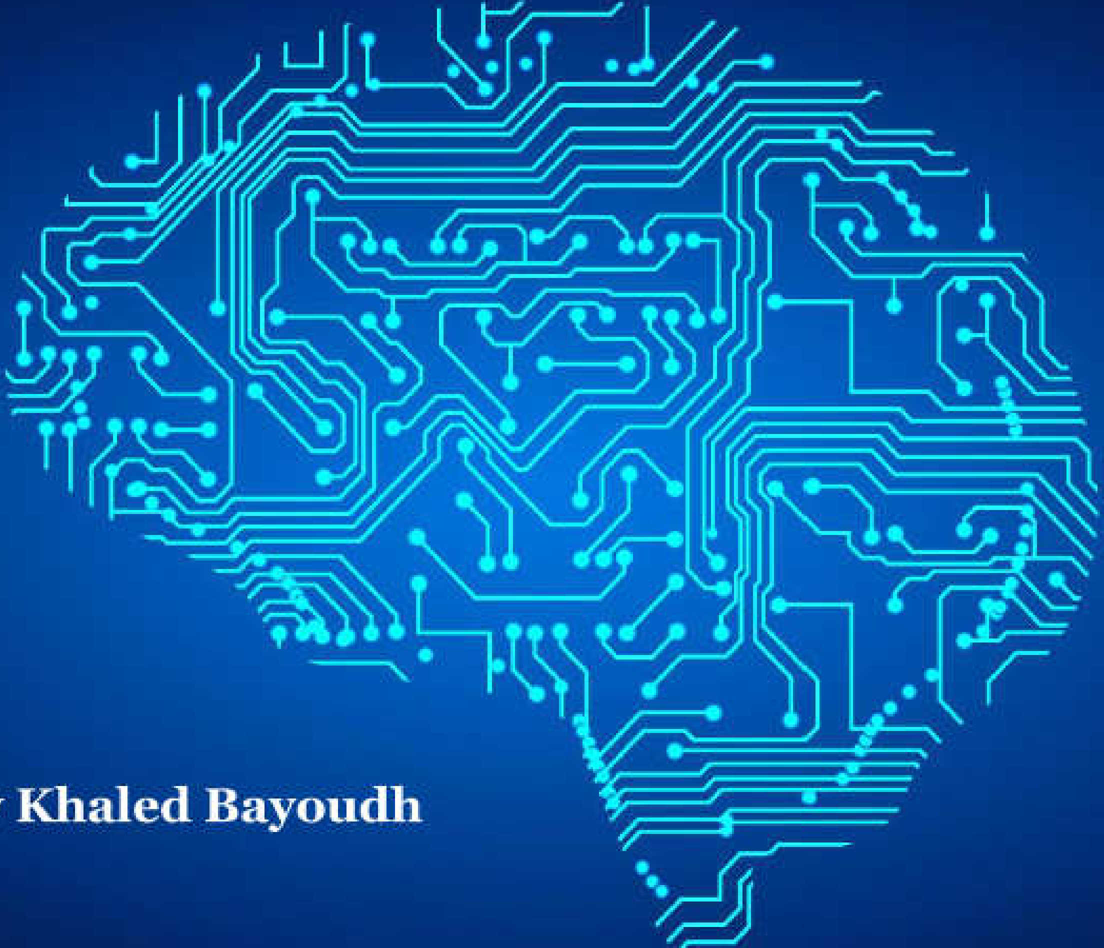
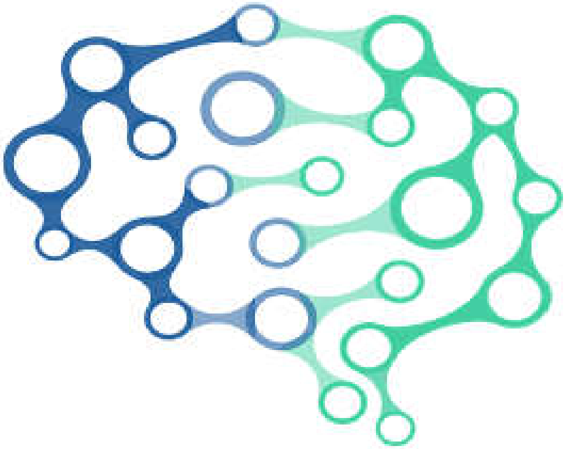
From Machine Learning to Deep Learning



by Khaled Bayoudh

From Machine Learning to Deep Learning



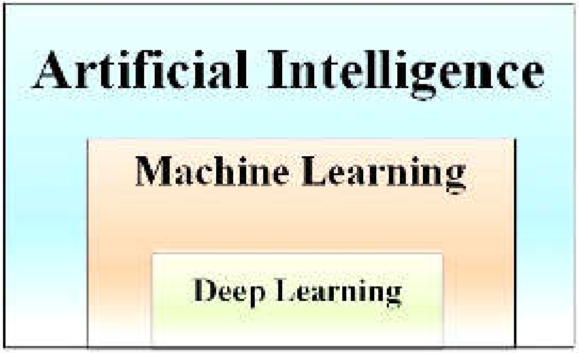
Basic Concepts in Machine / Deep Learning

by Khaled Bayoudh

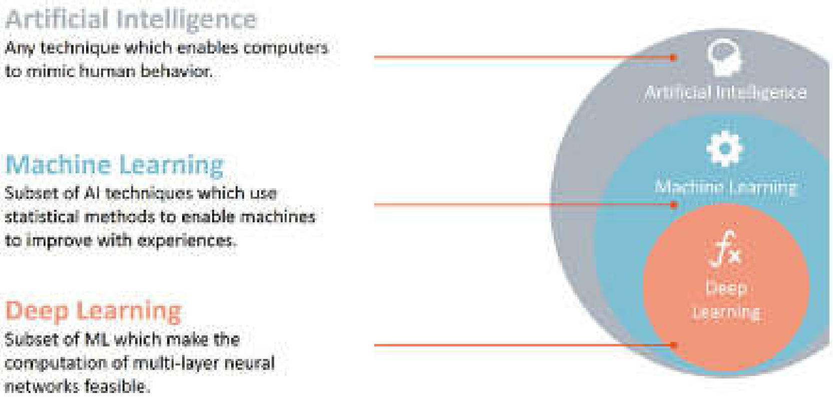
Table of Contents

Introduction to Artificial intelligence

Today, the world technology is witnessing a lot of confusion about Machine Learning and Deep Learning.
In fact, these concepts are only a subset of artificial intelligence.



Artificial intelligence is defined as one of the Research and Development (R&D) areas that is used to model an intelligent systems with intelligent behaviors. But the question that now assumes importance: why Machine Learning algorithms (especially Deep Learning) are prevalent at this stage, knowing that there are many improvements to these systems for many years. Most of the major high-tech companies are oriented towards the integration of artificial intelligence technologies in the industrial and research fields, especially after the significant advancement of computing power (GPUs) and the availability of large quantities of data (Big Data) on the Internet.


Artificial Intelligence
Any technique which enables computers to mimic human behavior.

Machine Learning
Subset of AI techniques which use statistical methods to enable machines to improve with experiences.

Deep Learning
Subset of ML which make the computation of multi-layer neural networks feasible.

A GLOSSARY OF ARTIFICIAL-INTELLIGENCE TERMS

- **ARTIFICIAL INTELLIGENCE**

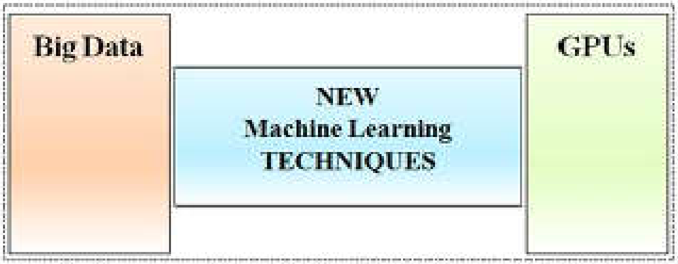
AI is the broadest term, applying to any technique that enables computers to mimic human intelligence, using logic, if-then rules, decision trees, and machine learning (including deep learning).

- **MACHINE LEARNING**

The subset of AI that includes abstruse statistical techniques that enable machines to improve at tasks with experience. The category includes deep learning.

- **DEEP LEARNING**

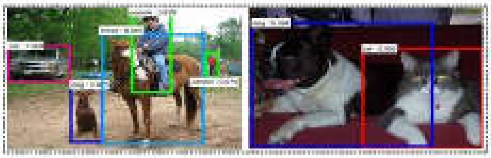
The subset of machine learning composed of algorithms that permit software to train itself to perform tasks, like speech and image recognition, by exposing multilayered neural networks to vast amounts of data.



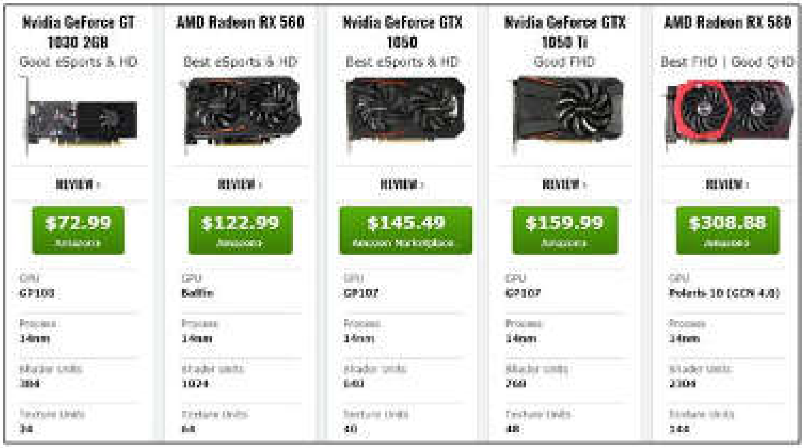
In recent years, artificial intelligence (especially Machine Learning techniques) has surprised people with results closer and some time better than humans such as Natural Language Processing (NLP), Face Recognition, Image Classification and Object Detection.

Natural language processing

Face recognition


Object detection and classification

The picture below shows a selection of GPUs and their prices:

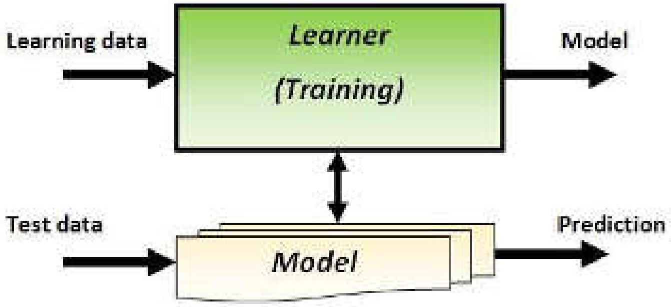


Nvidia GeForce GT 1030 2GB	AMD Radeon RX 560	Nvidia GeForce GTX 1050	Nvidia GeForce GTX 1050 Ti	AMD Radeon RX 580
Good eSports & HD	Best eSports & HD	Best eSports & HD	Good FHD	Best FHD | Good QHD
REVIEW ›	REVIEW ›	REVIEW ›	REVIEW ›	REVIEW ›
$72.99 Amazon	$122.99 Amazon	$145.49 Amazon Marketplace	$159.99 Amazon	$308.88 Amazon
GPU GP108	GPU Baffin	GPU GP107	GPU GP107	GPU Polaris 10 (GCN 4.0)
Process 14nm	Process 14nm	Process 14nm	Process 14nm	Process 14nm
Shader Units 384	Shader Units 1024	Shader Units 640	Shader Units 768	Shader Units 2304
Texture Units 24	Texture Units 64	Texture Units 40	Texture Units 48	Texture Units 144

In the following chapter, we will recall the basics of Machine Learning, including Supervised and Unsupervised learning.

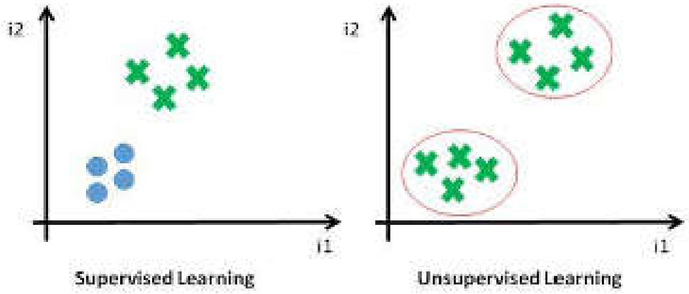
Machine Learning

Machine Learning is a branch of artificial intelligence and looks to make a smart computer by dealing with problems without *programming*.
In the beginning, we prepare a set of data (Train+Test data), train a model from learning data and use the trained model to predict new outputs from new data (test data). In fact, it's a way to make your computer to be able to create a hypothesis or model that gives outputs with known inputs.



The main idea is to provide to machine a set of examples or learning data (i. e. images) and expected outputs (labels), so after the training, we will have a model that will predict the output (classes) of new data.

The purpose of Machine Learning is to ensure that the computer dynamically learn from the data. It can be divided into two categories: Supervised and Unsupervised Learning.



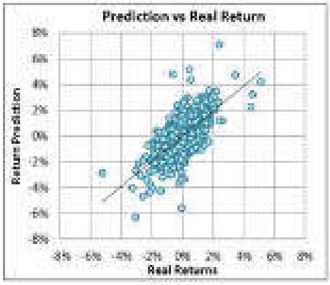
Supervised Learning:

Provide to the machine some couples (inputs, known outputs), when new inputs are introduced, so we have a *smart* outputs.

Now we will see two Supervised Learning techniques:

Classification: Train the machine with a set of examples and labels. By giving a new example to it, so it can predict the output (label) and recognize the new object.

Regression: Train the machine with a historical data, then it must estimate the new value on the future.



Regression VS Classification

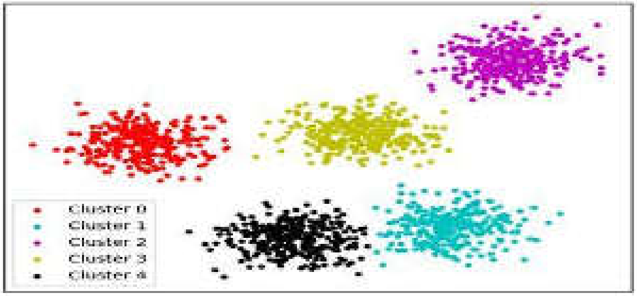
Unsupervised Learning:

The machine can learn from the data without any supervision and knowing what the estimated output.

Now we will see two unsupervised Learning techniques:

Clustering: It's about separating similar information on *clusters*. This technique can be mainly exploited in research and science.

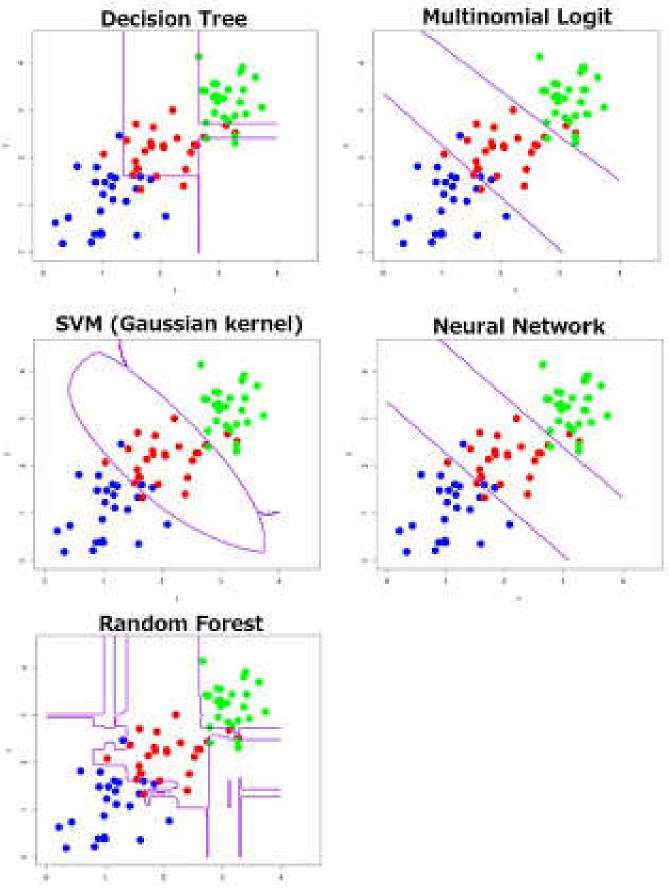
Generative Models: It's about generating more data from input data.



Clustering

There are many Machine Learning algorithms available. These algorithms depend on the problem you have to solve and the quantity of data (datasets).

On this book we will focus more on neural networks (NN).



Linear Algebra

Linear algebra consists of the study of vector spaces and linear applications between vector spaces. A vector space is a set of operations including *addition* and *scalar multiplication*.

What is the difference between Scalar, vector, matrix and tensor?

