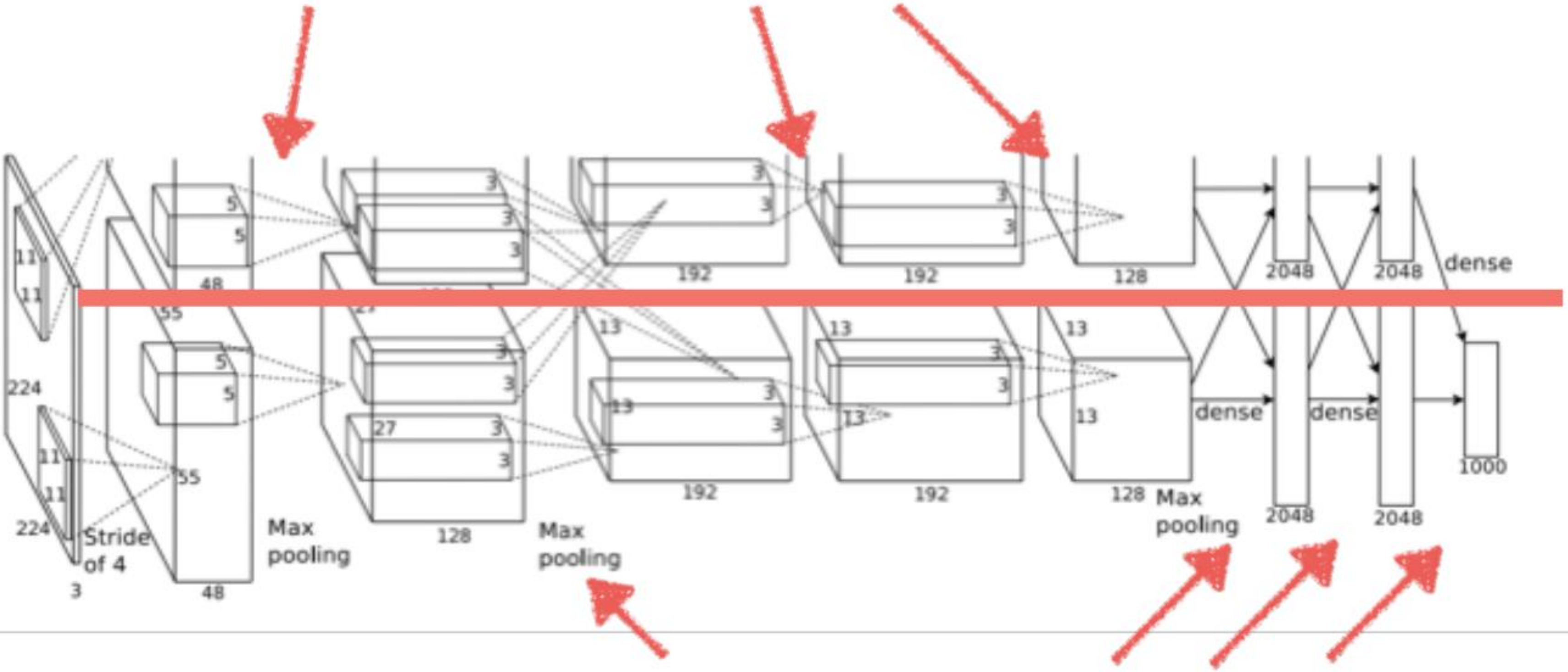
650,000 neurons
60 million parameters



Training on Multiple GPU’s

GPU #1

intra-GPU connections



inter-GPU connections

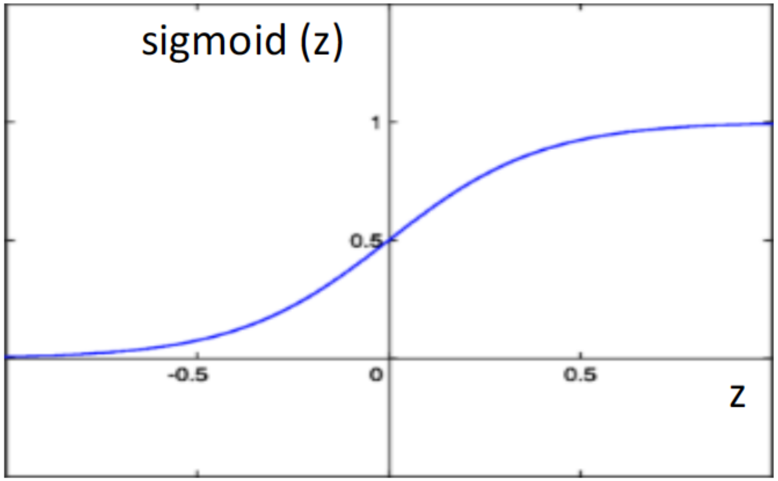
GPU #2

Rectified Linear Units (ReLU's)

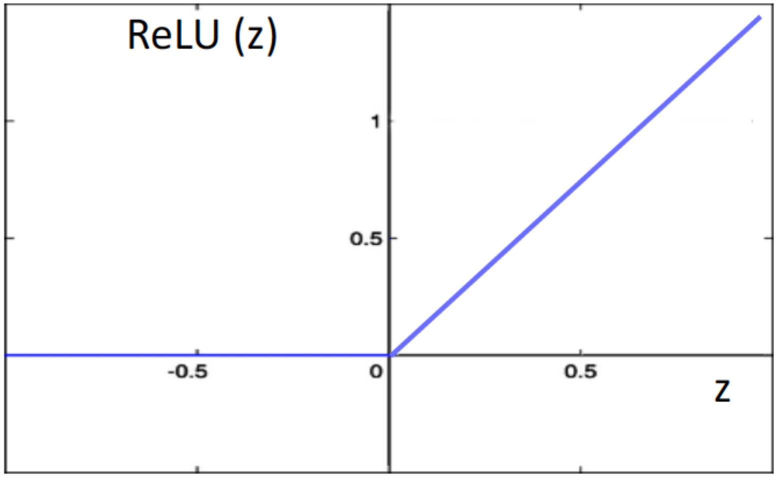
Problem: Sigmoid activation takes on values in (0,1). Propagating the gradient back to the initial layers, it tends to become 0 (vanishing gradient problem).

