UNITED COLLEGE OF EDUCATION AFFILIATED TO GURU GOBIND SINGH INDERPRASTHA UNIVERSITY (NEW DELHI)

Plot NO. 50-B, Knowledge park-III, Delhi-NCR Greater Noida, GB Nagar, U.P, INDIA, Pin-201301

MAJOR PROJECT REPORT
ON
"REAL-TIME OBJECT DETECTION USING DEEP LEARNING"

SUBMITTED FOR
Partial Fulfillment for the award of
Degree of Bachelor of computer Application (BCA)
To

Guru Gobind Singh Inderprastha University (New Delhi)

Under the Guidance of:

Dr.Meenu Sahni

(HOD, BCA)

Submitted by:

Saurabh kapasia

Roll no. 35527102018

CERTIFICATE

This is to certify that the report titled “Real-time Object Detection using deep learning” is an authentic record of Major Project work carried out by:

Name : Saurabh kapasia

Roll No. : 35527102018

Name : Prateek

Roll No. : 50427102018

Name : Mir Aizaz

Roll No. : 50327102018

Under my guidance & supervision, in the partial fulfillment of the requirements for the award of the Degree of Bachelor of Computer Application (BCA).

Dr. Meenu Sahni
(HOD, BCA)

DECLARATION

We hereby declare that this submission is our own work and that, to the best of our knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Name : Saurabh kapasia

Roll No. : 35527102018

Name : Prateek

Roll No. : 50427102018

Name : Mir Aizaz

Roll No. : 50327102018

ACKNOWLEDGEMENT

It gives us a great sense of pleasure to present the Major project report of the Project undertaken during BCA. We owe special debt of gratitude to Dr. Meenu Sahni, Head of Department, Bachelor of Computer Application, United College Of Education, GGSIPU, and DELHI for her constant support and guidance throughout the course of our work. Her sincerity, thoroughness and perseverance have been a constant source of inspiration for us.

We also take the opportunity to acknowledge the contribution of Gaurav Sir for his full support and assistance during the development of the project.

We also like to acknowledge the contribution of all faculty members of the department for their kind assistance and cooperation during the development of our project.

Last but not the least, we acknowledge all our teachers for their suggestions about project ideas and their contribution in the completion of our project.

Name : Saurabh kapasia

Roll No. : 35527102018

Name : Prateek

Roll No. : 50427102018

Name : Mir Aizaz

Roll No. : 50327102018

CONTRIBUTORS

- **Saurabh kapasia** BCA 3rd Year 5th Semester

Contribution in Designing, Programming and managing of the Overall project.

- **Prateek** BCA 3rd Year 5th Semester

Contribution in finding the References and programming of the project.

- **Mir Aziaz** BCA3rd Year 5th Semester

Contribution in making and writing the overall report for the project.

All these contribution helped make the team project a success.

ABSTRACT

Real time object detection is a vast, vibrant and complex area of computer vision. If there is a single object to be detected in an image, it is known as Image Localization and if there are multiple objects in an image, then it is Object Detection. This detects the semantic objects of a class in digital images and videos. The applications of real time object detection include tracking objects, video surveillance, pedestrian detection, people counting, self-driving cars, face detection, ball tracking in sports and many more. Convolution Neural Networks is a representative tool of Deep learning to detect objects using OpenCV(Open source Computer Vision), which is a library of programming functions mainly aimed at real-time computer vision.

Keywords: Computer vision, Deep Learning, Convolution Neural Networks.

Contents

INTRODUCTION:

A few years ago, the creation of the software and hardware image processing systems was mainly limited to the development of the user interface, which most of the programmers of each firm were engaged in. The situation has been significantly changed with the advent of the Windows operating system when the majority of the developers switched to solving the problems of image processing itself. However, this has not yet led to the cardinal progress in solving typical tasks of recognizing faces, car numbers, road signs, analyzing remote and medical images, etc.

Object recognition is to describe a collection of related computer vision tasks that involve activities like identifying objects in digital photographs. Image classification involves activities such as predicting the class of one object in an image. Object localization is refers to identifying the location of one or more objects in an image and drawing an abounding box around their extent. Object detection does the work of combines these two tasks and localizes and classifies one or more objects in an image. When a user or practitioner refers to the term "objects recognition", they often mean "objects detection". It may be challenging for beginners to distinguish between different related computer vision tasks.

So, we can distinguish between these three computer vision tasks with this example:

- Image Classification: This is done by Predict the type or class of an object in an image.
- Input: An image which consists of a single object, such as a photograph.
- Output: A class label (e.g. one or more integers that are mapped to class labels).
- Object Localization: This is done through, locate the presence of objects in an image and indicate their location with a bounding box.
- Input: An image which consists of one or more objects, such as a photograph.
- Output: One or more bounding boxes (e.g. defined by a point, width, and height).

- Object Detection: This is done through, locate the presence of objects with a bounding box and types or classes of the located objects in an image or in Video Stream.
- 2 Input: An image/Video Stream which consists of one or more objects, such as a photograph.
- Output: One or more bounding boxes (e.g. defined by a point, width, and height), and a class label for each bounding box

Project Objective:

The aim of object detection is to detect all instances of objects from a known class, such as people, cars or faces in an image or Video Stream.

Imparting intelligence to machines and making robots more and more autonomous and independent has been a sustaining technological dream for the mankind. It is our dream to let the robots take on tedious, boring, or dangerous work so that we can commit our time to more creative tasks. Unfortunately, the intelligent part seems to be still lagging behind. In real life, to achieve this goal, besides hardware development, we need the software that can enable robot the intelligence to do the work and act independently. One of the crucial components regarding this is vision, apart from other types of intelligences such as learning and cognitive thinking. A robot cannot be too intelligent if it cannot see and adapt to a dynamic environment.

The searching or recognition process in real time scenario is very difficult. So far, no effective solution has been found for this problem. Despite a lot of research in this area, the methods developed so far are not efficient, require long training time, are not suitable for real time application, and are not scalable to large number of classes. Object detection is relatively simpler if the machine is looking for detecting one particular object. However, recognizing all the objects inherently requires the skill to differentiate one object from the other, though they may be of same type. Such problem is very difficult for machines, if they do not know about the various possibilities of objects.

Reason and inspiration:

This project was created in order to help get a better understanding for deep machine-learning and get actual practical work with it. Learning from textbooks is fundamental, but to work with the methods in a practical way helps the understanding. However, the inspiration for the project came from not only wanting to get a better understanding but to also get a working program that has some purpose. Creating a model is one thing, but to also write a program that uses said model for inference is another. This project allows us to do both, and with the help from the official documentations, tutorials and various article on how to create a real-time application the project was made possible.

Object detection

INTRODUCTION TO OBJECT DETECTION

Object Detection is the process of finding and recognizing real-world object instances such as car, bike, TV, flowers, and humans out of an images or videos. An object detection technique lets you understand the details of an image or a video as it allows for the recognition, localization, and detection of multiple objects within an image.

It is usually utilized in applications like image retrieval, security, surveillance, and advanced driver assistance systems (ADAS).Object Detection is done through many ways:

- Feature Based Object Detection
- Viola Jones Object Detection
- SVM Classifications with HOG Features
- Deep Learning Object Detection

Object detection from a video in video surveillance applications is the major task these days. Object detection technique is used to identify required objects in video sequences and to cluster pixels of these objects.

The detection of an object in video sequence plays a major role in several applications specifically as video surveillance applications.

Object detection in a video stream can be done by processes like pre-processing, segmentation, foreground and background

extraction, feature extraction.

Humans can easily detect and identify objects present in an image. The human visual system is fast and accurate and can perform complex tasks like identifying multiple objects with little conscious thought. With the availability of large amounts of data, faster GPUs, and better algorithms, we can now easily train computers to detect and classify multiple objects within an image with high accuracy.

DIGITAL IMAGE PROCESSING:

Computerized picture preparing is a range portrayed by the requirement for broad test work to build up the practicality of proposed answers for a given issue. A critical trademark hidden the plan of picture preparing frameworks is the huge level of testing and experimentation that Typically is required before touching base at a satisfactory arrangement. This trademark infers that the capacity to plan approaches and rapidly model hopeful arrangements by and large assumes a noteworthy part in diminishing the cost and time required to land at a suitable framework execution.

WHAT IS DIP?

A picture might be characterized as a two-dimensional capacity f(x, y), where x, y are spatial directions, and the adequacy off at any combine of directions (x, y) is known as the power or dark level of the picture by then. Whenever x, y and the abundance estimation of are all limited discrete amounts, we call the picture a computerized picture. The field of DIP alludes to preparing advanced picture by methods for computerized PC. Advanced picture is made out of a limited number of components, each of which has a specific area and esteem. The components are called pixels.

Vision is the most progressive of our sensor, so it is not amazing that picture play the absolute most imperative part in human observation. Be that as it may, dissimilar to people, who are constrained to the visual band of the EM range imaging machines cover practically the whole EM range, going from gamma to radio waves. They can work likewise on pictures produced by sources that people are not acclimated to partner with picture.

There is no broad understanding among creators in regards to where picture handling stops and other related territories, for example, picture examination and PC vision begin. Now and then a qualification is made by characterizing picture handling as a teach in which both the info and yield at a procedure are pictures. This is constraining and to some degree manufactured limit. The range of picture investigation is in the middle of picture preparing and Pavilion.