Scalar: *A single element (number),*

Vector: *Array of elements (numbers),*

Matrix: *is a 2 dimensional (2D : row/column) array of elements.*

Scalar	Vector	Matrix
7	$[-1\ 2\ 7]$	$\begin{bmatrix} -1 & 2 & 7 \\ 3 & 5 & 8 \end{bmatrix}$

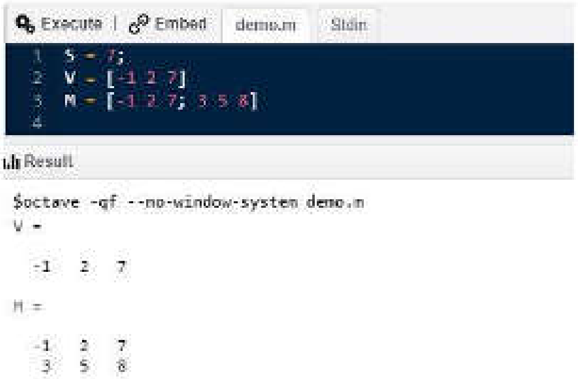
Tensor: tensors are geometric objects that describe linear relations between geometric vectors, scalars, and other tensors.

Now we will see its creation on MATLAB:

The scalar S has dimension of 1x1;

The vector V has a dimension of 1x3;

The matrix M has dimension of 2x3 .



```
S = 7;
V = [-1 2 7]
M = [-1 2 7; 3 5 8]
```

```
$octave -qf --no-window-system demo.m
V =

  -1   2   7

M =

  -1   2   7
   3   5   8
```

Now, we will see the operations of the matrix:

Addition, *subtraction* and *multiplication* operations are then the most usual matrix operations. These operations are among the basic concepts of Machine Learning.

<u>Addition and subtraction:</u> To add (or subtract) two matrices M1 and M2, we must add (or subtract) each element of the matrix M1 with the matrix M2, knowing that these two matrices must have the same dimension.

Multiplication: It concerns the dot product of two matrices M1 and M2 with dimensions of m × ℓ and n × ℓ respectively.The output of this operation for the entry (i, j) is the dot product of the *iTH* row of M1 with the *jTH* column of M2.

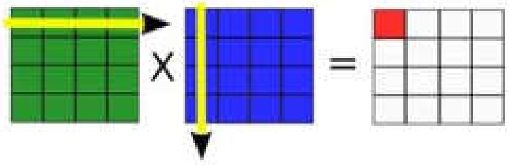
Addition (+):

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 7 \\ 4 & 4 \end{bmatrix}$$

Subtraction (-):

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0 & -2 \end{bmatrix}$$

Multiplication (*):

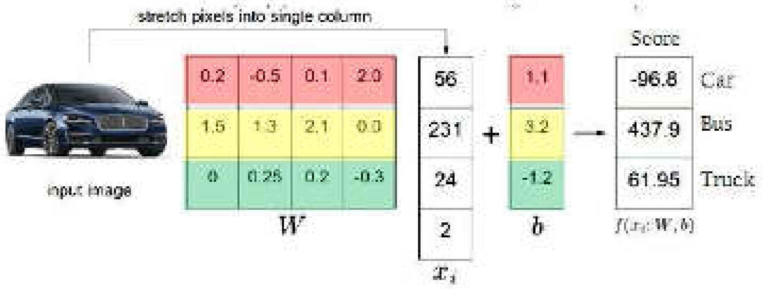


$$\begin{bmatrix} \boxed{2 \quad 3} \\ 2 \quad 1 \end{bmatrix} * \begin{bmatrix} \boxed{\begin{matrix} 1 \\ 2 \end{matrix}} & \begin{matrix} 4 \\ 1 \end{matrix} \end{bmatrix} = \begin{bmatrix} 8 & 11 \\ 4 & 9 \end{bmatrix}$$

- ✓ (2*1) + (3*2) = 8
- ✓ (2*4) + (3*1) = 11
- ✓ (2*1) + (1*2) = 4
- ✓ (2*4) + (1*1) = 9

Now, we will see the utility of the matrix multiplication operation (dot product) to construct a linear classifiers for Machine Learning.

Later, we will introduce you the definition of *linear classifiers*, but for now we show you the usefulness of linear algebra to solve some problems of Machine Learning (i.e. linear classification).

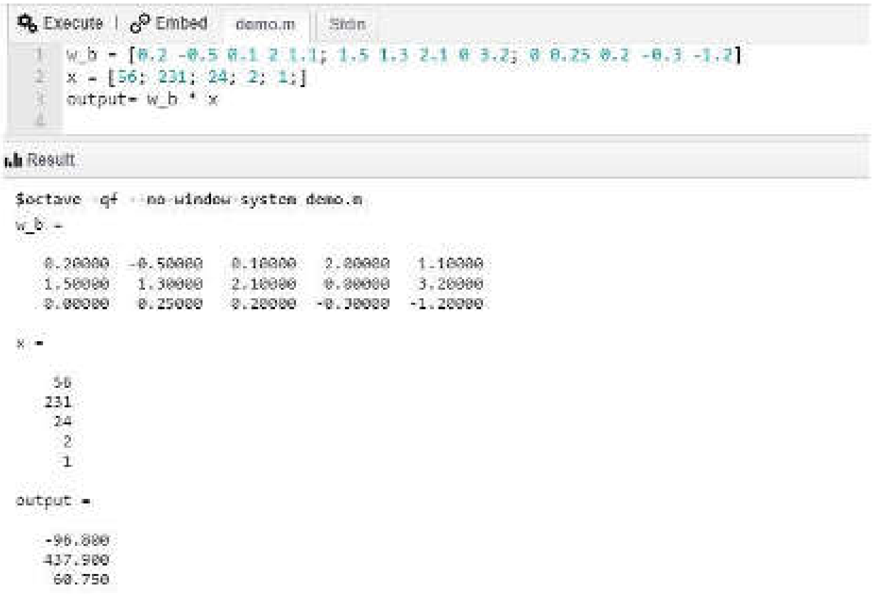


Knowing that:

W: weight vector
b: bias
x: input vector

The output equal to *Wx+b* (linear classifier).

Next, we will see its implementation on MATLAB.



```
w_b = [0.2 -0.5 0.1 2 1.1; 1.5 1.3 2.1 0 3.2; 0 0.25 0.2 -0.3 -1.2]
x = [56; 231; 24; 2; 1;]
output= w_b * x
```

```
$octave -qf --no-window-system demo.m
w_b =

   0.20000  -0.50000   0.10000   2.00000   1.10000
   1.50000   1.30000   2.10000   0.00000   3.20000
   0.00000   0.25000   0.20000  -0.30000  -1.20000

x =

    56
   231
    24
     2
     1

output =

   -96.800
   437.900
    60.750
```

-96.8, 437.9 and 60.75 represent the scores generated by the classifier (output).

Next, we will present the main concepts of the linear classification including Supervised Learning.

Supervised Learning

Supervised learning consists of learning model (or training model) from labeled data, ie with couples (input, output), knowing that the output is known in advance.

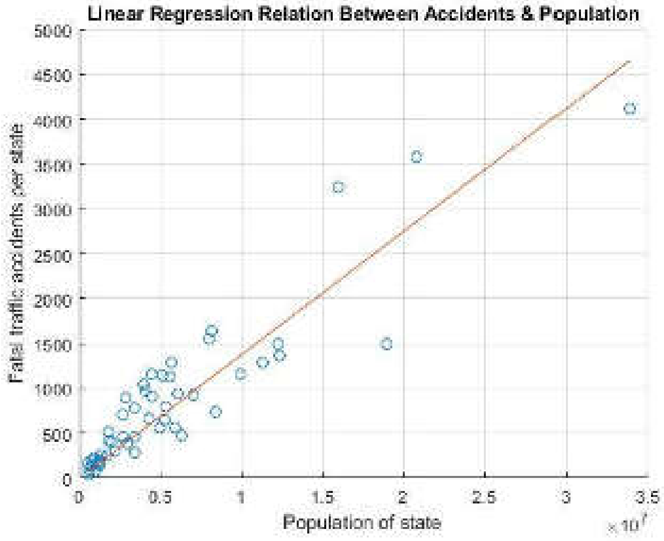
There are two supervised learning algorithms: *Linear Regression* and *Logistic Regression.*

The linear regression is about the problem of *regression,* but the logistic regression is about the problem of *classification.*

Linear Regression (Regression):

A regression algorithm is used to find *a linear model or hypothesis* that based on training data.

*The mode*l will give an estimate (by returning a scalar) on a new data (test data). Linear regression is the simplest model: It's about finding the best right that comes closest to the training data (let's see the following example).



It's about finding the linear regression relationship between the accidents per state and the population.

The predicted model by the linear regression algorithm will be:

$$\mathbf{Y = aX+b}$$

knowing that: *a* and *b* are the coefficients of the right .

In general, supervised learning allows model to learn a set of parameters from the training data until you expect the desired behavior.

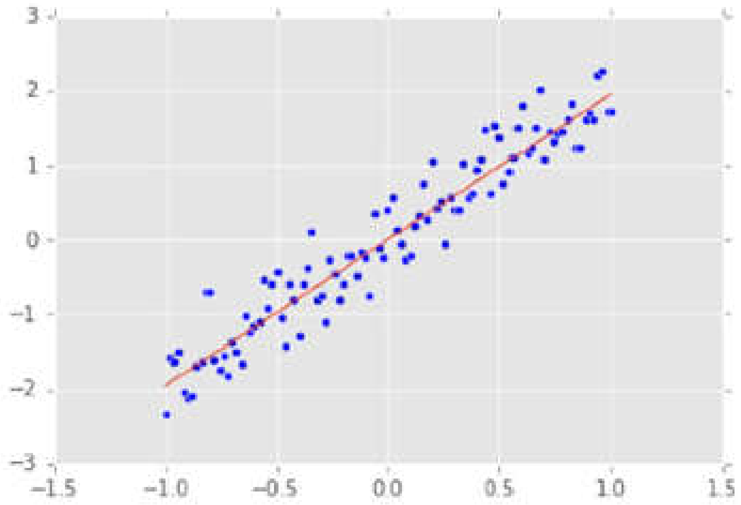
We implemented a Linear Regression model on Python:

Source code is available:

```
import theano

import theano.tensor as T
import numpy as np
import matplotlib.pyplot as plt
plt.ion()
# create artificial training data
x_train = np.linspace(-1, 1, 101)
t_train = 2 * x_train + np.random.randn(*x_train.shape) * 0.33
# plot data
plt.scatter(x_train, t_train)
# define symbolic Theano variables
x = T.scalar()
t = T.scalar()
# define model: linear regression
def model(x, w):
    return x * w
w = theano.shared(0.0)
y = model(x, w)
cost = T.mean((t - y) ** 2)
g = T.grad(cost, w)
updates = [(w, w - g * 0.01)]
# compile theano function
train = theano.function([x, t], cost, updates=updates)
# train model
for i in range(20):
    print "iteration %d" % (i + 1)
    for x, t in zip(x_train, t_train):
        train(x, t)
    print "w = %.8f" % w.get_value()
    print
# plot fitted line
plt.plot(x_train, w.get_value() * x_train)
```

We obtained the following result:

```
iteration 1
w = 0.97556865
iteration 2
w = 1.46437997
iteration 3
w = 1.70930020
iteration 4
w = 1.83201815
iteration 5
w = 1.89350631
iteration 6
w = 1.92431512
iteration 7
w = 1.93975195
iteration 8
w = 1.94748662
iteration 9
w = 1.95136210
iteration 10
w = 1.95330392
iteration 11
w = 1.95427687
iteration 12
w = 1.95476437
iteration 13
w = 1.95500864
iteration 14
w = 1.95513103
iteration 15
w = 1.95519235
iteration 16
w = 1.95522308
iteration 17
w = 1.95523847
iteration 18
w = 1.95524618
iteration 19
w = 1.95525005
iteration 20
w = 1.95525199
```



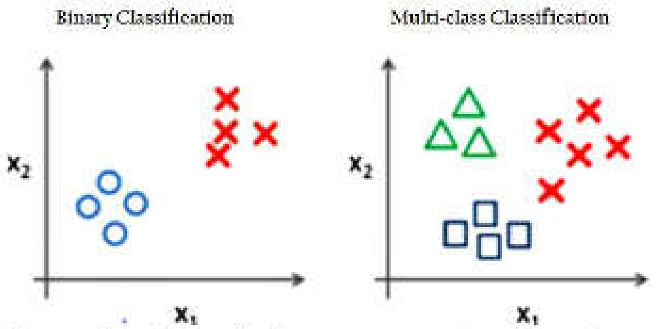
Logistic Regression (Classification):

If we have to talk about a classification problem, the variable to predict takes a discrete value.

Among the classification algorithms: Support Vector Machine (binary classification algorithm), Neural Networks (NN), Logistic Regression, ...

The purpose of these algorithms is to predict the class (or classes) from a set of data.

There are two type of classification: *Two-class classification (binary classification) and Multi-class classification.*



In the image above, the blue circles represent the first class, the red cross represent the second class and the triangles represent the third class.

We implemented a Logistic Regression model on Python (using Theano Framework):

Source code is available:

```
import theano

import numpy
x = theano.tensor.fvector('x')
target = theano.tensor.fscalar('target')
W = theano.shared(numpy.asarray([0.2, 0.7]), 'W')
y = (x * W).sum()
cost = theano.tensor.sqr(target - y)
gradients = theano.tensor.grad(cost, [W])
W_updated = W - (0.1 * gradients[0])
updates = [(W, W_updated)]
f = theano.function([x, target], y, updates=updates)
for i in xrange(10):
    output = f([1.0, 1.0], 20.0)
    print output
```

We obtained the following result:

```
0.9

8.54
13.124
15.8744
17.52464
18.514784
19.1088704
19.46532224
19.679193344
19.8075160064
```

Cost/Loss Function:

During the training parameters process, the cost function is used to evaluate whether the current weights are correctly adjusted.

This function return a value between zero (no error) and infinity (prediction errors).

It is essential when the classifier output must estimate probabilities (scores) for the classification task.

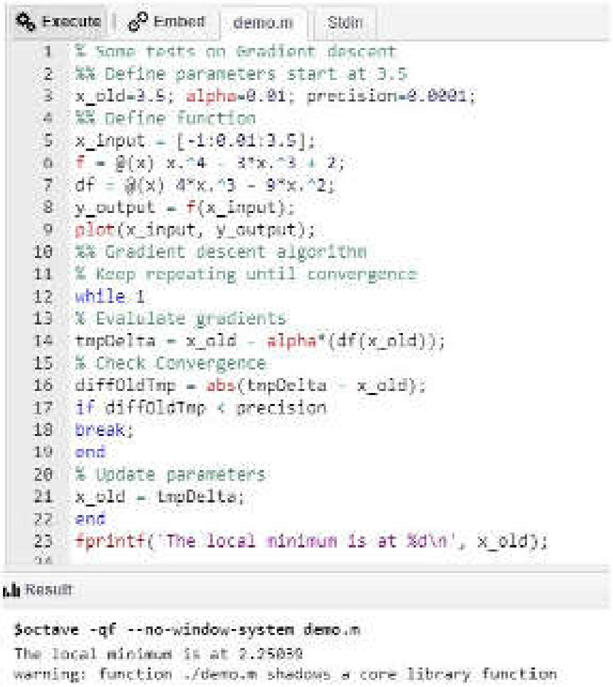
The main purpose of learning *is to minimize this function* in each iteration.

Gradient descent:

Gradient descent is a simple algorithm that is used to find the local minimum of the loss function. The goal is to minimize this function to optimize the models..

First, this algorithm needs to derive the loss function with respect to this parameters.

To help you to better understand the principle of this algorithm, we have implemented a simple example on MATLAB using the Gradient Descent to minimize a given function:



```
% Some tests on Gradient descent
%% Define parameters start at 3.5
x_old=3.5; alpha=0.01; precision=0.0001;
%% Define function
x_input = [-1:0.01:3.5];
f = @(x) x.^4 - 3*x.^3 + 2;
df = @(x) 4*x.^3 - 9*x.^2;
y_output = f(x_input);
plot(x_input, y_output);
%% Gradient descent algorithm
% Keep repeating until convergence
while 1
% Evaluate gradients
tmpDelta = x_old - alpha*(df(x_old));
% Check Convergence
diffOldTmp = abs(tmpDelta - x_old);
if diffOldTmp < precision
break;
end
% Update parameters
x_old = tmpDelta;
end
fprintf('The local minimum is at %d\n', x_old);
```

```
$octave -qf --no-window-system demo.m
The local minimum is at 2.25039
warning: function ./demo.m shadows a core library function
```

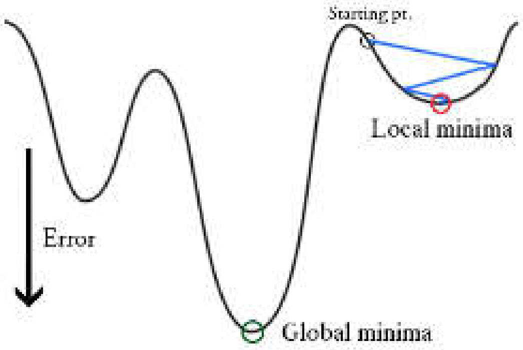
The function is:

$$f(x) = x^4 - 3.x^3 + 2$$

and the first derivative is:

$$\frac{\partial f(x)}{\partial x} = 4.x^3 - 9.x^2$$

The loss function can have several local minima when the number of parameters of model is increased. This can be a problem during training as the descent can be blocked in one of them.