From a practical perspective, this slows down the training procedure of the initial layers of the network.

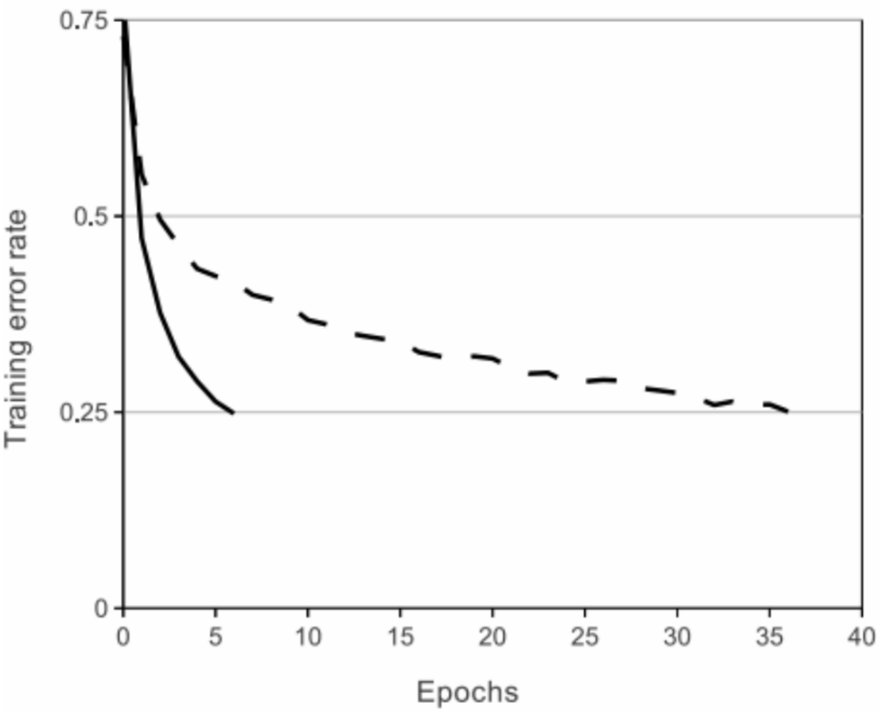
sigmoid(z) = $\frac{1}{1+e^{-z}}$



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



Rectified Linear Units (ReLU’s)



A 4 layer CNN with ReLUs (solid line) converges six times faster than an equivalent network with tanh neurons (dashed line) on CIFAR-10 dataset

Mini-batch Stochastic Gradient Descent

Loop:

1. **Sample** a batch of data
2. **Forward** prop it through the graph, get loss
3. **Backprop** to calculate the gradients
4. **Update** the parameters using the gradient

Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations

AlexNet uses two forms of this **data augmentation**.

- The first form consists of generating image translations and horizontal reflections.
- The second form consists of altering the intensities of the RGB channels in training images.

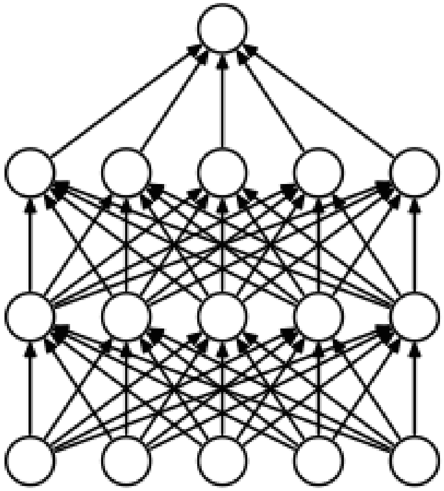
Dropout

Set to zero the output of each hidden neuron with probability 0.5.

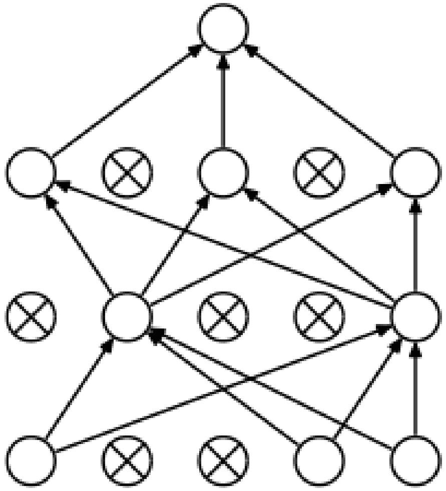
The neurons which are “dropped out” in this way do not contribute to the forward pass and do not participate in backpropagation.

So every time an input is presented, the neural network samples a different architecture, but all these architectures share weights.

Reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons.



Standard Neural Net



After applying dropout.