There are no obvious limits in the continuum from picture handling toward one side to finish vision at the other. In any case, one helpful worldview is to consider three sorts of mechanized procedures in this continuum: low, mid and abnormal state forms. Low-level process includes primitive operations, for example, picture preparing to decrease commotion differentiate upgrade and picture honing. A low-level process is described by the way that both its sources of info and yields are pictures.

Mid-level process on pictures includes assignments, for example, division, depiction of that Question diminish them to a frame reasonable for PC handling and characterization of individual articles

A mid-level process is portrayed by the way that its sources of info by and large are pictures however its yields are properties removed from those pictures. At long last more clevated amount handling includes "Understanding an outlet of perceived items, as in picture examination and at the farthest end of the continuum playing out the intellectual capacities typically connected with human vision. Advanced picture handling, as effectively characterized is utilized effectively in a wide scope of regions of outstanding social and monetary esteem.

WHAT IS AN IMAGE?

A picture is spoken to as a two dimensional capacity f(x, y) where x And y is spatial co-ordinates and the adequacy of "T" at any match of directions (x, y) is known as the power of the picture by then.

Fig. 1.1 digital image

Processing on image:

Processing on image can be of three types They are low-level, mid-level, high level.

Low-level Processing:

- Preprocessing to remove noise.
- Contrast enhancement.
- Image sharpening.

Medium Level Processing:

- Segmentation.
- Edge detection
- Object extraction

High Level Processing:

- Image analysis
- Scene interpretation

Why Image Processing?

Since the digital image is invisible, it must be prepared for viewing on one or more output device (laser printer, monitor at).The digital image can be optimized for the application by enhancing the appearance of the structures within it.

There is three of image processing used. They are

- Image to Image transformation
- Image to Information transformations
- Information to Image transformations

Pixel:

Pixel is the smallest element of an image. Each pixel corresponds to any one value. In an 8-bit gray scale image, the value of the pixel

between 0 and 255.Each pixel store a value proportional to the light intensity at that particular location. It is indicated in either Pixels per inch or Dots per inch.

Resolution:

The resolution can be defined in many ways. Such as pixel resolution, spatial resolution, temporal resolution, spectral resolution. In pixel resolution, the term resolution refers to the total number of count of pixels in an digital image. For example, If an image has M rows and N columns, then its resolution can be defined as MX N. Higher is the pixel resolution; the higher is the quality of the image.

Resolution of an image is of generally two types.

- Low Resolution image
- High Resolution

Since high resolution is not a cost effective process It is not always possible to achieve high resolution images with low cost. Hence it is desirable Imaging. In Super Resolution imaging, with the help of certain methods and algorithms we can be able to produce high resolution images from the low resolution image from the low resolution images.

GRAY SCALE IMAGE

A gray scale picture is a capacity I (xylem) of the two spatial directions of the picture plane. I (x, y) is the force of the picture force of picture at the point (x, y) on the picture plane. I (xylem) take non- negative expect the picture is limited by a rectangle.

COLOR IMAGE

It can be spoken to by three capacities, R (xylem) for red, G (xylem) for green and B (xylem) for blue. A picture might be persistent as for the x and y facilitates and further more in adequacy. Changing over such a picture to advanced shape requires that the directions and the adequacy to be digitized. Digitizing the facilitates esteems is called inspecting. Digitizing the adequacy esteems is called quantization.

RELATED TECHNOLOGY:

R-CNN

R-CNN is a progressive visual object detection system that combines bottom-up region proposals with rich options computed by a convolution neural network.

R-CNN uses region proposal ways to initial generate potential bounding boxes in a picture and then run a classifier on these proposed boxes.

SINGLE SIZE MULTI BOX DETECTOR

SSD discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At the time of prediction the network generates scores for the presence of each object category in each default box and generates adjustments to the box to better match the object shape.

Additionally, the network combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes.

ALEXNET

AlexNet is a convolutional neural Network used for classification which has 5 Convolutional layers, 3 fully connected layers and 1 softmax layer with 1000 outputs for classification as his architecture.

YOLO

YOLO is real-time object detection. It applies one neural network to the complete image dividing the image into regions and predicts bounding boxes and possibilities for every region.

Predicted probabilities are the basis on which these bounding boxes are weighted. A single neural network predicts bounding boxes and class possibilities directly from full pictures in one

evaluation. Since the full detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

VGG

VGG network is another convolution neural network architecture used for image classification.

MOBILENETS

To build lightweight deep neural networks MobileNets are used. It is based on a streamlined architecture that uses depth-wise separable

Convolutions. MobileNet uses 3×3 depth-wise separable convolutions that uses between 8 times less computation than standard convolution at solely a little reduction accuracy. Applications and use cases including object detection, fine grain classification, face attributes and large-scale-localization.

TENSOR FLOW

Tensor flow is an open source software library for high performance numerical computation. It allows simple deployment of computation across a range of platforms (CPUs, GPUs, TPUs) due to its versatile design also from desktops to clusters of servers to mobile and edge devices. Tensor flow was designed and developed by researchers and engineers from the Google Brain team at intervals Google's AI organization; it comes with robust support for machine learning and deep learning and the versatile

numerical computation core is used across several alternative scientific domains.

To construct, train and deploy Object Detection Models TensorFlow is used that makes it easy and also it provides a collection of Detection Models pre-trained on the COCO dataset, the Kitti dataset, and the Open Images dataset. One among the numerous Detection Models is that the combination of Single

Shot Detector (SSDs) and Mobile Nets architecture that is quick, efficient and doesn't need huge computational capability to accomplish the object Detection.

APPLICATION OF OBJECT DETECTION

The major applications of Object Detection are:

FACIAL RECOGNITION

“Deep Face” is a deep learning facial recognition system developed to identify human faces in a digital image. Designed and developed by a group of researchers in Facebook. Google also has its own facial recognition system in Google Photos, which automatically separates all the photos according to the person in the image.

There are various components involved in Facial Recognition or authors could say it focuses on various aspects like the eyes, nose, mouth and the eyebrows for recognizing faces.

PEOPLE COUNTING

People counting are also a part of object detection which can be used for various purposes like finding person or a criminal; it is used for analyzing store performance or statistics of crowd during festivals. This process is considered a difficult one as people move out of the frame quickly.

INDUSTRIAL QUALITY CHECK

Object detection also plays an important role in industrial processes to identify or recognize products. Finding a particular object through

visual examination could be a basic task that's involved in multiple industrial processes like sorting, inventory management, machining, quality management, packaging and so on. Inventory management can be terribly tough as things are hard to trace in real time. Automatic object counting and localization permits improving inventory accuracy.

SELF DRIVING CARS

Self-driving is the future most promising technology to be used, but the working behind can be very complex as it combines a variety of techniques to perceive their surroundings, including radar, laser light, GPS, odometer, and computer vision. Advanced control systems interpret sensory info to allow navigation methods to work, as well as obstacles and it. This is a big step towards Driverless cars as it happens at very fast speed.

SECURITY

Object Detection plays a vital role in the field of Security; it takes part in major fields such as face ID of Apple or the retina scan used in all the sci-fi movies. Government also widely use this application to access the security feed and match it with their

existing database to find any criminals or to detecting objects like car number involved in criminal activities. The applications are limitless.

OBJECT DETECTION WORKFLOW AND FEATURE EXTRACTION

Every Object Detection Algorithm works on the same principle and it's just the working that differs from others. They focus on extracting features from the images that are given as the input at hands and then it uses these features to determine the class of the image.

Deep Learning

INTRODUCTION

Deep learning is a machine learning technique. It teaches a computer to filter inputs through layers to learn how to predict and classify information. Observations can be in the form of images, text, or sound. The inspiration for deep learning is the way that the human brain filters information. Its purpose is to mimic how the human brain works to create some real magic. In the human brain, there are about 100 billion neurons. Each neuron connects to about 100,000 of its neighbors. We're kind of recreating that, but in a way and at a level that works for machines. In our brains, a neuron has a body, dendrites, and an axon. The signal from one neuron travels down the axon and transfers to the dendrites of the next neuron. That connection where the signal passes is called a synapse. Neurons by themselves are kind of useless. But when you have lots of them, they work together to create some serious magic. That's the idea behind a deep learning algorithm! You get input from observation and you put your input into one layer. That layer creates an output which in turn becomes the input for the next layer, and so on. This happens over and over until your final output signal! The neuron (**node**) gets a signal or signals (input **values**), which pass through the neuron. That neuron delivers the **output signal**.

Think of the input layer as your senses: the things you see, smell, and feel, for example. These are independent variables for one single observation. This information is broken down into numbers

and the bits of binary data that a computer can use. You'll need to

either standardize or normalize these variables so that they're within the same range. They use many layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output of the previous layer for its input. What they learn forms a hierarchy of concepts. In this hierarchy, each level learns to transform its input data into a more and more abstract and composite representation. That means that for an image, for example, the input might be a matrix of pixels. The first layer might encode the edges and compose the pixels. The next layer might compose an arrangement of edges. The next layer might encode a nose and eyes. The next layer might recognize that the image contains a face, and soon.

What happens inside the neuron?

The input node takes in information in a numerical form. The information is presented as an activation value where each node is given a number. The higher the number, the greater the activation. Based on the connection strength (weights) and transfer function, the activation value passes to the next node. Each of the nodes sums the activation values that it receives (it calculates the **weighted sum**) and modifies that sum based on its transfer function.Next, it applies an activation function.Anactivation function is a function that's applied to this particular neuron. From that, the neuron understands if it needs to pass along a signal or not.

