PROJECT REPORT

"**AUTOMATIC ATTENDANCE SYSTEM USING FACE RECOGNITION"**

Submitted by:
Dharshik G.S 2018504523
Sairushan A 2018504596
Sathiya Murthi S 2018504604

Reviewed By:
Dr. A.Viji
Department of Electronics Engineering
Anna University, MIT Campus
Chennai-600 044

January, 2022

Contents

Abstract

A facial recognition system is a technology capable of matching a human face from a digital image or a video frame against a database of faces, typically employed to authenticate users through ID verification services, works by pinpointing and measuring facial features from a given image.

Development began on similar systems in the 1960s, beginning as a form of computer application. Since their inception, facial recognition systems have seen wider uses in recent times on smartphones and in other forms of technology, such as robotics. Because computerized facial recognition involves the measurement of a human's physiological characteristics, facial recognition systems are categorized as biometrics. Although the accuracy of facial recognition systems as biometric technology is lower than iris recognition and fingerprint recognition, it is widely adopted due to its contactless process and this is very favourable during this pandemic situation.

Facial recognition systems have been deployed in advanced human-computer interaction, video surveillance and automatic indexing of images.

Objective and Motivation

To set up a contactless automatic attendance system using face recognition that can be easily deployed and implemented.
This would help in

- Eliminating chances of false attendance and missed entries
- Cheaper alternative to fingerprint authentication
- Ease of automation made available to Teachers and Employers
- Implementation option in online classes and sessions

Requirements

Software:

- Python
- MTCNN Face Recognition Library
- FaceNet by Keras
- TensorFlow
- OpenCV

For hardware implementation:

- Raspberry Pi
- Pi camera / Supported Camera
- Enclosure
- Server, Database

Python:

Python is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is popularly used for numerous applications in Image Processing and Machine Learning due to its support for multiple open source libraries that enables users to implement solutions for a variety of problems

TensorFlow:

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

OpenCV:

OpenCV (*Open Source Computer Vision Library*) 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 Apache 2 License.

Multi-task Cascaded Convolutional Networks (MTCNN):

MTCNN works based on the paper “Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks” by Zhang, Zhang and Zhifeng.

Multi-task Cascaded Convolutional Networks (MTCNN) is a framework developed as a solution for both face detection and face alignment. The process consists of three stages of convolutional networks that are able to recognize faces and landmark location such as eyes, nose, and mouth.

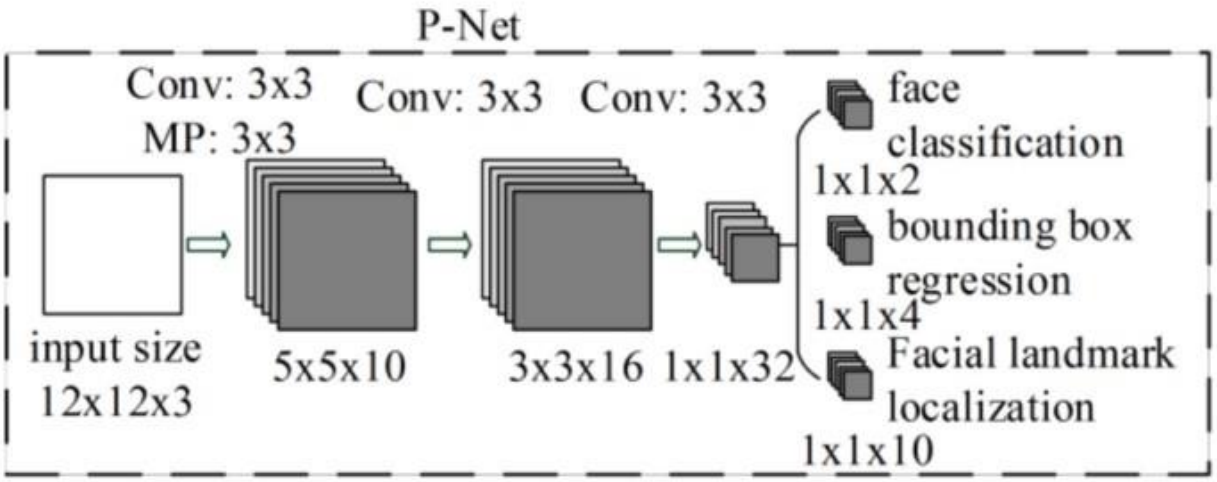
The paper proposes MTCNN as a way to integrate both tasks (recognition and alignment) using multi-task learning. In the first stage it uses a shallow CNN to quickly produce candidate windows. In the second stage it refines the proposed candidate windows through a more complex CNN. And lastly, in the third stage it uses a third CNN, more complex than the others, to further refine the result and output facial landmark positions.