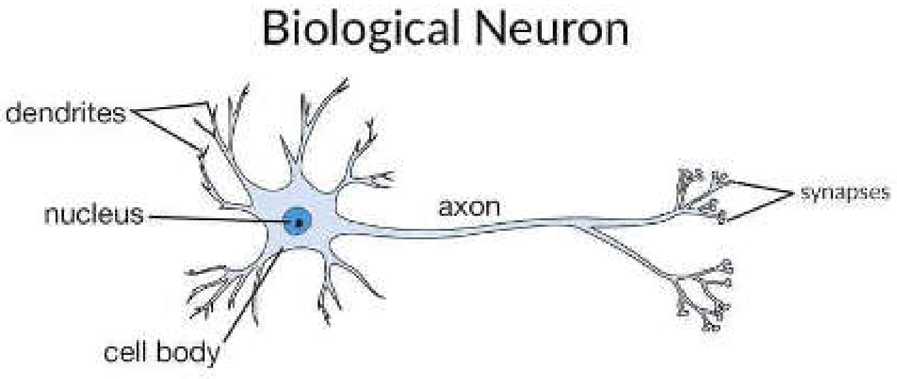
In the following section, we will recall the basics of Deep Learning, including Neural Networks.

Artificial Neural Networks

In biology, a neural network is a set of connected nerve cells that interact with each other.

The network transmits electrical signals via the axon from the synapses (outputs) of certain neurons to the dendrites (inputs) of the other neurons.

Each neuron has an activation threshold for which the sum of all input signals must exceed the threshold so that the neuron can transmit the signal through the network. Biological neural networks are very complex networks. The connection power between neurons is modeled with synaptic weight.



Deep Feedforward Network:

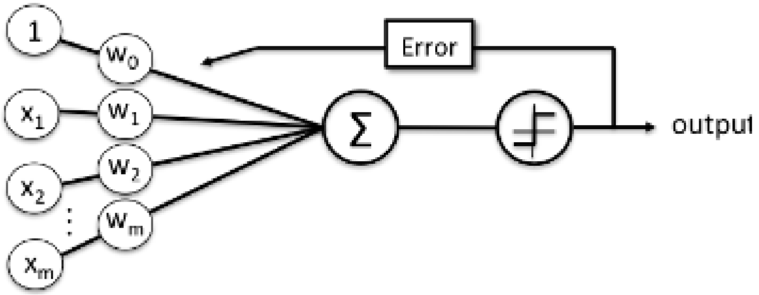
Deep Feedforward Network is often considered one of artificial neural networks. This network is very effective in a variety of tasks such as pattern recognition and classification.

The main property of a Feedforward neural network is that the information flows in the network in one direction.

Among the Deep Feedforward Networks:

Perceptron:

The perceptron was proposed by Frank Rosenblatt in 1957.This model can be regarded as the simplest neural network. It is a linear classifier. The perceptron has a single output to which all inputs are connected.

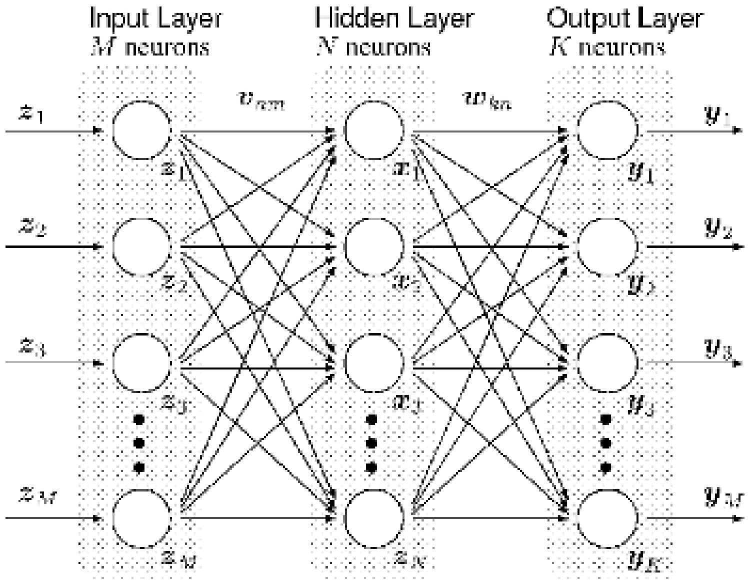


Multi-Layer Perceptron (MLP):

The perceptron can be stacked into a multi-layer network known as Multi-Layer Perceptron. The neurons in the intermediate layers are completely connected to each other.

The MLP generally consists of an input layer and an output layer with at least one hidden layer (intermediate layer).

Consider the bellow Neural network with 1 hidden layer, M input neurons, N hidden neurons and K output neuron.



Activation functions:

Activation functions are the fundamental components of artificial neural networks. Activation functions are usually non-linear functions often applied to all neurons in a hidden layer.

This section describes some commonly used activation functions and their properties.

<u>Sigmoid activation function:</u>

The sigmoid activation function is defined by the mathematical equation:

$$f(x) = \frac{1}{1 + e^{-x}}$$

It's a bounded function. It takes values between 0 and 1 (probability). This function facilitates the learning of parameters.

<u>Hyperbolic Tangent activation function:</u>

Hyperbolic Tangent activation function is defined by the mathematical equation:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

It has properties very similar to the sigmoid function except that tanH(x) takes values between -1 and 1.

<u>ReLU activation function:</u>

The ReLU (Rectified Linear Units) activation function is defined by the equation:

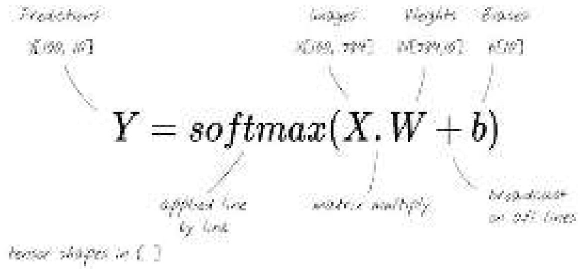
$$RELU: \quad relu(x) = \max(0, x)$$

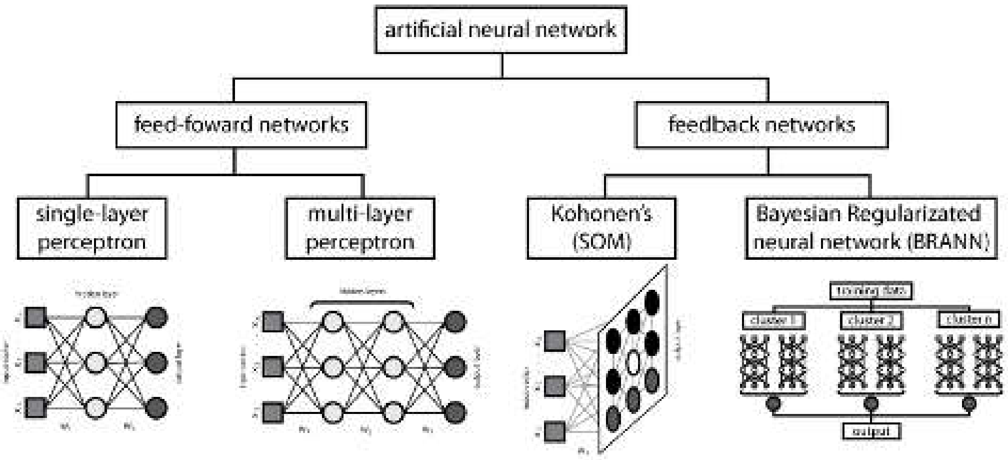
where x is the input of activation function and relu(x) is the resulting value. This function allows you to eliminate all negative values and replace them by zero. It calculates much faster than sigmoid and tanH activation functions.

<u>Softmax activation function:</u>

The Softmax function is a normalization function commonly used as an activation function for the last layer of neural network.

It is a non-linear classifier that can predict a class among a given set of classes. This function is essential when the output of neural network must estimate probabilities or scores to perform the classification.

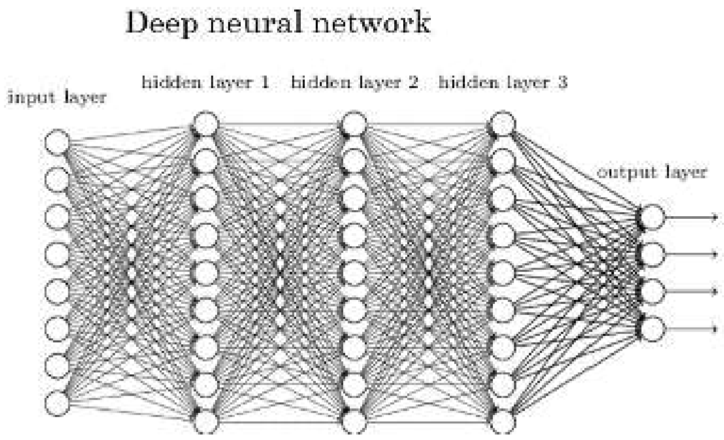




Neural networks can be classified according to their connections, the number of hidden layers, the activation functions and the training algorithm. As can be seen in the figure above, there are two neural networks categories:

- Feed-forward networks
- Feedback networks

When the number of hidden layers is greater than 2, we are talking about Deep Neural Networks, hence the emergence of the Deep Learning concept.



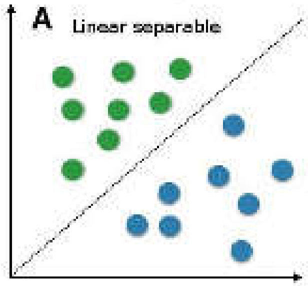
This kind of neural network is more complex and requires more data to train models (huge number of parameters).

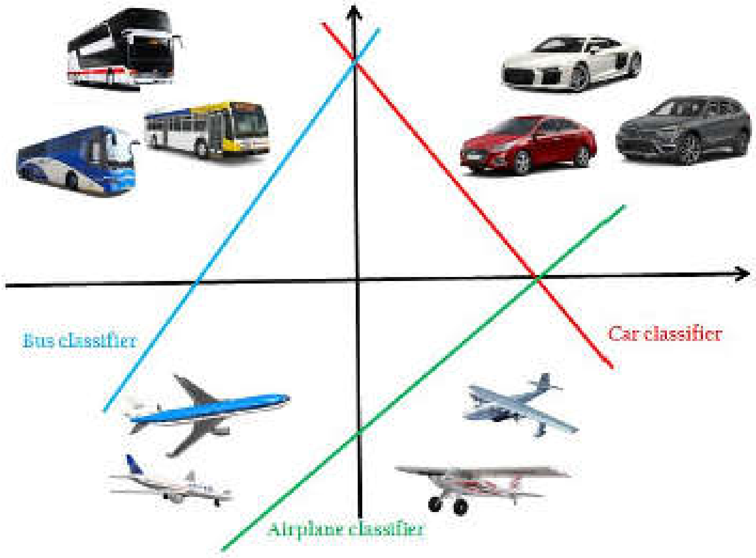
We will encounter several problem during the model training such as:

- *The increase of the prediction time,*
- *The overfitting,*
- *More local minimas to look for.*

The classification approach

We talk about *linear classification* when the classifier takes the decision by a linear separation of the characteristics (features).





Now, we will see the weight (W) and bias (b) effect.

The relationship between weight and bias is:

$$f(\vec{x}, \vec{W}, \vec{b}) = \sum_j (W_j x_j) + b$$

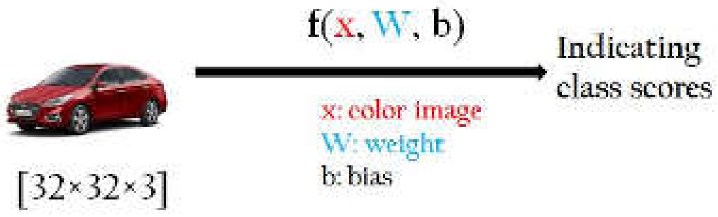
(x: input vector)

(W: Weight matrix)

(b: Bias vector)

The weight can change the separation right angle but the bias will move this right to the left or to the right.

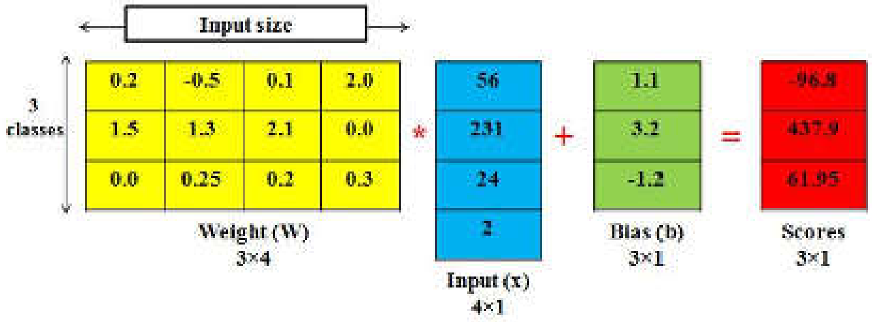
To better understand the meaning of this parameters (Weight and Bias), we will take the following example:



Let's understand what this means:

f(x, W, b): Score Function that will map from the input vector (x) to a score vector.

The weights and biases are parameters that can be trained by the learning algorithm.



Now some topics that are important on the picture above:

- The input image is characterized by a spatial dimension (height, width and depth).

- The number of rows in the weight matrix is corresponding to the number of classes to be determined (output elements).

- The score of each class is calculated by multiplying the elements of the input vector (x) with the specific line for that class.

For example:

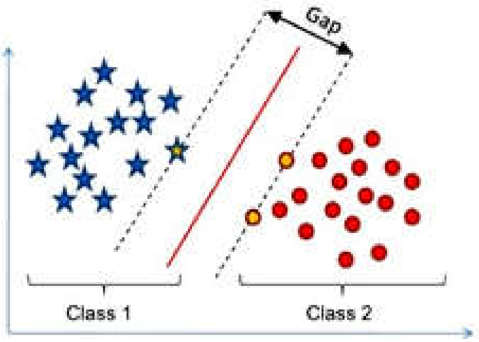
$$\textit{Score (Car)}= [0.2(56) - 0.5(231) + 0.1(24) + 2(2)] + 1.1 = -96.8$$

Support Vector Machine (SVM):

Support Vector Machine (SVM) is a supervised learning algorithm that introduced by Vapnik and Chervonenkis in 1964.

The SVM algorithm aims to find the separation between two objects classes (binary classification).

This algorithm is used to select the hyperplane that separates the features into two classes (binary response) so as to maximize the distance (Gap) between the hyperplane.



The SVM Drawbacks are:

- Difficulty of handling the large datasets,

-The treatment of multi-class problems.

Features and labels:

Now we will see the main difference between features and labels:

In Machine Learning, the characteristic (feature) represents a property of training data (ie the column name in our training dataset).

Suppose this is our training dataset:

Height	**Sex**	**Age**
44.2	M	20
72.5	F	30
555	M	41
51.5	F	51

- The features are: Height, Sex and Age.

- The label represents the output of the model after the training process.

For example:

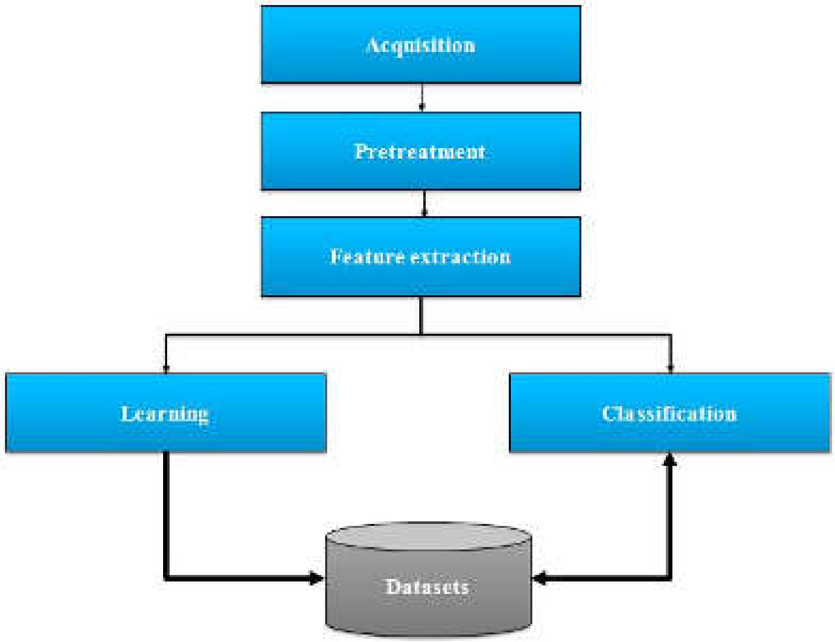
It will return the gender of a person: male or female.