Each of the synapses gets assigned weights, which are crucial to **Artificial Neural Networks** (ANNs). Weights are how ANNs learn.

By adjusting the weights, the ANN decides to what extent signals get passed along. When you're training your network, you're deciding how the weights are adjusted.

The activation runs through the network until it reaches the output nodes. The output nodes then give us the information in a way that we can understand. Your network will use a cost function to compare the output and the actual expected output. The model performance is evaluated by the cost function. It's expressed as the difference between the actual value and the predicted value.

There are many different cost functions you can use; you're looking at what the error you have in your network is. You're working to minimize loss function. (In essence, the lower the loss functions, the closer it is to your desired output). The information goes back, and the neural network begins to learn with the goal of minimizing the cost function by tweaking the weights. This process is called **back propagation**.

In **forward propagation**, information is entered into the input layer and propagates forward through the network to get our output values. We compare the values to our expected results.

Next, we calculate the errors and propagate the info backward. This allows us to train the network and update the weights. (Back propagation allows us to adjust all the weights simultaneously.) During this process, because of the way the algorithm is structured, you're able to adjust all of the weights simultaneously. This allows you to see which part of the error each of your weights in the neural network is responsible for.

When you've adjusted the weights to the optimal level, you're ready to proceed to the testing phase!

How does an artificial neural network learn?

There are two different approaches to get a program to do what you want. First, there's the specifically guided and hard-programmed approach. You tell the program exactly what you want it to do. Then there are **neural networks**. In neural networks, you tell your network the inputs and what you want for the outputs, and then you let it learn on its own.

By allowing the network to learn on its own, you can avoid the necessity of entering in all of the rules. You can create the architecture and then let it go and learn. Once it's trained up, you can give it a new image and it will be able to distinguish output.

Feed forward and feedback networks

A **feed forward** network is a network that contains inputs, outputs, and hidden layers. The signals can only travel in one direction (forward). Input data passes into a layer where calculations are performed. Each processing element computes based upon the weighted sum of its inputs. The new values become the new input values that feed the next layer (feed-forward). This continues through all the layers and determines the output. Feed forward networks are often used in, for example, data mining.

A **feedback network** (for example, a recurrent neural network) has feedback paths. This means that they can have signals traveling in both directions using loops. All possible connections between neurons are allowed. Since loops are present in this type of network, it becomes a non-linear dynamic system which changes continuously until it reaches a state of equilibrium. Feedback networks are often used in optimization problems where the network looks for the best arrangement of interconnected factors.

Weighted Sum

Inputs to a neuron can either be features from a training set or outputs from the neurons of a previous layer. Each connection between two neurons has a unique synapse with a unique weight attached. If you want to get from one neuron to the next, you have to travel along the synapse and pay the “toll” (weight). The neuron then applies an activation function to the sum of the weighted inputs

from each incoming synapse. It passes the result on to all the neurons

in the next layer. When we talk about updating weights in a network, we're talking about adjusting the weights on these synapses.

A neuron's input is the sum of weighted outputs from all the neurons in the previous layer. Each input is multiplied by the weight associated with the synapse connecting the input to the current neuron. If there are 3 inputs or neurons in the previous layer, each neuron in the current layer will have 3 distinct weights: one for each synapse.

In a nutshell, the activation function of a node defines the output of that node.

The activation function (or transfer function) translates the input signals to output signals. It maps the output values on a range like 0 to 1 or -1 to 1. It's an abstraction that represents the rate of action potential firing in the cell. It's a number that represents the likelihood that the cell will fire. At its simplest, the function is binary: **yes** (the neuron fires) or **no** (the neuron doesn't fire). The output can be either 0 or 1 (on/off or yes/no), or it can be anywhere in a range. If you were using a function that maps a range between 0 and 1 to determine the likelihood that an image is a cat, for example, an output of 0.9 would show a 90% probability that your image is, in fact, act.

Activation function

In a nutshell, the activation function of a node defines the output of

that node.

The activation function (or transfer function) translates the input signals to output signals. It maps the output values on a range like 0 to 1 or -1 to 1. It's an abstraction that represents the rate of action potential firing in the cell. It's a number that represents the likelihood that the cell will fire. At its simplest, the function is binary: **yes** (the neuron fires) or **no** (the neuron doesn't fire). The output can be either 0 or 1 (on/off or yes/no), or it can be anywhere in a range.

What options do we have? There are many activation functions, but these are the four very common ones:

Threshold function

This is a step function. If the summed value of the input reaches a certain threshold the function passes on 0. If it's equal to or more than zero, then it would pass on 1. It's a very rigid, straightforward, yes or no function.

Sigmoid function

This function is used in logistic regression. Unlike the threshold function, it's a smooth, gradual progression from 0 to 1. It's useful in the output layer and is used heavily for linear

Hyperbolic Tangent Function

This function is very similar to the sigmoid function. But unlike the sigmoid function which goes from 0 to 1, the value goes below zero,

from -1 to 1. Even though this isn't a lot like what happens in a brain, this function gives better results when it comes to training neural networks. Neural networks sometimes get "stuck" during training with the sigmoid function. This happens when there's a lot of strongly negative input that keeps the output near zero, which messes with the learning process.

Rectifier function

This might be the most popular activation function in the universe of neural networks. It's the most efficient and biologically plausible. Even though it has a kink, it's smooth and gradual after the kink at 0. This means, for example, that your output would be either "no" or a percentage of "yes." This function doesn't require normalization or other complicated calculations.

The field of artificial intelligence is essential when machines can do tasks that typically require human intelligence. It comes under the layer of machine learning, where machines can acquire skills and learn from past experience without any involvement of human. Deep learning comes under machine learning where artificial neural networks, algorithms inspired by the human brain, learn from large amounts of data. The concept of deep learning is based on humans' experiences; the deep learning algorithm would perform a task continuously so that it can improve the outcome. Neural networks have various (deep) layers that enable learning.

INTRODUCTION TO CONVOLUTIONAL NEURAL NETWORKS (CNN)

Artificial Neural Networks

The idea of ANNs is based on the belief that working of human brain by making the right connections can be imitated using silicon and wires as living neurons and dendrites.

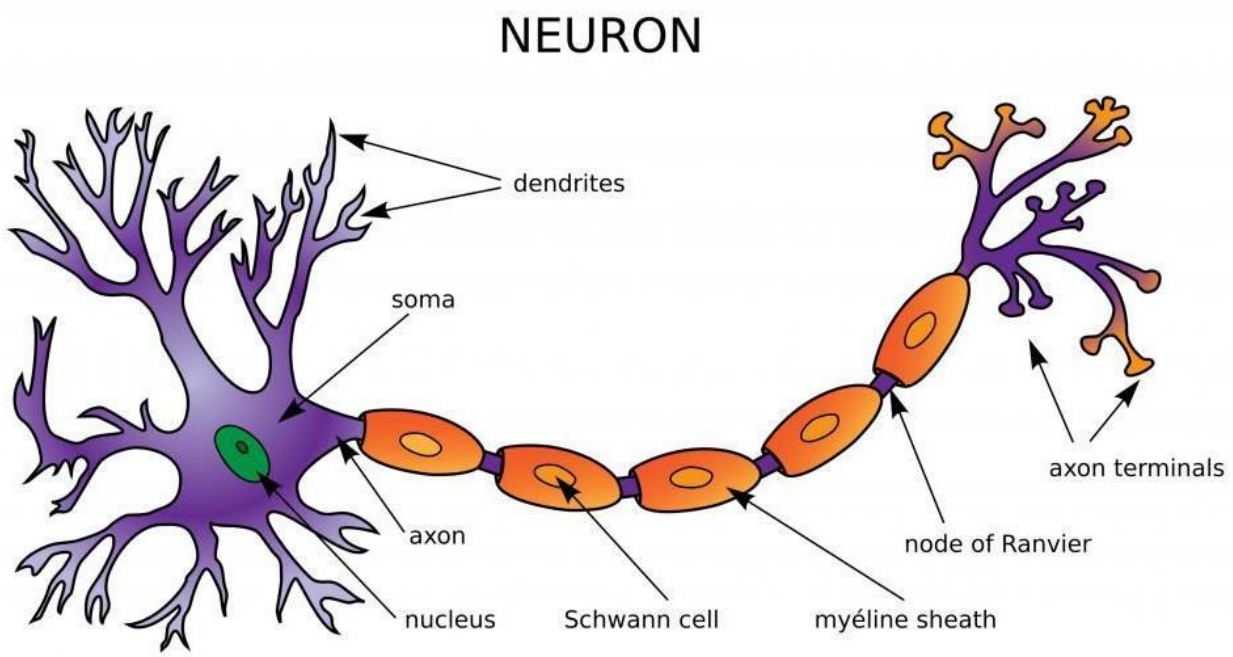


Fig:

The human brain is composed of 86 billion nerve cells called neurons. They are connected to other thousand cells by Axons. Stimuli from external environment or inputs from sensory organs are accepted by dendrites. These inputs create electric impulses,

which quickly travel through the neural network? A neuron can then send the message to other neuron to handle the issue or does not send it forward.

