TABLE OF CONTENTS

TITLE	PAGE NO.
ACKNOWLEDGEMENT	3
ABSTRACT	4
INTRODUCTION	5
LITERARY SURVEY	6
SYSTEM SPECIFICATIONS	9
DESIGN	13
SYSTEM IMPLEMENTATION	16
TESTING	24
RESULT	27
CONCLUSION AND FUTURE WORKS	30
APPENDIX	31

LIST OF FIGURES

TITLE	FIGURE NO.
UML USECASE DIAGRAM	4.1
UML CLASS DIAGRAM	4.2
UML DEPLOYMENT DIAGRAM	4.3
SCREENSHOTS OF SYSTEM WORKING	7.1 - 7.6

ACKNOWLEDGEMENT

We would like to extend our sincere thanks to **Dr. Sudha Sadasivam G**, Professor and Head of the Department of Computer Science and Engineering for giving us the opportunity to execute this project.

We would like to profusely thank our Tutor **Dr. K Sathiyapriya** , Assistant Professor, Department of Computer Science and Engineering for her constant motivation, direction and guidance throughout the entire course of the project execution.

We would like to express our gratitude to our panel members **Dr. K Sathiyapriya**, Assistant Professor, Department of Computer Science and Engineering, **Mr. Engels R**. Assistant Professor, Department of Computer Science and Engineering for their guidance and continued support throughout the projects.

We would also like to thank our internal guide, **Dr. Sudha Sadhasivam G**., Professor and Head of Department of Computer Science and Engineering for her guidance and continued support throughout the projects.

We would also like to thank our external guide and mentor, **Mr. Vijay Jayaseelan**, Chief Operating Officer, PSG-ST for his support and help throughout our project.

We also thank all the Faculty members and lab assistants of the Department of Computer Science and Engineering for their support.

ABSTRACT

The growing importance placed by educational institutions on attendance has made it a field of great interest for technology to lend its services to. The best way to do it would be reduce the human interaction or degree of interaction as the nodal person, a teacher, devotes a good amount of their class times in settling squabbles arising out of attendance or in figuring out proxies. The aim of technology is to not only automate the process but ween out the problems that confront us such as proxies and the so-called “attendance-as-a-weapon”. The students shall have their credentials recorded in a database and their faces recognized by one of the many edge devices at a specified point in time. This snapshot shall be processed and a timestamp will be issued, based on the records being matched and the existence of a timestamp will provide the student with attendance for those aligned periods else they shall be marked absent.

This is a fool-proof method that goes about achieving our stated objectives.

CHAPTER 1

INTRODUCTION

1.1 PROBLEM STATEMENT

Educational institutions have always been concerned about the regularity of students. This is mainly due to the fact that regularity influences the overall academic performance to a considerable measure. The conventional method of marking attendance which is followed across most institutions is by taking a roll call. This method is very much time consuming and difficult. Thus the need for a computer based student system which will assist the faculty in maintaining attendance records automatically. This project aims to automate the attendance process. It is to be based on facial recognition, which is very efficient and eliminates chances of proxy attendance. This system can be implemented in various fields where keeping track of personnel attendance plays an important role. The project aims to move the responsibility of taking attendance from the lecturers to automated systems in such a way that the lecturers' intervention will only be needed under True Negative cases .

1.2 OBJECTIVE

The stated objective is to Detect and Recognise students entering into a class by identifying their facial features and Registering their attendance in the database.

The edge devices used for surveillance under normal conditions, such as cameras, shall aid in recognizing and matching every student under the lens and transmit it to the server as a JSON file.

The onus on teachers and edge devices, like in traditional automated attendance systems, need to be reduced with most of the processing capacity placed in the servers.

1.3 FACIAL RECOGNITION USING ANN

Facial Recognition is the cumulative approach of recognizing the unique structures of faces and using appropriate representations in formats that can be used to match them against existing database records and identifying them. The identified records should have corresponding attributes to supplement the record. Some classical hindrances to this process being successful is the ever-changing nature of faces, i.e., presence/absence of facial hair,glasses or the gradient of light available in a photo that is used to match. Using Artificial Neural Networks, the systems shall be trained to use the winecup structure and identify and tag the relevant people with agreeable accuracy.

CHAPTER 2

LITERATURE SURVEY

FaceNet: A Unified Embedding for Face Recognition and Clustering 17, Jun 2015

Florian Schroff, Dmitry Kalenichenko, James Philbin (Google Inc.)

Despite significant recent advances in the field of face recognition [10, 14, 15, 17], implementing face verification and recognition efficiently at scale presents serious challenges to current approaches. In this paper we present a system,calledFaceNet,that directly learns mapping from face images to a compact Euclidean space where distances directly correspond to a measure of face similarity. Once this space has been produced, tasks such as face recognition, verification and clustering can be easily implemented usingstandardtechniqueswithFaceNetembeddingsasfeature vectors. Our method uses a deep convolutional network trained to directly optimize the embedding itself, rather than an intermediate bottleneck layer as in previous deep learning approaches. To train, we use triplets of roughly aligned matching / non-matching face patches generated using a novel online triplet mining method. The benefit of our approach is much greater representational efficiency: we achieve state-of-the-artfac recognition performance using only 128-bytes per face. On the widely used Labeled Faces in the Wild (LFW) dataset, our system achieves a new record accuracy of 99.63%. On YouTube Faces DB it achieves 95.12%. Our system cuts the error rate in comparison to the best published result [15] by 30% on both datasets. We also introduce the concept of harmonic embeddings, and a harmonic triplet loss, which describe different versions of face embeddings (produced by different networks) that are compatible to each other and allow for direct comparison between each other.

Stacked Dense U-Nets with Dual Transformers for Robust Face Alignment 5, Dec 2018

Jia Guo, Jiankang Deng (InsightFace, Shanghai, China)

Facial landmark localisation in images captured in-the-wild is an important and challenging problem. The current state-of-the-art revolves around certain

kinds of Deep Convolutional Neural Networks (DCNNs) such as stacked U-Nets and Hourglass networks. In this work, we innovatively propose stacked dense U-Nets for this task. We design a novel scale aggregation network topology structure and a channel aggregation building block to improve the model's capacity without sacrificing the computational complexity and model size. With the assistance of deformable convolutions inside the stacked dense U-Nets and coherent loss for outside data transformation, our model obtains the ability to be spatially invariant to arbitrary input face images. Extensive experiments on many in-the-wild datasets, validate the robustness of the proposed method under extreme poses, exaggerated expressions and heavy occlusions. Finally, we show that accurate 3D face alignment can assist pose-invariant face recognition where we achieve a new state of-the-art accuracy on CFP-FP (98.514%).

ArcFace: Additive Angular Margin Loss for Deep Face Recognition 9, Feb 2019

Jiankang Deng, Jia Guo, Niannan Xue (Imperial College London)

One of the main challenges in feature learning using Deep Convolutional Neural Networks (DCNNs) for large scale face recognition is the design of appropriate loss functions that enhance discriminative power. Centre loss penalises the distance between the deep features and their corresponding class centres in the Euclidean space to achieve intra-class compactness. SphereFace assumes that the linear transformation matrix in the last fully connected layer can be used as a representation of the class centres in an angular space and penalises the angles between the deep features and their corresponding weights in a multiplicative way. Recently, a popular line of research is to incorporate margins in well-established loss functions in order to maximise face class separability. In this paper, we propose an Additive Angular Margin Loss (ArcFace) to obtain highly discriminative features for face recognition. The proposed ArcFace has a clear geometric interpretation due to the exact correspondence to the geodesic distance on the hypersphere. We present arguably the most extensive experimental evaluation of all the recent state-of- the-art face recognition methods on over 10 face recognition benchmarks including a new large-scale image database with a trillion level of pairs and a large-scale video dataset. We show that ArcFace consistently outperforms the state-of-the-art and can be easily implemented with negligible computational overhead.

FAREC — CNN based efficient face recognition technique using Dlib

12, Sept 2018

Sharma, Sathees Kumar Ramasamy (Hubino Technologies Private Limited, Chennai)

Despite advancement in face recognition, it has received much more attention in the last few decades in the field of research and in commercial markets this project proposes an efficient technique for face recognition system based on Deep Learning using Convolutional Neural Network (CNN) with Dlib face alignment. The paper describes the process involved in the face recognition like face alignment and feature extraction. The paper also emphasizes the importance of the face alignment, thus the accuracy and False Acceptance

Rate (FAR) is observed by using proposed techniques. The computational analysis shows the better performance than other state-of-art approaches.

Comparison of Modules and Choosing Dlib

Phase 1 - Used LFW for comparisons

Labeled Faces in the Wild is a public benchmark for face verification, also known as pair matching.

FaceNet, Dlib and OpenCV were selected while the other 2 were rejected.

OpenCV achieves a score of 91%, while both Dlib and FaceNet achieve a score of 98% on the LFW benchmark .

Phase 2 - Compare Techniques. Dropped OpenCV

Facenet and Dlib give similar results in most of the categories. Both Dlib and Facenet have better performance than OpenCV which utilises haar cascade feature detection. For embedding isolated faces, we use OpenFace implementation which uses Google's FaceNet architecture which gives better output using dlib library.

Phase 3 - Deciding Exact Model - Dlib

While both FaceNet and Dlib were both similar efficiencies, Dlib won out on load on the servers and was thus ultimately chosen.

Advantages of Dlib

- ● Accuracy of 98.38% on the Labeled Faces in the Wild benchmark.
- ● Uses Wine Cup methodology which is easier to encode and faster.
- ● Built in function in Pypy module face_recognition.
- ● Transfer Learning size is relatively small compared to other explored pre-built
  Neural Networks.

● CUDA and threading is an option but not necessary.

