

# **Assisting the Visually Impaired People Using Image Processing**



This thesis was submitted in partial fulfillment of the requirement for the degree of

## **Bachelor of Computer Science and Engineering**

Under the Supervision of

**Dr Jia Uddin**

Assistant Professor, BRAC University

By

**Ferdousi Rahman** (15201004)

**Israt Jahan Ritun** (14101114)

**Nafisa Farhin** (14101113)

Department of Computer Science and Engineering

July 2018

BRAC University, Dhaka, Bangladesh

---

# Declaration

We, hereby declare that, this paper titled “Assisting the Visually Impaired People using Image Processing” bases on the results that we have derived ourselves throughout our research. Materials of work from researches conducted by others are mentioned in the references.

## Signature of Supervisor

---

Dr Jia Uddin

Assistant Professor

Department of Computer Science

and Engineering.

BRAC University.

## Signature of Authors

---

Ferdousi Rahman

ID: 15201004

---

Israt Jahan Ritun

ID: 14101114

---

Nafisa Farhin

ID: 14101113

---

## **Acknowledgement**

Our thesis focuses on object and face detection using different algorithms for higher accuracy by which can support blind people to maximum capacity. Regarding this thesis, our first and foremost appreciation goes to Almighty and our parents for making us capable to conduct this thesis.

Moreover, we are very pleased to acknowledge our utmost gratitude to the supervisor of this thesis, Dr. Jia Uddin, Assistant Professor, Department of Computer Science and Engineering, for his immense support and guidance throughout the conduct of the thesis. It has been an honor to work under his supervision and complete out thesis work.

Last but not the least, our special gratefulness to all the faculty members of BRAC University for sharing their knowledge and assistance for an enriched graduation period of our lives. They have blessed us with creative concept development and profound understanding.

## List of Figures

- Fig. 2.1 Basic convolutional layers diagram of the system of YOLO algorithm
- Fig. 2.2 Comparative Analysis of different Algorithms in the field of Image Processing and Detection
- Fig. 3.1 Basic Flowchart of Image processing for Identification
- Fig. 3.2 CPU Installation of tensorflow through anaconda prompt
- Fig. 3.3 Code Simulation Terminal for Customize Dataset Creation
- Fig. 3.4 Bounding box color declaration
- Fig. 3.5 Code Simulation Terminal for Customize Dataset Creation
- Fig. 3.6 Flowchart of the basic identification workflow
- Fig. 3.7 Command for training activation in command prompt
- Fig. 3.8 Training compilation
- Fig. 3.9 Accuracy at the initial stage of compilation of identification
- Fig. 3.10 Basic Convolutional Architecture for MTCNN Algorithm
- Fig. 3.11 Work Flow of Face Detection and Identification
- Fig. 3.12 Different Convolutional Layers declared in the mtcnn\_detect class
- Fig. 3.13 Code portion to transfer images to convolutional layers
- Fig. 3.14 Terminal for taking new face data input
- Fig. 3.15 Dataset from our input
- Fig. 3.16 Code compilation and Face identification with Name Label and Accuracy Rate

- Fig. 3.17 Code portion to draw bounded boxes
- Fig. 4.1 Basic Block Diagram for the hardware setup
- Fig. 4.2 Elements with basic connection wiring
- Fig 4.3 Basic functions required for real-time video input and audio the result output
- Fig 5.1 Detected object with name label and accuracy rate
- Fig 5.2 Detected face with name label and accuracy rate
- Fig 5.3 Detected face with using Pi camera

## List of Abbreviations

FLDA	Fisher's Linear Discriminant Analysis
Fddb	Function Designator Data Base
KVB	Kinect Vision Blind
MTCNN	Multi-task Cascaded Neural Networking
R-CNN	Region-based Convolutional Neural Networks
SPP	Spatial Pyramid Pooling
SSD	Single Shot Detector
OCR	Optical Character Recognition
YOLO	You Only Look Once

# Table of Contents

<b>Acknowledgement.....</b>	<b>ii</b>
<b>List of Figures.....</b>	<b>iii</b>
<b>List of Abbreviations.....</b>	<b>v</b>
<b>Abstract.....</b>	<b>1</b>
 <b>Chapter 1: Introduction</b>	
<b>1.1 Motivation.....</b>	<b>2</b>
<b>1.2 Literature Review.....</b>	<b>3</b>
<b>1.3 Objective.....</b>	<b>4</b>
<b>1.4 Thesis Overview.....</b>	<b>4</b>
 <b>Chapter 2: Background Study</b>	
<b>2.1 Different Image Processing Algorithms Versus Preferred Algorithm.....</b>	<b>5</b>
<b>2.1.1 Object Detection Algorithms.....</b>	<b>5</b>
<b>2.1.1.1 Single pyramid pooling (SPP-net).....</b>	<b>5</b>
<b>2.1.1.2 Faster R-CNN (Region-based Convolutional Neural Networks).....</b>	<b>6</b>
<b>2.1.1.3 Single Shot Detector (SSD).....</b>	<b>6</b>
<b>2.1.1.4 YOLO (You only Look Once).....</b>	<b>7</b>
<b>2.1.2 Face Identification Algorithms.....</b>	<b>8</b>
<b>2.1.2.1 Eigenfaces and Fisherfaces Algorithms.....</b>	<b>8</b>
<b>2.1.2.2 AdaBoost Algorithms.....</b>	<b>8</b>
<b>2.1.2.3 MTCNN (Multi-task Cascaded Neural Networking).....</b>	<b>9</b>
<b>2.2 Different Real-Time Methodologies with hardware implementation.....</b>	<b>9</b>
<b>2.3 User Compatibility and Accuracy analysis.....</b>	<b>10</b>

## Chapter 3: Object and Image Identification

<b>3.1</b>	Object Detection and Identification Methodology.....	11
<b>3.1.1</b>	Environment Setup.....	12
<b>3.1.2</b>	Detection and Identification processing.....	13
<b>3.1.3</b>	Custom-made Dataset Construction.....	15
<b>3.1.4</b>	Machine Training.....	16
<b>3.1.5</b>	Accuracy Analysis and Development.....	18
<b>3.2</b>	Face Identification Methodology.....	19
<b>3.2.1</b>	Installation Setup.....	19
<b>3.2.2</b>	Customized Dataset Development.....	20
<b>3.2.2.1</b>	Argument Parsing.....	20
<b>3.2.2.2</b>	Dataflow Graph.....	20
<b>3.2.2.3</b>	Face Alignment.....	20
<b>3.2.2.4</b>	Face-Feature Extraction.....	22
<b>3.2.2.5</b>	Detection using MTCNN.....	22
<b>3.2.3</b>	Loading MTCNN Face Detection Model.....	22
<b>3.2.3.1</b>	Proposal Network (P-Net).....	22
<b>3.2.3.2</b>	Refine Network (R-Net).....	23
<b>3.2.3.3</b>	Output Network (O-Net).....	23
<b>3.2.4</b>	Loading New Face Data.....	24
<b>3.2.5</b>	Compilation and Result Display.....	25



## **CHAPTER 4: HARDWARE SETUP**

<b>4.1</b>	Structure Establishment Process.....	27
<b>4.1.1</b>	Raspberry Pi 3 Model B Installation.....	28
<b>4.1.2</b>	Raspberry Pi Camera Model v2 and Earphone Connection.....	29
<b>4.1.3</b>	Input and Output Processing Methodologies and Libraries.....	29

## **CHAPTER 5: RESULT ANALYSIS**

<b>5.1</b>	Object Identification Result.....	31
<b>5.2</b>	Face Identification Result.....	32
<b>5.3</b>	Face Detection Result from Raspberry Pi.....	33

## **CHAPTER 6: CONCLUSION**

<b>6.1</b>	Implementation Challenges.....	35
<b>6.2</b>	Future Directions.....	35

<b>REFERENCES.....</b>	<b>37-39</b>
------------------------	--------------

## Abstract

Visually impaired people face difficulties in safe and independent movement which deprive them from regular professional and social activities in both indoors and outdoors. Similarly they have distress in identification of surrounding environment fundamentals. The proposed thesis suggests of detection of brightness and the major colors in real-time image by using RGB method by means of an external camera and thus identification of fundamental objects as well as facial recognition from personal dataset. For the Object identification and Facial Recognition, YOLO Algorithm and MTCNN Networking are used respectively. The software support is achieved by using OpenCV libraries of Python as well as implementing machine learning process. The major processor of our thesis, Raspberry Pi scans and detects the facial edges via Pi camera and objects in the image are captured and recognized using mobile camera. Image recognition results are transferred to the blind users by means of text-to-speech library. The device portability is achieved by using a battery. The object detection process achieved 8-15 FPS processing with an accuracy rate of 63-80%. The face identification process achieved 80-100% accuracy. The objective of the thesis is to give blind users the capability to move around in unfamiliar indoor environment, through a user friendly device by face and object identification system.

**Keywords** - *Visually Impaired, OpenCV, Image Processing, Object Detection, Face Detection, YOLO Algorithm, Deep Learning, Multi-task Cascaded Neural Networking.*

# CHAPTER 1

## Introduction

According to statistical analysis study of WHO (World Health Organization) [1], approximately 285 million people around the world are blind or have amblyopia, 246 million of whom have serious vision problems. Visually impaired people usually face difficulties in movement as well as identifying people and avoiding obstacles in their day to day activities. The conventional solutions to these situations are often seen to be usage of guide canes to detect obstacles in front of them or rely on vocal guessing for identification of persons. As an outcome, visually impaired people cannot predict the exact environment features about what types of objects lies in front of them or whom they are facing presence.

From previous study, many approaches had been discovered and implemented involving both hardware (wearable gadgets) and software (Smart phone apps) [33] which are discussed in Literature Review of this paper. But most of the previous methodologies lack the maximum efficiency of combination among the accuracy of data processed output with the user appliance. In this paper, the approach is to construct a module to feed the user with vital data such as individual identification and obstacle detection with better accuracy. Our proposed method is based on Deep Learning Algorithms and compatible and user friendly hardware for practical implementation.

This paper is serialized as follows; Chapter 2 contains the former works done which are familiar to our work plan as well as discuss the previous approaches of the algorithms and hardware setup notions. Chapter 3 and 4 we have proposed our methodology of software and hardware implementation respectively. In Chapter 5 we have displayed the practical result of our work with proper justification and finally a conclusion with the challenges faced and a future direction in Chapter 6.

### 1.1 Motivation

For the growing competitive world around, it is quite difficult for a visually impaired person to move around independently and identify surrounding objectives correctly with ease. With the

advancement of technology, there are several solutions but most of them have demerits such as low acceptance, high cost, difficult to usage etc. [2]. Based on the demand, devices supporting the visually impaired people has been introduced a long time now. On the other hand, keeping with pace, the more advanced algorithms and processing devices are introduced and progressing to higher accuracy and efficiency. This has inspired us to combine the concepts of implementing a processing device for serving the blind individuals with a higher efficient methodology.

## **1.2 Literature Review**

In different period of time versatile approaches has been in use for detect objects or identify people and support visually impaired individuals. This sector is highly involving image processing technology as visual activity is involved. In 2014, a thesis named Smart Vision objected to support blind users with the features of movement within unacquainted surrounding [3]. Another thesis [4] used different individual device portions for indoor and outdoor movements as well as incorporated GPS to track the coordinates of the position place of the user. The software implementation was done based on MATLAB. Also there are smartphone based guiding systems with obstacle identification and multiple modes for convenient user interfacing modes [5]. This thesis gave us the knowledge of how to handle continuously captured images and keep processing them for identification results. The process involved usage of GPU for deep computing method. The algorithm of this thesis is YOLO (You Only Look Once) which is efficient for real-time robust system development. Here the processing server is separate from the input device. There has to a negotiation connection over the cellular device and the server side database which has to be on online mode for maximum continuous facility.

Processing texts from real-time image is also a breakthrough for this field [6]. The thesis used Tesseract Optical Character Recognition (OCR) for text recognition from object's label for more fitting comfort for the blind users. This approach gave us the idea of pattern recognition from database provided. In 2017, face recognition from comparing with the database was also an appreciated paper in the IEEE paper conference [7]. There Kinect Vision Blind (KVB) was applied; the KVB is a multiyear long-term open-source thesis terminal for advanced application of technology. Face detection, face recognition and audio output module was merged in a portable and wearable device. From all these works we have gathered concepts to build our thesis around it and put image processing in an applicable way to support the visually impaired.

