# CHAPTER 1 INTRODUCTION

The face recognition and tracking are important in many computer vision applications. Here we develop a simple system for tracking a face in a live captured by web cam. Face recognition can be in one of two modes: Detection or tracking. In detection mode we can use a vision to detect a face in current frame. In tracking we must track the points using point tracker. Detection is used in biometrics, often as a part of (or together with) a facial recognition system. It is also used in video surveillance, human computer interface and image database management. Some recent digital cameras use face detection for autofocus. Face detection is gaining the interest of marketers. A webcam can be integrated into a television and detect any face that walks by. The system then calculates the race, gender, and age range of the face.

Face detection is also being researched in the area of energy conservation [Energy Conservation].  Energy Conservation can be done by Detecting the Face and reducing the brightness of complete image and then adjusting the brightness of the particular area of an image where the face is located using histogram equalization. Methodology for face recognition based on information theory approach of coding and decoding the face image is discussed in. Proposed methodology is connection of two stages – Face detection using Haar Based Cascade classifier and recognition using LBPH (Local Binary Pattern Histogram) algorithm. Various face detection and recognition methods have been evaluated and also solution for image detection and recognition is proposed as an initial step for video surveillance. Recently various methods for a **local feature extraction** emerged. To avoid the high-dimensionality of the input data only local regions of an image are described, the extracted features are (hopefully) more robust against partial occlusion, Illumination and small sample size. Algorithms used for a local feature extraction is Local Binary Pattern. It’s still an open research question what’s the best way to preserve spatial information when applying a local feature extraction, because spatial information is potentially useful information.

## 1.1 MOTIVATION

Object recognition is important in many computer vision applications including activity automotive safety, surveillance, mask detection, helmet detection. In this example we will develop a simple system for detecting and recognising a single face in a live video stream captured by a webcam.

## 1.2 PROBLEM DEFINATION

One of the most critical decision points in the design of a face recognition system is the choice of an appropriate face representation. Effective feature descriptors are expected to convey sufficient Invariant and non-redundant facial information. A current problem in the area of Computer Vision is the successful implementation of a working facial recognition system. As mentioned above, this area has excellent commercial potential for areas such as border control and could replace key lock mechanisms. In this scenario, the door could automatically be opened as a person approaches it given that they have a face held on the database. However, there are few facial recognition systems that work reliably and even fewer that are cost effective. Essentially, the field of facial recognition currently remains firmly in the realm of research. This will remain the case until certain problems have been overcome. This is not to say that considerable progress has not been made in recent years in some of the problem areas; a robust face detection system now exists that accurately locates a face in a complex background and in a variety of sizes (Viola & Jones, 2004). Further problems that are still under investigation exist and some of them are addressed in this study:

* + 1. **Facial Characteristics**:

Glasses, hair styles and partial facial occlusion can all affect the ability of the face detector to accurately localise the face. Some improvements have been made in this area and the Viola & Jones face detector (mentioned above) are now fairly difficult to fool.

* + 1. **Facial Expressions:**

A person that is smiling or frowning may lead to incorrect measurements being taken from around the subject’s mouth. This is due to the fact that many facial recognition systems use training data to train a system to be able to recognise a particular face. During this training period, the faces used are normalised in a single position and are therefore susceptible to changes in facial expressions.

* + 1. **Lighting Conditions:**

Variation in lighting conditions can throw off a feature extraction algorithm because the templates are essentially incorrect. It then becomes very difficult to locate features and a possible match may be missed.

* + 1. **Face Position:**

For many legacy facial recognition systems, it was expected that the face would be offered to the face detector as a „straight-on‟ representation of the face. Even now, variations on this are extremely difficult to deal with. Even though the Viola & Jones face detector has the ability to locate faces from a „side-on‟ view it is difficult to do this in an automated fashion. From a technical perspective, the face may be partially occluded if not straight on and the features will appear slightly closer together (or further apart) even if the face is accurately detected but the pose is partially off-centre.

## 1.3 OBJECTIVE

The objective of facial recognition system is for automatically identifying or verifying human faces from digital image or a video frame from video source. It detects facial features and ignores anything else, such as buildings, trees and bodies.

The main purpose of this thesis was to design and to implement a software project in which the learned methods and programming languages would be used. From the beginning, only open source code was used. The minimum requirements for the application were to be able to detect and to recognize different individuals by using their faces. The method used is the LBPH, which is considered to be the successful example of facial recognition technology. This method is not 100% accurate, but is a good start for people who are interested in this field.

## 1.4 SCOPE

The scope of this project is to capture and process video frames from the web cam in a loop to detect and track a face. In future it can replace biometric system for identification. In upcoming decades, face recognition is going to replace finger print attendance system, sensor door application, traffic camera police and ink marking during election.

## 1.5 APPLICATIONS

Face recognition has numerous applications based on either the identification or the verification of the presence of individuals in still images or video sequences. From entertainment to security, the ability to automatically recognise an individual can simplify or improve a number of tasks. According to Zhao et al. the main areas of application of face recognition are Information Security, surveillance and law enforcement, security application based on smart card and of course entertainment as well as in education system.

**1.5.1 Information Security:**

There are numerous instances in our everyday life where we are required to input passwords or PIN codes to use a device or access personal data. While such forms of security measures are essential, they are not always desirable or user-friendly. Using a robust face recognition system, such applications would be able to identify the user and allow or deny access accordingly. Systems like a computer logging on to the user’s account automatically when the user approached or an ATM only allowing the owner of the card to withdraw money could all be made possible.

**1.5.2 Surveillance & Law enforcement:**

Video surveillance has become a part of today’s society. Although opinions about CCTV are varied, security could in several occasions be improved using an automatic face recognition system.

**1.5.3 Smart card-based applications:**

Driving licences, passports and ID cards all require mug shots of the individuals to be taken under controlled conditions. Using a face recognition system, these mug shots could be used potentially simplifying processes such as airport checks.

**1.5.4 Education:**

We can modify our attendance system by replacing finger print and manual system.

**CHAPTER-2 LITERATURE SURVEY**

# 

## 2.1 BACKGROUND/MOTIVATION

The subject of face recognition is as old as computer vision, both because of the practical importance of the topic and theoretical interest from cognitive scientists. Despite the fact that other methods of identification (such as fingerprints, or iris scans) can be more accurate, face recognition has always remains a major focus of research because of its non-invasive nature and because it is people's primary method of person identification.

Perhaps the most famous early example of a face recognition system is due to Kohonen, who demonstrated that a simple neural net could perform face recognition for aligned and normalized face images. The type of network he employed computed a face description by approximating the eigenvectors of the face image's autocorrelation matrix these eigenvectors are now known as eigenfaces.

Kohonen's system was not a practical success, however, because of the need for precise alignment and normalization. In following years many researchers tried face recognition schemes based on edges, inter-feature distances, and other neural net approaches. Kirby and Sirovich (1989) later introduced an algebraic manipulation which made it easy to directly calculate the eigenfaces, and showed that fewer than 100 were required to accurately code carefully aligned and normalized face images. Turk and Pentland (1991) then demonstrated that the residual error when coding using the eigenfaces could be used both to detect faces in cluttered natural imagery, and to determine the precise location and scale of faces in an image. They then demonstrated that by coupling this method for detecting and localizing faces with the eigenface recognition method, one could achieve reliable, real-time recognition of faces in a minimally constrained environment. This demonstration that simple, real-time pattern recognition techniques could be combined to create a useful system sparked an explosion of interest in the topic of face recognition. After eigenfaces, LBPH was introduced which works more accurately than eigenfaces.

## 2.3 EXISTING SYSTEM

The Eigen face is the first method considered as a successful technique of face recognition. The Eigen face method uses Principal Component Analysis (PCA) to linearly project the image space to a low dimensional feature space.

* In existing system criminal information is stored in file format withoutdated images and low-quality images.
* Using this information, it is not possible for eye witness to guess

the criminal, so there is need to develop a better method for

identifying criminals like fingerprints, DNA etc.

* Among all these methods face recognition method is cost

effective and more accurate.

## 2.4 DISADVANTAGES

* It is contrast based.
* Eigen face has difficulty in capturing expression changes.
* Don't provide useful information regarding the actual face.
* Cannot handle varying illumination conditions effectively.
* First, the method is scale-sensitive, — requiring some pre-processing for scale normalization.
* Secondly, its recognition rate decreases for recognition under varying pose and illumination.
* Third, eigenface approach is not robust when dealing with extreme variations in pose as well as in expression and disguise.
* Fourth, the method may require uniform background which may not be satisfied in most natural scenes.
* The eigenfaces has been eclipsed by far more better, robust approaches. As such, it is now commonly used only as an introductory method.
* An **advantage** of this **algorithm** is that the **eigenfaces** were invented exactly for those purpose what makes the system very efficient [4]. A **drawback** is that it is sensitive for lightening conditions and the position of the head. **Disadvantages**-Finding the **eigenvectors** and **eigenvalues** are time consuming on PPC.

## 2.5 PROPOSED SYSTEM

LBPH algorithm extracts useful features that discriminate one

person from the others. The LBPH is especially useful when facial

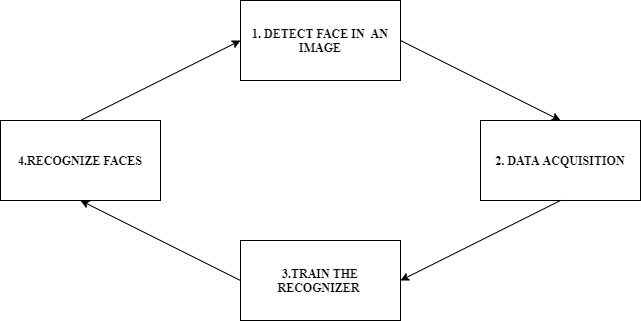
images have large variations in illumination and facial expression.

* Ease of recognition (Using this system eye witness can guess the criminals easily).
* High accuracy
* Easy and fast process

## 2.6 ADVANTAGES

* Performs dimensionality reduction.
* More accurate than Eigen faces.
* It produces good results even under varying illumination conditions.

**CHAPTER-3 PROCESS OF FACE RECOGNITION**

****

## *Fig 3.1 Process of Face Recognition*

## 3.1 FACE DETECTION

Face detection is a computer technology that determines the locations and sizes of human faces in digital images. It detects face and ignores anything else, such as buildings, trees and bodies. Face detection can be regarded as a more general case of face localization. In face localization, the task is to find the locations and sizes of a known number of faces (usually one). Face-detection algorithms focus on the detection of frontal human faces.

A crucial part of automated face recognition systems is their ability to successfully locate and extract faces from images. In order to achieve that without requiring user input, face objects need to be extracted from their background and the boundaries of these faces need to be detected.

The face detection problem can be divided in a number of different levels:

* Firstly, a face must be successfully distinguished from non-face objects. Locating faces in a busy image is a trivial task for human vision but it can prove challenging for a computer system. The computer’s understanding of what a face should look like is limited to the analysis performed on sets of pixels and as a result semantic segmentation of an image as performed by humans is very difficult if not impossible. Consequently, methods focus on the characteristics of a face that make it different to other objects such as the colour of the skin or the general structure of shadows on a face.
* Detecting a face in an image alone may not provide enough information in order to segment that face from its background. As a result, the boundaries of the face must be located in order to not simply locate the presence of a face but to extract it from the image.
* A large group of face recognition methods are based on facial features. Systems using such methods need to successfully locate and extract features such as the eyes or mouth from faces as well as the face itself.
  + 1. **Locating the Face**

Real world images can contain busy backgrounds and several different objects making the amount of information available in them overwhelming compared to the slight variations between different faces. Most face recognition systems need to be able to extract faces from the image in order to effectively process them. Face detection at the simplest level can be understood as the process of locating faces in images and extracting them from their background. Although it is a well defined problem in theory, variations in the image can pose several challenges:

* **Illumination:** Varying lighting conditions can greatly affect the appearance of a face. Details can be lost in darker areas or intense highlights. Moreover, characteristics of the face such as edges or brightness differences near feature points become less important when lighting conditions are more extreme. For instance, the same individual is pictured in figure 4.1 under varying illumination.
* **Pose:** Face rotation, size variations as well as the difficulty to predict where faces pose could be located in an image are obstacles that face detection algorithms need to overcome. A rotated face may be easily recognisable by a human viewer, however with enough rotation, the structure of the face from the new angle is sufficiently different to make its detection difficult for a face detection system. Similarly, extra information may be required in order to detect faces closer or further away from the camera.
* **Expression:** In controlled situations, such as passport or driving license photographs, facial expressions are limited. However, in less strict conditions, different expressions must be taken into account when trying to locate a face in an image, especially when detection is based on facial features.
* **Occlusion:** Faces in images can be partially occluded by other objects in the scene, accessories such as hats or glasses or simply by a rebellious hairstyle. Effective face detection systems should be able to successfully locate a face even using partial information depending on the application. However, it is important in this case to decide what percentage of the face should be visible in order for an object to be classified as a face.
* The above challenges alone can make face detection a difficult problem. However, in addition to them, most face detection or recognition systems would greatly benefit or even require fast detection of faces in images or video sequences.Based on the classification defined by Yang et al. detection methods can be divided in four main categories based on the techniques they use.
* **Knowledge based methods:** This class of methods tries to locate faces using information about what humans perceive as a face such as typical distances between features.
* **Feature invariant methods:** In this case, methods focus on elements of the face that stay invariant when pose, expression or illumination change.
* **Template matching methods:** This class of methods uses a number of templates for the face or for facial features and tries to locate a face by correlating each template with the given image.
* **Appearance based methods:** These methods use a model of a face created through training over a set of different faces.