CHAPTER 3

SYSTEM SPECIFICATIONS

3.1 HARDWARE SPECIFICATIONS

- Network connection to server - for processing and database updates
- Database to store student attendance
- NVIDIA Jetson Nano
- PiCam
- Raspberry Pi

3.2 SOFTWARE SPECIFICATIONS

- Python(>=3.1.0)

Modules used :

- dlib==19.19.0
- Click==7.0
- numpy==1.18.1
- opencv_python==4.2.0.32
- requests==2.23.0
- imutils==0.5.3
- Pillow==7.1.1
- Flask

3.3 FEASIBILITY ANALYSIS

A step-by-step analysis of whether there are in any glitches in actualizing the project.

3.3.1 ECONOMIC FEASIBILITY

This particular endeavour is cost heavy on the hardware side as a single Jetson Nano device costs between 100$ to 129$ in e-marketplaces. Another deterrent is the cost of acquiring cameras in scale. The project exhibits scope to be scaled given the funding is not constricted. In current form the project is feasible only at a level suited for demonstrations as the labs in the college also have monetary requisites to use the equipment.

3.3.2 TECHNICAL FEASIBILITY

If the unlikely need to use a Raspberry-Pi arises then its supporting modules will only provide accuracy that is lesser than what a Jetson Nano is offering now. The advantage of having a Jetson Nano device with a GPU, allows for it to implement the stated aim.

3.3.3 OPERATIONAL FEASIBILITY

All functions and constraints that have been stated are completely operational and approved by the company. The functionalities can be tested properly and the entire project is scalable and can accommodate any future requirements, if any do arise.

3.4 FUNCTIONAL REQUIREMENTS

3.4.1 FR1 : Convert face to a vector

Description : The selected features of the face are taken and converted into vector arrays and stored against a specific and unique tag.

Dependencies : No Dependencies.

3.4.2 FR2 : Identify face in image

Description : In a provided snapshot the module is objectively tasked with finding a human face in the snapshot.

Dependencies : No Dependencies.

3.4.3 FR3 : Train a classroom

Description : A large file/folder is fed, the faces in the folders are all taken up and they are trained to be matched against a specific and unique tag.

Dependencies : FR1, FR2

3.4.4 FR4 : Identify person from trained module

Description : From a video stream, individual frames are pulled up and the module goes about finding the faces in that particular frame.

Dependencies : FR1, FR2

3.4.5 FR5 : Recognizing faces by comparing to trained data

Description : The pulled up snapshot of a face is compared against the existing trained database and the vectors are checked against this same database for matches.

Dependencies : FR1, FR2, FR3, FR4

3.4.6 FR6 : Post event to server

Description : Once a face is positively tagged from a match, then the server is notified of this match along with an accompanying timestamp.

Dependencies : FR5

3.5 NON-FUNCTIONAL REQUIREMENTS

3.5.1 NFR1 : User registration in server

Description : The user is organization specified and their requisite data along with their key is generated in this stage.

Dependencies : No Dependencies.

3.5.2 NFR2 : Less accurate performance mode for weaker hardware

Description : Based on the capacity of the hardware the accuracy of tagging the face for lesser quality of pixels is determined, the lesser the capacity, the lower the capacity, vice-versa.

Dependencies : FR1, FR2, FR3

3.5.3 NFR3 : Querying in a server

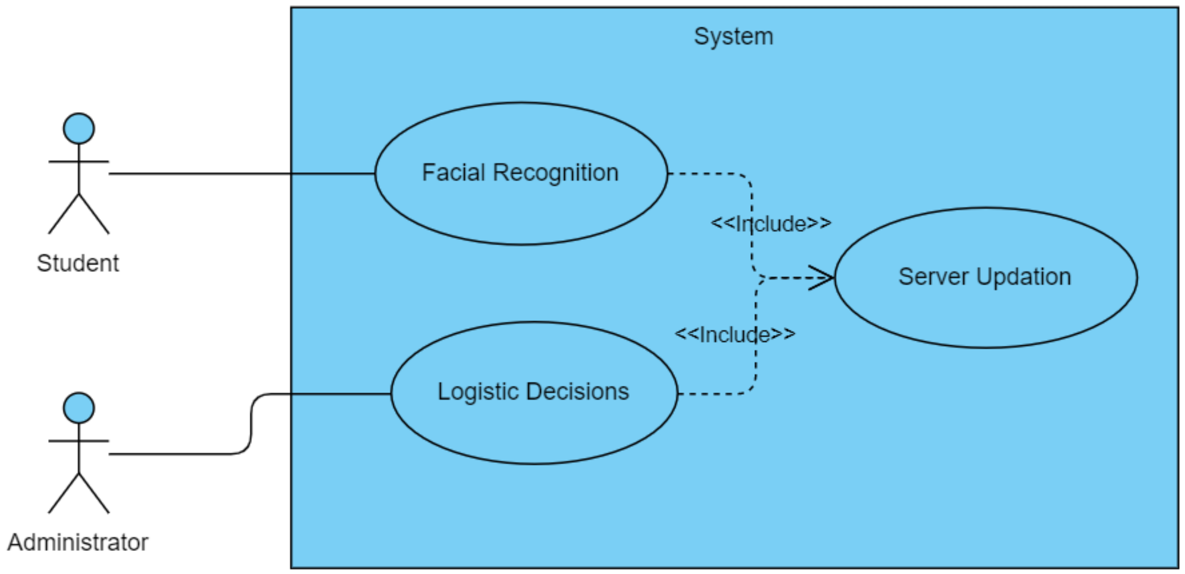
Description : The user can query about any student/face registered in the database to view their record, using their access key.

Dependencies : FR6, NFR1

CHAPTER 4

DESIGN

4.1 UML USECASE DIAGRAM



Stakeholders:

STUDENT

The students will have their faces' profile shots taken, to provide the system with a good domain set to learn from with its vast and stark differences. Their associated attributes such as their Roll Numbers and Course lists shall be taken along with the Winecup features of their face.

ADMINISTRATOR

The authorized personnel will be allowed to structure the schema to suit the institution's needs and also be able to query the attendance records of students and view them.

Processes

FACIAL RECOGNITION

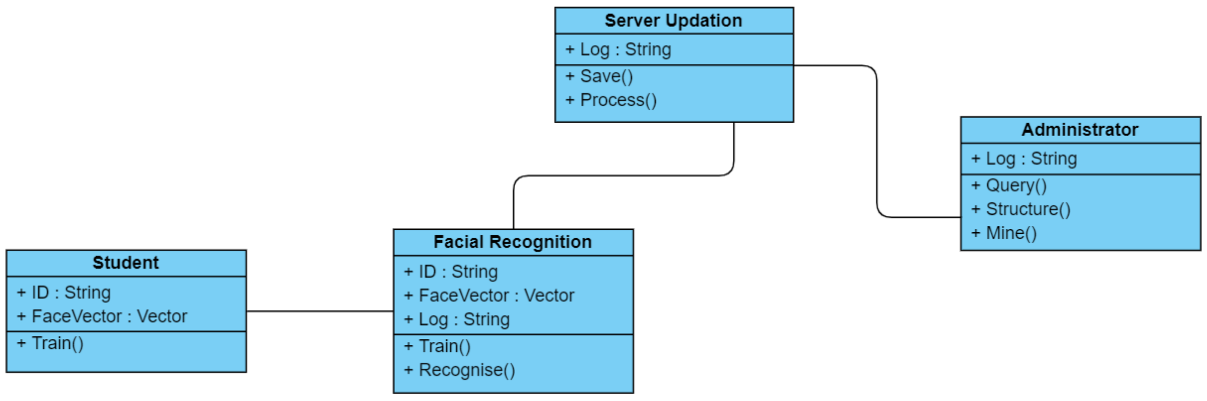
The photo that was taken will be converted to a corresponding face vector representing the winecup features and the Dlib model will aid in this process. The converted vectors are checked against the database and records and matching records cannot exceed 1 in number as each vector is unique.

The corresponding attributes and the timestamps are sent to the server.

SERVER UPDATION

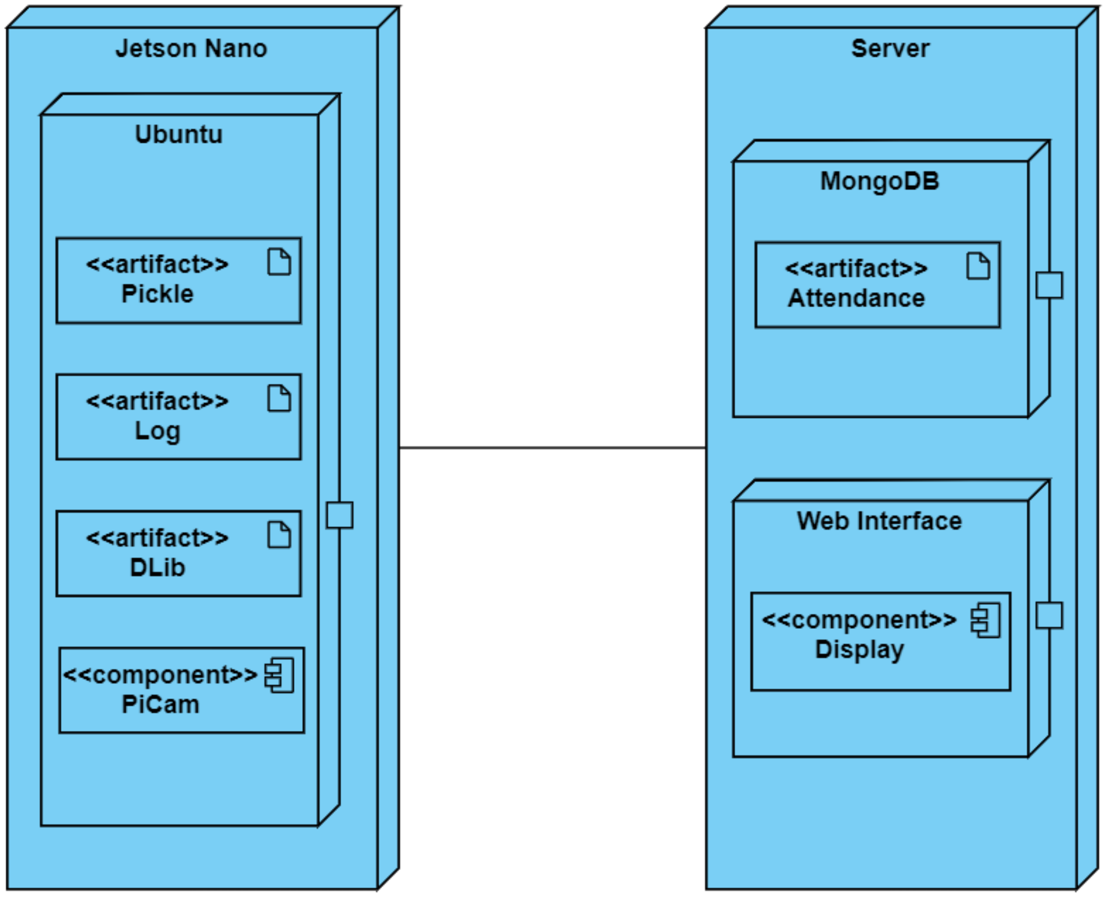
The server receives the attributes and the associated timestamps and then proceeds to match the timestamp to the period that was progressing within which the timestamp fell; a positive match will have the attendance of the student for that subject marked present; else they are marked absent and the records are tallied. On being queried an updated version of the records are presented to the Administrator.

