FACE RECOGNITION SMART DOOR USING PI

A PROJECT REPORT

Submitted by

Rajal Shethiya (15EL030)

Abhishek Dand (15EL042)

Parth Lad (15EL052)

In fulfillment for the award of the degree

Of

BACHELOR OF TECHNOLOGY

in

ELECTRONICS DEPARTMENT

BIRLA VISHVKARMA MAHAVIDYALAYA (BVM)-VVN

Gujarat Technological University, Ahmedabad

BIRLA VISHVKARMA MAHAVIDYALAYA (BVM)-VVN

ELECTRONICS DEPARTMENT

2018-19

CERTIFICATE

Date:

This is to certify that the dissertation entitled "FACE RECOGNITION SMART DOOR USING PI" has been carried out by Rajal Shethiya(15EL030), Abhishek Dand(15EL042) and Parth Lad(15EL052) under my guidance in fulfillment of the degree of Bachelor of Technology in ELECTRONICS ENGINEERING(8th Semester) of Gujarat Technological University, Ahmedabad during the academic year 2018-19.

Guide: Dr Mehfuza Holia

H.O.D: Dr. Tanmay Pawar

ACKNOWLEDGEMENT

The successful completion of our project is a result of a strong support of many people who are directly or indirectly associated with the project. We would like to express our gratitude towards all these people.

We extend our sincere thanks to our Principal Dr. Indrajit Patel for his motivation and support to accomplish the project.

We would like to thank our Head of the Department Dr. T. D. Pawar, for his providing us with necessary infrastructure and motivating us.

We are thankful to our project coordinator Dr. D. L. Vala, for his support.

We are greatly thankful to our project guide Dr. Mehfuza Holia, for her immense support and valuable guidance.

We would like to thank our lab assistants and non-teaching staffs for their support.

We are thankful to our families for their constant moral support and motivation.

Lastly we would like express an earnest thanks to all those who have helped and supported us in any way and helped us in our project.

ABSTRACT

In today's busy world, time is very crucial. A door that won't require a key can be very helpful and time saving. We have achieved this through digital image processing algorithms using Raspberry Pi.

In order to achieve facial recognition we have used combination of various machine learning algorithm. HoG is used to detect face in an image. Face Landmark estimation in used to obtain 68 face landmarks and used to generate 128d unique embeddings for an image and this embeddings are used to compare with the embeddings in our database. We have used Raspberry Pi board as it is a computer in itself and allows us to work on an OS and supports Python language which is very convenient to use and understand. Nothing nowadays is happening without the use of IoT. We have incorporated the AWS cloud service in our project where the captured images will be stored and can be accessed from anywhere. This helps in increased security and safety of the home

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

Chapter 1 INTRODUCTION

1.1 PROBLEM SUMMARY

We face problems and challenges almost every day in our life. Some of these problems can be easily solved while some need to be pondered upon. We faced a problem when we all became a part of the training and placement cell of our college. We were a team of more than 40 people but just two keys to the cell. We faced problems like searching for the keys, keeping it safe, work getting delayed due to the unavailability of the keys at the right time. All these problems led us to this idea of implementing face recognition door with security measures to ensure a safe, quick and efficient use of the resources by the registered or authorized persons without bothering or running here and there for the keys at the cost of the work which suffered a lot.

1.2 AIM AND OBJECTIVE

The aim of our project is to build a face recognition door system using digital image processing techniques which will ensure a quick and easy access to a home equipped with the factor of security by incorporating the much in use technology of the Internet of Things.

1.3PROBLEM SPECIFICATIONS

This project aims at achieving the following specifications:

1) Face detection: a person's face will be detected when the person will be standing within a range of 1-2 meters, facing the camera.
2) Face recognition: a registered person will be recognized after proper detection and the door will unlock. An unauthorized person will be recognized as unknown and the door will remain locked.
3) Cloud Platform: any entry either registered or unregistered will be captured and uploaded on a cloud platform with a date and time stamp.

1.4 PLAN OF THE WORK

WEEK 1: Researching about what components to use and in which manner.

WEEK 2 and 3: Choose Raspberry pi and working on its set up.

WEEK 4: Interfacing and learning OpenCV and Python.

WEEK 5: Working on face detection code.

WEEK 6: Working of eyes and lip detection code.

WEEK 7, 8, 9, 10: Working on different digital image processing techniques.

WEEK 11: Working on interfacing with the cloud platform.

WEEK 12: Making the model of the project.

Chapter 2 LITERATURE SURVEY

2.1 FACIAL RECOGNITION

A **facial recognition system** is a technology capable of identifying or verifying a person from a digital image or a video frame from a video source. There are multiple methods in which facial recognition systems work, but in general, they work by comparing selected facial features from given image with faces within a database. It is also described as a Biometric Artificial Intelligence based application that can uniquely identify a person by analyzing patterns based on the person's facial textures and shape.

2.2 HAAR CASCADE FACE DETECTION

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Here we will work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, Haar features shown in the below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under the

white rectangle from sum of pixels under the black rectangle.

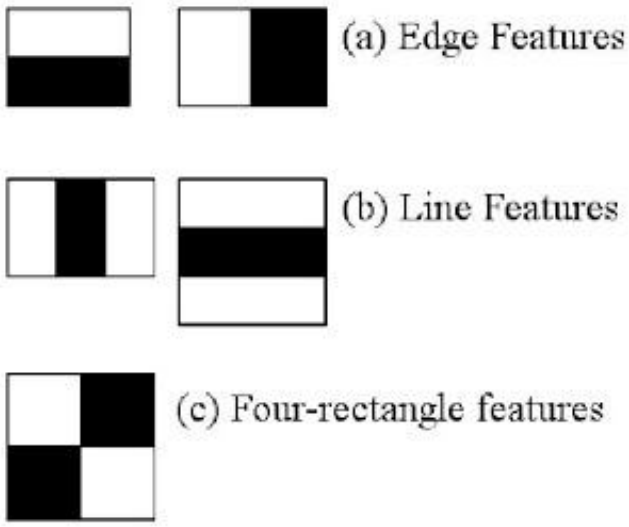

Figure 1. MATRIX FOR HAAR CASCADE

Now, all possible sizes and locations of each kernel are used to calculate lots of features. (Just imagine how much computation it needs? Even a 24x24 window results over 160000 features). For each feature calculation, we need to find the sum of the pixels under white and black rectangles. To solve this, they introduced the integral image. However large your image, it reduces the calculations for a given pixel to an operation involving just four pixels. Nice, isn't it? It makes things super-fast.

But among all these features we calculated, most of them are irrelevant. For example, consider the image below. The top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applied to cheeks or any other place is irrelevant. So how do we

select the best features out of 160000+ features? It is achieved by Adaboost.