### Face Detection Using Haar Like Features

Several different techniques can be applied in order to locate faces in images. They can be based on skin colour information, the general structure of a face, or even templates of different faces. The method chosen for this project is based on Haar like features and is developed by Viola and Jones. Haar features, as used for the detection or classification of objects, can be understood as rectangles containing darker and lighter sections. Different features are applied on sections of the image. Depending on the feature shape, operations are performed between the pixels covered by the darker and the lighter sections to determine if the structure of the tested area matches the expectations of the algorithm.

Features can have different shapes or sizes and they can be combined in different ways to describe more complex objects. The set of all possible combinations for features of a certain size however is very large. Although features can be evaluated quickly, the number of calculations that would be required for the complete set would make this method impractical. To solve this problem, a weak learning algorithm based on Ada Boost is used to select a small number of features that when combined can effectively describe a face. At each round, a single feature is selected from the complete set that gives the lowest error which forms a weak classifier. After looking through the possible features and rejecting a large number of them, the remaining ones are combined to form a strong classifier. Classifiers are then applied to the image in a cascading manner in order to locate a face. At each round, a successively more complex classifier is used. If at any round a part of the image is rejected, it will then be ignored by subsequent classifiers, thus gradually limiting the search space. More complex classifiers are only applied to regions of the image with potential interest. The cascade application of classifiers greatly improves the performance of the method as the section of the image that needs to be searched is reduced at every round. Another aspect of the Viola-Jones method that improves its performance is a representation of the image known as the Integral Image which is used instead of the intensity image. The Integral Image at each pixel can be calculated by the following formula ii(x, y) = P

x0≤x,y0≤y

i(x0, y0)

Which is simply the sum of the pixels left and above each pixel. Using this representation of the image, features applied on different sections can be evaluated rapidly at various scales. A number of reasons lead to the choice of this method as the primary face detection solution for this project. The Viola-Jones method can successfully detect upright faces in most situations without requiring any user input. This is critical for automatic face recognition systems as the correct detection and extraction of the faces in an image is the starting point of the recognition process. Furthermore, although fast detection was not essential in this situation, the almost real-time performance of this method gives it a clear advantage.

As this project focuses mainly on face recognition, an existing implementation of this method was used to provide the required face detection functionality. A well-known collection of various Vision algorithms is OpenCV, a C/C++ open-source library provided by Intel. The face detection method provided with this library is an implementation of the Viola-Jones algorithm with a number of additional improvements. Training of the algorithm is already performed and the appropriate classifiers required for face detection are selected.

## 3.2 DATA ACQUISITION

There is a wide variety of works related to facial landmark localization. The early researches extract facial landmarks without a global model. Facial landmarks, such as the eye corners and centres, the mouth corners and centre, the nose corners, chin and cheek borders are located based on geometrical knowledge. The first step consists of the establishment of a rectangular search region for the mouth and a rectangular search region for the eyes. The borders are extracted by applying corner detection algorithm such as SUSAN border extraction algorithm. Such methods are fast, however, they could not deal with faces of large variation in appearance due to pose, rotation, illumination and background changes.

### 3.2.1. Eye Localization

The eye localization is a crucial step towards automatic face recognition and facial landmark localization due to the fact that these face related applications need to normalize faces, measure the relative positions or extract features according to eye positions. Like other problems of object detection under complex scene such as face detection, car detection, eye patterns also have large variation in appearance due to various factors, such as size, pose, rotation, the closure and opening of eyes, illumination conditions, the reflection of glasses and the occlusion by hairs etc. Even having found the positions of faces grossly, robustly and precisely locating the eye’s center is still a challenging task. A variety of eye detection and tracking algorithms have been proposed in recent years, but most of them can only deal with part of these variations or be feasible under some constraints. We have devised a novel approach for precisely locating eyes in face areas under a probabilistic framework. The experimental results demonstrate that our eye localization method can robustly cope with different eye variations and achieve higher detection rate on diverse test sets.

Whatever faces we have detected in detection phase, e will store it in our directory for further use.

### 3.3 TRAIN THE RECOGNIZER

On this face, we are going to utilise our collected images which was collected in Data Acquisition phase. We will fetch all the images from the directory and train our recognizer to get the accurate result.

Once we have trained our recognizer, it will produce a .yml file which is a trained file. In this case our recognizer will be LBPH algorithm.

**LBPH (Local Binary Pattern Histogram) algorithm**

**Local Binary Pattern**(LBP) is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number.

It was first described in 1994 (LBP) and has since been found to be a powerful feature for texture classification. It has further been determined that when LBP is combined with histograms of oriented gradients (HOG) descriptor, it improves the detection performance considerably on some datasets.

Using the LBP combined with histograms we can represent the face images with a simple data vector.

As LBP is a visual descriptor it can also be used for face recognition tasks, as can be seen in the following step-by-step explanation.

# Step-by-Step

Now that we know a little more about face recognition and the LBPH, let’s go further and see the steps of the algorithm:

* **Parameters**: the LBPH uses 4 parameters:
* **Radius**: the radius is used to build the circular local binary pattern and represents the radius around the central pixel. It is usually set to 1.
* **Neighbors**: the number of sample points to build the circular local binary pattern. Keep in mind: the more sample points you include, the higher the computational cost. It is usually set to 8.
* **Grid X**: the number of cells in the horizontal direction. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector. It is usually set to 8.
* **Grid Y**: the number of cells in the vertical direction. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector. It is usually set to 8.

Don’t worry about the parameters right now, you will understand them after reading the next steps.

1. **Training the Algorithm**:

First, we need to train the algorithm. To do so, we need to use a dataset with the facial images of the people we want to recognize. We need to also set an ID (it may be a number or the name of the person) for each image, so the algorithm will use this information to recognize an input image and give you an output. Images of the same person must have the same ID. With the training set already constructed, let’s see the LBPH computational steps.

1. **Applying the LBP operation**:

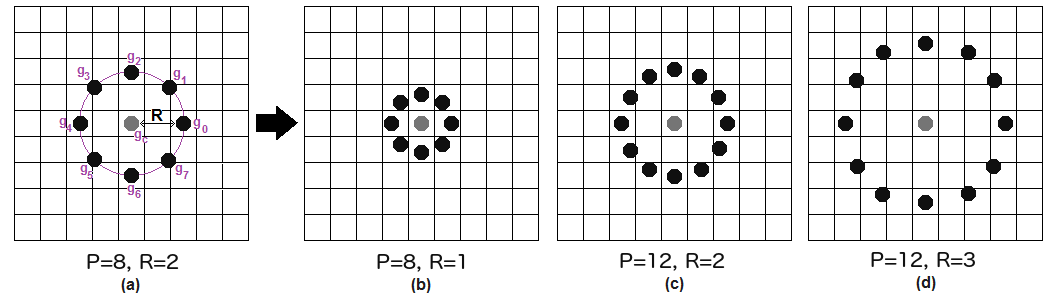
The first computational step of the LBPH is to create an intermediate image that describes the original image in a better way, by highlighting the facial characteristics. To do so, the algorithm uses a concept of a sliding window, based on the parameters, **radius** and **neighbors**.

The image below shows this procedure:



Based on the image above, let’s break it into several small steps so we can understand it easily:

* Suppose we have a facial image in grayscale.
* We can get part of this image as a window of 3x3 pixels.
* It can also be represented as a 3x3 matrix containing the intensity of each pixel (0~255).
* Then, we need to take the central value of the matrix to be used as the threshold.
* This value will be used to define the new values from the 8 neighbors.
* For each neighbor of the central value (threshold), we set a new binary value. We set 1 for values equal or higher than the threshold and 0 for values lower than the threshold.
* Now, the matrix will contain only binary values (ignoring the central value). We need to concatenate each binary value from each position from the matrix line by line into a new binary value (e.g. 10001101). Note: some authors use other approaches to concatenate the binary values (e.g. clockwise direction), but the final result will be the same.
* Then, we convert this binary value to a decimal value and set it to the central value of the matrix, which is actually a pixel from the original image.
* At the end of this procedure (LBP procedure), we have a new image which represents better the characteristics of the original image.
* **Note**: The LBP procedure was expanded to use a different number of radius and neighbors, it is called Circular LBP.

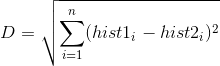


It can be done by using **bilinear interpolation**. If some data point is between the pixels, it uses the values from the 4 nearest pixels (2x2) to estimate the value of the new data point.

**3.4 RECOGNITION**

It automatically, identify a person face from live video and it displays rectangular frames on faces. In this step, the algorithm is already trained. Each histogram created is used to represent each image from the training dataset. So, given an input image, we perform the steps again for this new image and creates a histogram which represents the image.

* So, to find the image that matches the input image we just need to compare two histograms and return the image with the closest histogram.
* We can use various approaches to compare the histograms (calculate the distance between two histograms), for example: E**uclidean distance**, **chi-square**, **absolute value**, etc. In this example, we can use the Euclidean distance (which is quite known) based on the following formula:



* So, the algorithm output is the ID from the image with the closest histogram. The algorithm should also return the calculated distance, which can be used as a ‘**confidence**’ measurement. **Note**: don’t be fooled about the ‘confidence’ name, as lower confidences are better because it means the distance between the two histograms is closer.
* We can then use a threshold and the ‘confidence’ to automatically estimate if the algorithm has correctly recognized the image. We can assume that the algorithm has successfully recognized if the confidence is lower than the threshold defined.
* On this phase, we will capture a fresh photo and capture only face from the photo and convert it to grey scale followed by which we compare this reduced grey scale face to our dataset if it matches then we will return the respective name.

**CHAPTER-4 ANALYSIS**

## 4.1 SOFTWARE REQUIREMENT SPECIFICATION

The software requirements specification specifies the functional requirements and non-functional requirements.

Functional requirements refers to how the system is going to react according to the input provided and how it is going to behave in particular situations Like when the original message given is of other than English and how the system reacts when the text is of English and when the audio file is of other than „.au‟ format.

Non-functional requirements refer to Usability, Reliability, Availability, Performance, Security, Supportability, and Interface.

## 4.2 USER REQUIREMENT

The user requirement specifies what the user expects the software to be able to do. Once the required information is completely gathered it is documented in a URD, which is meant to spell out exactly what the software must do and becomes part of the contractual agreement. A customer cannot demand features not in the URD, whilst the developer cannot claim the product is ready if it does not meet an item of the URD. The URD can be used as a guide to planning cost, timetables, milestones, testing, etc.

### 4.2.1 Functional requirements

This application mainly consists of two modules:

* System
* Images/faces

### 4.2.2 Software requirements

The software requirements of the project are mentioned below:

* Platform: windows 10
* Environment: python 2.7/3.3

### 4.2.3 Hardware requirements

* System
* Camera

|  |  |  |
| --- | --- | --- |
| * Processor | : | Pentium IV 2.4 GHz |
| * RAM | : | 1 GB |
| * Hard disk | : | 40 GB |
|  |  |  |

## 4.3 PROCESS MODEL USED WITH JUSTIFICATION

**4.3.1. SDLC (Umbrella Model):**

SDLC is nothing but Software Development Life Cycle. It is a standard which is used by software industry to develop good software.

**4.3.2. Stages in SDLC:**

* Requirement Gathering
* Analysis
* Designing
* Coding
* Testing
* Maintenance

### Requirement Gathering Stage

The requirements gathering process takes as its input the goals identified in the high-level requirements section of the project plan. Each goal will be refined into a set of one or more requirements. These requirements define the major functions of the intended application, define Operational data areas and reference data areas, and define the initial data entities. Major functions include critical processes to be managed, as well as mission critical inputs, outputs and reports. A user class hierarchy is developed and associated with these major functions, data areas, and data entities. Each of these definitions is termed a Requirement. Requirements are identified by unique requirement identifiers and, at minimum, contain a requirement title and textual description.These requirements are fully described in the primary deliverables for this stage: the Requirements Document and the Requirements Traceability Matrix (RTM). The requirements document contains complete descriptions of each requirement, including diagrams and references to external documents as necessary. Note that detailed listings of database tables and fields are *not* included in the requirements document.

