COMPUTATIONAL INTELLIGENCE

Deep Learning Strategies and Networks

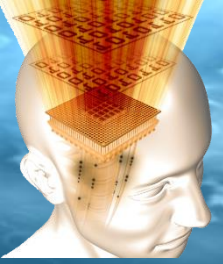
AGH

Adrian Horzyk

What is deep learning?

Deep learning (also known as hierarchical learning) is a class of machine learning algorithms and learning strategies that:

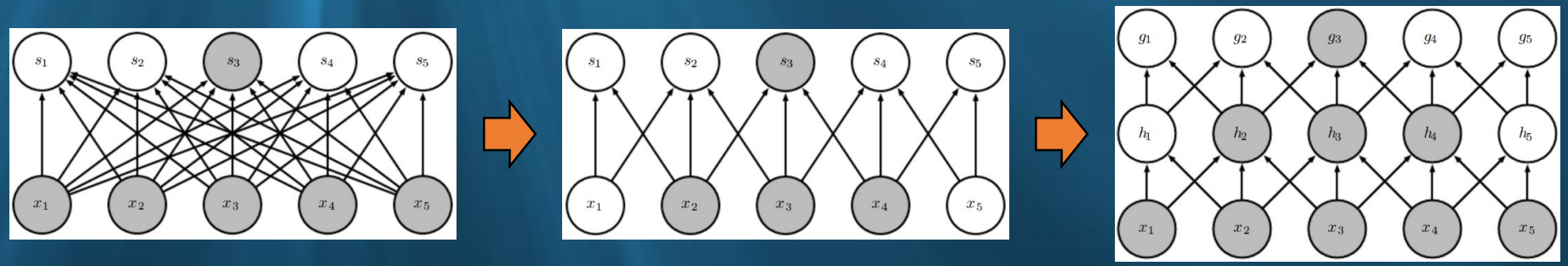
- ✓ Develop hierarchical structure and representation of primary and secondary (derived) features, representing different levels of abstraction.
- ✓ Use a cascade of many layers of neurons (or other processing units) of various kinds for gradual feature extraction and their transformation in order to achieve a hierarchy of secondary, derived features which can led to better final results of such constructed neural network. In this way, they try to determine higher level features which are derived from lower level features.
- ✓ Apply various supervised and unsupervised learning strategies for various layers,
- ✓ Gradually upgrade and develop a structure until significant improvement in performance is achieved.



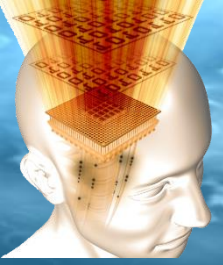
Deep learning strategies

Deep learning strategies assume the ability to:

- update only a selected part of neurons that respond best to the given input data, so the other neurons and their parameters (e.g. weights, thresholds) are not updated,
- avoid connecting all neurons between successive layers, so we do not use all-to-all connection strategy known and commonly used in MLP and other networks, but we try to allow neurons to specialize in recognizing of subpatterns that can be extracted from the limited subsets of inputs,



- create connections between various layers and subnetworks, not only between successive layers
- use many subnetworks that can be connected in different ways in order to allow neurons from these subnetworks to specialize in defining or recognizing of a limited subsets of features or subpatterns,
- let neurons specialize and not overlap represented regions and represent the same features or subpatterns.



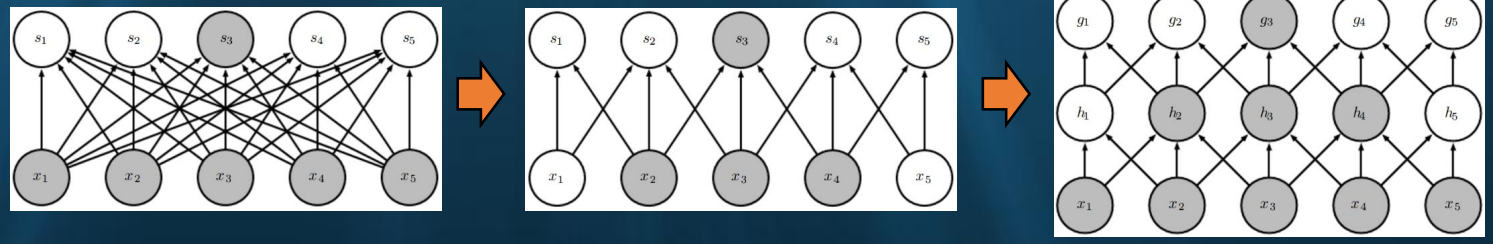
Deep learning strategies

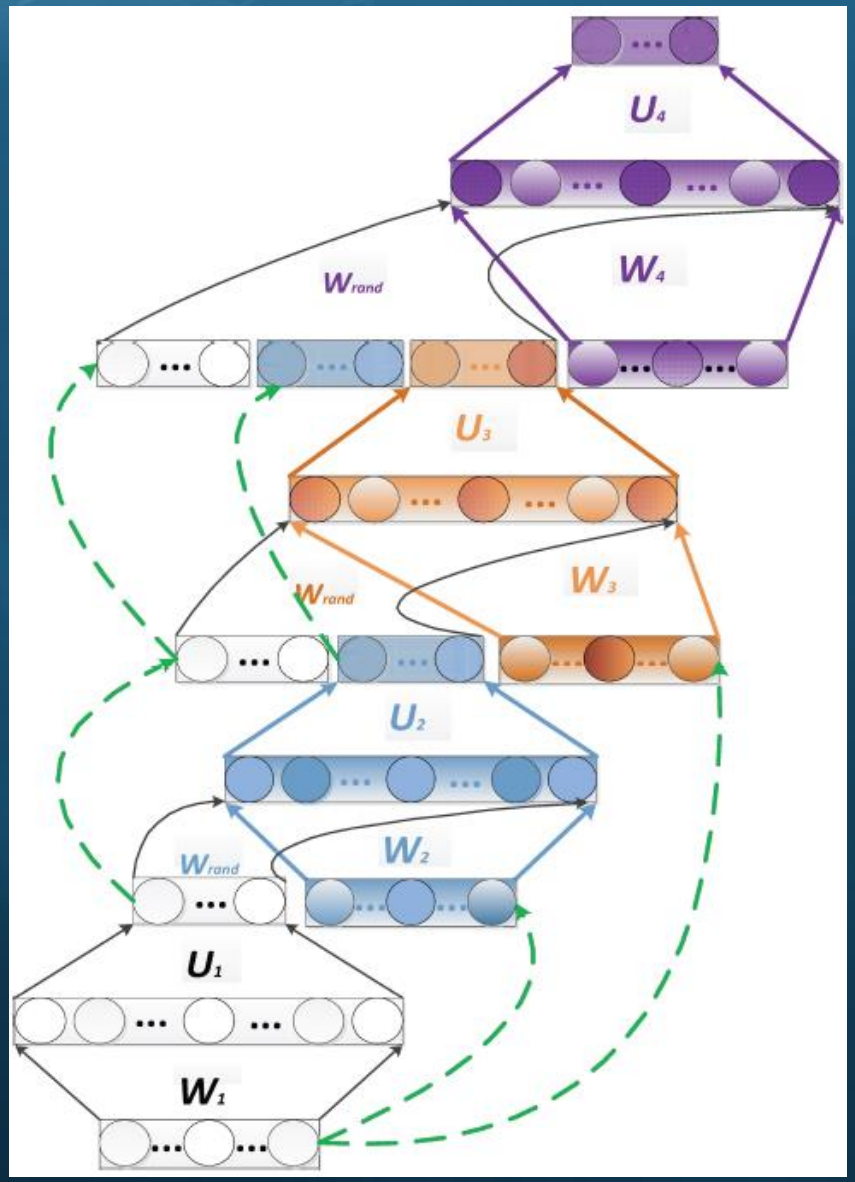
In deep learning architectures, neurons can have input connections coming from different layers, combining the variety of the previously extracted features to compute their outputs.

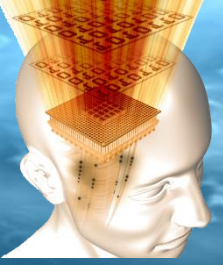
During our laboratory classes:

You may try to use this strategy instead of the classic MLP all-to-all connections and compare achieved training results.

You can use it together with limited number or connections between neurons in the successive layers.



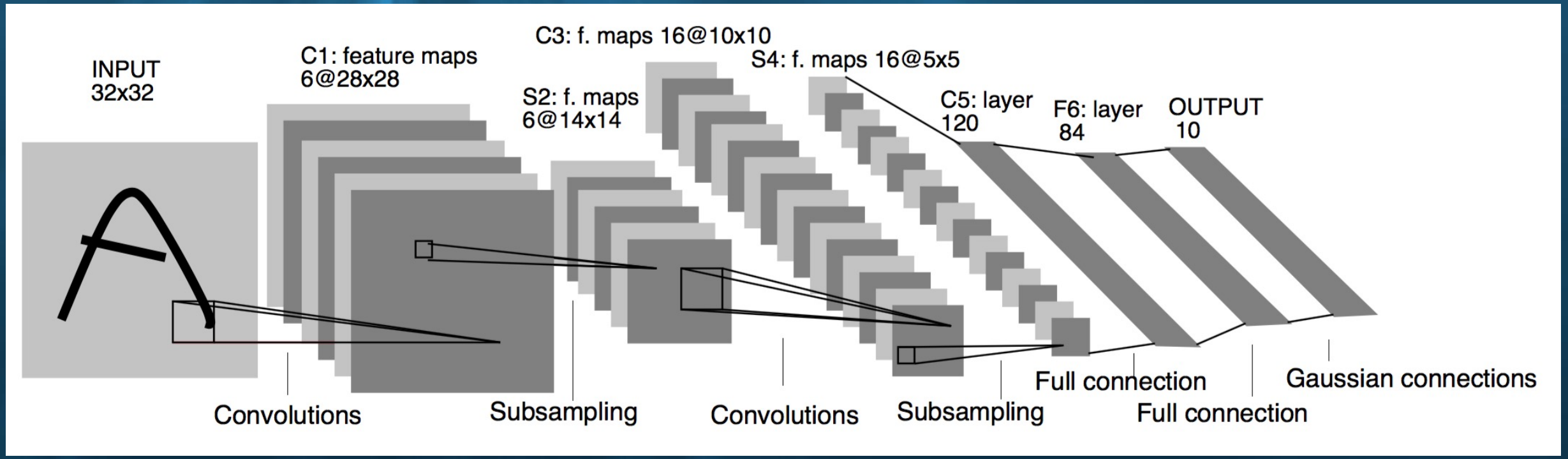


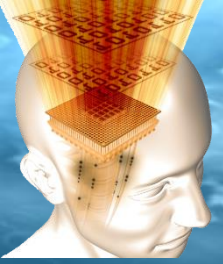


Variety of Deep Learning Architectures

Deep learning architectures can consist of many subnetworks and many layers of different kinds:

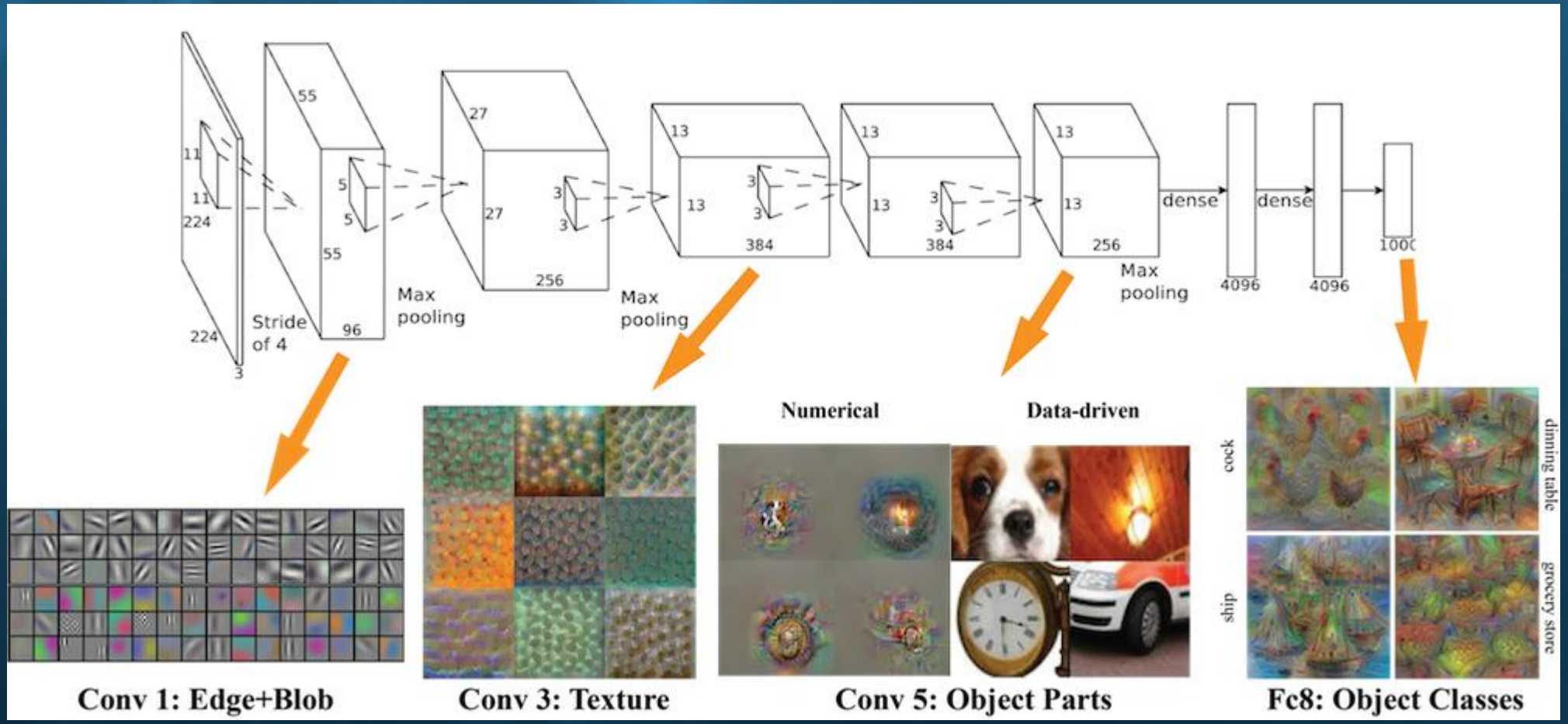
- Subsampling layers are combined with convolutional layers.
- In each layer we can distinguish many subnetworks of the same kind.

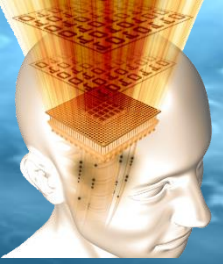




Variety of Deep Learning Architectures

Deep learning architectures can consist of many subnetworks specialized in classifying or recognizing special features, processing special kinds of input data or pooling data from previous layer computing e.g. maxima:

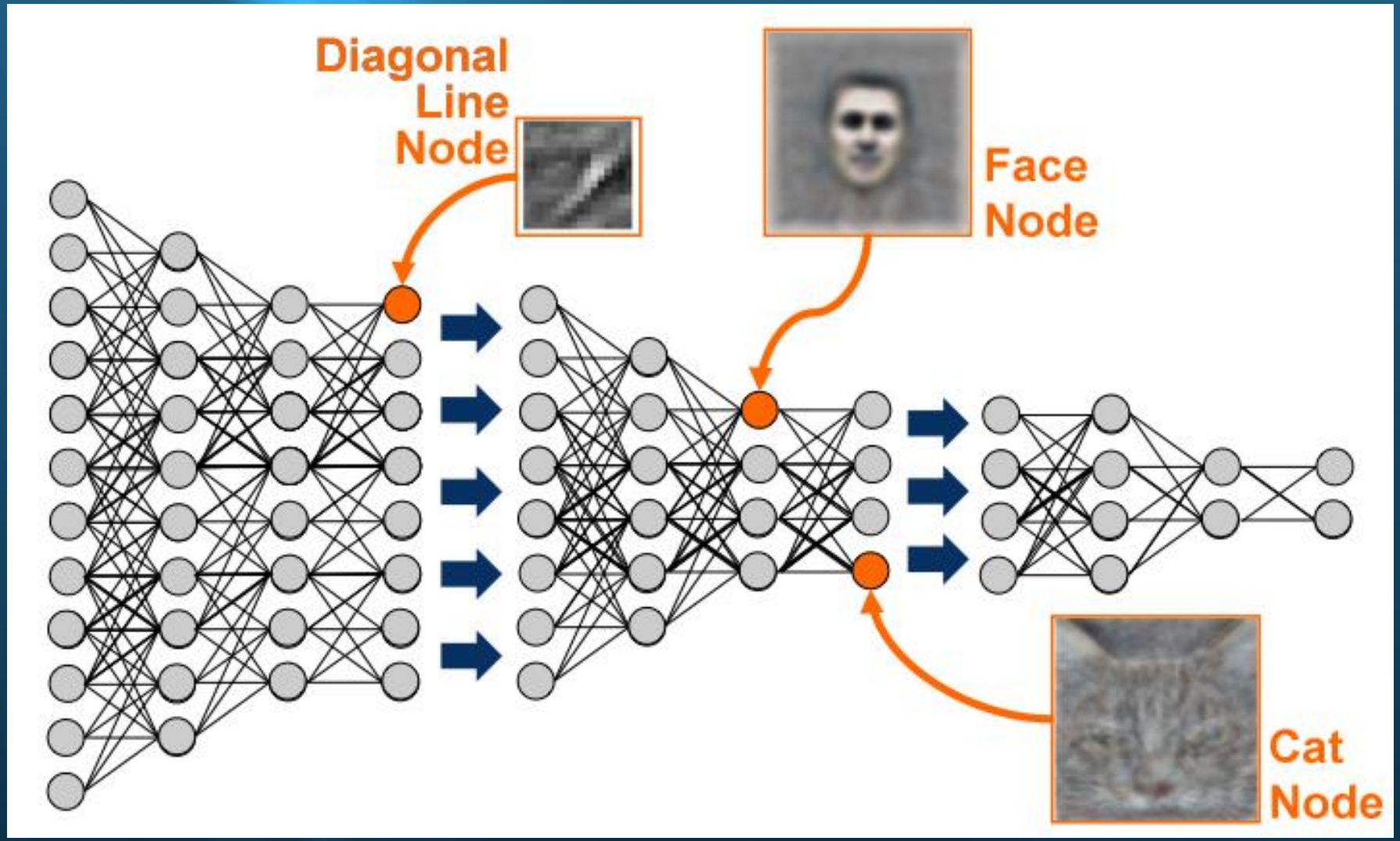


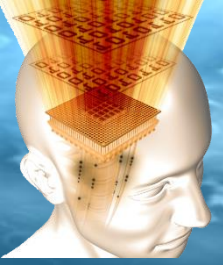


Variety of Deep Learning Architectures

An important part of each deep learning architecture is always a feature extraction process.