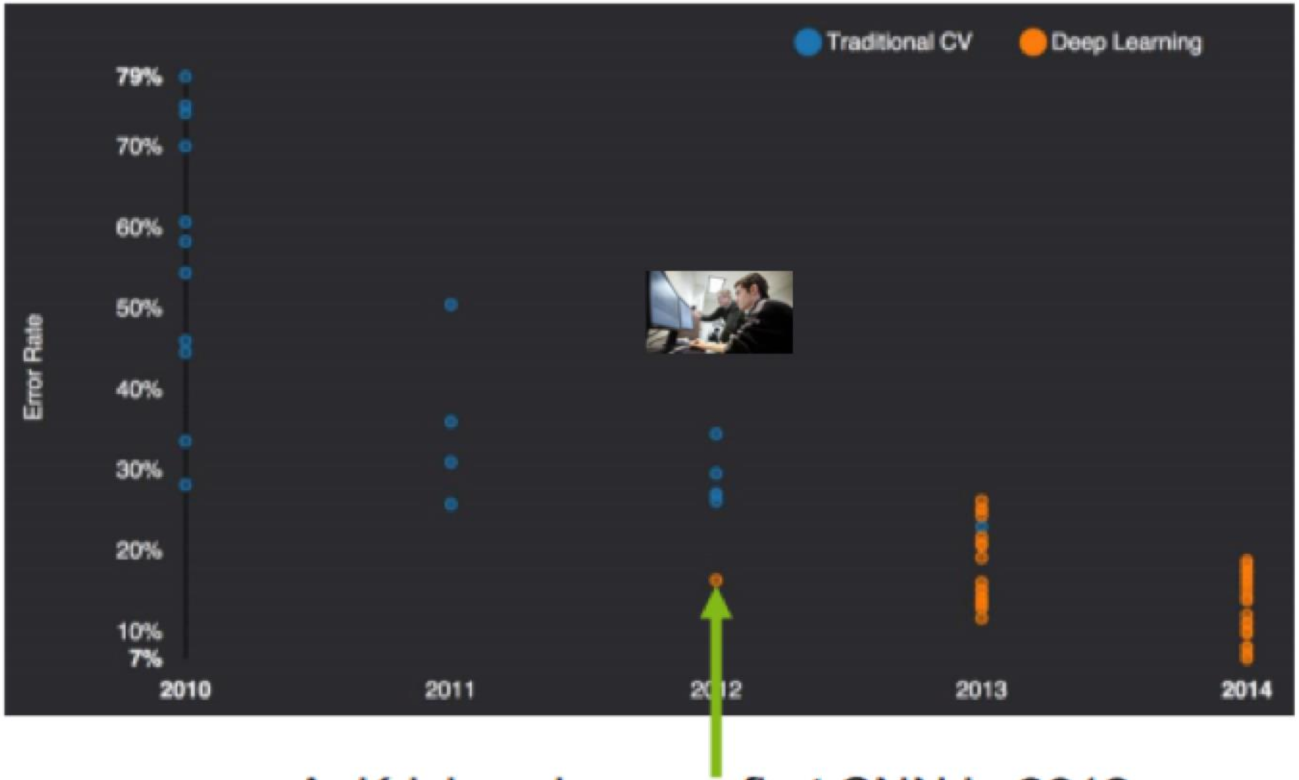
ImageNet

IMAGENET

[Deng et al. CVPR 2009]

- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon Turk
- Challenge: 1.2 million training images, 1000 classes