With the flow of technology advancement, more efficient algorithms, methods and systems are proposed and developed for robust service as well as real time detection with minimum delay and maximum accuracy. OpenCV has been a first-rate approach for image processing and gives a tranquil solve to our stated problem. In a journal publication [8], using the OpenCV libraries and basing on AdaBoost algorithm, the images are classified and pattern recognition feature of this algorithm gives advantage to use Haar Like features and output proper results. Another publication [9] also used the OpenCV application and emphasized on the object detection based on hue, saturation and color value (HSV) range. These works were inspirational and prodigious guidance for our target solution and marked the hindrance that we might face on our progress so our effort started based on those drawbacks to come up with more progressive solution and develop a smart system to support the blind people.

### **1.3 Objective**

Our objectives are to implement two different algorithms which are YOLO and MTCNN; individually for face identification and object detection as well as establish a hardware setup for hands-on result of the executed software program. We will also state the detailed description of the steps involved for the coding execution as well as result analysis with accuracy rate after the system training and testing session.

### **1.4 Thesis Overview**

In our thesis paper, Chapter 1 is for introduction to our thesis concept and implemented algorithm, motivation and objective overview. Chapter 2 holds the background analysis of previous works related to the concept and methods. Next Chapter 3 describes the extensive explanation of the software implementation of the system followed by Chapter 4 that explains practical implementation of the hardware setup. Then Chapter 5 highlights the results of our simulation. Finally Chapter 6 is the conclusion of our paper with a future direction.

## CHAPTER 2

### Background Study

The basic concept of facial and object detection system is a very commonly known factor. Not only for the aiding of visually impaired people, this notion is in implementation in many sectors such as security and industrial manufacturing. The efficiency and accuracy differs by the algorithm and processing functions. Different software system models are designed as such that it firstly takes the input images fetching from the database and implement the deep learning process to classify and then specifically identify the required result such as objects, facial identity or expression, forgery etc. for the real-time circumferences, the captured input images contain several entities and needs more efficient program to extract and derive the specified category and sometimes for real-time analysis, multiple detection are identified and it is a challenge to identify correctly. In this chapter we are discussing some of the algorithms and their proposed methodologies to state the base of our thesis analysis.

#### 2.1 Different Image Processing Algorithms versus Preferred Algorithm

Different algorithms have been used to implement our proposed system in different times executed with different processing devices. Below are some algorithms pronounced that have been implemented for our suggested system and our preferred nominated algorithm described with proper justification of decide on.

##### 2.1.1 Object Detection Algorithms

###### 2.1.1.1 Spatial Pyramid Pooling (SPP-net)

Generally known as SPP-net, it is an image classification method with the advantage of generating fixed-length presentation regardless the input image scale [13]. This processing method is significant in object detection. The other R-CNN methods crop or wrap the image which might not contain the overall object but SPP-net differs as feature maps are generated. In the paper [13] it has been executed that the feature map computation of the input is done only once and repetition is avoided, thus faster than R-CNN method still better accuracy achieved on Pascal VOC 2007.

### **2.1.1.2 Faster Region-based Convolutional Network (Faster R-CNN)**

The Faster Region-based Convolutional Network (Faster R-CNN) is an advanced concept of image processing which has derived from the combine idea of the RPN and the Fast R-CNN model. An RPN (Region Proposal Network) is a Convolutional Neural Network that takes image data and derives to a convolutional feature map whereas the Fast R-CNN is developed over previous concepts but with more efficiency to classify object proposals using deep convolutional networks [11]. This paper in [11] has worked over a dataset called PASCAL and with the shared convolutional features, attempted to get maximum accuracy and possible optimization. This process bases on GPU which results in time proficiency. But usage and availability of GPU is a challenge as well as expensive also has a trade-off to the accuracy [12].

### **2.1.1.3 Single Shot Detector (SSD)**

Single Shot Multi-Box Detector; also known as SSD, uses a single convolutional neural network to detect the object in an image [16]. While accurate, the above mentioned approaches have been too computationally exhaustive for embedded systems and, even with high-end hardware; those are too slow for instantaneous applications compared to SSD. The fundamental concept of SSD is predicting category scores and box offsets for a fixed set of default bounding boxes using small convolutional filters applied to feature maps. In the paper [16], for the detection accuracy, predictions of different measures from feature maps of different scales are thesised which are explicitly separate predictions by aspect ratio. These design features lead to simple end-to-end training even on low resolution input data, further refining the speed vs. accuracy trade-off. Addition of multiple convolution feature layers to the end of the base network to predict detections at multiple scales has been an effective effort on identification thesiss and SSD can detect various object categories and Compared to R-CNN, SSD has less localization error.

### 2.1.1.4 YOLO (You only Look Once)

You Only look Once or also known as YOLO is a real-time object detection algorithm that uses a single convolutional network and is much more faster compared to the other identification systems, even in real-time, it performs 150fps, thus it's capable of processing live streaming video with less than 25 milliseconds of latency. Along with the fast processing time efficiency, this method still holds a respectable measurement of accuracy in identification. In this process, a single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. The basic workflow of YOLO is shown in Fig.2.1 below:

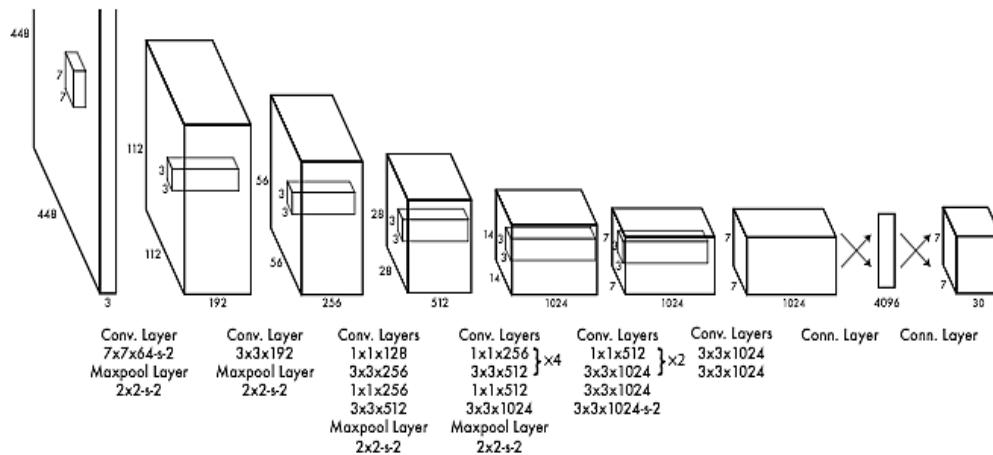


Fig.2.1: Basic convolutional layers diagram of the system of YOLO algorithm

In the paper [17], YOLO has been established particularly fast system as it can process 45 frames per second in real-time also it has been outpacing the previous identification systems for its time-efficiency. The paper also represents the system as a single convolutional network instantaneously forecasting multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images, straight optimizing detection performance. This unified model has standout from traditional methods of object detection. For training purposes, Darknet Framework has been implemented. We also came to know some drawbacks of YOLO from this paper which are, during the processing, each grid cell only predicts two boxes and can only have one class. This spatial restraint limits the number of predicted nearby objects. YOLO also struggles with small objects.



A more rationalized and efficient approach was introduced in 2017, YOLO9000, which could detect and identify more than 9000 categories of objects in real-time [18]. This system was trained with both COCO detection dataset and the ImageNet classification dataset which gave the system advantage of more precise identification. This version implemented Batch Normalization which resulted in performance improvement for approximately 2% [18]. Also the hierarchical view of object classification permits to combine different datasets together. The combination of datasets is done by WordTree which is a visualization technique for text-based data through a visual branching structure.

## **2.1.2 Face Identification Algorithms**

### **2.1.2.1 Eigenfaces and Fisherfaces Algorithms**

The Eigenface is the first successful system for face identification. The Eigenface method uses Principal Component Analysis (PCA) to linearly thesis the data which is image space to a low dimensional feature space. The Fisherface method is an improvement of the Eigenface technique that it uses Fisher's Linear Discriminant Analysis (FLDA or LDA) for the dimensionality reduction. The LDA maximizes the ratio of between-class scatter to that of within-class scatter; therefore, it works better than PCA for purpose of discrimination. The Fisherface is especially useful when facial images have large variations in illumination and facial expression. In paper [21], keeping the factor of security and authentication in mind, an attendance system has been developed implementing the Eigenfaces algorithm using Open CV 2.4.8. The paper also concludes the result that Eigenfaces perform better than Fisherfaces with higher accuracy number. Another paper [22] reversely stated with broad experimental results demonstration that the Fisherface method has lower error rates than those of the Eigenface technique for tests on the Harvard and Yale face databases.

### **2.1.2.2 AdaBoost Algorithms**

Adaptive Boosting, usually known as AdaBoost algorithm, is a machine learning meta-algorithm used in combination with other types of detection or identification algorithms to improve overall performance such as higher FPS and identification accuracy. This practical boosting is used in many real-time approaches to increase the efficiency which is measured by speed and precision. In paper [25], face is detected from images using the parameter of skin tone

color. This approach detects human face in different scales and lightning conditions by removing noisy regions and extracting the human face region from the images using Cascade Classifiers. Another paper [27] explains a three staged process where the second stage includes selection of a small number of critical visual features from a very large set of potential features using AdaBoost learning algorithm for real-time.

### **2.1.2.3 MTCNN (Multi-task Cascaded Neural Networking)**

Convolutional Neural Network or CNN is an artificial neural networking structure designed to analyze images and visual dataset. MT-CNN or Multitask Cascaded Neural Networking is an advanced version of CNN. In paper [26], the proposed model of MTCNN exploits the inherent correlation between detection and alignment of input frames to boost up the performance. This major research gave us a lot of idea about the concept of MTCNN. Multi Task Cascaded Convolutional Network brings relation between detection and alignment. this process achieves superior accuracy over the state-of-the-art techniques on the challenging FDDB and WIDER FACE benchmarks for face detection, and AFLW benchmark for face alignment, while keeps real time performance. It requires bounding box calibration from face detection with extra computational expense and ignores the inherent correlation between facial landmarks localization and bounding box regression. Compared to other multi-class objection detection and classification tasks, face detection is a challenging binary classification task, so it may need less numbers of filters per layer, by decreasing the filter value this algorithm process can increase the depth so it will accelerate the performance of this algorithm.

## **2.2 Different Real-Time Methodologies with Hardware Implementation**

Real-time approaches are done implementing Raspberry Pi in different scholar attempts. As Raspberry Pi is a user approachable and fast processing unit, it is vastly used in these kinds of methodologies. In paper [19], we studied an approach based on Raspberry Pi and camera to capture image and identify cars and humans and notify the blind user with an audio signal output. This paper constructed over the Haar like algorithm as it can work better than pixel-based algorithm thus deriving into 96% accuracy. Another approach we have taken in knowledge in paper [20] which works on speeded up robust feature (SURF) algorithm which is efficient in processing time and identifying feature points of currency notes. For the low vision patients, the

Raspberry Pi processor identifies text via image processing and gives audio output to the user with identified result.

### 2.3 User Compatibility and Accuracy Analysis

Different researchers have compared the image processing algorithm's performance and analyzed the accuracy based on particular datasets that has been guidance for us to determine the algorithm and processing training route for completing the desired thesis aim. Fig.2.2 shows a speed and accuracy comparison among some renowned algorithms.