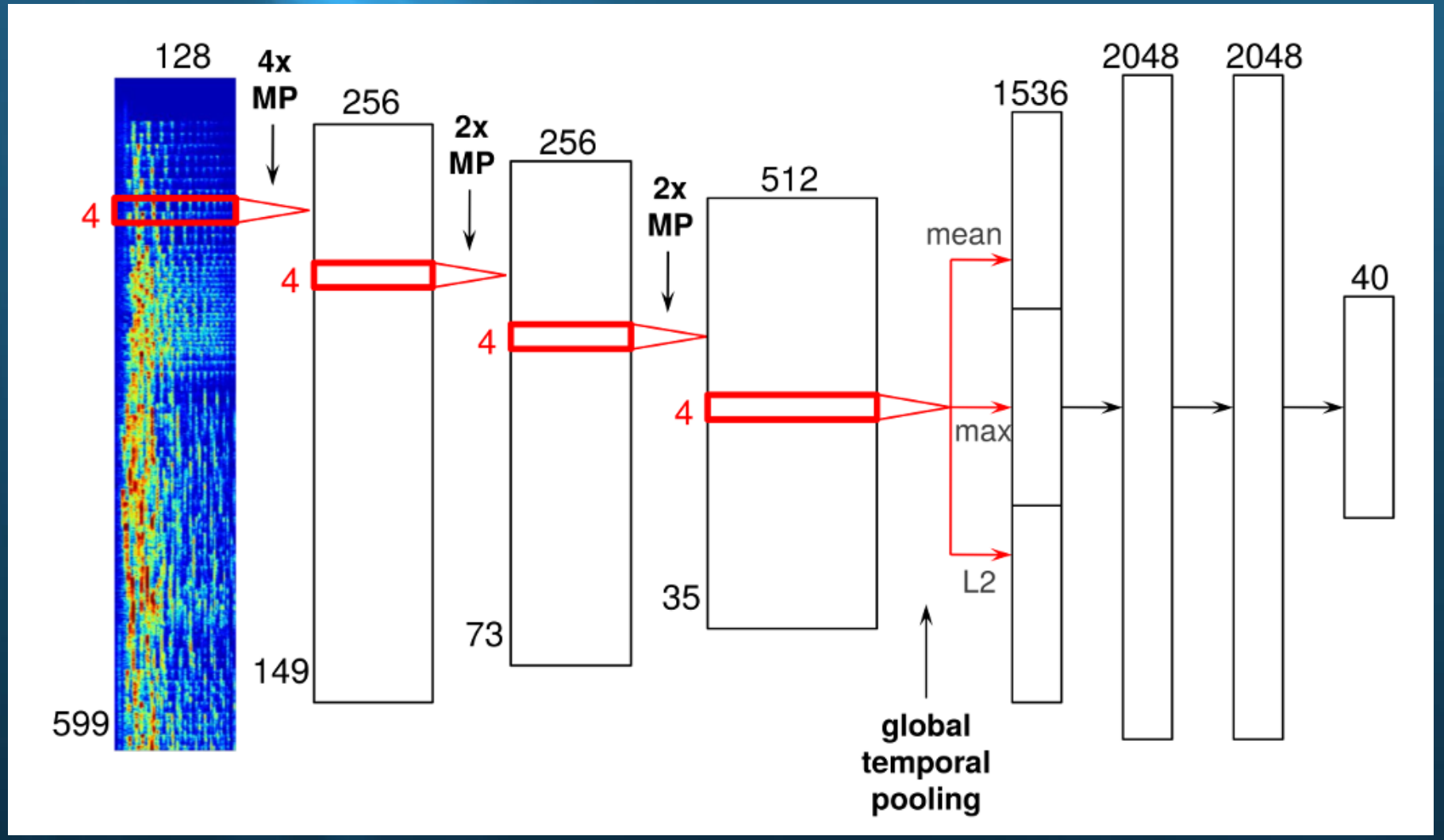
Sometimes we can point to a specific neuron that represents a specific feature.

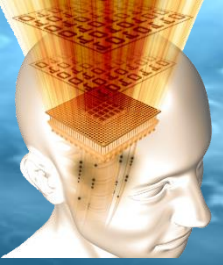




Variety of Deep Learning Architectures

In other cases, we try to pool, combine, or select data using maxima, average numbers, weighted sums, filters etc.



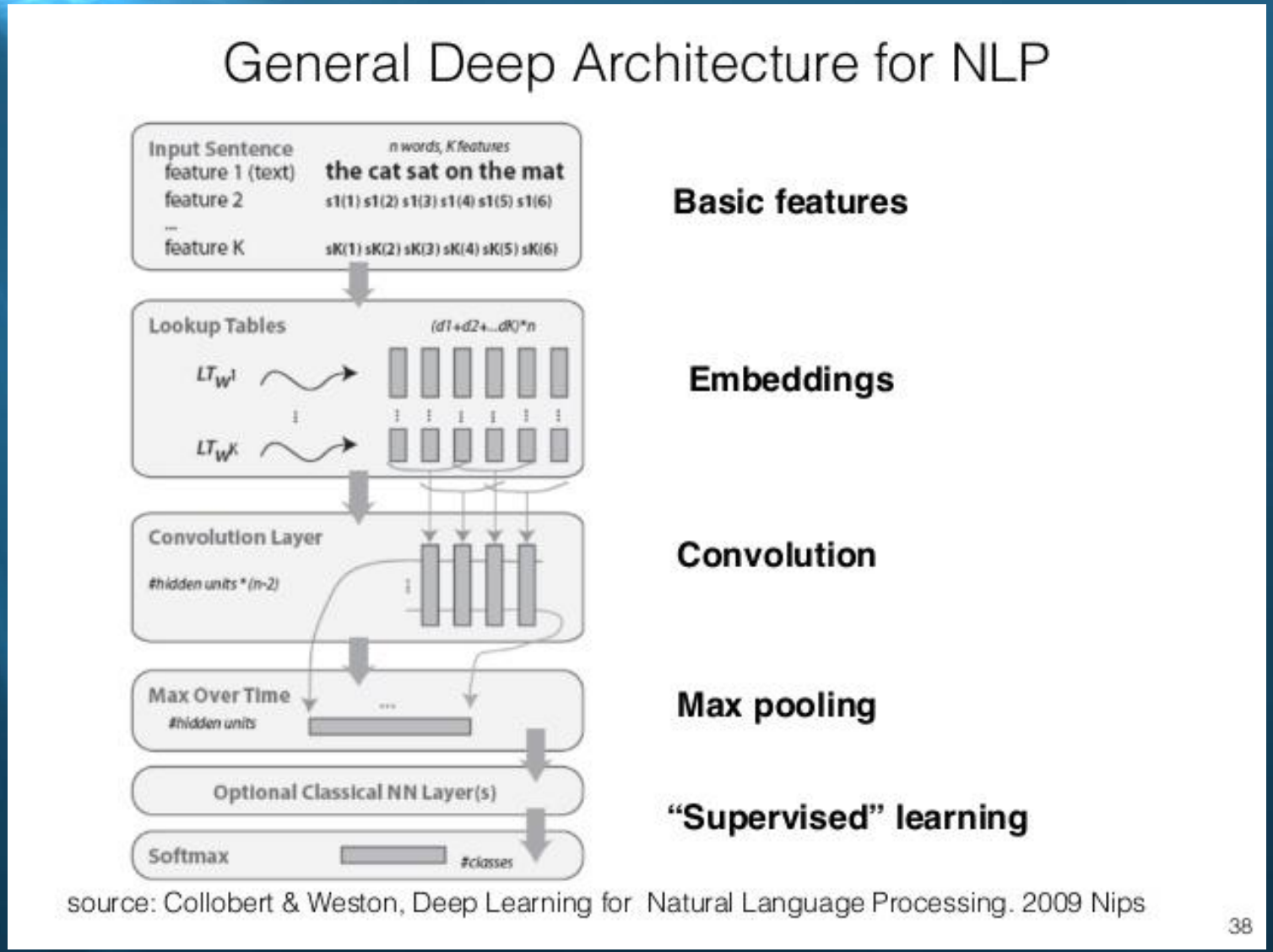


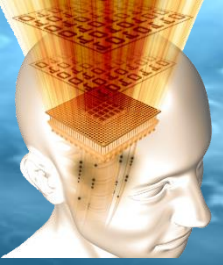
Variety of Deep Learning Architectures

Deep learning architectures try to divide the neural processing into a few phases, where some basic, secondary and derived features are recognized.

They can also include convolutional and pooling layers.

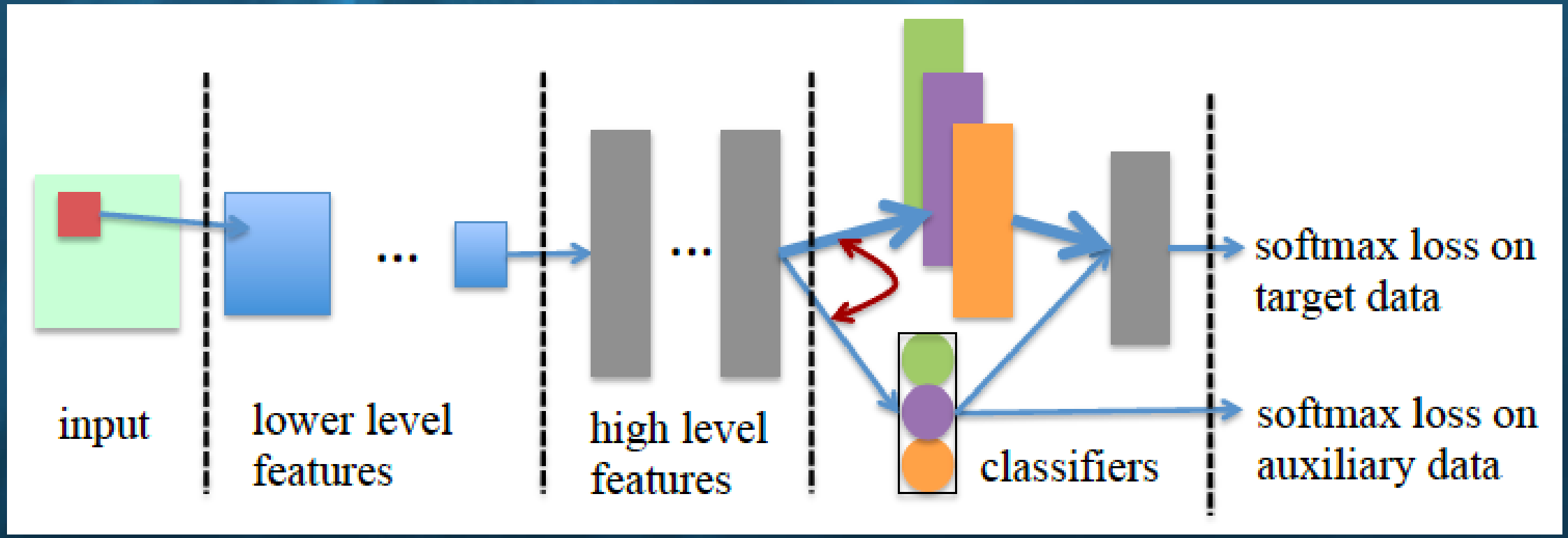
Usually last layer(s) is(are) trained using supervised learning algorithms, like backpropagation, gradient descent, in order to collect, finally process, or fine-tune output results.

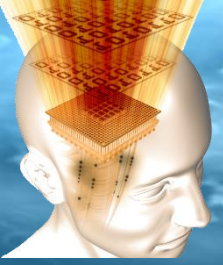




Variety of Deep Learning Architectures

Deep learning architectures usually try to extract valuable features at first, and then try to use other subnetworks that can classify or cluster them, or use them for regression or approximation.
Finally, the last layer filters the best results (using functions extracting minima and maxima)
or does the final approximation according to the known target classes during supervised learning:

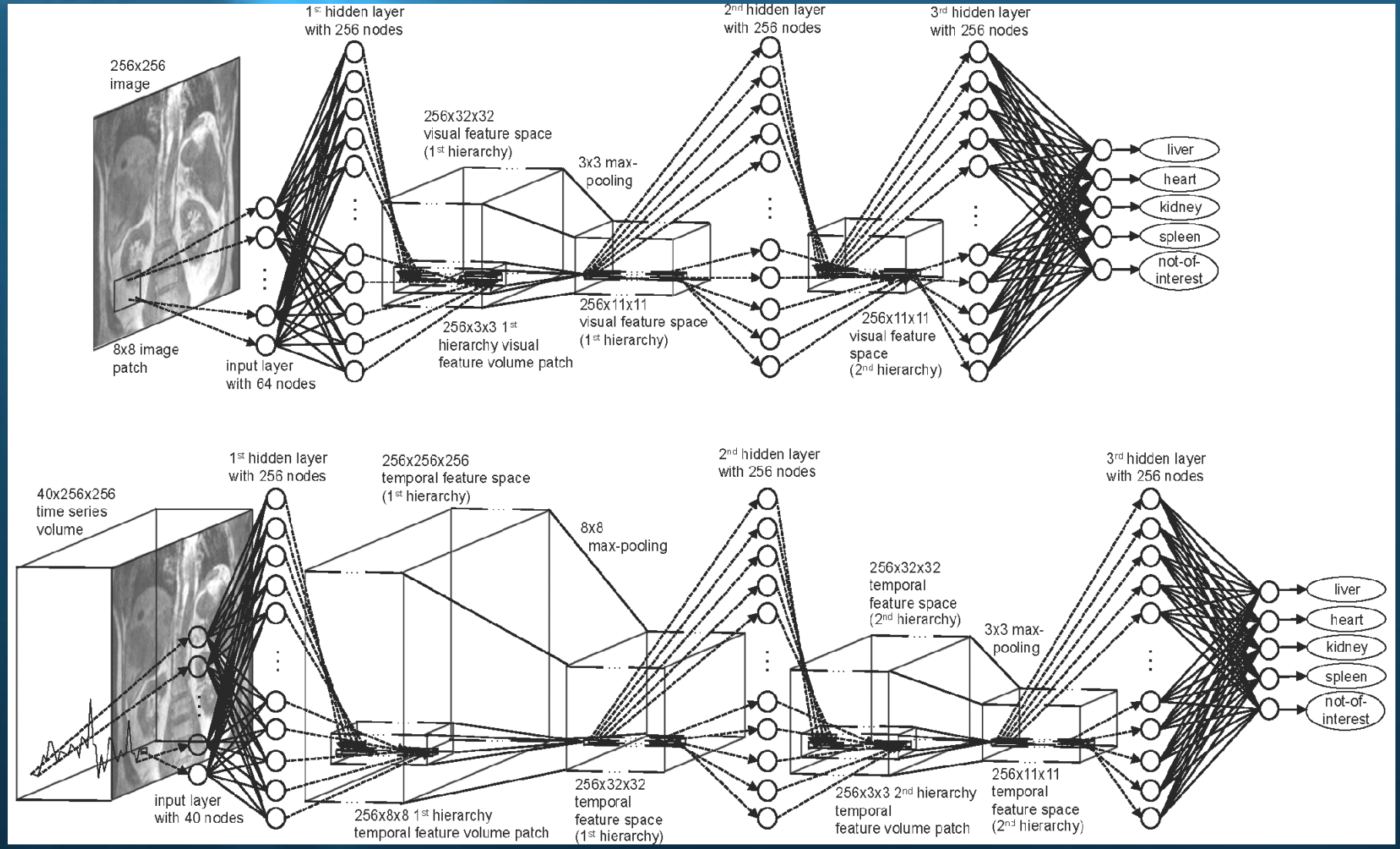


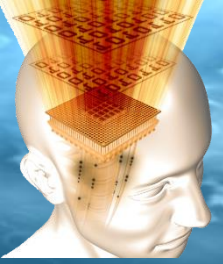


Variety of Deep Learning Architectures

Here we can see the deep learning architecture used for recognizing of human organs:

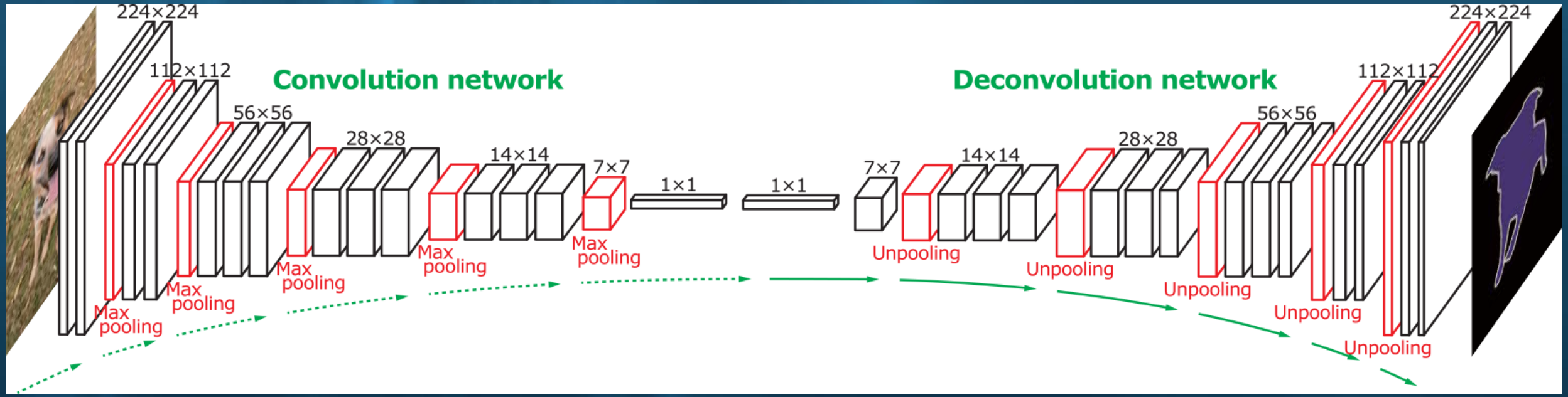
- Liver
- Heart
- Kidney
- Spleen
- and others