Next, we will see an example of Machine Learning Workflow.

Object classification and ML

In Machine Learning, object classification is the identifying and classifying objects in images or videos.

To classify an object in image, the standard process consists of 5 steps as shown in the figure below:



Acquisition: This is the phase of digitizing images from a video sensor (camera).

Pretreatment: This is to improve the characteristics of an image, for example, by removing noise (smoothing) to facilitate subsequent processing.

Feature extraction: It is an essential phase for classification. It is a question of detecting and extracting essential information from an image, this information generally called *feature* or even *primitive*.

Common feature extraction techniques include: *Histogram of oriented gradients (HOG), Speeded-up robust features (SURF), Haar wavelets and Color histograms.*

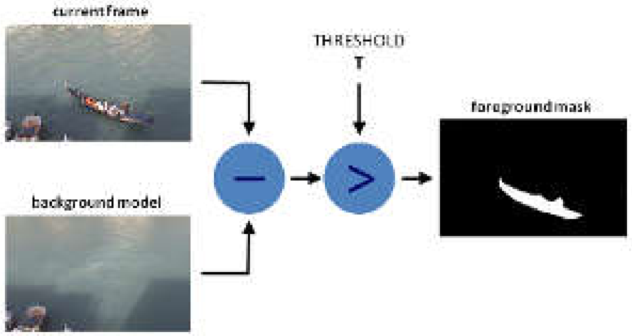
Learning: This is to train a model with the features extracted from each object.

Classification: It aims to classify objects in the image based on the extracted features.

Some Techniques and Methods for Detection of Moving Objects (before the advent of Deep Learning)

Background frame extraction:

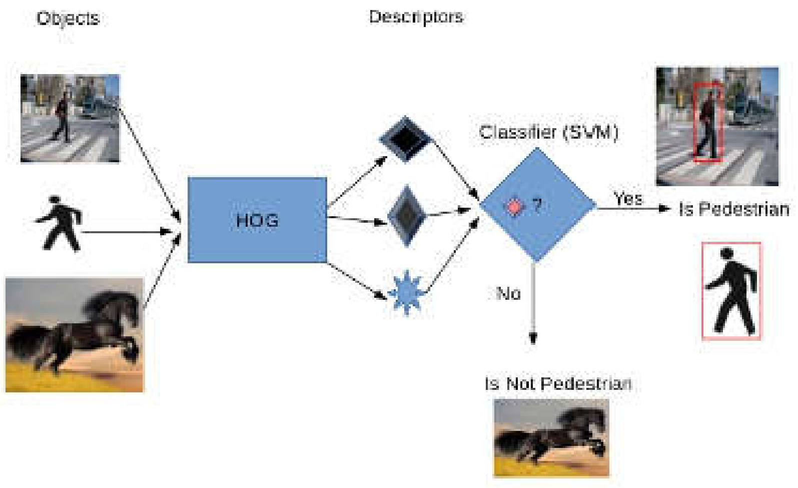
This method consists of detecting moving objects from the difference or threshold between the current and reference image (background).



HOG descriptor and SVM linear classifier:

This method is widely used for the detection of moving objects based on HOG (Histogram of Oriented Gradients) and SVM classifier. The features extracted by HOG descriptor are used for the training of SVM classifier. The SVM is used for the final classification of objects.

The figure below shows how to detect pedestrian using HOG detector and SVM classifier:



Pedestrian detection Process

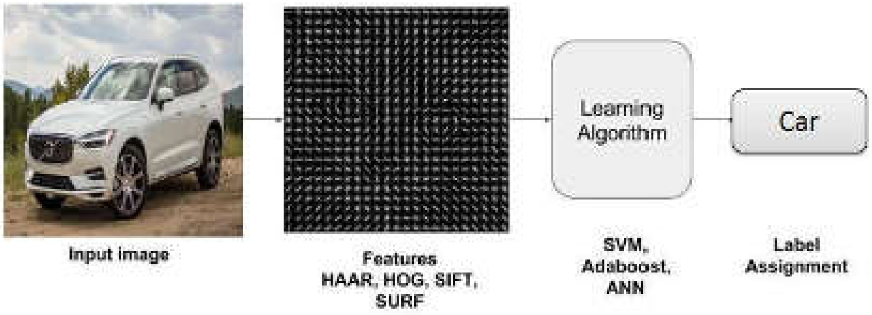
4

SIFT descriptor and SVM linear classifier:

This method allow you to use the Scale Invariant Feature Transform (SIFT) descriptor to extract points of interest in images.

The SVM used to predict whether this points of interest match the structure of our object or not in the image

The following figure illustrates the steps involved in a traditional image classifier.



Object Detection with Machine Learning and OpenCV

OpenCV (Open Source Computer Vision Library: http://opencv.org) is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms.

Now we will see how the OpenCV library helps us to detect and track the object.

First, we implemented a program on C ++ to detect some objects.

Source code is available:

```
#include <iostream>

    #include "opencv2/highgui/highgui.hpp"
    #include "opencv2/imgproc/imgproc.hpp"
    using namespace cv;
    using namespace std;

     int main( int argc, char** argv )
     {
        VideoCapture cap(0); //capture the video from web cam

        if ( !cap.isOpened() )  // if not success, exit program
        {
      cout << "Cannot open the web cam" << endl;
             return -1;
        }
```

Object Detection with Machine Learning and OpenCV

OpenCV (Open Source Computer Vision Library: http://opencv.org) is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms.

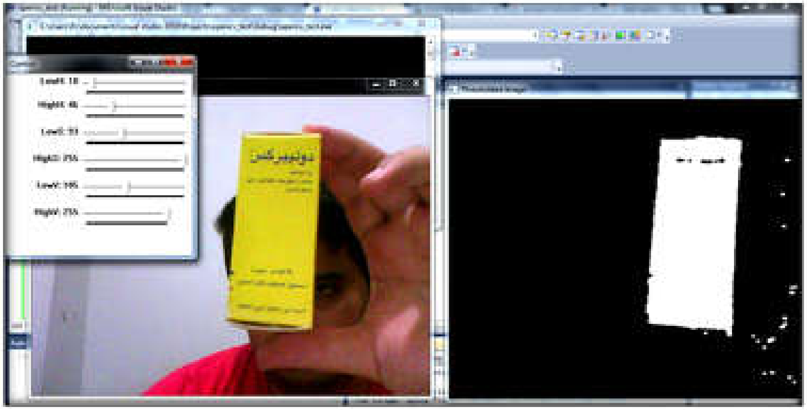
Now we will see how the OpenCV library helps us to detect and track the object.

First, we implemented a program on C ++ to detect some objects.

Source code is available:

```
#include <iostream>

    #include "opencv2/highgui/highgui.hpp"
    #include "opencv2/imgproc/imgproc.hpp"
    using namespace cv;
    using namespace std;

     int main( int argc, char** argv )
     {
        VideoCapture cap(0); //capture the video from web cam

        if ( !cap.isOpened() )  // if not success, exit program
        {
      cout << "Cannot open the web cam" << endl;
             return -1;
        }
Mat imgHSV;

       cvtColor(imgOriginal, imgHSV, COLOR_BGR2HSV); //Convert the captured
    frame from BGR to HSV

      Mat imgThresholded;

       inRange(imgHSV, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH, iHighS,
    iHighV), imgThresholded); //Threshold the image

      //morphological opening (remove small objects from the foreground)
      erode(imgThresholded, imgThresholded, getStructuringElement
    (MORPH_ELLIPSE, Size(5, 5)) );
      dilate( imgThresholded, imgThresholded, getStructuringElement
    (MORPH_ELLIPSE, Size(5, 5)) );

       //morphological closing (fill small holes in the foreground)
      dilate( imgThresholded, imgThresholded, getStructuringElement
    (MORPH_ELLIPSE, Size(5, 5)) );
      erode(imgThresholded, imgThresholded, getStructuringElement
    (MORPH_ELLIPSE, Size(5, 5)) );

       imshow("Thresholded Image", imgThresholded); //show the thresholded
    image
      imshow("Original", imgOriginal); //show the original image

      if (waitKey(30) == 27) //wait for 'esc' key press for 30ms. If 'esc'
key is pressed, break loop
           {
                cout << "esc key is pressed by user" << endl;
                break;
           }
        }


       return 0;

    }
```

We obtained the following result:



We also implemented a program on C++ using the OpneCV library for face recognition.

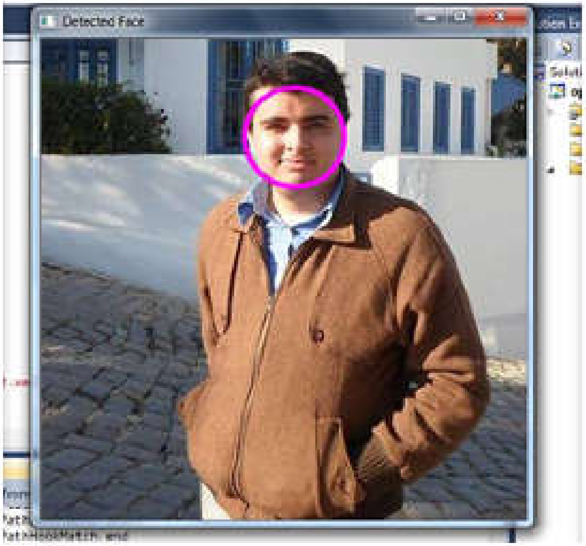
The OpenCV library makes it easy enough to detect a front face in an image using the *HAAR CASCADE Face Detector* method.

Source code is available:

```
include "opencv2/objdetect/objdetect.hpp"

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include <stdio.h>
using namespace std;
using namespace cv;
```

```
int main( )
{
    Mat image;
    image = imread("C:\\khaled2.jpg", CV_LOAD_IMAGE_COLOR);
    namedWindow( "window1", 1 );   imshow( "window1", image );

    // Load Face cascade (.xml file)
    CascadeClassifier face_cascade;
    face_cascade.load("C:
\\Opencv\\data\\Haarcascades\\haarcascade_frontalface_alt2.xml" );

    // Detect faces
    std::vector<Rect> faces;
    face_cascade.detectMultiScale( image, faces, 1.1, 2,
0|CV_HAAR_SCALE_IMAGE, Size(30, 30) );

    // Draw circles on the detected faces
    for( int i = 0; i < faces.size(); i++ )
    {
        Point center( faces[i].x + faces[i].width*0.5, faces[i].y + faces
[i].height*0.5 );
        ellipse( image, center, Size( faces[i].width*0.5, faces[i].
height*0.5), 0, 0, 360, Scalar( 255, 0, 255 ), 4, 8, 0 );
    }

    imshow( "Detected Face", image );

    waitKey(0);
    return 0;
}
```

We obtained the following result:



The cascade object detector uses the Viola-Jones algorithm to detect people faces, noses, eyes, mouth, or upper body. You can also use the Image Labeler to train a custom classifier to use with this system object.

In the next chapter, we will present the basic concepts of Deep Learning, including Convolutional Neural Networks.

Deep Learning

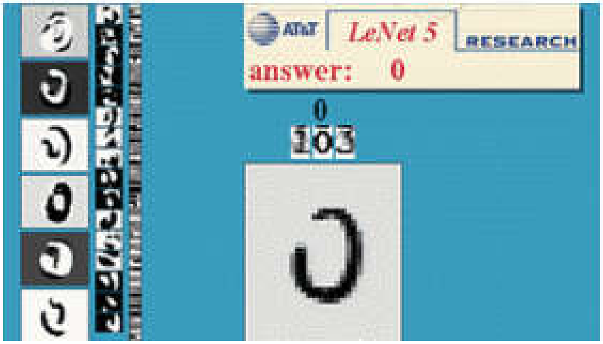
Nowadays, the field of Computer Vision has considerably evolved in recent years. Several factors have contributed to this strong growth especially with the deployment of Convolutional Neural Networks (CNNs or ConvNets), after the rapid growth of the amount of annotated data and the recent improvements on the Graphic Processing Units (GPUs), hence the appearance of Deep Learning techniques.

Formerly, the object detection systems are based on features extraction process from the conventional algorithms called detectors such as SIFT and HOG in combination with a linear classifier such as SVM or logistic regression.

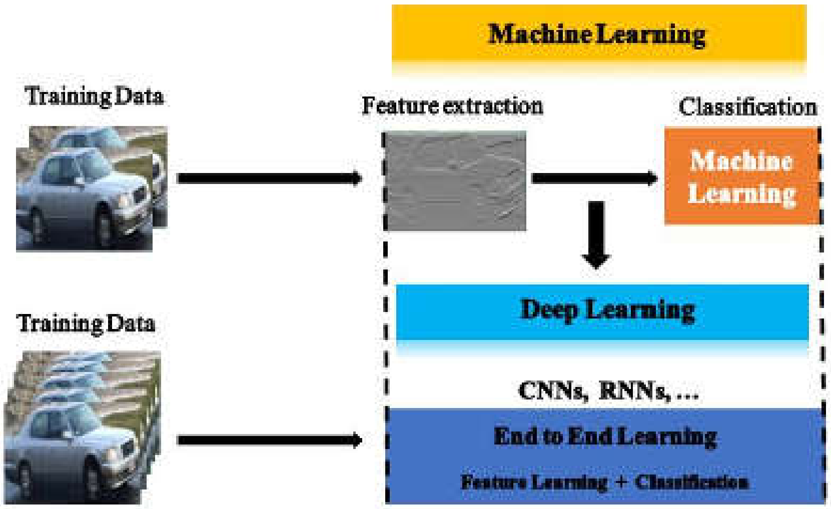
In recent years, Deep Learning has led to very good performance on a variety of tasks, such as pattern recognition, detection and classification, speech recognition, and Natural Language Processing (NLP).

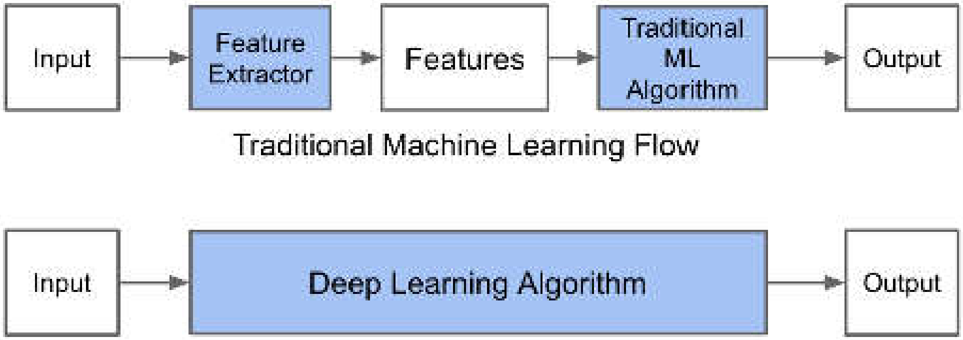
In 1998, *a pioneer* researcher in this field named Yann LeCun, presented a first model based on Convolutional Neuron Networks, called LeNet, which consists of recognizing handwritten digits.

Yann LeCun is a computer scientist with contributions in Machine/Deep Learning, computer vision, mobile robotics and computational neuroscience.



Unlike Machine Learning, Deep Learning could replace traditional feature extraction algorithms using CNNs in the form of trainable layers as shown in the figure below.

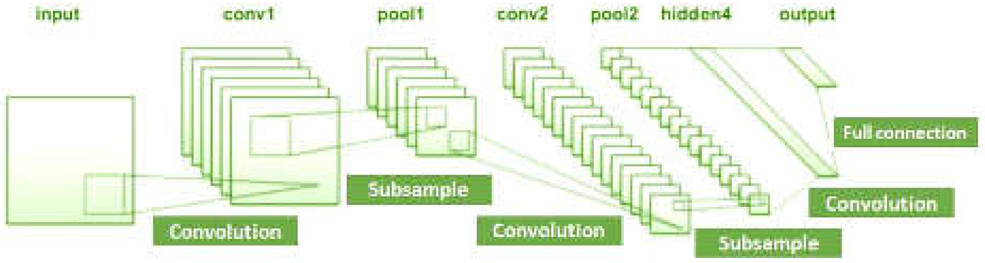




In what follows, we present the theoretical concepts and basic components of Deep Learning.

Convolutional Neural Networks

CNN is a feedforward neural network. Typically, the ConvNets consist of several layers of convolutions (following them with subsampling layers) followed by one or more fully connected layers (FCs).



This inspiration comes from the architecture of the visual perception mechanism.