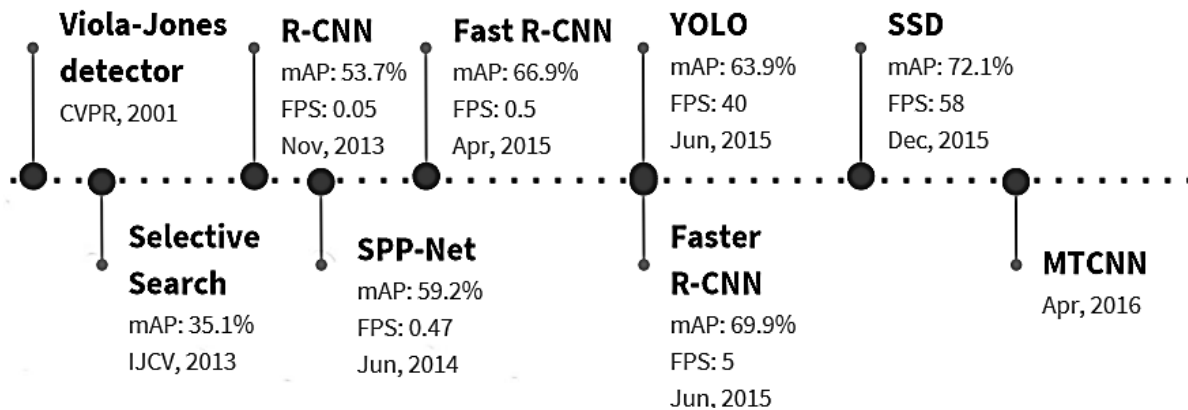


Fig. 2.2: Comparative Analysis of different Algorithms in the field of Image Processing and Detection

## CHAPTER 3

### Algorithm Implementation

Before identifying any object or facial identity from input images or real-time data, it is first essential to detect and extract the target region from the original picture frames. There are several extracting algorithms for these sectors of image processing. Also the collection of dataset needs a pre-processing to normalize the dimensions for better training afterwards that result in improved accuracy. Below resides the basic workflow that we are maintaining for both object and face identification:



Fig.3.1: Basic Flowchart of Image processing for Identification

#### 3.1 Object Detection and Identification Methodology

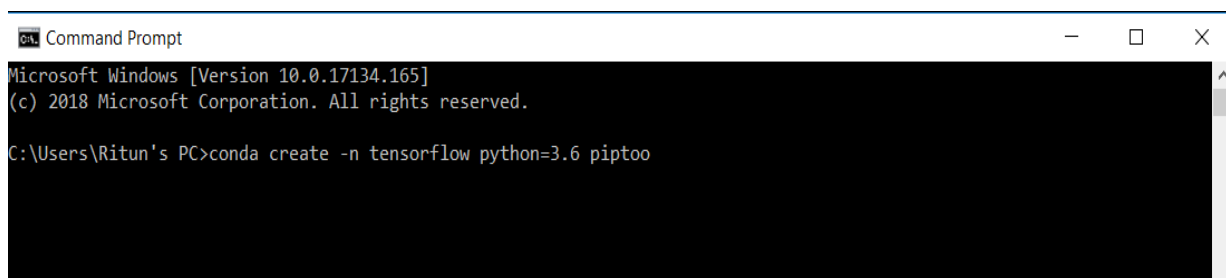
YOLO stands for you only look once. It's a jointly trained method for object detection and classification for real time video. Using this method YOLOv2 is trained simultaneously on the COCO detection dataset and ImageNet classification dataset. It is basically offering easy tradeoff between speed and accuracy and for this reason it's one of the most efficient algorithms for object detection in real time. For example YOLOv2 gets 40 FPS at 78.6 mAP on VOC 2007 which is really fast for real time [32].

Yolo divides up the image into a grid of 13x13 cells and each of the cells is responsible for predicting 5 bounding boxes. A bounding box describes the rectangle that encloses an object. Yolo also outputs a confidence score that tells how certain it is that the predicting bounding box actually encloses some objects. For each bounding box the cell also predicts a class. The

confidence score for the bounding box and the class prediction are combined into one final score that tells the probability that this bounding box contains a specific type of object. Since there are  $13 \times 13 = 169$  grid cells and each cell predicts 5 bounding boxes, it ends up with 845 bounding boxes in total which is a lot. But in final result there will be just those bounding boxes which have higher score value than the threshold value. Even though there were 845 separate predictions, they all made at the same time and the neural network just run once and that's why YOLO is so powerful and fast.

Yolo v2, also known as YOLO9000 is a high speed real-time video detection algorithm with approximately 60 frames per second but with trade-off of accuracy. The first version of YOLO is effective on huge amount of dataset whereas YOLO9000 uses a limited dataset range which is well-suited for our indoor object identification for blind people but the advantage of this system is time efficiency. Our object detection and identification procedure is described in below multiple steps:

**3.1.1 Environment Setup** The original version of YOLO was implemented on Darknet [23] which a Deep learning framework written in C programming language and uses CUDA [30]. But as this language is not user friendly so we have executed our program in python language. Python is also compatible with hardware implementation and easy to execute. For our program development, we used *Darkflow* framework, *Tensorflow* version. We have built the library



```
Command Prompt
Microsoft Windows [Version 10.0.17134.165]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Ritun's PC>conda create -n tensorflow python=3.6 pip
```

Fig 3.2: CPU Installation of tensorflow through anaconda prompt

needed through command prompt. Fig 3.2 has shown CPU Installation of tensorflow through anaconda prompt. For the initial simulation, we installed python 3.6 version for writing the program code file. The application that we have used to render our code is Visual Studio Code.

There are other rendering applications such as Spyder, Jupyter etc but these are less compatible in terms of some library usage and generate errors in the simulation. As we are implementing CPU based program, Tensorflow is outstandingly well-suited open source machine learning framework. With pip install in command, we setup the Tensorflow in the environment. For system application, *OpenCV* (Open Source Computer Vision) which is a library of programming functions mainly designed for real-time computer vision is required, but as it is not included in Anaconda, we downloaded separately. After the environment settlement, we download the prerequisite the weight corresponding file for the dataset. As we are using *tinyYOLO* dataset, we collected the weight file and required .cfg file which are tiny-yolo-voc.weights file and tiny-yolo-voc cfg file from the website of YOLO [23].The weights are loaded from the files to the system model from command terminal. After completing our environment setting we have downloaded a short video clip and also rendered the video and added boxes to check our pre trained model is working or not properly. Here we are calling YOLO from command line and saved the video so that we can see the result later.

```

1 import cv2
2 from darkflow.net.build import TFNet
3 import numpy as np
4 import time
5
6 option = {
7     'model': 'cfg/tiny-yolo-voc.cfg',
8     'load': 'bin/tiny-yolo-voc.weights',
9     'threshold': 0.3
10 }
11 tfnet = TFNet(option)

```

Fig 3.3: Code portion of weight load & package import

**3.1.2 Detection and Identification Processing in Real-time** YOLOv2 tends to derive more accuracy still maintaining the time efficiency. The foremost objective is to detect specific objects from the real-time input taken from a webcam and generate bounding boxes with object identified labels. For the computation of our model, firstly we imported a program instance from Darkflow. After that we have import another library called numpy. *Numpy* is a basic package for python. Lastly we have import library called *time* because we want to calculate time, how faster processing the system. The instance was specified with model, weights load and a threshold

which is to specify the confidence factor for generating the bounding box window within an object named 'Options'. Initially we intended to process a previously saved video input with a threshold value of 0.3 without training the system. Here we are setting a lower value of threshold to get more bounding boxes captured from the input video and as we are using CPU, we are processing 6-7 frames per second time. For the input captured from the camera, a specific value of width and height of the captured data is declared in the system parameters. Then we have passed this information as parameter through *TFNet* in darkflow by creating *tfnet*. This is going to initialize our model that going to make prediction. The processing result is structured to contain the co-ordinates of the frame bounded in the input video images along with the confidence value and label name of the framed object.

For object detection YOLO create boxes and here for creating different color boxes we are using this following line. Through this line we will get array of random numbers which is 3 elements long and output is also given below how it looks.

```
colors = [tuple(255 * np.random.rand(3)) for _ in range(10)]
```

```
color [(147.5233869049188, 183.08595967405034, 8.537941416778597), (126.21604066802168, 38.35153317857361, 176.01727881047074), (161.36559206052283, 192.49141474171395, 123.79262868116345), (1.432111315829581, 0.7503065872042547, 75.80912617815828), (242.48001053032925, 91.15101106272154, 208.09431817872692), (98.10005652218699, 131.63718855006837, 22.353959371555217), (151.38893157141396, 217.83605456613924, 33.09792441837568), (126.31925517704852, 55.36876338925023, 176.87280423666044), (97.42129238849935, 11.837942492327647, 196.0875437580878), (193.36644257536668, 254.68377567349518, 252.52566861130822)]
```

Fig 3.4: Bounding box color declaration

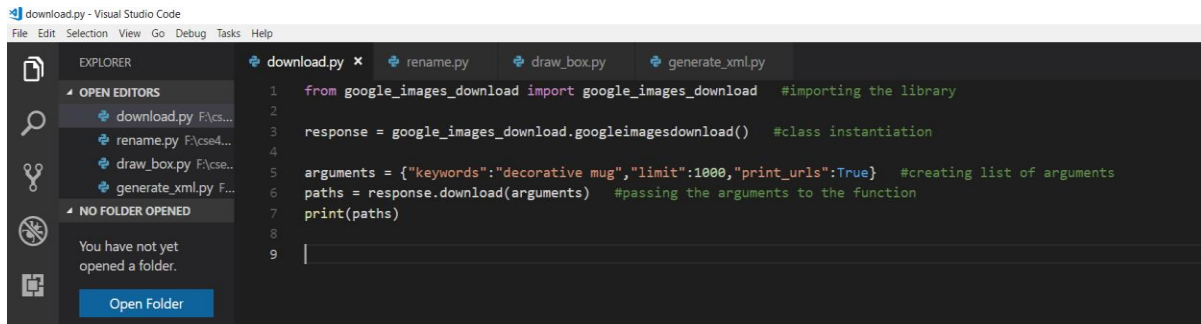
In our thesis, we have used *cv2.VideoCapture()* to connect our webcam for real time input with the support of *OpenCV*. In this function, we set '0' as parameter as we have connected with only one webcam. Moreover we have set our frame size through these two given line where width is 1920 and height is 1080.

```
capture = cv2.VideoCapture(0)
```

```
capture.set(cv2.CAP_PROP_FRAME_WIDTH, 1920)
```

```
capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)
```

**3.1.3 Custom-made Dataset Construction** For our system, we intended to develop our own personalized dataset created for the elected desired objects for which we collected the images from Google images. Python has a built-in library dedicated to generate such databases. After importing the library, we generated instances for each subject with argument of keyword and limitation number provided as parameters.



```

1 from google_images_download import google_images_download #importing the library
2
3 response = google_images_download.googleimagesdownload() #class instantiation
4
5 arguments = {"keywords":"decorative mug","limit":1000,"print_urls":True} #creating list of arguments
6 paths = response.download(arguments) #passing the arguments to the function
7 print(paths)
8
9

```

Fig.3.5: Code Simulation Terminal for Customize Dataset Creation

After creating our customized dataset we created our annotation files. In our annotation files we have created xml file of every pictures. Here we import few more libraries like *matplotlib*, *write\_xml*. For floating we import *matplotlib* and *RectangleSelector* libraries. Rectangle Selector basically allow us to draw the drag able rectangle on images and how we get the coordinates of it. Moreover we import *write\_xml* library for generating xml file of every images. For creating xml file we have specified image folder directory, save directory where our annotated file will be saved and object name.

First we display our image using *ax.imshow( )* function. Then we created a rectangular drag able selector tor select the object we want to detect. Then we created a call back, so after selecting the required object from top left to bottom right by drag able rectangle bounding box if we hit the “q” botton then a new window will popping up. By selecting object with this drag able bounding box we will get 3 top left coordinates and 3 bottom right coordinates. Here we specified name of the object so every time we draw box it put that object name in the list.

For getting xml file we import an xml library. Moreover we import *cElementTree* library and this is our main xml generating library that we gone use to generate our xml file. Also we have created a function passing some parameters like folder, image, top left, bottom right, save



directory and through this function and following libraries our xml file has created. In this xml file we keep record path of the image, height, width, depth of the images.

**3.1.4 Machine Training** For identifying the object tfnet plays a vital role. tfnet is mainly where yolo does all the training process. There are different library to obtain this process. Here *.import help* imports a file in a special XML format produced by exporting pages. *.import flow* is used here to export and import flow across environment with packaging. *import Header,Line* helps to create data in column and rows. *Import create\_framework* is used for manipulating dataset. Import Darknet is the portion which does the neural networking in C language. After using all these libraries the TfNet class works with different methods and parameters. *\_TRAINER* has all the optimizer in a directory. All there optimizer are used to optimize the performance of the Network. *\_get\_fps* method will get the Frame Rate which is the frequency of the video. The greater the FPS the smoother the video motion will be. *\_help.say ,this* is the program that reads python programs and carries out their instruction. *Help.camera* uses to capture images from live stream trough mobile camera that we are using for real time. *Flow.predict* is a method which is used for machine learning in python to make prediction. *Low.return\_predict* returns all these pretiction. FLAG is very important for new input. extension can use the flags to indicate and test when a given type structure contains a new feature. *.ntrain* variable is use to maintain the length of darknet layer. Graph is the core concept of tensorflow to present computation.