The Three Stages of MTCNN:

The first step is to take the image and resize it to different scales in order to build an image pyramid, which is the input of the following three-staged cascaded network.

Stage 1: The Proposal Network (P-Net)

The Proposal Network is used to obtain candidate windows and their bounding box regression vectors.
Bounding box regression is a popular technique to predict the localization of boxes when the goal is detecting an object of some predefined class, in this case faces. After obtaining the bounding box vectors, some refinement is done to combine overlapping regions. The final output of this stage is all candidate windows after refinement to downsize the volume of candidates.

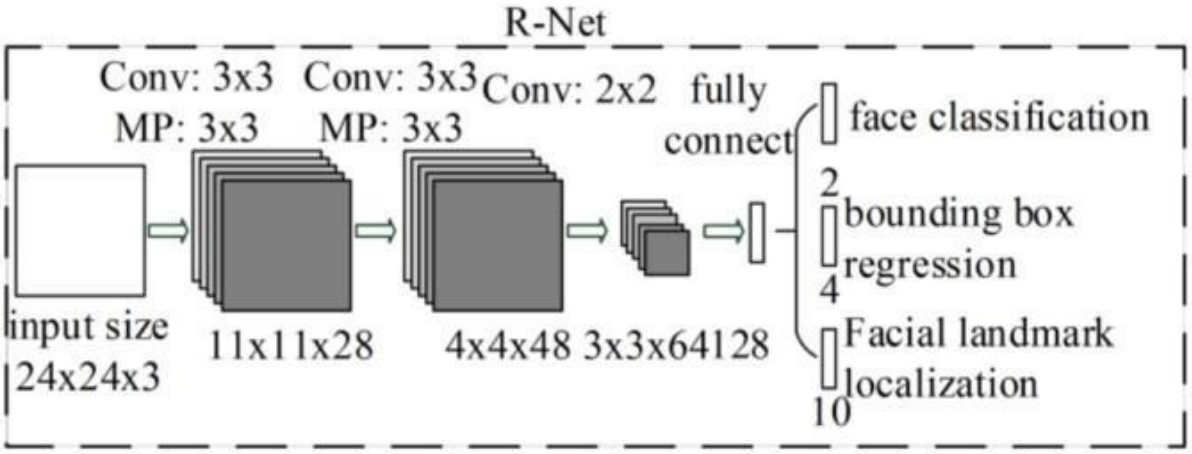


P-Net (from MTCNN paper)

Stage 2: The Refine Network (R-Net)

All candidates from the P-Net are fed into the Refine Network. Notice that this network is a CNN, not a FCN like the one before since there is a dense layer at the last stage of the network architecture. The R-Net further reduces the number of candidates, performs calibration with bounding box regression and employs non-maximum suppression (NMS) to merge overlapping candidates.

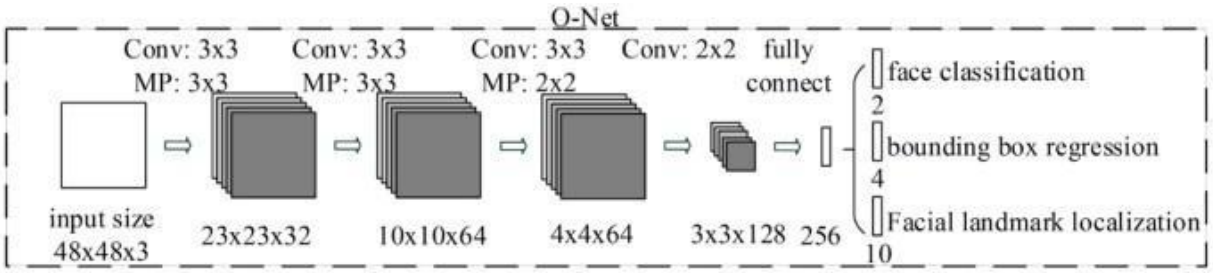
The R-Net outputs wether the input is a face or not, a 4 element vector which is the bounding box for the face, and a 10 element vector for facial landmark localization.