4.2 UML CLASS DIAGRAM



The students have their facial features scanned and the metrics are stored and used for training, with the help of strategically placed edge devices, cameras, their faces are captured at times determined by the management/staff. This is aided by timestamps, the captured snapshots are checked for matching vectors from the trained database and if a positive match is identified, then a message is sent to the server along with the time-stamp. The management needn't mark the attendance now as it is automatically logged in. This auto tabulation reduces for human interfacing thereby achieving the stated objective.

4.3 UML DEPLOYMENT DIAGRAM



CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 Folder Structure in Edge Device

Project Directory

1. Dataset
   a. Name_1
   b. :
   c. Name_n
2. Farfasa
a. __init__.py
   b. FarfasaCore.py
   c. FarfasaDetect.py
   d. FarfasaRecog.py
2. Encode.py
3. AddEncode.py
4. FaceRecog.py
5. Setup.py
6. Requirements.txt

1. Dataset <directory>

Contains subfolders which in turn contain imagesto train. Each subfolder is named after the subject for which it contains training images. The training images have to be of formats jpg, jpeg and png. Files of any other format will be ignored.

1. Farfasa <directory>

This module forms the backend of the whole application and consists of 4 files of which one is an initialization file. It uses dlib to get neural networks based on tensorflow and uses models to find faces in images and then convert the faces into vectors and also does other operations like comparisons and distance calculation.

a. __init__.py

Used to initialize the Farfasa module. Also defines the Authors, Mentor, Industry Mentor and company of the project. The code has an import statement to load all necessary components of FarfasaCore.

b. FarfasaCore.py

Contains core functions of Farfasa module. The file imports Python Image Library, numpy, dlib and face_recognition_models. It first creates global objects for different models and predictors and then defines a multitude of functions most of which are explained below.

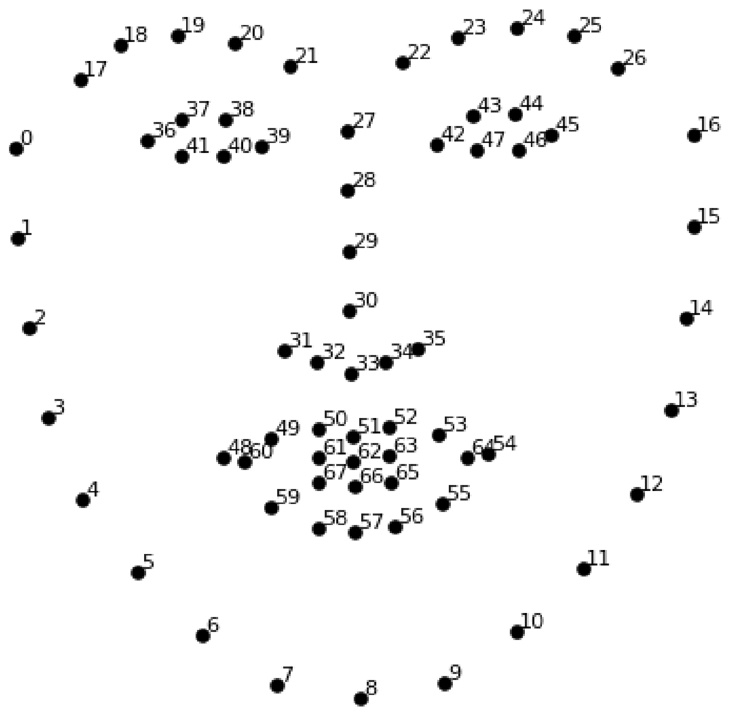
- compareFaces

Compare a list of known face encodings against a candidate with a given tolerance limit (0.6 default) and return a boolean list indicating match of candidate to a face.

- faceLandmarks

Returns a dict of face feature locations for each face in the given image. The feature locations are based on the picture below.:

Source : https://cdn-images-1.medium.com/max/1600/1*AbEg31EgkbXSQehuNJBlWg.png



- faceEncodings

  Returns the 128-dimension face encoding for each face in the given image in the form of a list of lists. It also has a parameter called numberOfJitters to control the number of times to resample the image for faces. A greater value increases accuracy at the cost of speed.

- faceLandmarksRaw

  Uses pose_predictor of dlib and face_recognition_models to identify face landmarks and returns the result as a list. It also has a parameter model which can be used for a tradeoff between speed and accuracy.

- rawFaceLocationsBatched

  Returns an 2d array of dlib rects of human faces in an image using the cnn face detector. It gets inputs as a list of images as numpy arrays and a number of scans. A larger number of scans can increase accuracy at the cost of speed.

- faceLocationsBatched

  Uses the cnn face detector to return a 2d array of bounding boxes of human faces in an image.
  It gets inputs as an image list, number of scans and a batch size. The batch size can be used to optimize functioning using GPUs.

- faceLocations

  Returns an array of bounding boxes of human faces in an image when we give inputs as an image as a numpy array, number of scans and model. Model can be hog or cnn. Hog is processing wise cheaper but also less accurate.

- imgLoad

  Loads an image file (.jpg / .png / .jpeg) into a numpy array after reading using PIL and converting to given mode. (default mode is RGB)

- rawFaceLocations

Returns an array of bounding boxes of human faces in an image when we give inputs as an image as a numpy array, number of scans and model. Model can be hog or cnn. Hog is processing wise cheaper but also less accurate.

- faceDist

  Compare Face to list of known faces and get a euclidean distance for each comparison. The distance tells you how similar the faces are. The function returns numpy ndarray with the distance for each face comparison in the same order as the 'faces' array.

- rect2Css

  Convert a dlib 'rect' object to a plain tuple in (top, right, bottom, left) order.

- css2Rect

  Convert a tuple in (top, right, bottom, left) order to a dlib `rect` object.

- trimCss

  Check if tuple is within the bounds of the image and return a trimmed plain tuple representation of the rect in (top, right, bottom, left) order.

c. FarfasaDetect.py

This file deals with the detection of faces in an image. It also uses the python multiprocessing module to make use of GPUs and multicore CPUs. It’s possible to control this number but the default option is to use the maximum available amount. It can work with both hog and cnn methodologies and has many functions within it to help it function. The functions are:

- Main

  run face detection on given images and then quit. Get image to check, number of CPUs (default = max) and model (hog / cnn) and then test each image after assigning the sub tasks to process pools.

- processPoolProcessImgs

  To multiprocess images for efficiency based on number of CPUs. The function uses multiprocessing module to stream all images to check to testImg function.

- folderImages

  Stream all jpeg , jpg , png images from a folder and return them as a list.

- Printing
  Print the locations of detected faces in the given file.

- testImg

  Test given image for face locations and prints locations of any faces in the given image file.

d. FarfasaRecog.py

This file deals with the recognising detected faces in an image. It also uses the python multiprocessing module to make use of GPUs and multicore CPUs. It’s possible to control this number but the default option is to use the maximum available amount. It can work with both hog and cnn methodologies and has many functions within it to help it function. The functions are :

- Main

  run face recognition on given images and then quit. Get an image to check, number of CPUs (default = max) and model (hog / cnn) and then test each image after assigning the sub tasks to process pools.

- processPoolProcessImgs

  To multiprocess images for efficiency based on number of CPUs. The function uses multiprocessing module to stream all images to check to testImg function.

- folderImages

Stream all jpeg , jpg , png images from a folder and return them as a list.

- ScanKnownPpl

  Scan known people from a given folder and return known faces and their names.

- Printing

  Print the names of detected faces and print unknown for missing faces.

- testImg

  Test given image for face locations and prints locations of any faces in the given image file.

1. Encode.py

   Gets command line arguments dataset, encodings and method.
   Scans the given dataset folders for subfolders and considers each subfolder to be a separate face and trains using the images inside those subfolders. Trains all images and then dumps it into the given encodings file as a pickle. Replaces any existing data in the file if it already exists without warning. The method can be either Hog or CNN. Hog is very fast but only takes 5 points for encoding which is much smaller than cnn which identifies 68 points from a face.

1. AddEncode.py

   Gets command line arguments dataset, encodings and method.
   Scans the given dataset folders for subfolders and considers each subfolder to be a separate face and trains using the images inside those subfolders. Trains facial feature vectors only for members which do not already have a face encoding and ignores faces which already have an encoding. It vectorizes new faces and appends it to the already existing list of faces and face encodings. It then dumps the new list into the given encodings file as a pickle. Replaces any existing data in the file if it already exists without warning. The method can be either hog or cnn. Hog is very fast but only takes 5 points for encoding which is much smaller than cnn which identifies 68 points from a face.