Variety of Deep Learning Architectures

We can also use some deep learning architectures for defining classes in order to compare them or for reconstructions of generalized objects:



Summary

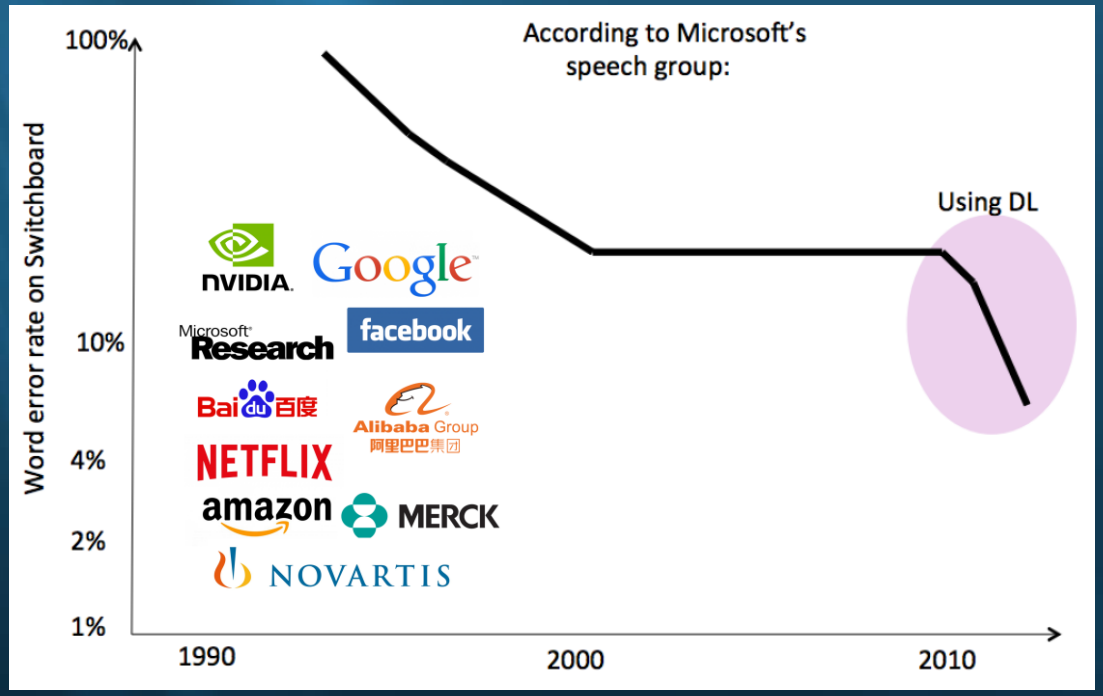
Deep learning algorithms supply us with:

- ✓ the ability to adapt hierarchical structures of cascade layers and subnetworks,
- ✓ representation of primary and derived, higher level features,
- ✓ variety of supervised and unsupervised learning strategies for various layers,
- ✓ gradual development of a structure and gradual learning of neurons or units in the subsequent layers,
- ✓ updating only a selected part of neurons with the best answers or the most differing features,
- ✓ the ability to connect neurons between various and not only successive layers,
- ✓ different levels of abstraction thanks to division of processing between layers and subnetworks.

Deep: Improvement of Learning

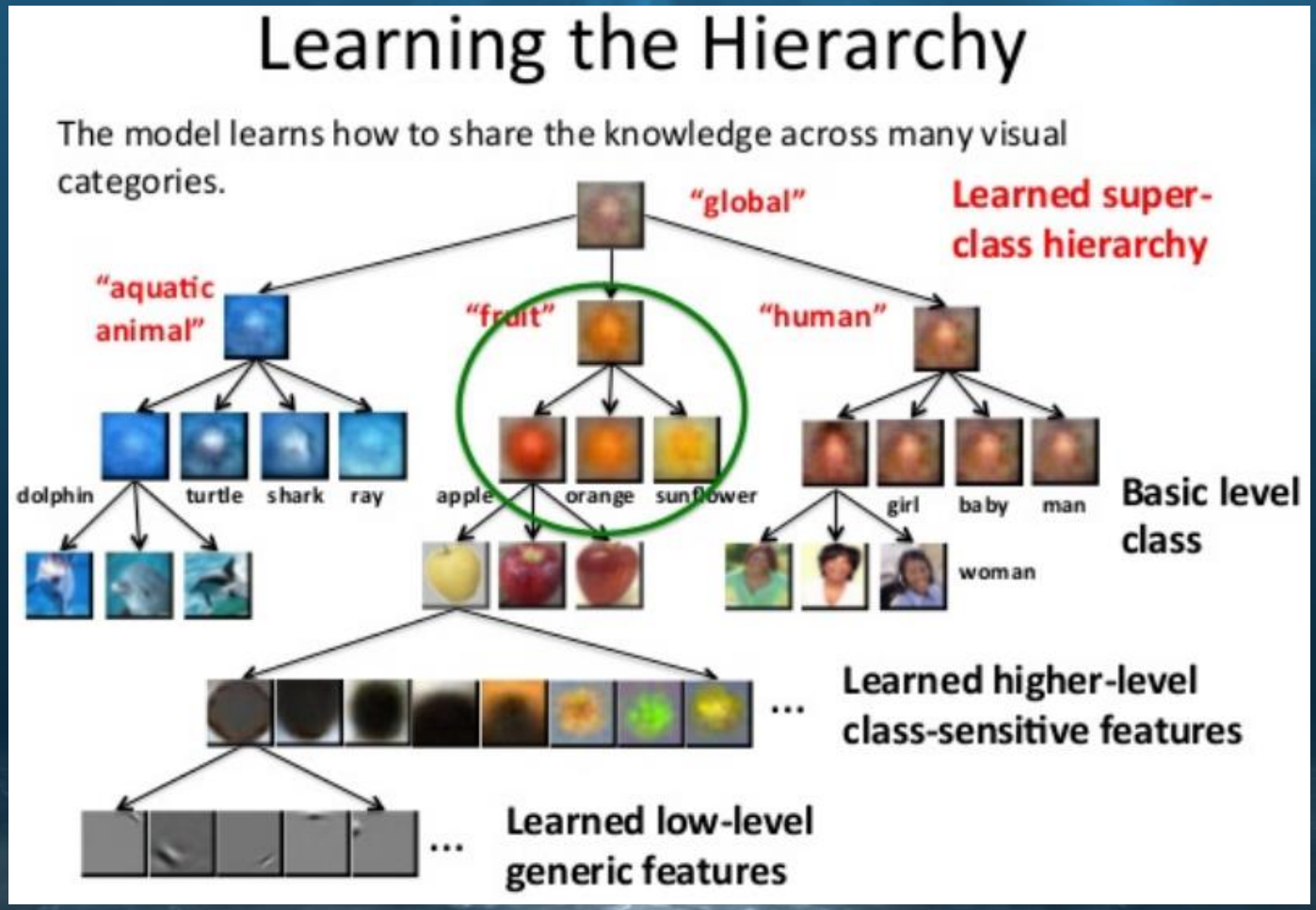
Deep learning algorithms, networks, and strategies usually improve learning outcomes in comparison to other learning techniques in many areas:

- Computer vision and pattern recognition
- Classification and clusterization
- Data mining and information search
- Speech recognition
- Natural language analysis
- Decision and recommendation systems



Deep: Improvement of Learning

Deep learning allows the network to learn various categories and a hierarchy of features to improve learning outcomes in comparison to other learning methods:

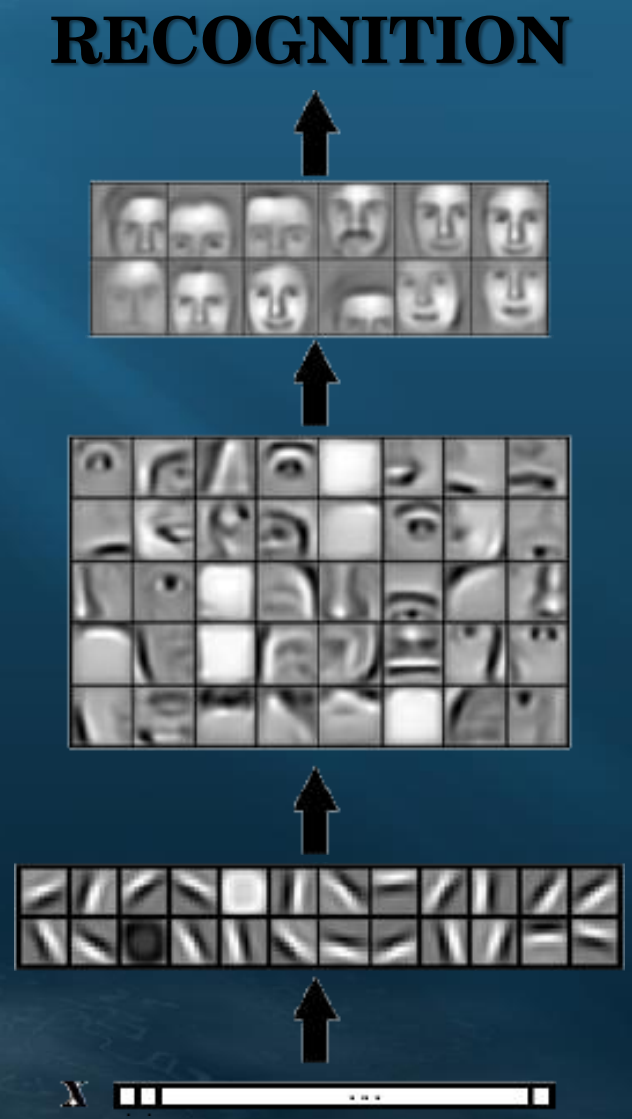


Automatic Extraction of Features

Deep networks improve learning outcomes thanks to the gradual process of features extraction from the raw data.

We look for features which are:

- ✓ discriminative
- ✓ robust
- ✓ invariant



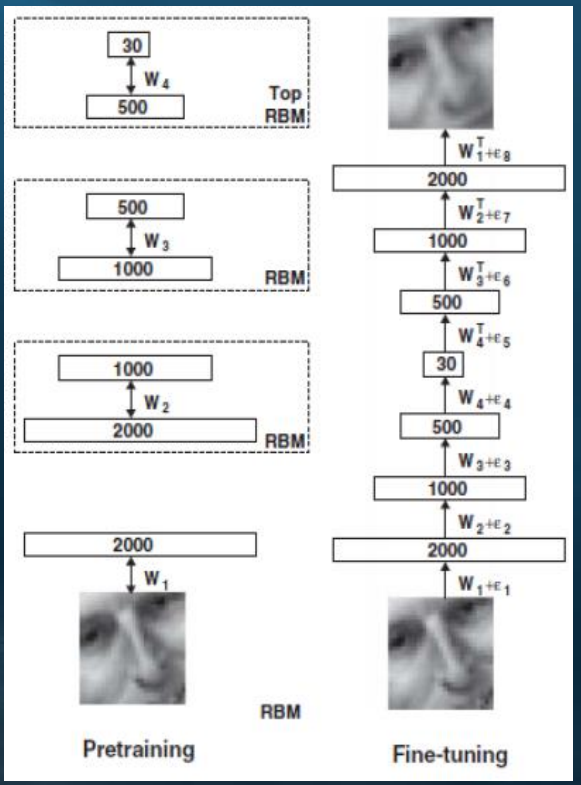
- Lee et al. Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations, Int. Conf. ICML 2009

Vanishing Gradient Problem

When using gradient-based learning strategies for many layers (e.g. MLPs) we usually come across the **problem of vanishing gradients**, because derivatives are always in range of [0, 1], so their multiple multiplications lead to very small numbers producing very small changes of weights in the neuron layers that are far away from the output of the MLP network.

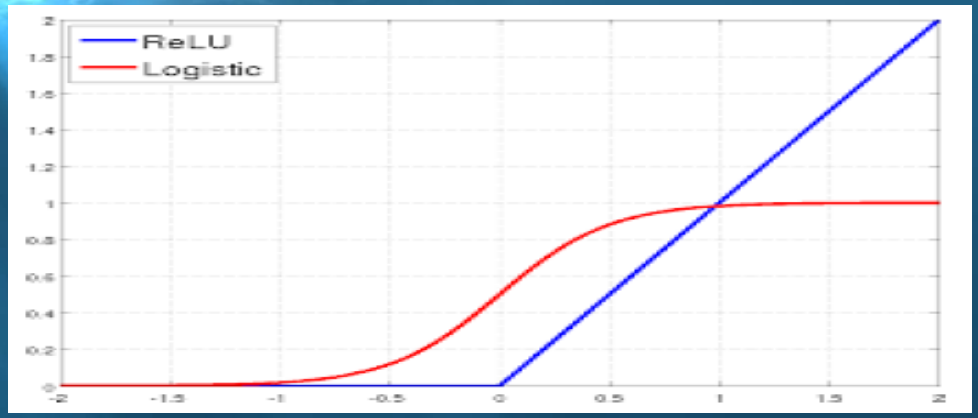
This problem can be solved using pre-training and fine-tuning strategy, which first trains the model layer after layer in the unsupervised way (e.g. using deep auto-encoder) and then we use backpropagation algorithm to fine-tune the network.

Hinton, Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. Science 2006



Rectified Linear Units (ReLU)

We can also use Rectified Linear Units (ReLU) to eliminate the problem of vanishing gradients.



ReLU units are defined as: f(x) = max(0, x) instead of using the logistic function.

The strategy using ReLU units is based on training of robust features thanks to sparse (less frequent) activations of these units.

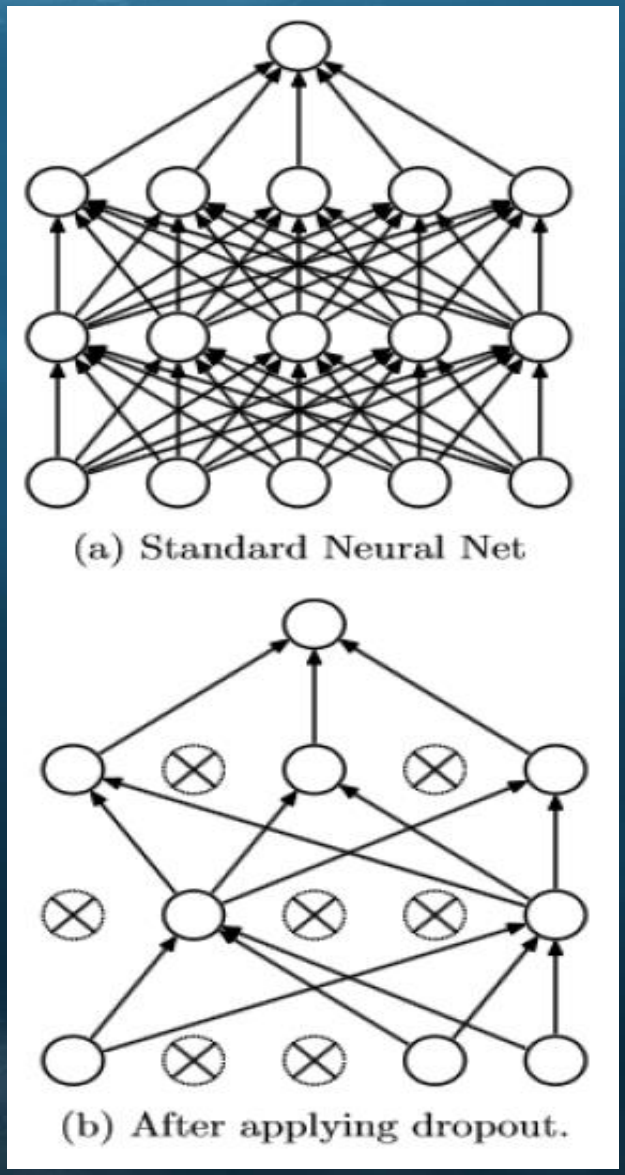
The other outcome is that the training process is also typically faster.

Nair, Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. ICML 2010

Dropout Regularization Technique

We can also use regularization techniques as dropout.

This training strategy selects only these neurons which are already the best adapted in various layers and performs training only to these neurons and their weights. This technique prevents neural networks from overfitting and also speeds up training. It also prevents other neurons from spoiling their weights parameters which can be good for other training patterns.

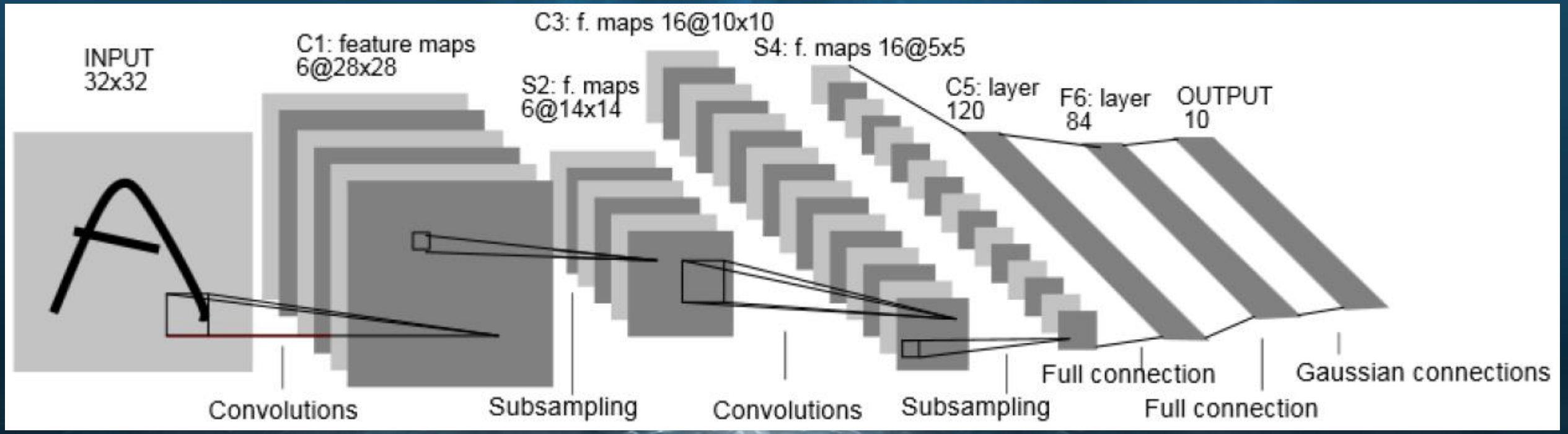


Srivastava et al. Dropout: A simple way to prevent neural networks from overfitting. JMLR 2014

Deep Convolutional Networks

Deep Convolutional Networks can gradually filter various parts of training data and **sharpen important features** for the following discrimination process used for recognition or classification of patterns.