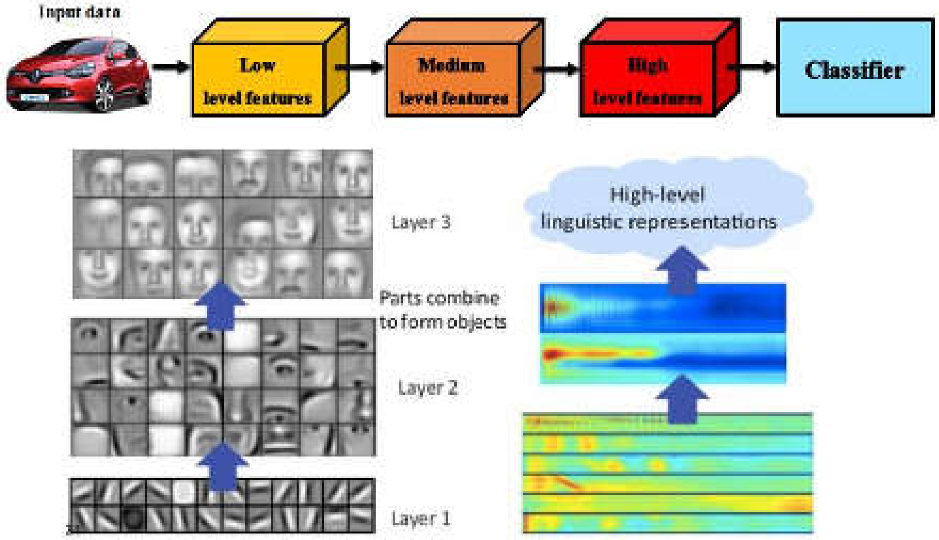
Like all artificial neural networks (ANNs), the CNNs are composed of neurons with weights and biases. Each neuron is characterized by inputs and output(s).

The connection power between two successive neurons is defined by a synaptic weight. The output of neuron's is the dot product between the input vector and the synaptic weight vector.

The CNNs architecture is composed of several intermediate layers (hidden layers), which gives them the characterization of being *DEEP* hence the name: *DEEP LEARNING*.

The depth of CNNs (the number of feature maps) generally increases from layer to layer, while their spatial dimension decreases until the desired behavior is achieved.

The whole idea of this architecture is that the input image representation is gradually increased in abstraction.



Next, we will see the backpropagation algorithm (Learning algorithm).

Back-propagation algorithm

Back-propagation is a common method for training Deep Neural Networks (CNNs), usually used in parallel with an optimization method namely Stochastic Gradient Descent (SGD).

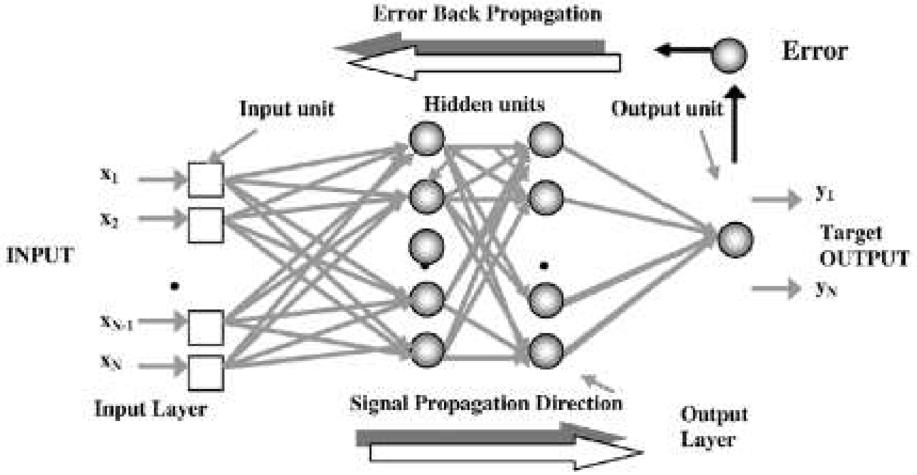
This algorithm is used to calculate the gradient (the partial derivative) of loss function (cost function) by respecting the parameters of network.

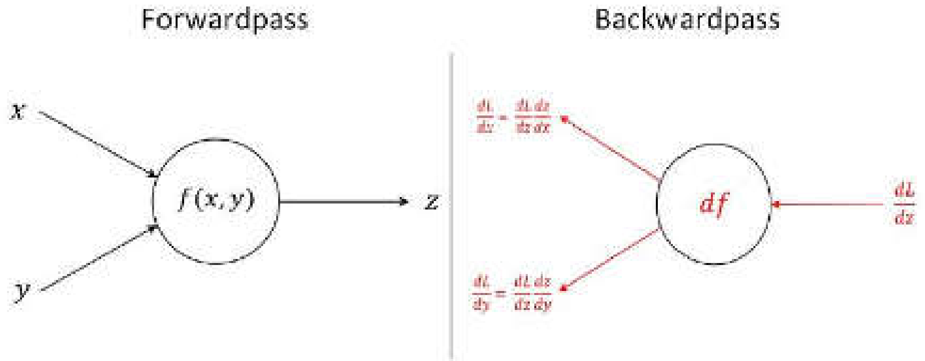
Generally, it has two phases: inference phase (forward propagation) and prediction phase (back-propagation).

- During the forward propagation phase, the outputs of network are calculated from the inputs.
- During the back-propagation phase, the parameters of network are updated with respect to the cost function.

The process is to pass data through the network, the loss calculation and the updating of parameters continues until the network has reached an acceptable performance.

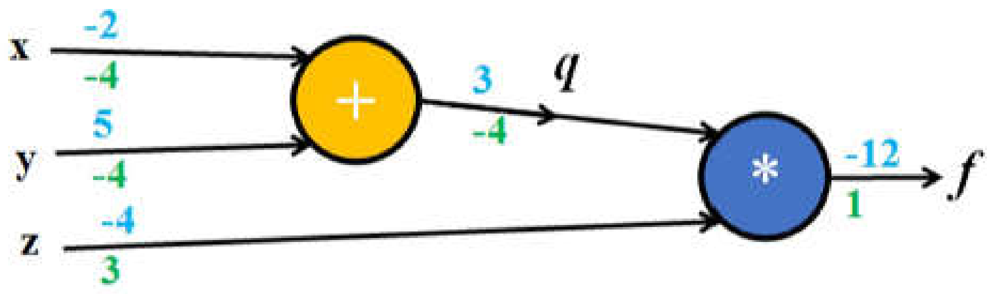
The figure below illustrates this algorithm:




Forwardpass
Backwardpass
x
y
$f(x,y)$
z
df
$\frac{dL}{dz}$
$\frac{dL}{dx} = \frac{dL}{dz}\frac{dz}{dx}$
$\frac{dL}{dy} = \frac{dL}{dz}\frac{dz}{dy}$

To calculate the gradients, we need the derivative of function *f(x, y)* and the input *dL/dz*, so multiply them.

To better understand how to calculate the gradient of a function (*f*), we will take the following example (the graph):



(1) Start from output node *f*, and consider that the gradient of *f* related to some criteria is 1 (out),

(2) *dq=(out(1) * z)*, which is -4,

(3) *dz=(out(1) * q)*, which is 3 ,

(4) The sum gate distribute it's input gradients, so dx=-4, dy=-4 .

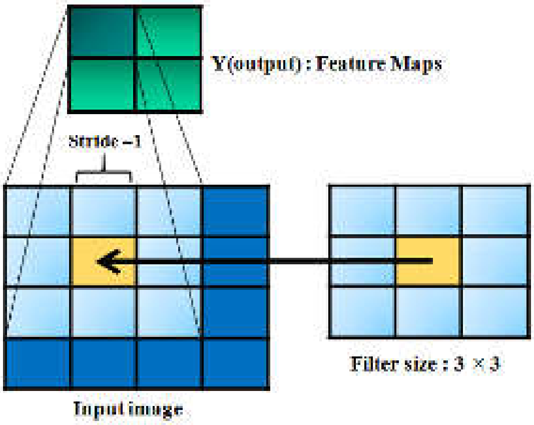
In what follows, we will present the basic components of CNNs.

Components of CNNs

There are several types of layers and connections that can be used to build deep convolutional networks, including convolution, pooling layer, etc
We describe in the following the different components of the CNNs:

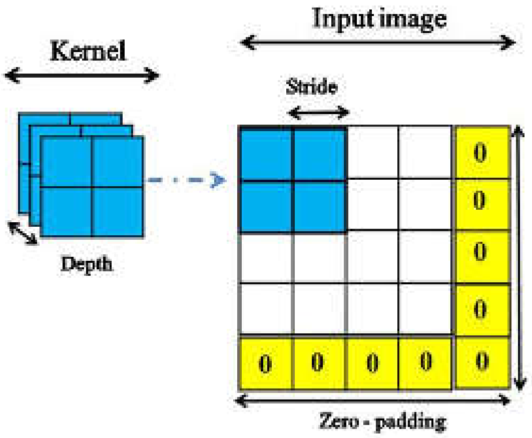
Convolution layer:

The convolution layer is defined by a kernel or filter denoted by k of dimension $k \times k$. This is a multiplication (dot product) between each kernel value and the values of the input image pixel. The convolution is designated by (*) operator and therefore: $Y - k * X$ (X: input / k: kernel).



Each convolution layer aims to produce a set of feature maps based on the number of filters to be used. .

In order to control the size of the output volume, the convolution layer is generally characterized by three hyper-parameters: *the depth* (the number of filters to be used), *the zero-padding (pad)* and *the stride.*



The depth has a significant influence on the number of parameters of network. If the size of the input volume is 32 × 32 × 3 and if the size of the receptive field (or filter size) is 5 × 5: then each neuron in the convolution layer will have weights in the region of the input volume, for a total of 5 × 5 × 3 = 75 parameters for a single filter (*depth = 1*).

The strategy for preserving spatial dimensions is to apply the *pad* by adding zeros around the border of the input volume.

The overlapping of receptive fields is defined by a *stride. The stride* is defined by the gap between two receptive fields.

By default,The convolution is applied without stride (stride = 1).

Parameters Sharing : This technique is used to control the number of parameters . This means that the same filter (weight+bias) is used for each pixel in a layer.

The advantage of this technique is to reduce the memory allocation and can improve the performance.

Pooling layer:

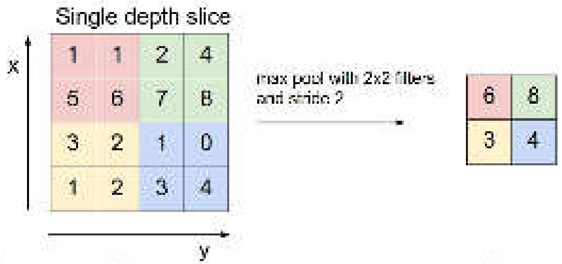
The pooling (or subsampling) layers are non-trainable layers, commonly used to reduce the number of network parameters and the spatial dimensions of feature maps.

Similar to the convolution layers, the pooling layers are associated with a *kernel* of dimension $k \times k$ and a *stride* p. Typically, the stride of the pooling layers is equal to $p = k$.

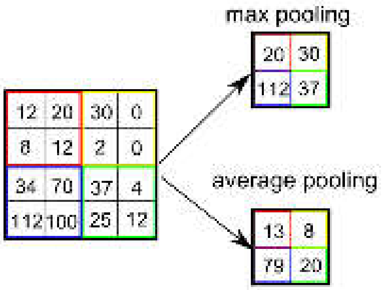
Most often, the pooling is applied by a stride equal to 2.

In what follows, we will present some types of pooling used in CNNs:

Max-Pooling: The Max-pooling layer allows you to reject the non-maximal information by performing the 'Max' operation on all feature map regions.

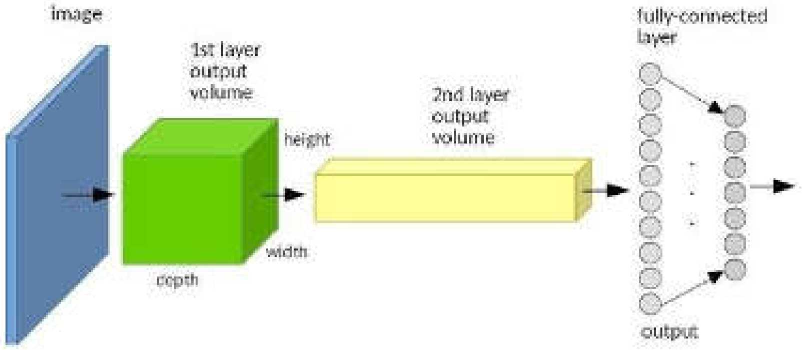


Average-Pooling: Unlike the previous case, the Average-pooling layer calculates the 'Average' on all feature map regions.



Fully connected layer:

The fully connected layers (FCs) are trainable layers. The neurons in the fully connected layer have full connections to all previous layers activations. The high-level of abstraction in CNNs is via these layers.



In what follows, we will present a case study: Handwritting recognition using CNNs.

Next, we will see some common methods of optimizing and regularizing of CNNs.

Case study: Handwritting recognition using CNNs (LeNet Model - Y. LeCun)

We have also implemented the LeNet model using the Theano framework.

Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently.

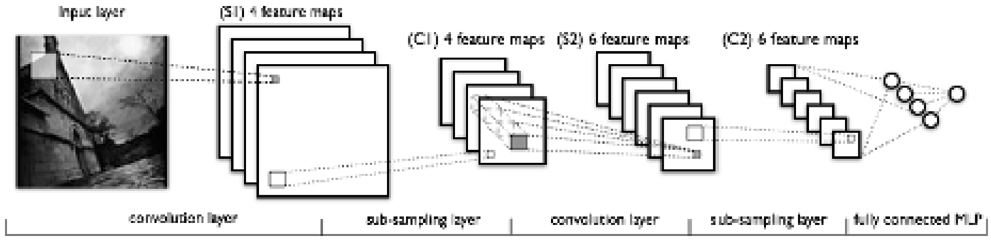
Theano features:

- tight integration with NumPy,
- transparent use of a GPU,
- efficient symbolic differentiation,
- speed and stability optimizations,
- extensive unit-testing and self-verification.

Details:

- The handwriting recognition using a 3 hidden layers.
- Each hidden layer is contains 30 nodes (neurons).
- The MNIST dataset is used for training (50.000 images).

The figure above represents the model architecture (LeNet):



The open source library '*sklearn-theano*' offers the following features:

- *Classification (Applications: image recognition, object localization.)*
- *Pretreatment (Feature extraction and normalization)*
- *Datasets*

Source Code are available:

Run.py file:

```
import pickle
import numpy as np
import mnist
import math
from PIL import Image
from numpy import array
import matplotlib.pyplot as plt
x1=784
y1=30
z1=10

theta1=pickle.load(open("theta1_new.npy","rb"))
theta2=pickle.load(open("theta2_new.npy","rb"))

images,labels = mnist.load_mnist('training')
a2=np.empty([y1+1,1],dtype='float64') #31x1
output=np.empty([z1,1],dtype='float64') #10x1
def g(z):
       return (1.0/(1.0+math.exp(-z)))

def feedforward2(input1):
       z2=np.dot(theta1,input1)
       z2=z2/100.0
       #print theta1
       for i in range (1,(y1+1)):
            a2[i] = g(z2[i-1])
       a2[0]=1.0
       z3=np.dot(theta2,a2)
       for i in range (0,z1):
            output[i] = g(z3[i])
       list1 = [a2,output]
       return list1;
```

```
im = Image.open("zero.jpg")
image1 = array(im,dtype='f')
image1 = image1.reshape(784,1)
#image1=images[142].reshape(784,1)
input1=np.zeros([785,1],dtype='float64')
for i in range(1,785):
        input1[i]=image1[i-1]/(255.0)
input1[0]=1.0

c=feedforward2(input1);
index=np.where(c[1]==max(c[1]));
#print c[1];
print "\n Number is ",index[0][0];
plt.bar(np.arange(10),c[1],1);
plt.xlabel('Digits');
plt.ylabel('Probability of recognition');
plt.title('Results');
plt.xticks(np.array([0.5,1.5,2.5,3.5,4.5,5.5,6.5,7.5,8.5,9.5]), ('0','1',
'2', '3', '4', '5','6','7','8','9'))
plt.show();
```