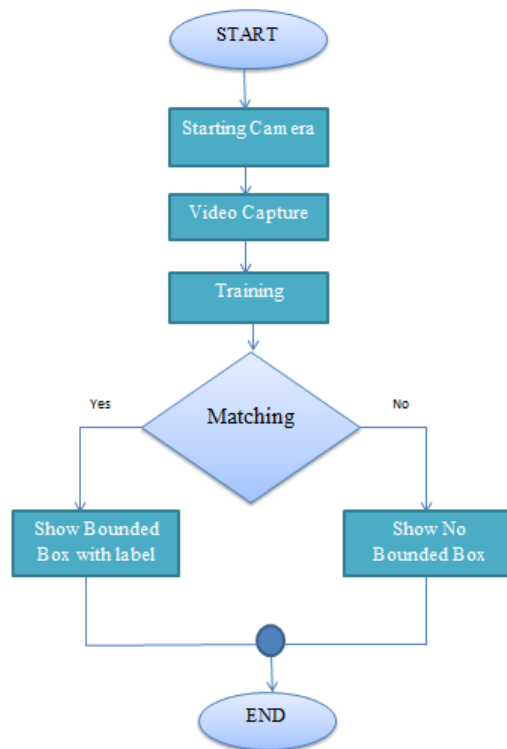


Fig. 3.6: Flowchart of the basic identification workflow

They use a tool called *protobuf* which can generate specific language stubs, that's where the *GraphDef* come from. It's a serialized version of Graph. One should read \*.pb file using *GraphDef* and bind the *GraphDef* to a (default) Graph, then use a session to run the Graph for computation. `State=identity(self.inp)` will take the input with *tf.placeholder* and after traversing darknet layers by a for loop it will get specific output. In this there are methods for GPU users but for limitation we had to use only CPU so for us only CPU methods did all the work for training and identification. Also if there is no *tfnet* then through these methods a machine can be trained from the beginning and a *tfnet* folder can be created with *os.makedirs*.

For training our own dataset we have changed the number of classes and filter number of our .cfg file. Then we have also changed our label.txt file where we put all the labels of our classes. After that we start training our model through this given command line in Fig.3.7.

```

Anaconda Prompt
(base) C:\Users\Ritun's PC>activate tensorflow
(tensorflow) C:\Users\Ritun's PC>python flow --model cfg/tiny-yolo-voc-3c.cfg --load bin/tiny-yolo-voc.weights --train --annotation Annotations/annotations --dataset Images/images

```

Fig.3.7: Command for training activation in command prompt

And this is how our model is training shown in Fig. 3.8.

```

C:\Windows\System32\cmd.exe - python flow --model cfg/tiny-yolo-voc-1c.cfg --load bin/tiny-yolo-voc.weights --train --annotatio...
step 2688 - loss 1.0671327114105225 - moving ave loss 1.1698668096117708
step 2689 - loss 1.1957106590270996 - moving ave loss 1.1724511945533038
step 2690 - loss 1.2485380172729492 - moving ave loss 1.1800598768252684
step 2691 - loss 1.165402889251700 - moving ave loss 1.1785941780679126
step 2692 - loss 1.1907505989074707 - moving ave loss 1.1798098201518683
step 2693 - loss 1.5314595699310303 - moving ave loss 1.2149747951297847
libpng warning: iCCP: known incorrect sRGB profile
step 2694 - loss 0.9035111665725708 - moving ave loss 1.1838284322740633
step 2695 - loss 1.4351309537887573 - moving ave loss 1.2089586844255327
step 2696 - loss 1.17952561378479 - moving ave loss 1.2060153773614586
step 2697 - loss 1.0945141315460205 - moving ave loss 1.1948652527799148
step 2698 - loss 1.1125577688217163 - moving ave loss 1.186634504384095
step 2699 - loss 1.1617929935455322 - moving ave loss 1.1841503533002387
step 2700 - loss 1.3689643144607544 - moving ave loss 1.2026317494162904
Finish 100 epoch(es)

```

Fig.3.8: training compilation

For our output we called `capture.read()` function which gives Boolean values. When the camera is on it will give true value and frame it. This frame is sent to TFNet as parameter through `tfnet.return_predict()` function. In this way we will get result of identifying objects. Then for each object we set different boxes of different colors. For each bounding boxes we are taking values of x and y coordinates by these two commands:-

```

tl = (result['topleft']['x'], result['topleft']['y'])
br = (result['bottomright']['x'], result['bottomright']['y'])

```

Then we set label and confidence values on the bounded box. After that for showing the label we set the text format, size and color of text. Finally we draw the rectangle bounding box based on frame, tl, br, color and put text on bounding box as label.

**3.1.4 Accuracy Analysis and Development** Initially after the model establishment, we run the model with the mobile camera connected to the computer via an application; we passed the captured data to the processing unit and successfully identified the object in real-time which is

shown in Fig.3.9. The accuracy rate is lower as the machine system is not trained to its full capacity.

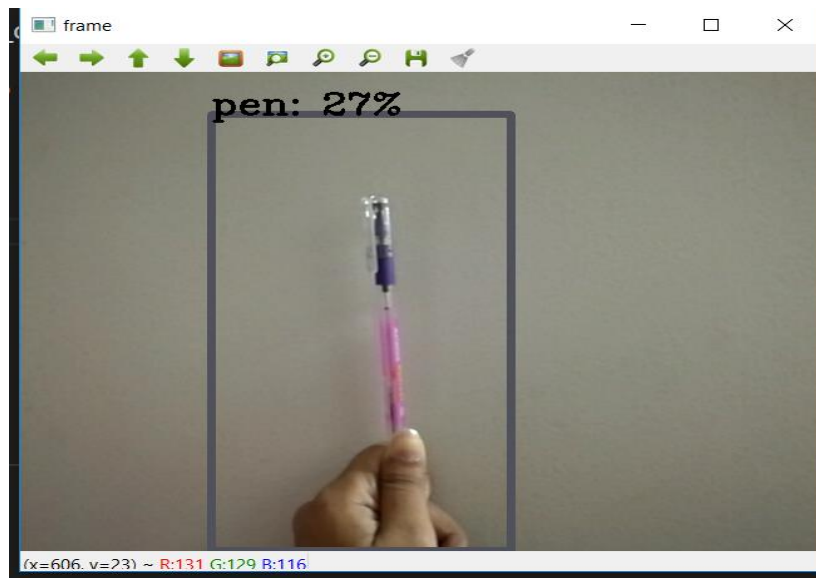


Fig 3.9: Accuracy at the initial stage of compilation of identification

### 3.2 Face Detection and Identification Methodology

Based on the analysis of previous research works and implementation of system based on dissimilar algorithms, for our robust face identification, we have focused on implementing MTCNN. The MTCNN algorithm basically mechanisms in three steps and uses one neural network for each. The first part is usually called a proposal network. It will predict potential face positions and their bounding boxes quickly through a shallow CNN. The outcome of this stage is a large number of face detections and lots of false detections. The second part uses images and outputs of the first prediction and improves the result to eliminate most of false detections and aggregate bounding boxes through a more complex CNN. The last part perfects even more the predictions and adds facial landmarks predictions. In fig 3.10 Basic Convolutional Architecture for MTCNN Algorithm is shown.

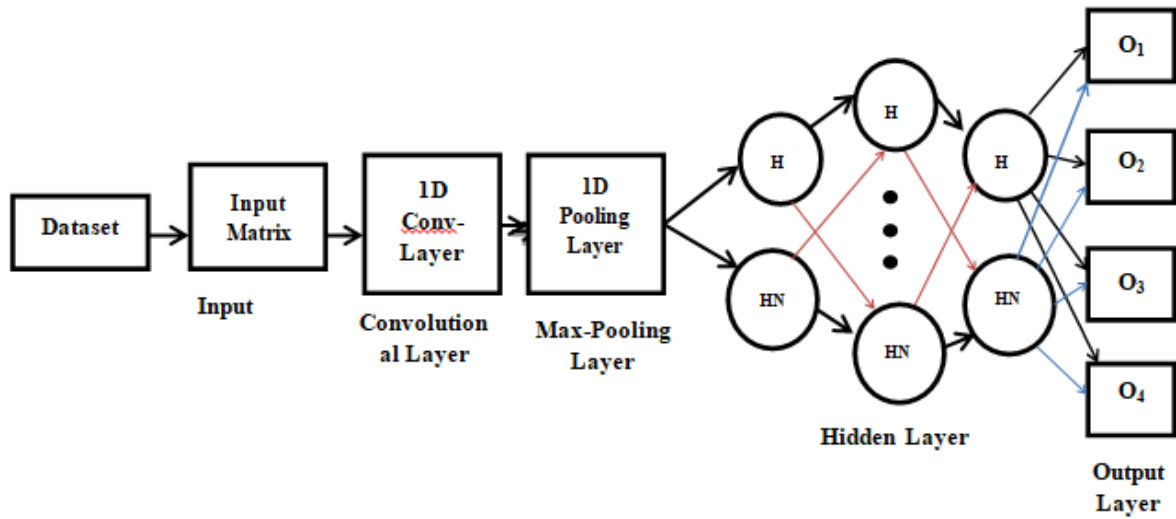


Fig 3.10: Basic Convolutional Architecture for MTCNN Algorithm

**3.2.1 Installation Setup** For our system development, we have chosen python language as it has more compatibility and accessibility with our next phase of hardware installation of Raspberry Pi. For the script editing and compilation, we used *Visual Studio Code* which has faster processing and compilation efficiency compared to other available open source editors such as Spyder, Jupyter etc. We also installed *OpenCV* library to which has been essential for our image capturing and processing activities. *Tensorflow*, an open source deep learning framework, and *Numpy*, an essential library for python programming language for computing arrays and matrices; are two basic libraries that has been installed for the proper undertaking of our thesis. Also, function named math has been imported for computation purposes necessary in different parameter calculation.

### 3.2.2 Customized Dataset Development

**3.2.2.1 Argument Parsing** For user-to-user customization, we have designed the system to develop with its own dataset collected from the user interface. Using *argparse*, a built in module which is an optparse inspired command-line parsing library, produces highly informative usage messages, also handles both optional and positional arguments. In our thesis, this module is used with a parameter of mode that controls the decision of when the camera will take new input data

to store or only identify. Then with `.add_argument` particular portion is added with parse input and store in a variable.

**3.2.2.2 Dataflow Graph** As per our system code advances, in the `main.py` code block, the next stage will go to `tf_graph`. TensorFlow uses a dataflow graph to represent the computation in terms of the dependencies between individual operations. This leads to a low-level programming model in which you first define the dataflow graph, then create a TensorFlow session to run parts of the graph across a set of local and remote devices. *Dataflow* is a common programming model for parallel computing. In a dataflow graph, the nodes represent units of computation, and the edges represent the data consumed or produced by a computation.

**3.2.2.3 Face Alignment** The third step we developed is the alignment of the face, for that a function in the code `align_custom` will be working. For the alignment, we implemented *Dlib* face alignment strategy. The distinctive fact in this method or approach is that it doesn't deform the original image like *Dlib* usually does. Here the whole face portion gets positioned in three parts, Right, Left, and Center. As a new input list, a loop runs to make 2 dimensional matrixes and the created matrix is transposed. The shapes gets computed by *Mean* and *Coefficient of Variable(cov)* . Then *Affine transformation* is applied on the matrix which basically refers to combination of linear transformations and translations. The Align face is done in *BGR format*. BGR format is the convention for OpenCV. So the term BGR (and BGRA) is a leaky abstraction where the graphics library is explaining how the integer is logically ordered. This makesour code more readable as that is directly accessing the color components individually. Size of image, 3D array which has the pixels that detects face, another 3D array with align and the left, right and center position of face is stored. Also there are mean parameter stored in (X, Y). Fig. 3.11 shows WorkFlow of Face Detection and Identification.