The title of each requirement is also placed into the first version of the RTM, along with the title of each goal from the project plan. The purpose of the RTM is to show that the product components developed during each stage of the software development lifecycle are formally connected to the components developed in prior stages.

In the requirements stage, the RTM consists of a list of high-level requirements, or goals, by title, with a listing of associated requirements for each goal, listed by requirement title. In this hierarchical listing, the RTM shows that each requirement developed during this stage is formally linked to a specific product goal. In this format, each requirement can be traced to a specific product goal, hence the term requirements traceability. The outputs of the requirements definition stage include the requirements document, the RTM, and an updated project plan. Feasibility study is all about identification of problems in a project.

No. of staff required to handle a project is represented as Team Formation, in this case only modules are individual tasks will be assigned to employees who are working for that project. Project Specifications are all about representing of various possible inputs submitting to the server and corresponding outputs along with reports maintained by administrator

### Analysis Stage

The planning stage establishes a bird's eye view of the intended software product, and uses this to establish the basic project structure, evaluate feasibility and risks associated with the project, and describe appropriate management and technical approaches.

The most critical section of the project plan is a listing of high-level product requirements, also referred to as goals. All of the software product requirements to be developed during the requirements definition stage flow from one or more of these goals. The minimum information for each goal consists of a title and textual description, although additional information and references to external documents may be included. The outputs of the project planning stage are the configuration management plan, the quality assurance plan, and the project plan and schedule, with a detailed listing of scheduled activities for the upcoming Requirements stage, and high level estimates of effort for the out stages.

### Designing Stage

The design stage takes as its initial input the requirements identified in the approved requirements document. For each requirement, a set of one or more design elements will be produced as a result of interviews, workshops, and/or prototype efforts. Design elements describe the desired software features in detail, and generally include functional hierarchy diagrams, screen layout diagrams, tables of business rules, business process diagrams, pseudo code, and a complete entity-relationship diagram with a full data dictionary. These design elements are intended to describe the software in sufficient detail that skilled programmers may develop the software with minimal additional input.

### Development Stage

The development stage takes as its primary input the design elements described in the approved design document. For each design element, a set of one or more software artifacts will be produced. Software artifacts include but are not limited to menus, dialogs, and data management forms, data reporting formats, and specialized procedures and functions. Appropriate test cases will be developed for each set of functionally related software artifacts, and an online help system will be developed to guide users in their interactions with the software.

The RTM will be updated to show that each developed artifact is linked to a specific design element, and that each developed artifact has one or more corresponding test case items. At this point, the RTM is in its final configuration. The outputs of the development stage include a fully functional set of software that satisfies the requirements and design elements previously documented, an online help system that describes the operation of the software, an implementation map that identifies the primary code entry points for all major system functions, a test plan that describes the test cases to be used to validate the correctness and completeness of the software, an updated RTM, and an updated project plan.

### Integration and Test Stage

During the integration and test stage, the software artifacts, online help, and test data are migrated from the development environment to a separate test environment. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite confirms a robust and complete migration capability. During this stage, reference data is finalized for production use and production users are identified and linked to their appropriate roles. The final reference data (or links to reference data source files) and production user list are compiled into the Production Initiation Plan.

The outputs of the integration and test stage include an integrated set of software, an online help system, an implementation map, a production initiation plan that describes reference data and production users, an acceptance plan which contains the final suite of test cases, and an updated project plan.

### Installation & Acceptance Stage

During the installation and acceptance stage, the software artifacts, online help, and initial production data are loaded onto the production server. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite is a prerequisite to acceptance of the software by the customer. After customer personnel have verified that the initial production data load is correct and the test suite has been executed with satisfactory results, the customer formally accepts the delivery of the software.

The primary outputs of the installation and acceptance stage include a production application, a completed acceptance test suite, and a memorandum of customer acceptance of the software. Finally, the PDR enters the last of the actual labour data into the project schedule and locks the project as a permanent project record. At this point the PDR "locks" the project by archiving all software items, the implementation map, the source code, and the documentation for future reference.

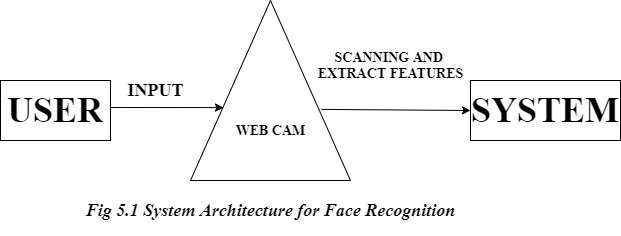
### Maintenance

Outer rectangle represents maintenance of a project, Maintenance team will start with requirement study, understanding of documentation later employees will be assigned work and they will undergo training on that particular assigned category. For this life cycle there is no end, it will be continued so on like an umbrella (no ending point to umbrella sticks).

**CHAPTER-5 DESIGN**

Systems design is the process of defining the [architecture,](https://en.wikipedia.org/wiki/Systems_architecture) modules, interfaces, and [data](https://en.wikipedia.org/wiki/Data) for a [system](https://en.wikipedia.org/wiki/System) to satisfy specified [requirements.](https://en.wikipedia.org/wiki/Requirement) Systems design could be seen as the application of [systems theory](https://en.wikipedia.org/wiki/Systems_theory) to [product development.](https://en.wikipedia.org/wiki/Product_development) There is some overlap with the disciplines of [systems analysis,](https://en.wikipedia.org/wiki/Systems_analysis) [systems architecture](https://en.wikipedia.org/wiki/Systems_architecture) and [systems engineering.](https://en.wikipedia.org/wiki/Systems_engineering)

## 5.1 SYSTEM ARCHITECTURE



## 5.2 UML DIAGRAMS

Unified Modelling language (UML) is a standardized modelling language enabling developers to specify, visualize, construct and document artifacts of a software system. Thus, UML makes these artifacts scalable, secure and robust in execution. UML is an important aspect involved in object-oriented software development. It uses graphic notation to create visual models of software systems.

The Unified Modelling Language allows the software engineer to express an analysis model using the modelling notation that is governed by a set of syntactic semantic and pragmatic rules.

A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagrams, which is as follows.

1) User Model View

* This view represents the system from the user’s perspective.
* The analysis representation describes a usage scenario from the end-user’s perspective.

2) Structural model view

* In this model the data and functionality are arrived from inside the system.
* This model view models the static structures.

3) Behavioural Model View

* It represents the dynamic of behavioural as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.

4) Implementation Model View

* In this the structural and behavioural as parts of the system are represented as they are to be built.

5) Environmental Model View

* In this, the structural and behavioural aspects of the environment in which the system is to be implemented are represented.
* UML is specifically constructed through two different domains they are:
* UML Analysis modelling, this focuses on the user model and structural model views of the system.
* UML design modelling, which focuses on the behavioural modelling, implementation modelling and environmental model views.

### UML Concepts

The Unified Modelling Language (UML) is a standard language for writing software blue prints.

The UML is a language for

* Visualizing
* Specifying
* Constructing
* Documenting the artifacts of a software intensive system.

The UML is a language which provides vocabulary and the rules for combining words in that vocabulary for the purpose of communication. A modelling language is a language whose vocabulary and the rules focus on the conceptual and physical representation of a system.

Modelling yields an understanding of a system.

### Building Blocks of the UML

The vocabulary of the UML encompasses three kinds of building blocks:

* Things
* Relationships
* Diagrams

Things are the abstractions that are first-class citizens in a model; relationships tie these things together; diagrams group interesting collections of things

### Things in the UML

There are four kinds of things in the UML:

* Structural things
* Behavioural things
* Grouping things
* Annotational things

1. **Structural Things:**

These are the nouns of UML models. The structural things used in the project design are first.

### Class

A class is a description of a set of objects that share the same attributes, operations, relationships and semantics.

|  |
| --- |
| Window |
| Origin  Size |
| open() close() move()  display() |

**class**

### Interface

Interface defines a set of operations, which specify the responsibility of a class.



**interface**

### Collaboration

Collaboration defines an interaction between elements.



**collaboration**

### Use Case

A use case is a description of set of sequence of actions that a system performs that yields an observable result of value to particular actor.



**Use Case**

### Component

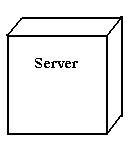
Component describes the physical part of a system.



**Component**

### Node

A node is a physical element that exists at runtime and represents a computational resource, generally having at least some memory and often processing capability.



### Node

1. **Behavioural Things:**

Theseare the dynamic parts of UML models. The behavioural thing used is:

**Interaction**

An interaction is a behaviour that comprises a set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose.



**Messages**

### State machine

State machine is useful when the state of an object in its life cycle is important. It defines the sequence of states an object goes through in response to events. Events are external factors responsible for state change



### State

1. **Grouping Things:**

These can be defined as a mechanism to group elements of a UML model together. There is only one grouping thing available.

### Package

Package is the only one grouping thing available for gathering structural and behavioural things.



### Package

1. **Annotational Things:**

These can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements.

### Note

It is the only one Annotational thing available. A note is used to render comments, constraints, etc. of an UML element.



**Note**

### Relationships in the UML

There are four kinds of relationships in the UML:

* Dependency
* Association
* Generalization
* Realization

**Dependency:** It is a semantic relationship between two things in which a change to one thing may affect the semantics of the other thing (the dependent thing).



### Dependencies

**Association:** Itis a structural relationship that describes a set links, a link being a connection among objects. Aggregation is a special kind of association, representing a structural relationship between a whole and its parts.



### Association

**Generalization:** Itis a specialization/ generalization relationship in which objects of the specialized element (the child) are substitutable for objects of the generalized element (the parent).



### Generalization

**Realization:** It is a semantic relationship between classifiers, where in one classifier specifies a contract that another classifier guarantees to carry out.



**Realization**

### Use case Diagram

**Use case:** It is a description of set of sequence of actions that a system performs that yields an observable result of value to particular actor.



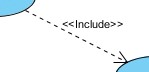
### Use case

**Actors:** They are the entities that interact with a system. Although in most cases, actors used to represent the users of system, actors can actually be anything that needs to exchange information with the system. So, an actor may be people, computer hardware, other systems, etc.



### Actor

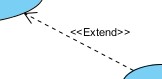
**Include**



### Include

An include relationship specifies how the behaviour for the inclusion use case is inserted into the behaviour defined for the base use case.

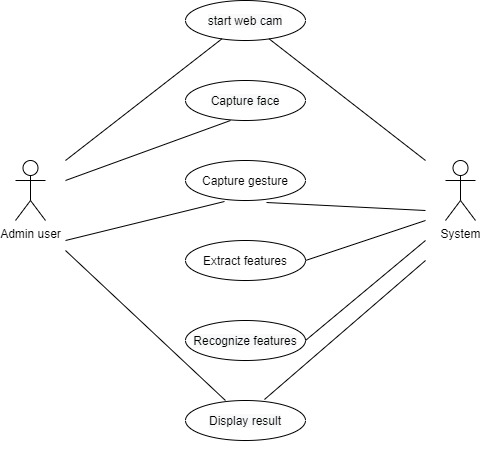
### Extend



### Extend

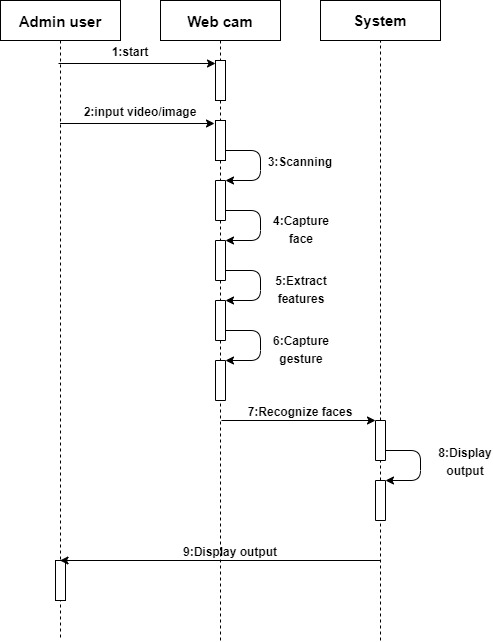
An extend relationship specifies how the behaviour of the extension use case can be inserted into the behaviour defined for the base use case.