Mnist.py file:

```
import os, struct
from array import array as pyarray
from numpy import append, array, int8, uint8, zeros
import numpy as np
from pylab import *
def load_mnist(dataset="training", digits=np.arange(10), path="."):
    if dataset == "training":
        fname_img = os.path.join(path, 'train-images.idx3-ubyte')
        fname_lbl = os.path.join(path, 'train-labels.idx1-ubyte')
    elif dataset == "testing":
        fname_img = os.path.join(path, 't10k-images-idx3-ubyte')
        fname_lbl = os.path.join(path, 't10k-labels-idx1-ubyte')
else:
        raise ValueError("dataset must be 'testing' or 'training'")
    flbl = open(fname_lbl, 'rb')
    magic_nr, size = struct.unpack(">II", flbl.read(8))
    lbl = pyarray("b", flbl.read())
    flbl.close()

    fimg = open(fname_img, 'rb')
    magic_nr, size, rows, cols = struct.unpack(">IIII", fimg.read(16))
    img = pyarray("B", fimg.read())
    fimg.close()

    ind = [ k for k in range(size) if lbl[k] in digits ]
    N = len(ind)

    images = zeros((N, rows, cols), dtype=uint8)
    labels = zeros((N, 1), dtype=int8)
    for i in range(len(ind)):
        images[i] = array(img[ ind[i]*rows*cols : (ind[i]+1)*rows*cols ]).
reshape((rows, cols))
        labels[i] = lbl[ind[i]]

    return images,labels

#images, labels = load_mnist('training', digits=[2])
#imshow(images.mean(axis=0), cmap=cm.gray)
#show()
```

Handwrittingtriall.py file:

```
import numpy as np
import mnist
import math
from random import randint
import pickle
import random
```

```
images,labels = mnist.load_mnist('training')
#print labels[550][0]
#print images[550].reshape(784,1)
x1=784
y1=30
z1=10
theta1 = np.random.random_sample((y1,(x1+1)))
theta2 = np.random.random_sample((z1,(y1+1)))
#theta1 = np.ones([y1,x1+1],dtype='float64')
#theta2 = np.ones([z1,y1+1],dtype='float64')
alpha = 3.0
#m=200

a2=np.empty([(y1+1),1],dtype='float64') #31x1
output=np.empty([z1,1],dtype='float64') #10x1

def g(z):
       return (1.0/(1.0+math.exp(-z)))

def feedforward(trial_input):
       z2=np.dot(theta1,trial_input)
       z2=z2/100.0
       for i in range (1,(y1+1)):
             a2[i] = g(z2[i-1])
       a2[0]=1.0
       z3=np.dot(theta2,a2)
       for i in range (0,z1):
             output[i] = g(z3[i])
list1 = [a2,output]
       return list1;
image1=images[0].reshape(784,1)
```

```
#input1=np.empty([785,1],dtype='float64')
#for i in range(1,785):
#       input1[i]=image1[i-1]
#input1[0]=1.0
#a=feedforward(input1);
#print a[1]

######-----------------------------------------------------
#####################
def backprop():
        global theta1
        global theta2
        global images
        global labels
        for l in range(0,30):
                for j in range(0,5000): #number of iterations for gradient
descent
        ##loop from 1 to 1000
                        D1 = np.zeros((y1,(x1+1)))
                        D2 = np.zeros((z1,(y1+1)))
                        x=np.empty([785,1],dtype='float64')
                        lower=10*j
                        upper=lower+10
                        for i in range(lower,upper):
#for every input
                                temp1=images[i].reshape(784,1)
                                temp1=temp1/(255.0)
                                for k in range(1,(x1+1)):
                                        x[k]=temp1[k-1]
                                x[0]=1.0
## format output y
```

```
temp2 = labels[i][0]
                              if(temp2 == 0):
                                     y=np.array([[1.0],[0.0],[0.0],[0.0],
[0.0],[0.0],[0.0],[0.0],[0.0],[0.0]])
                              elif(temp2 == 1):
                                     y=np.array([[0.0],[1.0],[0.0],[0.0],
[0.0],[0.0],[0.0],[0.0],[0.0],[0.0]])
                              elif(temp2 == 2):
                                     y=np.array([[0.0],[0.0],[1.0],[0.0],
[0.0],[0.0],[0.0],[0.0],[0.0],[0.0]])
                              elif(temp2 == 3):
                                     y=np.array([[0.0],[0.0],[0.0],[1.0],
[0.0],[0.0],[0.0],[0.0],[0.0],[0.0]])
                              elif(temp2 == 4):
                                     y=np.array([[0.0],[0.0],[0.0],[0.0],
[1.0],[0.0],[0.0],[0.0],[0.0],[0.0]])
                              elif(temp2 == 5):
                                     y=np.array([[0.0],[0.0],[0.0],[0.0],
[0.0],[1.0],[0.0],[0.0],[0.0],[0.0]])
                              elif(temp2 == 6):
                                     y=np.array([[0.0],[0.0],[0.0],[0.0],
[0.0],[0.0],[1.0],[0.0],[0.0],[0.0]])
                              elif(temp2 == 7):
                                     y=np.array([[0.0],[0.0],[0.0],[0.0],
[0.0],[0.0],[0.0],[1.0],[0.0],[0.0]])
                              elif(temp2 == 8):
                                     y=np.array([[0.0],[0.0],[0.0],[0.0],
[0.0],[0.0],[0.0],[0.0],[1.0],[0.0]])
                              else:
                                     y=np.array([[0.0],[0.0],[0.0],[0.0],
[0.0],[0.0],[0.0],[0.0],[0.0],[1.0]])
```

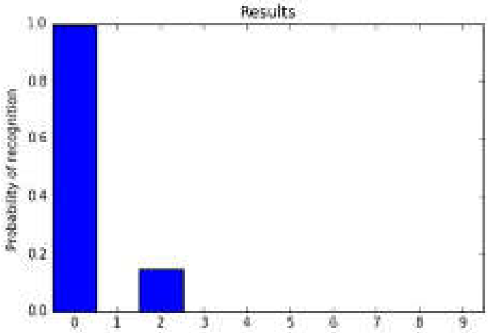
```
#preprocessing done correctly for each input
                             list1 = feedforward(x)
a2 = list1[0];
                             output = list1[1];
               #now calculate deltas
                             delta3 = output - y
                             ones = np.ones([(y1+1),1])
                             temp3 = ones - a2
                             temp4 = np.empty(((y1+1),1),dtype='float64')
                             for k in range(0,(y1+1)):
                                    temp4[k] = a2[k] * temp3[k]
                             temp5 = np.dot(np.transpose(theta2),delta3)
                             temp6 = np.empty(((y1+1),1),dtype='float64')
                             for k in range(0,(y1+1)):
                                    temp6[k] = temp5[k] * temp4[k]
                             delta2 = temp6
               ##--------not sure of this-----
                             delta2_new = np.empty((y1,1),dtype='float64')
               #because of dimension problem in D later, I am removing the
first row of delta2 which is always zero.
               #this sort of makes sense because delta2[0] represents error
of bias unit.
                             for k in range(0,y1):
                                    delta2_new[k]=delta2[k+1]
##--------------------------------

               # now for each input, we have calculated errors delta3 and
delta2..we need to get the sums of errors for all inputs. D1 and
                #  D2 for layers respectively.
                             D1=D1 + np.dot(delta2_new,np.transpose(x))
                             D2=D2 + np.dot(delta3,np.transpose(a2))
```

```
#outside loop..now we have sum of errors as D1 and D2 and we use this in
gradient descent to update theta values

                      temp_matrix1 = theta1 - (((alpha)/(10.0))*D1)
                      temp_matrix2 = theta2 - (((alpha)/(10.0))*D2)
                      theta1 = temp_matrix1
                      theta2 = temp_matrix2

training_data=zip(images,labels)
        random.shuffle(training_data)
        images,labels = zip(*training_data)

backprop();
#print theta1
pickle.dump(theta1,open("theta1_new.npy","wb"))
pickle.dump(theta2,open("theta2_new.npy","wb"))
image1=images[4].reshape(784,1)
image2=images[5].reshape(784,1)
input1=np.empty([785,1],dtype='float64')
input2=np.empty([785,1],dtype='float64')
for i in range(1,785):
        input1[i]=image1[i-1]/255.0
        input2[i]=image2[i-1]/255.0

input1[0]=1.0
input2[0]=1.0
c=feedforward(input1);
print c[1]
c1=feedforward(input2);
print c1[1]
```

We obtained the following results:

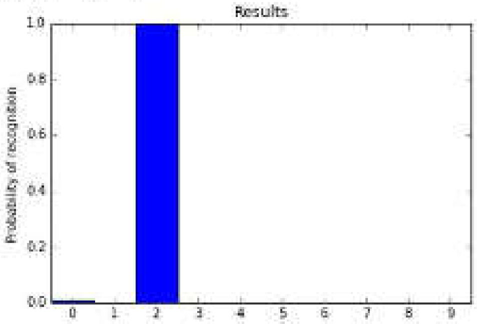
1. `im = Image.open("zero.jpg") : Input Image`

```
Number is 0
```



2. `im = Image.open("twho.jpg") : Input Image`

```
Number is 2
```



Datasets

Now we will see the different public datasets according to its main area of application.

Datasets: These data sets can be used for comparative study of Deep Learning algorithms.

Natural Images Datasets:

1. **MNIST**: handwritten digits (**http://yann.lecun.com/exdb/mnist/**)

2. **NIST**: similar to MNIST, but larger

3. **Perturbed** NIST: a dataset developed in Yoshua's class (NIST with tons of deformations)

4. **CIFAR10 / CIFAR100**: 32×32 natural image dataset with 10/100 categories (**http://www.cs.utoronto.ca/~kriz/cifar.html**)

5. **Caltech 101**: pictures of objects belonging to 101 categories (**http://www.vision.caltech.edu/Image_Datasets/Caltech101/**)

6. **Caltech 256**: pictures of objects belonging to 256 categories (**http://www.vision.caltech.edu/Image_Datasets/Caltech256/**)

7. **Caltech Silhouettes**: 28×28 binary images contains silhouettes of the Caltech 101 dataset

8. **STL-10** dataset is an image recognition dataset for developing unsupervised feature learning, deep learning, self-taught learning algorithms. It is inspired by the **CIFAR-10 dataset** but with some modifications. **http://www.stanford.edu/~acoates//stl10/**

9. **The Street View House Numbers (SVHN) Dataset - http://ufldl.stanford.edu/housenumbers/**

10. **NORB:** binocular images of toy figurines under various illumination and pose (**http://www.cs.nyu.edu/-ylclab/data/norb-v1.0/**)

11. **Imagenet:** image database organized according to the WordNethierarchy (**http://www.image-net.org/**)

12. **Pascal VOC:** various object recognition challenges (**http://pascallin.ecs.soton.ac.uk/challenges/VOC/**)

13. **Labelme:** A large dataset of annotated images, **http://labelme.csail.mit.edu/Release3.0/browserTools/php/dataset.php**

14. **COIL 20:** different objects imaged at every angle in a 360 rotation(**http://www.cs.columbia.edu/CAVE/software/softlib/coil-20.php**)

15. **COIL100:** different objects imaged at every angle in a 360 rotation (**http://www1.cs.columbia.edu/CAVE/software/softlib/coil-100.php**)

Faces Datasets:

1. **Labelled Faces in the Wild:** 13,000 images of faces collected from the web, labelled with the name of the person pictured (**http://vis-www.cs.umass.edu/lfw/**)

2. **Toronto Face Dataset**

3. **Olivetti**: a few images of several different people (**http://www.cs.nyu.edu/-roweis/data.html**)

4. **Multi-Pie:** The CMU Multi-PIE Face Database (**http://www.multipie.org/**)

5. **Face-in-Action (http://www.flintbox.com/public/project/5486/)**

6. **JACFEE: Japanese and Caucasian Facial Expressions of Emotion (http://www.humintell.com/jacfee/)**

7. **FERET: The Facial Recognition Technology Database (http://www.itl.nist.gov/iad/humanid/feret/feret_master.html)**

8. **mmifacedb**: MMI Facial Expression Database (http://www.mmifacedb.com/)

9. **IndianFaceDatabase**: http://vis-www.cs.umass.edu/-vidit/IndianFaceDatabase/) (e.g. The Yale Face Database (http://vision.ucsd.edu/content/yale-face-database) and The Yale Face Database B (http://vision.ucsd.edu/-leekc/ExtYaleDatabase/ExtYaleB.html)).

Text Datasets:

1. **20 newsgroups**: classification task, mapping word occurences to newsgroup ID (http://qwone.com/-jason/20Newsgroups/)

2. Reuters (RCV*) Corpuses: text/topic prediction (http://about.reuters.com/researchandstandards/corpus/)

3. **Penn Treebank** : used for next word prediction or next character prediction (http://www.cis.upenn.edu/-treebank/)

4. **Broadcast News**: large text dataset, classically used for next word prediction (http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC97S44)

5. Wikipedia Dataset

6. Multidomain sentiment analysis dataset: http://www.cs.jhu.edu/-mdredze/datasets/sentiment/

Speech Datasets:

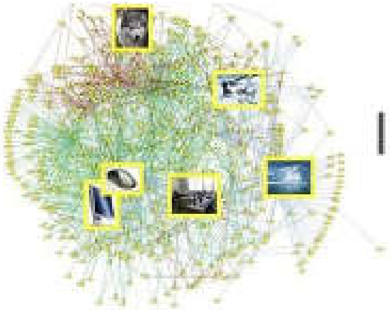
1. TIMIT Speech Corpus: phoneme classification (http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC93S1)

2. **Aurora** : Timit with noise and additional information

ImageNet Dataset:

Currently, ImageNet is the largest dataset in the field of Machine/Deep Learning and pattern recognition for object detection and classification. It contains more than 3.2 million images for 1000 objects categories. ImageNet will become a useful resource for researchers, educators, students and all of you who share our passion for pictures. The ImageNet project is inspired by a growing sentiment in the image and vision research field – the need for more data (more details: image-net.org).

.



IMAGENET

The Large Scale Visual Recognition Challenge (ILSVRC) evaluates algorithms for object localization/detection from images/videos at scale:

- *Object localization for 1000 categories.*

- *Object detection for 200 fully labeled categories.*

- *Object detection from video for 30 fully labeled categories.*

Deep Learning Frameworks

Today, the most common Deep Learning frameworks are:

(1) Caffe

(2) Torch

(3) TensorFlow

(4) Theano

(5) CNTK

Caffe Framework:

One of the most basic feature of caffe is that is easy to train models.

Features:

- Good Performance, allows training with multiple GPUs,
- Implementation for CPU and GPU,
- Source code is easy to read,
- Allow layer definition in Python and has bidings for Python and Matlab,
- Allows network definition with text language (No need to write code),
- Fast dataset access through LMDB,
- Allows network vizualization.

Torch Framework:

Really good for research, the problem is that use a new language called Lua.

Features:

- Flexible,

- Very easy source code,

- Easy biding with C/C++,

- Web interface (Digits).

TensorFlow Framework:

Features:

- Flexible

- Good for RNN

- Allow distributed training

- Tensorboard for signal visualization

- Python Numpy

CNTK Framework:

Features:

- Flexible

- Good for RNN

- Allows distributed training

Caffe Microsoft CNTK DEEPLEARNING4J

dmlc mxnet TensorFlow theano

torch

The table below represents a comparative analysis between Deep Learning Frameworks:

	Caffe	Torch	Theano	TensorFlow
Language	C++, Python	Lua	Python	Python
Pretrained	Yes ++	Yes ++	Yes (Lasagne)	Inception
Multi-GPU: Data parallel	Yes	Yes cunn. DataParallelTable	Yes platoon	Yes
Multi-GPU: Model parallel	No	Yes fbcunn.ModelParallel	Experimental	Yes (best)
Readable source code	Yes (C++)	Yes (Lua)	No	No
Good at RNN	No	Mediocre	Yes	Yes (best)

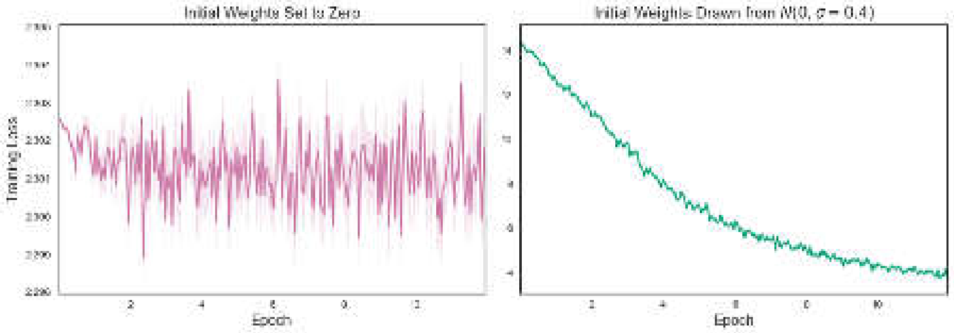
Optimization approaches

The loss function is strongly non-convex as well as the deep network is usually composed of tens of millions of parameters so the optimization algorithms should be applied.

Initialization of weights and bias:

The Deep Neural Network has a huge number of parameters and its loss function is non-convex, which makes training very difficult.

Weight initialization is one of the most important conditions for achieving rapid convergence in training and avoiding the problem of vanishing gradient. However, the bias may be initialized to zero, but the weight must be initialized by a random values so that the network converges faster.



Stochastic Gradient Descent (SGD):

Stochastic Gradient Descent aims to optimize models. This algorithm makes it possible to update the parameters of network for each iteration in order to minimize the loss function.