ImageNet Challenges

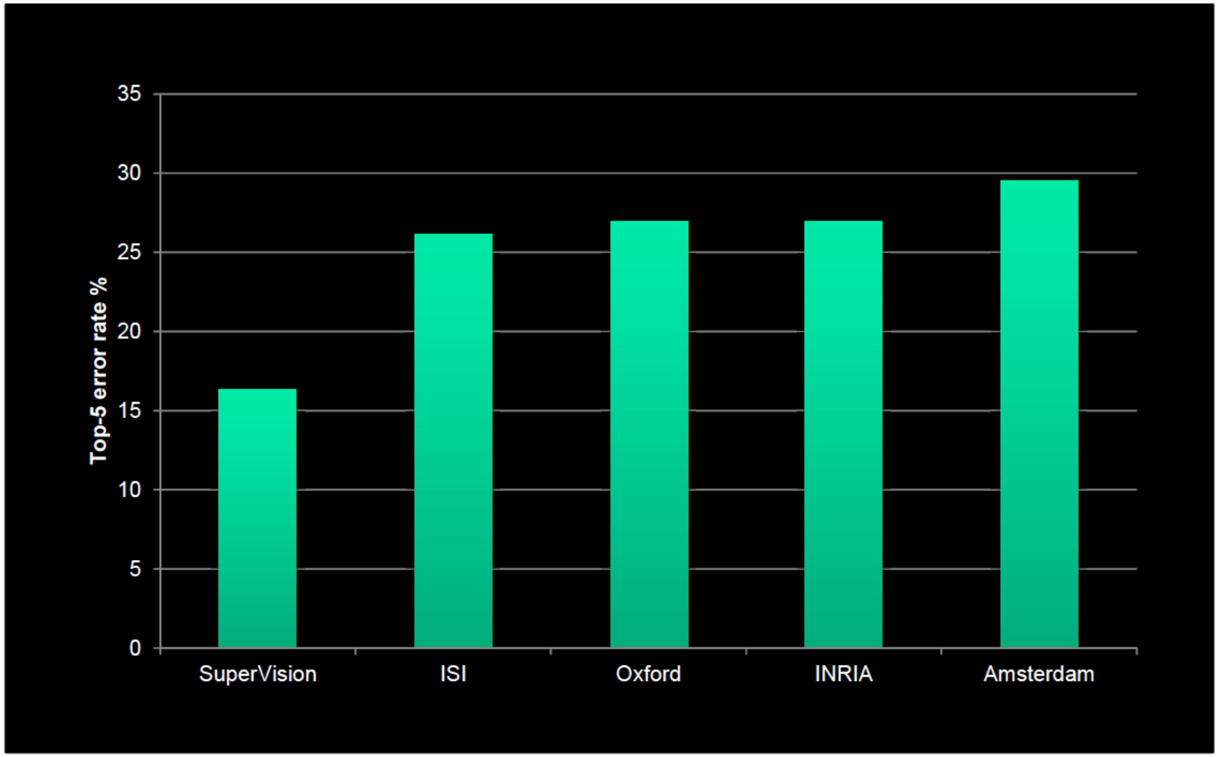


A. Krizhevsky uses first CNN in 2012.
Trained on Gaming Graphic Cards

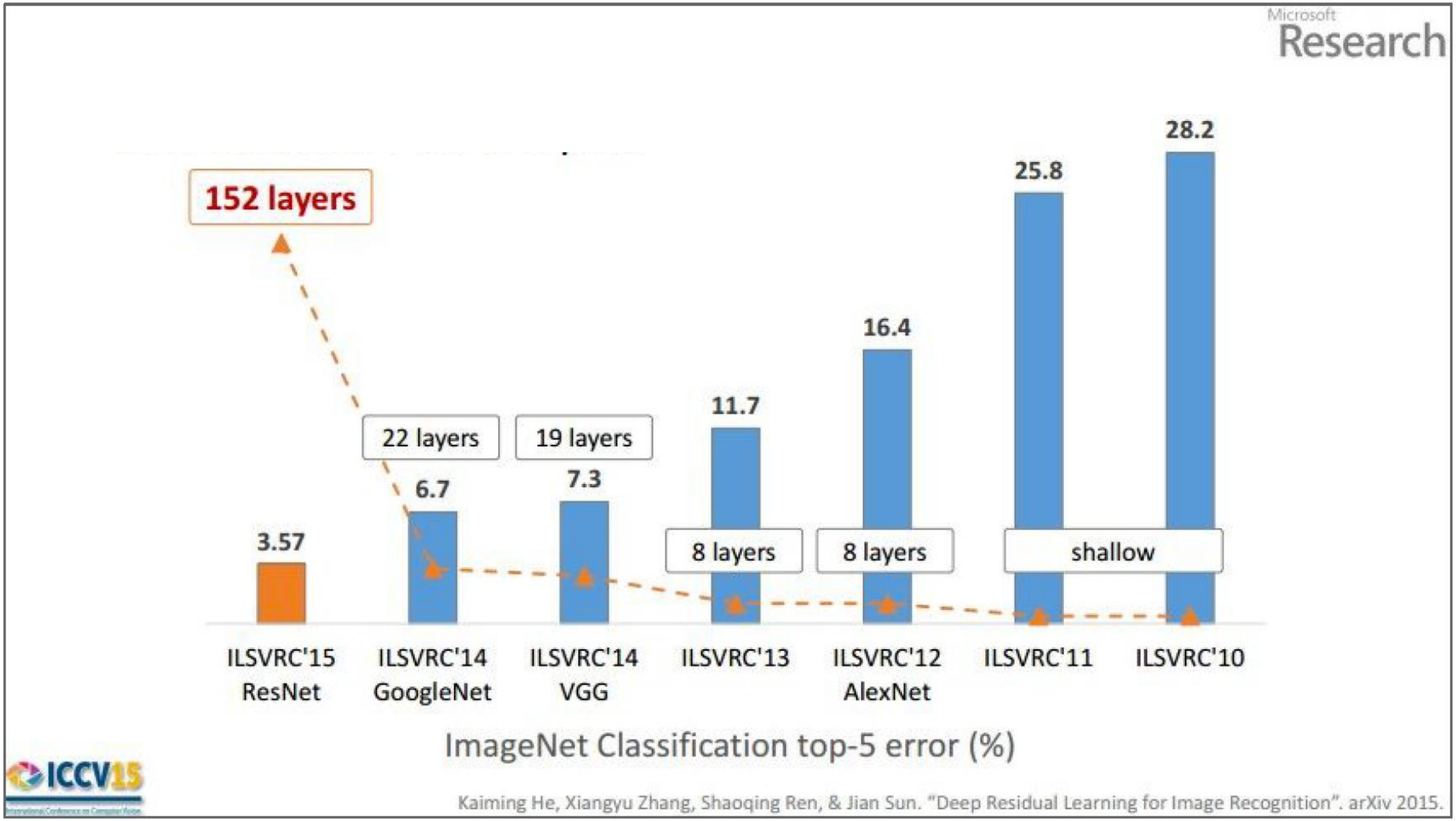
ImageNet Challenge 2012

Krizhevsky et al. -- **16.4% error** (top-5)