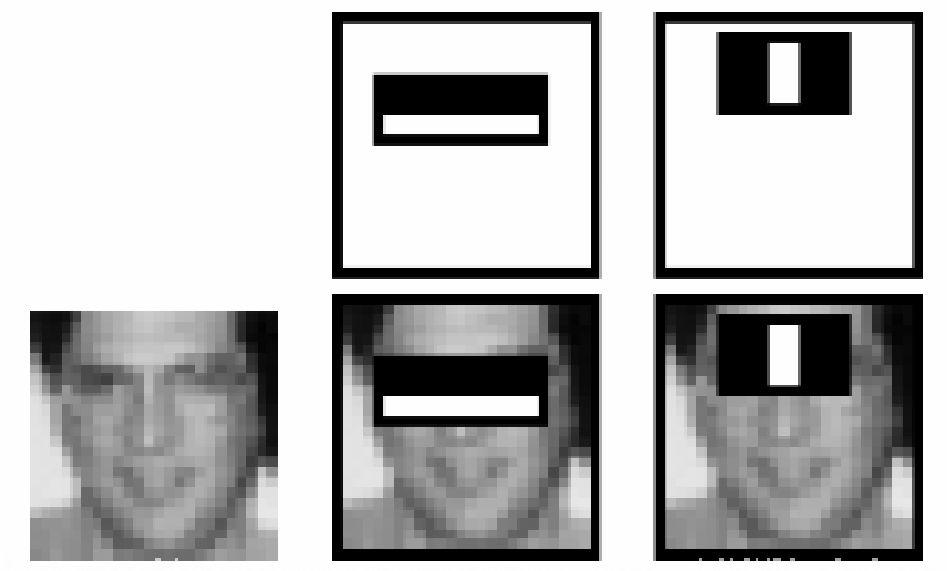

Figure 2. IMPLEMENTATION OF HAAR FOR DETECTING FACE

For this, we apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. Obviously, there will be errors or misclassifications. We select the features with minimum error rate, which means they are the features that most accurately classify the face and non-face images. (The process is not as simple as this. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. Then the same process is done. New error rates are calculated. Also new weights. The process is continued until the required accuracy or error rate is achieved or the required number of features is found).

The final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together with others forms a strong classifier. The paper says even 200 features provide detection with 95% accuracy.

Their final setup had around 6000 features. (Imagine a reduction from 160000+ features to 6000 features. That is a big gain).

So now you take an image. Take each 24x24 window. Apply 6000 features to it. Check if it is face or not.

In an image, most of the image is non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot, and don't process it again. Instead, focus on regions where there can be a face. This way, we spend more time checking possible face regions.

For this they introduced the concept of Cascade of Classifiers. Instead of applying all 6000 features on a window, the features are grouped into different stages of classifiers and applied one-by-one. (Normally the first few stages will contain very many fewer features). If a window fails the first stage, discard it. We don't consider the remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region. How is that plan!

The authors' detector had 6000+ features with 38 stages with 1, 10, 25, 25 and 50 features in the first five stages. (The two features in the above image are actually obtained as the best two features from Adaboost). According to the authors, on average 10 features out of 6000+ are evaluated per sub-window.

So this is a simple intuitive explanation of how Viola-Jones face detection works. Now we know how a face is detected. So, let us now Recognize Faces.

2.3 LBPH FACE RECOGNITION

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.

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.

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.

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:

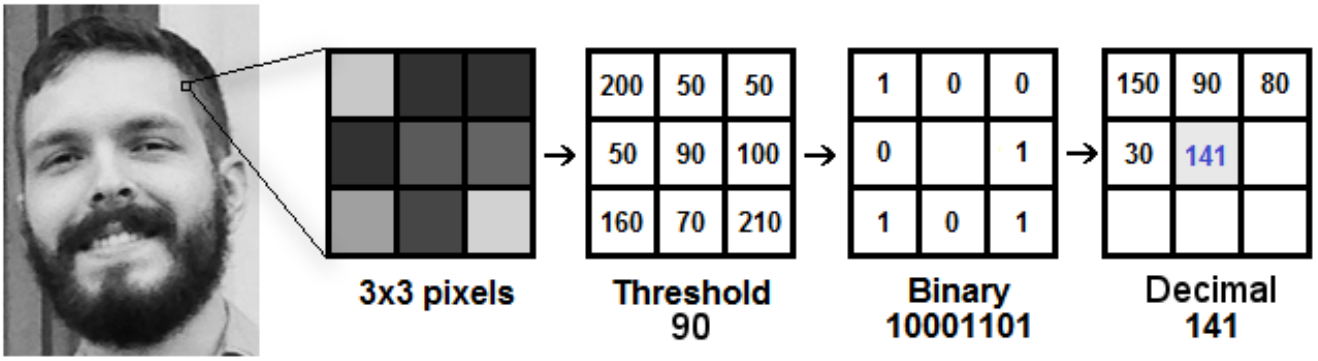

Figure 3. CALCULATION FOR CREATING INTERMEDIATE IMAGE

Based on the image, 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.

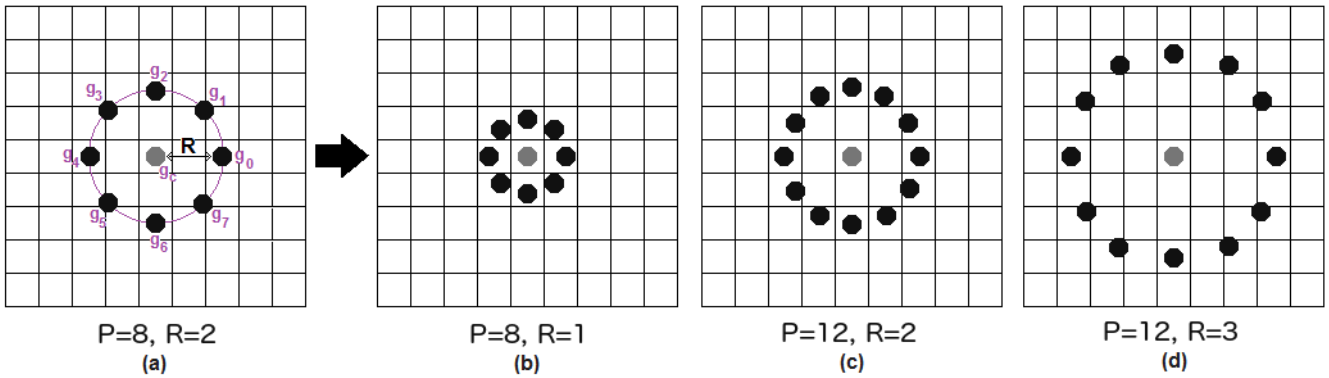

Figure 4. IMPLEMENTATION USING DIFFERENT RADIUS AND POINTS

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.

Extracting the Histograms: Now, using the image generated in the last step, we can use the Grid X and Grid Y parameters to divide the image into multiple grids, as can be seen in the following image:

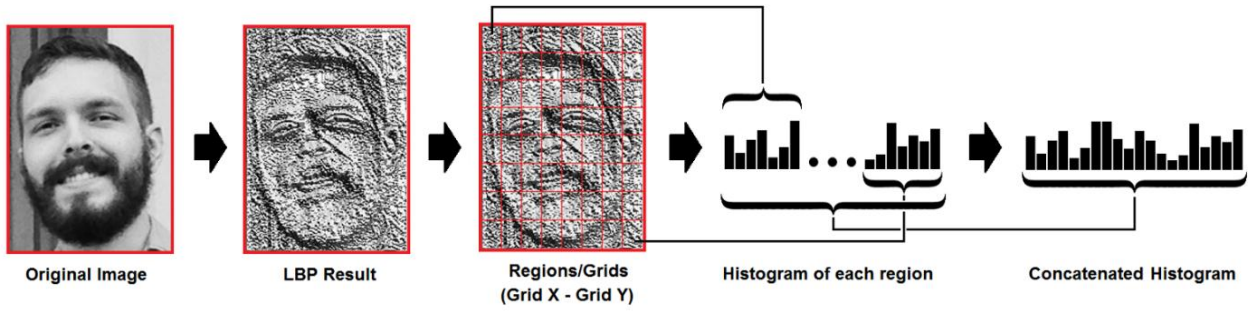

Figure 5. GENERATING HISTOGRAM FROM IMAGE

Based on the image above, we can extract the histogram of each region as follows:

As we have an image in grayscale, each histogram (from each grid) will contain only 256 positions (0~255) representing the occurrences of each pixel intensity.