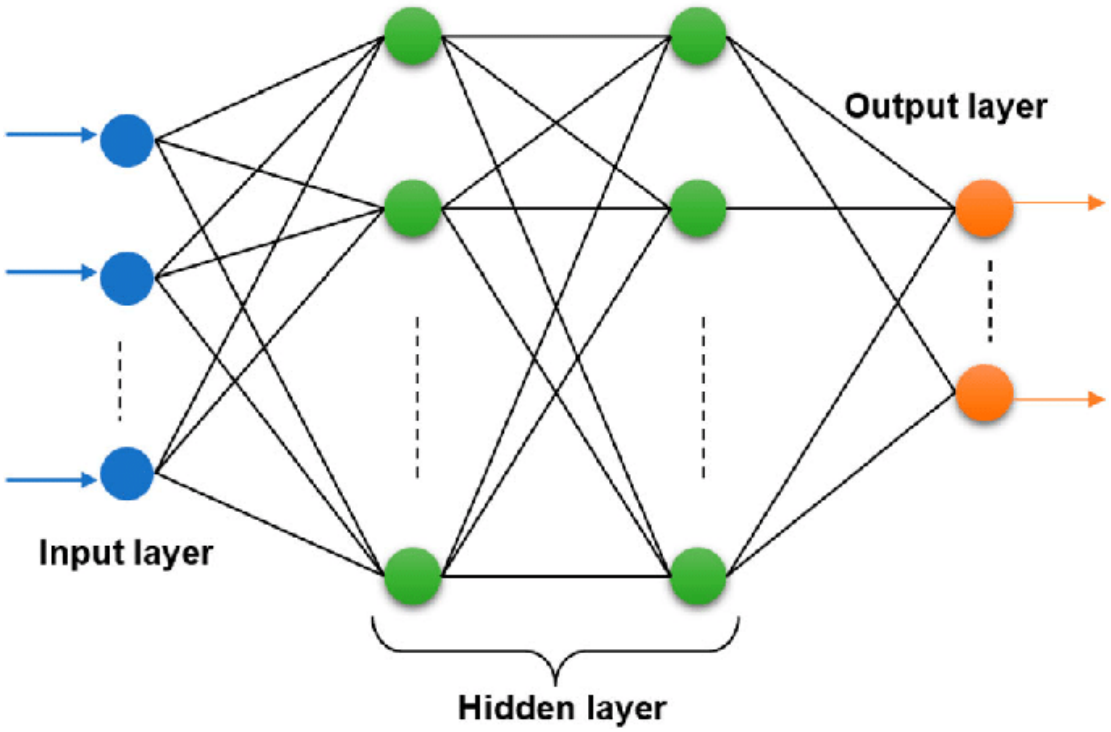
ANNs are composed of multiple nodes, which imitate biological neurons of human brain. The neurons are connected by links and they interact with each other. The nodes can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each node is called its activation or node value. Each link is associated with weight. ANNs are capable of learning, which takes place by altering weight values.

Neural network:

A neural network is a network or circuit of neurons, or in a modern sense, an artificial neural network, composed of artificial neurons or nodes. Thus a neural network is either a biological neural network, made up of real biological neurons, or an artificial neural network, for solving artificial intelligence (AI) problem. The connections of the biological neuron are modeled as weights. A positive weight reflects an excitatory connection, while negative values mean inhibitory connections. All inputs are modified by a weight and summed. This activity is referred as linear combination. Finally, an activation function controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1, or it could be - 1 and 1.

These artificial networks may be used for predictive modeling, adaptive control and applications where they can be trained via a dataset. Self-learning resulting from experience can occur within networks,

Which can derive conclusions from a complex and seemingly unrelated set of information?

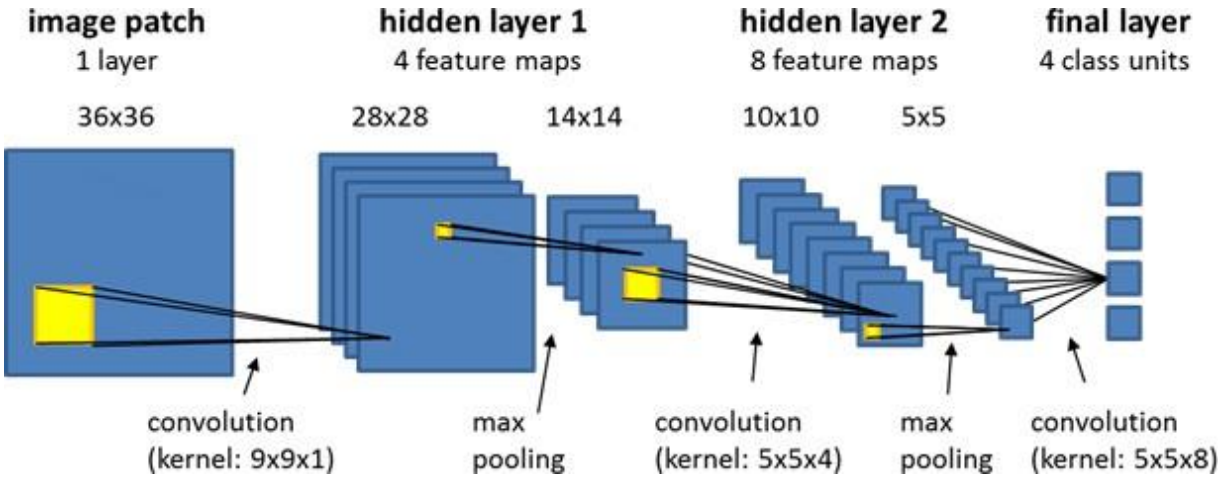


A simple neural network

A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers. The DNN finds the correct mathematical manipulation to turn the input into the output, whether it is a linear relationship or a non-linear relationship.

CONVOLUTIONAL NEURAL NETWORKS:

Convolutional Neural Networks are very similar to ordinary Neural Networks from the previous chapter: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity.



Convolutional Neural Networks (CNNs) are analogous to traditional ANNs in that they are comprised of neurons that self-optimize through learning. Each neuron will still receive an input and

Perform an operation (such as a scalar product followed by a non-linear function) - the basis of countless ANNs. From the input raw image vectors to the final output of the class score, the entire of the network will still express a single.

CNN ARCHITECTURE:

CNNs are feed forward networks in that information flow takes place in one direction only, from their inputs to their outputs. Just as artificial neural networks (ANN) are biologically inspired, so are CNNs. The visual cortex in the brain, which consists of alternating layers of simple and complex cells (Hubel & Wiesel, 1959, 1962), motivates their architecture.

CNN architectures come in several variations; however, in general, they consist of convolutional and pooling (or sub sampling) layers, which are grouped into modules. Either one or more fully connected layers, as in a standard feed forward neural network, follow these modules. Modules are often stacked on top of each other to form a deep model. It illustrates typical CNN architecture for a toy image classification task. An image is input directly to the network, and this is followed by several stages of convolution and pooling. Thereafter, representations from these operations feed one or more fully connected layers.

Finally, the last fully connected layer outputs the class label. Despite this being the most popular base architecture found in the literature, several architecture changes have been proposed in recent years with the objective of improving image classification accuracy or reducing computation costs. Although for the remainder of this section, we merely fleetingly introduce standard CNN architecture.

OVERALL ARCHITECTURE:

CNNs are comprised of three types of layers. These are convolutional layers, pooling layers and fully- connected layers. When these layers are stacked, CNN architecture has been formed. A simplified CNN architecture for MNIST classification is illustrated in Figure2. input 0 9 convolution w/Rely pooling output fully-connected w/ Rely fully-connected ... Fig. 2: An simple CNN architecture, comprised of just five layers The basic functionality of the example CNN above can be broken down into four key areas.

1. As found in other forms of ANN, the input layer will hold the pixel values of the image.

2. The convolutional layer will determine the output of neurons of which are connected to local regions of the input through the calculation of the scalar product between their weights and the region connected to the input volume. The rectified linear unit (commonly shortened to Rely) aims to apply an 'element wise' activation function such as sigmoid to the output of the activation produced by the previous layer.

3. The pooling layer will then simply perform down sampling along the spatial dimensionality of the given input, further reducing the number of parameters within that activation.

4. The fully-connected layers will then perform the same duties found in standard ANNs and attempt to produce class scores from the activations, to be used for classification. It is also suggested that Rely may be used between these layers, as to improve performance. Through this simple method of transformation, CNNs are able to transform the original input layer by layer using convolutional and down sampling techniques to produce class scores for classification and regression purposes. However, it is important to note that simple understanding the overall architecture of CNN architecture will not suffice. The creation and optimization of these models can take quite some time, and can be quite confusing. We will now explore in detail the individual layers, detailing their hyper parameters and connectivity's.

CONVOLUTIONAL LAYERS:

The convolutional layers serve as feature extractors, and thus they learn the feature representations of their input images. The neurons in the convolutional layers are arranged into feature maps. Each neuron in a feature map has a receptive field, which is connected to a neighborhood of neurons in the previous layer via a set of trainable weights, sometimes referred to as a filter bank. Inputs are convolved with the learned weights in order to compute a new feature map, and the convolved results are sent through a nonlinear activation function.

All neurons within a feature map have weights that are constrained to be equal; however, different feature maps within the same convolutional layer have different weights so that several features can be extracted at each location.

As the name implies, the convolutional layer plays a vital role in how CNNs operate. The layers parameters focus around the use of learnable kernels.

These kernels are usually small in spatial dimensionality, but spreads along the entirety of the depth of the input. When the

Data hits a convolutional layer; the layer convolves each filter across the spatial dimensionality of the input to produce a 2D activation map. These activation maps can be visualized.