Next best (non-convnet) – **26.2% error**

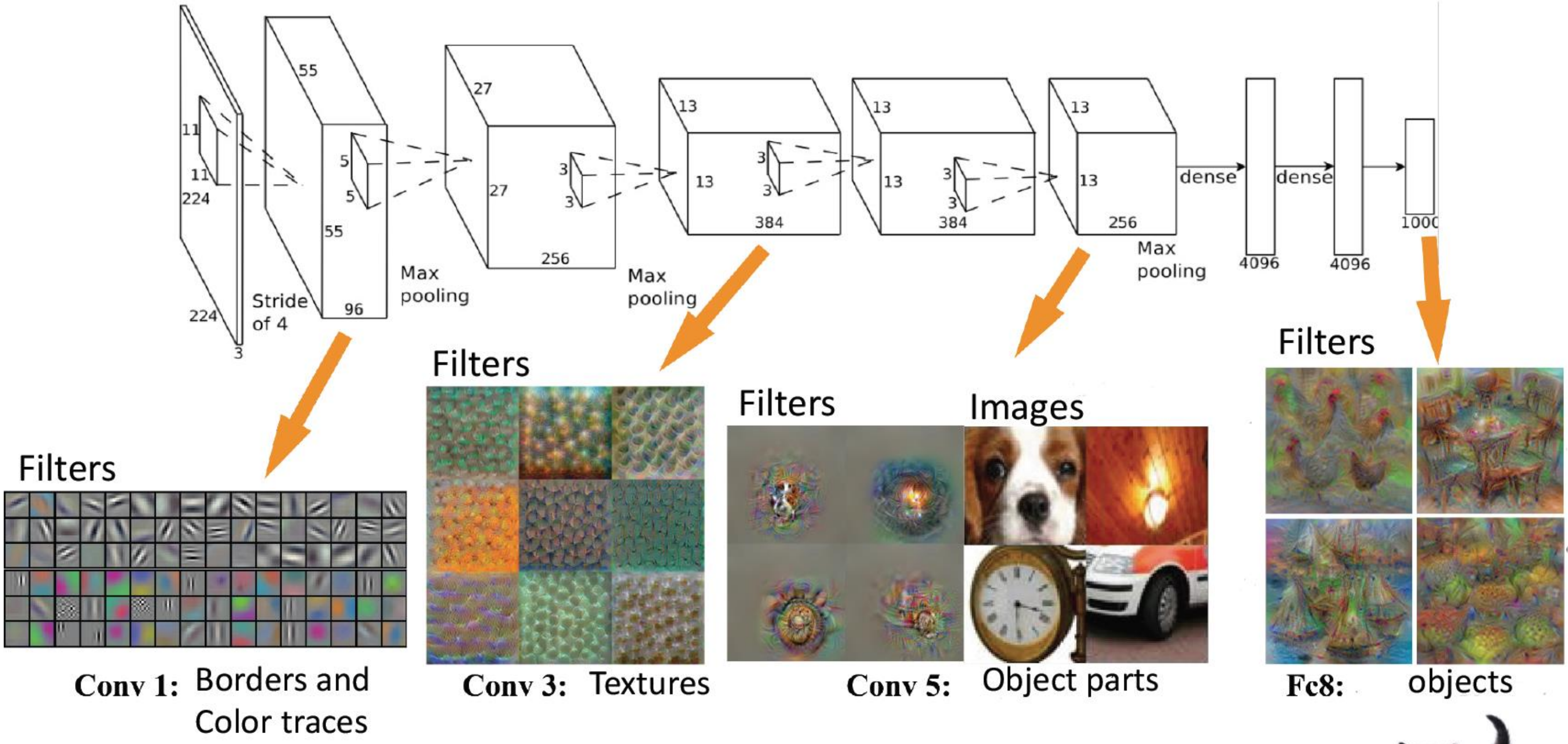


Revolution of Depth



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

A Hierarchy of Features

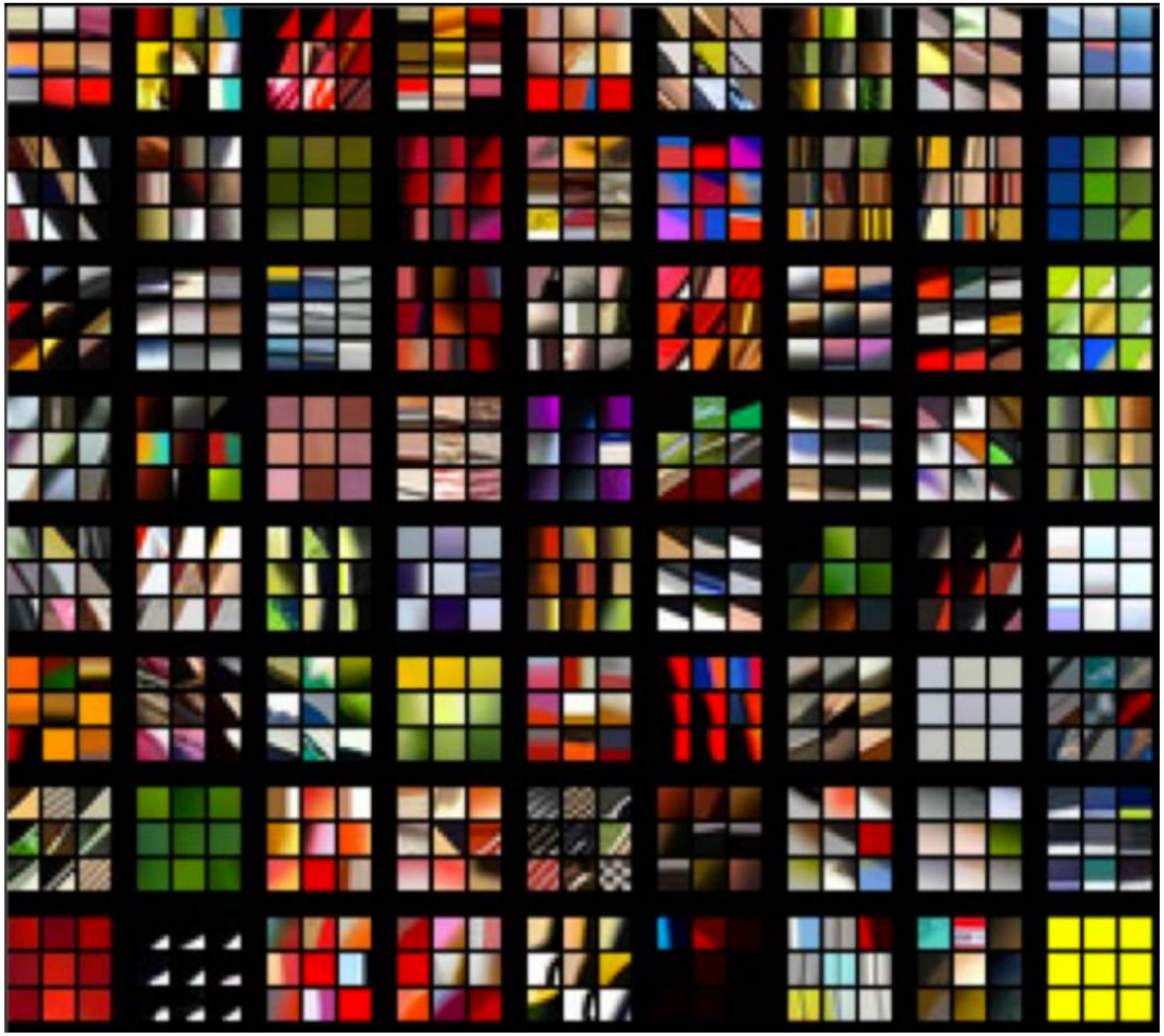