LeCun et al. Gradient-Based Learning Applied to Document Recognition. Proc. of IEEE 1998

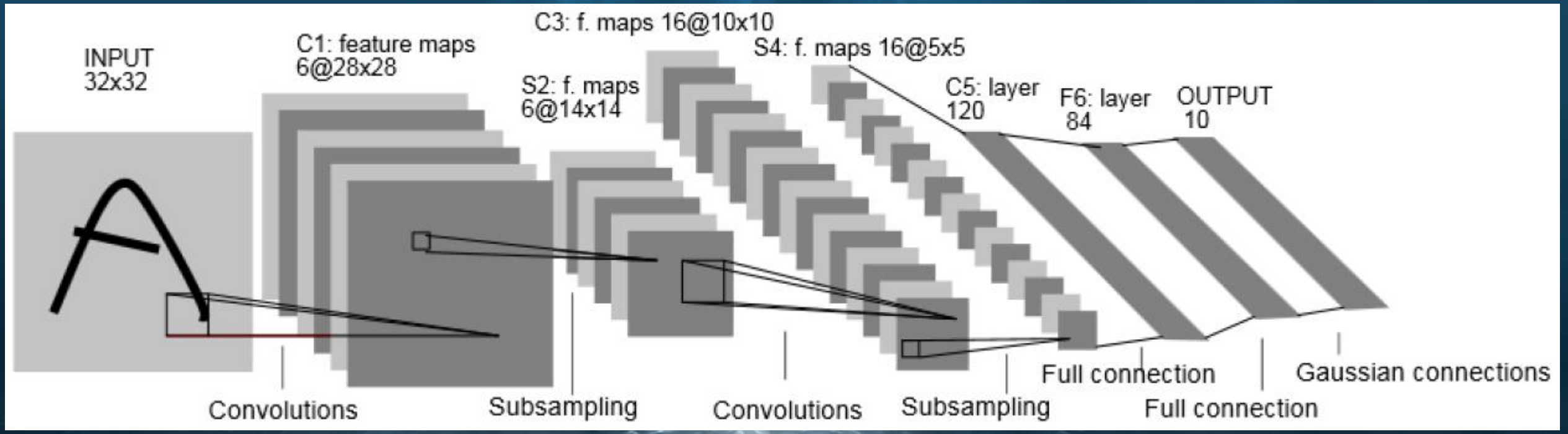


Deep Convolutional Networks

In each convolution we can distinguish:

- The number of parameters in a layer: No of channels * No of filters * filter width * filter height
- The number of hidden units in layer: No of filters * pattern width * pattern height

LeCun et al. Gradient-Based Learning Applied to Document Recognition. Proc. of IEEE 1998

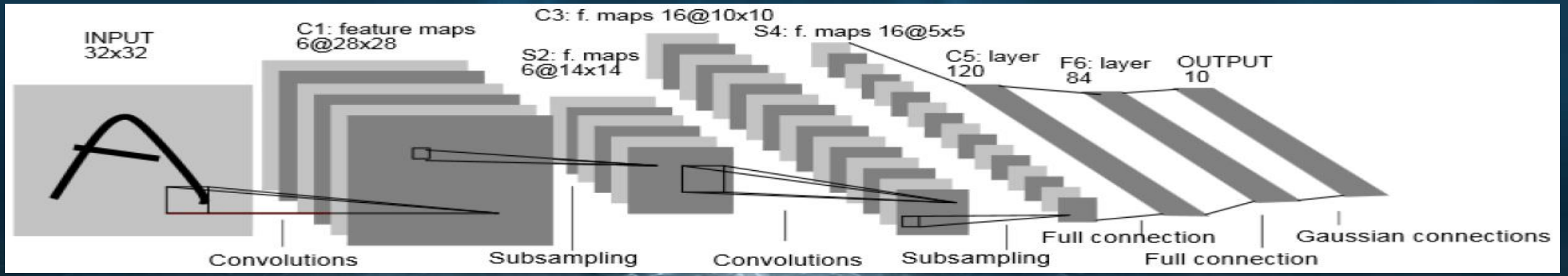


Pooling and MaxPooling

Pooling layer is used to progressively reduce the spatial size of the representation to reduce the number of features and the computational complexity of the network.

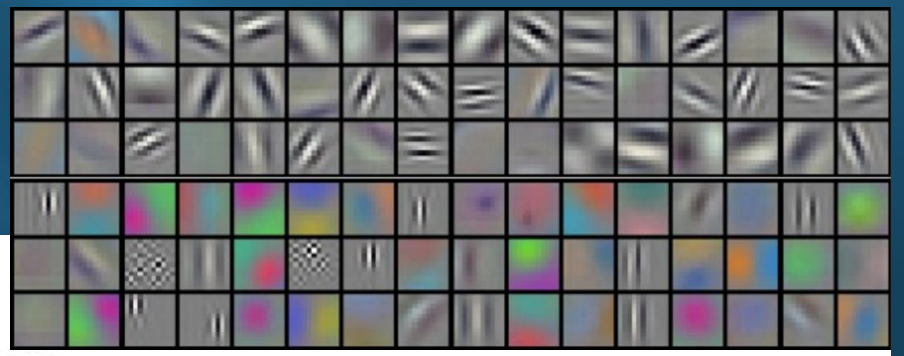
The mostly used is the **MaxPool layer** in many convolutional neural networks that traverses the 2x2 filters over the entire matrix to pick the largest values from the window to be included in the next representation map. The main reason for using of the pooling layers is to prevent the model from overfitting. Sometimes there is used the dropout layer which succeed the pool.

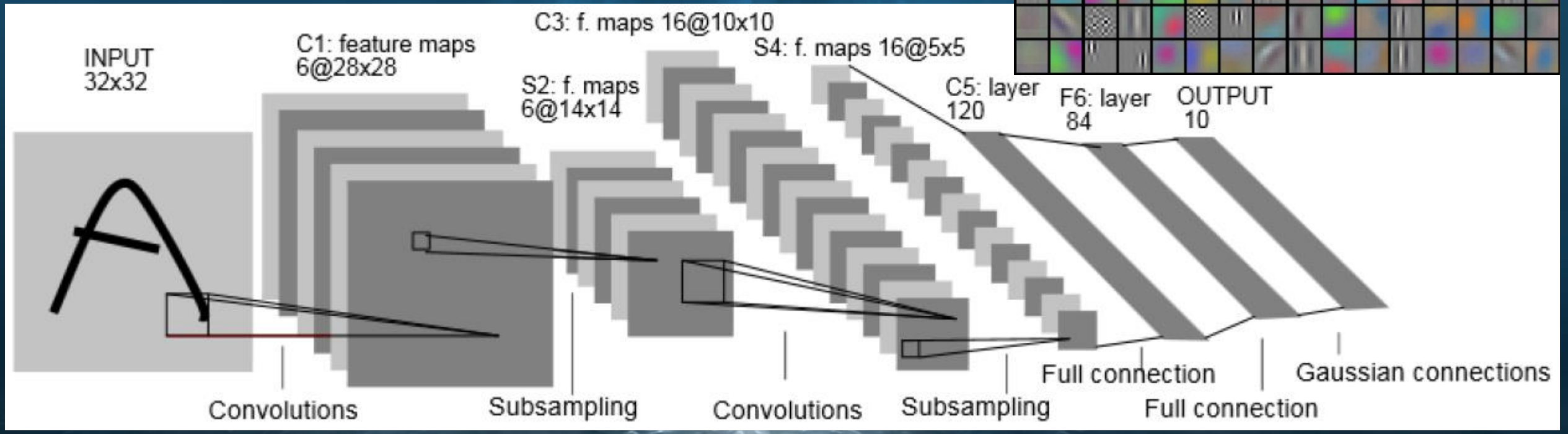
Be careful in the use of the pooling layers, particularly in vision tasks, because it might cause to lose the location sensitivity in the model while it would help significantly reduce its complexity.



Convolutions and Subsampling

Convolutions allow for extraction of simple features at the beginning layers of the network, e.g. edges of some orientation or a blotch (spot) of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. We have an entire set of filters in each convolutional layer (e.g. 8 filters), and each of them will produce a separate 2D activation map. We stack these activation maps along the depth dimension and produce the output volume.





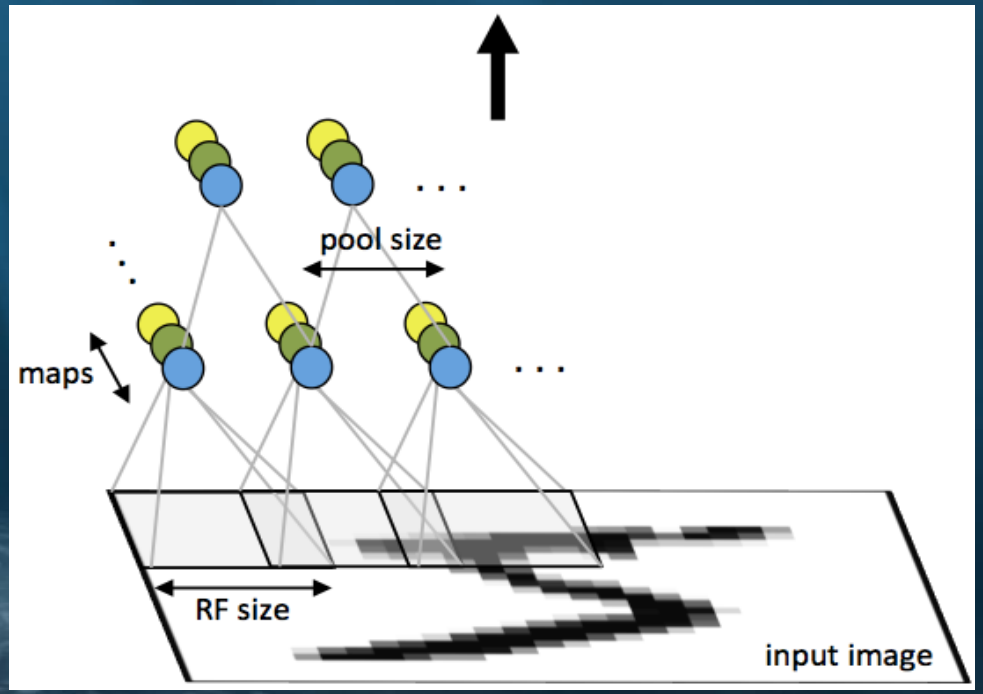
Convolutional Neural Networks

A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers (typically with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network (e.g. MLPs), SVM, SoftMax etc.
A Deep CNN consists of more layers. The CNNs are easier to train and have many fewer parameters (using the same weights) than typical neural networks with regards to the number of convolutional layers and their size.

This kind of networks are naturally suited to perform computations on 2D structures (images).

In the figure, the first layer of a convolutional neural network with pooling. Units of the same color have tied weights and units of different color represent different filter maps:

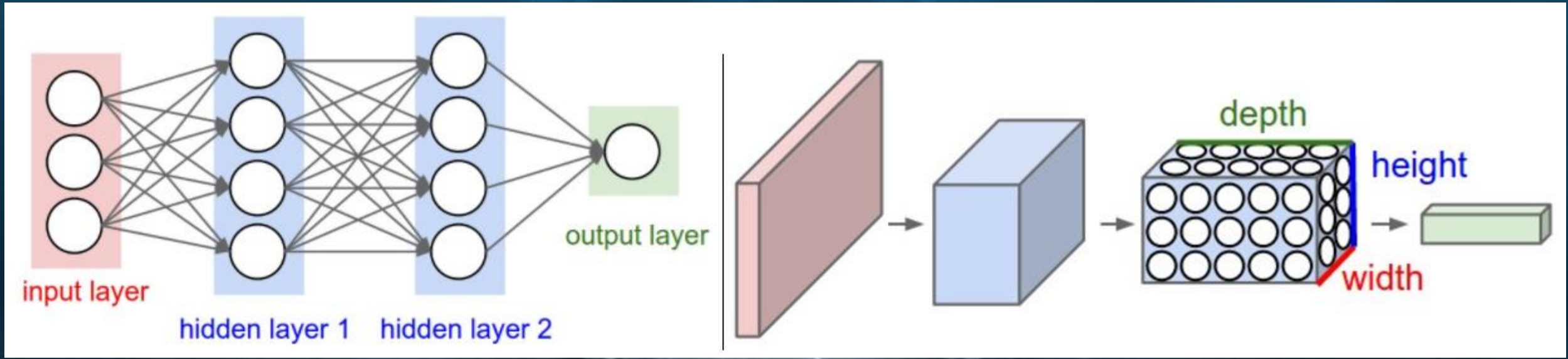
http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/



Convolutional Neural Networks

Convolutional Neural Networks arrange computational units („neurons”) in 3D: **width, height, and depth**. The neurons in each layer are only connected to a small region of the previous layer instead of all-to-all (fully-connected) met in typical artificial neural networks. Moreover, CNNs (e.g. CIFAR-10) reduces full images to a single output vector of class scores, arranged along the depth dimension as shown in the figure below.

The figure presents the comparison of typical and deep convolutional architectures:



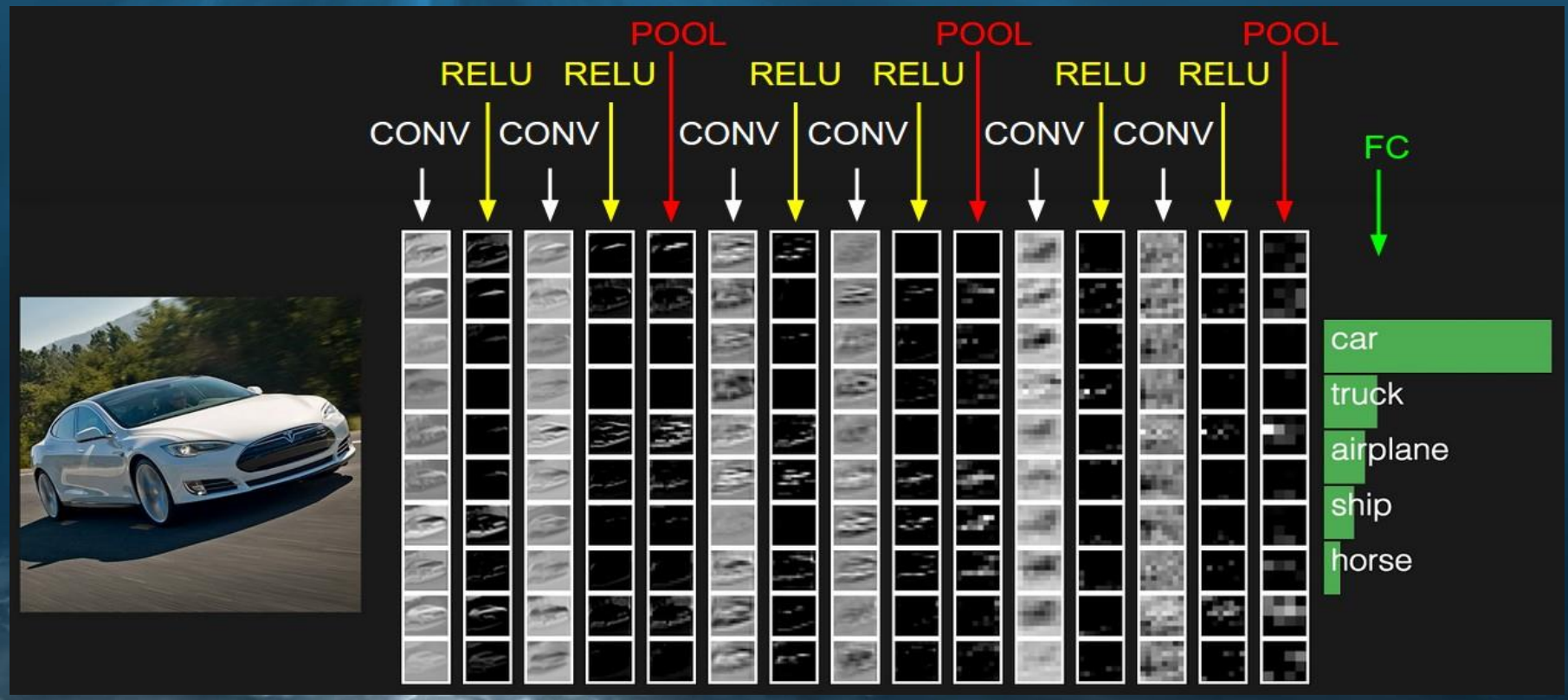
Convolutional Neural Networks

A Convolutional Neural Network is usually a sequence of layers and every layer transforms one volume of activations to another through a differentiable function (in order to be able to use backpropagation to fine-tune network parameters).

CNNs (ConvNet) usually consist of three main types of layers:

- **Convolutional Layer** consists of a set of small learnable filters, e.g. [5x5x3]
- **Pooling Layer**,
- **Fully-Connected Layer** implementing an MLP, SVM, or SoftMax network.

[Demo of the ConvNetJS on the CIFAR-10 data.]



Convolutional Neural Networks

Example of a Convolutional Neural Network:

1. Input image [32x32x3] where the third parameter codes colors from R, G, and B channels separately.
2. **Convolutional layer (CONV)** computes the output of neurons that are connected to local regions in the input image, each layer computes a **dot product** between their weights and a small region. This may result in volume such as [32x32x8] if we decide to use 8 convolutional filters.
3. **ReLU layer (RELU)** applies an elementwise activation function (such as the max(0,x) introduced before) thresholding at zero. This layer leaves the size of the volume unchanged [32x32x8].
4. **Pooling layer (POOL)** performs a downsampling operation along the spatial dimension (width x height), resulting in the volume such as [16x16x8]
5. **Fully connected layer** of a selected artificial neural network (FCNN) computes the **class scores** (classification), resulting in volume of size [1x1x5], where each individual output corresponds to one of 5 classes (scores, categories). This layer is fully connected to all outputs of the previous layer and is trained using a gradient descent method.