Mathematically, the following formula makes it possible to update the weight *W* at *t + 1* iteration:

$$W_{t+1} = W_t + V_{t+1}$$

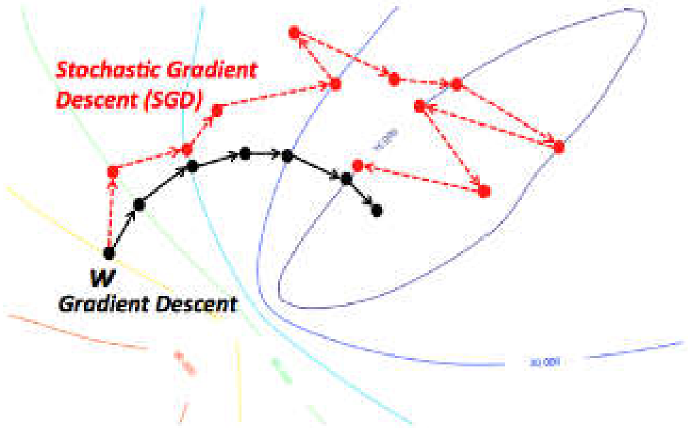
Knowing that:

W_{t+1}: represents the adjusted weight,

W_t: represents the current weight,

V_{t+1}: represents the update value.

This algorithm can be much faster because the number of updates is much larger.

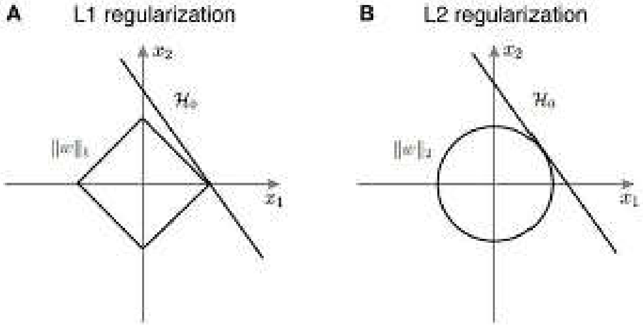


Regularization approaches

Overfitting is a serious problem in the case of deep CNN's , which can generally be reduced by regularization approaches. Overfitting means that performance in the training is much better than performance in the test.
In this section, we explore some regularization techniques.

L1 and L2 Regularization:

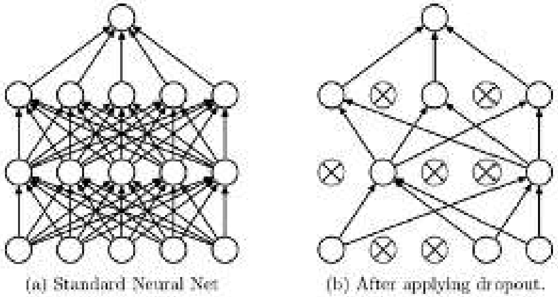
L1 and L2 Regularization are two interrelated techniques that can be used by training algorithms to reduce the overfitting of model.
This technique makes it possible to regularize the loss function by penalizing certain configurations of network parameters. L2 penalties are much better for minimizing prediction error rather than L1 penalties.



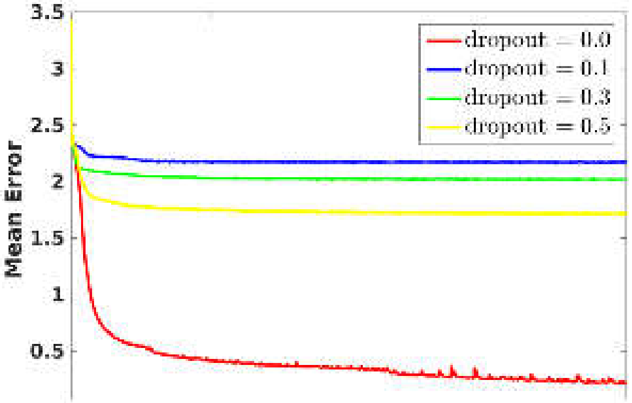
Dropout:

Dropout was introduced by Nitish Srivastava (2014). This approach has been proven effective in reducing the overfitting problem.

Dropout is typically applied in fully connected layers, which consists of zeroing the output of each neuron in a hidden layer by a probability of 0.5.



Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.



Normalization LRN:

The Local Response Normalization (LRN) layer was introduced by Krizhevsky (2012). It serves to perform a "*lateral inhibition*" by normalizing the input regions.

This layer is useful when we use the ReLU activation function. ReLU is an unsaturated activation function, so we need the LRN layer to normalize it.

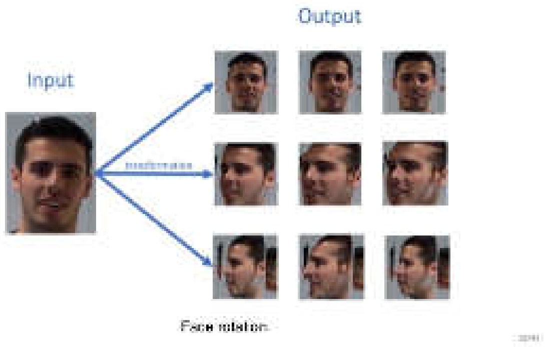
Data Augmentation:

Since deep neural networks must be trained on a large number of training images to obtain satisfactory performances.

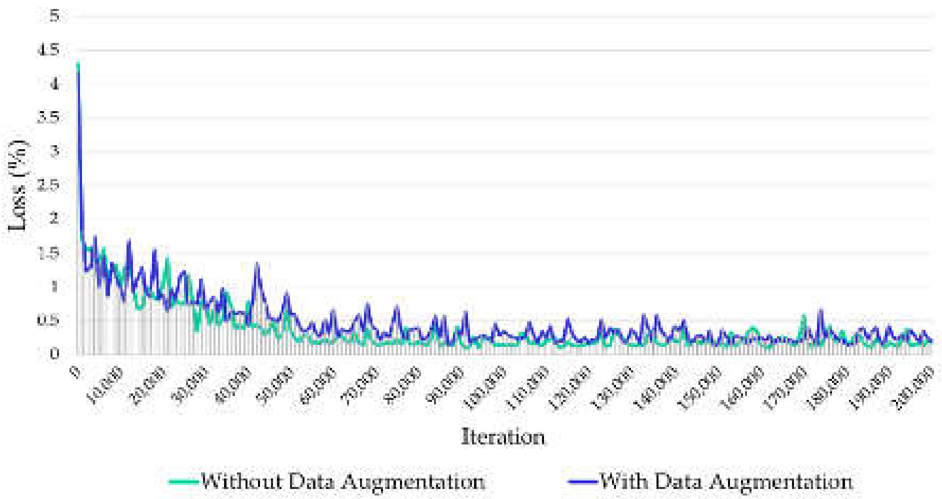
To build a powerful classifier using very little training dataset, data augmentation is usually required to increase the performance of deep networks. Data augmentation artificially creates training images through different ways of processing such as random rotation, shifts, shear and flips, etc. More precisely, this method is used to create new images by applying some transformations on the input data.

In addition, data augmentation should be done when training a deep network.

The figure below illustrates an example of processing (transformation) to ensure the data augmentation:



The figure below shows the effect of data augmentation on the training cost:



How To Improve Deep Learning Performance?

There are several methods and techniques used to improve the performance of Deep Learning models (Deep Neural Networks).
In this section we will see some approaches and algorithms that help us to increase the perfomance and the robustness of classifiers.
We can divide the approaches into 2 sub-themes:

(1) Data effect

(2) Architectural effect

Data effect:

The processing of training data has positive effects on the improvement of the models performance such as:

Get More Data:

The quality of your models is generally constrained by the quality of your training data. Deep learning and other modern nonlinear Machine Learning techniques get better with more data. It is one of the main points that make Deep Learning so exciting.

Invent More Data:

Deep Learning algorithms often perform better with more data. If you can't reasonably get more data, you can invent more data.

If your data are images, create randomly modified versions of existing images.

Rescale the Data:

The activation functions are used to rescale and normalize the training data.

For example:

If you are using sigmoid activation functions, rescale your data to values between 0 and-1. If you're using the Hyperbolic Tangent (tanh), rescale to values between -1 and 1.

Architectural effect:

We have always been wondering what happens if we can implement more hidden layers. In theory, it has been established that many of the functions will converge in a higher level of abstraction. So it seems more layers better results.

From my experience, I have found that when the hidden layers increase, it can lead to better accuracy.

But it is not the base of the thumb. You only have to test it with a different number of layers.

The table below illustrates a comparative study between the CNN architectures of the literature:

	#conv. layers	#MACCs [millions]	#params [millions]	#activations [millions]	ImageNet top-5 error
ZynqNet CNN	18	530	2.5	8.8	15.4%
AlexNet	5	1 140	62.4	2.4	19.7%
Network-in-Network	12	1 100	7.6	4.0	~19.0%
VGG-16	16	15 470	138.3	29.0	8.1%
GoogLeNet	22	1 600	7.0	10.4	9.2%
ResNet-50	50	3 870	25.6	46.9	7.0%
Inception v3	48	5 710	23.8	32.6	5.6%
Inception-ResNet-v2	96	9 210	31.6	74.5	4.9%
SqueezeNet	18	860	1.2	12.7	19.7%
SqueezeNet v1.1	18	390	1.2	7.8	19.7%

As can be seen from the table above, the number of hidden layers, ie the convolution layers, has a significant influence on the performance and accuracy of the models.

From CNN to Mask R-CNN (Classification to Segmentation)

First, we need to know what is the difference between Image Classification, Localization, Recognition and Segmentation?

Image Classification:

Categorize an image based on the objects inside it.
Image classification is the task of classifying what this image contains from a set of predefined classes.

Image Localization:

This is to predict the area (region) that contains the object in the image.

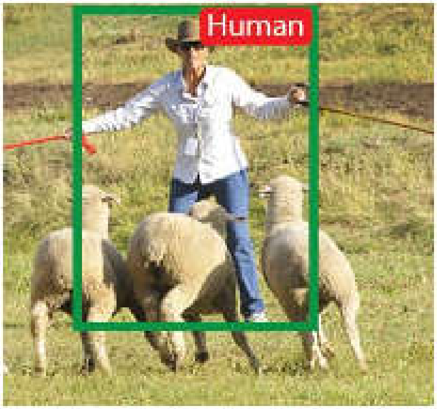
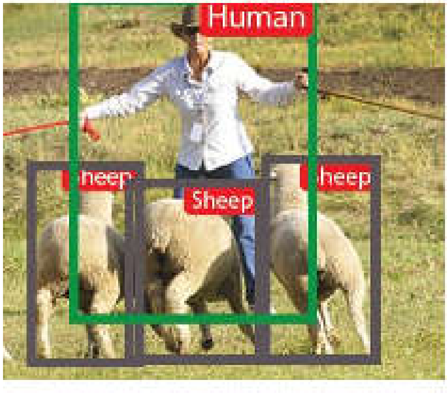


Image Recognition:

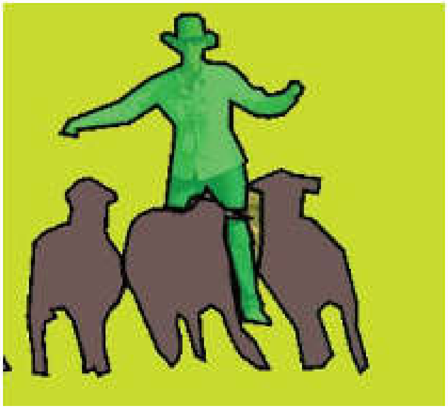
This task is used to locate and classify all objects appearing in the image.

Recognition - Localization + Classification.



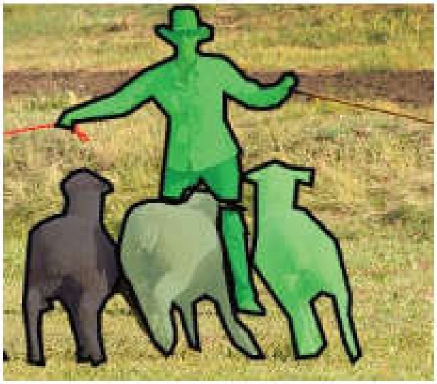
Semantic Segmentation:

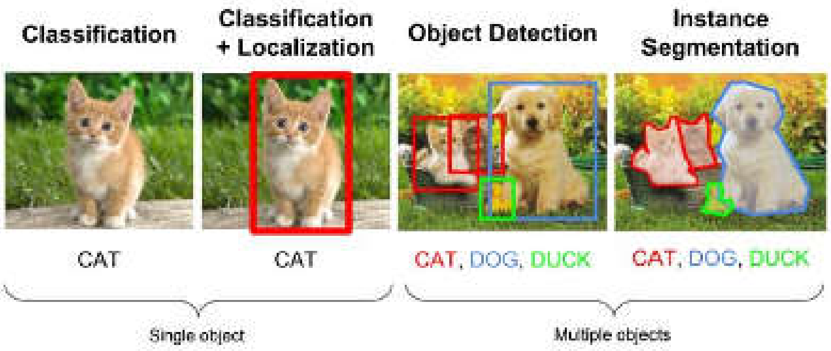
Label each pixel of an image by *the object class* that it belongs to.



Instance Segmentation:

Label each pixel of an image by *the object class* and *object instance* that it belongs to.





In this section we will learn about the best-known CNNs architectures in the literature, including:

- *AlexNet model* ***(2012)***

- *ZFNet model* ***(2013)***

- *R-CNN model* ***(2013)***

- *VGGNet Model* ***(2014)***

- *GoogleNet Model* ***(2014)***

- *Fast R-CNN model* ***(2015)***

- *ResNet model* ***(2015)***

- *Faster R-CNN model* ***(2015)***

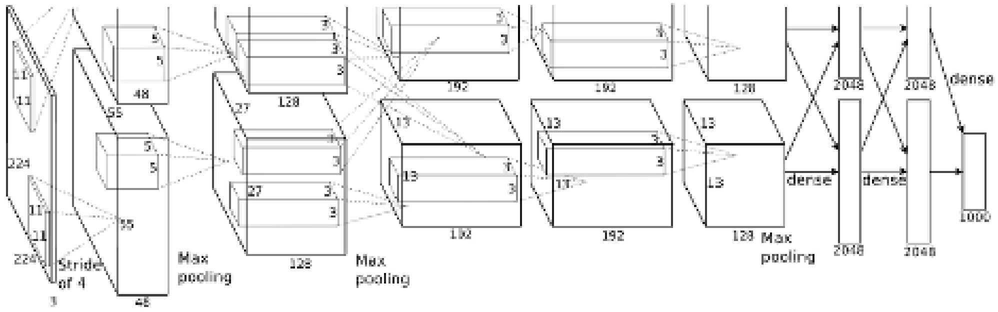
- *Mask R-CNN model* ***(2017)***

AlexNet model:

The Convolutional Neural Networks popularized by the AlexNet model (Krizhevsky, 2012) have achieved impressive results in image classification.

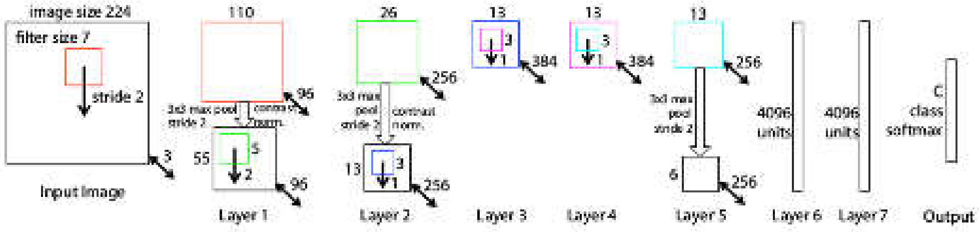
In 2012, this model won the ImageNet competition, which launched the current wave of interest for neural networks.

It combined with *data augmentation* (regularization), *ReLU* (activation function) , *dropout* (regularization), and *GPU* implementation.



ZFNet model:

ZFNet is the winner of ILSVRC 2013, which is essentially AlexNet with a minor modification on the architecture. (kernel size of convolution layers).



ZF Net Architecture

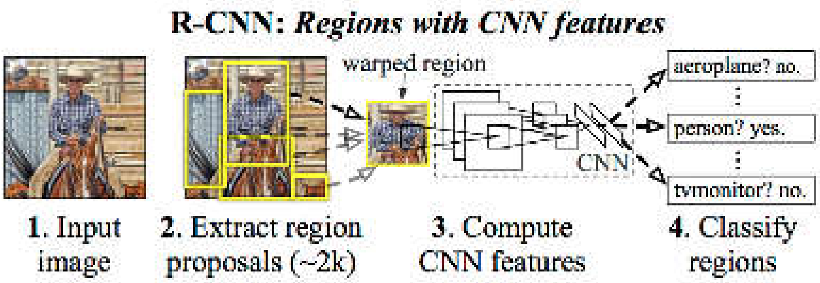
R-CNN model:

In recent years, region-based CNNs methods have become the focus actions on in the field of Deep Learning. The tasks of detecting and locating objects in images are among the most difficult problems in computer vision.