1. FaceRecog.py

   Requires an input of the encodings file where the pretrained model is. This file connects to a videostream and analyzes every even frame for faces and marks them on the screen. It also creates a dictionary during every frame with details such as current time in HMS format and then periodically posts a message to a server saying that person “X” was recognized at “Camera Y” at time “T”. These JSON posts are stored in a mongoDB in the server and all policies for attendance will be applied and results queried by a separate application not relating to this project.

1. Setup.py

   A python file with setuptools to install the farfasa module directly into your python base environment or into a virtual environment. It can also be used to create a development environment for the current project alone. It can be used using command “python setup.py install” to install in base or virtual environment and “python setup.py develop” to create a development environment for the current project alone.

1. Requirements.txt

   A text file containing all required modules for Farfasa to run along with the versions of each module to avoid cross version confusion. Using the command “pip install -r ‘requirements.txt’ “ can be used to install all dependencies for the programs though using setup.py would install all dependencies along with Farfasa 1.0.1 as a module.

5.2 SERVER SIDE IMPLEMENTATION

Project Directory

- Db(dir)
- Web(dir)
- serverTest(dir)
- docker-compose.yml

DB

In this directory the docker file(DockerFile) necessary to create and initiate the db is present. The db can be logically accessed from this.

Web

- DockerFile
- App.py
- requirements.txt

DockerFile

Contains the instructions to create, install and initiate the event driven server framework is present.

App.py

Event driven code with the necessary url path to receive the attendance data as a JSON file, register a new user and also query the db. Once a proper schedule is obtained a new document shall be created for recording those appropriately.

Requirements.txt

The required python modules are mentioned here and they are all installed from here.

Docker-Compose.yml

This file contains the instructions to create the entire server environment with all the dependencies satisfied.

CHAPTER 6

TESTING

6.1 FILE : encode.py and addencode.py

S.NO	TEST SCENARIOS	TEST CASES	EXPECTED RESULTS	OBSERVED RESULTS
1.	Out of disk space	The server has run out of disk space.	MongoDB throws an exception which can't take write-lock while out of disk space.	Error 404 message is displayed.
2.	Ideal situation	Server receives a properly formatted pathname.	Displays the query's results.	Displays the query's results.
3.	Ill-formatted but right pathname is sent	Correct pathname receives ill-formatted data.	Error is thrown.	The server receives and stores the data.

6.2 FILE: setup.py

S.NO	TEST SCENARIOS	TEST CASES	EXPECTED RESULTS	OBSERVED RESULTS
1.	Out of disk space	The server has run out of disk space.	MongoDB throws an exception which can't take write-lock while out of disk space.	Error 404 message is displayed.
2.	Ideal situation	Server receives a properly formatted pathname.	Displays the query's results.	Displays the query's results.

3.	Ill-formatted but right pathname is sent	Correct pathname receives ill-formatted data.	Error is thrown.	The server receives and stores the data.

6.3 FILE : facerecog.py

S.NO	TEST SCENARIOS	TEST CASES	EXPECTED RESULTS	OBSERVED RESULTS
1.	Unknown person is detected	Previously unknown face is detected in the video stream.	Stores the face so as to intimate the the administrator of the unknown face.	Prints “unknown” and no message is sent to the server.
2.	Known face with drastic changes to the facial appearance	Person who has grown a thick beard whilst being clean shaven during the training period.	Recognizes the face and sends a corresponding message to the server.	Classifies the face as “unknown”.
3.	Known face with no changes to the facial appearance	Person with minimal to no changes to their facial appearance.	Recognizes the face and sends a corresponding message to the server.	Recognizes the face and sends a corresponding message to the server.

6.4. FaRFASA MODULE

S.NO	TEST SCENARIOS	TEST CASES	EXPECTED RESULTS	OBSERVED RESULTS
1.	Unknown person is detected	Previously unknown faces are detected in the video stream.	Stores the face so as to intimate the administrator of the unknown face.	Prints “unknown” and no message is sent to the server.

2.	Known face with drastic changes to the facial appearance	Person who has grown a thick beard whilst being clean shaven during the training period.	Recognizes the face and sends a corresponding message to the server.	Classifies the face as “unknown”.
3.	Known face with no changes to the facial appearance	Person with minimal to no changes to their facial appearance.	Recognizes the face and sends a corresponding message to the server.	Recognizes the face and sends a corresponding message to the server.

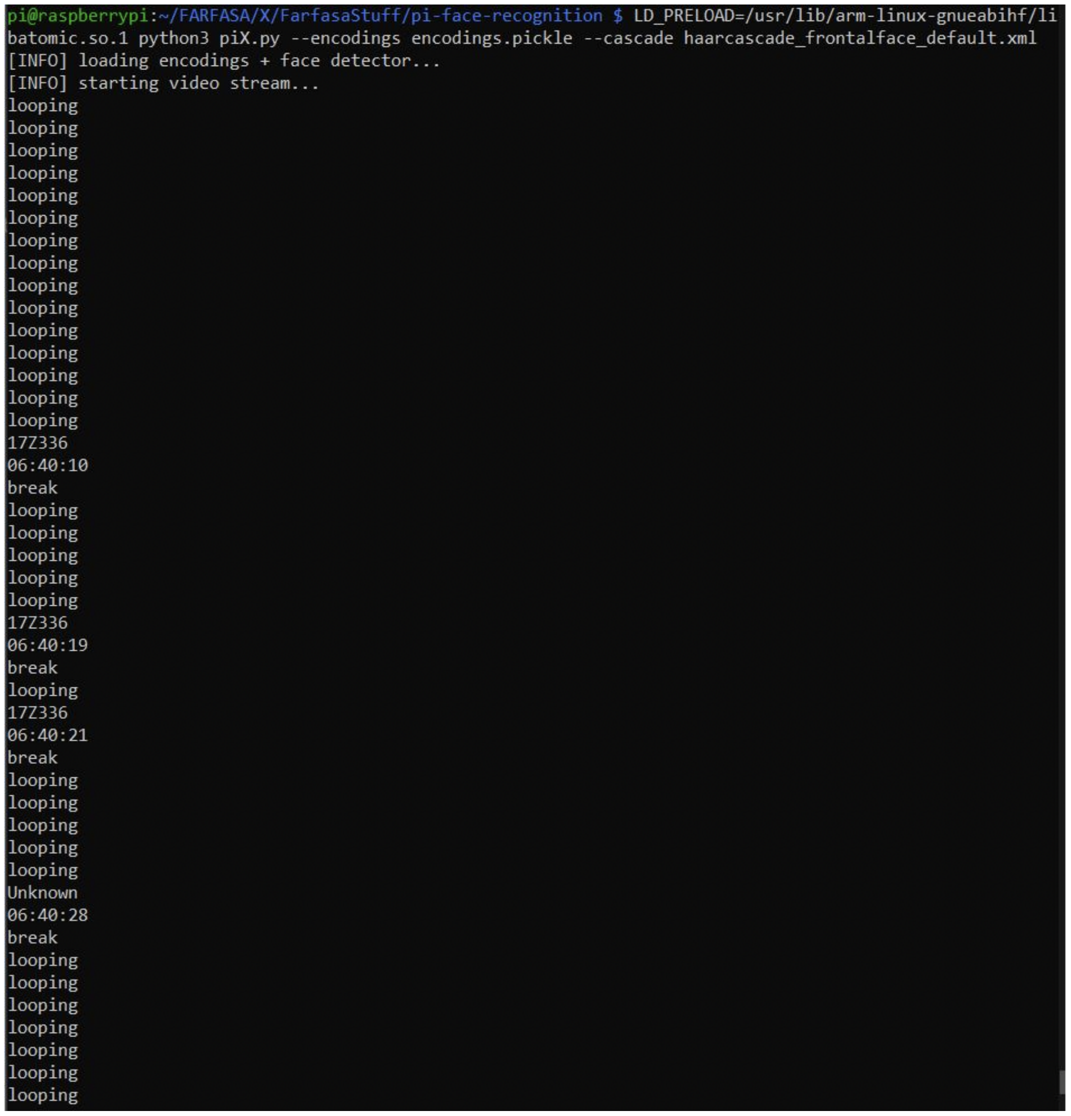
6.5 SERVER SIDE TESTING

S.NO	TEST SCENARIOS	TEST CASES	EXPECTED RESULTS	OBSERVED RESULTS
1.	Unknown person is detected	Previously unknown faces are detected in the video stream.	Stores the face so as to intimate the administrator of the unknown face.	Prints “unknown” and no message is sent to the server.
2.	Known face with drastic changes to the facial appearance	Person who has grown a thick beard whilst being clean shaven during the training period.	Recognizes the face and sends a corresponding message to the server.	Classifies the face as “unknown”.
3.	Known face with no changes to the facial appearance	Person with minimal to no changes to their facial appearance.	Recognizes the face and sends a corresponding message to the server.	Recognizes the face and sends a corresponding message to the server.