Computing a Dot Product

A **dot product** (called also a **scalar product**) is an algebraic operation that takes two equal-length sequences of numbers (usually vectors, however matrices can be used as well) and returns a single number that is computed as a sum of products of equivalent values from these two sequences (vectors or matrices):

Suppose, we have two vectors:

$$A = [a_1, a_2, \dots, a_n] \quad \text{and} \quad B = [b_1, b_2, \dots, b_n]$$

The dot product of these two vectors is defined as:

$$A \cdot B = \sum_{i=1}^{n} a_i \cdot b_i$$

Training of Convolutional Layers

We use a modified backpropagation or stochastic gradient descent learning to adapt weights of convolutional layers.

The delta (error) is usually propagated back only to the winner.

The convolutional filter for the j-th slice of the l-th convolutional layer the output is computed:

$$z_j^{l+1} = \sum_{i=1}^{n} w_{j,i}^{l} \cdot a_i^{l}$$

We usually define: Total Error = $\sum$ ½ (target probability – output probability) ^{2}

CNN implementation details can be found in the Liu at all paper
„Implementation of Training Convolutional Neural Networks”

Shared Weights and Biases

Each depth **slice** uses the same weights and bias for all neurons. In practice, every neuron in the volume will compute the gradient for its weights during backpropagation, but these gradients will be added up across each depth slice and only update a single set of weights per slice. Thus, all neurons in a single slice are using the same weight vector. The convolutional layer using this vector computes a convolution of the neuron's weights with the input volume. Because the same set of weights is used it can be treated as an **adaptive filter** convolving the input into the output scalar value.

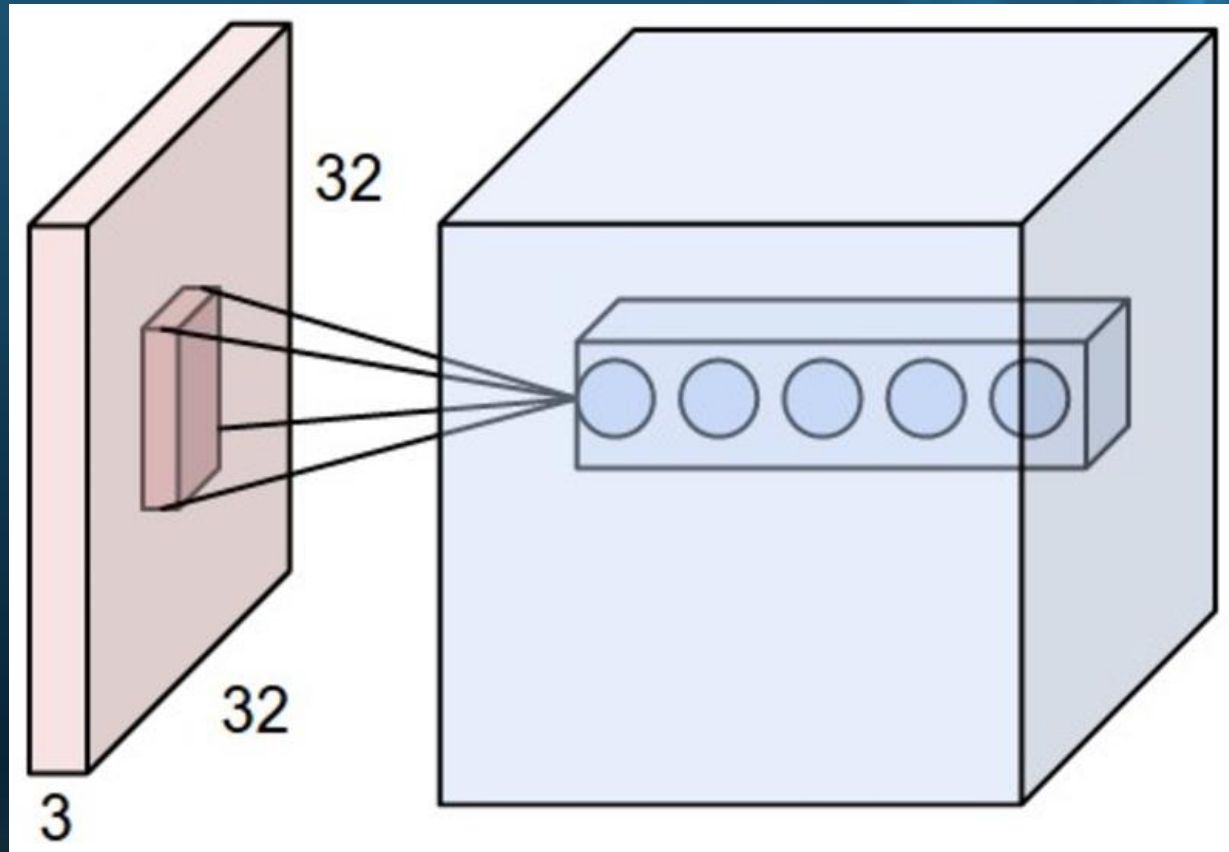
Example of 96 filters [11x11x3] learned by Krizhevsky at al. Each filter is shared by 55x55 neurons in one depth slice.

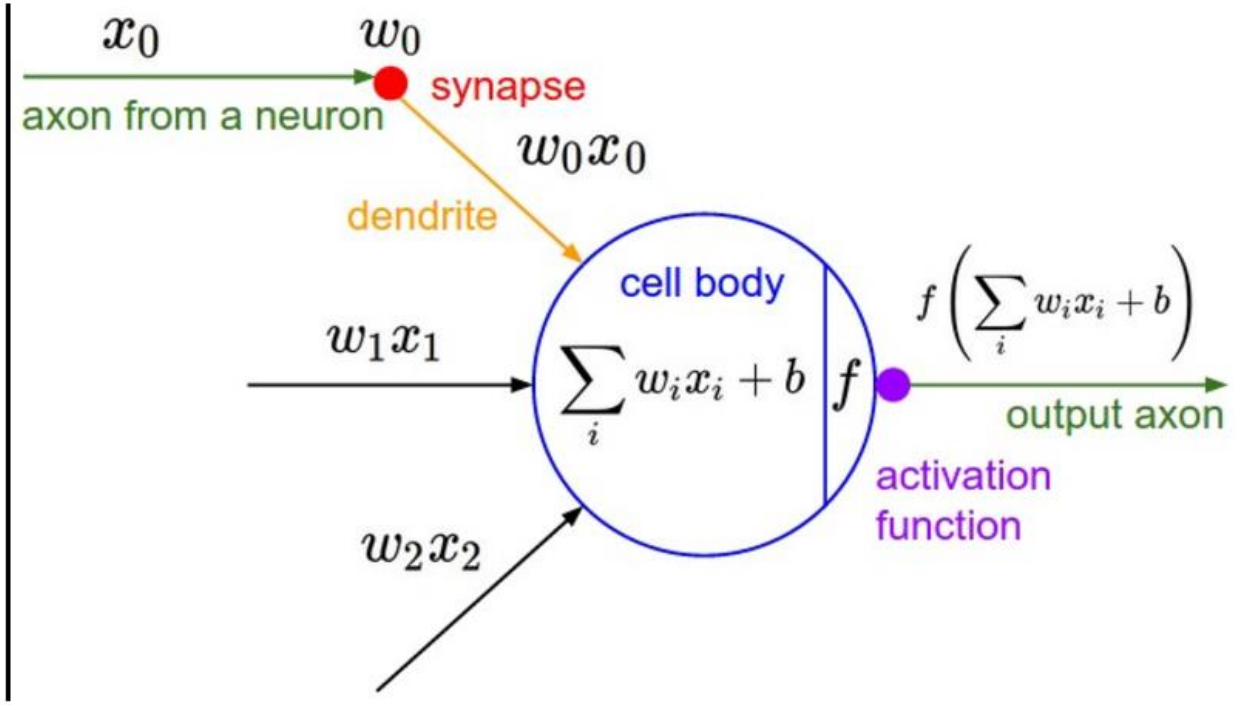
If detecting e.g. vertical line at some location in the image, it should be useful at some other location as well due to the translationally-invariant structure of images.

Therefore, **we do not need to relearn to detect a vertical line at every one** of the 55x55 distinct locations in the convolutional layer output volume.

Convolutional Layers in CNN

In this example, we can notice that there are multiple neurons (5 of them computing a dot product of their weights with the restricted input) along the depth, all connected to the same region in the input volume, where the connectivity is restricted to be local spatially:





Filters

A convolutional layer works as an **adaptive filter**, that allow to set values in such matrices:

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

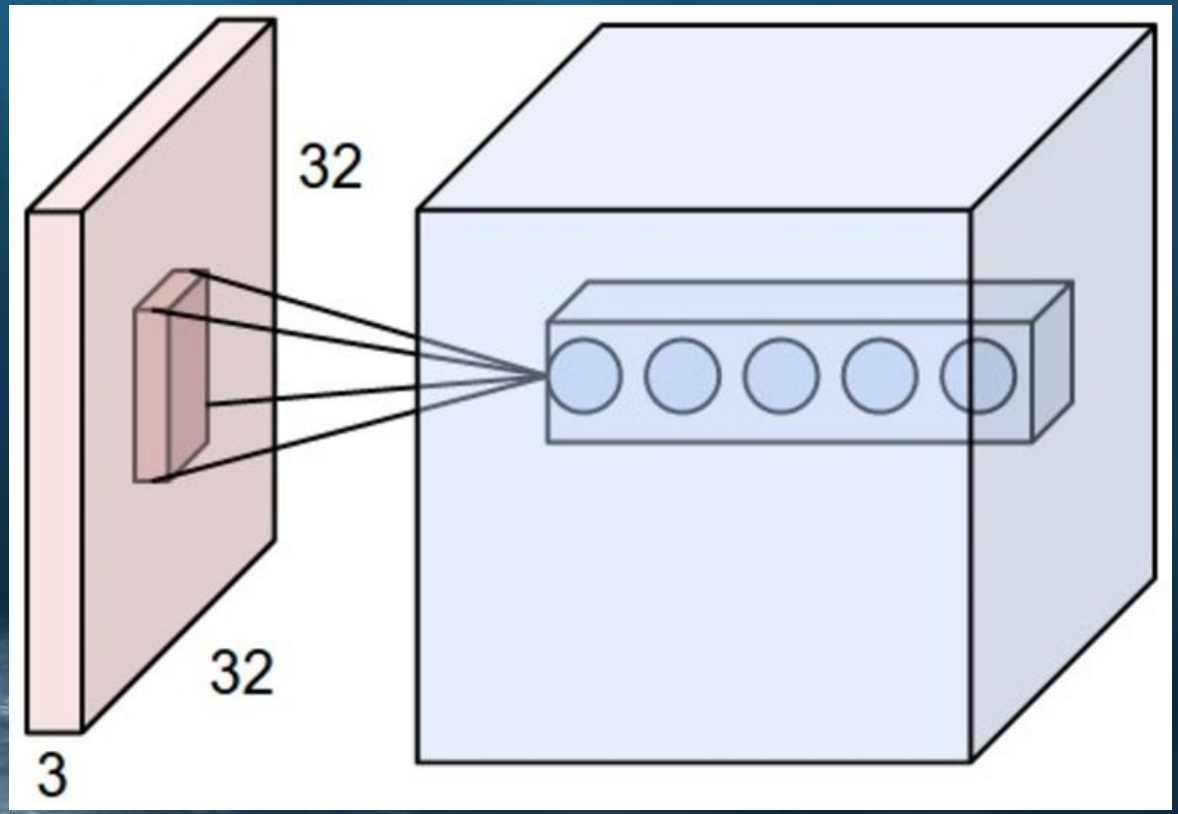
Using the other well-known filters we can convolve an input image as shown on the right:

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Number of Filters in the Convolutional Layers

The depth of the output volume is a hyperparameter that corresponds to the **number of filters** we would like to use. Each filter learns to look for something different in the input volume, e.g. the first convolutional layer takes as an input the raw image, and different neurons along the depth dimension (which form a depth column called also a **fibre**) may activate in presence of various oriented edges or color blobs.

We slide each filter in the input volume defining the **stride** parameter: When the stride is 1 then we move the filters one pixel a time, when 2 then the filters jump 2 pixels at a time as we slide them around. This will always produce smaller output volumes spatially. Sometimes, it will be convenient to **pad** the input volume with zeros around the border. Then the input and output width and height are the same.

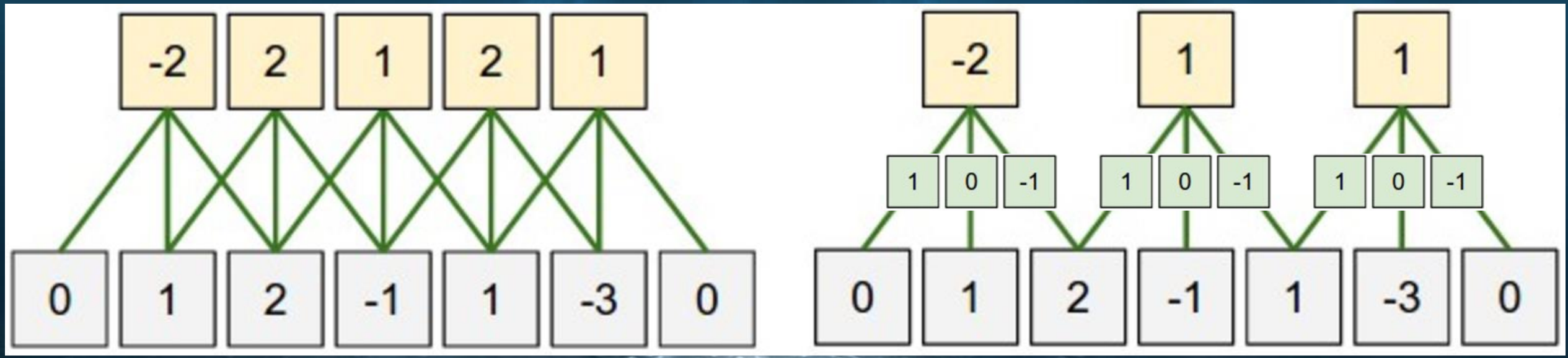


The Output Size Due to the Stride

We can compute the spatial size of the output volume **(W-F+2·P)/S+1** as a function of the **input volume size W**, the **receptive field size** of the convolutional layer neurons **F**, the **stride S**, and the amount of zero **padding P** used on the border:

- If we have 7x7 input and a filter 3x3 with **stride 1** and pad 0, then we get a 5x5 output: (7-3+2·0)/1+1=5
- If we have 7x7 input and a filter 3x3 with **stride 2** and pad 0, then we get a 3x3 output: (7-3+2·0)/2+1=3

The graphical presentation for only a single dimension (width or height)
with the same weights (in the green boxes) shared across all yellow neurons:



EXAMPLE OF CONVOLUTION

In this example, 3 separate tables are used to visualize 3 slice of the 3D input volume [5x5x3].

The input volume is in blue, the weight volumes are in red, and the output volume is in green.

In this convolutional layer we will use the following parameters:

K = 2 (number of filters),
F = 3 (filter size 3x3),
S = 2 (stride),
P = 1 (padding), which makes the outer border of the input volume zero (in grey).

Hence, the output volume size is equal $(5 - 3 + 2 \cdot 1) / 2 + 1 = 3$

The following visualization iterates over the green output activations, and shows that each element is computed by elementwise multiplying the highlighted blue input with the red filter, summing it up, and then offsetting the result by the bias.

EXAMPLE OF CONVOLUTION

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

EXAMPLE OF CONVOLUTION

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

EXAMPLE OF CONVOLUTION

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

EXAMPLE OF CONVOLUTION

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

EXAMPLE OF CONVOLUTION

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

EXAMPLE OF CONVOLUTION

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

EXAMPLE OF CONVOLUTION

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

EXAMPLE OF CONVOLUTION

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

EXAMPLE OF CONVOLUTION

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	2	1	1	0
0	2	1	1	2	2	0
0	2	0	0	0	0	0
0	1	0	2	1	0	0
0	2	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	2	2	1	2	1	0
0	0	2	1	1	1	0
0	0	2	2	1	1	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	2	2	0	0
0	0	2	0	1	1	0
0	2	0	0	1	0	0
0	1	1	1	0	0	0
0	1	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	1	1
0	-1	0

w0[:,:,1]

-1	-1	-1
-1	0	0
1	0	0

w0[:,:,2]

0	1	-1
0	1	0
0	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	-1
1	1	-1
1	1	0

w1[:,:,1]

1	0	1
0	1	1
1	-1	0

w1[:,:,2]

-1	-1	1
0	1	0
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	2	0
-3	-7	-3
3	1	-3

o[:,:,1]

4	8	8
12	5	2
6	5	5

toggle movement

Comparison of ANN to CNN

Typical fully-connected Artificial Neural Networks can easy overfit for medium and large images (e.g. 100x100x3, 3 color channels R, G, and B) because of the huge number of connection weights (parameters) (100*100*3 = 30000) in comparison to the number of trained objects. Moreover, such representation is wasteful and computationally expensive!