Some recent works based on this approach, attempts to improve the precision and accuracy by reducing the number of regions (RoI), which could increase the accuracy and performance.

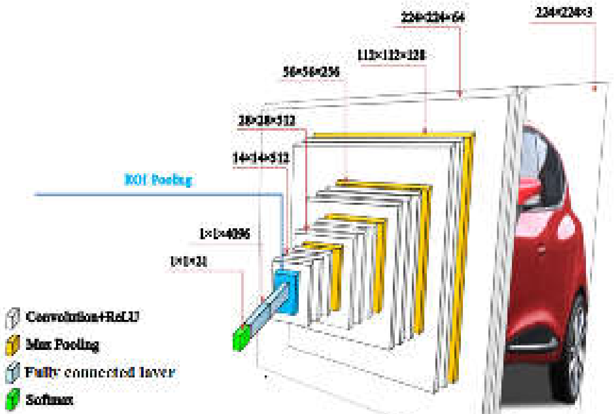
R-CNN (Girshick, 2014) is a network that combines the proposed regions (rois) with the features extracted by a Convolution Neural Network.

R-CNN is composed of 3 modules: the first module allows to generate regions (about 2000 regions) via the Selective Search technique, the second module is a CNN that extracts a fixed-length feature vector for each region and the third is a SVM classifier that used to classify these regions.



VGGNet Model:

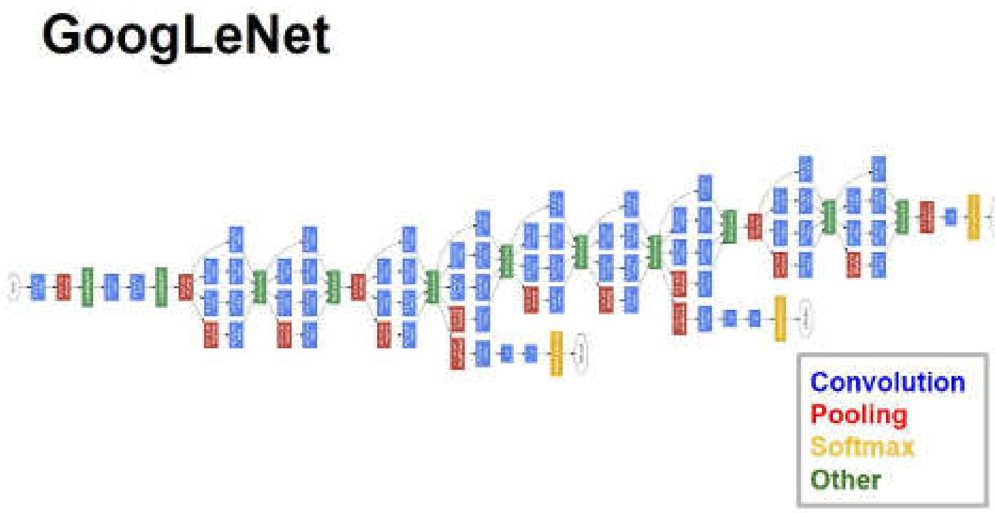
The VGG network contains 13 convolution layers and 3 fully connected layers. Each level contains feature maps produced by intermediate layers (convolution, pooling, etc.). This model has shown that the network depth is essential for good performance.



<u>GoogleNet Model:</u>

GoogleNet is the winner of the Imagenet Large-Scale Visual Recognition Challenge in 2014. It introduces new techniques and insights for image classification.

The figure above do the following instructions:



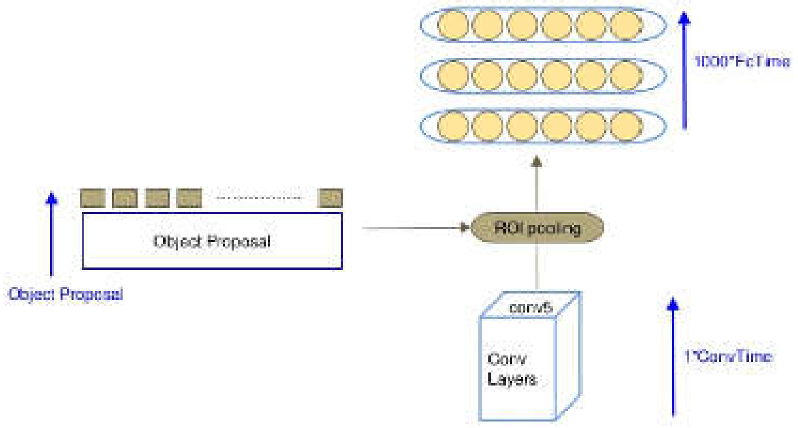
- Obviously, GoogLeNet has a modular structure : easy to add and modify,
- The network is still using Dropout (regularization),
- It tends to increase the depth of the model (number of layers= 22) in order to obtain high-quality models,

- Too many parameters, easy to overfit if the training dataset is limited.

<u>Fast R-CNN model:</u>

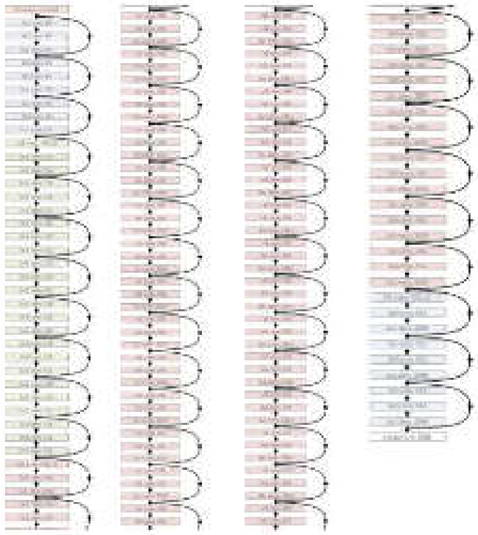
Fast R-CNN receives regions via the Selective Search technique. These regions will be sent to ROI Pooling layer which resizes them to a fixed size, and then move them to the fully connected layer.

In addition, Fast R-CNN replaced the SVM classifier with the Softmax classifier for the classification of regions .



ResNet model:

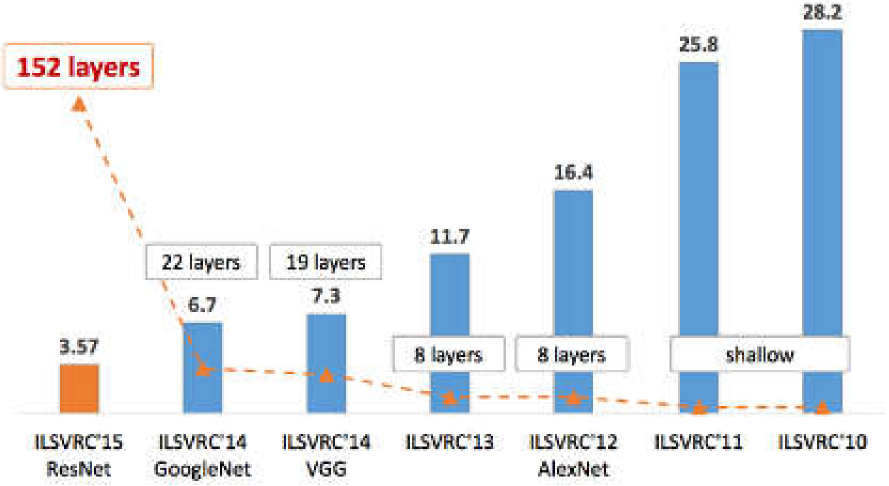
The structure of the network is shown in the figure below:



The depth of the network is crucial for the performance of the model.

As the number of network layers increases, the model can extract more complex patterns, so that theoretically better results can be obtained when the model is deeper. The ResNet model is composed of several layers so it can get good results in terms of performance and accuracy.

The following figure represents a comparison between the models already treated.

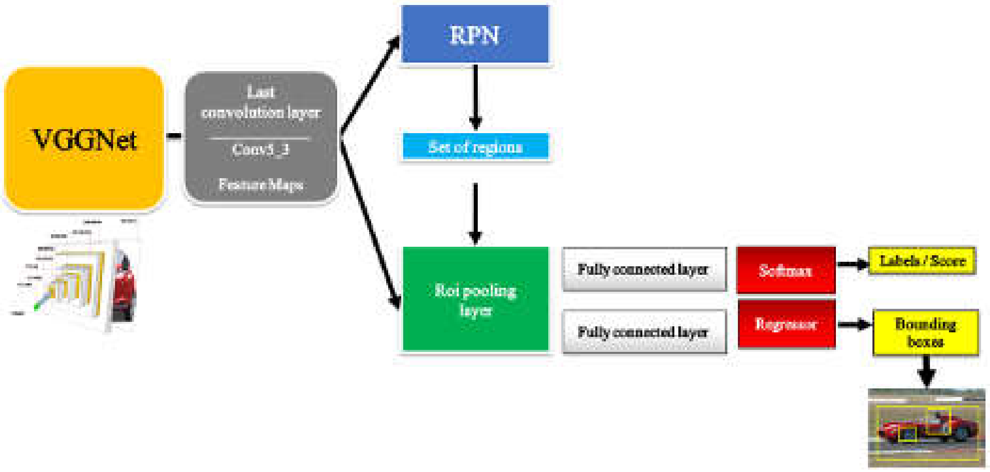


- ResNet won the ILSVRC 2015 competition with an unbelievable 3.57% error rate (human performance is 5-10%)
- GoogleNet won the ILSVRC 2014 competition with a 6.7% error rate.
- VGGNet won the ILSVRC 2014 competition with a 7.3% error rate
- AlexNet won the ILSVRC 2012 competition with a 16.4% error rate.

Faster R-CNN model:

In 2015, *Shaoqing Ren et al.* have proposed a new object detection system called Faster R-CNN based on a new region proposal method called RPN (Region Proposal Network) that share convolution layers with Fast R- CNN.

Faster R-CNN consists of two modules: the first module is a deep neural network that serves to provide high quality regions, and the second module is Fast R-CNN detector that uses the proposed regions for classification.

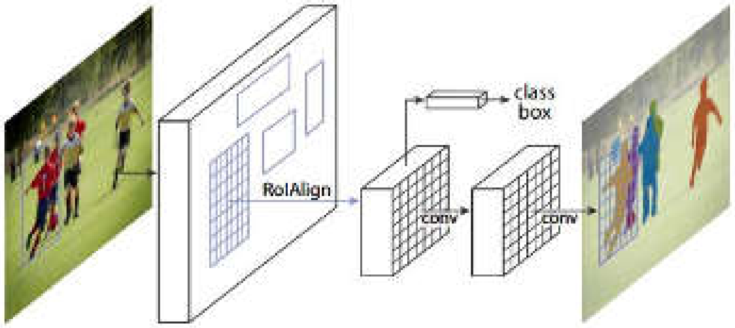


In Faster R-CNN, RPN is a small CNN (3x3 conv » 1x1 conv » 1x1 conv) looking at the conv5_3 global feature volume in the sliding window fashion.

Mask R-CNN model (Based on Instance Segmentation task)

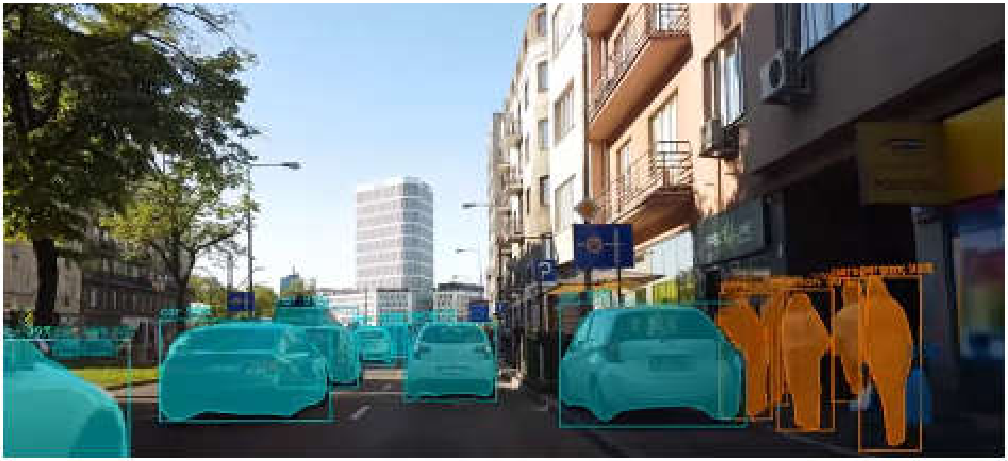
Mask R-CNN is a simple, flexible and general framework for object instance segmentation.

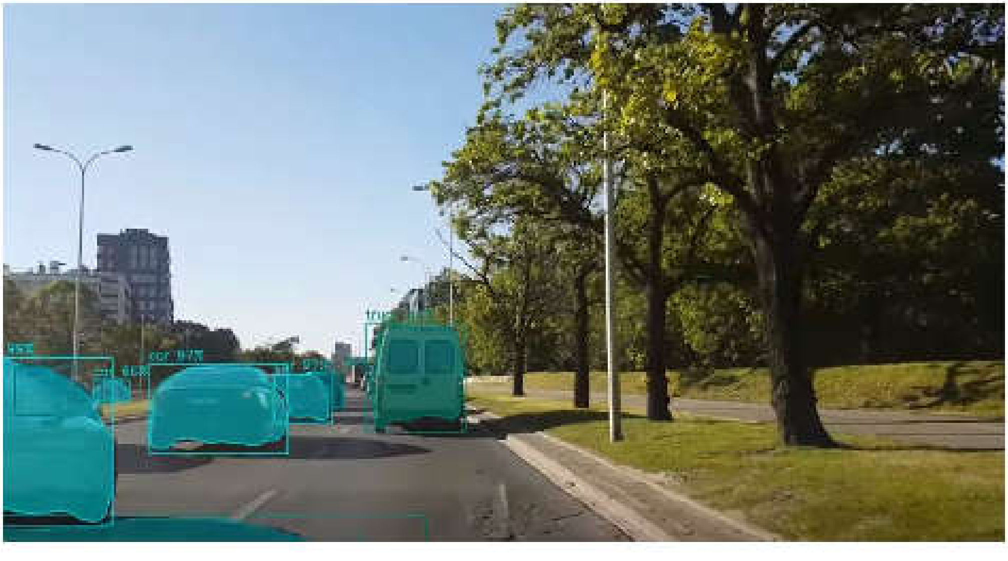
This method extends Faster R-CNN by adding a branch for predicting an object mask in parallel with the existing branch for object localization (bounding box). Mask R-CNN is simple to train and adds only a small overhead to Faster R-CNN, running at 5 fps.



Mask R-CNN adopts the same two-stage procedure: RPN and Fast R-CNN. In the second stage (Fast R-CNN), in parallel to predicting the class and box offset, Mask R-CNN also outputs a binary mask for each RoI.

The results of this network are illustrated by the following figures:





Model Evaluation Metrics

In terms of evaluation, a good detection system must be:

- robust to the presence (or absence) of objects in arbitrary scenes,
- invariant under translation, rotation, and scaling,
- able to detect partially hidden objects.

Now, we will see some criteria for the evaluation of models:

Precision and recall:

The recall is used to indicate how many objects have been detected, while the precision gives us information about the number of false alarms called also false positives.

Precision and recall can be defined by the following equations:

$$\text{Precision} = \frac{Vp}{Vp + Fp}$$

$$\text{Recall} = \frac{Vp}{Vp + Fn}$$

where Vp represents the true positives, Fp represents the false positives and Fn represents the false negatives.

mean Average Precision (mAP):

mAP is one of the most used metrics for evaluating model performance.

To calculate the mAP value, first we need to calculate the average precision for each class based on the prediction models.

Then, we take the average of these average precision which gives us the mean average precision (mAP).

ROC curve:

Currently, the ROC (Receiver Operating Characteristic) curve has become one of the standard techniques for evaluating models.

The ROC analysis consists in measuring the binary response of a detection system by calculating the True positive rate (Tvp) and the False positive rate (Tfp) in accordance with the following equations:

$$\text{Tvp} - {}^{\text{True positives}}/_{\text{Total positives}}$$

$$\text{Tfp} - {}^{\text{False positives}}/_{\text{Total positives}}$$

From Machine Learning to Deep Learning

There are a growing number of people who are seeking to understand the main concepts of Machine/Deep Learning and what powers them up. And if you are of these people, then this book is for you!

This book discusses the Machine/Deep Learning algorithms, methods, concepts, functions and code that make Deep Neural Networks such as Convolutional Neural Networks work well. You are going to follow codes that are being used to create these algorithms. You are going to see how Deep Neural Networks make decisions using several layers and functions.