**Use Case Diagram**



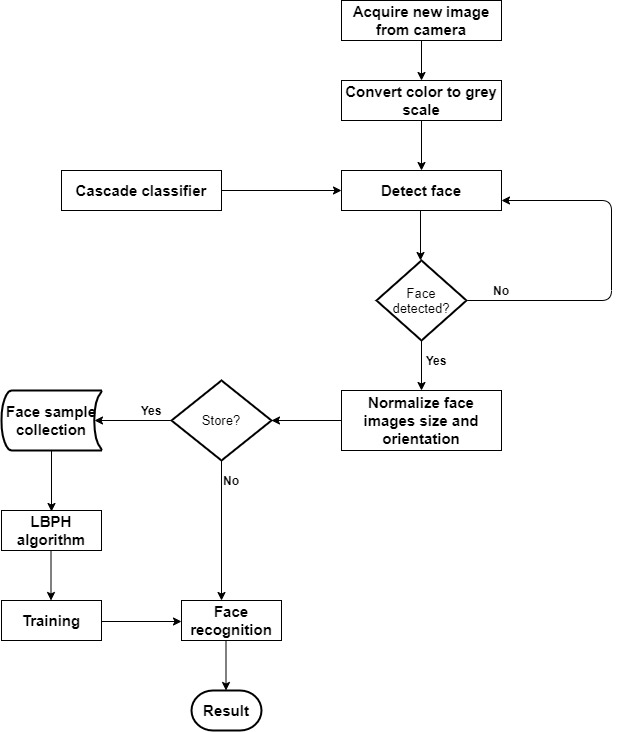
### Fig 5.2.1 Use Case Diagram for Face Recognition

**Sequence Diagram**

****

### Fig 5.2.2 Sequence Diagram for Face Recognition

**Flow Chart Diagram**

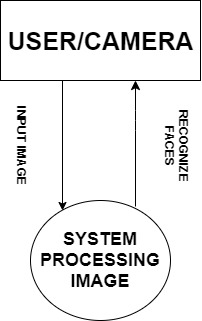
****

### Fig 5.2.3 Flow Chart Diagram for Face Recognition

**Data Flow Diagram**

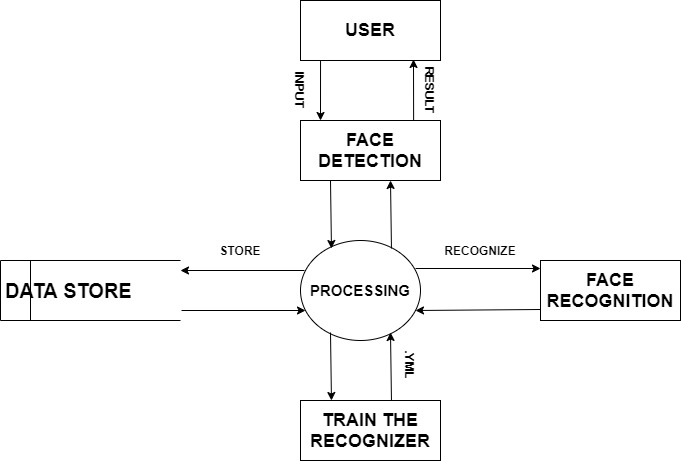
* Data flow diagram is a graphical representation of data flow through an information system.
* It is capable of depicting incoming data flow, outgoing data flow and stored data.
* There is a prominent difference between DFD and Flowchart. The flowchart depicts flow of control in program modules. DFDs depict flow of data in the system at various levels.
* DFD Components:
  1. Entities
  2. Process
  3. Data Store
  4. Data Flow

**Zero Level DFD**

****

### Fig 5.2.4 Zero Level DFD for Face Recognition

**One Level DFD**

****

### Fig 5.2.5 One Level DFD for Face Recognition

**CHAPTER-6 IMPLEMENTATION**

### 6.1. Implementation of Software/Hardware

Processing takes the video input from the webcam and uses the OpenCV library to analyse the video. If a face is detected in the video, the OpenCV library will give the Processing sketch the coordinates of the face. The processing sketch will determine where the face is located in the frame, relative to the centre of the frame, and send this data through a serial connection to an Arduino. The Arduino will use the data from the Processing sketch to move the servos connected the Servo setup as shown in figure 9. a) Basically haar-cascade classifier is used for detecting the faces. b) The input video frame is read from camera and temporary memory storage is created to store this frame. c) A window is created to capture the display frame and frame is continuously monitored for its existence. d) A function is called to detect the face where the frame is passed as parameter. e) Steps b-d is kept in a continuous loop until the user defined key is pressed. f) The classifier, frame, memory storage & the window are destroyed. g) The (X, Y) coordinate of the image is plotted according to movement of face. h) The difference between face position and centre is calculated and sent to Arduino serially

Basically, Arduino will analyse a serial input for commands and set the servo positions accordingly. A command consists of two bytes: a servo ID and a servo position. If the Arduino receives a servo ID, then it waits for another serial byte and then assigns the received position value to the servo identified by the servo ID. The Arduino Servo library is used to easily control the pan and tilt servos. There's a character variable that will be used to keep track of the characters that come in on the Serial port. a) Library named servo.h is used in Arduino to control the servo motors, based on the data obtained by the OpenCV through COM port. b) Depending on the difference found in step8 the 2 servo motors are sent with appropriate controls for the pan-tilt movement of camera. c) Step b is kept in a continuous loop.

## THEORY OF OPENCV FACE RECOGNIZERS

Thanks to OpenCV, coding facial recognition is now easier than ever. There are three easy steps to computer coding facial recognition, which are similar to the steps that our brains use for recognizing faces. These steps are:

**Data Acquisition:** Gather face data (face images in this case) of the persons you want to identify.

**Train the Recognizer:** Feed that face data and respective names of each face to the recognizer so that it can learn.

**Recognition:** Feed new faces of that people and see if the face recognizer you just trained recognizes them.

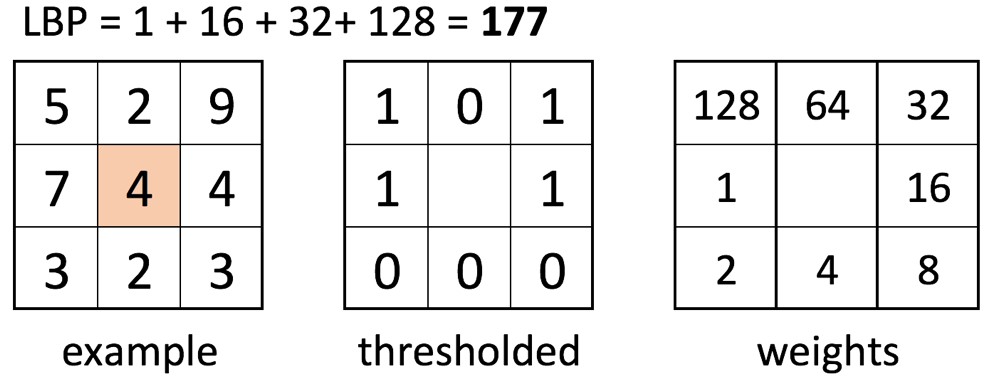
OpenCV has three built-in face recognizers and thanks to its clean coding, you can use any of them just by changing a single line of code. Here are the names of those face recognizers and their OpenCV calls:

* EigenFaces – cv2.face.createEigenFaceRecognizer()
* FisherFaces – cv2.face.createFisherFaceRecognizer()
* Local Binary Patterns Histograms (LBPH) – cv2.face.createLBPHFaceRecognizer()

In this project, we will focus on LBPH algorithm due to its higher accuracy.

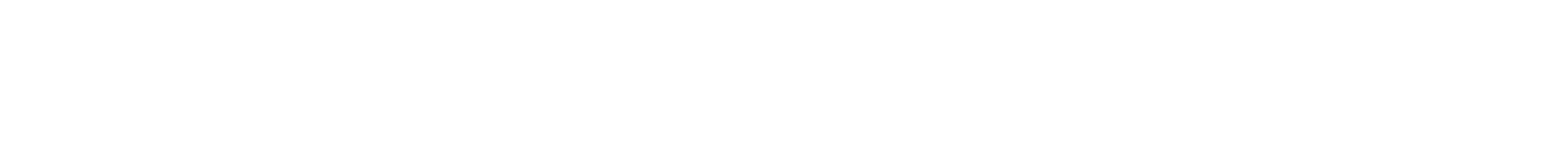
### Local Binary Patterns Histograms (LBPH)

This algorithm also requires grayscale pictures for processing the training. In contrast to the previous algorithms, this one is not a holistic approach. The aim of LBPH (Ahonen, Hadid, & Pietik, 2004) (Mäenpää, Pietikäinen, & Ojala, 2000) (Wagner, 2011) is to work by blocks of 3x3 pixels. The pixel in the center is compared to its neighbors. Each neighbor which is smaller than the pixel in the middle, the value 0 will be added to the thresholded square (figure 6) which is in charge to store the results, otherwise, a 1 will be added. The thresholded square and weights square are not present in the picture, they are only a representation to understand the process. When all the comparisons have been completed, each result will be multiplied by a weight. Each pixel has a weight to the power of two from 2x to 2y. Each pixel in the center of a 3x3 square has 8 neighbors. These eight pixels represent one byte which explains the reason of using these weights. The weights are affected in a circular order. It does not matter which weight is affected to which pixel, however, the weight of a pixel does not change. For example, if the pixel top left has a weight of 128, it will keep this weight for all the comparisons in the picture. Then, the sum of the weights is calculated and becomes the value of the pixel in the middle of the square. Figure 6 shows the results of the comparisons and the weight which is related to each pixel.



When this process has been completed for each part of the picture, the picture is divided into a certain number of regions. Then, a histogram is extracted from each region and all the histograms are concatenated. For recognizing a face, exactly the same process is performed, and the final histogram is compared to each final histogram in the training data. The label related to the closest histogram is the prediction of the algorithm. As for the hog detector, this algorithm is not sensitive to a variation of luminosity.

LBPH has been modified in different ways (what-when-how). One of them is called Extended LBPH. This extension is using a circular neighborhood which is composed of a radius and a number of sampling points. This approach allows a pixel to have more than eight neighbors. Depending on the radius (figure 7), the pixel in the middle could be compared to some pixels which are not next to it. Another extension is called uniform pattern (what-when-how). This extension takes into consideration the number of transition in the result byte. One transition is represented by a change in the byte from a 0 to a 1 or a1 to a 0. For example, 00000001 has one transition and, 00011000 has two transitions. It has been shown that patterns with a number of transitions from 0 to 2 are the most common (Ahonen & al., 2004). The patterns with two or fewer transitions usually have a specific signification how it can be seen in figure 7. All histograms with more than two transitions are regrouped together. This modification makes the vector representing the histograms smaller.



**Figure**

Visualization of the most c

ommon pattern on pictures with two or less tran-

sitions. Each pattern has a specific signification

(

what

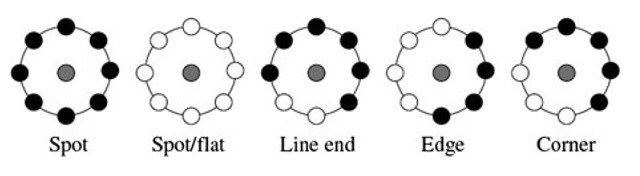
-

when

-

how, n.d.)

.



### 

### Code Dependencies

Install the following dependencies:

1. [OpenCV 3.2.0](http://opencv.org/releases.html)
2. [Python v3.5](https://www.python.org/downloads/)
3. [NumPy](http://www.numpy.org/) that makes computing in Python easy. It contains a powerful implementation of N-dimensional arrays which we will use for feeding data as input to OpenCV functions.
4. OS module
5. PIL (Python Image Library)

## REQUIRED MODULES

Import the following modules:

**cv2:** This is the OpenCV module for Python used for face detection and face recognition.

**Numpy:** This module converts Python lists to numpyarrays as OpenCV face recognizer needs them for the face recognition process.

**OS:** This module is used to manage system files and directories.

**PIL:** This library is used for images to be processed.

## LANGUAGES USED

### Python

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python‟s elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Website, [https://www.python.org/,](https://www.python.org/) and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well.

For a description of standard objects and modules, see [The Python Standard Library.](https://docs.python.org/3/library/index.html#library-index) [The Python Language Reference](https://docs.python.org/3/reference/index.html#reference-index) gives a more formal definition of the language. To write extensions in C or C++, read [Extending and Embedding the Python Interpreter](https://docs.python.org/3/extending/index.html#extending-index) and [Python/C API Reference Manual.](https://docs.python.org/3/c-api/index.html#c-api-index) There are also several books covering Python in depth.