The deep network gradually learns more complex and abstract notions

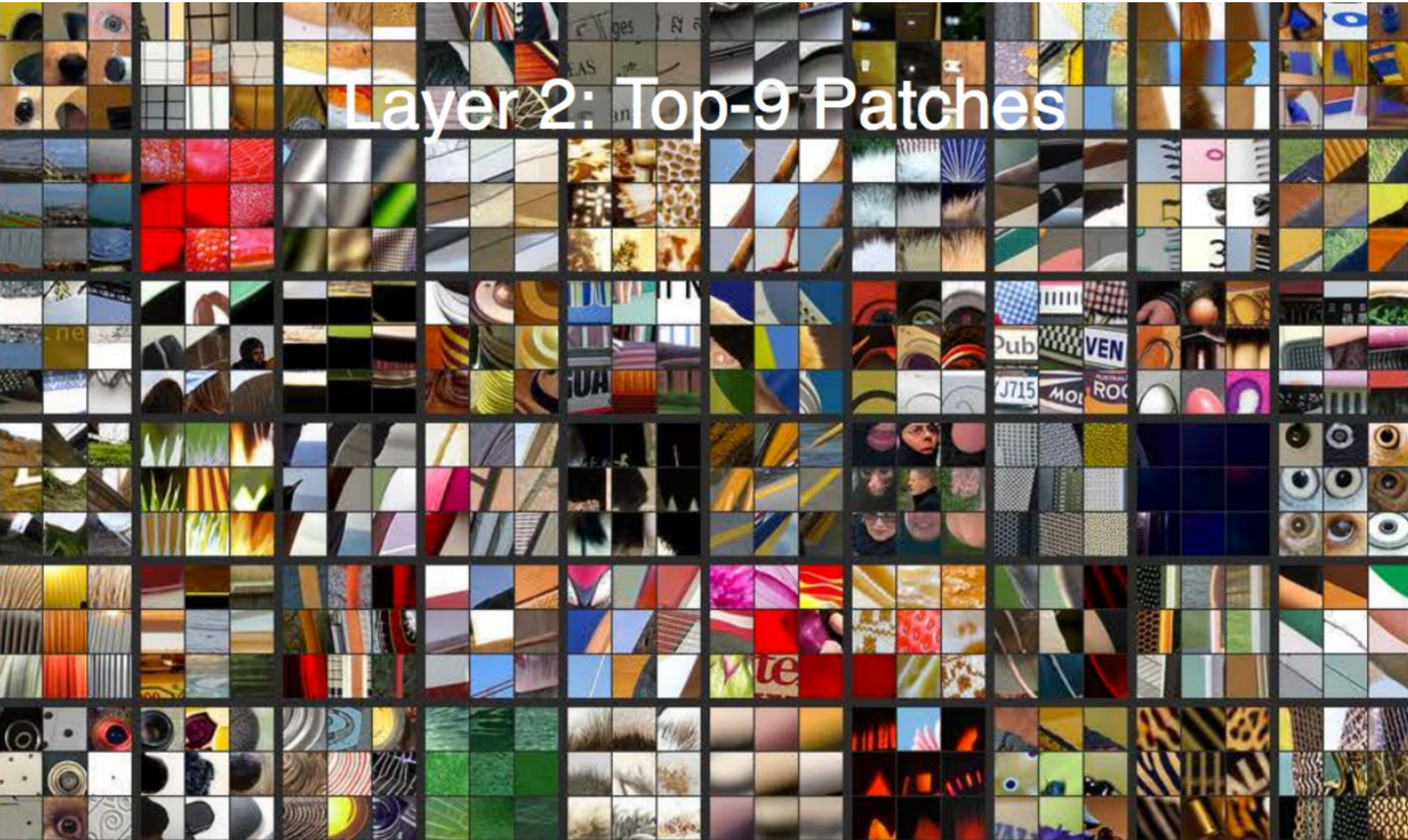
From: B. Biggio

Layer 1

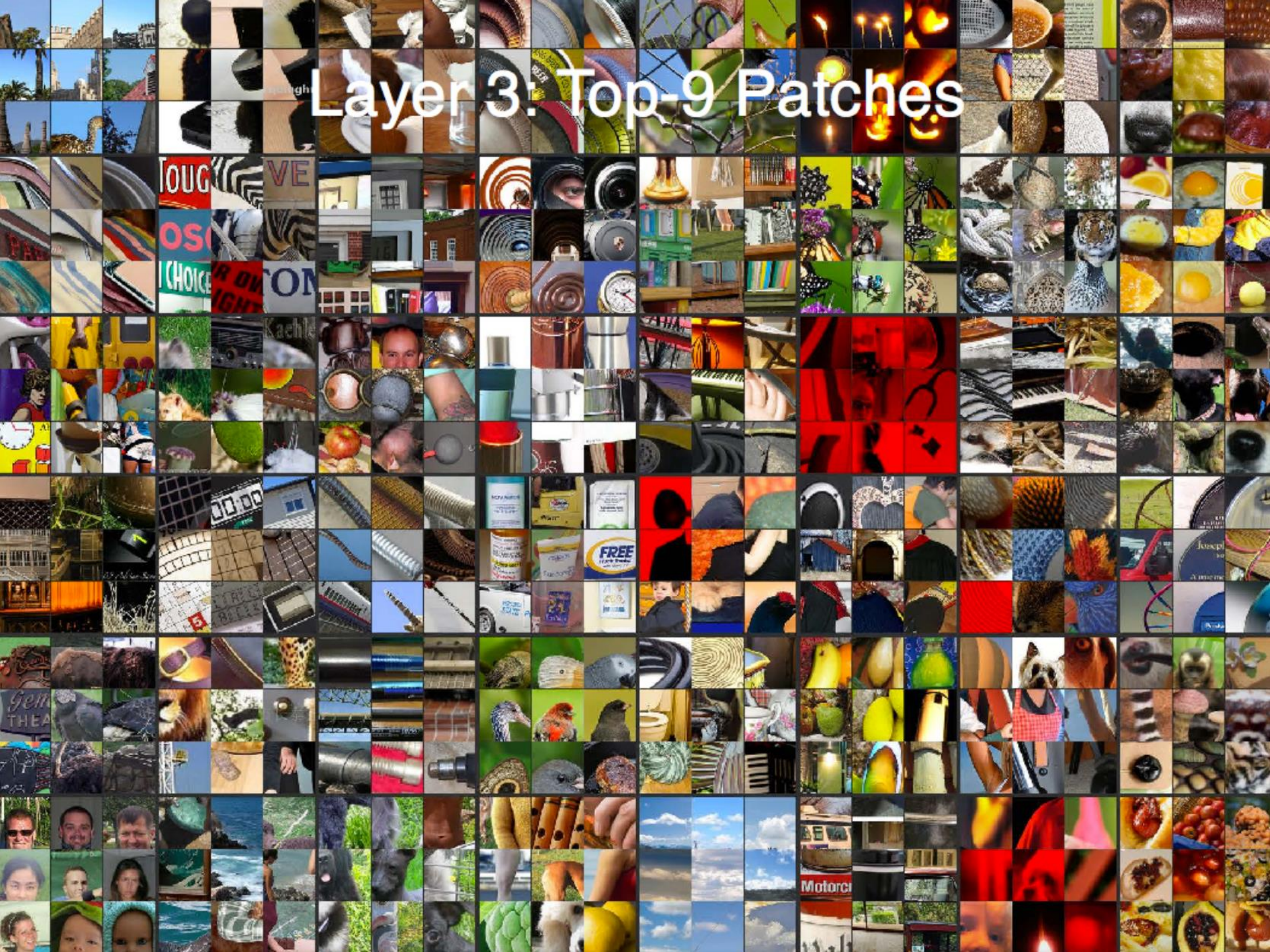
Each 3x3 block shows the top 9 patches for one filter