R-Net (from MTCNN paper)

Stage 3: The Output Network (O-Net)

This stage is similar to the R-Net, but this Output Network aims to describe the face in more detail and output the five facial landmarks' positions for eyes, nose and mouth.



O-Net (from MTCNN paper)

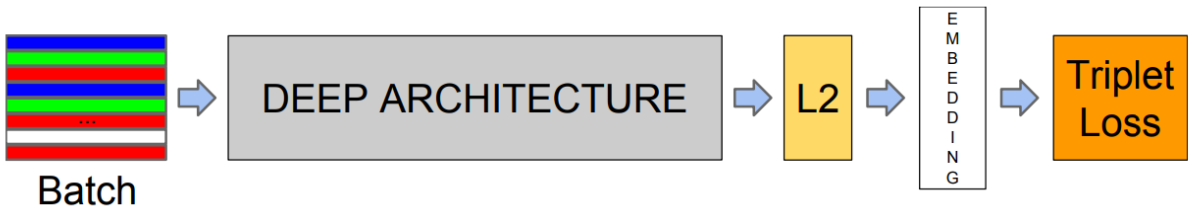
MTCNN performs - Face Classification, Bounding Box Regression, Facial Landmark Localization. We utilize the Face Bounding Box Regression to crop the face of the user.

FaceNet

FaceNet is the facial recognition system that was proposed by Google Researchers in 2015 in the paper titled *FaceNet: A Unified Embedding for Face Recognition and Clustering*. It achieved state-of-the-art results in the many benchmark face recognition dataset such as Labeled Faces in the Wild (LFW) and Youtube Face Database.

They proposed an approach in which it generates a high-quality face mapping from the images using deep learning architectures such as ZF-Net and Inception. Then it used a method called triplet loss as a loss function to train this architecture.

Architecture:



FaceNet employs end to end learning in its architecture. It uses ZF-Net or Inception as its underlying architecture. It also adds several *1*1* convolutions to decrease the number of parameters. These deep learning models outputs an embedding of the image

f(x) with L_2 normalization performed on it. These embeddings then passed into the loss function to calculate the loss. The goal of this loss function is to make the squared distance between two image embedding independent of image condition and pose of the same identity is small, whereas the squared distance between two images of different identity is large. Therefore a new loss function called Triplet loss is used. The idea of using triplet loss in our architecture is that it makes the model to enforce a margin between faces of different identities.

The embedding of an image is represented by *f(x)* such that x belongs to R. This embedding is in the form of a vector of size *128* and it is normalized.

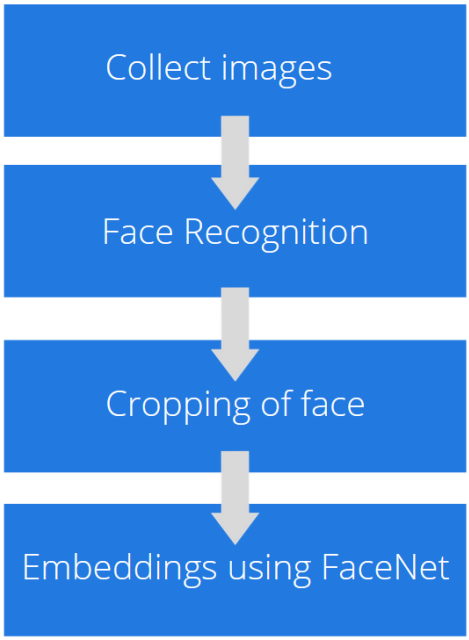
It makes sure that the anchor image of a person is closer to a positive image (image of the same person) as compared to a negative image (image of another person)

If triplets are easily satisfied above property then it would not help training, so it is important to have the triplets that violate the above principle.