This tutorial does not attempt to be comprehensive and cover every single feature, or even every commonly used feature. Instead, it introduces many of Python’s most noteworthy features, and will give you a good idea of the language’s flavour and style. After reading it, you will be able to read and write Python modules and programs, and you will be ready to learn more about the various Python library modules described in [The Python Standard Library.](https://docs.python.org/3/library/index.html#library-index)

### PACKAGES USED IN PYTHON

#### 1. OpenCV

OpenCV is the most popular library for computer vision. Originally written in C/C++, it now provides bindings for Python. OpenCV uses machine learning algorithms to search for faces within a picture. For something as complicated as a face, there isn’t one simple test that will tell you if it found a face or not. Instead, there are thousands of small patterns/features that must be matched. The algorithms break the task of identifying the face into thousands of smaller, bite-sized tasks, each of which is easy to solve. These tasks are also called [classifiers.](http://en.wikipedia.org/wiki/Statistical_classification)

For something like a face, you might have 6,000 or more classifiers, all of which must match for a face to be detected (within error limits, of course). But therein lies the problem: For face detection, the algorithm starts at the top left of a picture and moves down across small blocks of data, looking at each block, constantly asking, *“*Is this a face? Is this a face? Is this a face*?”* Since there are 6,000 or more tests per block, you might have millions of calculations to do, which will grind your computer to a halt.

To get around this, OpenCV uses [cascades.](http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html) What’s a cascade? The best answer can be found from the [dictionary:](http://dictionary.reference.com/browse/cascade) A waterfall or series of waterfalls Like a series of waterfalls, the OpenCV cascade breaks the problem of detecting faces into multiple stages. For each block, it does a very rough and quick test. If that passes, it does a slightly more detailed test, and so on. The algorithm may have 30-50 of these stages or cascades, and it will only detect a face if all stages pass. The advantage is that the majority of the pictures will return negative during the first few stages, which means the algorithm won’t waste time testing all 6,000 features on it. Instead of taking hours, face detection can now be done in real time. OpenCV was started at Intel in 1999 by Gary Bradsky, and the first release came out in 2000. Vadim Pisarevsky joined Gary Bradsky to manage Intel's Russian software OpenCV team. In 2005, OpenCV was used on Stanley, the vehicle that won the 2005 DARPA Grand Challenge. Later, its active development continued under the support of Willow Garage with Gary Bradsky and Vadim Pisarevsky leading the project. OpenCV now supports a multitude of algorithms related to Computer Vision and Machine Learning and is expanding day by day.

OpenCV supports a wide variety of programming languages such as C++, Python, Java, etc., and is available on different platforms including Windows, Linux, OS X, Android, and iOS. Interfaces for high-speed GPU operations based on CUDA and OpenCL are also under active development. OpenCV-Python is the Python API for OpenCV, combining the best qualities of the OpenCV C++ API and the Python language.

### OpenCV-Python

OpenCV-Python is a library of Python bindings designed to solve computer vision problems. Python is a general-purpose programming language started by **Guido van Rossum** that became very popular very quickly, mainly because of its simplicity and code readability. It enables the programmer to express ideas in fewer lines of code without reducing readability.

Compared to languages like C/C++, Python is slower. That said, Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. This gives us two advantages: first, the code is as fast as the original C/C++ code (since it is the actual C++ code working in background) and second, it easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.

OpenCV-Python makes use of **Numpy**, which is a highly optimized library for numerical operations with a MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpy arrays. This also makes it easier to integrate with other libraries that use Numpy such as SciPy and Matplotlib.

### Cascades in Practice

Though the theory may sound complicated, in practice it is quite easy. The cascades themselves are just a bunch of XML files that contain OpenCV data used to detect objects. You initialize your code with the cascade you want, and then it does the work for you.

Since face detection is such a common case, OpenCV comes with a number of built-in cascades for detecting everything from faces to eyes to hands and legs. There are even cascades for non-human things. For example, if you run a banana shop and want to track people stealing bananas, [this guy](http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html) has built one for that!

### History

Officially launched in 1999, the OpenCV project was initially an [Intel Research](https://en.wikipedia.org/wiki/Intel_Research_Lablets) initiative to advance [CPU-](https://en.wikipedia.org/wiki/Central_processing_unit)intensive applications, part of a series of projects including [real-time](https://en.wikipedia.org/wiki/Real-time_computing) [ray tracing](https://en.wikipedia.org/wiki/Ray_tracing_(graphics)) and [3D display](https://en.wikipedia.org/wiki/3D_Display) walls. The main contributors to the project included a number of optimization experts in Intel Russia, as well as Intel’s Performance Library Team. In the early days of OpenCV, the goals of the project were described as:

* Advance vision research by providing not only open but also [optimized code](https://en.wikipedia.org/wiki/Code_optimization) for basic vision infrastructure. No more [reinventing the wheel.](https://en.wikipedia.org/wiki/Reinventing_the_wheel)
* Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
* Advance vision-based commercial applications by making [portable,](https://en.wikipedia.org/wiki/Portability_(computer_science)) performance-optimized code available for free – with a license that did not require code to be open or free itself.

The first alpha version of OpenCV was released to the public at the [IEEE Conference on Computer Vision and Pattern Recognition](https://en.wikipedia.org/wiki/Conference_on_Computer_Vision_and_Pattern_Recognition) in 2000, and five betas were released between 2001 and 2005. The first 1.0 version was released in 2006. A version 1.1 "pre-release" was released in October 2008.

The second major release of the OpenCV was in October 2009. OpenCV 2 includes major changes to the [C++](https://en.wikipedia.org/wiki/C%2B%2B) interface, aiming at easier, more type-safe patterns, new functions, and better implementations for existing ones in terms of performance (especially on multi-core systems). Official releases now occur every six monthsand development is now done by an independent Russian team supported by commercial corporations. In August 2012, support for OpenCV was taken over by a non-profit foundation OpenCV.org, which maintains a developer and user site. On May 2016, Intel signed an agreement to acquire it seese, the leading developer of OpenCV.

### Installing OpenCV

First, you need to find the correct setup file for [your operating system](http://opencv.org/releases.html) I found that installing OpenCV was the hardest part of the task. If you get strange unexplainable errors, it could be due to library clashes, 32/64 bit differences, etc. I found it easiest to just use a Linux virtual machine and install OpenCV from scratch.

* If you have previous/other version of OpenCV installed (e.g. cv2 module in the root of Python’s site-packages), remove it before installation to avoid conflicts.
* To further avoid conflicts and to make development easier, Python’s [virtual environments](https://docs.python.org/3/library/venv.html) are highly recommended for development purposes.
* 1.If you have an existing opencv-contrib-python installation, run pip uninstall opencv-contrib-python
* 2.Install this package:
* pip install opencv-python

import cv2

The package contains haar-cascade files. cv2.data.haarcascades can be used as a shortcut to the data folder. For example:

cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade\_frontalface\_default.xml")

#### 2. Numpy

NumPy is a library for the [Python programming language,](https://en.wikipedia.org/wiki/Python_(programming_language)) adding support for large, multidimensional [arrays](https://en.wikipedia.org/wiki/Array_data_structure) and [matrices,](https://en.wikipedia.org/wiki/Matrix_(math)) along with a large collection of [high-level](https://en.wikipedia.org/wiki/High-level_programming_language) [mathematical](https://en.wikipedia.org/wiki/Mathematics) [functions](https://en.wikipedia.org/wiki/Function_(mathematics)) to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by [Jim Hugunin](https://en.wikipedia.org/wiki/Jim_Hugunin) with contributions from several other developers. In 2005, [Travis Oliphant](https://en.wikipedia.org/wiki/Travis_Oliphant) created Numpy by incorporating features of the competing Num array into Numeric, with extensive modifications. NumPy is [open-source software](https://en.wikipedia.org/wiki/Open-source_software) and has many contributors.

### History

The Python programming language was not initially designed for numerical computing, but attracted the attention of the scientific and engineering community early on, so that a special interest group called matrix-sig was founded in 1995 with the aim of defining an array computing package. Among its members was Python designer and maintainer [Guido van Rossum,](https://en.wikipedia.org/wiki/Guido_van_Rossum) who implemented extensions to [Python's syntax](https://en.wikipedia.org/wiki/Python_syntax_and_semantics) (in particular the indexing syntax) to make array computing easier.

An implementation of a matrix package was completed by Jim Fulton, then generalized byJim Huguni[n](https://en.wikipedia.org/wiki/Jim_Hugunin) become Numeric, also variously called Numerical Python extensions or NumPy. Hugunin, a graduate student at [MIT,](https://en.wikipedia.org/wiki/Massachusetts_Institute_of_Technology) joined [CNRI](https://en.wikipedia.org/wiki/CNRI) to work on [JPython](https://en.wikipedia.org/wiki/Jython) in 1997leaving Paul Dubois of [LLNL](https://en.wikipedia.org/wiki/Lawrence_Livermore_National_Laboratory) to take over as maintainer. Other early contributors include David Ascher, Konrad Hinsen and [Travis Oliphant.](https://en.wikipedia.org/wiki/Travis_Oliphant)

A new package called Num array was written as a more flexible replacement for Numeric.[[6]](https://en.wikipedia.org/wiki/NumPy#cite_note-cise-6) Like Numeric, it is now deprecated. Num array had faster operations for large arrays, but was slower than Numeric on small ones. so for a time both packages were used for different use cases. The last version of Numeric v24.2 was released on 11 November 2005 and num array v1.5.2 was released on 24 August 2006.

There was a desire to get Numeric into the Python standard library, but Guido van Rossum decided that the code was not maintainable in its state then. In early 2005, NumPy developer [Travis Oliphant](https://en.wikipedia.org/wiki/Travis_Oliphant) wanted to unify the community around a single array package and ported Num array's features to Numeric, releasing the result as NumPy 1.0 in 2006. This new project was part of [SciPy.](https://en.wikipedia.org/wiki/SciPy) To avoid installing the large SciPy package just to get an array object, this new package was separated and called NumPy. Support for Python 3 was added in 2011 with NumPy version 1.5.0. In 2011, [PyPy](https://en.wikipedia.org/wiki/PyPy) started development on an implementation of the NumPy API for PyPy. It is not yet fully compatible with NumPy.

## 6.2 ALGORITHMS USED FOR IMPLEMENTATION

[OpenCV (Open Source Computer Vision)](http://opencv.org/) is a popular computer vision library started by [Intel](http://www.intel.com/) in 1999. The cross-platform library sets its focus on real-time image processing and includes patentfree implementations of the latest computer vision algorithms. In 2008 [Willow Garage](http://www.willowgarage.com/) took over support and OpenCV 2.3.1 now comes with a programming interface to C,

C++, [Python](http://www.python.org/) and [Android.](http://www.android.com/) OpenCV is released under a BSD license so it is used in academic projects and commercial products alike.

OpenCV 2.4 now comes with the very new [Face Recognizer](https://www.docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_api.html#FaceRecognizer : public Algorithm) class for face recognition, so you can start experimenting with face recognition right away. This document is the guide I‟ve wished for, when I was working myself into face recognition. It shows you how to perform face recognition with [Face Recognizer](https://www.docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_api.html#FaceRecognizer : public Algorithm) in OpenCV (with full source code listings) and gives you an introduction into the algorithms behind. I’ll also show how to create the visualizations you can find in many publications, because a lot of people asked for.

The currently available algorithms are:

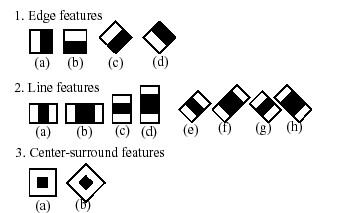
* Eigen faces (see [create EigenFaceRecognizer())](https://www.docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_api.html#Ptr<FaceRecognizer> createEigenFaceRecognizer(int num_components , double threshold))
* Fisher faces (see [create FisherFaceRecognizer(**)**)](https://www.docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_api.html#Ptr<FaceRecognizer> createFisherFaceRecognizer(int num_components , double threshold))
* Local Binary Patterns Histograms (see [create LBPHFaceRecognizer())](https://www.docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_api.html#Ptr<FaceRecognizer> createLBPHFaceRecognizer(int radius, int neighbors, int grid_x, int grid_y, double threshold))

If you have built OpenCV with the samples turned on, chances are good you have them compiled already! Although it might be interesting for very advanced users, I’ve decided to leave the implementation details out as I am afraid that they confuse new users. In this project, we are going to use LBPH algorithm.

### 6.2.1 HAAR CASCADE

The Haar Cascade is a machine learning based approach, an algorithm created by Paul Viola and Michael Jones; which (as mentioned before) are trained from many positive images (with faces) and negatives images (without faces).