Then, we need to concatenate each histogram to create a new and bigger histogram. Supposing we have 8x8 grids, we will have 8x8x256=16.384 positions in the final histogram. The final histogram represents the characteristics of the image original image.

The LBPH algorithm is pretty much it.

Performing the face recognition: 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 create 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: Euclidean 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.

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.

2.4 EIGEN FACE RECOGNITION

Eigenfaces refers to an appearance-based approach to face recognition that seeks to capture the variation in a collection of face images and use this information to encode and compare images of individual faces in a holistic (as opposed to a parts-based or feature-based) manner. Specifically, the eigenfaces are the principal components of a distribution of faces, or equivalently, the eigenvectors of the covariance matrix of the set of face images, where an image with N pixels is considered a point (or vector) in N-dimensional space. The idea of using principal components to represent human faces was developed by Sirovich and Kirby (Sirovich and Kirby 1987) and used by Turk and Pentland (Turk and Pentland 1991) for face detection and recognition. The Eigenface approach is considered by many to be the first working facial recognition technology, and it served as the basis for one of the top commercial face recognition technology products. Since its initial development and publication, there have been many extensions to the original method and many new developments in automatic face recognition systems. Eigenfaces is still often considered as a baseline comparison method to demonstrate the minimum expected performance of such a system.

The motivation of Eigenfaces is twofold:

Extract the relevant facial information, which may or may not be directly related to human intuition of face features such as the eyes, nose, and lips. One way to do so is to capture the statistical variation between face images.

Represent face images efficiently. To reduce the computation and space complexity, each face image can be represented using a small number of

parameters.

The eigenfaces may be considered as a set of features which characterize the global variation among face images. Then each face image is approximated using a subset of the eigenfaces, those associated with the largest eigenvalues. These features account for the most variance in the training set.

Figure 6. AVERAGE IMAGE PRODUCED BY EIGEN FACE

2.5 DEEP LEARNING BASED FACIAL RECOGNITION

Deep Learning is the part of machine learning. We will use Deep Learning Algorithm using Python to achieve high accuracy. In python we have dlib deep learning library and Adam Geitgey's Face Recognition library to recognize face. Let us now understand step-by-step how it works:

Step 1: Detecting Face from image using Haar Cascade.

Step 2: Posing and Projecting Faces

When, we isolated the faces in our image. But now we have to deal with the problem that faces turned different directions look totally different to a computer. To account for this, we will try to warp each picture so that the eyes and lips are always in the sample place in the image. This will make it a lot easier for us to compare faces in the next steps. To do this, we are going to use an algorithm called face landmark estimation. There are lots of ways to do this, but we are going to use the approach invented in 2014 by Vahid Kazemi and Josephine Sullivan.

The basic idea is we will come up with 68 specific points (called landmarks) that exist on every face—the top of the chin, the outside edge of each eye, the inner edge of each eyebrow, etc. Then we will train a machine learning algorithm to be able to find these 68 specific points on any face:

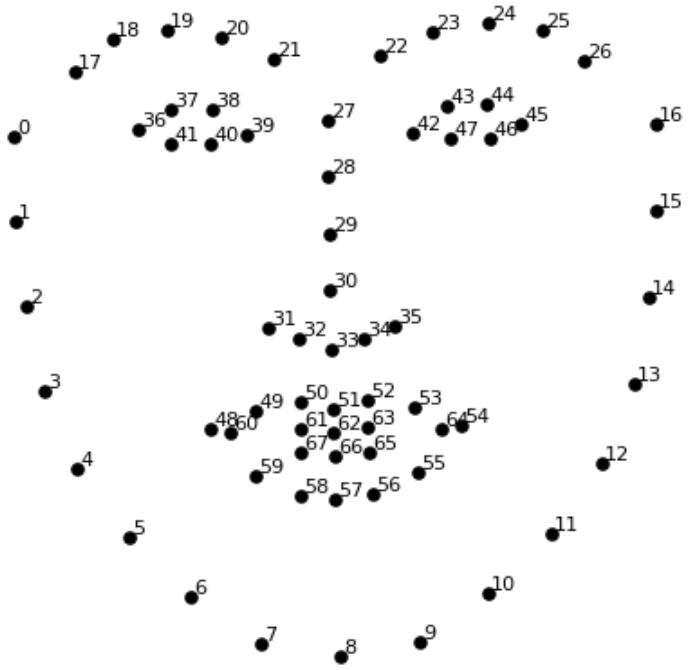

Figure 7. 68 POINT FACIAL LANDMARK TO RECOGNIZE FACE

Step 3: Encoding Faces

The simplest approach to face recognition is to directly compare the unknown face we found in Step 2 with all the pictures we have of people that have already been tagged. When we find a previously tagged face that looks very similar to our unknown face, it must be the same person. Seems like a pretty good idea, right? There's actually a huge problem with that approach. What we need is a way to extract a few basic measurements from each face. Then we could measure our unknown face the same way and find the known face with the closest measurements. It turns out that the measurements that seem obvious to us humans (like eye color) don't really make sense to a computer looking at individual pixels in an image. Researchers have discovered that the most accurate approach is to let

the computer figure out the measurements to collect itself. Deep learning does a better job than humans at figuring out which parts of a face are important to measure.

The solution is to train a Deep Convolutional Neural Network. But instead of training the network to recognize pictures objects like we did last time, we are going to train it to generate 128 measurements for each face.

The training process works by looking at 3 face images at a time:

1. Load a training face image of a known person
2. Load another picture of the same known person
3. Load a picture of a totally different person

Then the algorithm looks at the measurements it is currently generating for each of those three images. It then tweaks the neural network slightly so that it makes sure the measurements it generates for #1 and #2 are slightly closer while making sure the measurements for #2 and #3 are slightly further apart:

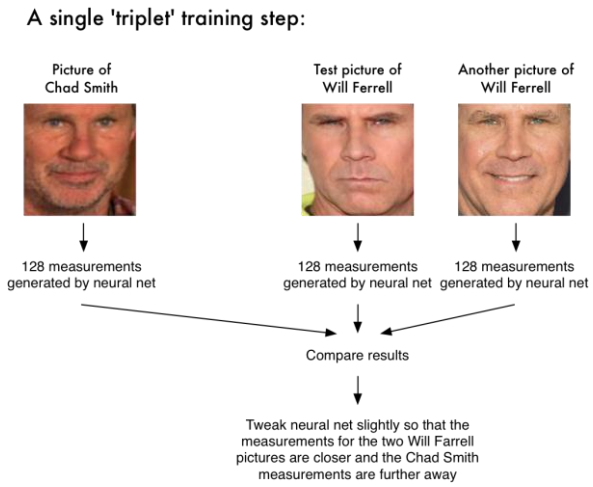

Figure 8. TRAINNING THE NEURAL NETWORK

After repeating this step millions of times for millions of images of thousands of different people, the neural network learns to reliably generate 128 measurements for each person. Any ten different pictures of the same person should give roughly the same measurements.