As we glide through the input, the scalar product is calculated for each value in that kernel. From this the network will learn kernels that 'fire' when they see a specific feature at a given spatial position of the input. These are commonly known as activations.

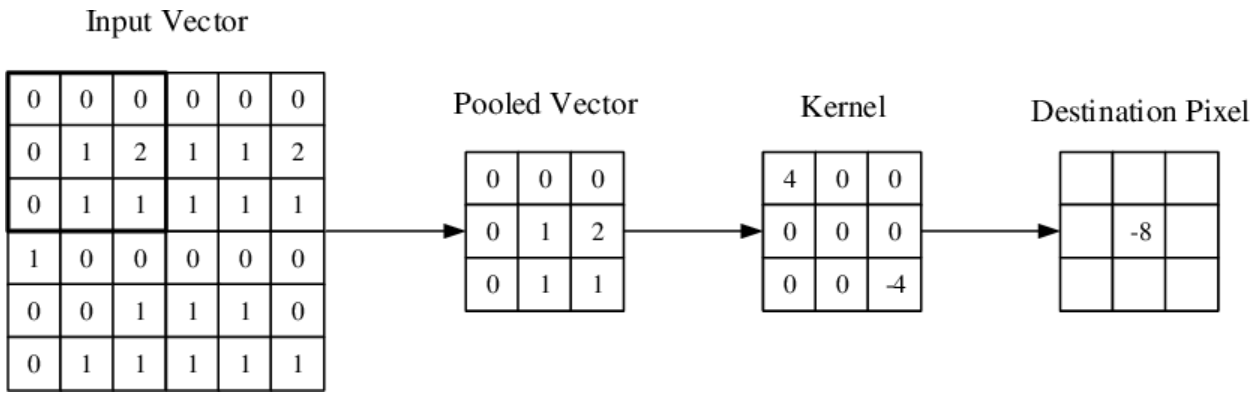


Fig: Visual representation of a convolution layers

The centre element of the kernel is placed over the input vector, of which is then calculated and replaced with a weighted sum of itself and any nearby pixels.

Every kernel will have a corresponding activation map, of which will be stacked along the depth dimension to form the full output volume from the convolutional layer.

As we alluded to earlier, training ANNs on inputs such as images results in models of which are too big to train effectively? This

Comes down to the fully connected manner of Stan arid ANN neurons, so to mitigate against this every neuron in a convolutional layer is only connected to small region of the input volume. The dimensionality of this region is commonly referred to as the receptive field size of the neuron. The magnitude of the connectivity through the depth is nearly always equal to the depth of the input.

For example, if the input to the network is an image of size 64 × 64 × 3 (a RGB colored image with a dimensionality of 64 × 64) and we set the receptive field size as 6 × 6, we would have a total of 108 weights on each neuron within the convolution layer. (6 × 6 × 3 where 3 is the magnitude of connectivity across the depth of the volume) To put this into perspective, a standard neuron seen in other forms of ANN would contain 12, 288 weights each.

Convolutional layers are also able to significantly reduce the complexity of the model through the optimization of its output. These are optimized through three hyper parameters, the depth, the stride and setting zero-padding.

The depth of the output volume produced by the convolution layers can be manually set through the number of neurons within the layer to the same region of the input. This can be seen with other forms of ANNs, where the all of the neurons in the hidden layer are directly connected to every single neuron beforehand. Reducing this hyper parameter can significantly minimize the total numbers of neurons of the network, but it can also significantly reduce the pattern recognition capabilities of the model.

We are also able to define the stride in which we set the depth around the spatial dimensionality of the input in order to place the receptive field. For example if we were to set a stride as 1, then we would have a heavily overlapped receptive field producing extremely large activations. Alternatively, setting the stride to a greater number will reduce the amount of overlapping and produce an output of lower spatial dimensions.

Zero-padding is the simple process of padding the border of the input, and is an effective method to give further control as to the dimensionality of the output volumes.

It is important to understand that through using these techniques, we will alter the spatial dimensionality of the convolution layers output.

Despite our best efforts so far we will still find that our models are still enormous if we use an image input of any real dimensionality. However, methods have been developed as to greatly curtail the overall number of parameters within the convolution layer.

Parameter sharing works on the assumption that if one region feature is useful to compute at a set spatial region, then it is likely to be useful in another region. If we constrain each individual activation map within the output volume to the same weights and bias, then we will see a massive reduction in the number of parameters being produced by the convolution layer.

As a result of this as the back propagation stage occurs, each neuron in the output will represent the overall gradient of which can be totaled across the depth - thus only updating a single set of weights, as opposed to every single one.

Pooling Layers

The purpose of the pooling layers is to reduce the spatial resolution of the feature maps and thus achieve spatial invariance to input distortions and translations. Initially, it was common practice to use average pooling aggregation layers to propagate the average of all the input values, of a

small neighborhood of an image to the next layer. However, in more recent models, max pooling aggregation layers propagate the maximum value within a receptive field to the next layer.

Pooling layers aim to gradually reduce the dimensionality of the representation, and thus further reduce the number of parameters and the computational complexity of the model.

The pooling layer operates over each activation map in the input, and scales its dimensionality using the "MAX" function. In most CNNs, these come in the form of max-pooling layers with kernels of
A dimensionality of 2 × 2 applied with a stride of 2 along the spatial dimensions of the input. This scales the activation map down to 25% of the original size - whilst maintaining the depth volume to its standard size.

Due to the destructive nature of the pooling layer, there are only two generally observed methods of max-pooling. Usually, the stride and filters of the pooling layers are both set to 2 × 2, which will allow the layer to extend through the entirety of the spatial dimensionality of the input. Furthermore overlapping pooling may be utilized, where the stride is set to 2 with a kernel size set to 3. Due to the destructive nature of pooling, having a kernel size above 3 will usually greatly decrease the performance of the model.

It is also important to understand that beyond max-pooling, CNN architectures may contain general- pooling. General

pooling layers are comprised of pooling neurons that are able to perform a multitude of common operations including L1/L2-normalisation, and average pooling. However, this tutorial will primarily focus on the use ofmax-pooling.

Fully Connected Layers

Several convolution and pooling layers are usually stacked on top of each other to extract more abstract feature representations in moving through the network. The fully connected layers that follow these layers interpret these feature representations and perform the function of high-level reasoning. . For classification problems, it is standard to use the softmax operator on top of a DCNN. While early success was enjoyed by using radial basis functions (RBFs), as the classifier on top of the convolution towers found that replacing the softmax operator with a support vector machine (SVM) leads to improved classification accuracy.

The fully-connected layer contains neurons of which are directly connected to the neurons in the two adjacent layers, without being connected to any layers within them. This is analogous to way that neurons are arranged in traditional forms of ANN.

Despite the relatively small number of layers required to form a CNN, there is no set way of formulating CNN architecture. Through reading of related literature it is obvious that much like other forms of ANNs; CNNs tend to

follow a common architecture. This common architecture has convolutional layers stacked, followed by pooling layers in a repeated manner before feeding forward to fully-connected layers.

Convolutional Neural Networks differ to other forms of Artificial Neural Network in that instead of focusing on the entirety of the problem domain, knowledge about the specific type of input is exploited. This in turn allows for much simpler network architecture to be setup.

This paper has outlined the basic concepts of Convolutional Neural Networks, explaining the layers required to build one and detailing how best to structure the network in most image analysis tasks.

Research in the field of image analysis using neural networks has somewhat slowed in recent times. This is partly due to the incorrect belief surrounding the level of complexity and knowledge required to begin modeling these superbly powerful machine learning algorithms. The authors hope that this paper has in some way reduced this confusion, and made the field more accessible to beginners.

Training

CNNs and ANN in general use learning algorithms to adjust

their free parameters in order to attain the desired network output. The most common algorithm used for this purpose is back propagation. Back propagation computes the gradient of an objective function to determine how to adjust a network's parameters in order to minimize errors that affect performance. A commonly experienced problem with training CNNs, and in particular DCNNs, is over fitting, which is poor performance on a held-out test set after the network is trained on a small or even large training set. This affects the model's ability to generalize on unseen data and is a major challenge for DCNNs that can be assuaged by regularization.

Model used in project

Caffe Model

Caffe is a framework of Deep Learning and it was made used for the implementation and to access the following things in an object detection system.

- Expression: Models and optimizations are defined as plaintext schemas in the caffe model unlike others which use codes for this purpose.
- Speed: for research and industry alike speed is crucial for state-of-the-art models and massive data.
- Modularity: Flexibilityandextensionismajorlyrequiredforthenewtasksanddif ferentsettings.
- Openness: Common code, reference models, and reproducibility are the basic requirements of scientific and applied progress.

Types of Caffe Models:

Open Pose

The first real-time multi-person system is portrayed by Open Pose which can collectively sight human body, hand, and facial keypoints (in total 130 keypoints) on single pictures.

Fully Convolutional Networks for Semantic Segmentation

In the absolutely convolutional networks (FCNs) Fully

Convolutional Networks are the reference implementation of the models and code for the within the PAMI FCN and CVPR FCN papers.

Cnn-vis

Cnn-vis is an open-source tool that lets you use convolutional neural networks to generate images. It has taken inspiration from the Google's recent Inceptionism blogpost.

Speech Recognition

Speech Recognition with the caffe deep learning framework.