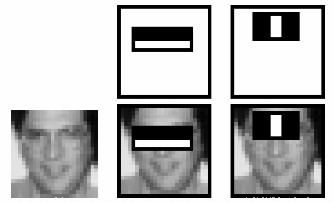
It starts by extracting Haar features from each image as shown by the windows below:



#### Method of extracting features

Each window is placed on the picture to calculate a single feature. This feature is a single value obtained by subtracting the sum of pixels under the white part of the window from the sum of the pixels under the black part of the window.

Now, all possible sizes of each window are placed on all possible locations of each image to calculate plenty of features.



#### Extracting Haar features from image

For example, in above image, we are extracting two features. The first one focuses on the property that the region of the eyes is often darker than the area of the nose and cheeks. The second feature relies on the property that the eyes are darker than the bridge of the nose.

But among all these features calculated, most of them are irrelevant. For example, when used on the cheek, the windows become irrelevant because none of these areas are darker or lighter than other regions on the cheeks, all sectors here are the same. So, we promptly discard irrelevant features and keep only those relevant with a fancy technique called AdaBoost.AdaBoost is a training process for face detection, which selects only those features known to improve the classification (face/non-face) accuracy of our classifier.

In the end, the algorithm considers the fact that generally: most of the region in an image is a non-face region. Considering this, it’s a better idea to have a simple method to check if a window is a non-face region, and if it's not, discard it right away and don’t process it again. So, we can focus mostly on the area where a face is.

## 6.3 CODING

**6.3.1. Front-end:**

**from tkinter import \***

**# Fucntion will run by pressing datasets button..**

**from face\_detect import detect**

**from face\_datasets import face\_datasets**

**from training import train**

**import cv2, time**

**window from face\_recognition import recognize**

**window= Tk()**

**window.wm\_title("Face Recognition System")**

**l1 = Label(window, text = "Face Recognition System", font = "100")**

**l1.grid(row = 0, column = 0, rowspan = 2, columnspan = 2)**

**b1 = Button(window, text="Detect", width=15, height=5, font='50', command=detect)**

**#b1.grid(row = 6, column = 1)**

**b1.grid(row = 4, column = 0)**

**b2 = Button(window, text= "Datasets", width = 15, height = 5, font = "50", command = face\_datasets)**

**#b2.grid(row = 4, column = 0)**

**b2.grid(row = 4, column = 1)**

**b1 = Button(window, text = "Train", width = 15, height = 5, font = "50", command = train)**

**#b1.grid(row = 4, column = 1)**

**b1.grid(row = 6, column = 0)**

**b2 = Button(window, text= "Recognize", width = 15, height = 5, font = "50", command = recognize)**

**#b2.grid(row = 6, column = 0)**

**b2.grid(row = 6, column = 1)**

**b3 = Button(window, text = "Close", width = 15, height = 5, font = "50",  command = window.destroy)**

**b3.grid(row = 8, column = 0)**

**window.mainloop()**

**6.3.2. Data Acquisition:**

**# Import OpenCV2 for image processing**

**import cv2**

**import os**

**#from getId import face\_id**

**def face\_datasets():**

**def assure\_path\_exists(path):**

**dir = os.path.dirname(path)**

**if not os.path.exists(dir):**

**os.makedirs(dir)**

**# Start capturing video**

**vid\_cam = cv2.VideoCapture(0)**

**# Detect object in video stream using Haarcascade Frontal Face**

**face\_detector = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')**

**# For each person, one face id**

**# face\_id = 1**

**face\_id = int(input("Enter an Id for images"))**

**# Initialize sample face image**

**count = 0**

**assure\_path\_exists("dataset/")**

**# Start looping**

**while(True):**

**# Capture video frame**

**\_, image\_frame = vid\_cam.read()**

**# Convert frame to grayscale**

**gray = cv2.cvtColor(image\_frame, cv2.COLOR\_BGR2GRAY)**

**# Detect frames of different sizes, list of faces rectangles**

**faces = face\_detector.detectMultiScale(gray, 1.3, 5)**

**# Loops for each faces**

**for (x,y,w,h) in faces:**

**# Crop the image frame into rectangle**

**cv2.rectangle(image\_frame, (x,y), (x+w,y+h), (255,0,0), 2)**

**# Increment sample face image**

**count += 1**

**# Save the captured image into the datasets folder**

**cv2.imwrite("dataset/User." + str(face\_id) + '.' + str(count) + ".jpg", gray[y:y+h,x:x+w])**

**# Display the video frame, with bounded rectangle on the person's face**

**cv2.imshow('frame', image\_frame)**

**# To stop taking video, press 'q' for at least 100ms**

**if cv2.waitKey(100) & 0xFF == ord('q'):**

**break**

**# If image taken reach 100, stop taking video**

**elif count>100:**

**break**

**# Stop video**

**vid\_cam.release()**

**# Close all started windows**

**cv2.destroyAllWindows()**

**6.6.3. Face Detection:**

**import numpy as np**

**import cv2**

**def detect():**

**# multiple cascades: https://github.com/Itseez/opencv/tree/master/data/haarcascades**

**faceCascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')**

**cap = cv2.VideoCapture(0)**

**cap.set(3,640) # set Width**

**cap.set(4,480) # set Height**

**while True:**

**ret, img = cap.read()**

**img = cv2.flip(img, 1)**

**gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)**

**faces = faceCascade.detectMultiScale(gray, 1.3, 5)**

**for (x,y,w,h) in faces:**

**cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)**

**roi\_gray = gray[y:y+h, x:x+w]**

**roi\_color = img[y:y+h, x:x+w]**

**cv2.imshow('video',img)**

**k = cv2.waitKey(30) & 0xff**

**if k == ord('q'):**

**break**

**cap.release()**

**cv2.destroyAllWindows()**

**6.6.4. Training:**

**# Import OpenCV2 for image processing**

**# Import os for file path**

**import cv2, os**

**# Import numpy for matrix calculation**

**import numpy as np**

**# Import Python Image Library (PIL)**

**from PIL import Image**

**import os**

**def train():**

**def assure\_path\_exists(path):**

**dir = os.path.dirname(path)**

**if not os.path.exists(dir):**

**os.makedirs(dir)**

**# Create Local Binary Patterns Histograms for face recognization**

**recognizer = cv2.face.LBPHFaceRecognizer\_create()**

**# Using prebuilt frontal face training model, for face detection**

**detector = cv2.CascadeClassifier("haarcascade\_frontalface\_default.xml");**

**# Create method to get the images and label data**

**def getImagesAndLabels(path):**

**# Get all file path**

**imagePaths = [os.path.join(path,f) for f in os.listdir(path)]**

**# Initialize empty face sample**

**faceSamples=[]**

**# Initialize empty id**

**ids = []**

**# Loop all the file path**

**for imagePath in imagePaths:**

**# Get the image and convert it to grayscale**

**PIL\_img = Image.open(imagePath).convert('L')**

**# PIL image to numpy array**

**img\_numpy = np.array(PIL\_img,'uint8')**

**# Get the image id**

**id = int(os.path.split(imagePath)[-1].split(".")[1])**

**# Get the face from the training images**

**faces = detector.detectMultiScale(img\_numpy)**

**# Loop for each face, append to their respective ID**

**for (x,y,w,h) in faces:**

**# Add the image to face samples**

**faceSamples.append(img\_numpy[y:y+h,x:x+w])**

**# Add the ID to IDs**

**ids.append(id)**

**# Pass the face array and IDs array**

**return faceSamples,ids**

**# Get the faces and IDs**

**faces,ids = getImagesAndLabels('dataset')**

**# Train the model using the faces and IDs**

**recognizer.train(faces, np.array(ids))**

**# Save the model into trainer.yml**

**assure\_path\_exists('trainer/')**

**recognizer.save('trainer/trainer.yml')**

**6.6.5. Face Recognition:**

**# Import OpenCV2 for image processing**

**import cv2**

**# Import numpy for matrices calculations**

**import numpy as np**

**import os**

**def recognize():**

**def assure\_path\_exists(path):**

**dir = os.path.dirname(path)**

**if not os.path.exists(dir):**

**os.makedirs(dir)**

**# Create Local Binary Patterns Histograms for face recognization**

**recognizer = cv2.face.LBPHFaceRecognizer\_create()**

**assure\_path\_exists("trainer/")**

**# Load the trained mode**

**recognizer.read('trainer/trainer.yml')**

**# Load prebuilt model for Frontal Face**

**cascadePath = "haarcascade\_frontalface\_default.xml"**

**# Create classifier from prebuilt model**

**faceCascade = cv2.CascadeClassifier(cascadePath);**

**# Set the font style**

**font = cv2.FONT\_HERSHEY\_SIMPLEX**

**# Initialize and start the video frame capture**

**cam = cv2.VideoCapture(0)**

**# Loop**

**while True:**

**# Read the video frame**

**ret, im =cam.read()**

**# Convert the captured frame into grayscale**

**gray = cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY)**

**# Get all face from the video frame**

**faces = faceCascade.detectMultiScale(gray, 1.2,5)**

**# For each face in faces**

**for(x,y,w,h) in faces:**

**# Create rectangle around the face**

**cv2.rectangle(im, (x-20,y-20), (x+w+20,y+h+20), (0,255,0), 4)**

**# Recognize the face belongs to which ID**

**Id, confidence = recognizer.predict(gray[y:y+h,x:x+w])**

**# Check the ID if exist**

**ids = ['Unknown', 'Jamshed', 'Jawed']**

**for i in range(len(ids)):**

**if(Id == i):**

**Id = "{0} {1:.2f}%".format(ids[i], round(100 - confidence, 2))**

**# Put text describe who is in the picture**

**cv2.rectangle(im, (x-22,y-90), (x+w+22, y-22), (0,255,0), -1)**

**cv2.putText(im, str(Id), (x,y-40), font, 1, (255,255,255), 3)**

**# Display the video frame with the bounded rectangle**

**cv2.imshow('im',im)**

**# If 'q' is pressed, close program**

**if cv2.waitKey(10) & 0xFF == ord('q'):**

**break**

**# Stop the camera**

**cam.release()**

**# Close all windows**

**cv2.destroyAllWindows()**

**CHAPTER-7 TESTING**

## 7.1 INTRODUCTION TO TESTING

Testing is a process, which reveals errors in the program. It is the major quality measure employed during software development. During software development. During testing, the program is executed with a set of test cases and the output of the program for the test cases is evaluated to determine if the program is performing as it is expected to perform.

## 7.2 TESTING STRATEGIES

In order to make sure that the system does not have errors, the different levels of testing strategies that are applied at differing phases of software development are:

**7.2.1 UNIT TESTING** Unit Testing is done on individual modules as they are completed and become executable. It is confined only to the designer's requirements. Each module can be tested using the following two strategies:

### 1. Black Box Testing

In this strategy some test cases are generated as input conditions that fulsly execute all functional requirements for the program. This testing has been uses to find errors in the following categories:

* Incorrect or missing functions
* Interface errors
* Performance errors
* Initialization and termination errors.

### 2. White Box Testing

In this the test cases are generated on the logic of each module by drawing flow graphs of that module and logical decisions are tested on all the cases. It has been uses to generate the test cases in the following cases

* Guarantee that all independent paths have been Executed.
* Execute all logical decisions on their true and false Sides.
* Execute all loops at their boundaries and within their operational bounds 33

### 7.2.2 INTEGRATION TESTING

Integration testing ensures that software and subsystems work together a whole. It tests the interface of all the modules to make sure that the modules behave properly when integrated together.Integration tessting (sometimes called integration and testing, abbreviated I&T) is the phase in [software testing](https://en.wikipedia.org/wiki/Software_testing) in which individual software modules are combined and tested as a group. It occurs after [unit testing](https://en.wikipedia.org/wiki/Unit_testing) and before [validation testing.](https://en.wikipedia.org/wiki/Software_verification_and_validation) Integration testing takes as its input [modules](https://en.wikipedia.org/wiki/Module_(programming)) that have been unit tested, groups them in larger aggregates, applies tests defined in an integration [test plan](https://en.wikipedia.org/wiki/Test_plan) to those aggregates, and delivers as its output the integrated system ready for [system testing.](https://en.wikipedia.org/wiki/System_testing)

### 7.2.3 SYSTEM TESTING

It Involves in house testing of the entire system before delivery to the user. It's aim is to satisfy the user the system meets all requirements of the client's specifications.System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified [requirements.](https://en.wikipedia.org/wiki/Requirements) System testing falls within the scope of [black-box testing,](https://en.wikipedia.org/wiki/Black-box_testing) and as such, should require no knowledge of the inner design of the code or logic.