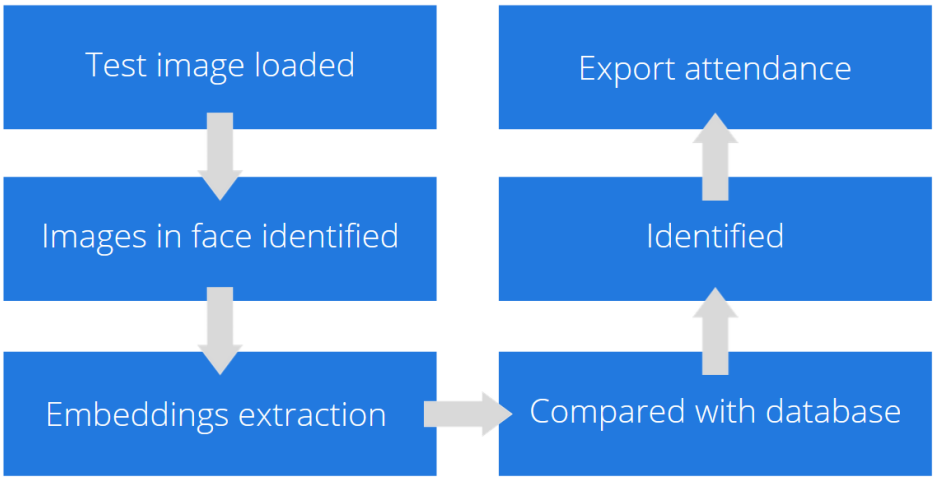
From FaceNet, embeddings of faces can be extracted from the trained model which gives us a representation of the features of the face.

Block Diagram

Training phase



Testing



Methodology

- Training
    - Collect images of the intended users of the applications and these images are placed in individual folders named based on the user's name in the same directory
    - These pictures ideally include only the user's face and not any other images.
    - These images are loaded into a python dictionary that contains the user's names as the keys and
    - The images dictionary is then forwarded to MTCNN model for face recognition which returns the coordinates of the bounding boxes of the faces in the images
    - Using the coordinates, the faces are cropped and stored in a new dictionary. This is fed to the FaceNet model's embedder
    - The Embeddings provided by the FaceNet model are then stored in a database in the format of a dictionary

- Testing
    - Testing images are fed to the program and are loaded using OpenCV
    - Images in the face are identified and cropped using MTCNN
    - Embeddings are extracted for the faces in the test image
    - These embeddings are compared with database for similarity
    - Based on threshold a face may either be classified as one of the users with which it has the minimum distance or as none of the members in the database
    - For every identified face, attendance is logged as an entry into a CSV file

Code

Training:

Importing Libraries

```
import tensorflow as tf
import numpy as np
from keras_facenet import FaceNet
import cv2
import os
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.simplefilter('ignore')
from utils.face_mtcnn import *
import csv
import datetime
```

Loading model

```
embedder = FaceNet()
```

Loading images

```
base_dir = "data/train_images"
persons = os.listdir(base_dir)
images = {}
# image_paths = []
for person in persons:
    images[person] = []
    paths = os.listdir(os.path.join(base_dir,person))[:5]
    for path in paths:
        # image_paths.append(os.path.join(base_dir,person,path))
        images[person].append(cv2.cvtColor(cv2.imread((os.path.join(base_dir,person,path))),cv2.COLOR_BGR2RGB))
print("Total number of people found = ", len(images))
```

```
Total number of people found =  3
```

Face Detection and Cropping

```
def Face_detection(images):

    crop_img = {}

    # faceCascade= cv2.CascadeClassifier("utils/haarcascade_frontalface_default.xml")
    for a in images.keys():
        crop_img[a] = []
        for img in images[a]:
            faces = detect_Faces_img(img)
            if len(faces) != 0:
                for b in faces:
                    (x,y,w,h) = b['box']
                    Cropped = img[y : y + h , x: x+w ]
                    crop_img[a].append(Cropped)

    return crop_img
```

```
# cropped images
Crop_Images = Face_detection(images)
```

Embedding Extraction

```
embeddings = {}
for a in Crop_Images.keys():
    embeddings[a] = embedder.embeddings(Crop_Images[a])
```

```
#creating a database for people we expect to visit us.
database_size = 5
database = {}
for person in persons:
    database[str(person)] = embeddings[person][:database_size]

database
```