Machine learning people call the 128 measurements of each face an embedding. The idea of reducing complicated raw data like a picture into a list of computer-generated numbers comes up a lot in machine learning (especially in language translation).

Step 4: Finding the person's name from the encoding

This last step is actually the easiest step in the whole process. All we have to do is find the person in our database of known people who has the closest measurements to our test image.

You can do that by using any basic machine learning classification algorithm. No fancy deep learning tricks are needed. We'll use a simple linear SVM classifier, but lots of classification algorithms could work. All we need to do is train a classifier that can take in the measurements from a new test image and tells which known person is the closest match. Running this classifier takes milliseconds. The result of the classifier is the name of the person!

2.6 COMPARISION BETWEEN THESE ALGORITHMS

We started with LBPH facial recognition technique for facial recognition but to achieve more accuracy we studied and compare other algorithms as well and finally implemented using deep learning.

Table 1. COMPARISION BETWEEN VARIOUS RECOGNITION TECHNIQUES

CRITERIA	EIGEN FACE	LBPH	DEEP LEARNING
Principle of dataset generation	Component Based	Pixel Based	Facial Features (68 face Landmark)
Basic principle	PCA	Histogram	-
Efficiency	LOW	HIGH	HIGHEST
How to increase accuracy	Gather data in all conditions of lighting	More Images more Accuracy	Using More facial Features, other ML algorithms and Software libraries
Timing efficiency	FAST	FAST	LAGS A BIT

2.7 AWS CLOUD SERVICES-S3

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. This means customers of all sizes and industries can use it to store and protect any amount of data for a range of use cases, such as websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics. Amazon S3 provides easy-to-use management features so you can organize your data and configure finely-tuned access controls to meet your specific business, organizational, and compliance requirements. Amazon S3 is designed for 99.999999999% (11 9's) of durability, and stores data for millions of applications for companies all around the world.

Figure 9. SERVICES OFFERED BY AWS

Chapter 3 IMPLEMENTATION

3.1 COMPONENTS USED

3.1.1 HARDWARE COMPONENTS

RASPBERRY PI:

The **Raspberry Pi** is a series of small single-board computers developed in the United Kingdom by the Foundation to promote teaching of basic computer science. Several generations of Raspberry Pis have been released. All models feature a Broadcom system on a chip (SoC) with an integrated ARM-compatible central processing unit (CPU) and on-chip graphics processing unit (GPU). The B-models have an 8P8C Ethernet port and the Pi 3 and Pi Zero W have on-board Wi-Fi 802.11n and Bluetooth. Prices range from US$5 to $35.

Raspberry Pi 3 Model B was released in February 2016 with a 1.2 GHz 64-bit quad core processor, on-board WiFi, Bluetooth and USB boot capabilities. The Raspberry Pi 3 specifications are as given below:

SoC: Broadcom BCM2837
CPU: 4× ARM Cortex-A53, 1.2GHz
GPU: Broadcom VideoCore IV
RAM: 1GB LPDDR2 (900 MHz)
Networking: 10/100 Ethernet, 2.4GHz 802.11n wireless
Bluetooth: Bluetooth 4.1 Classic, Bluetooth Low Energy
GPIO: 40-pin header, populated
Ports: HDMI, 3.5mm analogue audio-video jack, 4× USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)

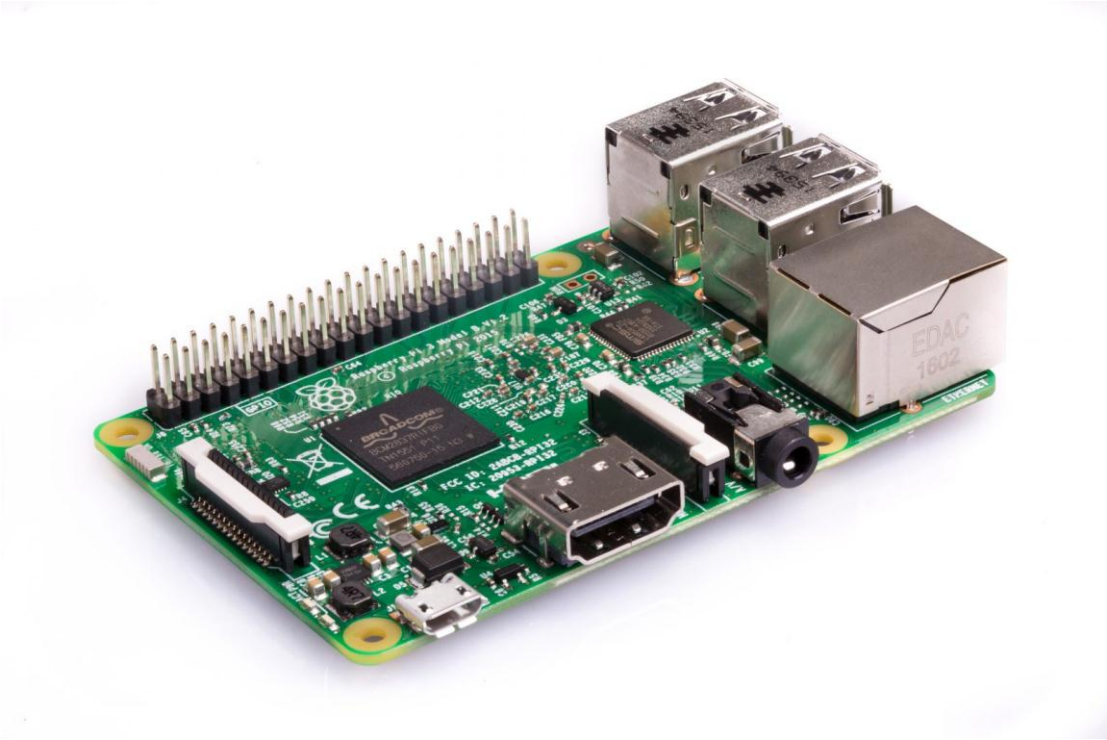

Figure 10. RASPBERRY PI 3B

WEBCAM:

A webcam is a video camera that feeds or streams its image in real time to or through a computer to a computer network. When "captured" by the computer, the video stream may be saved, viewed or sent on to other networks travelling through systems such as the internet, and e-mailed as an attachment. When sent to a remote location, the video stream may be saved, viewed or on sent there. A webcam is generally connected by a USB cable, or similar cable.

Popular uses of webcam include security surveillance, computer vision, video broadcasting, and for recording social videos. Webcams are known for their low manufacturing cost and their high flexibility. They also have a decent resolution which makes them suitable for small scale security purposes.

Figure 11. USB WECAM

SERVO MOTOR:

A **servomotor** is a rotary actuator or linear actuator that allows for precise control of angular or linear position, velocity and acceleration. It consists of a suitable motor coupled to a sensor for position feedback. It also requires a relatively sophisticated controller. A servomotor is a closed-loop servomechanism that uses position feedback to control its motion and final position. The input to its control is a signal (either analogue or digital) representing the position commanded for the output shaft. Servomotors are generally used as a high-performance alternative to the stepper motor.

Figure 12. SERVO MOTOR

3.1.2 SOFTWARE COMPONENTS

OpenCV:

OpenCV (*Open source computer vision*) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license. OpenCV supports the deep learning frameworks.