As a rule, system testing takes, as its input, all of the "integrated" software components that have passed [integration testing](https://en.wikipedia.org/wiki/Integration_testing) and also the software system itself integrated with any applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called assemblages) or between any of the assemblages and the hardware. System testing is a more limited type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

### 7.2.4 ACCEPTANCE TESTING

It is a pre-delivery testing in which entire system is tested at client's site on real world data to find errors. Acceptance testing is a test conducted to determine if the requirements of a [specification](https://en.wikipedia.org/wiki/Specification) or [contract](https://en.wikipedia.org/wiki/Contract) are met. It may involve [chemical tests,](https://en.wikipedia.org/wiki/Chemical_test) [physical tests,](https://en.wikipedia.org/wiki/Physical_test) or [performance tests.](https://en.wikipedia.org/wiki/Performance_test_(assessment)) In [systems engineering](https://en.wikipedia.org/wiki/Systems_engineering) it may involve [black-box testing](https://en.wikipedia.org/wiki/Black-box_testing) performed on a [system](https://en.wikipedia.org/wiki/System) (for example: a piece of [software,](https://en.wikipedia.org/wiki/Software_system) lots of manufactured mechanical parts, or batches of chemical products) prior to its delivery.

In [software testing](https://en.wikipedia.org/wiki/Software_testing) the [ISTQB](https://en.wikipedia.org/wiki/International_Software_Testing_Qualifications_Board) defines acceptance as: formal testing with respect to user needs, requirements, and business processes conducted to determine whether a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system. Acceptance testing is also known as user acceptance testing (UAT), end-user testing, [operational acceptance testing](https://en.wikipedia.org/wiki/Operational_acceptance_testing) (OAT) or field (acceptance) testing.

Testing Can Be Done In Two Ways

* Bottom up approach
* Top down approach

**Bottom Up Approach:** Testing can be performed starting from smallest and lowest level modules and proceeding one at a time. For each module in bottom up testing a short program executes the module and provides the needed data so that the module is asked to perform the way it will when embedded within the larger system. When bottom level modules are tested attention turns to those on the next level that use the lower level ones they are tested individually and then linked with the previously examined lower level modules.

**Top Down Approach:** This type of testing starts from upper level modules. Since the detailed activities usually performed in the lower level routines are not provided stubs are written. A stub is a module shell called by upper level module and that when reached properly will return a message to the calling module indicating that proper interaction occurred. No attempt is made to verify the correctness of the lower level module.

## 7.3 DESIGN FOR TEST CASES AND SCENARIOS

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| **TEST**  **CASE**  **NUMBER** | TEST CASE | INPUT | **EXPECTED OUTPUT** | **ACTUAL**  **OUTPUT** | **RESULT** |
|  | Face  Detection | Live  Images | Detect Human Faces from Input Images | Detects  Human  Faces from  Input Images | Pass |
| 01 |

**Table 7.1 Test Case for Face Detection**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TEST**  **CASE**  **NUMBER** | TEST CASE | INPUT | **EXPECTED OUTPUT** | **ACTUAL**  **OUTPUT** | **RESULT** |
|  | Normalization | Human  Faces | Normalize facial landmarks | Normalize  facial landmarks | pass |
| 01 |

**Table 7.2 Test Case for Normalization**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TEST**  **CASE**  **NUMBER** | TEST CASE | INPUT | **EXPECTED OUTPUT** | **ACTUAL**  **OUTPUT** | **RESULT** |
|  | Feature  Extraction | Human  Faces | Extract Facial  Features | Extracts  Facial  Features | pass |
| 01 |

**Table 7.3 Test Case for Feature Extraction**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TEST**  **CASE**  **NUMBER** | TEST CASE | INPUT | **EXPECTED OUTPUT** | **ACTUAL**  **OUTPUT** | **RESULT** |
|  | Face  Recognition | Live  Images | Identify Human Faces and Display rectangular Frames on Faces | Identify Faces and Display rectangular  Frames on  Faces | pass |
| 01 |

**Table 7.4 Test Case for Face Recognition**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TEST**  **CASE**  **NUMBER** | TEST CASE | INPUT | **EXPECTED OUTPUT** | **ACTUAL**  **OUTPUT** | **RESULT** |
|  | Object  Detection | Objects | Identify Objects and  Display Rectangular  Frames on Objects | Objects Are  Not Detected | Fail |
| 01 |

**Table 7.5 Test Case for Object Detection**

## 7.4 VALIDATION

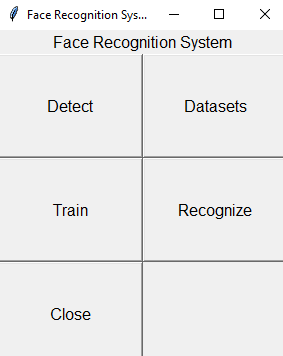
The system has been tested and implemented successfully and thus ensured that all the requirements as listed in the software requirements specification are completely fulfilled. In case of erroneous input corresponding error messages are displayed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No** | **Input** | **Expected**  **Result** | **Actual**  **Result** | **Pass/Fail** |
| 1 | Human Faces | Recognise  Faces | Recognise  Faces | Pass |
| 2 | Objects | Recognise objects | Objects are not  Recognised | Fail |
| 3 | Animal Faces | Recognise  Faces | Animal Faces are Not Recognised | Fail |
| 4 | Human Faces in  Phone | Recognise  Faces | Recognise  Faces | Pass |
| 5 | Human Faces in  Photo | Recognise  Faces | Recognise  Faces | Pass |

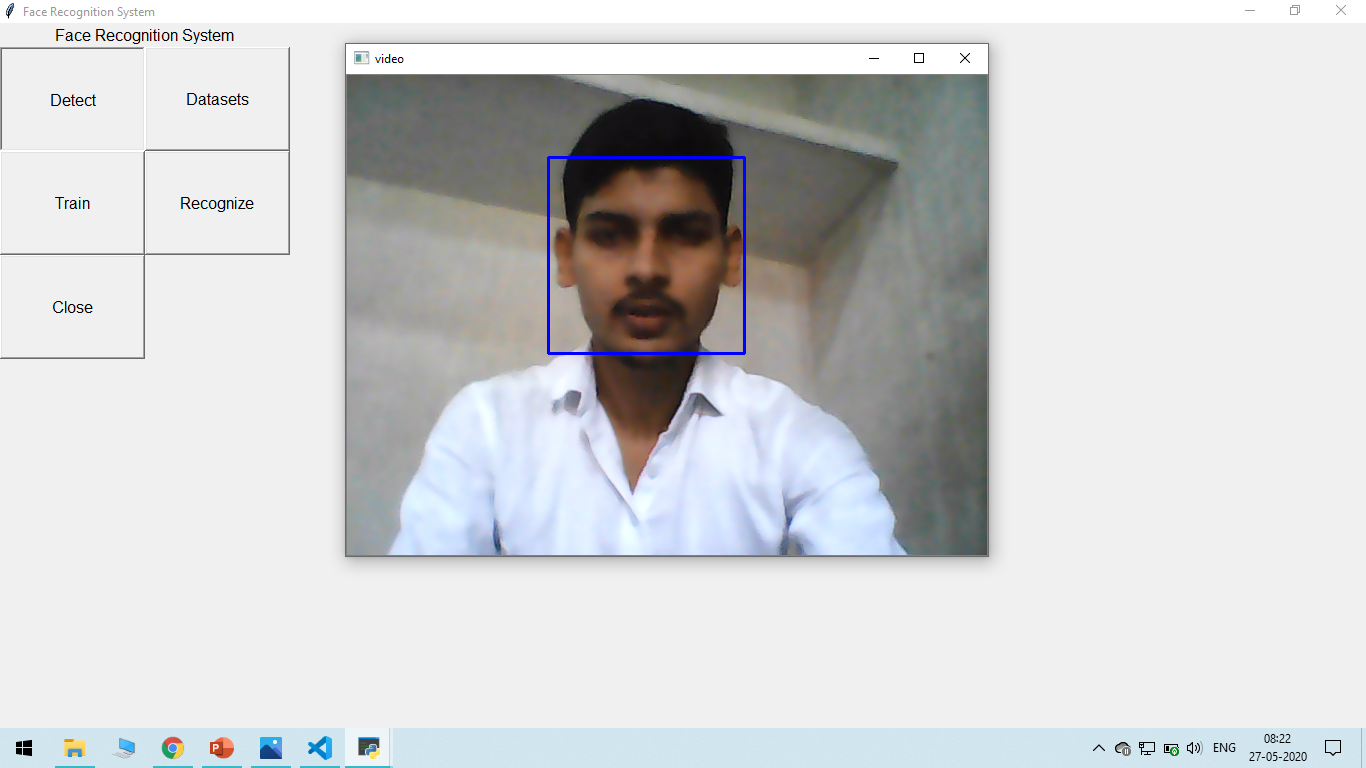
**Table 7.6 Validation**

**CHAPTER-8 OUTPUT**

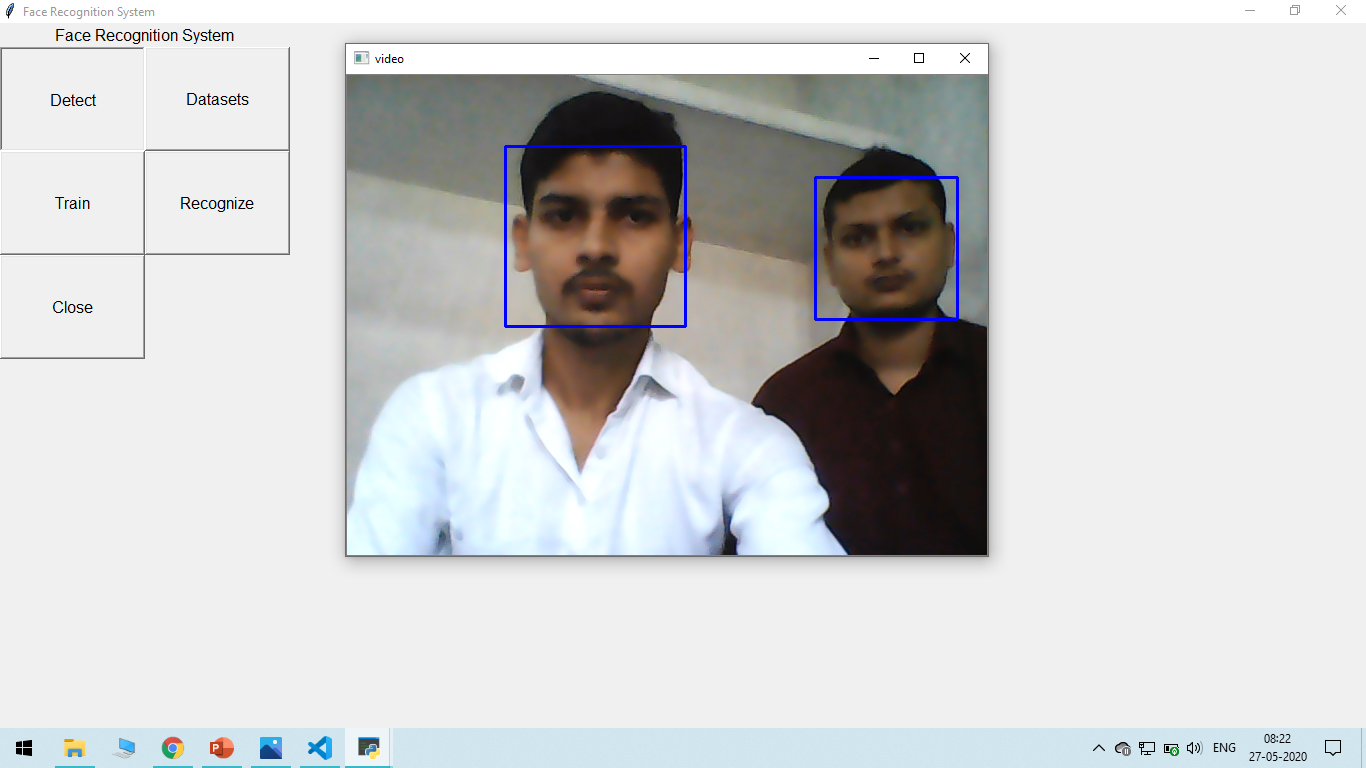
1. We have completed our implementation of our project. Now, its time to verify whether our project works according to our requirements and expectations.
2. When we run our project file frontview.py. we get a graphical user interface to interact it. On this GUI few buttons are available as if Detect, Datasets, Train, Recognize and Close.
3. When we start clicking on each button one by one it works accordingly.
4. When we click on Detect button the system start detecting the faces through the webcam. If no faces are available in front of the camera, it will not detect anything but once a face will come around it, it detects that face.
5. We have also one Datasets button which ask an id from the user and start capturing faces from the images and store it to a dataset directory when id is given.
6. After that we press the Train button to train our model so that it can identify the faces efficiently.
7. Finally, we press Recognize button to verify the output if a person is registered along with his/her face then our model will predict his/her name on top of his head.
8. Our model will detect and identify/recognize multiple faces at the same time.