Testing:

```
def Face_recog(image_path , alpha = 2, output_file = False):
    '''
    image_path : list of individual input image path.
    alpha : it is a hyperparameter
    '''

    #detecting face
    img = cv2.cvtColor(cv2.imread(image_path),cv2.COLOR_BGR2RGB)
    faces = detect_Faces_img(img)
    crop_imgs = []
    if len(faces) != 0:
        for b in faces:
            (x,y,w,h) = b['box']
            Cropped = img[y : y + h , x: x+w ]
            crop_imgs.append(Cropped)

    present = []

    img_embedding = embedder.embeddings(crop_imgs)
    # for crop in crop_imgs:
    #     img_embedding.append(embedder.embeddings(crop))

    #calculate dist wrt to database images
    min_dist = 100
    i=0

    for embedding in img_embedding:
        for (name,db_emb) in database.items():

            for emb in db_emb:

                dist = np.linalg.norm(embedding - db_emb)

                if dist < min_dist:
                    min_dist = dist
                    identity = name

        plt.axis("off")
        plt.imshow(crop_imgs[i])
        plt.show()
```

```
            i += 1
            if min_dist > alpha:
                print("Not found in Database")

            else:
                print(f"{identity} identified")
                present.append(identity)

        if output_file:
            save_attendance(present)
```

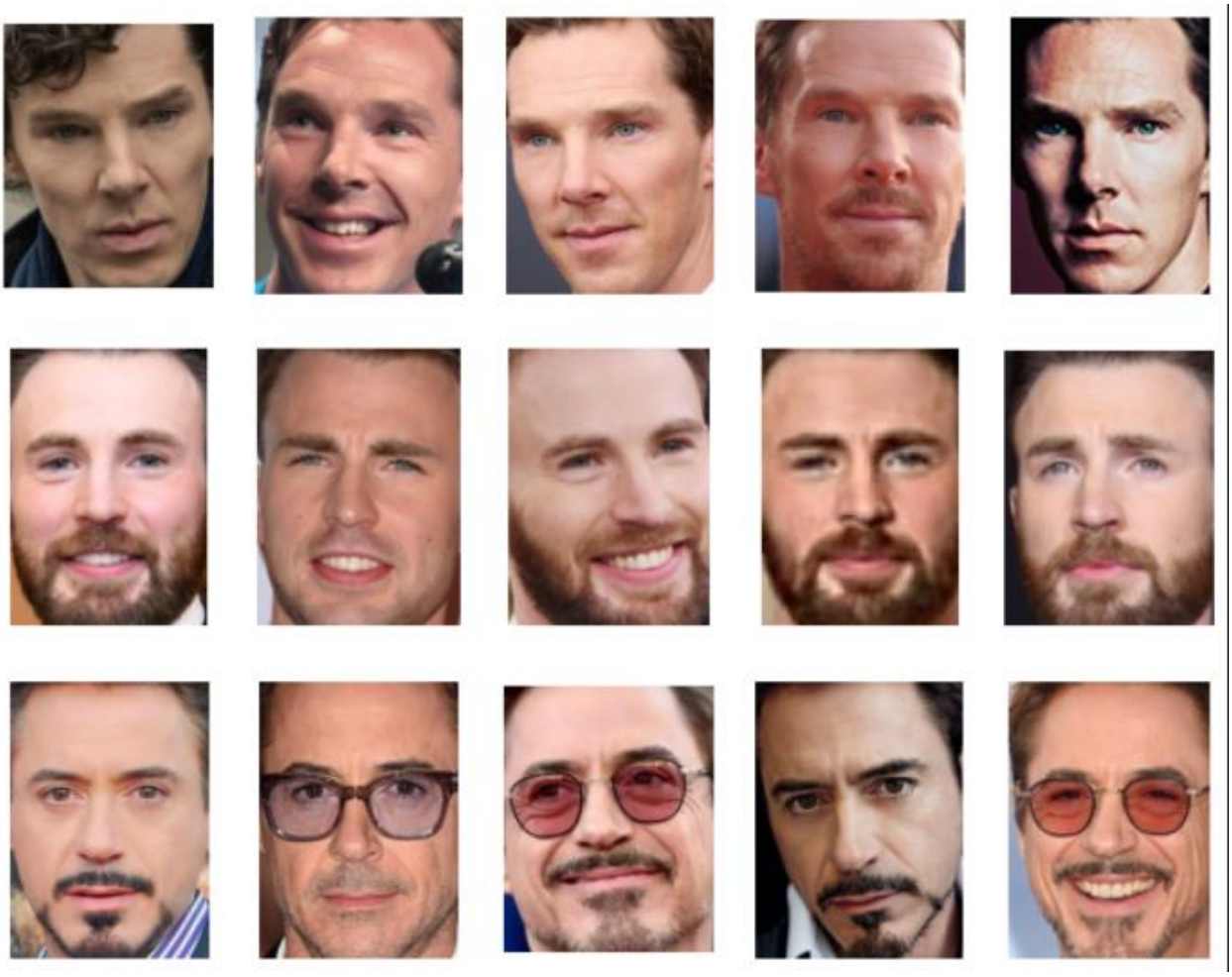
```
def save_attendance(present):
    # open the file in the write mode
    f = open('log.csv', 'a')

    # create the csv writer
    writer = csv.writer(f)
    for a in present:
        # write a row to the csv file
        writer.writerow([a, datetime.datetime.now()])

    # close the file
    f.close()
```

```
test_img_path = 'data/test/00000042.jpeg'
Face_recog(test_img_path, output_file= True)
```