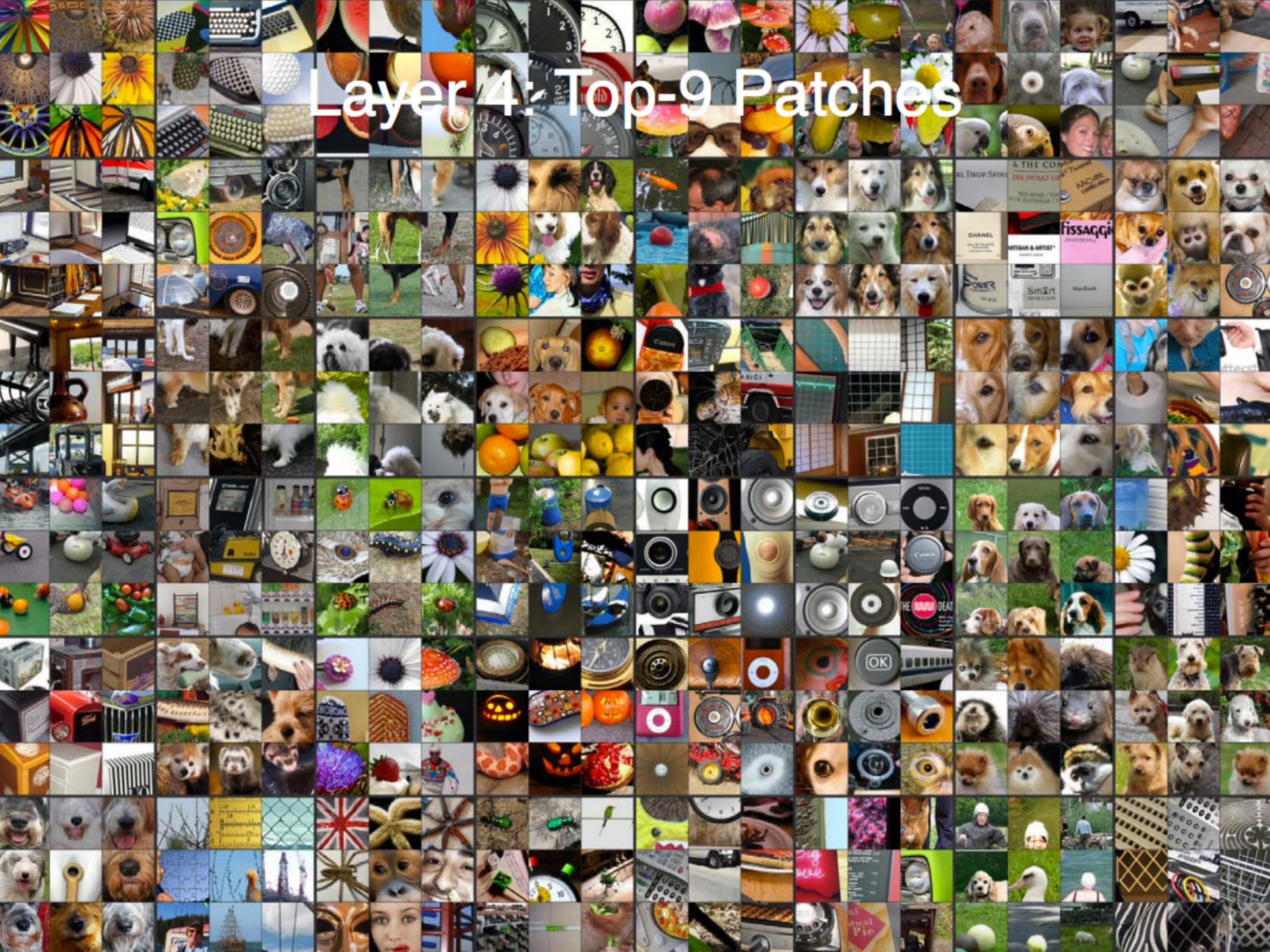
Layer 2



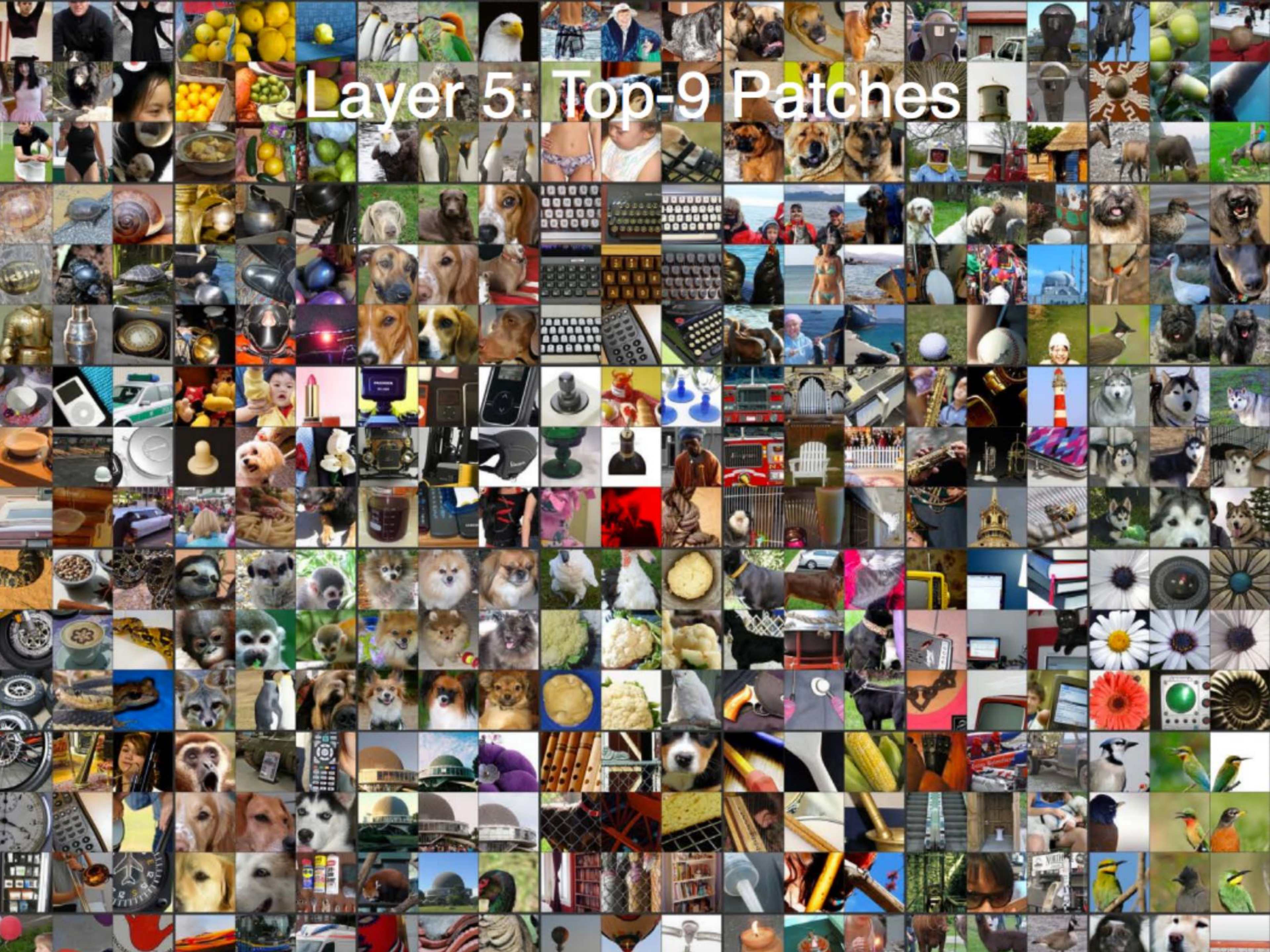
Layer 3: Top-9 Patches



Layer 4: Top-9 Patches



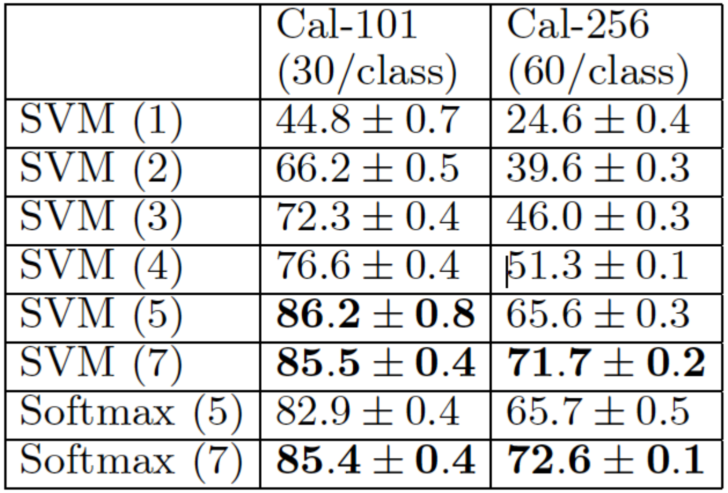
Layer 5: Top-9 Patches



Feature Analysis

- A well-trained ConvNet is an excellent **feature extractor**.

- Chop the network at desired layer and use the output as a feature representation to train an SVM on some other dataset (Zeiler-Fergus 2013):



	Cal-101	Cal-256
	(30/class)	(60/class)
SVM (1)	44.8 ± 0.7	24.6 ± 0.4
SVM (2)	66.2 ± 0.5	39.6 ± 0.3
SVM (3)	72.3 ± 0.4	46.0 ± 0.3
SVM (4)	76.6 ± 0.4	51.3 ± 0.1
SVM (5)	**86.2 ± 0.8**	65.6 ± 0.3
SVM (7)	**85.5 ± 0.4**	**71.7 ± 0.2**
Softmax (5)	82.9 ± 0.4	65.7 ± 0.5
Softmax (7)	**85.4 ± 0.4**	**72.6 ± 0.1**

- Improve further by taking a pre-trained ConvNet and re-training it on a different dataset (Fine tuning).

Other Success Stories of Deep Learning

Today deep learning, in its several manifestations, is being applied in a variety of different domains besides computer vision, such as:

- Speech recognition
- Optical character recognition
- Natural language processing
- Autonomous driving
- Game playing (e.g., Google’s AlphaGo)
- …

References

- http://neuralnetworksanddeeplearning.com
- http://deeplearning.stanford.edu/tutorial/
- http://www.deeplearningbook.org/
- http://deeplearning.net/

Platforms:

- Theano
- Torch
- TensorFlow
- …