CHAPTER 7

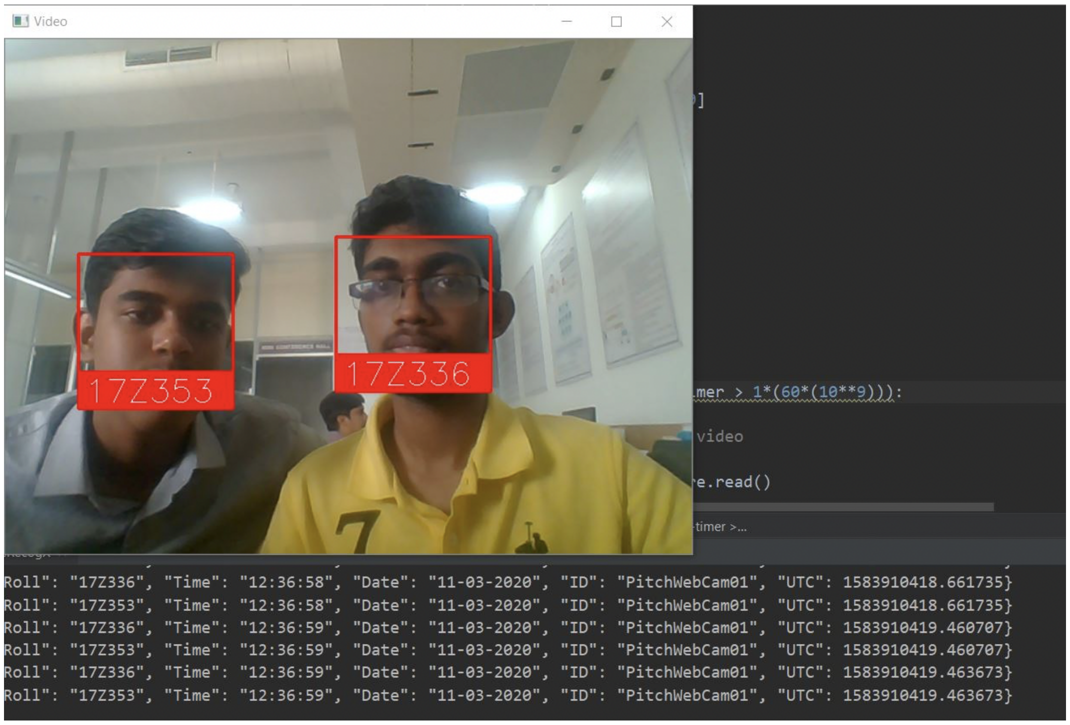
RESULT

7.1 Figure (a)



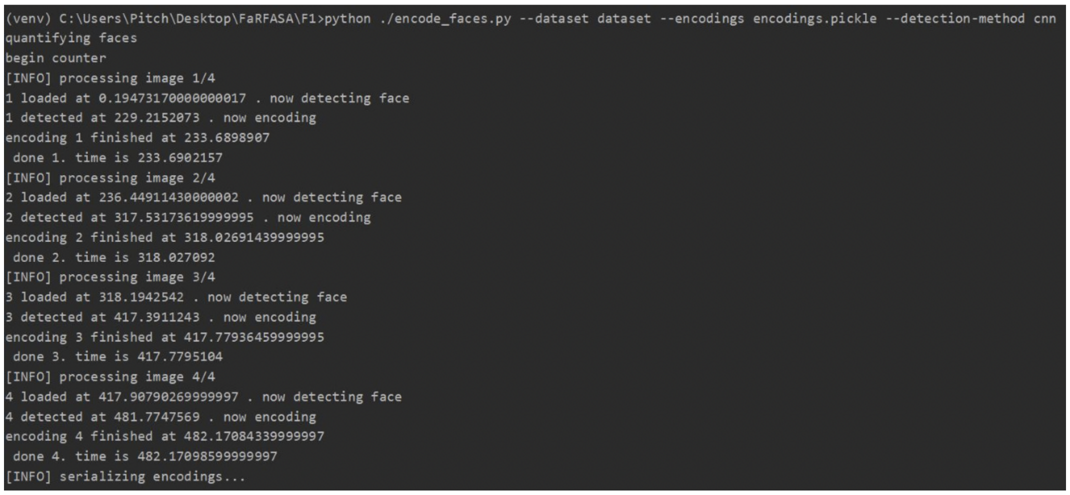
```
pi@raspberrypi:~/FARFASA/X/FarfasaStuff/pi-face-recognition $ LD_PRELOAD=/usr/lib/arm-linux-gnueabihf/li
batomic.so.1 python3 piX.py --encodings encodings.pickle --cascade haarcascade_frontalface_default.xml
[INFO] loading encodings + face detector...
[INFO] starting video stream...
looping
looping
looping
looping
looping
looping
looping
looping
looping
looping
looping
looping
looping
looping
looping
17Z336
06:40:10
break
looping
looping
looping
looping
looping
17Z336
06:40:19
break
looping
17Z336
06:40:21
break
looping
looping
looping
looping
looping
Unknown
06:40:28
break
looping
looping
looping
looping
looping
looping
looping
```

7.2 Figure (b)



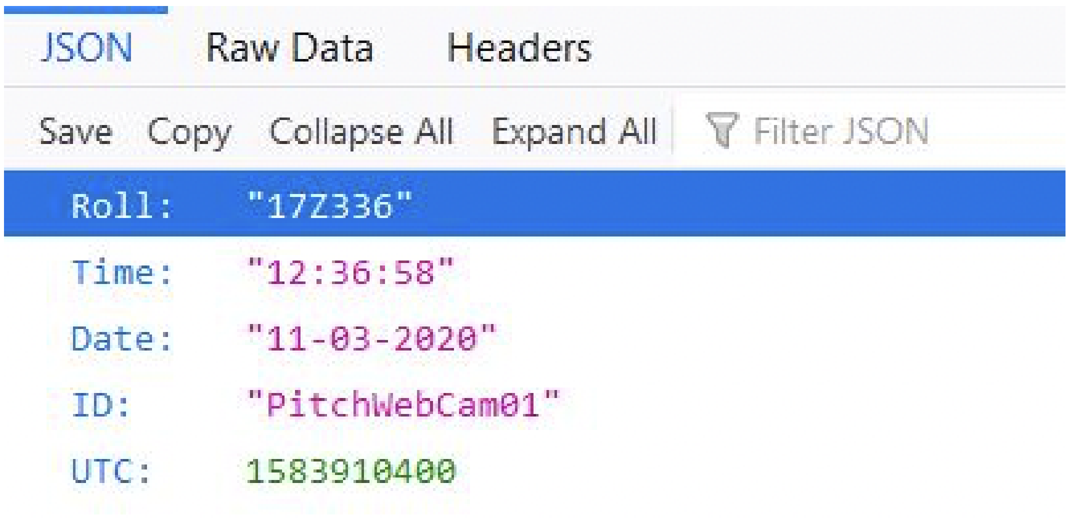
```
Roll": "17Z336", "Time": "12:36:58", "Date": "11-03-2020", "ID": "PitchWebCam01", "UTC": 1583910418.661735}
Roll": "17Z353", "Time": "12:36:58", "Date": "11-03-2020", "ID": "PitchWebCam01", "UTC": 1583910418.661735}
Roll": "17Z336", "Time": "12:36:59", "Date": "11-03-2020", "ID": "PitchWebCam01", "UTC": 1583910419.460707}
Roll": "17Z353", "Time": "12:36:59", "Date": "11-03-2020", "ID": "PitchWebCam01", "UTC": 1583910419.460707}
Roll": "17Z336", "Time": "12:36:59", "Date": "11-03-2020", "ID": "PitchWebCam01", "UTC": 1583910419.463673}
Roll": "17Z353", "Time": "12:36:59", "Date": "11-03-2020", "ID": "PitchWebCam01", "UTC": 1583910419.463673}
```

7.3 Figure (c)



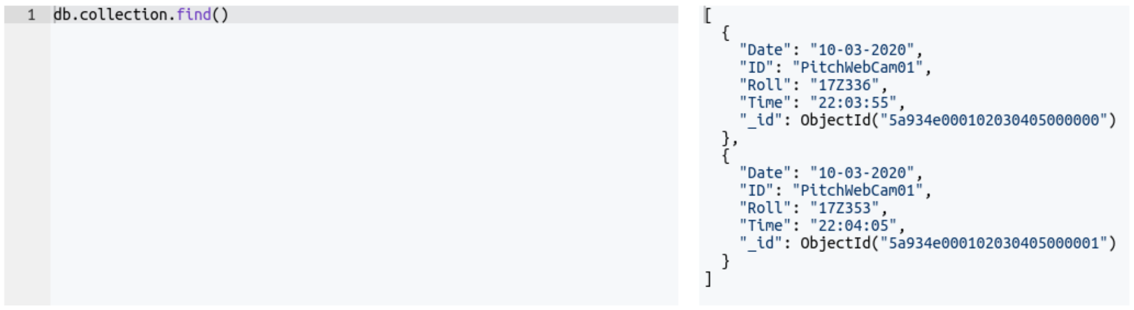
```
(venv) C:\Users\Pitch\Desktop\FaRFASA\F1>python ./encode_faces.py --dataset dataset --encodings encodings.pickle --detection-method cnn
quantifying faces
begin counter
[INFO] processing image 1/4
1 loaded at 0.19473170000000017 . now detecting face
1 detected at 229.2152073 . now encoding
encoding 1 finished at 233.6898907
 done 1. time is 233.6902157
[INFO] processing image 2/4
2 loaded at 236.44911430000002 . now detecting face
2 detected at 317.53173619999995 . now encoding
encoding 2 finished at 318.02691439999995
 done 2. time is 318.027092
[INFO] processing image 3/4
3 loaded at 318.1942542 . now detecting face
3 detected at 417.3911243 . now encoding
encoding 3 finished at 417.77936459999995
 done 3. time is 417.7795104
[INFO] processing image 4/4
4 loaded at 417.90790269999997 . now detecting face
4 detected at 481.7747569 . now encoding
encoding 4 finished at 482.17084339999997
 done 4. time is 482.17098599999997
[INFO] serializing encodings...
```

7.4 Figure (d)



```
Roll:    "17Z336"
Time:    "12:36:58"
Date:    "11-03-2020"
ID:      "PitchWebCam01"
UTC:     1583910400
```