Inference

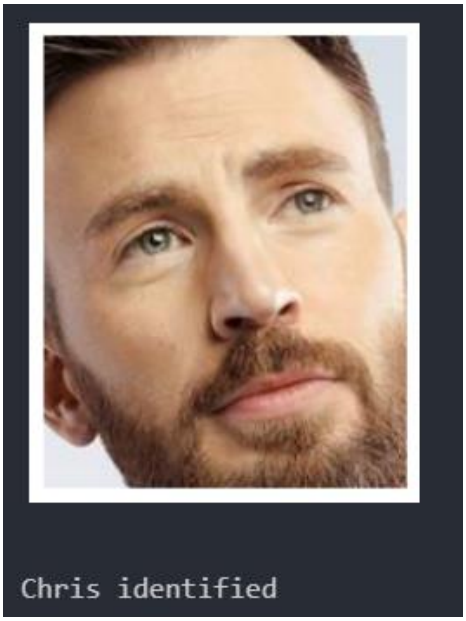
For testing, the database was trained using images of celebrities like Chris Evans, Robert Downey Jr. and Benedict Cumberbatch. Below are the results when the model was tested.

Sample Training Images:



Test image:

Output:



It is seen that the program has correctly recognised the user's face, cropped accordingly and identified the face correctly as Chris by extracting the embeddings of the test image and comparing it with the database.

Test image:

Output:

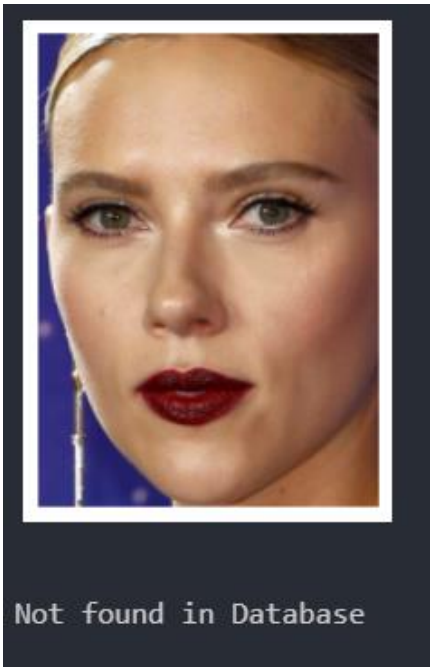
Test image:

Output:

When a member not stored in the database is present in the image, the model returns that the face is not found in the database.

Test image:

Output:



Test image:

Output:

Conclusion

Thus we can see that the python script can perform facial recognition and detection which is in turn implemented for attendance validation. Test images have been used to estimate its performance and the script has been able to correctly identify the users as expected by using embeddings extracted by FaceNet.

The above algorithm can either be loaded onto a microprocessor like Raspberry Pi for onboard processing or can be loaded to a remote server that can fetch the images from a classroom's CCTV footage to do the processing off-board. It can also be implemented as an Android app that can be used by Teachers and Employers to maintain attendance.

Implementation was also tested using a Raspberry Pi wherein the Pi was made to upload an image captured by an attached web-camera to a cloud database and then this image was fetched by a computer system acting as a server for testing purposes and by running the script, attendance logging was validated.

Thus we have implemented an Automatic Attendance System using Face Recognition in Python.

GitHub Link: https://github.com/sam189239/face_attendance

References

https://en.wikipedia.org/wiki/Python_(programming_language)

https://medium.com/@iselagradilla94/multi-task-cascaded-convolutional-networks-mtcnn-for-face-detection-and-facial-landmark-alignment-7c21e8007923

https://www.geeksforgeeks.org/facenet-using-facial-recognition-system/

https://en.wikipedia.org/wiki/OpenCV