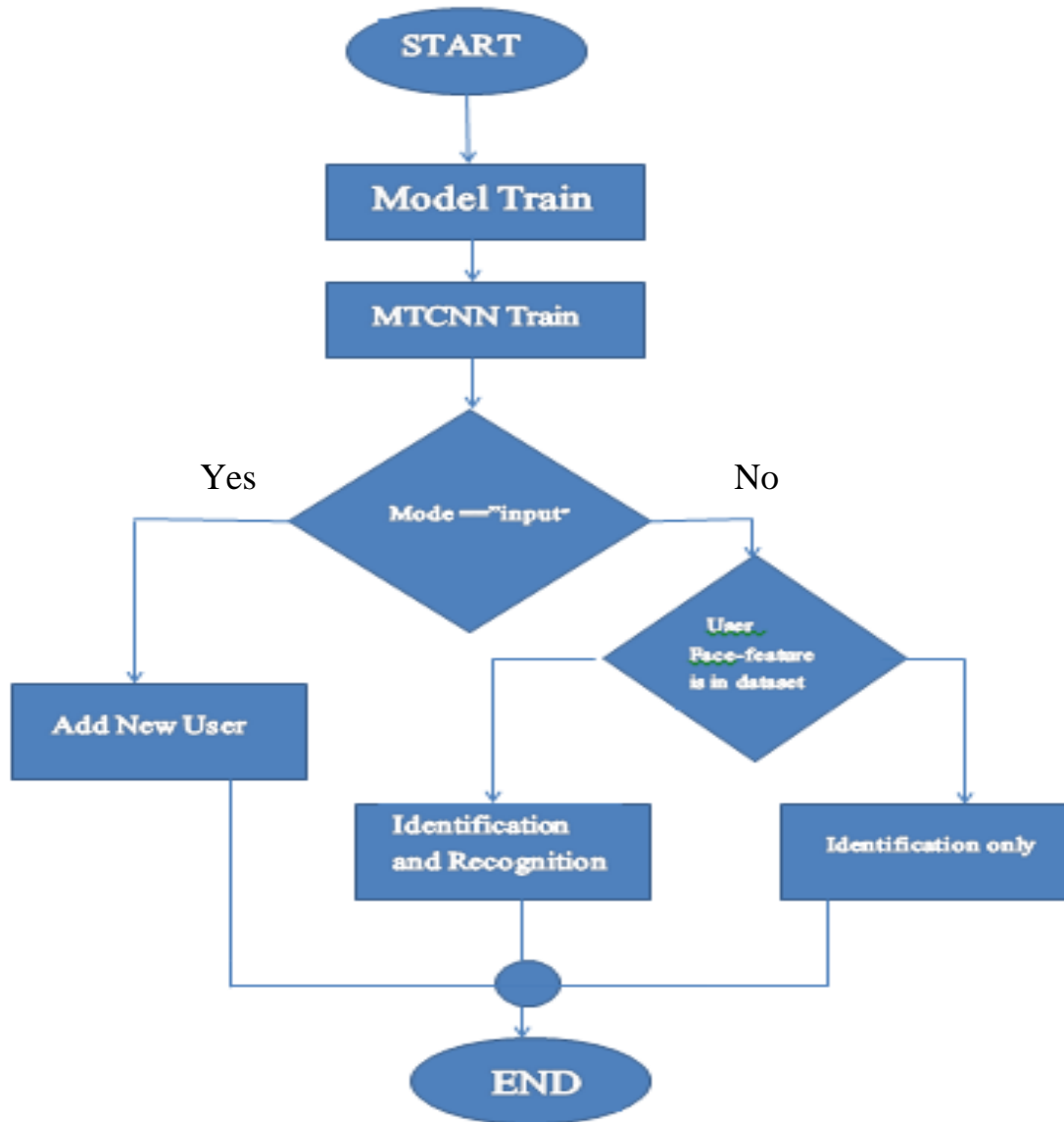


Fig. 3.11: Work Flow of Face Detection and Identification

$mean\_face\_shape\_x = [0.224152, 0.75610125, 0.490127, 0.254149, 0.726104]$

$mean\_face\_shape\_y = [0.2119465, 0.2119465, 0.628106, 0.780233, 0.780233]$

Then padding is added to the points for desired view and all points again are converted in Matrix. The Matrix rotated with `cv2.getRotationMatrix` and then `cv2.warpAffine(img, rot_mat, (desired_size, desired_size))` is returned to main class.

**3.2.2.4 Face-Feature Extraction** As our system task steps advance, the next step in our code will result in `face_feature`. Here we had to import `tensorflow` and `inception_resnet_v1`. We

proposed a deep convolutional neural network architecture codenamed "*Inception*". The main hallmark of this architecture is the improved utilization of the computing resources inside the network. This was accomplished by a wisely constructed strategy that allows for increasing the depth and width of the network while keeping the computational budget constant [28]. Here for tensorflow, *tf.placeholder('float', [None,160,160,3])* is provided which is the default input for *tf.nn.l2\_normalize*. Here we get the data that are already previously trained and stored in model folder. Previously trained images go through the matrix process and returned to *main.py*.

**3.2.2.5 Detection using MTCNN** At this stage, *mtcnn\_detect* function is called by main class. In this file the images uses Tensorflow implementation of the mtcnn face detection algorithm. The images go through image pyramid which helps in smoothing and subsampling images. Here for pyramid representation by default it uses *.709* as a parameter. Model path of the previously trained data also linked in this part. The threshold is of 70%, factor for smoothing is saved *.709* and also *scale\_factor* for scaling images.

### 3.2.3 Loading MTCNN Face Detection Model

The actual processing portion works on three stages: Proposal Network (P-Net), Refine Network (R-Net) and Output Network (O-Net).

**3.2.3.1 Proposal Network (P-Net)** We exploit a fully convolutional network, called Proposal Network (P-Net), to obtain the candidate facial windows and their bounding box regression vectors. Then candidates are calibrated based on the estimated bounding box regression vectors. After that, we employ non-maximum suppression (NMS) to merge highly overlapped candidates.

**3.2.3.2 Refine Network (R-Net)** All candidates are fed to another CNN, called Refine Network (R-Net), which further rejects a large number of false candidates, performs calibration with bounding box regression, and conducts NMS.

**3.2.3.3 Output Network (O-Net)** This stage is similar to the second stage, but in this stage we aim to identify faces regions with more supervision. In particular, the network will output five facial landmarks positions.



```

349     normalize = tf.reduce_sum(target_exp, axis, keep_dims=True)
350     softmax = tf.div(target_exp, normalize, name)
351     return softmax
352
353 class PNet(Network):
354     def setup(self):
355         (self.feed('data') # pylint: disable=no-value-for-parameter, no-member
356          .conv(3, 3, 10, 1, 1, padding='VALID', relu=False, name='conv1')
357          .prelu(name='PRELU1')
358          .max_pool(2, 2, 2, name='pool1')
359          .conv(3, 3, 16, 1, 1, padding='VALID', relu=False, name='conv2')
360          .prelu(name='PRELU2')
361          .conv(3, 3, 32, 1, 1, padding='VALID', relu=False, name='conv3')
362          .prelu(name='PRELU3')
363          .conv(1, 1, 2, 1, 1, relu=False, name='conv4-1')
364          .softmax(3, name='probi'))
365
366         (self.feed('PRELU3') # pylint: disable=no-value-for-parameter
367          .conv(1, 1, 4, 1, 1, relu=False, name='conv4-2'))
368
369 class RNet(Network):
370     def setup(self):
371         (self.feed('data') # pylint: disable=no-value-for-parameter, no-member
372          .conv(3, 3, 20, 1, 1, padding='VALID', relu=False, name='conv1')

```

Fig 3.12: Different Convolutional Layers declared in the mtcnn\_detect class

In Fig 3.12 the layers of mtcnn\_detect class codes are shown. In MTCNN Network, there is a class for constructing the network; which has input nodes for the network, current list of terminal nodes, mapping from layer names to layers. If nodes are true, then the resulting variables are set as trainable data. The network also loads network weights, serializes weights in *numpy* by using the current tensorflow\_Session. These parameters are then used to execute graphs or part of graphs. It also allocates resources for results and holds the actual values of intermediate results and variables. Then four methods are used before putting images into layers to train and to create bounded box shown in figure 3.13.

```

def get_output(self):
    '''Returns the current network output.'''
    return self.terminals[-1]

def get_unique_name(self, prefix):
    '''Returns an index-suffixed unique name for the given prefix.
    This is used for auto-generating layer names based on the type-
    prefix.
    '''
    ident = sum(t.startswith(prefix) for t, _ in self.layers.items()) + 1
    return '%s_%d' % (prefix, ident)

def make_var(self, name, shape):
    '''Creates a new TensorFlow variable.'''
    return tf.get_variable(name, shape, trainable=self.trainable)

def validate_padding(self, padding):
    '''Verifies that the padding is one of the supported ones.'''
    assert padding in ('SAME', 'VALID')

```

Fig 3.13: Code portion to transfer images to convolutional layers

After each convolutional layer, it is convention to apply a nonlinear layer (or *activation layer*) immediately afterward. Fig 3.8 shows the code portion of transferring imaged into convolutional layers. The purpose of this layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the conv layers (just element wise multiplications and summations). In the past, nonlinear functions like tanh and sigmoid were used, but researchers found out that *ReLU layers* work far better because the network is able to train a lot faster (because of the computational efficiency) without making a significant difference to the accuracy [29]. Another method called *max pooling* uses the maximum value from each of a cluster of neurons at the prior layer. This serves two main purposes- first is that the amount of parameters or weights is reduced by 75%, thus lessening the computation cost and second is that it will control *overfitting*. This term refers to when a model is so tuned to the training examples that it is not able to generalize well for the validation and test sets. [29]

### 3.2.4 Loading New Face Data

After completing the above steps, the command portion from part 3.2.2.1 then decides if the system will take new input or will just recognize who is in front of the camera. In this section we are going to explain the process how our system registers new person in its dataset. A function called *create\_manual\_data()* is used to create new dataset. With the help of *cv2. videocapture* camera will open and wait till it gets the input name of the person. After getting the input it goes to the *facerec\_128D.txt* and with the support of *json.load* parameters of a person, face image is stored. While storing the features of face, it maintains 3 positions which are left, right and center. The frame we used is for capturing the images of minimum 80x80. But if data of a person already exists in the dataset then by *json.dumps()*, the previous features will be removed and new features will be added. When the user is done registering their data, they will press “q” to close the window. Fig 3.14 shows the terminal for taking new data input.

```

PROBLEMS 35 OUTPUT DEBUG CONSOLE TERMINAL 1: powershell
keep_dims is deprecated, use keepdims instead
WARNING:tensorflow:From F:\thesis_face\FaceRec-master\mtcnn_detect.py:349: calling reduce_sum (from tensorflow
w.python.ops.math_ops) with keep_dims is deprecated and will be removed in a future version.
Instructions for updating:
keep_dims is deprecated, use keepdims instead
MTCNN Model loaded
Ritun
Please start turning slowly. Press 'q' to save and add this new user to the dataset
PS F:\thesis_face\FaceRec-master>