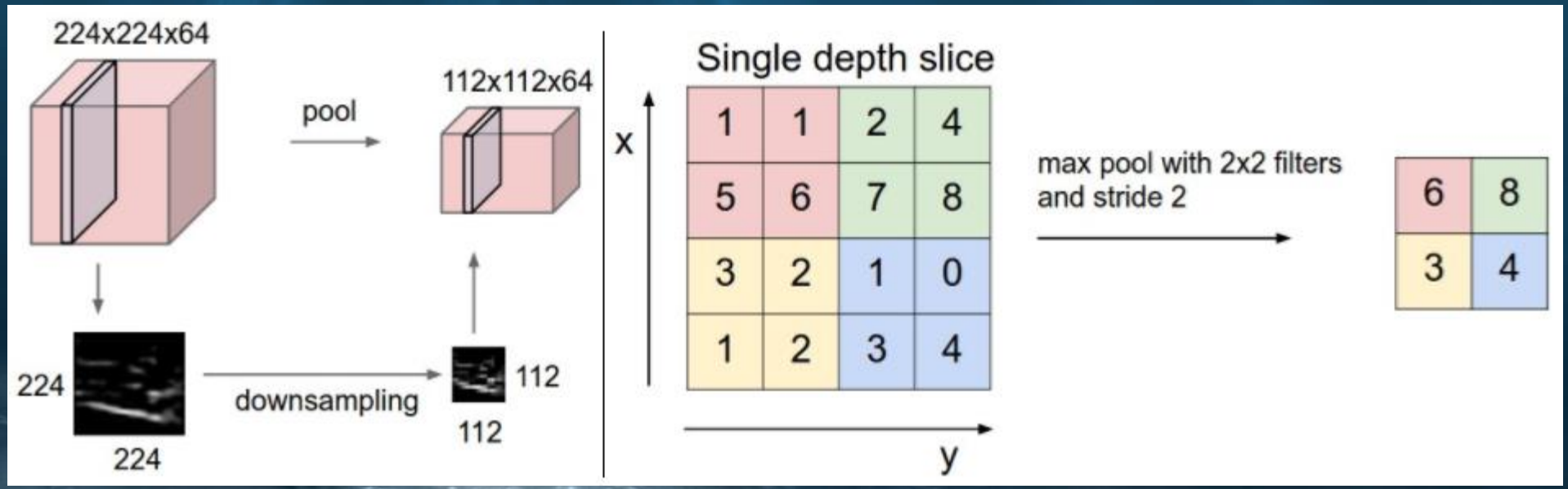
In the CNN structure, each neuron is connected to only a local region of the input volume. The local region is defined in width and height dimension, while the **depth is always full along the entire input volume**. The extent of the connectivity along the depth axis of the CNN is always equal to the depth of the input volume. This limited connectivity hyperparameter is called a **receptive field**, e.g.:

Suppose that the input volume has size [32x32x**3**]. If the receptive field is 5x5, then each neuron in the convolutional layer will have connection weights to a [5x5x**3**] region in the input volume (5*5*3=75 weights + 1 bias parameter). The **depth** of the input volume is here **3**.

POOLING LAYER – MAX OPERATION

It is very common to periodically insert a **pooling layer** in-between successive **convolutional layers** in the CCN architecture. Its main function is to progressively **reduce** the **spatial size of the representation** and the **number of parameters** as well as **computational effort**. It also helps to **control overfitting** because the less parameters we have the less problems with overfitting we have. The pooling layer typically uses **MAX operation** independently on every depth slice of the input and **resizes it spatially**.

The most common form of pooling is to use filters of size 2x2 applied with the stride 2, downsampling every depth slice in the input by 2 along both width and height, discarding 75% of the activations, because we always choose 1 maximum activation from four activations in the region 2x2 in each depth slice. The depth is always preserved.

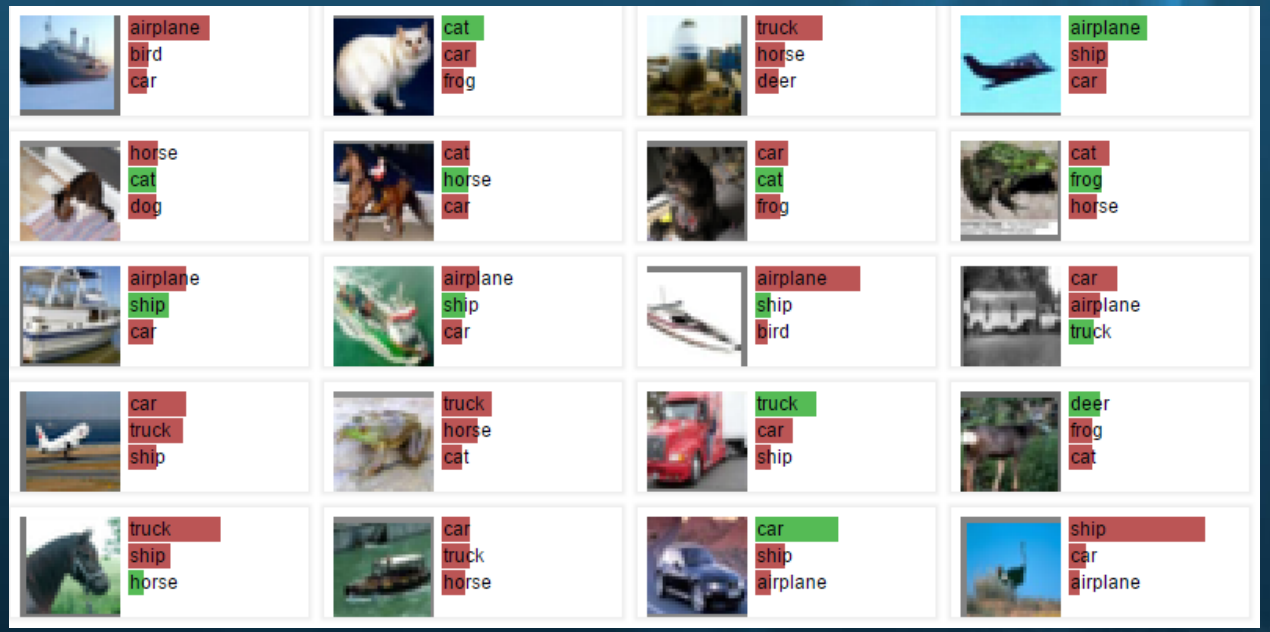


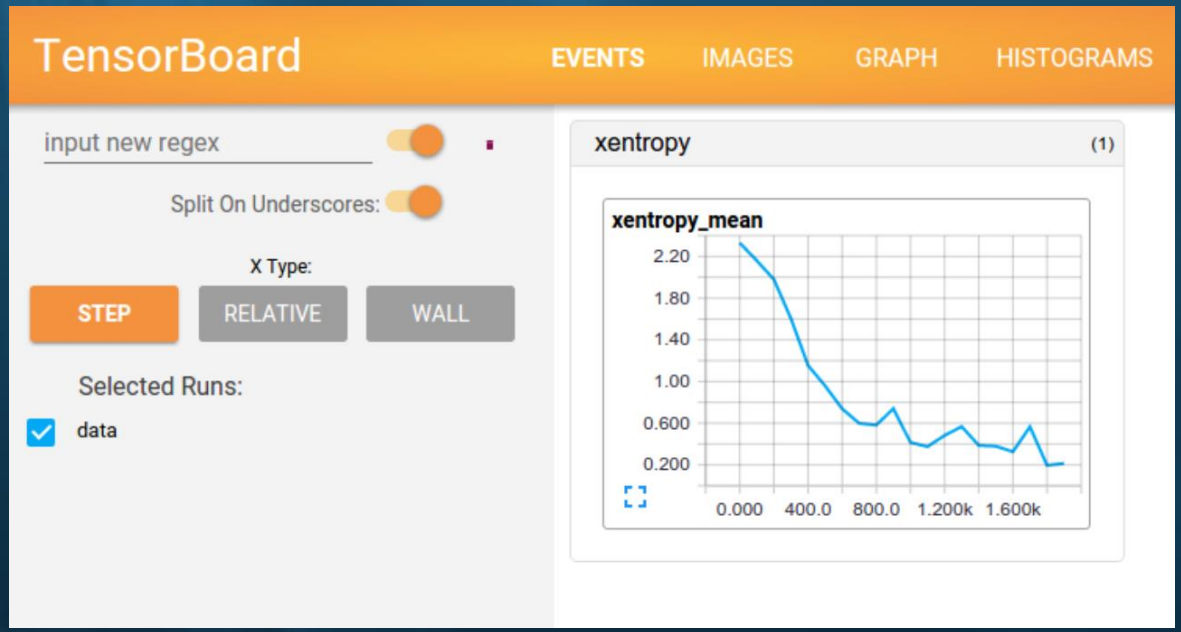
EXAMPLES OF CNN ARCHITECTURES

Examples of CNN architectures: AlexNet, GoogLeNet, LeNet, ResNet, VGGNet.

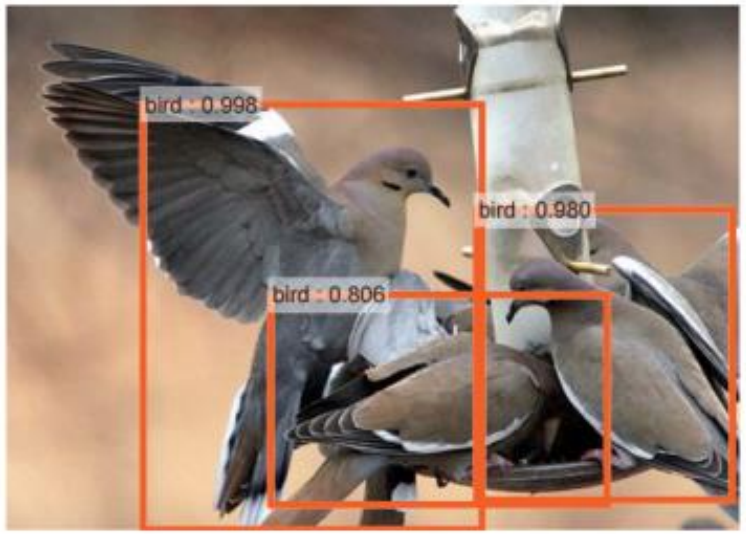
Bechmark training data: MNIST, CIFAR-10, CIFAR-100, STL-10, and SVHN.

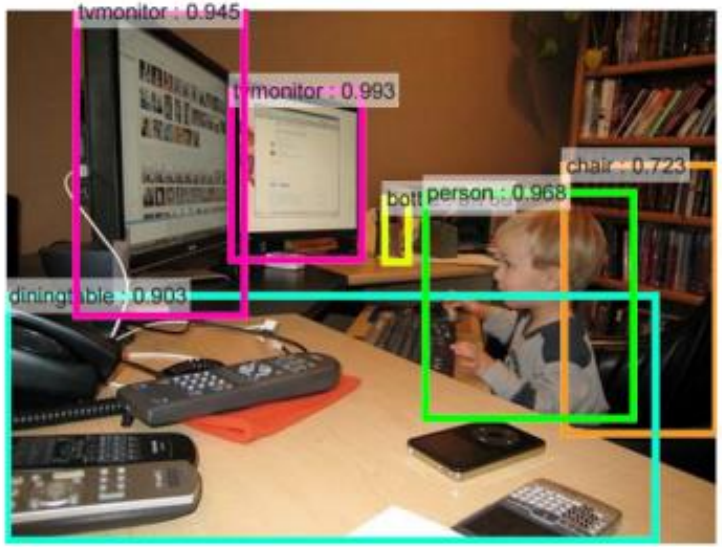
CNN tools: Theano, PyLearn2, Lasagne, Caffe, Torch7, Deeplearning4j, TensorFlow.