OpenCV is written in C++ and its primary interface is in C++, but it still retains a less comprehensive though extensive older C interface. There are bindings in Python, Java and MATLAB/OCTAVE. The API for these interfaces can be found in the online documentation.

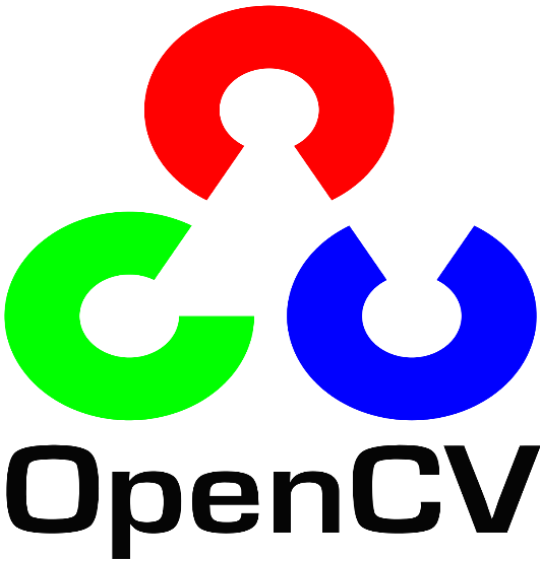

Figure 13. OPENCV

AWS CLOUD PLATFORM:

Amazon Web Services (**AWS**) is a subsidiary of Amazon that provides on-demand cloud computing platforms to individuals, companies and governments, on a metered pay-as-you-go basis. In aggregate, these cloud computing web services provide a set of primitive, abstract technical infrastructure and distributed computing building blocks and tools. The AWS Cloud provides a broad set of infrastructure services, such as computing power, storage options, networking and databases, delivered as a utility: on-demand, available in seconds, with pay-as-you-go pricing.

Figure 14. AWS

RASPBIAN OS:

Raspbian is a Debian-based computer operating system for Raspberry Pi. There are several versions of Raspbian including Raspbian Stretch and Raspbian Jessie. Since 2015 it has been officially provided by the Raspberry Pi Foundation as the primary operating system for the family of Raspberry Pi single-board computers. Raspbian was created by Mike Thompson and Peter Green as an independent project. The initial build was completed in June 2012.The operating system is still under active development. Raspbian is highly optimized for the Raspberry Pi line's low-performance ARM CPUs.

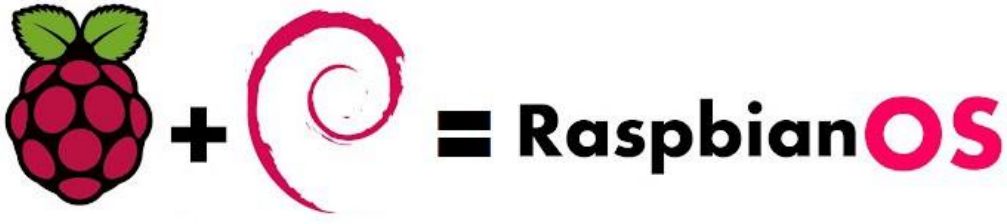

Figure 15. RASPBIAN OS FOR PI

PYTHON:

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Van Rossum led the language community until July 2018.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python features a comprehensive standard library, and is referred to as "batteries included". Python interpreters are available for many operating systems.

Figure 16. PYTHON

3.2 IMPLEMENTATION OF FACE DETECTION

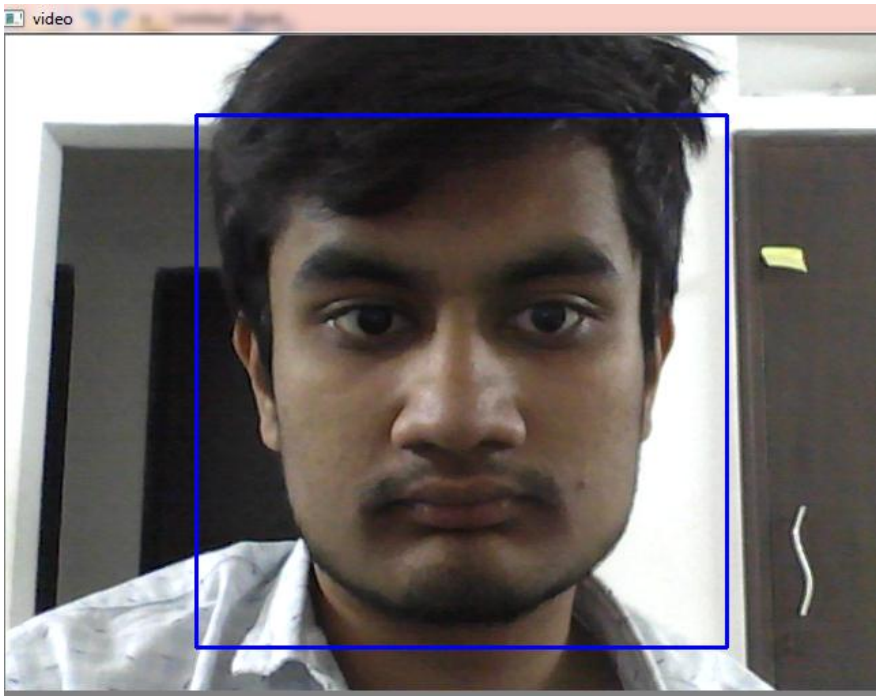

Figure 17. FACE DETECTION SUCCESFULLY IMPLEMETNED

3.3 WORKING

The main objective of the experiment work is to implement a Face Recognition System based on Convolutional Neural Networks (Face Recognition) in a real-time embedded system such as Raspberry Pi and demonstrate a practical application in the form of Automated Door Access. Further improvements to the above system include an anti-spoofing technique in the form of eye-blink detection program and an intruder alert mail sent to the owner containing a picture of the visitor at the door.

The face recognition system called Face Recognition includes the following stages

- Histograms of Oriented Gradients, Face Landmark Estimation, Convolutional Neural Network from Adam Geitgey and Support Vector Machines. Histograms of Oriented Gradients are used for face detection. The picture initially is converted as a black and white picture. Every pixel in the image is replaced by an arrow called as gradient those points to the direction of the darker pixel. This is to ensure that the same picture differing in brightness does not get identified as a different picture altogether. The whole image containing all the gradients will then look like an outline of a face if face exists in picture.

The HOG algorithm applies a 'sliding window' across the entire image. For every stage of the sliding window, an HOG descriptor for that region is calculated. The descriptor is then compared to a template using a trained SVM to determine if a person's face is located in that region or not. The detection window is 64 pixels wide by 128 pixels tall. The 64 x 128-pixel window is divided into 7 rows and 15 columns, a total of 105 blocks. Each block is comprised of 4 cells, with each cell having a 9 bin histogram. Thus the total vector size will be 7x15x4x9 = 3780 values. This is to account for different poses of the face. It starts with identifying 68 points that are there in almost any face.

We then want to identify the positions of these points. These points include the position of edge of eyes and chin etc. After these positions are identified, these points are rotated, scaled and sheared until they are all aligned to the desired and fixed position of these 68 features. The facial landmark detector based on the One Millisecond Face Alignment with an Ensemble of Regression Trees paper by Kazemi and Sullivan (2014) is implemented in the Dlib library.