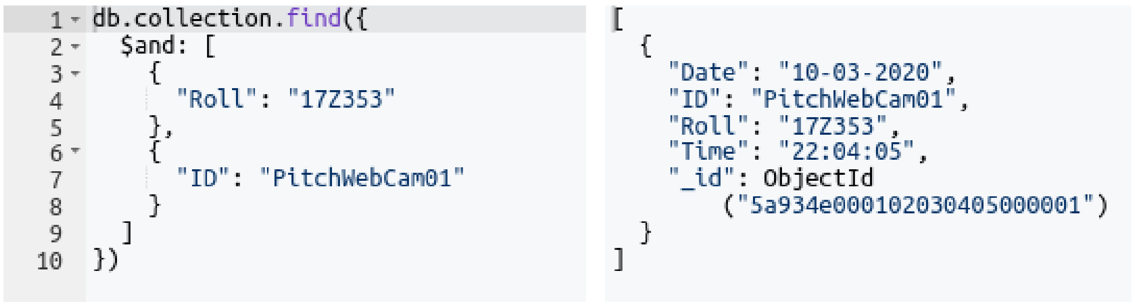
7.5 Figure (e)



```
db.collection.find()
```

```
[
  {
    "Date": "10-03-2020",
    "ID": "PitchWebCam01",
    "Roll": "17Z336",
    "Time": "22:03:55",
    "_id": ObjectId("5a934e000102030405000000")
  },
  {
    "Date": "10-03-2020",
    "ID": "PitchWebCam01",
    "Roll": "17Z353",
    "Time": "22:04:05",
    "_id": ObjectId("5a934e000102030405000001")
  }
]
```

7.6 Figure (f)



```
db.collection.find({
  $and: [
    {
      "Roll": "17Z353"
    },
    {
      "ID": "PitchWebCam01"
    }
  ]
})
```

```
[
  {
    "Date": "10-03-2020",
    "ID": "PitchWebCam01",
    "Roll": "17Z353",
    "Time": "22:04:05",
    "_id": ObjectId
        ("5a934e000102030405000001")
  }
]
```

CHAPTER 8

CONCLUSION AND FUTURE WORKS

The project was successfully completed and able to be run on MacOS, Windows and Linux platforms whilst meeting stated functional and non-functional requirements.The server was dockerized to meet parameter objectives. The future course of the project will be to mount the same on a Jetson Nano, an edge device, and run the project to fruition.

The working of the project hinges on the following factors:

a. Unchanged facial features of the subject since training period.
b. Lighting
c. Processing power

The above factors can be negotiated and negated in entirety by the proper positioning of edge devices.