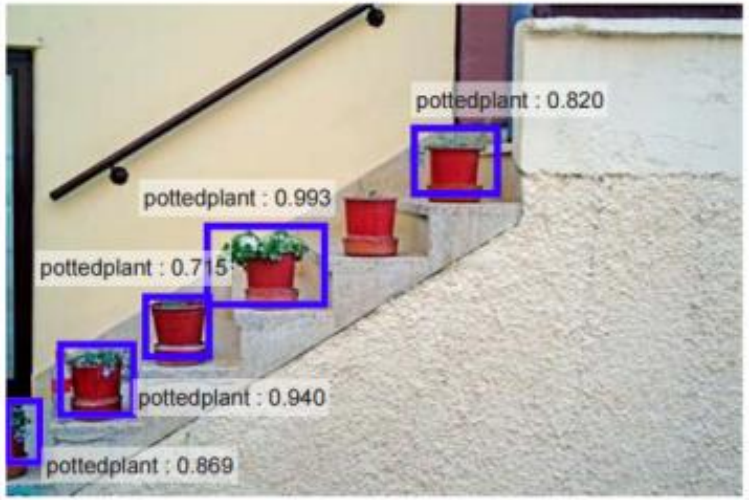


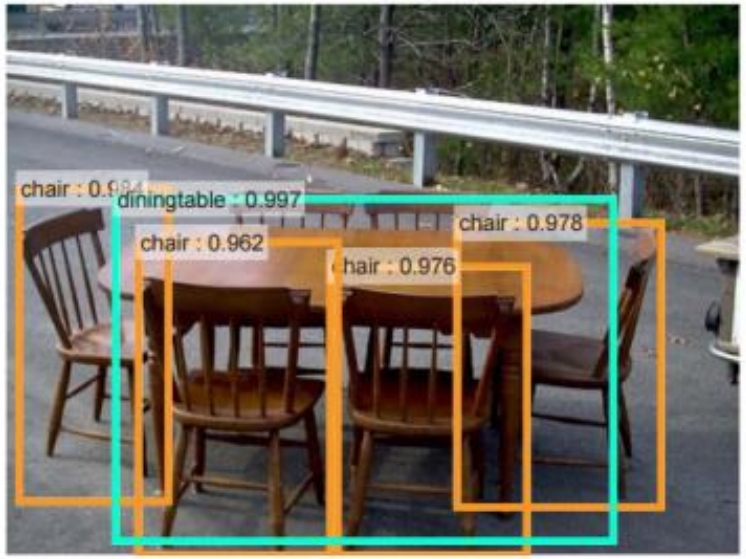


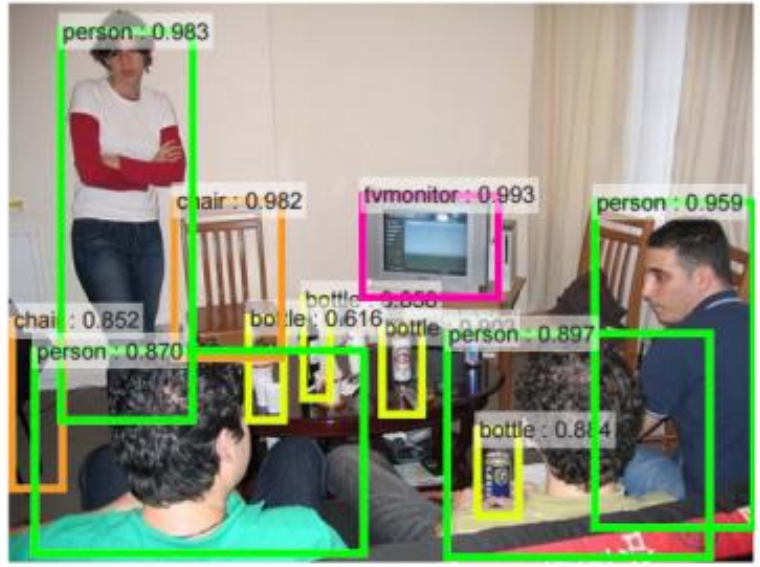
EXAMPLES OF CONVOLUTIONAL NN



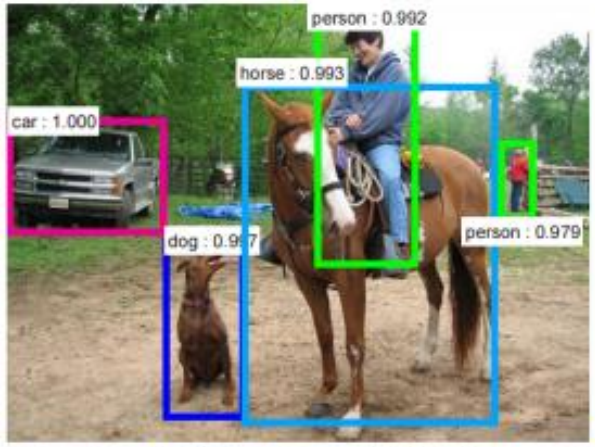


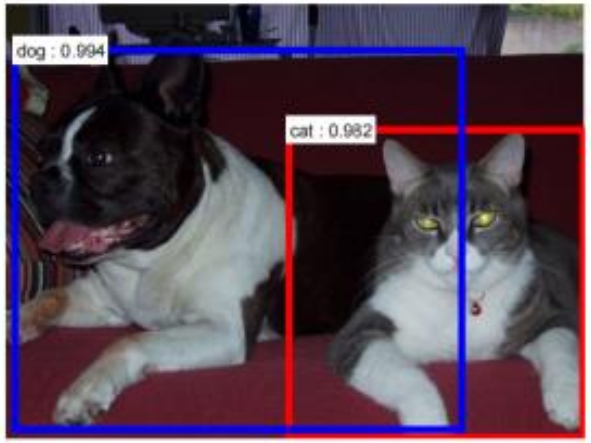


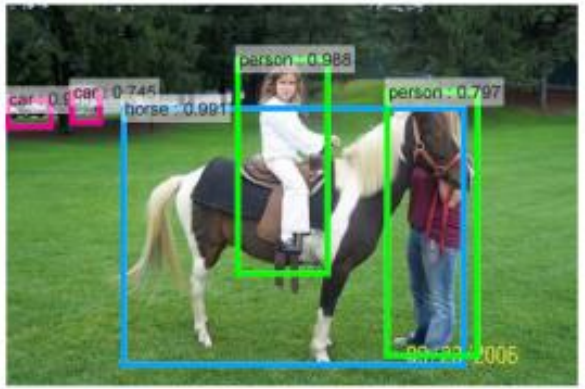


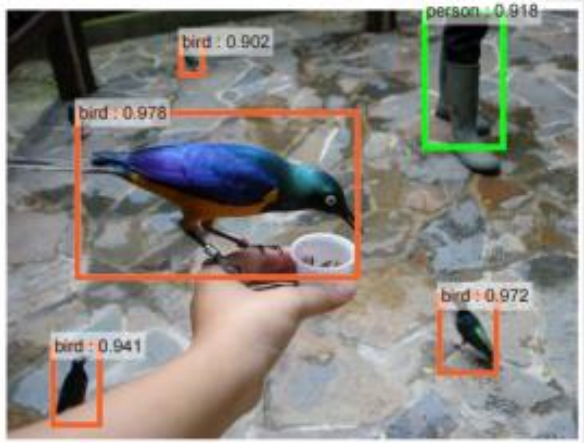


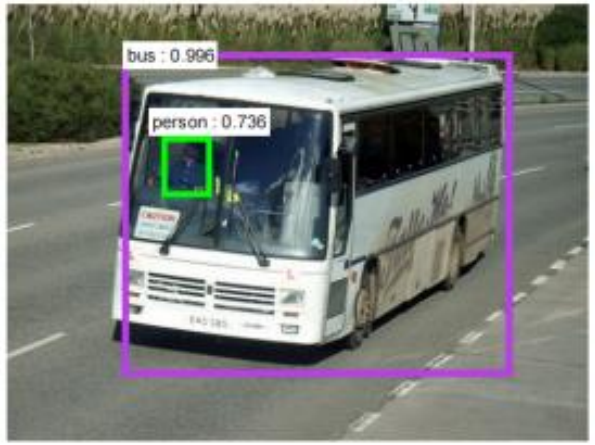
EXAMPLES OF CONVOLUTIONAL NN

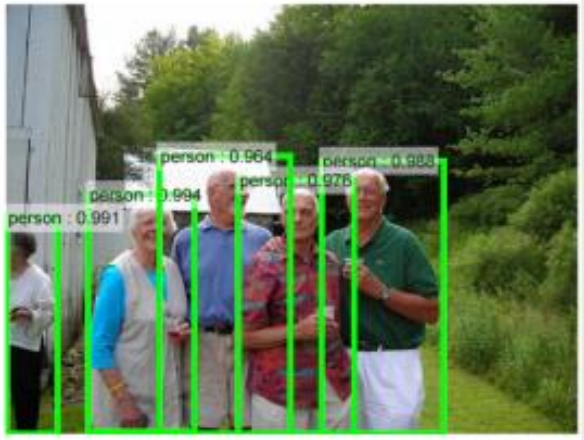












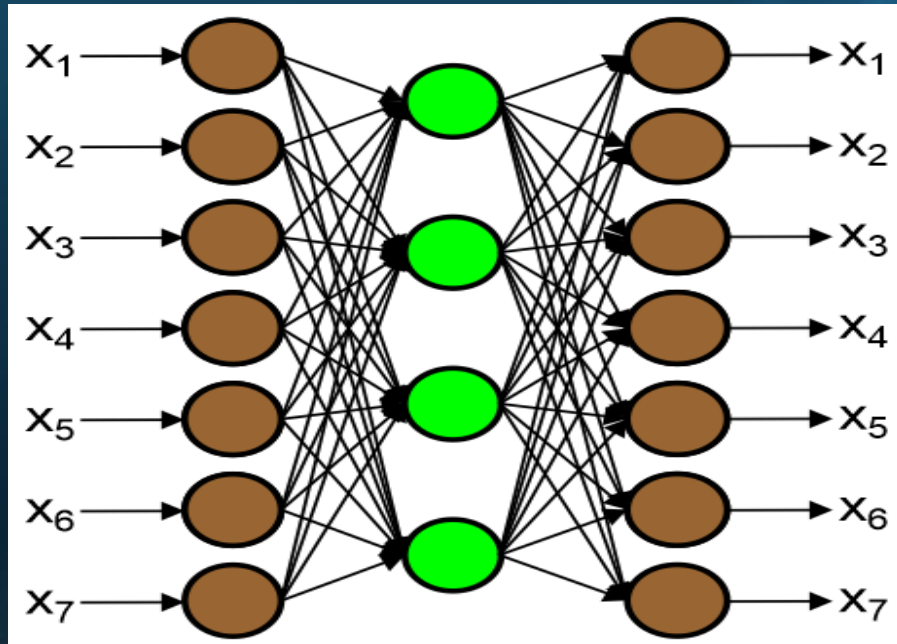
EXAMPLES OF CONVOLUTIONAL NN

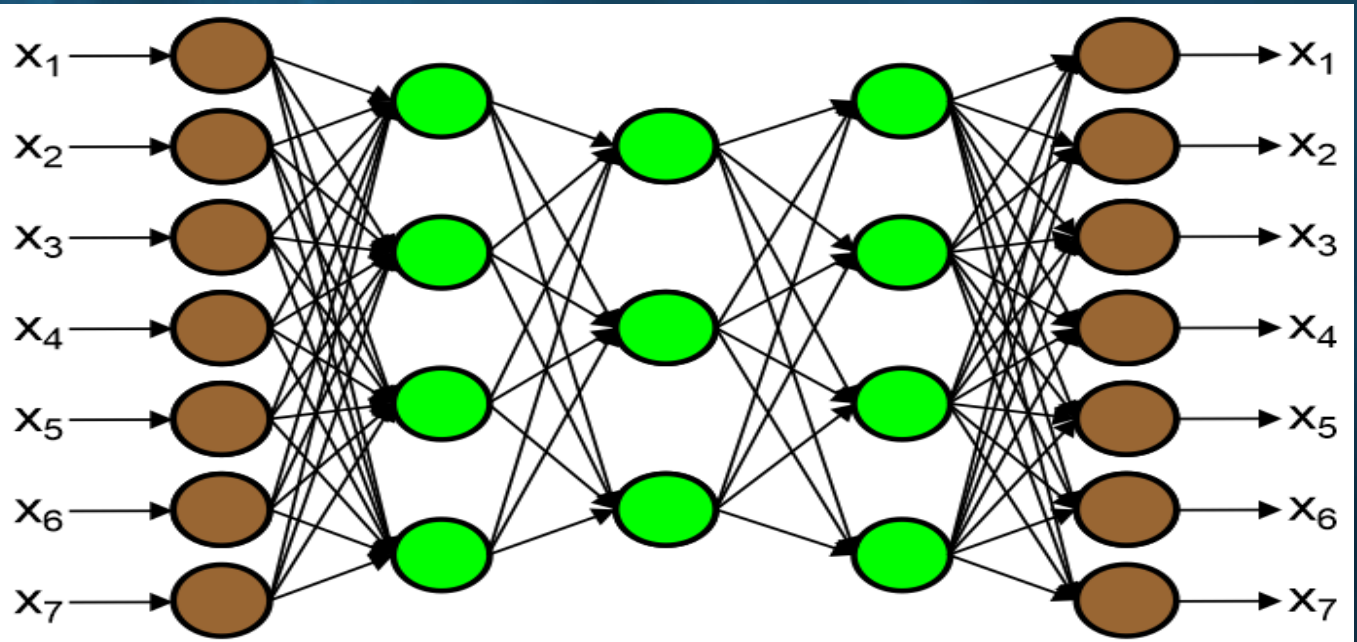
Input

Feature Map

AUTOENCODER – AUTOASSOCIATOR

In **deep neural networks**, we often use **autoencoders**, which are a kind of artificial neural networks used for **unsupervised learning** and an **efficient coding**. There is used a set of inputs that is trained in such a way to get outputs identical with inputs. The **main purpose** of such training is to find out a reduced number of neurons in comparison to the input data dimension, which is able to represent these data without distortions. Thus, we can achieve **dimensionality reduction** in the hidden layer of autoencoder neural network:



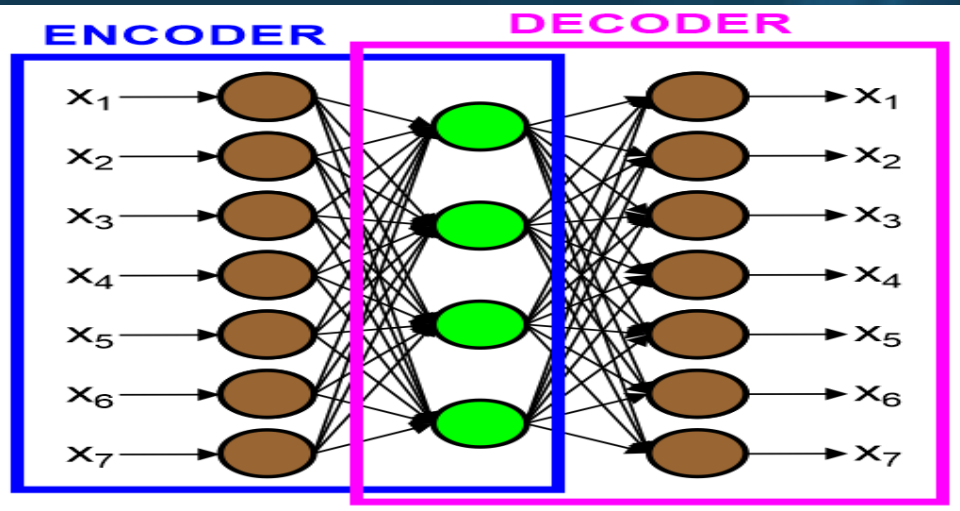


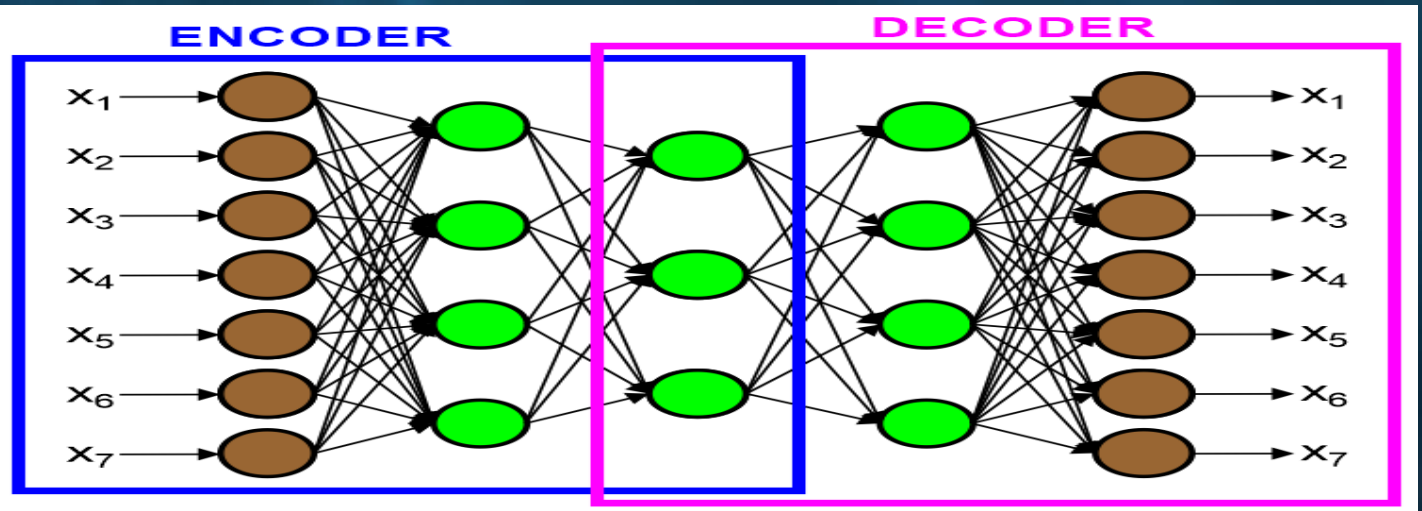
TRAINING OF AUTOENCODERS

Autoencoders can be adapted using gradient descent methods used for supervised training because the outputs are identical with inputs, so we can easily compute output error and delta parameters to use e.g. backpropagation to adapt these kind of neural networks.

The question is why to learn a neural network the autoassociative projection?

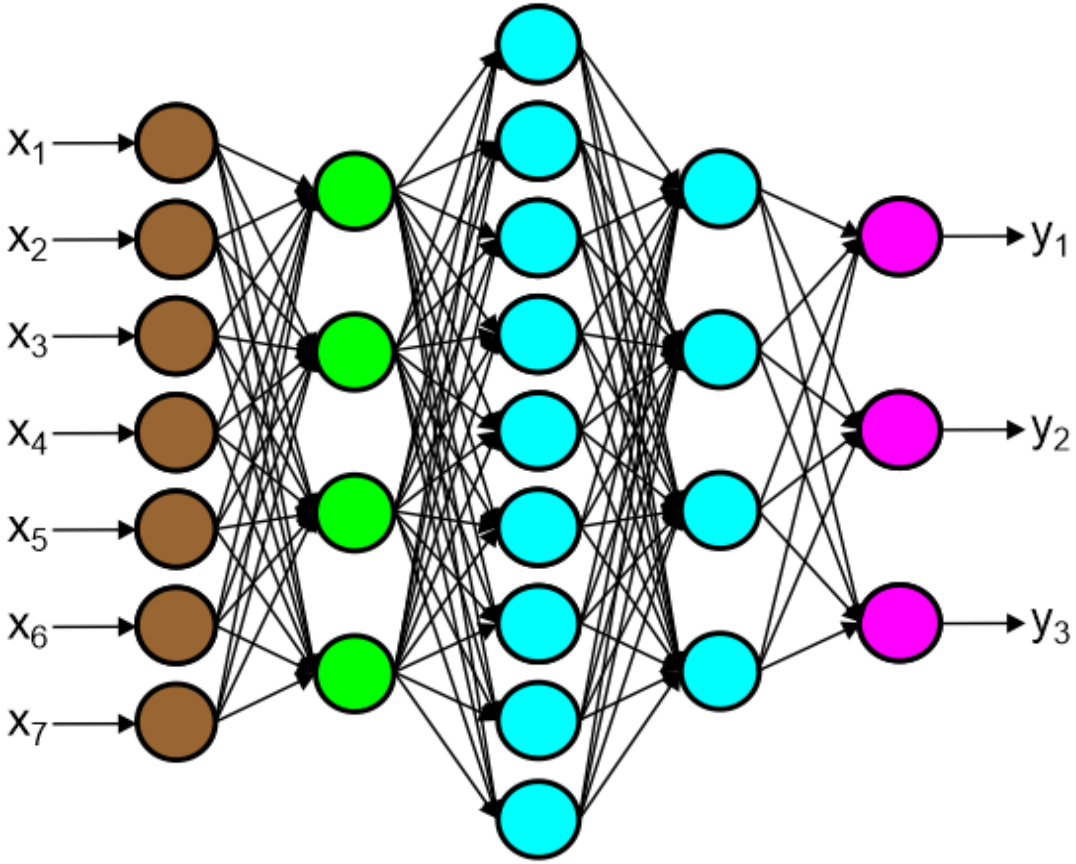
The main reason is hidden in the hidden layer(s), which sparingly represent input data in a less dimensional space. In practice, it means that the hidden layer neurons must represent similarities (similar features) of the input objects to be able to decode them on the outputs:

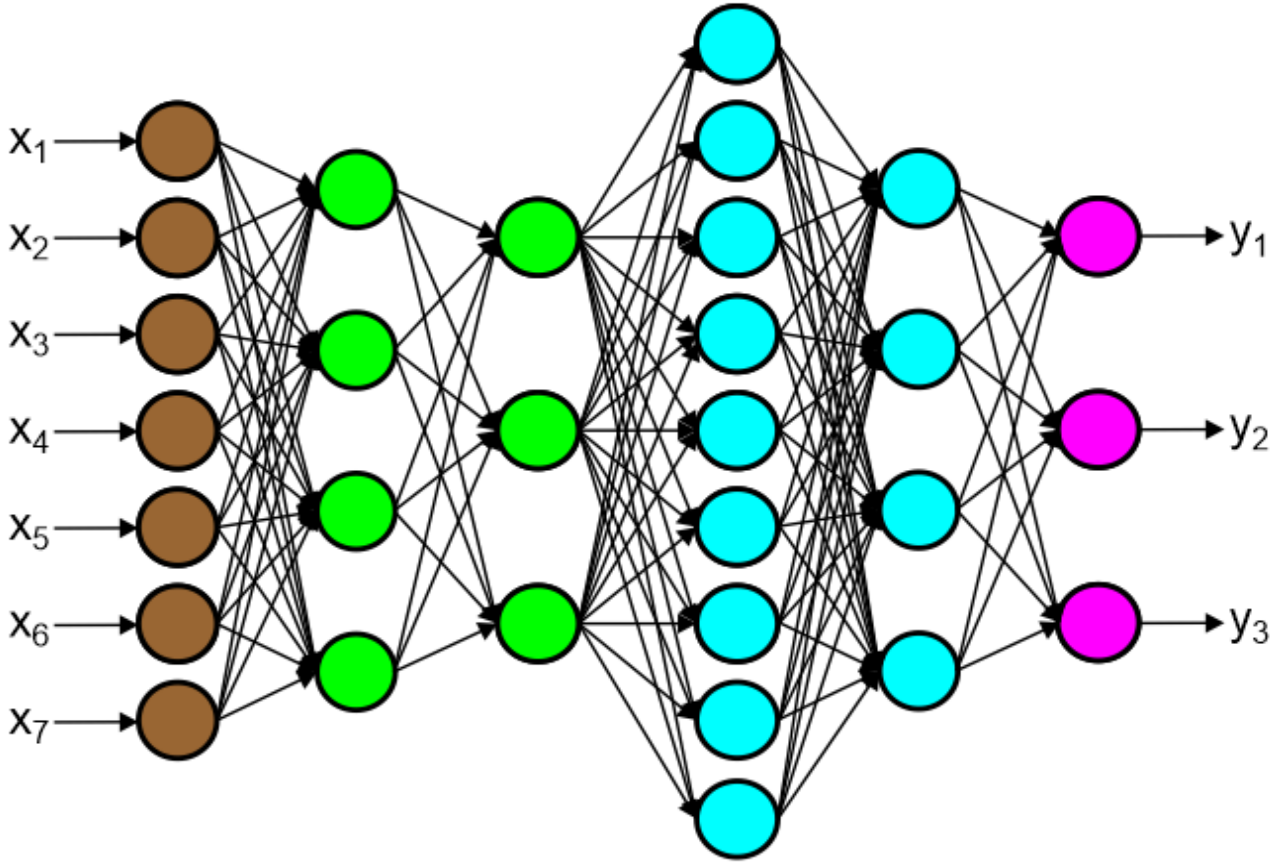




USING AUTOENCODERS IN DEEP ARCHITECTURES

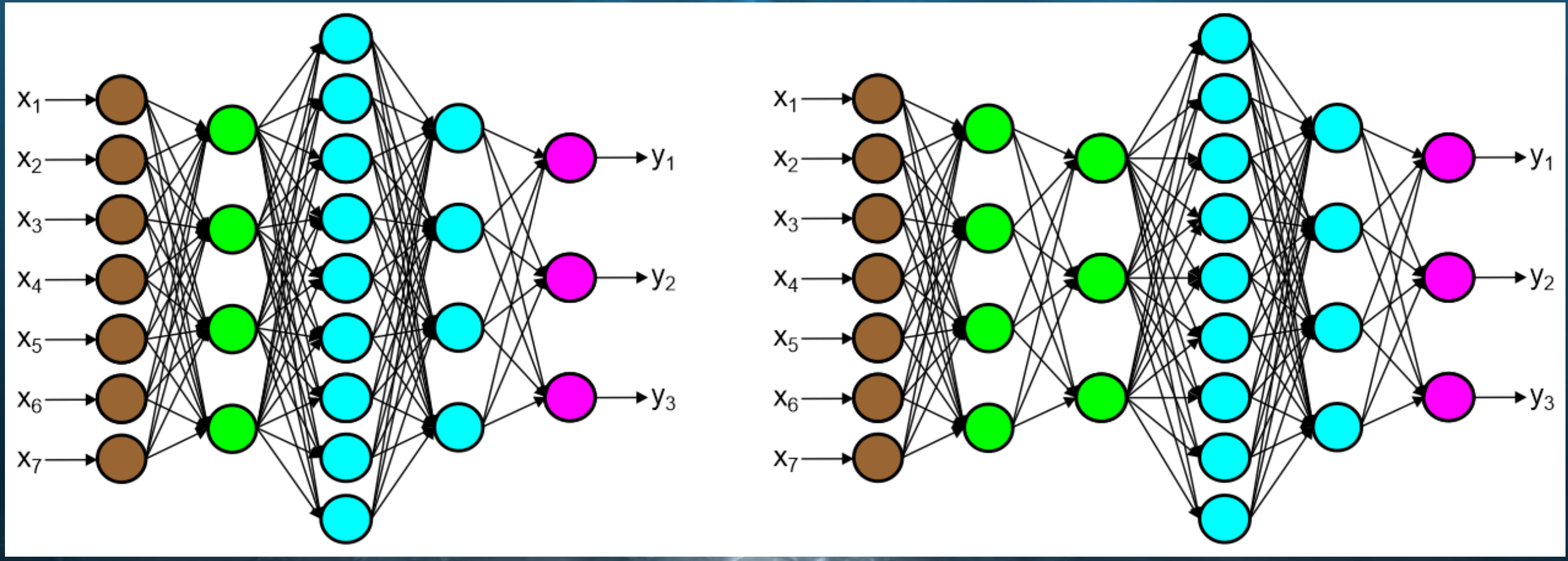
Autoencoders are typically used in deep architectures as layers that do not demand supervised training but they are trained in an unsupervised ways in order to find out and represent the most important features which can be used for next classifications:





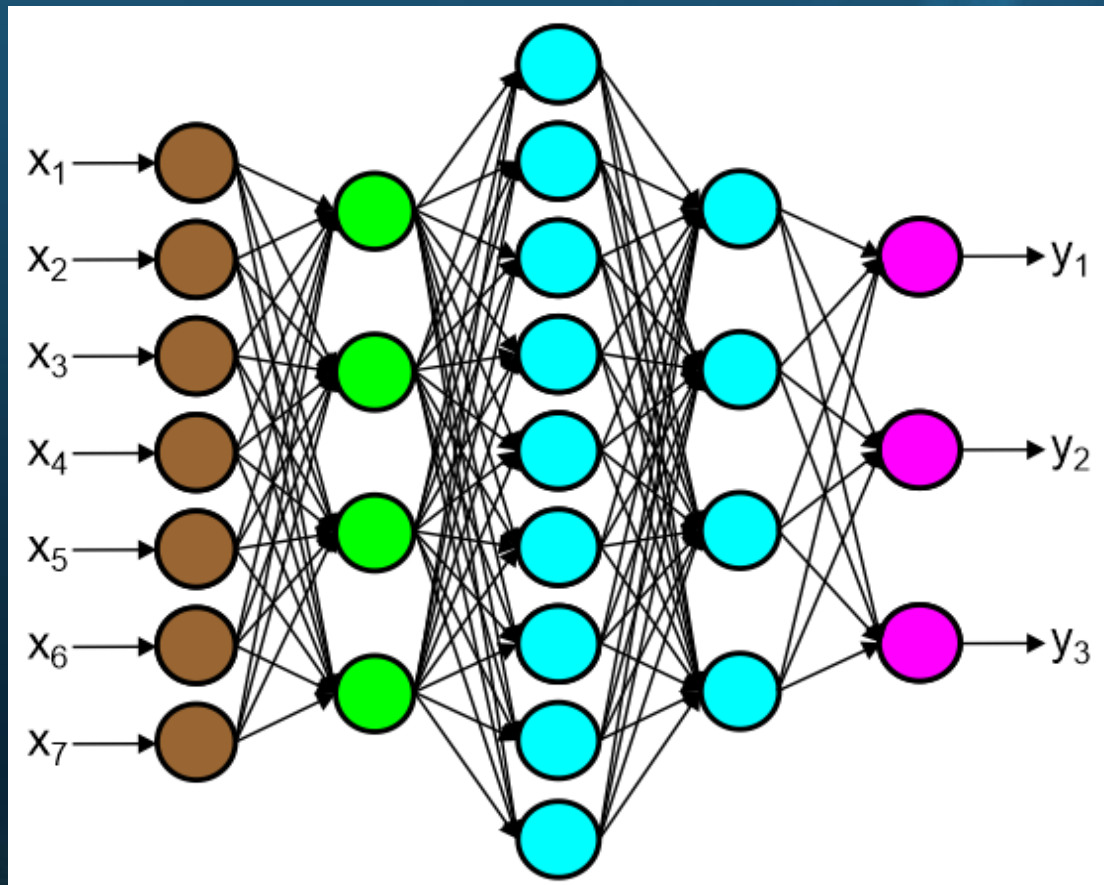
USING AUTOENCODERS IN DEEP ARCHITECTURES

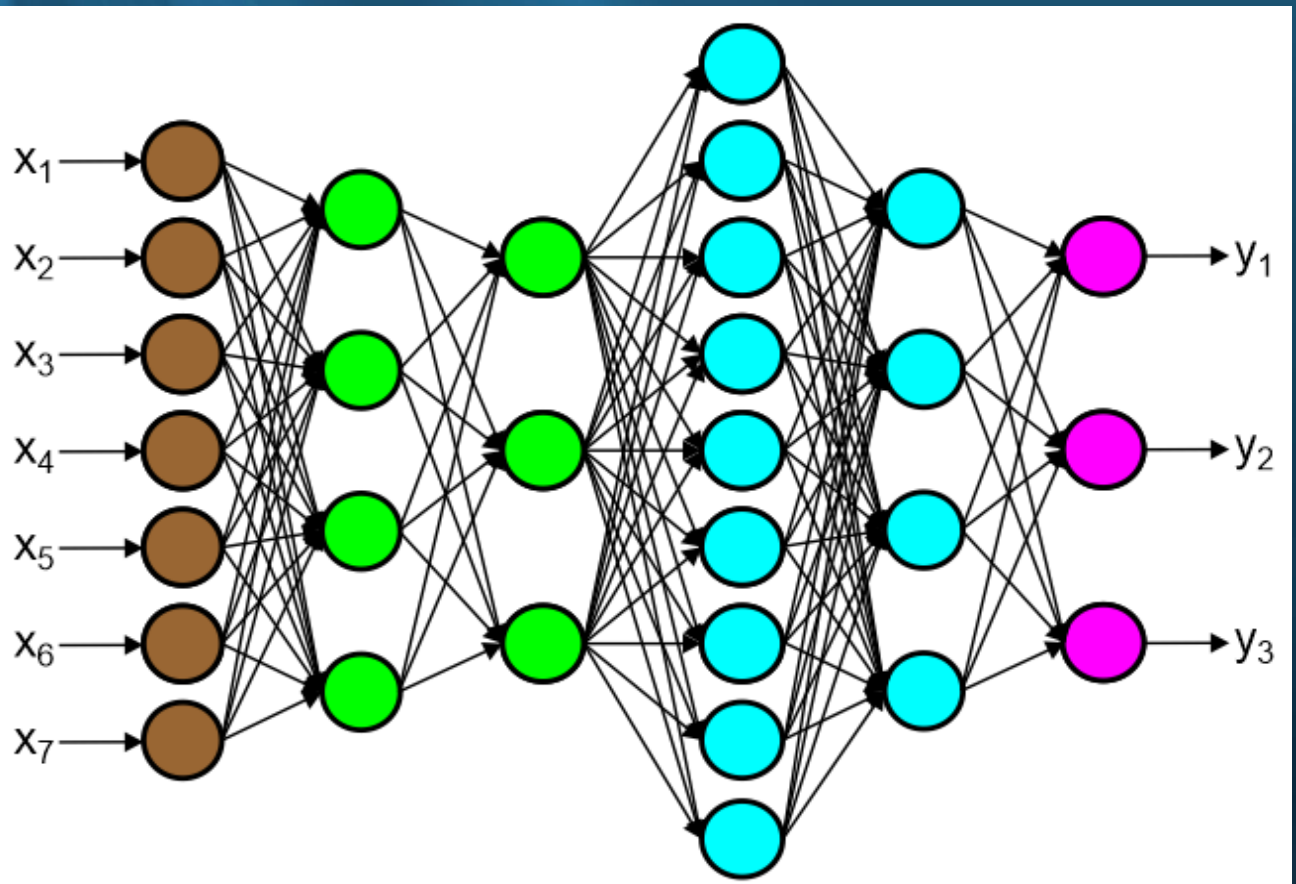
Autoencoders are trained in the unsupervised manner in the first preliminary stage of the whole adaptation process, and then the supervised training of the remaining part of the neural network is proceeded to fine-tune the outputs for e.g. classification:



USING AUTOENCODERS IN DEEP ARCHITECTURES

First, **autoencoders** are trained, then we cut off the last layers and use the rest of the network (the input layer and the hidden layers) to further build a deep network structure, which in the final layers is trained in a supervised manner and can also fine-tune the layers of autoencoders:

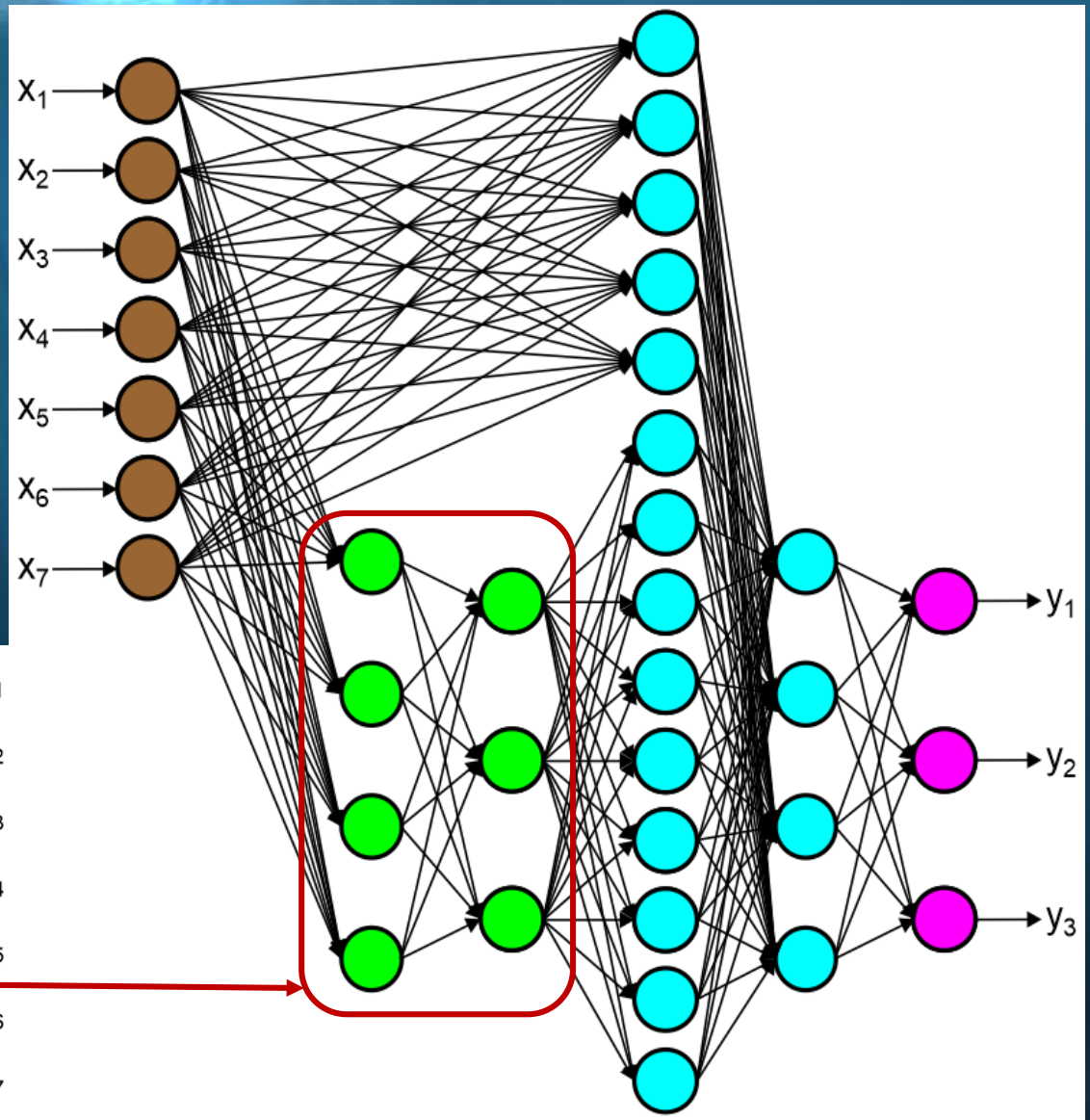


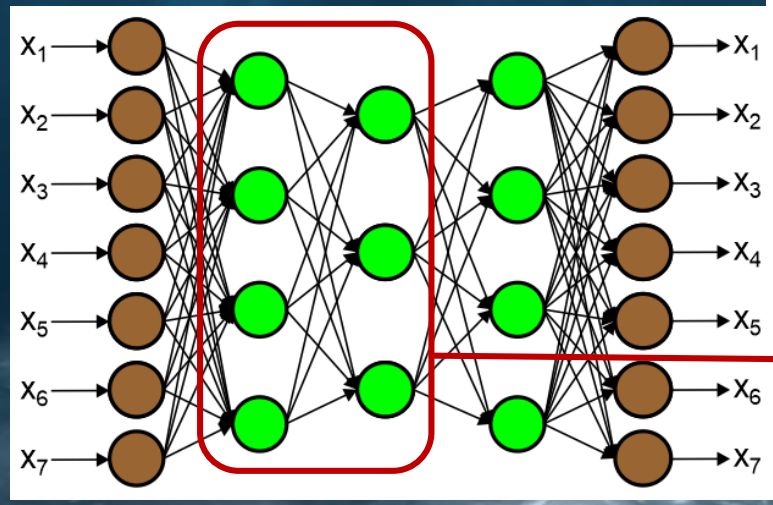


HYBRID DEEP ARCHITECTURES INCLUDING AUTOENCODERS

Autoencoders can be used in various hybrid deep neural network architectures for feature extraction, similarly to convolutional layers in CNNs:

It means that the main problem in various **classification and recognition tasks** is the process of **feature extraction**, and the next training results are based on the quality achieved in this preliminary stage of data processing!





BIBLIOGRAPHY AND LITERATURE

- Ian Goodfellow, Yoshua Bengio and Aaron Courville, Deep Learning, MIT Press book, 2016.
- DeepMind Video - How it works?
- Convolutional Neural Networks for Visual Recognition
- Convolutional Neural Network (Stanford)
- ImageNet Classification with Deep CNNs
- Intuitive explanation of ConvNets
- Image Style Transfer Using Convolutional Neural Networks, Leon A. Gatys, Alexander S. Ecker, Matthias Bethge.
- Visualizing and Understanding Convolutional Networks, Zeiler, Fergus, ECCV 2014
- Pattern Recognition and Machine Learning (Information Science and Statistics), Bishop, Christopher M., 2006
- Neural Networks and Deep Learning, Michale A. Nielsen, Determination Press, 2015
- An Intuitive Explanation of Convolutional Neural Networks
- Convolutional Neural Networks (LeNet)
- Tianyi Liu, Shuangsang Fang, Yuehui Zhao, Peng Wang, Jun Zhang Implementation of Training Convolutional Neural Networks
- Neural Networks and Deep Learning
- Unsupervised Feature Learning and Deep Learning
- Theano Convolution Arithmetic Tutorial
- Backpropagation In Convolutional Neural Networks

Bibliography and References

http://home.agh.edu.pl/~horzyk/lectures/ahdydci.php

ACADEMIC WEBSITE - ADRIAN HORZYK, PhD, Prof.

AGH University of Science and Technology in Cracow, Poland
Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering
Department of Automatics and Biomedical Engineering, Field of Biocybernetics

AGH

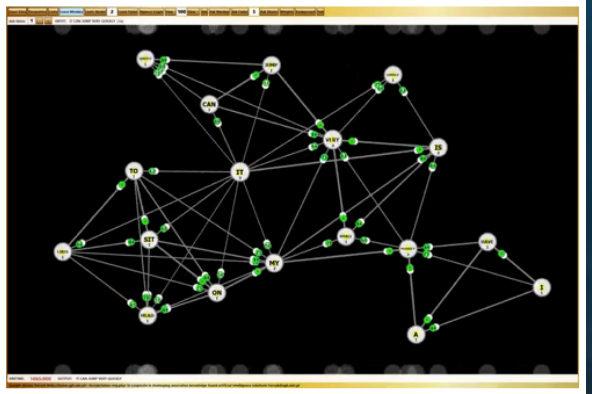
LECTURES

- Introduction to Artificial, Cognitive and Computational Intelligence
- Introduction to Neural Networks
- Models of Neurons
- Construction and Learning Strategies
- Overview of Neural Network Models
- Multilayer Perceptron MLP
- Radial Basis Function Networks RBF
- Support Vector Machine SVM
- Reinforcement Learning
- Deep Learning DL
- Unsupervised Training and Networks
- Self Organizing Maps SOM
- Recurrent Neural Networks
- Associative and Semantic Memories
- Associative Neural Systems

COMPUTATIONAL INTELLIGENCE

is already under construction and will be available at the spring semester in 2016/2017.

This course will include 28 lectures, 14 laboratory classes and 14 project classes.

What is this course about?

This course is intended to give students a broad overview and deep knowledge about popular solutions and efficient neural network models as well as to learn how to construct and train intelligent learning systems in order to use them in everyday life and work. During the course we will deal with the popular and most efficient models and methods of neural networks, fuzzy systems and other learning systems that enable us to find specific highly generalizing models solving difficult tasks. We will also tackle with various CI and AI problems and work with various data and try to model their structures in such a way to optimize operations on them throughout making data available without necessity to search for them. This is a unique feature of associative structures and systems. These models and methods will allow us to form and represent knowledge in a modern and very efficient way which will enable us to mine it and automatically draw conclusions. You will be also able to understand solutions associated with various tasks of motivated learning and cognitive intelligence.