This method includes:

1. A training set that has the coordinates of each feature and these features are manually labeled.

2. Priors or more specifically, the probability on distance between pairs of input pixels.

Such a face landmark detector is made available in the Dlib library. It is used to generate the 68 x-y coordinates that represent the location of specific facial structures on the face.

The final step in the face recognition system involves utilisation of a linear svm classifier. The Support Vector Machine takes in the 128 measurements from input test image and compares it with the 128 measurements of trained test images stored in the database. If a match is found, the svm classifier will return the name of the person who has the closest measurements to that of input image. SVMs are known to achieve significantly higher accuracy than traditional methods just after few rounds of relevance feedback. If the above steps give a positive result and the captured image is recognized by the system, the door will unlock via a servo mechanism. A servo motor connected to the door latch will turn 90 to unlock the door and after a few seconds will lock the door again. The angle of rotation of the servo motor is controlled by pulse width modulation technique.

Once the doorbell is pressed and the face is detected then we send a photo of the person to AWS cloud using boto3. We first set an account on AWS and the use S3 service of AWS. In S3 we make a bucket and we get an API key. We configure connection between our PI and AWS using this API key. Whenever any person

presses the doorbell a photo is captured and then uploaded to the cloud with timestamp.

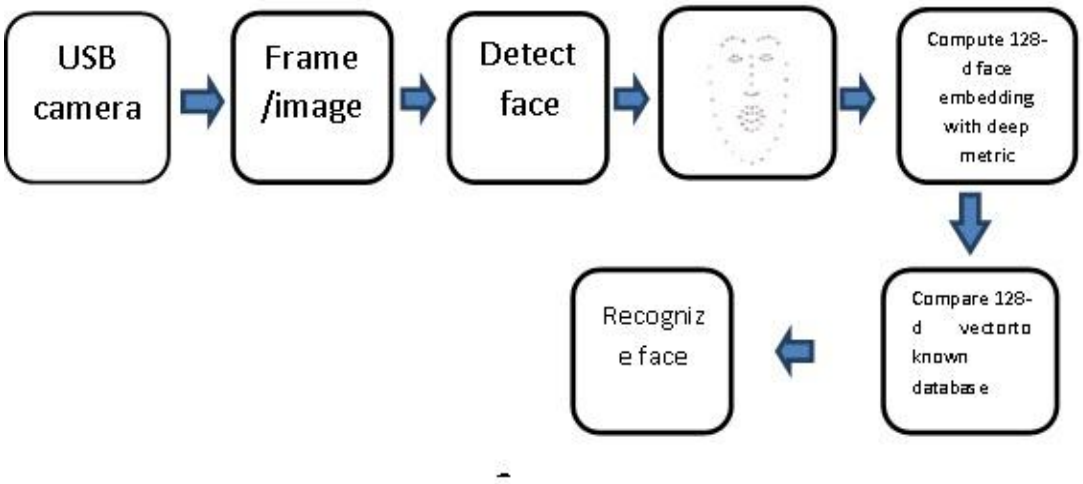

Figure 18. SOFTWARE BLOCK DIAGRAM

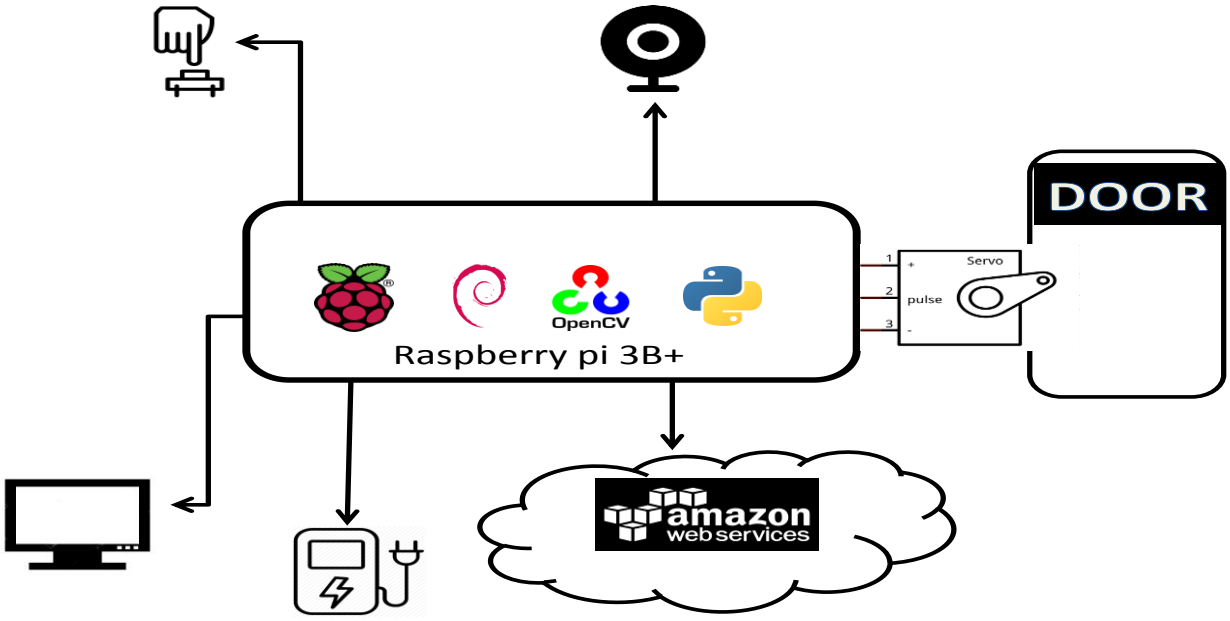

Figure 19. HARDWARE BLOCK DIAGRAM

3.4 CODE FOR FACE DETECTION

```
import numpy as np
import cv2

faceCascade =
cv2.CascadeClassifier('Cascades/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,

        scaleFactor=1.2,
        minNeighbors=5
        ,
        minSize=(20, 20)
    )

    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 == 27: # press 'ESC' to quit
        break
cap.release()
cv2.destroyAllWindows()
```