```

Fig. 3.14: Terminal for taking new face data input

In the command that was processed in part 3.2.2.1, if there is no “input” portion it will go to *camera\_recog()* to detect face and will try to identify if the dataset has the person’s data. If not then it will show “unknown”. To recognize, the window will open with cv2 and will show “camera sensor warming up”. The Fig. 3.15 shows the dataset structure from input.

```

FaceRec_128D - Notepad
File Edit Format View Help
{"name": "left", "left": [-0.16686329519748688, -0.10320915281772614, -0.013180938871502876, -0.036437664180994034, 0.013527474366128445, 0.0776028150606155, 0.01895785311
-0.15640908578103605, -0.00810727715482249, -0.07095236611031851, 0.04643670890993464, -0.1091607348854085, 0.0606921713247892, -0.02130037246608734, 0.051610003554
0.2675546377897263, 0.1187082156538633, 0.0509228220721245, -0.02173381125159264, 0.1428208202123642, -0.06130366678480958, -0.108077097569348, -0.047661372009992
7549533844, -0.015083459205952545, 0.003567591309547423, 0.03122933952069283, 0.06645719707012177, 0.06371396780014038, -0.01198652366653442, 0.0157171766495186, -
0.180435, -0.0062881174159431, -0.040347885340451394, -0.12164516746987813, -0.007993373088536647, -0.0024655845795128107, -0.010807787999510765, 0.1275609246393585, -
586111069, -0.0334807041334007, -0.0377736732363701, 0.002538098720833659, 0.03560018911957741, -0.0503515266523361, 0.0964082658624649, 0.05205131322145462, 0.01001
956877804, -0.02668854646372795, 0.074562327618599, 0.023172389715989958, 0.06781580995766, 0.02320213429628027, -0.01630521103140786, -0.0078566755175728, 0.4
918187379837, -0.0237428846347332, -0.003752277698307834, -0.008871915292762982, 0.026296313852071702, 0.008759819902479649, -0.03188956156373024, 0.0096444094367981,
44748920282552, -0.03778115621328354, 0.055402208119630814, 0.09459313085208969, 0.12401261180639267, -0.06901971250772476, 0.09118223190307617, 0.07480910420417786, 0
09520227462053239, 0.0943509758234024, 0.045380913966123409, -0.00606017597913742, 0.030973130837082863, 0.00176230800490379, -0.05032404978437424, 0.01193058031184812
6134, 0.02114296518266201, 0.09908837825059891, -0.026019470766186714, -0.05145781487226486, -0.05944950133562088, -0.1280232071876526, 0.08323434564628873, -0.01317237
0744851166009903, -0.21493422985876904, 0.05124680697917938, 0.0211218042820692062, 0.0019592847675808068, 0.014948562952526028, -0.05548720061799022, -0.0250135366616
0624079293012619, 0.06351806360971451, -0.120381322389212, 0.1303161416053772, 0.055483085768661, 0.032323841340876, 0.1440188060130655, 0.15830475091342004, 0.4
005213826895, -0.05089680372476578, 0.035716358572244644, -0.16752204208973083, 0.02600243878942146, -0.11988905072212219, 0.1300787627696991, 0.0210567553809105, 0.4
6918, -0.0745445266366005, 0.05625130236148834, 0.1002267524600829, -0.0097106434448485, -0.0363575412610135, -0.1562446028045438, 0.06572482929791832, -0.08634992250001
24199140323, -0.0175647632995703, 0.0028873181207376, 0.006728430280802, 0.14940211109757843, -0.0541676309087906, 0.1643236364058177, -0.05047175788879395, -
6, -0.0359862656566906, 0.01686964728393364, -0.017906932160258293, 0.12804517149925232, 0.13424254953861237, -0.04884018376469612, -0.1154780025060823, 0.03253474831
6, 0.0539372038609725, 0.0404483191609087, 0.0904154080978236, -0.012013160203635693, 0.05029920835353095, 0.153452545044342, 0.11370468090704, 0.0647057335370735
.0981872964897955, -0.13487552106380463, -0.0016337479464709759, -0.093157980518341, -0.1234444891929626, -0.015380138531327248, 0.033700428061102875, -0.01704426296
4046, 0.00356193453073302, 0.0814020112156808, 0.111636941151619, 0.04792601051590506, 0.0993862897157669, -0.12668853998184204, 0.0561127561736107, 0.083909340200208
94853997285, -0.1390816715849744, 0.006171151818427, 0.009730305604147, -0.0044132301394367, -0.035319164077045, 0.0757460400452702, 0.0473695446299174, 0.4
466062, 0.002628425453051758, -0.037240158041217804, -0.09763730317354202, 0.040785662282975946, -0.006681723794108629, 0.007074790905357342, -0.146950915717026, 0.4
1732, -0.04179881140859714, 0.08769388496875763, -0.0072355906248093, -0.03352916987501984, -0.014007052406668603, -0.020382385700941086, -0.0626685250080080, 0.066
3108183, -0.00538062674536419, 0.010032976077527142, -0.1251337862739563, 0.09649081061616351, 0.07866211716414309, 0.006081599112600088, -0.009112386632380957, 0.4
48875618, -0.1167251542105789, -0.0321713425219059, 0.013426645658910275, 0.004447146784514189, 0.03725072929668427, -0.051675956696271896, -0.005040257453022, -0.02
7, -0.1072805210950994, 0.00439727433919987, -0.07206204682350159, -0.023071750952738855, -0.0392064579997063, 0.0400923263125722, -0.01769737039211464, 0.01527821
791, 0.0775969494915089, -0.0708051516151515, -0.0098850054989247, -0.02621073719501495, -0.008012111616134644, -0.03166342514145241, -0.04324100801256935, -0.03938
04119082892, -0.06566701829433441, -0.025546135380864143, 0.0087819454073906, 0.014582104080945587, 0.0064526703933743, -0.051588188856404134, -0.0617689549621582, 0.4
0704, -0.00544708412459612, 0.003568308660769522, 0.0260607096014023, 0.14816464080809095, 0.025145553052425385, 0.004721938834915, -0.021204443640457527, 0.0651203
5274, 0.03534514829516411, -0.0596593469831324, 0.008815217763185001, -0.05010474845767021, 0.09146716445684433, 0.09374775502349854, 0.1255348028221346, 0.00434400040
011022564955055714, 0.1485283523797989, 0.06199253723025122, -0.0086732643391017, 0.03538096323609352, 0.0417529195471039, 0.123590484261512761, "center", "0.00816
01171094, -0.044008438700259, -0.10771045090051086, -0.010808546191042092, 0.00184660971852982, 0.14339658212661743, -0.0421508109509496, -0.0054959033300479, 0.4
31531309877, -0.05488807821097374, -0.05189176648852094, 0.012194390406694744, -0.001726107766851783, 0.0614312067279068, 0.06031585484743118, 0.024359216913500804, 0.4
3722202456398, 0.0936586484130112, 0.11052018596363008, -0.0470398154620754, 0.1007347074008432, -0.0945640015670406, -0.0167777095762398, -0.01817611232999405, 0.4
73110145, 0.05409710106496704, 0.229645164001158, 0.149592958861657, 0.07380653466091849, 0.02044263517907143, -0.01730291453120131, 0.06023026733946335, 0.022
595602989, 0.00780186902122498, 0.1253513246746735, -0.0670846626162529, -0.0350267179103218, -0.0642649382352829, 0.000020538052589, -0.002305, -0.002305
0.06395736167430878, -0.03793196242783544, -0.1054902218341827, 0.055640786546468735, -0.00998726487159729, 0.015904095633029938, -0.0528512980413437, 0.0078669175

```

Fig. 3.15: Dataset from our input

### 3.2.5 Compilation and Result Display

In this portion of the code previously explained codes are being compiled. So if the *face\_features* matches with any dataset it will show the user name and with accuracy or distance between points saved in dataset and shown in camera in real time. This function basically does a simple linear search for the 128D vector with the min distance to the 128D vector of the face on screen. Here Fig 3.16 shows the code compilation result and Fig 3.17 shows code of drawing bounding boxes.

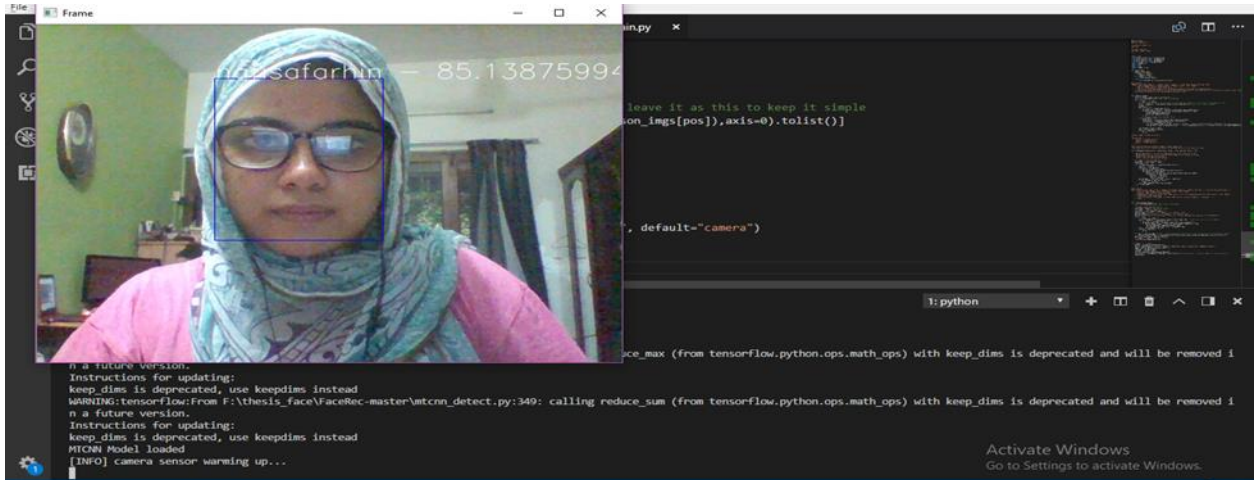


Fig. 3.16: Code compilation and Face identification with Name Label and Accuracy Rate

```

cv2.rectangle(frame,(rect[0],rect[1]),(rect[0]
+rect[2],rect[1]+rect[3]),(255,0,0))

cv2.putText(frame,recog_data[i][0]+" -
"+str(recog_data[i][1])+"%",(rect[0],rect[1]),cv2.FONT_HERSHEY_SIMPLEX,1,(255,255
,255),1,cv2.LINE_AA)
  
```

Fig. 3.17: Code portion to draw bounded boxes

The first line creates the bounding box and the second shows the name with accuracy on the box. This is the whole process we maintained for Face detection and Identification with MTCNN algorithm and using Deep Learning Network.

## CHAPTER 4

### Hardware Setup

#### 4.1 Structure Establishment Process

The central processing unit of our proposed system setup is Raspberry Pi 3 Model B. It is a single board operating system unit with built-in wireless LAN and Bluetooth connectivity. It is a vastly used efficient device for the implementation of our proposed method. For the input data collection of real-time image capturing, Raspberry Pi Camera Module V2 has been used. This module is resourceful for capturing high definition instantaneous video as well as still photographs. Raspberry Pi is powered using a portable battery or power bank. For our thesis, the output result has to be an audio output as the target users are visually impaired. So for user convenience we have used an earphone. Our thesis has two major individual identifications from which we are implementing hardware for the face recognition first. The structure establishment follows the below Fig.4.1 basic step diagram:

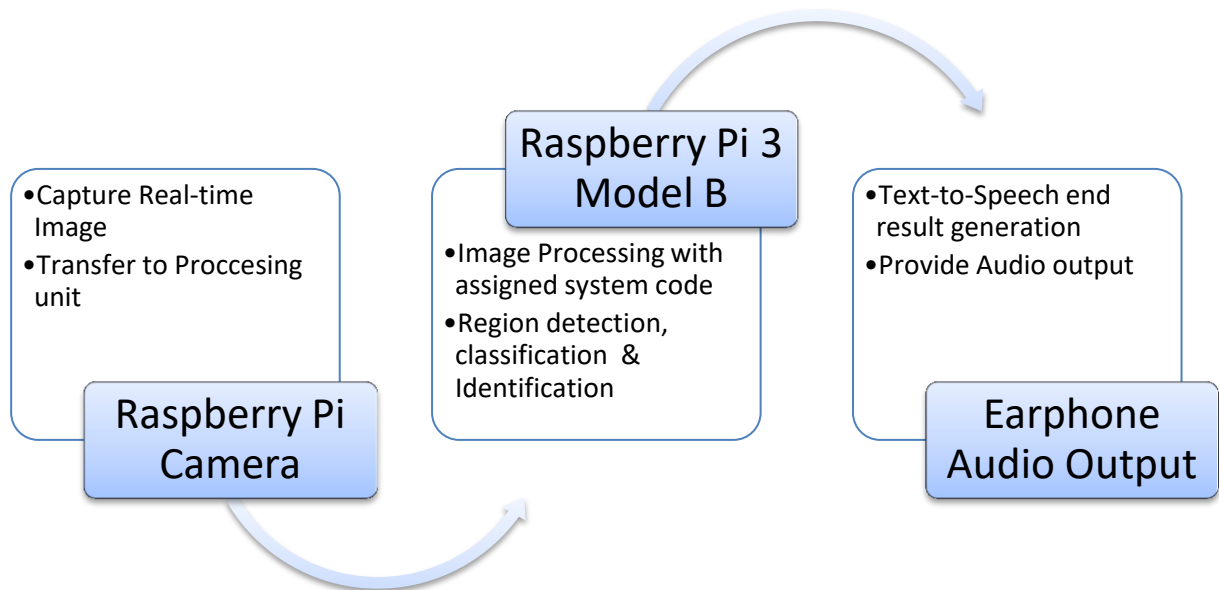


Fig.4.1: Basic Block Diagram for the hardware setup

### 4.1.1 Raspberry Pi 3 Model B Installations

The Raspberry Pi 3 has a required operating system called Raspbian which is a version of Linux specially developed for the Raspberry Pi. We have installed the Raspbian from NOOBS (New Out Of Box Software) using external SD card as it has Micro SD port for loading your operating system and storing data. The Raspberry Pi 3 Model B is the earliest model of the third-generation Raspberry Pi. It has Quad Core 1.2GHz Broadcom BCM2837 64bit CPU with 1GB RAM, BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board. The CSI camera port has given us liberty for connecting a Raspberry Pi camera without any difficulty. The upgraded switched Micro USB power source up to 2.5A is much efficient and user approachable [14].As Raspbian is compatible with programming languages like Python, C, C++etc., we have combined our identification code file within the operating system easily. Fig.4.2 shows the setup of our thesis implementation.