DeconvNet

Learning Deconvolution Network for Semantic Segmentation.

Coupled Face Generation

This is the open source repository for the Coupled Generative Adversarial Network (Coupled AN or CoGAN) work. These models are compatible with Caffe master, unlike earlier FCNs that required a pre- release branch.

Codes for Fast Image Retrieval

To create the hash-like binary codes it provides effective framework for fast image retrieval.

SegNet and Bayesian SegNet

SegNet is real-time semantic segmentation architecture for scene understanding.

Deep Hand

It gives pre-trained CNN models.

DeepYeast

Deep Yeast may be an 11-layer convolutional neural network trained on biaural research pictures of yeast cells carrying fluorescent proteins with totally different subcellular localizations.

Python VS other languages for Object Detection: Object detection may be a domain-specific variation of the machine learning prediction drawback. Intel's OpenCV library that is implemented in C/C++ has its interfaces offered during a} very vary of programming environments like C#, Matlab, Octave, R, Python and then on.

More choices in graphics packages and toolsets Supervised learning also plays an important role.

The utility of unsupervised pre-training is usually evaluated on the premise of what performance is achieved when supervised fine-tuning. This paper reviews and discusses the fundamentals of learning as well as supervised learning for classification models, and also talks about the mini batch stochastic gradient descent algorithm that is used to fine-tune many of the models.

Shape-Based

A mixture of image-based and scene based object parameters such as image blob (binary large object) area, the as pectration of blob bounding box and camera zoom is given as input to this detection system. Classification is performed on the basis of the blob at each and every frame. The results are kept in the histogram.

Motion-Based

When an easy image is given as an input with no objects in motion, this classification isn't required. In general, non-rigid articulated human motion shows a periodic property; therefore this has been used as a powerful clue for classification of moving objects. Based on this useful clue, human motion is distinguished from different objects motion. Color Based- though color isn't an applicable live alone for police investigation and following objects, but the low process value of the color primarily based algorithms makes the coloura awfully smart feature to be exploited. As an example, the color- histogram based technique is employed for detection of vehicles in period. Color bar chart describes the color distribution in a very given region that is powerful against partial occlusions.

Texture-Based

The texture-based approaches with the assistance of texture pattern recognition work just like motion- based approaches.

It provides higher accuracy, by exploitation overlapping native distinction social control however might need longer, which may be improved exploitation some quick techniques. I. proposed WORK Authors have applied period object detection exploitation deep learning and OpenCV to figure to work with video streams and video files. This will be accomplished using the highly efficient open computer vision. Implementation of proposed strategy includes caffe-model based on Google Image Scenery; Caffe offers the model definitions, optimization settings, pre- trained weights [4]. Prerequisite includes Python 3.7, OpenCV 4 packages and numpy to complete this task of object detection. Numpy is the elementary package for scientific computing with Python. It contains among other things: a strong N-dimensional array object, subtle (broadcasting) functions tools for integrating C/C++ and Fortran code, helpful linear algebra, Fourier transform, and random number capabilities. Numpy works in backend to provide statistical information of resemblance of object with the image scenery caffemodel database. Object clusters can be created according to fuzzy value provided by Numpy. This project can detect live objects from the videos and images.

LEARNING FEATURE HIERARCHY:

Learn hierarchy all the way from pixels classifier one layer extracts features from output of previous layer, train all layers jointly

Zero-One Loss

The models given in these deep learning tutorials are largely used for classification. The major aim of training a classifier is to reduce the amount of errors (zero-one loss) on unseen examples

Negative Log-Likelihood Loss

Optimizing it for large models (thousands or millions of parameters) is prohibitively expensive (computationally) because the zero-one loss isn't differentiable. In order to achieve this maximization of the log-likelihood is done on the classifier given all the labels in a training set [14].The likelihood of the correct class and number of right predictions is not the equal, but they are pretty similar from the point of view of a randomly initialized classifier. As the likelihood and zero-one loss are different objectives but we should always see that they are co-related on the validation set but sometimes one will rise while the other falls, or vice-versa.

Stochastic Gradient Descent

Ordinary gradient descent is an easy rule within which we repeatedly create tiny steps downward on an error surface defined by a loss function of some parameters. For the aim of normal gradient descent we take into account that the training data is rolled into the loss function. Then the pseudo code of this algorithm can be represented as stochastic gradient descent (SGD) works according to similar principles as random gradient

descent (SGD) operates on the basis of similar principles as normal gradient descent. It quickly proceeds by estimating the gradient from simply a few examples at a time instead of complete training set. In its purest kind, we use simply one example at a time to estimate the gradient.

Caffe is a deep learning framework or else we can say a library it's made with expression speed and modularity in mind they will put by Berkeley artificial intelligence research and created by young King Gia there is many deep learning or machine learning frameworks for computer vision like tensorflow, Tiano, Chairs and SVM [2]. But why exactly we implement edition cafe there as on is its expressive architecture we can easily switch between CPU and GPU while training on GPUmachine modules and optimization for our problem is defined by configuration without hard coding. It supports extensible code since cafes are open source library. It is four foot by over twenty thousand and developers and github since its birth it offers coding platform in extensible languages like Python and C++. The next reason is speed for training the neural networks speed is the primary constraint. Caffe can process over million images in a single day with the standard media GPU that is milliseconds per image. Whereas the same dataset of million images can take weeks for Tiana and Kara's Caffe is the fastest convolution neural network present community as mentioned earlier since its open source library huge number of research are powered by caffe and every single day something new is coming out of it.

System Requirement

SYSTEMREQUIREMENT:

Install Python on your computer system

1. Install python version 3 and its dependencies like imutils, Numpy, OpenCV, etc.

2. Download the Object Detection model file.

Steps to be followed:-

1) DownloadandinstallPythonversion3fromofficialPythonLanguagewebsite

https://python.org

2) Install the following dependencies via pip:

(i) Imutils:

A series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, displaying Matplotlib images, sorting contours, detecting edges, and much more easier with OpenCV and both Python 2.7 and Python 3.

Pip installs imutils - command

(ii) Numpy:

Numpy is library of Python programming language, adding support for large, multi-dimensional array and matrices, along with large collection of high-level mathematical function to operate over these arrays. The ancestor of Numpy, Numeric, was originally

created by Jim Hugunin with contributions from several developers. In 2005 Travis Olphant created Numpy by incorporating features of computing Numarray into Numeric, with extension modifications. Numpy is open-source software and has many contributors.

Pip installs numpy - command

(iii) OpenCV:

OpenCV is an library of programming functions mainly aimed on real time computer vision. Originally developed by Intel, it is later supported by Willow Garage then Itseez. The library is a cross-platform and free to use under the open-source BSD license.

Pip installs OpenCV-python - command

(iv) Argparse:

The argprase module makes it easy to write user-friendly command-line interfaces. The program defines what arguments it requires, and argparse will figure out how to parse those out of sys.argv. The argparse module also automatically generates help and usage messages and issues errors when users give the program invalid arguments.

Pip installs argprase – command

(v) Time:

The Python time module provides many ways of representing time in code, such as objects, numbers, and strings. It also provides functionality other than representing time, like waiting during code execution and measuring the efficiency of your code.

(vi) Download the MobileNetSSD model file that will be used for the object detection using following link:

https://github.com/gamecoder6/Object_detection

Copy the MobileNetSSD_deploy.prototxt.txt file and MobileNetSSD_deploy.caffemodel file from the github or either clones it.

Real-time object detection

METHODOLOGY:

SqueezeNet:

SqueezeNet is name of a DNN for computer vision. SqueezNet is developed by researchers at DeepScale, University of California, Berkeley, and Stanford University together. In SqueezeNet design, the author's goal is to create a smaller neural network with few parameters that can more easily fit into memory of computer and can more easily be transmitted over a computer network.

SqueezeNet is originally released in 2016. This original version of SqueezeNet was implemented on top of the Caffe deep learning software framework. The open-source research community ported SqueezeNet to a number of other deep learning frameworks. And is released in additions, in 2016, Eddie Bell released a part of SqueezeNet for the Chained deep learning framework. in 2016, Guo Haria released a part of SqueezeNet for the Apache MXNet framework. 2016, Tammy Yang released a port of SqueezeNet for the Keras framework. In 2017, companies including Baidu, Xilnx, Imagination Technologies, and Synopsys demonstrated SqueezedNet running on low-power processing platforms such as smart phones, FPGAs, andcustomprocessors.

SqueezeNet ships as part of the source code of a number of deep learning frameworks such as PyTorch, Apache MXNet, and Apple CoreML.In addition, 3rd party developers have created

implementation of SqueezeNet that is compatible with frameworks such as TensorFlow. Below is summary of frameworks that support SqueezeNet.