APPENDIX

EDGE DEVICE CODE

```
# May
The
Force Be
With You

           # Team FARFASA

           #
           ---------------------------------------------------------------------------
           -

           __author__          = "Team FaRFASA"
           __version__         = '1.0.1'
           __members__         = "Aadhav Krishna S, Balaji U, Pitchappan P RM, Srihari
           M, Tarun Aravind R"
           __mentor__          = "Dr. Sudhasadhasivam"
           __industryMentor__  = "Mr. Vijay Jayaseelan"
           __company__         = "PSG Software Technologies"

           from .FarfasaCore import imgLoad, faceLocations, faceLocationsBatched,
           faceLandmarks, faceEncodings, compareFaces, faceDist

           import PIL.Image
           import dlib
           import numpy as np
           from PIL import ImageFile
           import face_recognition_models

           ImageFile.LOAD_TRUNCATED_IMAGES = True

           faceDetector = dlib.get_frontal_face_detector()

           predictor68pt = face_recognition_models.pose_predictor_model_location()
           predictorPose68pt = dlib.shape_predictor(predictor68pt)

           predictor5pt =
           face_recognition_models.pose_predictor_five_point_model_location()
           predictorPose5pt = dlib.shape_predictor(predictor5pt)

           cnnFaceDetectModel =
           face_recognition_models.cnn_face_detector_model_location()
           cnnFaceDetector = dlib.cnn_face_detection_model_v1(cnnFaceDetectModel)

           faceRecogModel = face_recognition_models.face_recognition_model_location()
           faceEncoder = dlib.face_recognition_model_v1(faceRecogModel)


           def compareFaces(knownFaceEncodings, candidate, tolerance=0.6):
               # Compare a list of face encodings against a candidate

               # PARAMETERS
               # knownFaceEncodings: A list of known face encodings
               # candidate: A single face encoding to compare against the list
```

```
    # tolerance: How much distance between faces to consider it a match.
Lower = Stricter

    # Return list of T/F values indicating which knownFaceEncodings match
the face encoding to check

    return list(faceDist(knownFaceEncodings, candidate) <= tolerance)


def faceLandmarks(faceImg, faceLocations=None, model="large"):
    # Returns a dict of face feature locations for each face in the given
image

    # PARAMETERS
    # faceImg           image to search
    # faceLocations     Optionally provide a list of face locations to
check.
    # model             "large" (default) or "small" which only returns 5
points but is faster

    # Returns list of dicts of face feature locations

    landmarks = faceLandmarksRaw(faceImg, faceLocations, model)
    landmarksTuples = [[(p.x, p.y) for p in _.parts()] for _ in landmarks]

    # For a definition of each point index
    # https://cdn-images-1.medium.com/max/1600/1*AbEg31EgkbXSQehuNJBlWg.png

    if model == 'large':
        return [{
            "chin": points[0:17],
            "left_eyebrow": points[17:22],
            "right_eyebrow": points[22:27],
            "nose_bridge": points[27:31],
            "nose_tip": points[31:36],
            "left_eye": points[36:42],
            "right_eye": points[42:48],
            "top_lip": points[48:55] + [points[64]] + [points[63]] +
[points[62]] + [points[61]] + [points[60]],
            "bottom_lip": points[54:60] + [points[48]] + [points[60]] +
[points[67]] + [points[66]] + [points[65]] + [
                points[64]]
        } for points in landmarksTuples]

    elif model == 'small':
        return [{
            "nose_tip": [points[4]],
            "left_eye": points[2:4],
            "right_eye": points[0:2],
        } for points in landmarksTuples]

    else:
        raise ValueError("Invalid landmarks model type. Supported models
are ['small', 'large'].")


def faceEncodings(faceImg, knownFaceLocations=None, numberOfJitters=1,
model="small"):
    # Return the 128-dimension face encoding for each face in given image.

    # PARAMETERS
    # faceImg: input image
    # knownFaceLocations: known bounding boxes of faces
    # numberOfJitters: How many times to re-sample the face when
calculating encoding
    # model: "large" (default) or "small" which only returns 5 points but
is faster.

    # Returns list of 128-dimensional face encodings (one for each face in
the image)

    raw_landmarks = faceLandmarksRaw(faceImg, knownFaceLocations, model)
    return [np.array(faceEncoder.compute_face_descriptor(faceImg, _,
numberOfJitters)) for _ in raw_landmarks]


def faceLandmarksRaw(faceImg, faceLocations=None, model="large"):
    # Uses pose_predictor of dlib and face_recognition_models to identify
face landmarks

    # PARAMETERS
    # faceImg           image containing face
    # faceLocations     locations of face
    # model             small/large

    # Returns list of pose_predictors

    if faceLocations is None:
        faceLocations = rawFaceLocations(faceImg)
    else:
        faceLocations = [css2Rect(_) for _ in faceLocations]

    pose_predictor = predictorPose68pt

    if model == "small":
        pose_predictor = predictorPose5pt

    return [pose_predictor(faceImg, _) for _ in faceLocations]


def rawFaceLocationsBatched(imageList, numberOfScans=1, batchSize=128):
    # Returns an 2d array of dlib rects of human faces in a image using the
cnn face detector

    # PARAMETERS
    # imageList         A list of numpy array images
    # numberOfScans     How many times to scan the image looking for faces

    # Returns list of dlib 'rect' objects of face locations

    return cnnFaceDetector(imageList, numberOfScans, batch_size=batchSize)


def faceLocationsBatched(imageList, numberOfScans=1, batchSize=128):
    # Uses the cnn face detector to return a 2d array of bounding boxes of
human faces in an image

    # PARAMETERS
    # imageList         A list of imageList (each as a numpy array)
    # numberOfScans     How many times to scan the image looking for faces
    # batchSize         How many imageList to include in each GPU
processing batch.
```

```
    # Returns list of tuples of found face locations in css

    def cnn2Css(detections):
        return [trimCss(rect2Css(_.rect), imageList[0].shape) for _ in
detections]

    rawDetectionsBatched = rawFaceLocationsBatched(imageList,
numberOfScans, batchSize)

    return list(map(cnn2Css, rawDetectionsBatched))


def faceLocations(img, numberOfScans=1, model="hog"):
    # Returns an array of bounding boxes of human faces in a image

    # PARAMETERS
    # img                 Image as numpy array
    # numberOfScans       How many times to scan the image for faces
    # model               hog or cnn

    # Returns list of tuples of found face locations in css

    if model == "cnn":
        return [trimCss(rect2Css(_.rect), img.shape) for _ in
rawFaceLocations(img, numberOfScans, "cnn")]
    else:
        return [trimCss(rect2Css(_), img.shape) for _ in
rawFaceLocations(img, numberOfScans, model)]


def imgLoad(file, mode='RGB'):
    # Loads an image file (.jpg / .png / .jpeg) into a numpy array

    # PARAMETERS
    # file      image file name or file object to load
    # mode      format to convert the image to.

    # Returns image as numpy array

    im = PIL.Image.open(file)
    if mode:
        im = im.convert(mode)
    return np.array(im)


def rawFaceLocations(img, numberOfScans=1, model="hog"):
    # Returns an array of bounding boxes of human faces in a image

    # PARAMETERS
    # img                 Image as numpy array
    # numberOfScans       How many times to scan the image for faces
    # model               hog or cnn

    # Returns list of dlib 'rect' objects of found face locations

    if model == "cnn":
        return cnnFaceDetector(img, numberOfScans)
    else:
        return faceDetector(img, numberOfScans)
```

```
def faceDist(faceEncodings, comparisionFace):
    # Compare Face to list of known faces and get a euclidean distance for
each comparision
    # The distance tells you how similar the faces are.

    # PARAMETERS
    # faceEncodings             List of face encodings to compare
    # comparisionFace           A face encoding to compare against

    # Returns numpy ndarray with the distance for each face comparision in
the same order as the 'faces' array

    if len(faceEncodings) == 0:
        return np.empty((0))

    return np.linalg.norm(faceEncodings - comparisionFace, axis=1)


def rect2Css(rect):
    # Convert a dlib 'rect' object to a plain tuple in (top, right, bottom,
left) order

    # PARAMETERS
    # rect      dlib rect object

    # Returns a tuple of (top, right, bottom, left)

    return rect.top(), rect.right(), rect.bottom(), rect.left()


def css2Rect(css):
    # Convert a tuple in (top, right, bottom, left) order to a dlib `rect`
object

    # PARAMETERS
    # css       a tuple of (top, right, bottom, left)

    # Returns a dlib `rect` object

    return dlib.rectangle(css[3], css[0], css[1], css[2])


def trimCss(css, imgShape):
    # Check if tuple is within the bounds of the image.

    # PARAMETERS
    # css               plain tuple representation of the rect
    # image_shape       numpy shape of the image array

    # Returns a trimmed plain tuple representation of the rect in (top,
right, bottom, left) order

    return max(css[0], 0), min(css[1], imgShape[1]), min(css[2],
imgShape[0]), max(css[3], 0)


from __future__ import print_function
                                        import click
                                        import os
                                        import re
import farfasa.FarfasaCore as FarfasaCore
import multiprocessing
import sys
import itertools


@click.command()
@click.argument('img2Check')
@click.option('--numCPU', default=1,
help='number of CPU cores to use. -1 = max')
@click.option('--model', default="hog",
help='face detection model hog / cnn.')

def main(img2Check, numCPU, model):
    # Main driver function

    # PARAMETERS
    # img2Check         image to check for
faces
    # numCPU            number of CPUs to use.
-1 = max
    # model             hog / cnn

    # run face detection on given images and
quit

    # Multi-core processing only supported on
Python 3.4 or greater
    if (sys.version_info < (3, 4)) and numCPU
!= 1:
        numCPU = 1

    if os.path.isdir(img2Check):
        if numCPU == 1:
            [testImg(image_file, model) for
image_file in folderImages(img2Check)]
        else:

processPoolProcessImgs(folderImages(img2Check)
, numCPU, model)
    else:
        testImg(img2Check, model)


def processPoolProcessImgs(images2Check,
numCPU, model):
    # To multiprocess images for efficiency
based on number of CPUs

    # PARAMETERS
    # images2Check      images to check for
face locations
    # numCPU            number of CPUs to use
. -1 = max
    # model             cnn / hog

    # uses multiprocessing module to stream
all images to check to testImg function

    if numCPU == -1:
        processes = None
    else:
        processes = numCPU

    context = multiprocessing
    if "forkserver" in
multiprocessing.get_all_start_methods():
        context =
multiprocessing.get_context("forkserver")

    pool = context.Pool(processes=processes)

    function_parameters = zip(
        images2Check,
        itertools.repeat(model),
    )

    pool.starmap(testImg, function_parameters)


def folderImages(folder):
    # stream all jpeg , jpg , png images from
a folder

    # PARAMETERS
    # folder        path of folder to Scan

    # returns list of all images in folder
with extensions jpeg, jpg, png

    return [os.path.join(folder, f) for f in
os.listdir(folder) if
re.match(r'.*\.(jpg|jpeg|png)', f,
flags=re.I)]


def printing(file, location):
    # Print the locations of detected faces

    # PARAMETERS
    # file          file where person was
recognized
    # location      location of detected face

    # prints and then exits function

    top, right, bottom, left = location
    print("{},{},{},{},{}".format(file, top,
right, bottom, left))


def testImg(img2Check, model):
    # Test given image for face locations

    # PARAMETERS
    # img2Check     image to scan and detect
face locations
    # model         cnn / hog

    # prints locations of any faces in the
given image image_file
```

```
    unknownImg =
FarfasaCore.imgLoad(img2Check)
    faceLocations =
FarfasaCore.faceLocations(unknownImg,
numberOfScans=0, model=model)

    for Location in faceLocations:
        printing(img2Check, Location)


if __name__ == "__main__":
    main()

from __future__ import print_function

import click
import os
import re
import farfasa.FarfasaCore as FarfasaCore
import multiprocessing
import itertools
import sys
import PIL.Image
import numpy as np


@click.command()
@click.argument('folder')
@click.argument('img2Check')
@click.option('--numCPU', default=1,
help='number of CPU cores to use . -1 = max')
@click.option('--tolerance', default=0.6,
help='Tolerance for face comparisons. lower =
stricter')
@click.option('--showDist', default=False,
type=bool, help='Output face distance')
def main(folder, img2Check, numCPU, tolerance,
showDist):
    # Main driver function

    # PARAMETERS
    # folder            folder containing
known faces and names
    # img2Check         image to compare known
list with
    # numCPU            number of CPUs to use.
-1 = max
    # tolerance         tolerance . lower =
stricter
    # showDist          boolean option to show
distance

    # run face recoginition on given images
and quit

    known_names, known_face_encodings =
scanKnownPpl(folder)

    # Multi-core processing only supported on
Python 3.4 or greater
    if (sys.version_info < (3, 4)) and numCPU
!= 1:
        numCPU = 1

    if os.path.isdir(img2Check):
        if numCPU == 1:
            [testImg(image_file, known_names,
known_face_encodings, tolerance, showDist) for
image_file in
             folderImages(img2Check)]
        else:

processPoolProcessImgs(folderImages(img2Check)
, known_names, known_face_encodings, numCPU,
tolerance,
                                   showDist)
    else:
        testImg(img2Check, known_names,
known_face_encodings, tolerance, showDist)


def folderImages(folder):
    # stream all jpeg , jpg , png images from
a folder

    # PARAMETERS
    # folder        path of folder to Scan

    # returns list of all images in folder
with extensions jpeg, jpg, png

    return [os.path.join(folder, f) for f in
os.listdir(folder) if
re.match(r'.*\.(jpg|jpeg|png)', f,
flags=re.I)]


def processPoolProcessImgs(images2Check,
knownNames, knownFaces, numCPU, tolerance,
showDist):
    # To multiprocess images for efficiency
based on number of CPUs

    # PARAMETERS
    # images2Check      images to check for
faces
    # knownNames        list of known names
    # knownFaces        list of known face
encodings
    # numCPU            number of CPUs to use
. -1 = max
    # tolerance         tolerance . lower =