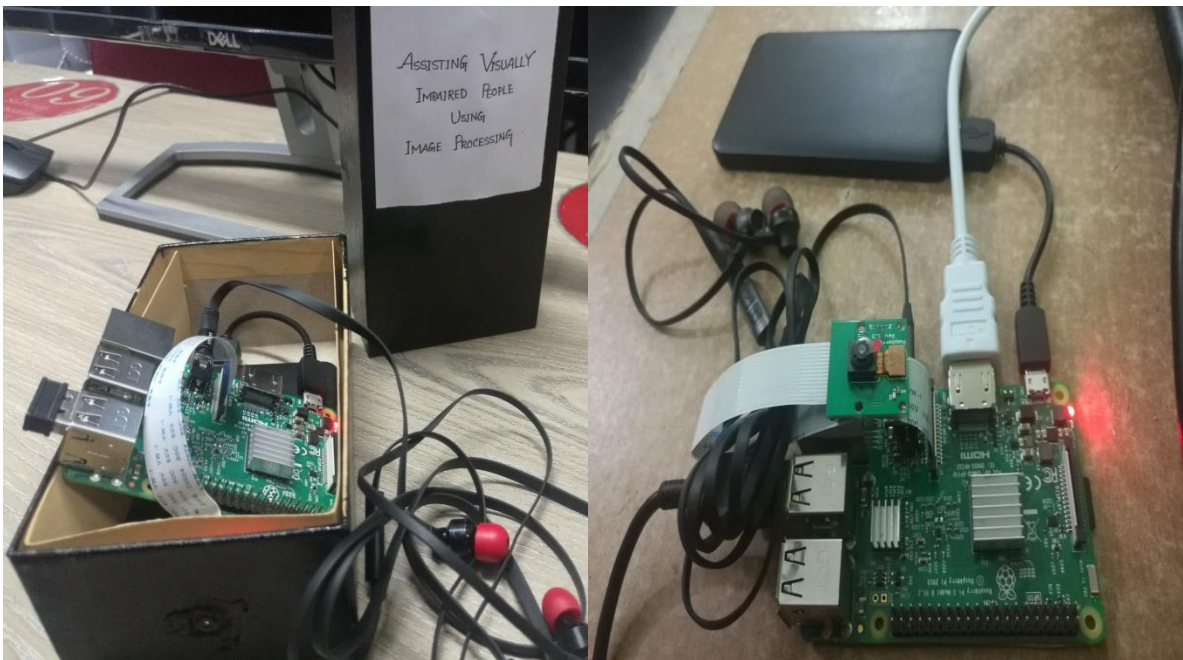


Fig.4.2: Elements with basic connection wiring

### 4.1.2 Raspberry Pi Camera Model v2 and Earphone Connection

The Raspberry Pi v2 Camera Module has a Sony IMX219 8-megapixel sensor [15] which has been accommodating in our thesis for the input video. We connected the Pi camera in the specified slot of the Raspberry Pi. After the installation of the operating system, we have installed the camera module from the system terminal. Initially the module is rebooted with these below commands:-

```
pi@raspberrypi ~ $ sudo apt-get update
```

```
pi@raspberrypi ~ $ sudo apt-get upgrade
```

Then the “*sudo raspi-config*” command in the prompt opens the menu from where we enabled the Pi camera. For the setup of the camera input we have used “*raspivid*” which is dedicated command line module for video recording via Pi camera. Afterwards, for the audio output we have connected a headphone, which is a general audio device compatible with most of the audio output, to the audio output jack of the Raspberry Pi. In the terminal, we installed a library called *espeak* for the audio output.

### 4.1.3 Input and Output Processing Methodologies and Libraries

For our thesis, capturing real-time video and after the processing deliver the identified result in an audio form by text-to-speech is an essential task. Firstly we identified and test-run the functions and libraries that are eligible for this task as well as compatible with our processing code with well-suited versions of programming parameters.

For the video capture installation, we imported *picamera* which is a package for interfacing Raspberry Pi camera module and it is used companionable with python 3 versions which we have used in our identification processing system. The *picamera* library contains numerous classes, but the primary one that all users are likely to interact with is *PiCamera* that we have also used. Upon construction, this class initializes the Pi camera. The *camera\_num* parameter defaulting to 0 which means one camera device selects the camera module that the instance will represent. *camera.start\_preview()* is used to initial start the camera viewing. The resolution and frame rate are declared. Using *PiRGBArray* we are collecting the camera input with a proper size specified which is running in a loop for real-time video capturing.

```

File Edit View Run Tools Help
Thonny - /home/pi/Desktop

main.py x
26 -> each cropped face is categorized in 3 types: Center, Left, Right
27 -> Extract 128D vectors( face features)
28 -> Search for matching subjects in the dataset based on the types of face positions.
29 -> The preexisting face 128D vector with the shortest distance to the 128D vector of the
30 (Distance threshold is 0.6, percentage threshold is 70%)
31 ...
32
33 def camera_recog():
34     print("[INFO] camera sensor warming up...")
35
36     camera = PiCamera()
37     camera.resolution = (640, 480)
38     camera.framerate = 32
39     rawCapture = PiRGBArray(camera, size=(640, 480))
40     #vs = camera.start_preview(); #get input from webcam
41     for image in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
42         frame = image.array
43         #I can certainly add a roi here but for the sake of a demo I'll just leave it as simple
44         rects, landmarks = face_detect.detect_face(frame,80);#min face size is set to 80x80
45         aligns = []
46         positions = []
47
48         #...
49
50         #...
51
52         #...
53         else:
54             print("Align face failed") #log
55             if(len(aligns) > 0):
56                 features_arr = extract.feature.get_features(aligns)
57                 recog_data = findPeople(features_arr,positions);
58                 for (i,rect) in enumerate(rects):
59                     cv2.rectangle(frame,(rect[0],rect[1]),(rect[0] + rect[2],rect[1]+rect[3]),(255,0,0))
60                     cv2.putText(frame,recog_data[i][0]+" : "+str(recog_data[i][1])+"%",(rect[0],rect[1]))
61                     name = recog_data[i][0]
62                     os.system("espeak '{}'.format(name)"]
63
64             cv2.imshow("Frame",frame)
65             key = cv2.waitKey(1) & 0xFF
66
67         rawCapture.truncate(0)

```

Fig 4.3: Basic functions required for real-time video input and audio the result output

Fig 4.3 shows the compiled code portions for the basic functions. For the audio output, we have to use a function that can convert text result into speech as our target uses cannot see the label of identified person. A *text-to-speech (TTS)* system converts normal language text into speech. For this type of system, there are multiple ways to implement in python. Pyttsx is a platform for such speech generation and it is widely used. But in our version of python in which we developed our system, it is not compatible. So we have used *espeak*. eSpeak is a compact open source software speech synthesizer for English and other languages. After importing the package, it can compile speech from text. For our system, we firstly run the main identification code set up in the system with proper library and environment installation, then the identified result label is passed via an object variable to espeak for text to speech conversion from which the result is passed into the audio device connected with the raspberry pi.



# CHAPTER 5

## Experimental Result

### 5.1 Object Identification Result

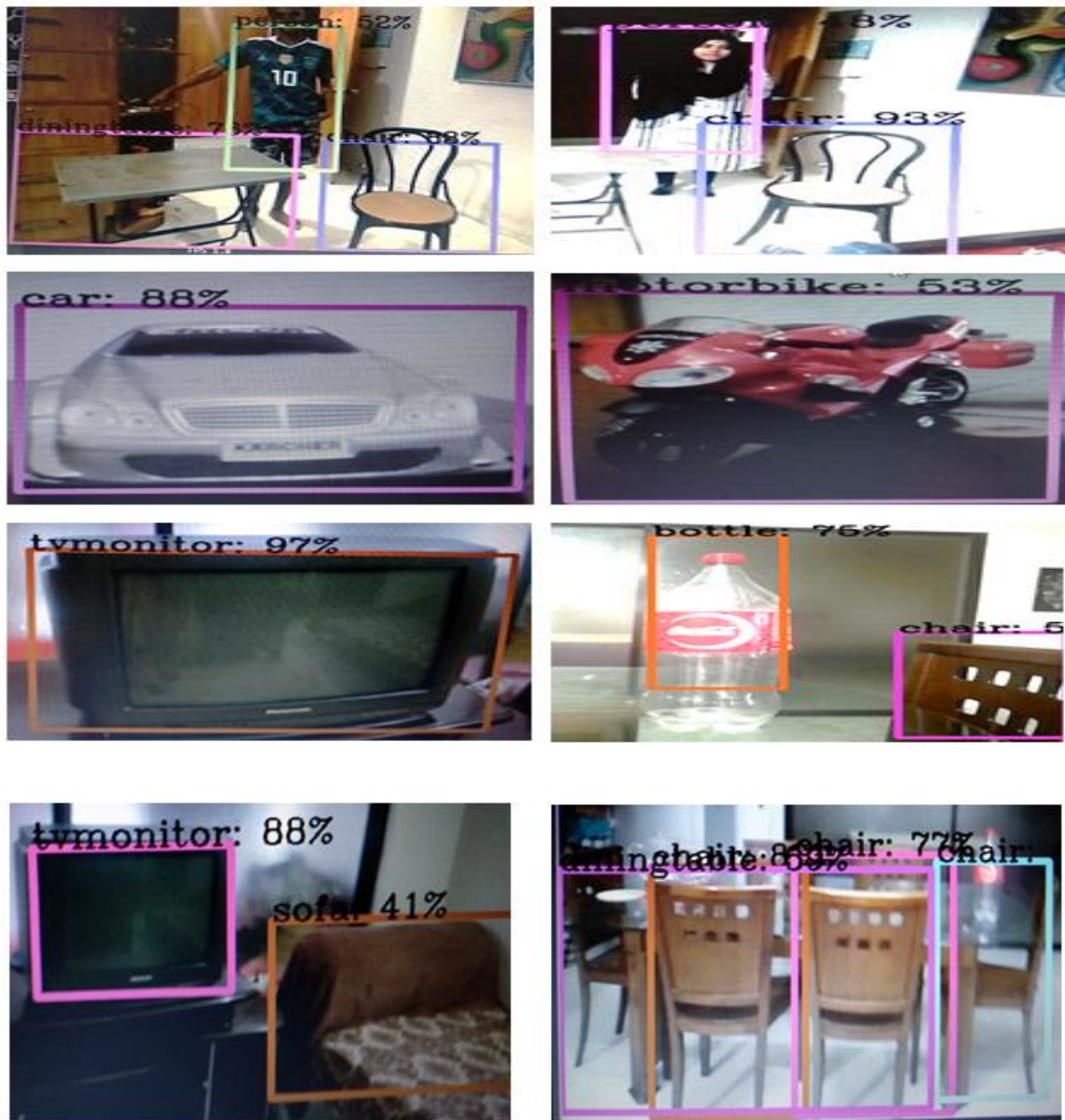


Fig 5.1: Detected object with name label and accuracy rate

Fig 5.1 shows the result of our object detection. Here we have seen the bounded box surrounded a specific object. This specifies the machine has the capability to identify object from its surrounding. The code has its portion which has the capability to match the *confidence factor* with the previously trained dataset. By matching the confidence factor it finds the most appropriate confidence factor from the pre trained model and thus it represents the name shown in the figure. As the object's name is showing correctly so the identification has done properly. Next to the name of the object a percentage number is shown, this number represents the accuracy rate of object identification. This percentage number is subject to maximum matching of the confidence factor in real time video frame with previously trained model confidence factor. Also there are values of Frame per Second (FPS) which shows how fast our Algorithm (YOLO) is able to detect object in real time. The percentage that shows the accuracy of identification of an object is approximately 63-80% accuracy rate and the FPS is little bit low about 8-15 because the machine needs to train more on the dataset it has. For not having a GPU supported machine the system is taking more time to process the whole dataset. To achieve perfection in accuracy and FPS the system is still on training which will help us to detect and identify object really fast from real time video.