3.5 CODE FOR FACIAL RECOGITION DOOR LOCK

```
# python pi_face_recognition.py --cascade haarcascade_frontalface_default.xml --
encodings encodings.pickle

# import the necessary packages

from imutils.video import VideoStream
from imutils.video import FPS
import face_recognition
import argparse
import imutils
import pickle
import time
import cv2
import RPi.GPIO as GPIO
import datetime
import boto3


servoPIN = 17
GPIO.setmode(GPIO.BCM)
GPIO.setup(servoPIN,GPIO.OUT)

p=GPIO.PWM(servoPIN,50)
p.start(2.5)

# construct the argument parser and parse the arguments

# load the known faces and embeddings along with OpenCV's Haar
# cascade for face detection
print("[INFO] loading encodings + face detector...")
data = pickle.loads(open("encodings.pickle", "rb").read())
cascadePath = "haarcascade_frontalface_default.xml"
detector = cv2.CascadeClassifier(cascadePath)

imgcount=0
```

```
s3=boto3.client('s3')

# initialize the video stream and allow the camera sensor to warm up
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
# vs = VideoStream(usePiCamera=True).start()
time.sleep(2.0)

# start the FPS counter
fps = FPS().start()

# loop over frames from the video file stream
while True:
    # grab the frame from the threaded video stream and resize it
    # to 500px (to speedup processing)
    frame = vs.read()
    frame = imutils.resize(frame, width=500)
    frame=cv2.flip(frame,-1)


    # convert the input frame from (1) BGR to grayscale (for face
    # detection) and (2) from BGR to RGB (for face recognition)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # detect faces in the grayscale frame
    rects = detector.detectMultiScale(gray, scaleFactor=1.1,
        minNeighbors=5, minSize=(30, 30),
        flags=cv2.CASCADE_SCALE_IMAGE)

    # OpenCV returns bounding box coordinates in (x, y, w, h) order
    # but we need them in (top, right, bottom, left) order, so we
    # need to do a bit of reordering
    boxes = [(y, x + w, y + h, x) for (x, y, w, h) in rects]

    # compute the facial embeddings for each face bounding box
    encodings = face_recognition.face_encodings(rgb, boxes)
    names = []
```

```
# loop over the facial embeddings
for encoding in encodings:
    # attempt to match each face in the input image to our known
    # encodings
    matches = face_recognition.compare_faces(data["encodings"],
        encoding)
    name = "Unknown"

    # check to see if we have found a match
    if True in matches:
        # find the indexes of all matched faces then initialize a
        # dictionary to count the total number of times each face
        # was matched
        matchedIdxs = [i for (i, b) in enumerate(matches) if b]
        counts = {}

        # loop over the matched indexes and maintain a count for
        # each recognized face face
        for i in matchedIdxs:
            name = data["names"][i]
            counts[name] = counts.get(name, 0) + 1

        # determine the recognized face with the largest number
        # of votes (note: in the event of an unlikely tie Python
        # will select first entry in the dictionary)
        name = max(counts, key=counts.get)

    # update the list of names
    names.append(name)

# loop over the recognized faces
for ((top, right, bottom, left), name) in zip(boxes, names):
    # draw the predicted face name on the image
    cv2.rectangle(frame, (left, top), (right, bottom),
        (0, 255, 0), 2)
    y = top - 15 if top - 15 > 15 else top + 15
    cv2.putText(frame, name, (left, y), cv2.FONT_HERSHEY_SIMPLEX,
        0.75, (0, 255, 0), 2)
```

```
        if name != 'Unknown':
            p.ChangeDutyCycle(12.5)
            time.sleep(5)
            p.ChangeDutyCycle(2.5)
            time.sleep(2)


    # display the image to our screen
    cv2.imshow("Frame", frame)
    if imgcount==0:
        now=datetime.datetime.now()
        now_string=str(now.strftime("%Y-%m-%d_%H:%M:%S"))
        cv2.imwrite("logs/"+now_string+".png", frame)
        imgcount=imgcount+1

    key = cv2.waitKey(1) & 0xFF

    # if the `q` key was pressed, break from the loop
    if key == ord("q"):
        break

    # update the FPS counter
    fps.update()

s3.upload_file("logs/"+now_string+".png","doorlog","logs/"+now_string+".png")
# stop the timer and display FPS information
fps.stop()
print("[INFO] elasped time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
p.stop()
GPIO.cleanup()
```

3.6 CODE FOR ENCODING FACES INTO 128d

```
# import the necessary packages
from imutils import paths
import face_recognition
import argparse
import pickle
import cv2
import os

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--dataset", required=True,
	help="path to input directory of faces + images")
ap.add_argument("-e", "--encodings", required=True,
	help="path to serialized db of facial encodings")
ap.add_argument("-d", "--detection-method", type=str, default="cnn",
	help="face detection model to use: either `hog` or `cnn`")
args = vars(ap.parse_args())

# grab the paths to the input images in our dataset
print("[INFO] quantifying faces...")
imagePaths = list(paths.list_images(args["dataset"]))

# initialize the list of known encodings and known names
knownEncodings = []
knownNames = []

# loop over the image paths
for (i, imagePath) in enumerate(imagePaths):
	# extract the person name from the image path
	print("[INFO] processing image {}/{}".format(i + 1,
		len(imagePaths)))
	name = imagePath.split(os.path.sep)[-2]

	# load the input image and convert it from RGB (OpenCV ordering)
	# to dlib ordering (RGB)
	image = cv2.imread(imagePath)
```

```
        rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # detect the (x, y)-coordinates of the bounding boxes
        # corresponding to each face in the input image
        boxes = face_recognition.face_locations(rgb,
                model=args["detection_method"])

        # compute the facial embedding for the face
        encodings = face_recognition.face_encodings(rgb, boxes)

        # loop over the encodings
        for encoding in encodings:
                # add each encoding + name to our set of known names and
                # encodings
                knownEncodings.append(encoding)
                knownNames.append(name)

# dump the facial encodings + names to disk
print("[INFO] serializing encodings...")
data = {"encodings": knownEncodings, "names": knownNames}
f = open(args["encodings"], "wb")
f.write(pickle.dumps(data))
f.close()
```