DNN Model	Application	Original Implementation	Other Implementatis
SqueezeDet	Object Detection on Images	TensorFlow	Caffe, Keras
SqueezeSeg	Semantic Segmentation of LIDAR	TensorFlow	
SqueezeNext	Image Classification	Caffe	TensorFlow, Keras, PyTorch
SqueezeNAS	Neural Architecture Search for Semantic Segmentation	PyTorch	

TABLE 1

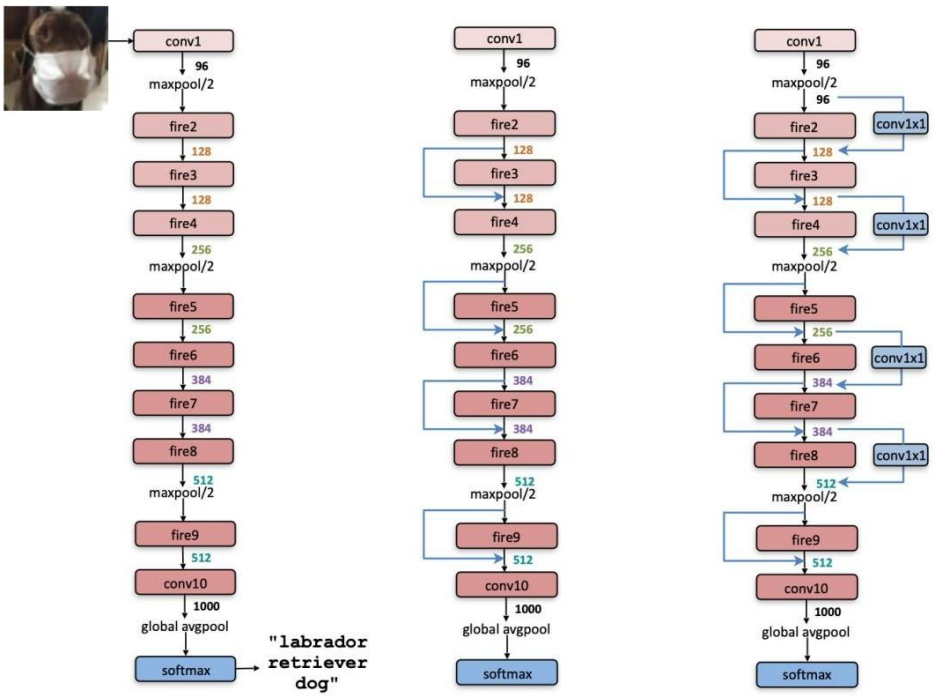
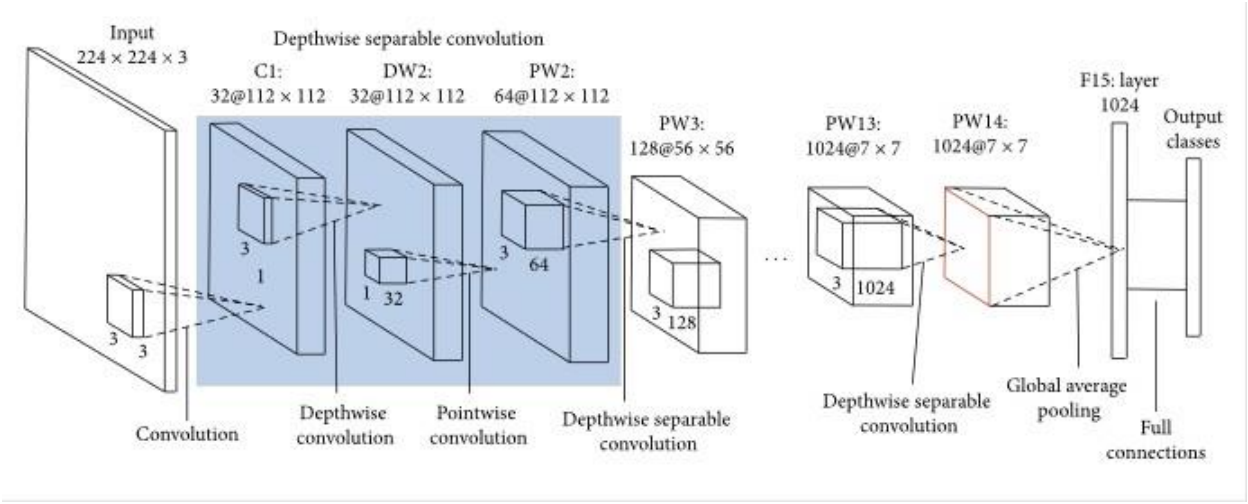


Figure 2: Macroarchitectural view of our SqueezeNet architecture. Left: SqueezeNet (Section 3.3); Middle: SqueezeNet with simple bypass (Section 6); Right: SqueezeNet with complex bypass (Section 6).

Squeezenet Architectural Diagram

DNN (Deep neural Network) used in project:

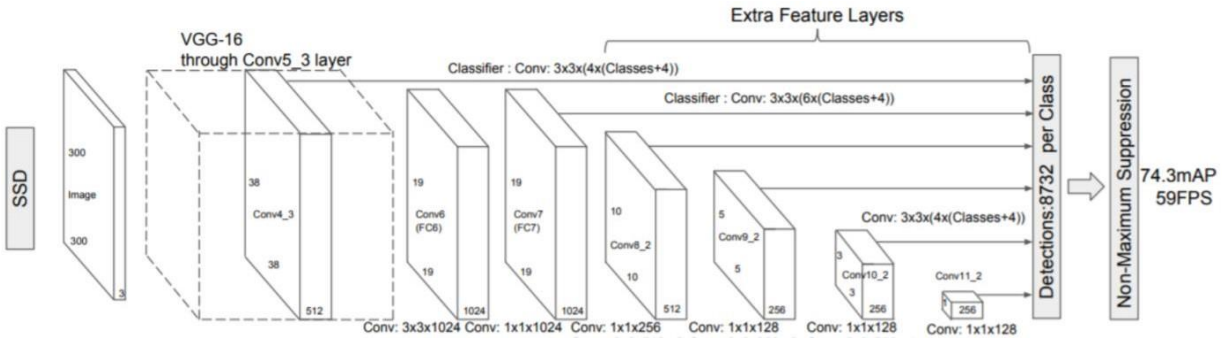
MobileNetSSD:
MobileNet is a light-weight deep neural network architecture designed for mobiles and embedded vision applications.



In many real-world applications such as a self-driving car, the recognition tasks need to be carried out in a timely fashion on a computationally limited device. To fulfill this requirement, MobileNet was developed in 2017.

The core layers of MobileNet is built on depth-wise separable filters. The first layer, which is a full convolution, is an exception.

Around the same time (2016), SSD: Single Shot detector was also developed by Google Research team to cater the need for models that can run real-time on embedded devices without a significant trade-off in accuracy.



Single Shot object detection or SSD takes one single shot to detect multiple objects within the image. The SSD approach is based on a feed-forward convolutional network that produces a fixed-size collection of bounding boxes and scores for the presence of object class instances in those boxes.

It's composed of two parts:
1. Extract feature maps, and
2. Apply convolution filter to detect objects

SSD is designed to be independent of the base network, and so it can run on top of any base networks such as VGG, YOLO, and MobileNet.

In the original paper, Wei Liu and team used VGG-16 network as the base to extract feature maps.

To further tackle the practical limitations of running high resource and power-consuming neural networks on low-end devices in real-time applications, MobileNet was integrated into the SSD framework. So, when MobileNet is used as the base network in the SSD, it became **MobileNet SSD.**

Code and explanation:

```
1  # import the necessary packages
2  from imutils.video import VideoStream
3  from imutils.video import FPS
4  import numpy as np
5  import argparse
6  import imutils
7  import time
8  import cv2
```

From **line 2-8** the necessary packages are installed and imported like imutils, Numpy, OpenCV (cv2), time.

And from imutils. Video we import video stream and FPS from real-time object detection and measuring the frame rate.

Line 5 argprase package is a command line argument. Command line arguments are flags given to a program/script at runtime. They contain additional information for our program so that it can execute

You can draw the analogy that a command line argument is similar to a function parameter.

```
10 # construct the argument parse and parse the arguments
11 ap = argparse.ArgumentParser()
12 ap.add_argument("-p", "--prototxt", required=True,
13     help="path to Caffe 'deploy' prototxt file")
14 ap.add_argument("-m", "--model", required=True,
15     help="path to Caffe pre-trained model")
16 ap.add_argument("-c", "--confidence", type=float, default=0.2,
17     help="minimum probability to filter weak detections")
18 args = vars(ap.parse_args())
19
```

--prototxt: The path to the Caffe prototxt file.

--model: The path to the pre-trained model.

--confidence: The minimum probability threshold to filter weak detections. The default is 20%.

```
20  # initialize the list of class labels MobileNet SSD was trained to
21  # detect, then generate a set of bounding box colors for each class
22  CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
23      "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
24      "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
25      "sofa", "train", "tvmonitor"]
26  COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))
```

Lines 21-25 build a list called CLASSES containing our labels. This is followed by a list, COLORS **which contains corresponding random colors for bounding boxes (**Line 26**)?**

```
28  # load our serialized model from disk
29  print("[INFO] loading model...")
30  net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
31
32  # initialize the video stream, allow the cammera sensor to warmup,
33  # and initialize the FPS counter
34  print("[INFO] starting video stream...")
35  vs = VideoStream(src=0).start()
36  time.sleep(2.0)
37  fps = FPS().start()
```

We load our serialized model, providing the references to our prototxt and model files on **Line 30** — notice how easy this is in OpenCV 3.3.