## 5.2 Face Identification Result

Fig 5.2 shows the accuracy of face identification. Faces that are in the dataset that was taken as input matches with the faces in front of the camera. As the machine could match one of its dataset values with the bounded box it shows the output with accuracy. The bounded box shows the proper face detection. The percentage is the accuracy that was calculated through code and the name with the percentage was shown according to the input user typed while giving their face data. How well the machine is able to detect and recognize depends on the percentage number that is shown in the figure. As the System uses Deep Learning so more time and data will be needed to provide more accurate result. As the system is still on improvement the dataset is small for now, it will have multiple people's faces in its set of data and will be able to detect and recognize multiple people at a time while the video is on real time.



Fig 5.2: Detected face with name label and accuracy rate

### 5.3 Face Detection Result from Raspberry Pi

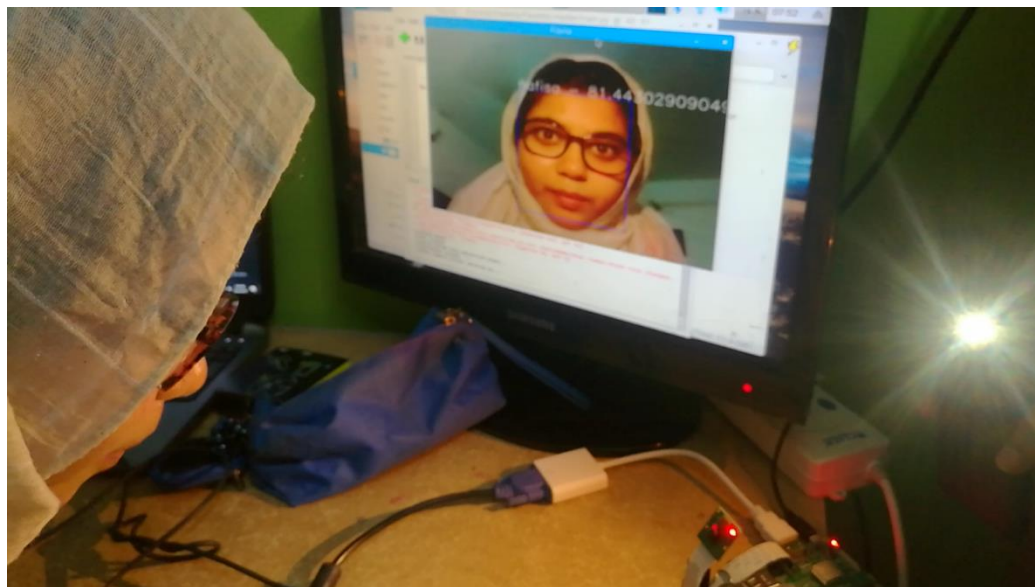


Fig 5.3: Detected face with using Pi camera

In Fig 5.3 there is a bounded box around the face accumulated through using raspberry pi camera. This bounded box came because the face identification code was implemented in the raspberry pi. As we can see the rectangle box detecting the face with proper labeling and

satisfying percentage of accuracy so it means our system is able to detect human faces, which will be the expected output for face recognition process. Also we have used the hardware to let visual impaired people get the audio of names of those who can be recognized through face recognition process. For this we have used text to audio process which we are not able to show it in the paper but from our practical implementation we have witnessed the successful implementation of the audio output of the identified person's name from the database implied in the model.

## **CHAPTER 6**

### **Conclusion**

The proposed thesis proposes of detection and thus identification of major objects as well as face recognition from personal dataset. For the Object and Facial Recognition, YOLO Algorithm and MTCNN Networking are implemented respectively. The software is designed using OpenCV libraries of Python as well as implementing machine learning process. The major processor of our thesis, Raspberry Pi scans and detects the facial edges via Pi camera and objects are recognized via mobile camera. Image recognition results are transferred to the blind users by means of text-to-speech library and an audio device. The object detection process achieved 8-15 FPS processing with an accuracy rate of 63-80%. The face identification process achieved 80-100% accuracy.

#### **6.1 Implementation Challenges**

The first and foremost challenge that we have faced was the generation of modified dataset creation. As we have customized dataset for specific objects and user-to-user different personal face dataset, so the dataset for our system has been custom developed. The annotation process with multiple objects has been a complex process for which the training of machine has been delayed and accuracy result derivation was halted. Also we face difficulties to merge different libraries with the programming language due to the version compatibility conflict. The proper efficient combination of latest algorithm and modern versions of libraries and packages has been a task for us. Last but not the least our merge of hardware has been an encounter as the raspberry Pi had to be properly built up with necessary environment setup and accurate installation of the program code so that the power source connection implies the running mode of the device.

#### **6.2 Future Directions**

In our thesis we have worked on the object detection and face identification independently for maximum efficiency. Also we have set up a hardware implementation system to run our structure in real-time. For our further advancement of this thesis, we would like to merge these two portions side-by-side such that they can run in parallel and human 7 object can both be

identified. Another development that we have in our future direction is to design the hardware structure in more user-friendly and easy approachable model for the blind users as our executed module is a beta version to check the implementation possibility of our software design. It is our desire in implant Movidius VPU on our hardware structure as this model of Intel processing unit is faster and appropriate for real-time video processing as well as AI powered. This will make our model a better performing one and fulfill our target.

We would like to state that, working and implementing a working model to support the visually impaired people by face and object identification. Our thesis work has been a successful attempt to our target and we would like to work in the project in future to develop more user friendly model and precise accuracy gain.

## References

- [1] Global Data on Visual Impairments 2010. Available online: <http://www.who.int/blindness/GLOBALDATAFINALforweb.pdf> (accessed on 23 April 2017).
- [2] M.Goria, G.Cappaglia, A.Tonellia, G.Baud-Bovyb, S.Finocchiettia; “Devices for visually impaired people: High technological devices with low user acceptance and no adaptability for children; Neuroscience and Bio behavioral Reviews”; vol. 69, pp. 79-88, 2016.
- [3] F.Henriques, H.Fernandes, H.Paredes, P.Martinsand, J.Barroso ; “A prototype for a blind navigation system; 2nd World Conference on Information Technology;”
- [4] Prof.R.R.Bhambare, A.Koul, S.Mohd Bilal, S.Pande; “SMART VISION SYSTEM FOR BLIND; International Journal of Engineering and Computer Science”, ISSN: 2319-7242, vol. 3(5), pp. 5790-579, May 2014.
- [5] B.Shing Lin, C.Che Lee and P.Ying Chian; “Simple Smartphone-Based Guiding System for Visually Impaired People; Sensors 2017, 17, 1371; doi:10.3390/s17061371; Received: 23 April 2017; Accepted: 9 June 2017; Published: 13 June 2017.
- [6] Rajesh M., K Bindhu Rajan, A. Roy, K. Almaria Thomas, Ancy Thomas, B. Tharakan T, C Dinesh; “Text recognition and face detection aid for visually impaired person using Raspberry PI”; International Conference on circuits Power and Computing Technologies [ICCPCT]; 20-21 April 2017.
- [7] L. Britto Neto, F. Grijalva, V. Regina, M. Lima Maike, LuizCésar Martini, D. Florencio, M. CecíliaCalani Baranauskas, A. Anderson Rocha, S. Goldenstein, “A Kinect-Based Wearable Face Recognition System to Aid Visually Impaired Users” ; IEEE Transactions On Human-Machine Systems, vol. 47, No. 1, February 2017.
- [8] Prof. P Y Kumbhar, Mohd Attaullah, S. Dhere, S. Kumar Hipparagi; “Real Time Face Detection and Tracking Using OpenCV”; E-ISSN: 2349-7610; International Journal For Research In Emerging Science And Technology, vol-4(4), Apr-2017.
- [9] Nidhi; “Image Processing and Object Detection”; IJAR 2015; 1(9): 396-399 Online: [www.allresearchjournal.com](http://www.allresearchjournal.com); Received: 02-06-2015 Accepted: 05-07-2015.

- [10] [Online]: <https://www.cs.toronto.edu/~kriz/cifar.html> (Accessed on: 31th March, 2018).
- [11] Ren, S., He, K., R. Girshick, , J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems”, pp. 91-99, 2015.
- [12] [online].<https://www.mathworks.com/help/vision/examples/object-detection-using-faster-r-cnn-deep-learning.html#> (Accessed 7<sup>th</sup> July 2018)
- [13] He K., X. Zhang, S. Ren , J. Sun, “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition”, “Fleet D., Pajdla T., Schiele B., Tuytelaars T. (eds) Computer Vision – ECCV 2014”, Lecture Notes in Computer Science, vol 8691. Springer.
- [14] [Online]. <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/> (Accessed on : 10<sup>th</sup> July 2018)
- [15] [Online]. <https://www.raspberrypi.org/products/camera-module-v2/> (Accessed on : 10<sup>th</sup> July 2018)
- [16] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, F.Cheng-Yang, C. Alexander Berg; “SSD: Single Shot MultiBox Detector”; version-5,2016,Cornell University Library.
- [17] J. Redmon, S. Divvala, R. Girshick, A. Farhadi; “You Only Look Once: Unified, Real-Time Object Detection”; 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR); Electronic ISSN: 1063-6919; Published on: 12 December 2016.
- [18] J. Redmon, A. Farhadi; “YOLO9000: Better, Faster, Stronger; 2017” IEEE Conference on Computer Vision and Pattern Recognition (CVPR); Print ISSN: 1063-6919; Published on: 09 November 2017.
- [19] Nada, A. Mashaly, Samiaand , A. Fakhr, Mahmoud, S.Ahmed, “Human and Car Detection System for Blind People”; Conference: The 4th International Conference on Biomedical Engineering and Biotechnology – ICBEB 2015, Shanghai, China.
- [20] C.Ezhilarasi, R. Jeyameenachi, Mr.A.R. Aravind; “A Raspberry Pi based Assistive Aid for Visually Impaired Users”; International Journal of Advance Research and Innovative Ideas in Education; ISSN(O)-2395-4396; Vol-3(2) 2017.



- [21] A. R. S. Siswanto, A. S. Nugroho and M. Galinium, "Implementation of face recognition algorithm for biometrics based time attendance system," 2014 International Conference on ICT for Smart Society (ICISS), Bandung, 2014, pp. 149-154.
- [22] P. N. Belhumeur, J. P. Hespanha and D. J. Kriegman, "Eigenfaces vs. Fisherfaces: recognition using class specific linear thesision," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 7, pp. 711-720, Jul 1997.
- [23] [Online]. <https://pjreddie.com/darknet/yolo/> (Accessed on: 14<sup>th</sup> July 2018).
- [24] A. I. Arrahmah, A. Rahmatika, S. Harisa, H. Zakaria and R. Mengko, "Text-to-Speech device for patients with low vision," 2015 4th International Conference on Instrumentation, Communications, Information Technology, and Biomedical Engineering (ICICI-BME), Bandung, 2015, pp. 214-219.
- [25] Y. Wu and X. Ai, "Face Detection in Color Images Using AdaBoost Algorithm Based on Skin Color Information," First International Workshop on Knowledge Discovery and Data Mining (WKDD 2008), Adelaide, SA, 2008, pp. 339-342.
- [26] K. Zhang, Z. Zhang, Z. Li and Y. Qiao, "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks," in IEEE Signal Processing Letters, vol. 23, no. 10, pp. 1499-1503, Oct. 2016.
- [27] P. Viola and M. J. Jones, "Robust real-time face detection. International journal of computer vision," vol. 57, no.2, pp. 137-154, 2004.
- [28] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, A. Rabinovich, "Going deeper with convolutions, In Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 1-9,2015.
- [29] [Online]. <https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/> (Accessed on 19<sup>th</sup> July 2018).
- [30] [Online]. <https://www.tensorflow.org/> (Accessed on 29<sup>th</sup> July 2018).
- [31] D. Impiombato, S. Giarrusso, T. Mineo, O. Catalano, C. Gargano, G. La Rosa, F. Russo, G. Sottile,
- [32] S. Billotta, G. Bonanno, S. Garozzo, A. Grillo, D. Marano, and G. Romeo, "You Only Look Once:Unified, Real-Time Object Detection,"Nuclear Instruments and Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, vol. 794, pp. 185–192, 2015.

- [33] J. Hu, L. Peng and L. Zheng, "XFace: A Face Recognition System for Android Mobile Phones," 2015 IEEE 3rd International Conference on Cyber-Physical Systems, Networks, and Applications, Hong Kong, 2015, pp. 13-18.