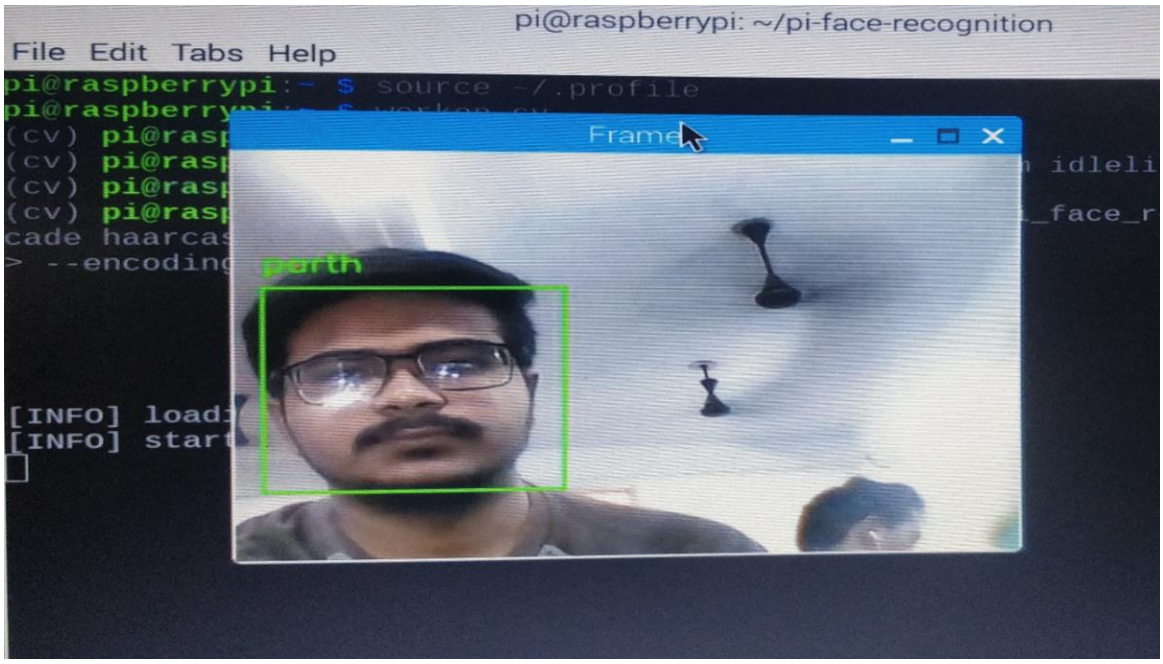

Figure 20 FACE RECOGNITION

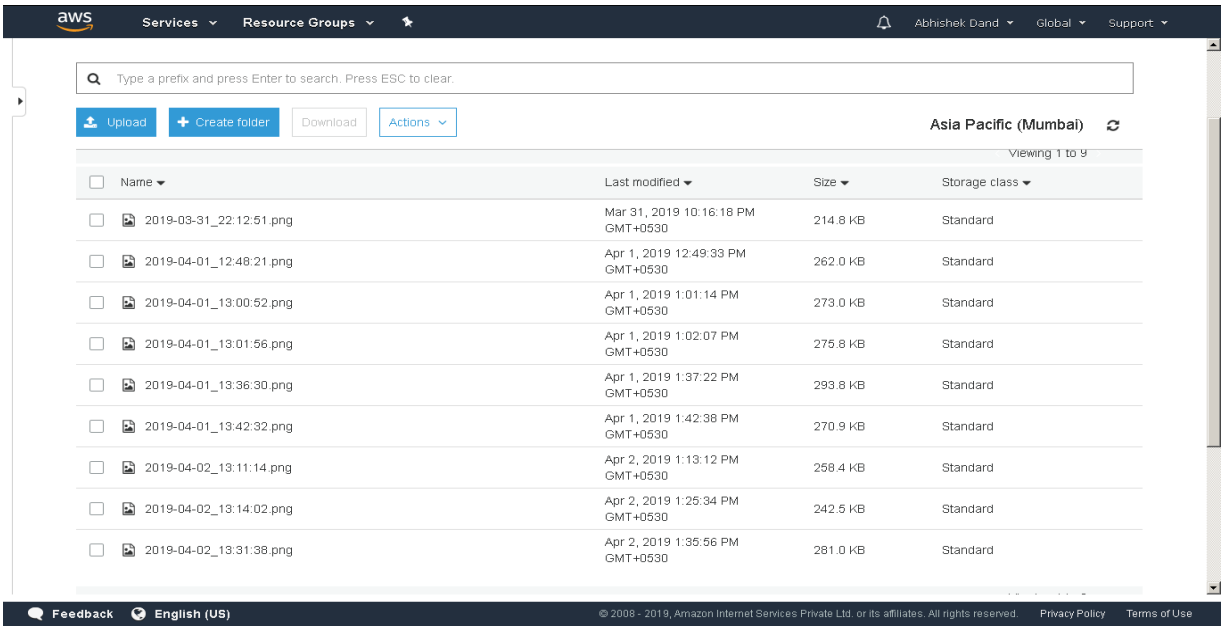

Figure 21. UPLOADING VISITOR IMAGE ON CLOUD

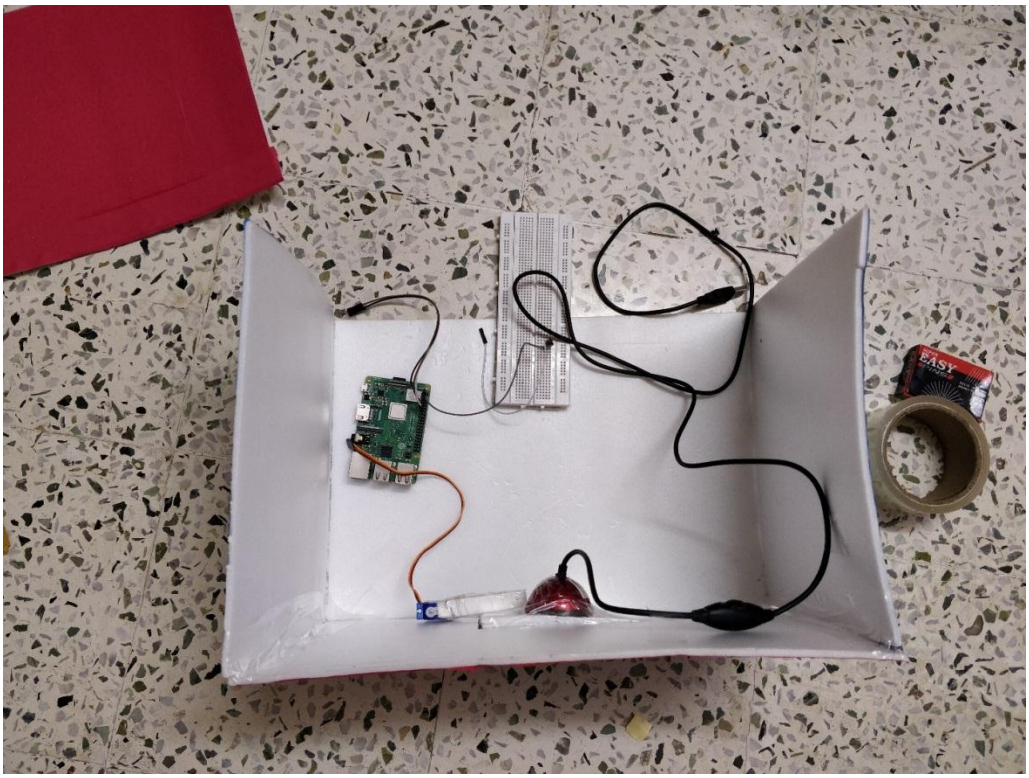

Figure 22. IMPLEMENTATION ON MODEL

Figure 23. MODEL OF SMART DOOR

Chapter 4 CONCLUSION

4.1 SUMMARY OF RESULTS

The result of our project is that we have been able to achieve our aim of building a face recognition door using deep learning technology for image processing and by providing the factor of security and remote access to data by uploading it on Amazon Web Service cloud platform thus involving the Internet of Things.

4.2 MERITS

Our project has secured the following merits:

- It is cost effective.
- It provides real time data on cloud thus ensuring security.
- It saves time as one doesn't need to hassle for the keys.

4.3 DEMERITS

Every coin has two sides. This project also has some demerits. They are as follows:

- A photograph can be used for unlocking the door.
- The system can be hacked.
- There is a little lagging in the process.

4.4 FUTURE SCOPE

Nothing is perfect and there is always a scope of improvement in everything. We can always make things better and our project is no different. There are a few concepts and areas that can be worked upon to get much advanced and better results for the same product. The areas of future scope for our project are as follows:

- Still image detection or motion detection can be implemented to avoid unlocking the system with a photograph of a registered person.
- The code can be made for secure to prevent hacking and data theft.
- Lagging can be overcome by implementing a more compatible and better development board.
- We can also incorporate API's to edit and update the database in a convenient manner.
- Additional security measures can be implemented.
- We can add algorithm for detecting real face so that it cannot be unlocked by photo of the owner.
- We can use more advanced board to get good computational power.
- We can even use cloud services for face recognition with better accuracy.

REFERENCES

1. Druva Kumar S, Syeda Arzoo Irshad, Channayya, Rakshith "Implementation of Face Recognition System Using Convolutional Neural Networks for Automated Door Access Using Raspberry Pi" International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, Vol. 6, Issue 6, June 2017 DOI:10.15662/IJAREEIE.2017.0606131
2. https://www.pyimagesearch.com/2018/06/25/raspberry-pi-face-recognition/
3. SudhaNarang, Kriti Jain, MeghaSaxena, AashnaArora "Comparison of Face Recognition Algorithms Using Opencv for Attendance System" International Journal of Scientific and Research Publications, Volume 8, Issue 2, February 2018 ISSN 2250-315
4. https://tutorials-raspberrypi.com/raspberry-pi-servo-motor-control/
5. https://docs.aws.amazon.com/s3/index.htm