stricter
    # showDist          boolean option to show
distance

    # uses multiprocessing module to stream
all images to check to testImg function

    if numCPU == -1:
        processes = None
    else:
        processes = numCPU
    context = multiprocessing
    if "forkserver" in
multiprocessing.get_all_start_methods():
        context =
multiprocessing.get_context("forkserver")

    pool = context.Pool(processes=processes)

    function_parameters = zip(
        images2Check,
        itertools.repeat(knownNames),
        itertools.repeat(knownFaces),
        itertools.repeat(tolerance),
        itertools.repeat(showDist)
    )

    pool.starmap(testImg, function_parameters)


def scanKnownPpl(folder):
    # Scan known people from given folder

    # PARAMETERS
    # folder        path to folder where faces
are there

    # returns known faces and their names

    knownNames = []
    knownFaces = []

    for file in folderImages(folder):
        basename =
os.path.splitext(os.path.basename(file))[0]
        img = FarfasaCore.imgLoad(file)
        encodings =
FarfasaCore.faceEncodings(img)

        if len(encodings) > 1:
            click.echo("WARNING: More than one
face found in {}. Considering first
face.".format(file))

        if len(encodings) == 0:
            click.echo("WARNING: No faces
found in {}. Ignoring file.".format(file))
        else:
            knownNames.append(basename)
            knownFaces.append(encodings[0])

    return knownNames, knownFaces


def printing(file, name, distance,
showDist=False):
    # Print the names of recognized people

    # PARAMETERS
    # file          file where person was
recognized
```

```
    # name          name of recognized guy
    # distance      distance of recognized
person
    # showDist      option to show distance (
boolean )

    # prints and then exits function

    if showDist:
        print("{},{},{}".format(file, name,
distance))
    else:
        print("{},{}".format(file, name))


def testImg(img2Check, knownNames, knownFaces,
tolerance=0.6, showDist=False):
    # Test the image for people by finding
faces and comparing them to known faces.

    # PARAMETERS
    # img2Check         image as numpy array
to scan
    # knownNames        list of known names
    # knownFaces        list of known face
encodings
    # tolerance         tolerance level. lower
= stricter
    # showDist          boolean variable to
show distance

    # Test image for known people and print
then, print unknown if a person found but name
unknown

    unknownImg =
FarfasaCore.imgLoad(img2Check)

    # Scale down image for efficiency
    if max(unknownImg.shape) > 1600:
        pil_img =
PIL.Image.fromarray(unknownImg)
        pil_img.thumbnail((1600, 1600),
PIL.Image.LANCZOS)
        unknownImg = np.array(pil_img)

    unknownEnc =
FarfasaCore.faceEncodings(unknownImg)

    for unknown in unknownEnc:
        distances =
FarfasaCore.faceDist(knownFaces, unknown)
        result = list(distances <= tolerance)

        if True in result:
            [printing(img2Check, name, dist,
showDist) for isMatch, name, dist in
zip(result, knownNames, distances) if isMatch]
        else:
            printing(img2Check,
"unknown_person", None, showDist)
```

```
                    if not unknownEnc:
                        printing(img2Check,
                "no_persons_found", None, showDist)


                if __name__ == "__main__":
                    main()
from imutils import paths
import farfasa as farfasa
import argparse
import pickle
import cv2
import os
import time

# Argument Parser and 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())


# Path to input images in dataset folder
print("quantifying faces")
imagePaths = list(paths.list_images(args["dataset"]))

# Initialize the list of known encodings and known names
knownEncodings = []
knownNames = []
print("begin counter")
T = time.perf_counter()
# 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 RGB (dlib ordering)
       image = cv2.imread(imagePath)
       rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
       print("{} loaded at {} . now detecting
face".format(i + 1, (time.perf_counter()-T)))
       # Detect the (x, y)-coordinates of the bounding
boxes corresponding to each face in the input image
       boxes = farfasa.faceLocations(rgb,
model=args["detection_method"])
       print("{} detected at {} . now
encoding".format(i+1, (time.perf_counter()-T)))
       # Compute the facial embedding for the face
       encodings = farfasa.faceEncodings(rgb, boxes)
       print("encoding {} finished at {}".format(i+1,
(time.perf_counter()-T)))
```

```
        # 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)
        print(" done {}. time is {}".format(i+1,
(time.perf_counter()-T)))
# Dump the facial encodings + names to disk as a pickle
file
print("[INFO] serializing encodings...")
data = {"encodings": knownEncodings, "names": knownNames}

f = open(args["encodings"], "wb")
f.write(pickle.dumps(data))
f.close()
from imutils import paths

import farfasa as farfasa
import argparse
import pickle
import cv2
import os
import time

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())


print("quantifying faces")
imagePaths = list(paths.list_images(args["dataset"]))

key1 = []
val1 = []
data = pickle.loads(open(args["encodings"], "rb").read())
for key in data:
    key1.append(key)
    val1.append(data[key])

known_face_encodings = val1[0]
known_face_names = val1[1]

knownEncodings = known_face_encodings
knownNames = known_face_names
print("begin counter")
T = time.perf_counter()
# 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]

    if(name not in knownNames):
```

```
                # Load the input image and convert it from RGB
        (OpenCV ordering) to RGB (dlib ordering)
                image = cv2.imread(imagePath)
                rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                print("{} loaded at {} . now detecting
        face".format(i + 1, (time.perf_counter()-T)))
                # Detect the (x, y)-coordinates of the bounding
        boxes corresponding to each face in the input image
                boxes = farfasa.faceLocations(rgb,
        model=args["detection_method"])
                print("{} detected at {} . now
        encoding".format(i+1, (time.perf_counter()-T)))
                # Compute the facial embedding for the face
                encodings = farfasa.faceEncodings(rgb, boxes)
                print("encoding {} finished at {}".format(i+1,
        (time.perf_counter()-T)))
                # 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)
                print(" done {}. time is {}".format(i+1,
        (time.perf_counter()-T)))
            else:
                print("already there")

        # Dump the facial encodings + names to disk as a pickle
        file
        print("[INFO] serializing encodings...")
        data = {"encodings": knownEncodings, "names": knownNames}

        f = open(args["encodings"], "wb")
        f.write(pickle.dumps(data))
        f.close()
import
cv2
         import numpy as np
         import farfasa as farfasa
         import argparse
         import imutils
         import pickle
         import time as t
         from datetime import datetime,time,timedelta
         import requests
         import json

         #ap = argparse.ArgumentParser()
         #ap.add_argument("-e", "--encodings", required=True, help="path to serialized
         db of facial encodings")
         #args = vars(ap.parse_args())
         #input()

         def Attendance(args):
             timer = t.perf_counter_ns()
             key1 = []
             val1 = []
             video_capture = cv2.VideoCapture(0)
             print("loading encodings")
             data = pickle.loads(open(args["encodings"], "rb").read())
             for key in data:
```

```
        key1.append(key)
        val1.append(data[key])

    known_face_encodings = val1[0]
    known_face_names = val1[1]

    # Initialize variables
    face_locations = []
    face_names = []
    process_this_frame = True
    #AttDictList = []
    XXX = []
    Attendees = []
    Times = []

    while True:
        # change val to change timer
        if(t.perf_counter_ns()-timer > .25*(60*(10**9))):
            break

        TempDict = []
        # Grab a single frame of video
        ret, frame = video_capture.read()

        # Resize frame of video to 1/4 size for faster face recognition
processing
        small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

        # Convert the image from BGR color (which OpenCV uses) to RGB color
(which face_recognition uses)
        rgb_small_frame = small_frame[:, :, ::-1]

        # Only process every other frame of video to save time
        if process_this_frame:
            # Find all the faces and face encodings in the current frame of
video
            face_locations = farfasa.faceLocations(rgb_small_frame)
            face_encodings = farfasa.faceEncodings(rgb_small_frame,
face_locations)

            face_names = []
            for face_encoding in face_encodings:
                # See if the face is a match for the known face(s)
                matches = farfasa.compareFaces(known_face_encodings,
face_encoding)
                name = "Unknown"

                # # If a match was found in known_face_encodings, just use the
first one.
                # if True in matches:
                #     first_match_index = matches.index(True)
                #     name = known_face_names[first_match_index]

                # Or instead, use the known face with the smallest distance to
the new face
                face_distances = farfasa.faceDist(known_face_encodings,
face_encoding)
                best_match_index = np.argmin(face_distances)
                if matches[best_match_index]:
                    name = known_face_names[best_match_index]
                face_names.append(name)

        process_this_frame = not process_this_frame


        # Display the results
        for (top, right, bottom, left), name in zip(face_locations,
face_names):
            # Scale back up face locations since the frame we detected in was
scaled to 1/4 size
            top *= 4
            right *= 4
            bottom *= 4
            left *= 4

            # Draw a box around the face
            cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

            # Draw a label with a name below the face
            cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0,
255), cv2.FILLED)
            font = cv2.FONT_ITALIC
            cv2.putText(frame, name, (left + 6, bottom - 6), font, 1.0, (255,
255, 255), 1)
            Attendees.append(name)
            Times.append(datetime.now().strftime("%H:%M:%S"))
            X = json.dumps({"Roll": name, "Time":
datetime.now().strftime("%H:%M:%S"), "Date":
datetime.now().strftime("%d-%m-%Y"), "ID" : "PitchWebCam01", "UTC":
(datetime.utcnow()-datetime(1970,1,1)).total_seconds()})
            #AttDictList.append(X)
            if int(round(t.time()*1000))%5 == 0:
                XXX.append(X)
                SendToServer(X)
            #print(X)
        # Display the resulting image
        cv2.imshow('Video', frame)

        # Hit 'q' on the keyboard to quit!
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    # Release handle to the webcam
    video_capture.release()
    cv2.destroyAllWindows()

    def convertList(a):
        dict1 = {a[i]:'a' for i in range(0,len(lst))}
        return dict1

    lst = known_face_names

    AttendanceDict = convertList(lst)

    for i in lst:
        if i in Attendees:
            AttendanceDict[i] = 'p'
    a = args["encodings"]
    a = a.replace(".pickle", "")

    def getPeriod(S=None):
            if(8*60+0 <= S < 9*60+20):
                return 1
            elif(9*60+20 <= S < 10*60+10):
                return 2
            elif (10*60+30 <= S < 11*60+20):
                return 3
            elif (11*60+20 <= S < 12*60+10):
                return 4
            elif (13*60+40 <= S < 14*60+30):
                return 5
            elif (14*60+30 <= S < 15*60+20):
                return 6
            elif (15*60+30 <= S < 16*60+20):
                return 7
            elif (16*60+20 <= S < 17*60+10):
                return 8
            else:
                return 99



        times = Times
        T = str(timedelta(seconds=sum(map(lambda f: int(f[0])*3600 + int(f[1])*60
    + int(f[2]), map(lambda f: f.split(':'), times)))/len(times)))
        T1 = T.split(".")
        T = T1[0]

        def getTimeSec(T):
            h,m,s = T.split(":")
            return int(h)*60 + int(m)

        S = getTimeSec(T)

        P = getPeriod(S)
        #print(P)

        AttendanceDict["Class"] = str(P)+"-"+a
        AttendanceDict["Date"] = datetime.now().strftime("%d-%m-%Y")
        AttendanceDict["ID"] = "PitchWebCam01"
        #print(AttendanceDict)
        #print(AttDictList)
        SendToServer(AttendanceDict)
        #print(XXX)

    def SendToServer(Dict1):
        print(Dict1)
        # API_ENDPOINT = "111.111.111.111"
        # API_KEY = "XXXXXXXXXXXXXXXXXXXXXX"
        # User = "Cam1"
        # r = requests.post(url = API_ENDPOINT, json = Dict1, auth
    =(User,API_KEY))

    if __name__ == "__main__":
        args = {"encodings":"0"}
        #E = input()
        E = "encodings.pickle"
        args["encodings"] = E
        Attendance(args)
from
setuptools
```

```
import
setup

        requirements = [
            'dlib==19.19.0',
            'Click==7.0',
            'numpy==1.18.1',
            'face_recognition_models==0.3.0',
            'opencv_python==4.2.0.32',
            'requests==2.23.0',
            'imutils==0.5.3',
            'Pillow==7.1.1'
        ]

        setup(
            name='farfasa',
            version='1.0.1',
            description="FaceRecognition using dlib and
        face_recognition_models",
            author="Team farfasa",
            author_email='pitchappanprm.com',
            url='https://github.com/Pitch2342/Farfasa',
            packages=[
                'farfasa',
            ],
            package_dir={'farfasa': 'farfasa'},

            install_requires=requirements,
            license="GNU General Public License v3.0e",
            zip_safe=False,
            keywords='farfasa',
            classifiers=[
                'Development Status :: Beta',
                'Intended Audience :: Students',
                'Natural Language :: English',
                'Programming Language :: Python :: 3.5',
                'Programming Language :: Python :: 3.6',
                'Programming Language :: Python :: 3.7',
                'Programming Language :: Python :: 3.8',
            ],
        )
dlib==19.19.0
          Click==7.0
          numpy==1.18.1
          face_recognition_models==0.3.0
          opencv_python==4.2.0.32
          requests==2.23.0
          imutils==0.5.3
          Pillow==7.1.1
```