Next let's initialize our video stream (this can be from a video file or a camera). First we start the VideoStream (**Line 35**), then we wait for the camera to warm up (**Line 36**), and finally we start the frames per second counter (**Line 37**). The VideoStream and FPS classes are part of imutils packages.

```
41  while True:
42      # grab the frame from the threaded video stream and resize it
43      # to have a maximum width of 400 pixels
44      frame = vs.read()
45      frame = imutils.resize(frame, width=400)
46
47      # grab the frame dimensions and convert it to a blob
48      (h, w) = frame.shape[:2]
49      blob = cv2.dnn.blobFromImage(cv2.resize(frame, (1000, 1000)),
50          0.007843, (300, 300), 127.5)
51
52      # pass the blob through the network and obtain the detections and
53      # predictions
54      net.setInput(blob)
55      detections = net.forward()
```

First, we read a frame (**Line 44**) from the stream, followed by resizing it (**Line 45**).

Since we will need the width and height later, we grab this now on **Line 48**.This is followed by converting the frame to a blob with the den module (**Lines 49 and 50**).Now for the heavy lifting: we set the blob as the input to our neural network (**Line 54**) and feed the input through the net (**Line 55**) which gives us our detections.

At this point, we have detected objects in the input frame. It is now time to look at confidence values and determine if we should draw a box + label surrounding the object,

```
57      # loop over the detections
58      for i in np.arange(0, detections.shape[2]):
59          # extract the confidence (i.e., probability) associated with
60          # the prediction
61          confidence = detections[0, 0, i, 2]
62
63          # filter out weak detections by ensuring the `confidence` is
64          # greater than the minimum confidence
65          if confidence > args["confidence"]:
66              # extract the index of the class label from the
67              # `detections`, then compute the (x, y)-coordinates of
68              # the bounding box for the object
69              idx = int(detections[0, 0, i, 1])
70              box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
71              (startX, startY, endX, endY) = box.astype("int")
72
73              # draw the prediction on the frame
74              label = "{}: {:.2f}%".format(CLASSES[idx],
75                  confidence * 100)
76              cv2.rectangle(frame, (startX, startY), (endX, endY),
77                  COLORS[idx], 2)
78              y = startY - 15 if startY - 15 > 15 else startY + 15
79              cv2.putText(frame, label, (startX, y),
80                  cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)
```

We start by looping over our detections, keeping in mind that multiple objects can be detected in a single image. We also apply a check to the confidence (i.e., probability) associated with each detection. If the confidence is high enough (i.e. above the threshold), then we'll display the prediction in the terminal as well as draw the prediction on the image with text and a colored bounding box. Let's break it down line-by-line:

Looping through our detections, first we extract the confidence value (**Line 62**).

If the confidence is above our minimum threshold (**Line 66**), we extract the class label index (**Line 70**) and compute the bounding box coordinates around the detected object (**Line 71**).

Then, we extract the *(x, y)*-coordinates of the box (**Line 72**) which we will use shortly for drawing a rectangle and displaying text.

We build a text label containing the CLASS name and

the confidence (**Lines 75 and 76**).

Let's also draw a colored rectangle around the object using our class color and previously extracted *(x, y)*-coordinates (**Lines 77 and 78**).

In general, we want the label to be displayed above the rectangle, but if there isn't room, we'll display it just below the top of the rectangle (**Line 79**).

Finally, we overlay the colored text onto the frame using the *y*-value that we just calculated (**Lines 80 and 81**).

The remaining steps in the frame capture loop involve (1) displaying the frame, (2) checking for a quit key, and (3) updating our frames per second counter:

```
82      # show the output frame
83      cv2.imshow("Frame", frame)
84      key = cv2.waitKey(1) & 0xFF
85
86      # if the `q` key was pressed, break from the loop
87      if key == ord("q"):
88          break
89
90      # update the FPS counter
91      fps.update()
92
93  # stop the timer and display FPS information
94  fps.stop()
95  print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
96  print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))
97
98  # do a bit of cleanup
99  cv2.destroyAllWindows()
100 vs.stop()
```

The above code block is pretty self-explanatory — first we display the frame (**Line 84**). Then we capture a key press (**Line 85**) while checking if the 'q' key (for "quit") is pressed, at which point we break out of the frame capture loop (**Lines 88 and 89**).

Finally we update our fps counter (**Line 91**).

When we've exited the loop, we stop the fps counter (**Line 94**) and print information about the frames per second to our terminal (**Lines 95 and 96**).We close the open window (**Line 99**) followed by stopping the video stream (**Line 100**).

Result of the program

Result:

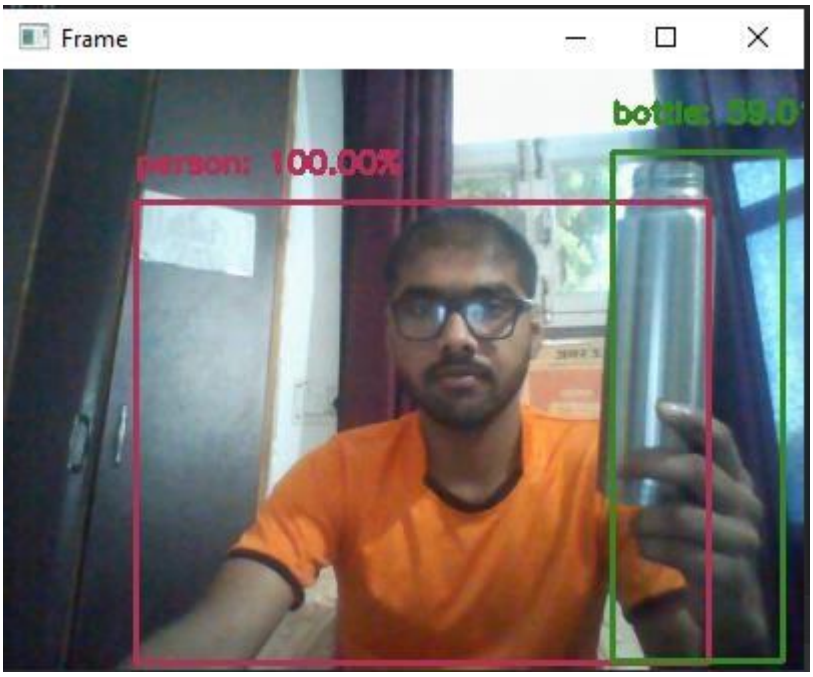


Fig: Real-time object detection

Fig: Console/terminal image of above detection

CONCLUSION:

By using this thesis and based on experimental results we are able to detect object more precisely and identify the objects individually. This Report also provides experimental results on different methods for object detection and identification and compares each method for their efficiencies.

FUTURE ENHANCEMENTS

The object recognition system can be applied in the area of surveillance system, face recognition, fault detection, character recognition etc. The objective of this thesis is to develop an object recognition system to recognize the 2D and 3D objects in the image/VideoStream. The performance of the object recognition system depends on the features used and the classifier employed for recognition.

As a scope for future enhancement,

- Features either the local or global used for recognition can be increased, to increase the efficiency of the object recognition system.
- Geometric properties of the image can be included in the feature vector for recognition.150
- Using unsupervised classifier instead of a supervised classifier for recognition of the object.
- The proposed object recognition system uses grey-scale image and discards the color information. The color information in the image can be used for recognition of the object. Color based object recognition plays vital role in Robotics

  Although the visual tracking algorithm proposed here is robust in many of the conditions, it can be made more robust by eliminating some of the limitations as listed below:
- In the Single Visual tracking, the size of the template

remains fixed for tracking. If the size of the object reduces with the time, the background becomes more dominant than the object being tracked. In this case the object may not be tracked.

- Fully occluded object cannot be tracked and considered as a new object in the next frame.

- For Night time visual tracking, night vision mode should be available as an inbuilt feature in the CCTV camera.

To make the system fully automatic and also to overcome the above limitations, in future, multi- view tracking can be implemented using multiple cameras. Multi view tracking has the obvious advantage over single view tracking because of wide coverage range with different viewing angles for the objects to be tracked.

In this thesis, an effort has been made to develop an algorithm to provide the base for future applications such as listed below.

- In this research work, the object Identification and Visual Tracking has been done through the use of ordinary webcam. The concept is well extendable in applications like Intelligent Robots, Automatic Guided Vehicles, Enhancement of Security Systems to detect the suspicious behavior along with detection of weapons, identify the suspicious movements of enemies on boarders with the help of night vision cameras and many such applications.

- Foreground object extraction depends on the binary

segmentation which is carried out by applying threshold techniques. So blob extraction and tracking depends on the threshold value.

- Splitting and merging cannot be handled very well in all conditions using the single camera due to the loss of information of a 3D object projection in 2Dimages.

- In the proposed method, background subtraction technique has been used that is simple and fast. This technique is applicable where there is no movement of camera. For robotic application or automated vehicle assistance system, due to the movement of camera, backgrounds are continuously changing leading to implementation of some different segmentation techniques like single Gaussian mixture or multiple Gaussian mixture models.

- Object identification task with motion estimation needs to be fast enough to be implemented for the real time system. Still there is a scope for developing faster algorithms for object identification. Sucalgorithms can be implemented using FPGA or CPLD for fast execution

References/Bibliography:

- **Robust Real-time Object Detection By Paul Viola Michael J. Jones (https://www.hpl.hp.com/techreports/Compaq-DEC/CRL-2001-1.pdf)**

- **Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun (https://arxiv.org/abs/1506.01497)**

- **Deep Learning and its application to CV and NLP By Fei Yan University of Surrey (https://udrc.eng.ed.ac.uk/sites/udrc.eng.ed.ac.uk/files/attachments/Deep%20Learning%20and%20its%20application%20to%20CV%20and%20NLP_0.pdf)**

- **www.Youtube.com**

- **www.wikipedia.com**

- **www.google.com**