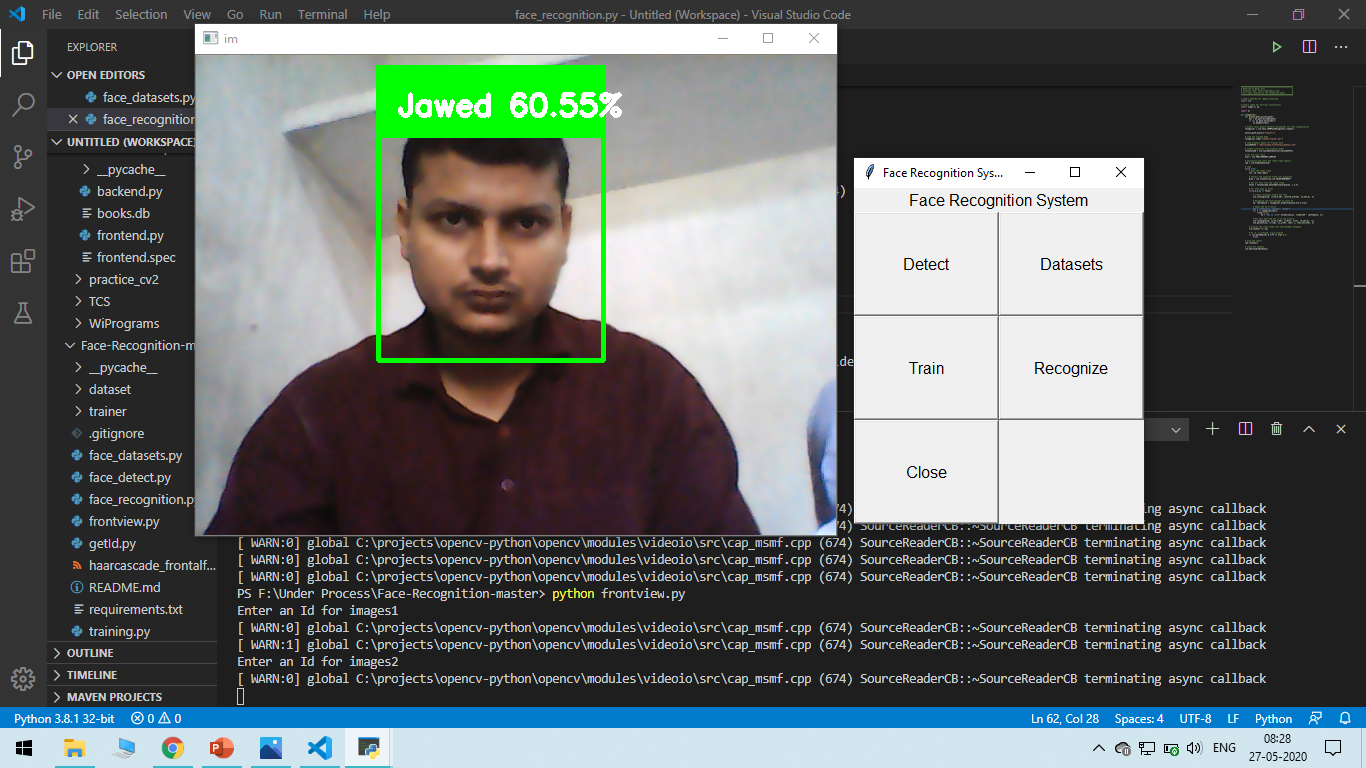
## Fig 8.1 Execution

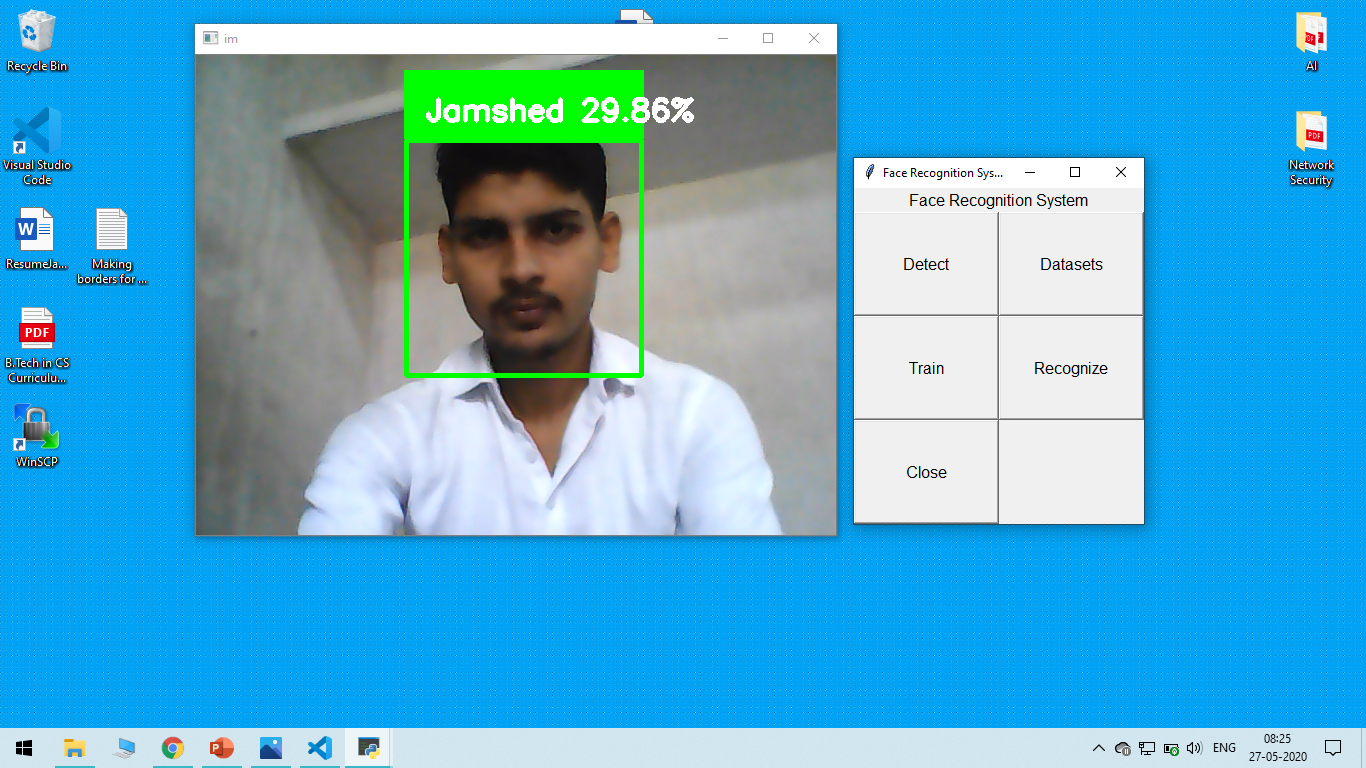


## Fig 8.2 Face Detection from live video

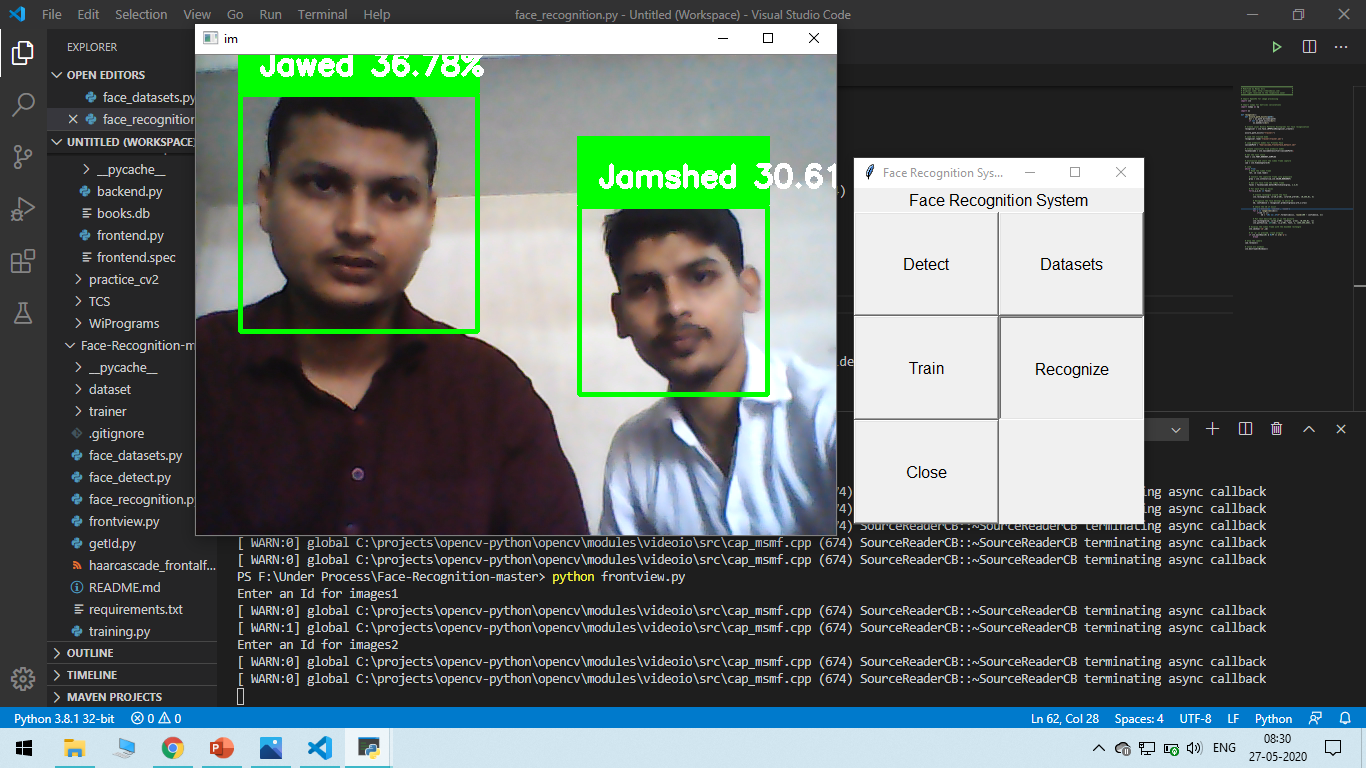


## Fig 8.3 Multiple Face Detection from Live Video





**Fig 8.4 Face Recognition through webcam**



**Fig 8.5 Multiple Face Recognition through webcam**



## Fig 8.6 Object was not recognised



## Fig 8.7 Object was not recognised

**CHAPTER-9 CONCLUSION**

## 9.1 PROJECT CONCLUSION

As the fastest growing technology today, facial recognition will have many users in the near future. While facial recognition can be used to protect your privacy, it can as easily be used to invade your privacy by taking your picture when your entirely unaware of camera.

## 9.2 FUTURE ENHANCEMENT

The future work can be done on server-client application which could hold more number of images in both sides. If the server system fails the same image database can be recovered from the client side. Any alternative algorithm can be used in face recognition to identify the variations of face still more clearly.

**CHAPTER-10 REFERENCES**

|  |  |
| --- | --- |
| [1] | R. C. G. &. R. E. Woods, Digital Image Processing, London: Pearson; 3rd Edition, 2007. |
| [2] | A. E. &. C. Gurel, “Design of a Face Recognition System,” in *The 15th International Conference on Machine Design and Production*, Pamukkale, 2012. |
| [3] | T. A. E.-H. a. Y. K. M. Abdelmgeid A. Ali, “An Accurate System for Face Detection and Recognition,” *Journal of Advances in Mathematics and Computer Science,* p. 19, 2019. |
| [4] | J. Z. S. Y. Yongzhong LU, “A SURVEY OF FACE DETECTION, EXTRACTION AND RECOGNITION,” Ladislav Hluchy, Wuhan, 2003. |
| [5] | M. V. R. S. a. K. S. S.K.Singh, *A COMPARATIVE STUDY OF VARIOUS FACE RECOGNITION ALGORITHMS,* Varanasi: International Workshop on Computer Architectures for Machine Perception (CAMP), 2003. |
| [6] | W. G. a. I. Intel Corporation, “OpenCV,” Intel Corporation, Willow Garage and Itseez, 03 April 2020. [Online]. Available: opencv.org. |
| [7] | P. V. a. S. Ashish, “ATTENDANCE SYSTEM USING MULTI-FACE RECOGNITION,” *International Journal of Pure and Applied Mathematics,* vol. 118, no. 22, p. 4, 2018. |
| [8] | S. Kumar, “Face Recognition System,” Rajib Mall, Kharagpur, 2019. |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |