Unconstrained Contemporaneous Video-Based Human Profile Recognition

Team Members

1. Adhithya K
2. Sharon Tincy B S
3. Akash Murugesh K

TABLE OF CONTENTS

TITLE **PAGE NO**

TABLE OF CONTENTS

ABSTRACT

As the digital world and real world merge more and more together, how to accurately and effectively identify users and improve information security has become an important research topic. Since the 9-11 terrorist attacks, governments all over the world have made urgent demands on this issue, prompting the development of emerging identification methods. Traditional identity recognition technology mainly relies on the individual's own memory (password, username, etc.) or foreign objects (ID card, key, etc.). However, whether by virtue of foreign objects or their own memory, there are serious security risks. If the identification items that prove their identity are stolen or forgotten, it is not only difficult to regain the original identity material, but also the identity information can be easily acquired by others. As a result, if the identity is impersonated by others, then there will be serious consequences.

Different from the traditional identity recognition technology, biometrics is the use of the inherent characteristics of the body for identification, such as fingerprints, irises, face and so on. When comparing the differences between different biometrics, we can see that the cost of facial recognition is low, the acceptance from users is easy, and the acquisition of information is easy. Besides being non-intrusive, more natural and easy to use, it can also be captured at a distance and in a covert manner.

Facial recognition is the use of computer vision technology and related algorithms, from the pictures or videos to find faces, and then analyse the identity. In addition, further analysis of the acquired face may provide some additional attributes of the individual, such as gender, age, emotion, and etc. While the main focus was on image-based face recognition in the beginning, it shifted more and more towards video-based approaches in the last years. These are developed in order to overcome shortcomings of image-based recognizers like sensitivity to low resolution, pose variations and partial occlusion. Compared with static images, video carries a larger amount of data and a more complicated scene too. In this project, a method of video-based face recognition is proposed which is fast, robust, not complex and achieves greater accuracy with relatively simple and easy to comprehend algorithms and techniques for real-time application.

ABSTRACT – TAMIL

எண்முறை உலகமும் நிஜ உலகமும் மேலும் மேலும் ஒன்றிணைவதால், பயனர்களை எவ்வாறு துல்லியமாகவும் திறம்படமாகவும் அடையாளம் கண்டுகொள்வது மற்றும் தகவல் பாதுகாப்பை மேம்படுத்துவது என்பது ஒரு முக்கியமான ஆராய்ச்சித் தலைப்பாக மாறியுள்ளது. 9-11 பயங்கரவாதத் தாக்குதல்களுக்குப் பிறகு, உலகெங்கிலும் உள்ள அரசாங்கங்கள் இந்த பிரச்சினையில் அவசர கோரிக்கைகளை முன்வைத்துள்ளன. பாரம்பரிய அடையாள அங்கீகார தொழில்நுட்பம் முக்கியமாக தனிநபரின் சொந்த நினைவகம் (கடவுச்சொல், பயனர்பெயர், முதலியன) அல்லது பொருட்களை (அடையாள அட்டை, சாவி போன்றவை) சார்ந்துள்ளது. இருப்பினும், வெளிநாட்டுப் பொருட்கள் அல்லது அவற்றின் சொந்த நினைவாற்றல் காரணமாக, கடுமையான பாதுகாப்பு அபாயங்கள் உள்ளன. அவர்களின் அடையாளத்தை நிரூபிக்கும் அடையாளப் பொருட்கள் திருடப்பட்டாலோ அல்லது மறந்துவிட்டாலோ, அசல் அடையாளப் பொருளை மீட்டெடுப்பது கடினம் மட்டுமல்ல, அடையாளத் தகவலை மற்றவர்களால் எளிதாகப் பெற முடியும். இதன் விளைவாக, அடையாளம் மற்றவர்களால் ஆள்மாறாட்டம் செய்யப்பட்டால், கடுமையான விளைவுகள் ஏற்படும்.

பாரம்பரிய அடையாள அங்கீகார தொழில்நுட்பத்திலிருந்து வேறுபட்டது, பயோமெட்ரிக்ஸ் . இது கைரேகைகள், கருவிழிகள், முகம் மற்றும் பலவற்றை அடையாளம் காண உடலின் உள்ளார்ந்த பண்புகளை பயன்படுத்துவதாகும். வெவ்வேறு பயோமெட்ரிக்குகளுக்கு இடையிலான வேறுபாடுகளை ஒப்பிடும் போது, முகத்தை அடையாளம் காணும் செலவு குறைவாக இருப்பதையும், பயனரிடமிருந்து ஏற்றுக்கொள்வது எளிதானது என்பதையும், தகவல்களைப் பெறுவது எளிதானது என்பதையும் பார்க்கலாம்.

முக அங்கீகாரம் என்பது கணினி பார்வை தொழில்நுட்பம் மற்றும் தொடர்புடைய வழிமுறைகள், படங்கள் அல்லது வீடியோக்களில் இருந்து முகங்களைக் கண்டறிந்து, பின்னர் அடையாளத்தை பகுப்பாய்வு செய்வதாகும். இந்த திட்டத்தில், வீடியோ அடிப்படையிலான முகத்தை அடையாளம் காணும் முறை முன்மொழியப்பட்டது, இது வேகமானது, வலிமையானது, சிக்கலானது அல்ல, மேலும் நிகழ்நேர பயன்பாட்டிற்கான வழிமுறைகள் மற்றும் நுட்பங்களுடன் ஒப்பீட்டளவில் எளிமையானது மற்றும் எளிதில் புரிந்துகொள்வதன் மூலம் அதிக துல்லியத்தை அடைகிறது.

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATION

ASCII - American Standard Code for Information Interchange

CCTV - closed-circuit television

CDF - Cumulative Distribution Function

CNN - Convolutional Neural Network

DT - Decision Tree

FRS - Face Recognition System

KNN - K Nearest Neighbours

LR - Logistic Regression

NB - Naive Bayes

PDF - Probability Density Function

RGB - Red Green Blue

ROC - Receiver Operating Characteristic

SIFR - Still Image-based Face Recognition

SVD - Singular Value Decomposition

SVM - Support Vector Machine

TBE-CNN - Trunk Branch Ensemble CNN

VFR - Video-based Face Recognition

CHAPTER 1
INTRODUCTION

1.1 PROBLEM STATEMENT

In the past years, several cases of child abduction, impersonation, shoplifting, etc., went unsolved due to the lack of a robust face recognition system. A comprehensive face recognition system could aid in solving at least a portion of the 7,50,000 cases that get closed every year due to a lack of evidence in India. The project proposes an unconstrained video-based face recognition system to achieve the aforementioned trait.

1.2 OVERALL OBJECTIVES

The central goal is to build a real-time capable face recognition system (FRS) for uncontrolled environments. It is supposed to robustly handle real-life situations with all the challenges they bring along that make the task harder.

The central objective is to achieve unobtrusive recognition, i.e. to create a system that operates in the background and does not need specific user interaction. This is essential to grant users the freedom to behave naturally. As a consequence of this freedom, difficulties arise from varying poses, like out-of-plane rotations, and different facial expressions. The aim is to develop a model to efficiently handle these difficulties while maintaining accuracy.

1.3 PROPOSED SOLUTIONS

A method of video-based face recognition is developed which is fast, robust, not complex and achieves greater accuracy with relatively simple and easy to comprehend algorithms and techniques for real-time application.

The system built is aimed to effectively handle

i) Pose variations
ii) Occlusion
iii) Variation in illumination

1.4 OVERVIEW

The need for Video-based Facial Recognition in today's society has been increasing lately due to the surge in the number of undetected crimes. If a comprehensive tool/ software to handle the same is developed, the benefits would be many. But, the task comes with a vast number of challenges in the form of occlusions, pose variations and illumination defects, which need to be handled effectively in order to yield the desired fruits. Thus, in this project, an overarching solution that recognizes faces from a real-time video in an unconstrained environment is proposed.

CHAPTER 2
RELATED WORK

2.1 VIDEO-BASED FACE RECOGNITION

Existing methods on VFR can be divided into three categories:

1) Methods for frame quality evaluation.
2) Methods that exploit redundant information between frames.
3) Methods to achieve robust feature extraction from each frame.

Some recent VFR studies [1], [2], [3], unlike frame quality evaluation methods, attempt to make use of redundant information contained between video frames.

Due to the relative motion between the subjects and cameras, video frames suffer from severe image blur. CNN-based methods haven't yet been used to handle the image blur problem in VFR, as per our knowledge based on literature. Faces in video frames show high pose, illumination, and expression variations and occlusion, but existing studies[4], [5] tend to directly extend feature extractors designed for SIFR to VFR.

This proposed method comes under the third category of methods. When compared with previous VFR methods [6], [7], an efficient CNN model to automatically learn face representations that are robust to image blur, pose variations and occlusions is proposed. Several research attempts [8], [9], [10] have been made to employ synthetic profile face images as augmented extra data to balance the pose variations.

2.2 DEEP NEURAL NETWORKS FOR FACE RECOGNITION

Convolutional Neural Networks (CNNs) have been widely applied into some vision tasks [11], [12] including image classification, object detection and semantic segmentation. DeepFace [13] first brought CNNs into face recognition, improving the performance of face recognition greatly. With the success of CNNs in face recognition, Some face networks such as VGGFace [14], Light CNN [15], FaceNet and SphereFace [16] are proposed to further improve the performance of face recognition.

2.3 VARIATIONS FROM THE BASE PAPER

The base paper uses PaSC, COX Face, and YouTube Faces datasets, which are predefined ones, but this project has its own dataset where videos are captured and chipped into individual frames which are later labelled and used for recognition purposes.

To achieve image enhancement while capturing videos, Histogram Equalisation is proposed which would in turn help achieve seamless Face Recognition by improving the contrast of frames.

There aren't ample resources and documentation available for TBE-CNN Implementation. But, making use of the base paper, TBE-CNN model for facial recognition is built.

Generally most face recognition models do not have any exclusive features to identify and recognize small faces in videos. But Image Pyramiding technique is used to resize the frames to identify even the small faces present in the video.

CHAPTER 3
SYSTEM ARCHITECTURE

3.1 PROPOSED ARCHITECTURE DIAGRAM

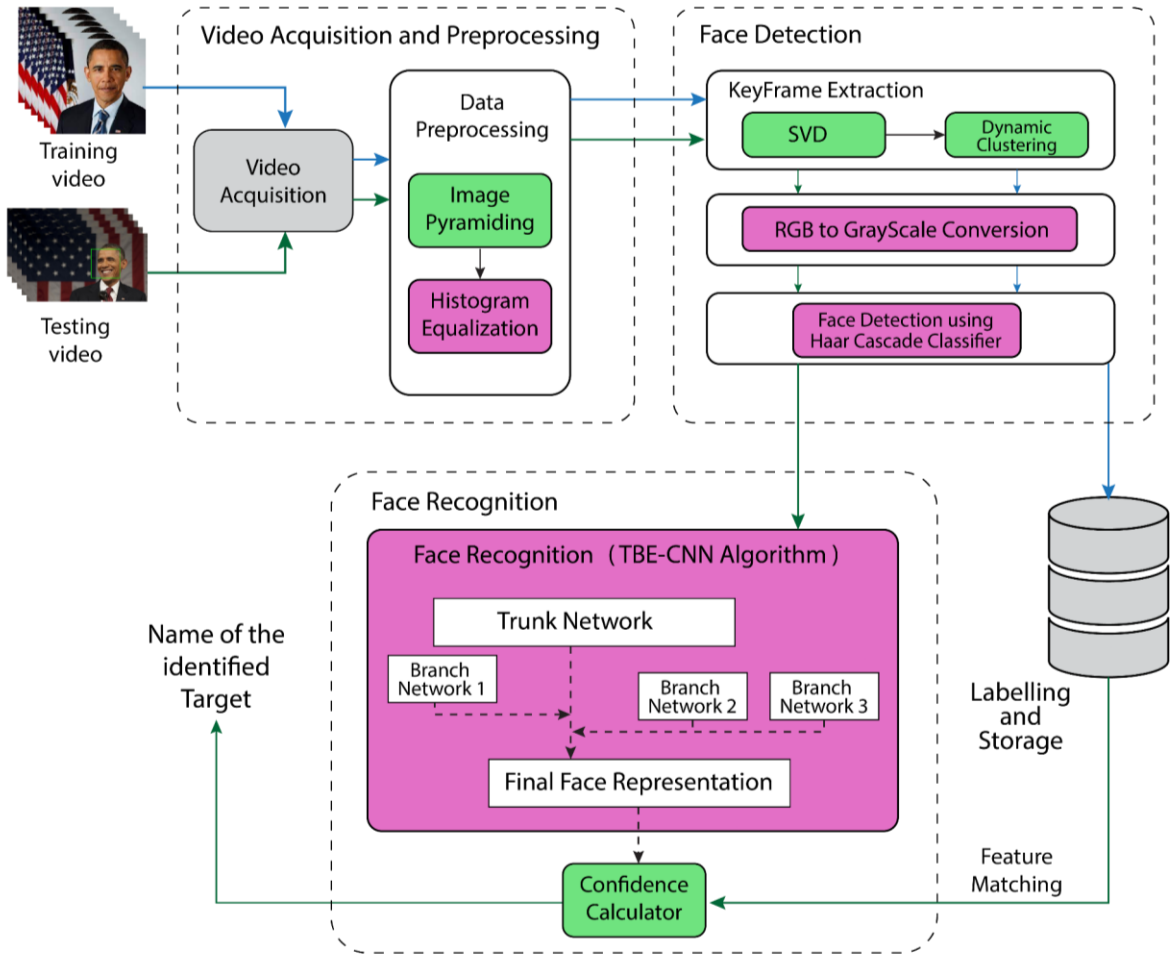


Figure 3.1.1 Architecture Diagram

As shown in Figure 3.1.1, initially, the input video of a person is captured using the Video Acquisition Module. Then Preprocessing techniques like Image Pyramiding and Histogram Equalization are applied over the obtained frames. Later, the significant keyframes are extracted from the existing frames using the Dynamic Clustering approach. The keyframes are then passed on to the Haar Cascade Classifier to get bounding boxes enclosing the detected faces. Finally, the detected faces are passed to the proposed Face recognition module which encompasses the Trunk Branch Ensemble Convolution Neural Network to give us the name of the person whose face appears on the frame along with a confidence value.

CHAPTER 4
LIST OF MODULES

4.1 Video Acquisition & Pre-processing module
4.2 Face detection
4.3 Face Recognition

4.1 VIDEO ACQUISITION & PRE PROCESSING

MODULE 1: Video Acquisition & Pre-processing
INPUT: Real-time Video
OUTPUT: Pre-processed data

Figure 4.1.1 Module 1 flow diagram

The model acquires the video from livestream through the Video Acquisition submodule. To reduce the variability in the faces, the videos must be processed before they are fed into the network as shown in Figure 4.1.1. To achieve this feat, the Image Pyramiding technique and Histogram Equalisation are proposed.

4.1.1 IMAGE PYRAMIDING

The pyrUp() function increases the size to double of its original size and pyrDown() function decreases the size to half as shown in Figure 4.1.1.1.

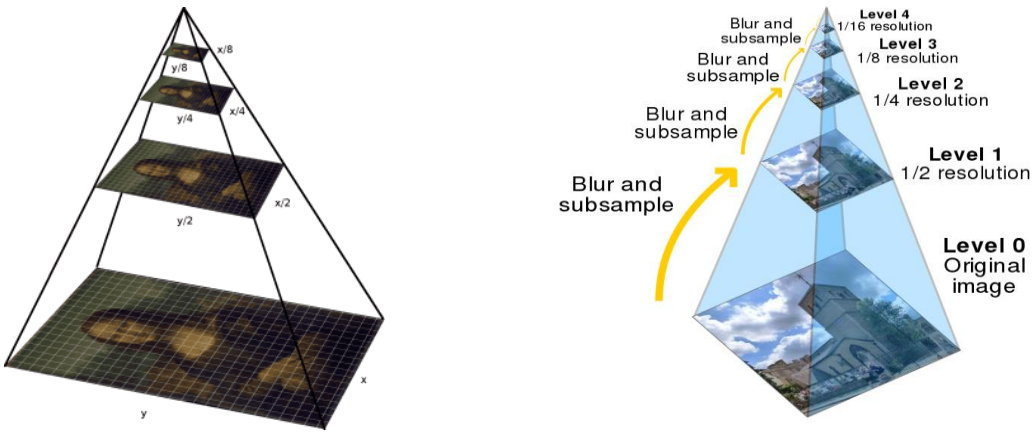


Figure 4.1.1.1 Image Pyramid portraying PyrUp() and PyrDown() functions

If the original image is kept as a base image and pyrDown function is applied on it and the images are kept in a vertical stack, it will look like a pyramid. The same is true for upscaling the original image by pyrUp function.

4.1.2 HISTOGRAM EQUALISATION DESCRIPTION

It is a well-known contrast enhancement technique due to its performance on almost all types of image. A histogram is a representation of frequency distribution. A histogram of an image is the graphical interpretation of the image's pixel intensity values. It can be interpreted as the data structure that stores the frequencies of all the pixel intensity levels in the image.

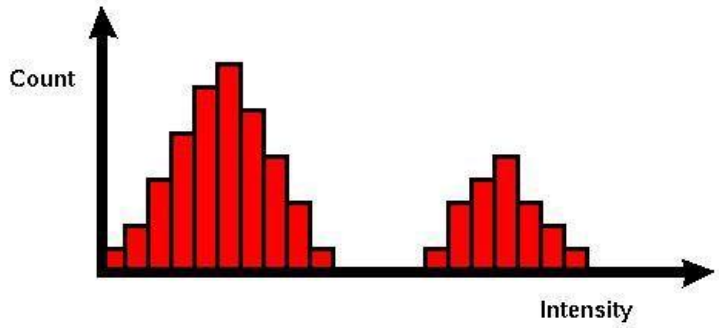
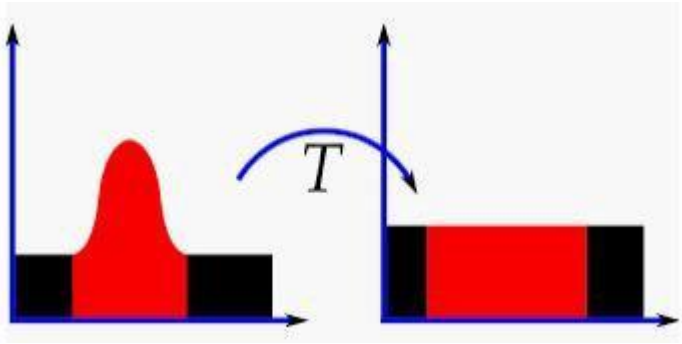


Figure 4.1.2.1 A Sample Histogram

As shown in the Figure 4.1.2.1 above, the X-axis represents the pixel intensity levels of the image. The intensity level usually ranges from 0 to 255. The Y-axis of the histogram indicates the frequency or the number of pixels that have specific intensity values.

Histogram Equalization is an image processing technique that adjusts the contrast of an image by using its histogram. To enhance the image's contrast, it spreads out the most frequent pixel intensity values or stretches out the intensity range of the image. By accomplishing this, histogram equalization allows the image's areas with lower contrast to gain a higher contrast as shown in Figure 4.1.2.2.



x axis - intensity
y axis - count

Figure 4.1.2.2 Transformation of histogram after Equalization

Histogram Equalization can be used when you have images that look washed out because they do not have sufficient contrast. In such photographs, the light and dark areas blend together creating a flatter image that lacks highlights and shadows.

4.1.3 HISTOGRAM EQUALIZATION ALGORITHM

Step1: Compute Normalised Histogram for input image (PDF = frequency of each pixel / sum of all frequencies of all gray levels)
Step2: Compute Cumulative Histogram (CDF)
Step3: Multiply CDF with maximum gray level value in the image to get the output pixel value.
Step4: Transform all input pixels into output pixels in the image to get the enhanced image.

4.2 FACE DETECTION

MODULE 2: Face Detection
INPUT: Pre-processed data
OUTPUT: Bounding boxes enclosing faces

Figure 4.2.1 Module 2 flow diagram

The pre-processed frames procured from Module 1 are fed into the Face Detection framework, to enclose the faces in them with bounding boxes as shown in Figure 4.2.1. Face Detection is the fundamental step in any of the operations carried out in the face recognition process. Once the face is detected, it can be cropped and stored as a sample image for analysis. Instead of directly applying any Face detection algorithm over all frames, first key frames are extracted and only faces in those frames are detected.

4.2.1 KEY FRAME EXTRACTION

Key frame extraction is a powerful tool that selects a set of summary key frames to represent video sequences. Key-frame refers to the image frame in the video that represents and summarises it. These can be selected as shot boundaries if we can detect shots and ignore transitions between them. Color Histogram, SVD and Dynamic Clustering Methods are proposed to obtain Key-Frames from a video.

Singular Value Decomposition

The color histogram for each of the frames in the video are generated in all three channels (RGB). The histograms are then concatenated to form a feature vector for every frame to create a feature-frame matrix for the entire video. Then dimensionality reduction is performed on the matrix using SVD (Singular Value Decomposition), thus reducing the dimensions of the feature-frame matrix.

Algorithm

The Singular Value Decomposition (SVD) of a matrix is a factorization of that matrix into three matrices as shown in Figure 4.2.1.1.

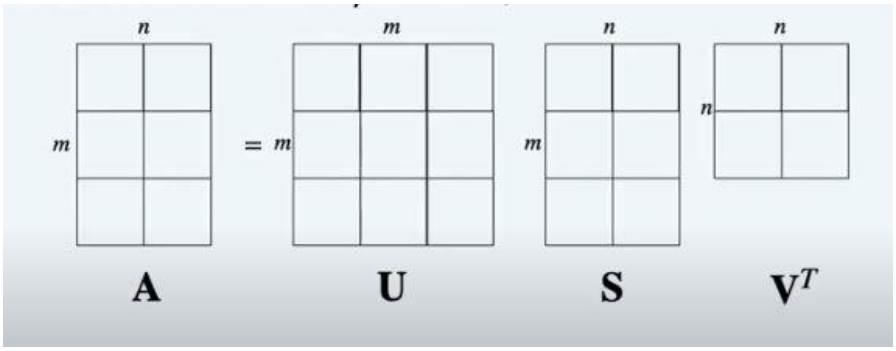
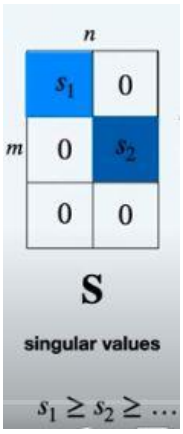


Figure 4.2.1.1 Decomposing Matrix A into Matrices U, S and V^T

Here, A denotes the matrix to be decomposed. SVD states that any matrix A can be decomposed into or written as a product of 3 matrices - U, S (Sigma) and V^T. The columns of matrix U are called "Left Singular Vectors." Matrix U is an Orthogonal Matrix, so $U^T = U^{-1}$ and U contains the orthonormal eigenvectors of AA^T. The columns of V (or the rows V^T) are called "Right Singular vectors". Matrix V^T contains orthonormal eigenvectors of A^TA.



S is a diagonal matrix containing the singular values (which are the square roots of the Eigenvalues of A^TA. The singular values appear in a non-increasing order.

We can represent the decomposition in another form as shown in Figure 4.2.1.2.

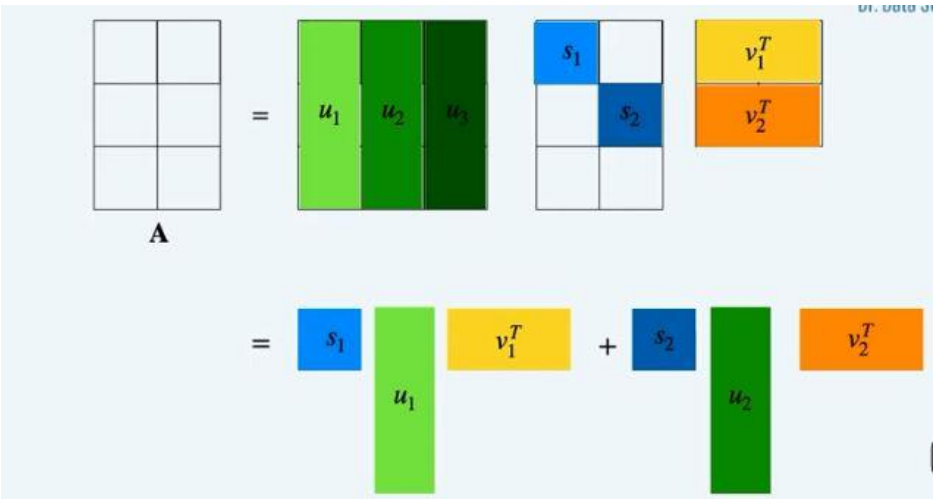


Figure 4.2.1.2 Visualisation of Singular Value Decomposition

If s2<<s1, the second term in the addition can be ignored, as the first term is much more important than the second term as shown in Figure 4.2.1.3. By this, we can achieve data reduction and approximate A with just the first term. By this, we have reduced the rank of matrix A from 2 to 1 (terms reduced from 2 to 1).

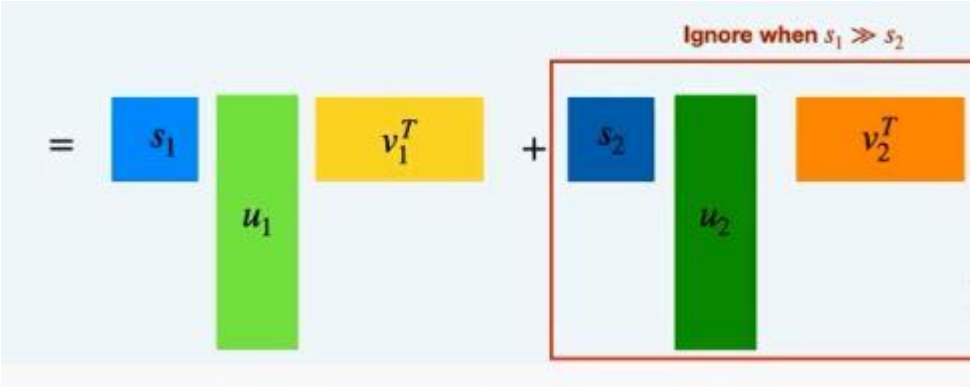


Figure 4.2.1.3 Ignoring negligible singular values

Dynamic Clustering

Then, clusters of consecutive frames are created by using Cosine Similarity to check whether the new frame was similar to the last cluster formed or not. This method is called Dynamic Clustering. Cosine similarity is a metric, helpful in determining, how similar the data objects are irrespective of their size. The formula to find the cosine similarity between two vectors is –

$$Cos(x, y) = x \cdot y / ||x|| * ||y||$$

where,

- **x . y** = product (dot) of the vectors ‘x’ and ‘y’.
- **||x||** and **||y||** = length of the two vectors ‘x’ and ‘y’.
- **||x|| * ||y||** = cross product of the two vectors ‘x’ and ‘y’.

The cosine similarity between two vectors is measured in ‘θ’.

- If θ = 0°, the ‘x’ and ‘y’ vectors overlap, thus proving they are similar.
- If θ = 90°, the ‘x’ and ‘y’ vectors are dissimilar.

The frames in the sparse clusters are considered as transitions between shots, so they were ignored. The frames in a dense cluster make a shot and we choose shots' last added frame as a key-frame.

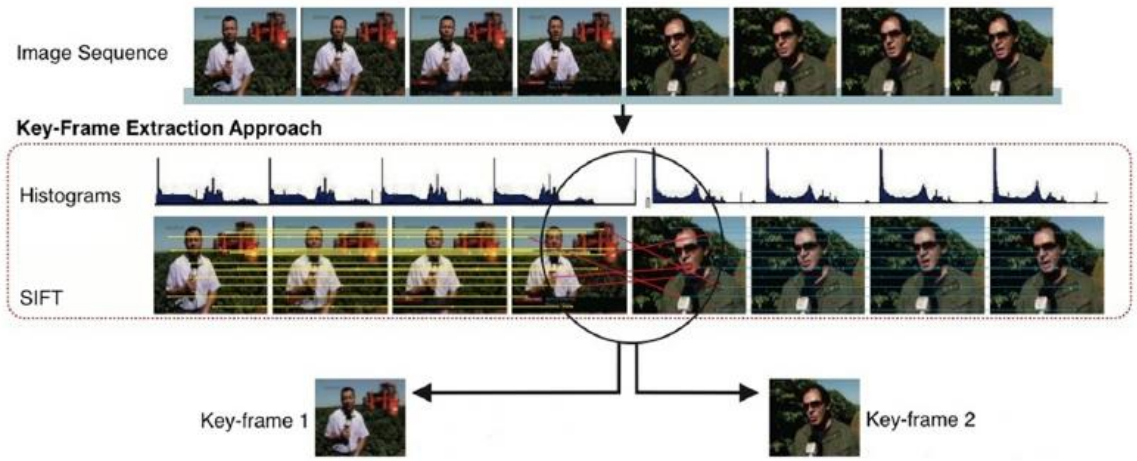


Figure 4.2.1.4 Sample key frame extraction

A sample extraction of key frames is shown above in Figure 4.2.1.4.

4.2.2 RGB TO GRAYSCALE CONVERSION

The main reason why grayscale representations are often used for extracting descriptors instead of operating on colour images directly is that grayscale simplifies the algorithm and reduces computational requirements. A grayscale (or gray level) image is simply one in which the only colours are shades of gray as shown in Figure 4.2.2.1. The reason for differentiating such images from any other sort of color image is that less information needs to be provided for each pixel.

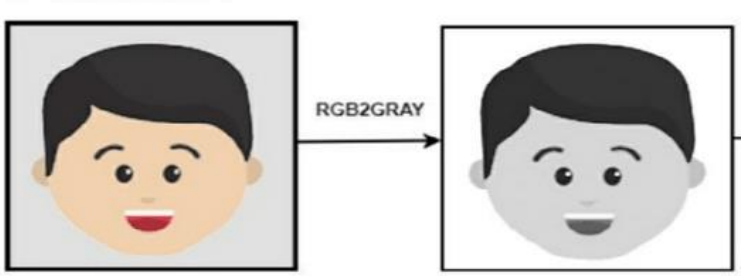


Figure 4.2.2.1 Conversion from RGB to grayscale

4.2.3 HAAR CASCADE CLASSIFIER DESCRIPTION

Now, the Haar Cascades Classifier (or) Viola Jones Face Detection Technique is used to detect faces in the key frames. It is an Object Detection Algorithm used to identify faces in an image or a real time video. The algorithm uses edge or line detection features proposed by Viola and Jones in their research paper “Rapid Object Detection using a Boosted Cascade of Simple Features” published in 2001. The algorithm is given a lot of positive images consisting of faces, and a lot of negative images not consisting of any face to train on them.

The first contribution to the research was the introduction of the haar features shown below in Figure 4.2.3.1. These features on the image makes it easy to find out the edges or the lines in the image, or to pick areas where there is a sudden change in the intensities of the pixels.

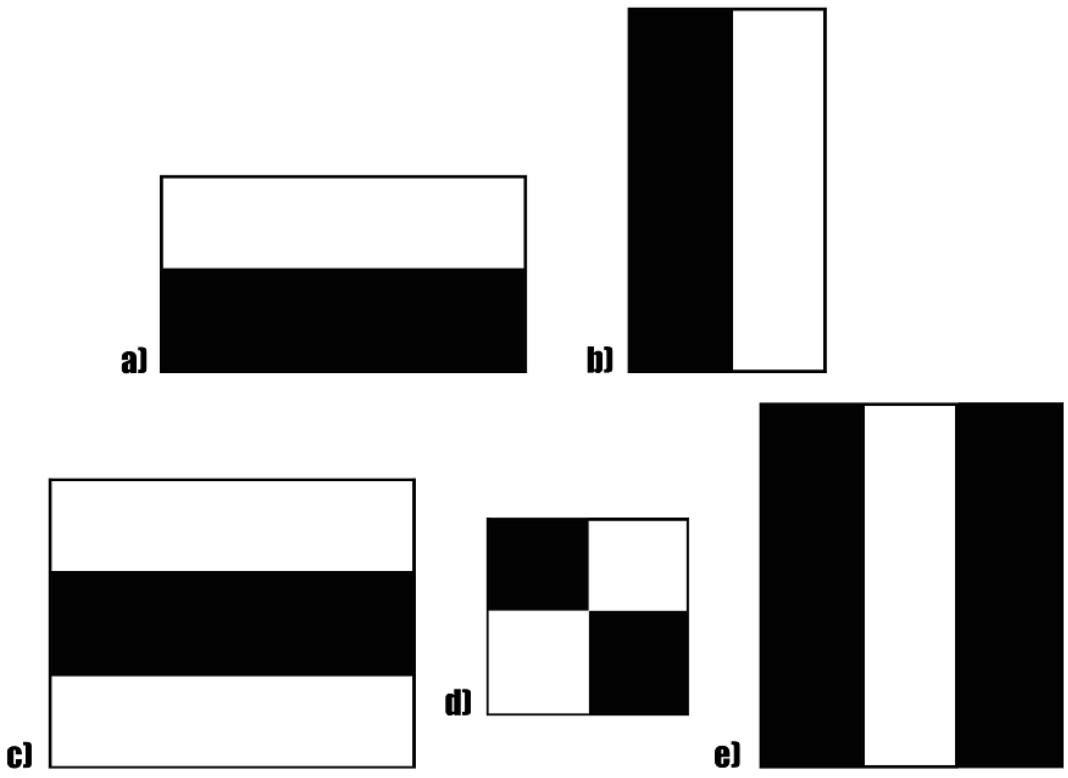


Figure 4.2.3.1 Sample Haar features used for Face Detection

A sample calculation of Haar value from a rectangular image section has been shown in Figure 4.2.3.2.

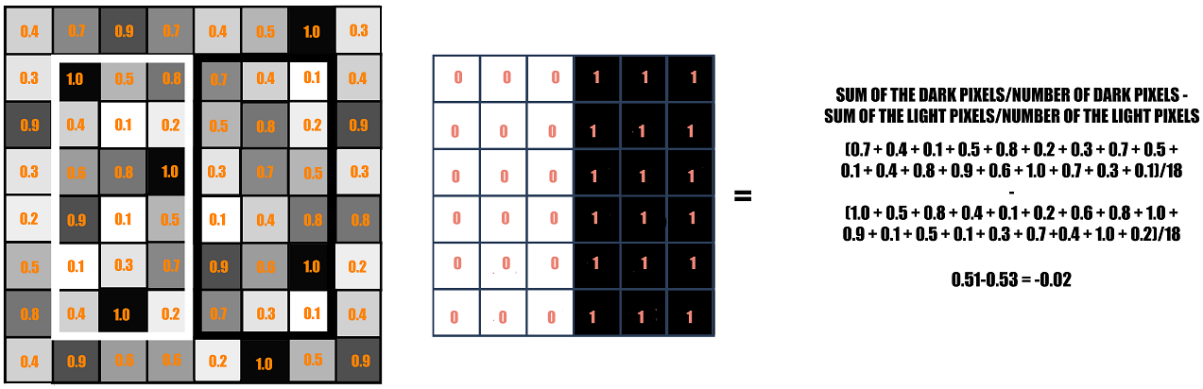


Figure 4.2.3.2 Sample Calculation of Haar value from a rectangular image section

The rectangle on the left is a sample representation of an image with pixel values 0.0 to 1.0. The rectangle at the centre is a haar kernel which has all the light pixels on the left and all the dark pixels on the right. The haar calculation is done by finding out the difference of the average of the pixel values at the darker region and the average of the pixel values at the lighter region. If the difference is close to 1, then there is an edge detected by the haar feature.

The darker areas in the haar feature are pixels with values 1, and the lighter areas are pixels with values 0. Each of these is responsible for finding out one particular feature in the image such as an edge, a line or any structure in the image where there is a sudden change of intensities. For example, in the image above, the haar feature can detect a vertical edge with darker pixels at its right and lighter pixels at its left.

This is just one representation of a particular haar feature separating a vertical edge. Now there are other haar features as well, which will detect edges in other directions and any other image structures as shown in Figure 4.2.3.3. To detect an edge anywhere in the image, the haar feature needs to traverse the whole image and also all possible sizes of the haar features will be applied.

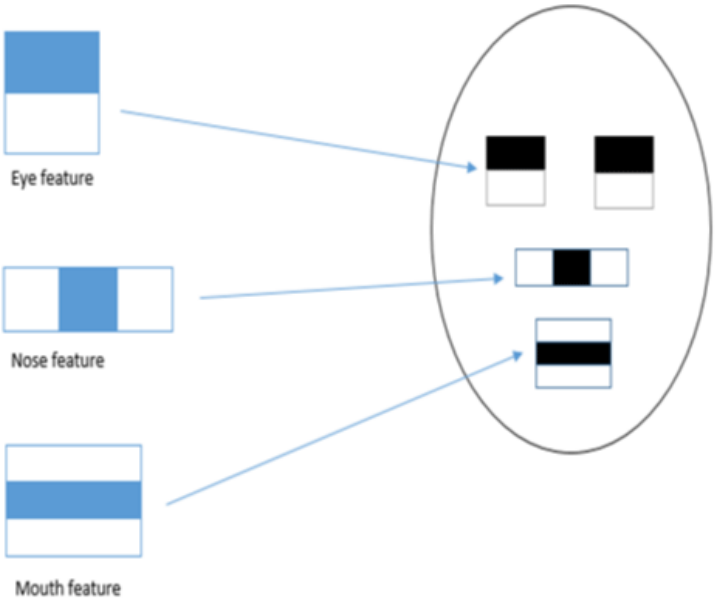


Figure 4.2.3.3 Representation of different Haar features used for face detection

Now, the haar features traversal on an image would involve a lot of mathematical calculations. As we can see for a single rectangle on either side, it involves 18 pixel value additions (for a rectangle enclosing 18 pixels). Imagine doing this for the whole image with all sizes of the haar features. This would be a hectic operation even for a high performance machine.

To tackle this, they introduced another concept known as The Integral Image to perform the same operation. An Integral Image is calculated from the Original Image in such a way that each pixel in this is the sum of all the pixels lying in its left and above in the Original Image.

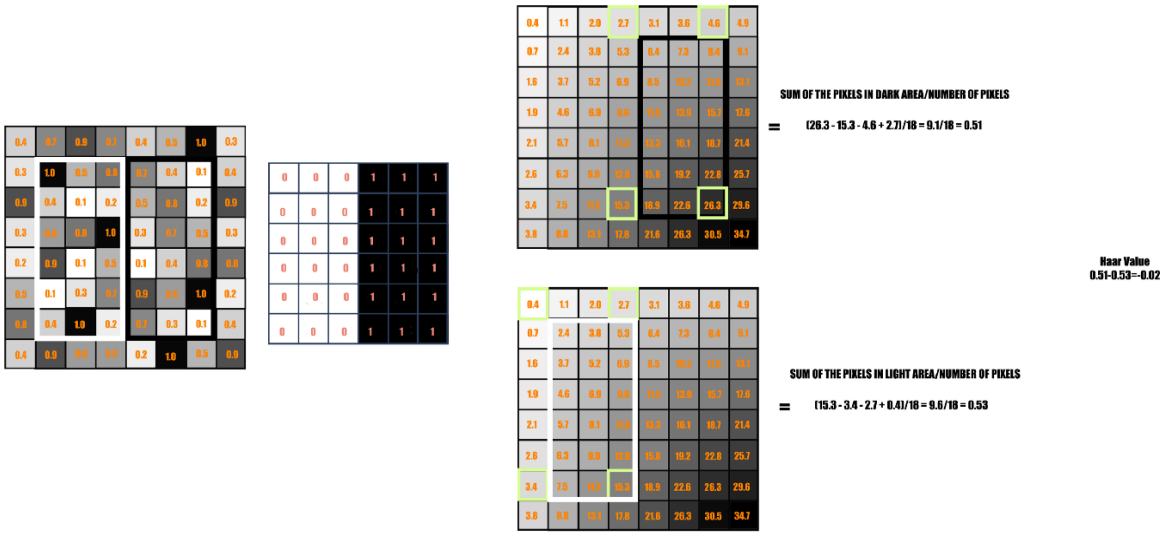


Figure 4.2.3.4 Calculation of Haar value for an Integral image

With the Integral Image, only 4 constant value additions are needed each time for any feature size (with respect to the 18 additions earlier) as shown in Figure 4.2.3.4. This reduces the time complexity of each addition gradually, as the number of additions does not depend on the number of pixels enclosed anymore.

In the above image, there is no edge in the vertical direction as the haar value is -0.02, which is very far from 1. Now comes the Cascading part. The subset of all 6000 Haar features will again run on the training images to detect if there's a facial feature present or not. Now the authors have taken a standard window size of 24x24 within which the feature detection will be running. It's again a tiresome task.

To simplify this, they proposed another technique called The Attentional Cascade. The idea behind this is, not all the features need to run on each and every window. If a feature fails on a particular window, then we can say that the facial features are not present there. Hence, we can move to the next window where there can be facial features present. This way a lot of processing time will be saved, as the irrelevant windows will not be processed as shown in Figure 4.2.3.5.

- Features are applied on the images in stages. The stages in the beginning contain simpler features, in comparison to the features in a later stage which are complex, complex enough to find the nitty gritty details on the face. If the initial stage won't detect anything on the window, then discard

the window itself from the remaining process, and move on to the next window. This way a lot of processing time will be saved, as the irrelevant windows will not be processed in the majority of the stages.

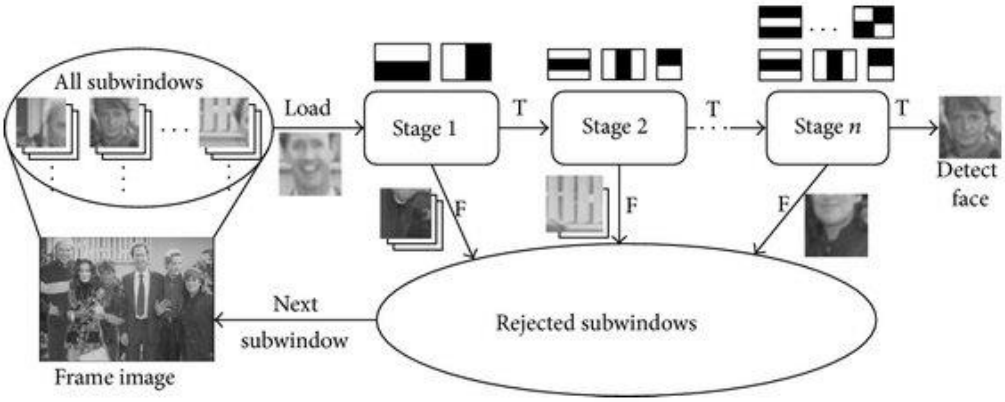


Figure 4.2.3.5 Attentional Cascade in Haar Cascade Classifier

- The second stage processing would start, only when the features in the first stage are detected in the image. The process continues like this, i.e. if one stage passes, the window is passed onto the next stage, if it fails then the window is discarded.

In the Viola — Jones research, they had a total of 38 stages for something around 6000 features.
The number of features in the first five stages are 1, 10, 25, 25, and 50, and this increased in the subsequent stages.

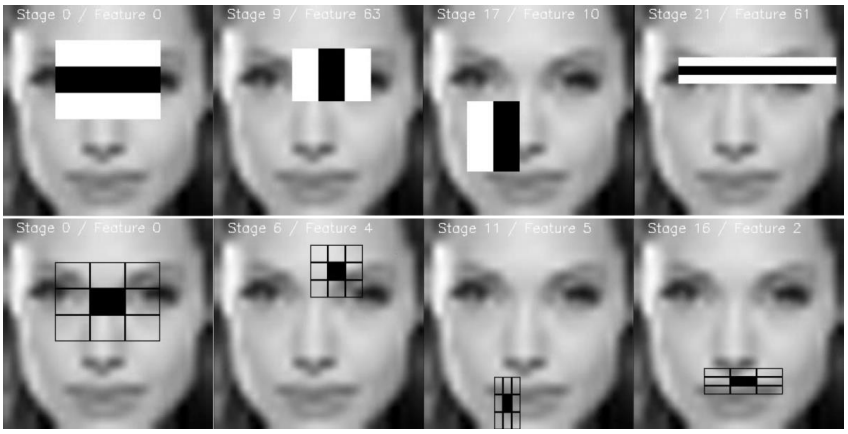


Figure 4.2.3.6 Traversal of Haar features over a sample image

The initial stages with simpler and lesser number of features removed most of the windows not having any facial features, thereby reducing the false negative

ratio, whereas the later stages with more complex and more number of features can focus on reducing the error detection rate, hence achieving a low false positive ratio as shown in Figure 4.2.3.6.

Haar Cascade Classifier Algorithm:

for i <- 1 to num of scales in pyramid of images
do
    Downsample image to create $image_i$
    Compute integral image, $image_{ii}$
    for j <- 1 to num of shift steps of sub-window
    do
        for k <- 1 to num of stages in cascade classifier
        do
            for l <- 1 to num of filters of stage k
            do
                Filter Detection Subwindow
                Accumulate filter outputs
            end for
            if accumulation fails per-stage threshold **then**
                Reject sub-window as not face
                Break this k for loop.
            end if
        end for
        if subwindow passed all per-stage checks **then**
            Accept this sub-window as a face
        end if
    end for
end for

4.3 FACE RECOGNITION

4.3.1 TBE-CNN DESCRIPTION

MODULE 3: Face Recognition
INPUT: Detected faces
OUTPUT: Identification of target in video

Figure 4.3.1.1 Module 3 flow diagram

The detected faces obtained from Module 2 are fed into the proposed Face Recognition module as shown in Figure 4.3.1.1. To enhance robustness of CNN features to pose variations and occlusion, the project proposes a Trunk-Branch Ensemble CNN model (TBE-CNN), which extracts complementary information from holistic face images and patches cropped around facial components. The proposed TBE-CNN is an end-to-end model that extracts features efficiently by sharing the low- and middle-level convolutional layers between the trunk and branch networks.

The proposed architecture consists of one trunk network and several branch networks. The trunk network will study global features, whereas the branch networks will study the local features in image input. The TBE-CNN layers are divided into three levels: the low level layers, middle-level layers, and high-level layers.

Since low- and middle-level features represent local information, the trunk network and branch networks can share low- and middle-level layers. In comparison, high-level features represent abstract and global information; therefore, different models should have independent high-level layers.

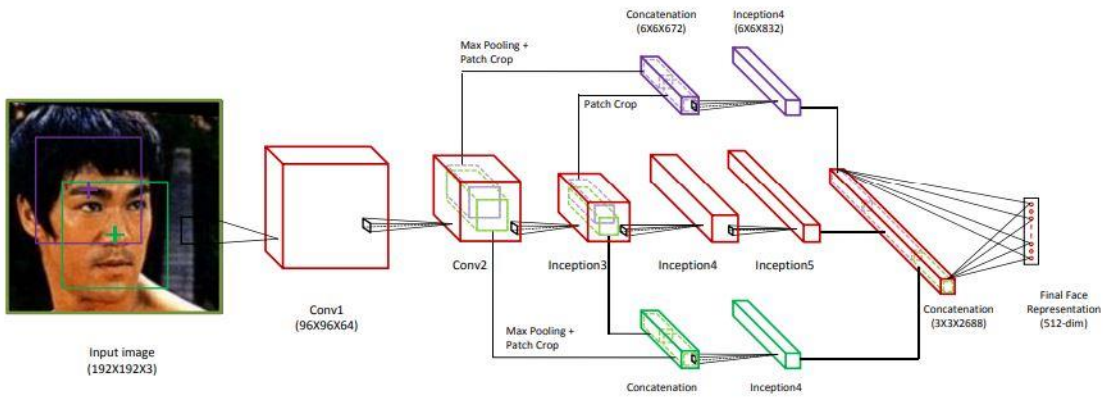


Figure 4.3.1.2 Architecture of TBE-CNN

The output feature maps of the trunk network and branch networks are fused by concatenation to form an over-complete face representation. The feature vector is utilised as the final face representation of one video frame as shown in Figure 4.3.1.2.

4.3.2 TBE-CNN ALGORITHM

Step1: Use Inception blocks to compose trunk and branch networks for the detected face.
Step2: Concatenate the feature maps of trunk and branch networks to get a holistic face representation.
Step3: Store the facial representation of the face along with the person's ID and name in the gallery.
Step4: Calculate and compose the facial representation of the test face using trunk and branch networks.
Step5: Calculate the Euclidean distance of test face's feature map with the gallery of trained face feature maps, by using the following equation,

Dist = sqrt(sum((I1 - I2) .^ 2))
Where, I1 is the test face feature map, and I2 is the face feature map from the database.
Step5: Display result according to the Euclidean Distance of the two face features as shown in Figure 4.3.1.3.

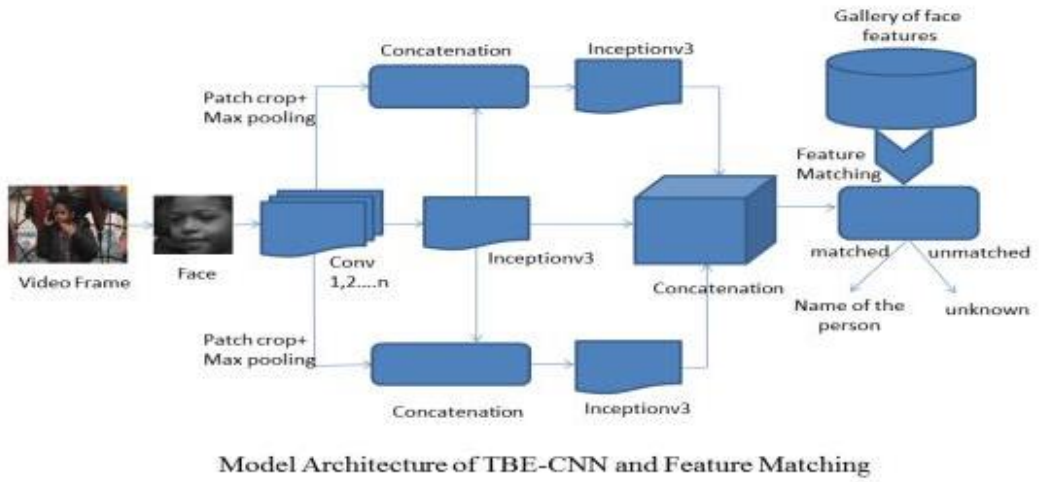


Figure 4.3.1.3 Model Architecture of TBE-CNN and Feature Matching

CHAPTER 5
IMPLEMENTATION DETAILS AND RESULTS

5.1 DATASET

Since the proposed project revolves around recognizing individuals from a livestream, an own dataset was composed. The Video Acquisition module built in the project is responsible for achieving this seamless data capture. Video of the particular person is captured through a webcam and the individual frames are extracted from this processed video and stored in a repository after labelling it with the name of the corresponding individual. Similarly, the datastore is scaled to store images of various persons along with their names.

5.2 IMPLEMENTATION AND RESULTS

5.2.1 Module 1: Video Acquisition and Pre-processing

```
In [5]: import numpy as np
        import pandas as pd
        import time
        import matplotlib.pyplot as plt
        %matplotlib inline
        import cv2
        %run image_pyramid.ipynb
        %run Histogram_Equilization.ipynb
```

Initially, the necessary packages and libraries for further processing are imported. NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labelled” data both easy and intuitive.

Python has defined a module, “time” which allows us to handle various operations regarding time, its conversions and representations, which find its use in various applications in life. Matplotlib is a cross-platform, data visualisation and graphical plotting library for Python. Cv2 is the module import name for OpenCV. OpenCV is a great tool for image processing and performing computer vision tasks. It is an open-source library that can be used to perform tasks like face detection, objection tracking, landmark detection, and much more.

```
In [3]: cap = cv2.VideoCapture(0)

        arr = np.empty((0, 1944), int)
```

OpenCV allows a straightforward interface to capture live streams with the camera (webcam) through the VideoCapture function. The empty() function is used to create a new array of given shape and type, without initialising entries. We use np.empty() to create an array with 0 rows [each row represents each frame in the input video, and rows will be added during processing of input video; if there are 10 frames in the video, 10 rows will be appended] and 1944 columns for storing color histogram values.

```
D=dict()   #to store the original frame (array)
count=0    #counting the number of frames
start_time = time.time()
```

Python dict() Function is used to create a Python dictionary. A dictionary is a collection of key-value pairs. We declare a variable count to keep track of the number of frames. The Python time() function retrieves the current time to store it as the starting time.

```
while (True):

    # Read the video file.
    ret, frame = cap.read()

    # If we got frames.
    if ret == True:

        #Image Pyramiding
        pyramid_list = imgPyramid(frame)
```

Each frame is captured and the video file is read. To reduce the variability in the faces, the Image Pyramiding technique is employed as shown below.

IMAGE PYRAMIDING

```
In [3]: import cv2
        import numpy as np

        def imgPyramid(img):
            layer = img.copy()
            gaussian_pyramid_list = [layer]

            for i in range(4):
                layer = cv2.pyrUp(layer)
                gaussian_pyramid_list.append(layer)

            return gaussian_pyramid_list
```

PyrDown() function is used to create 4 different layers of each frame with varying sizes and resolutions.

A Sample Output for Image Pyramiding is shown below in Figure 5.2.1.1

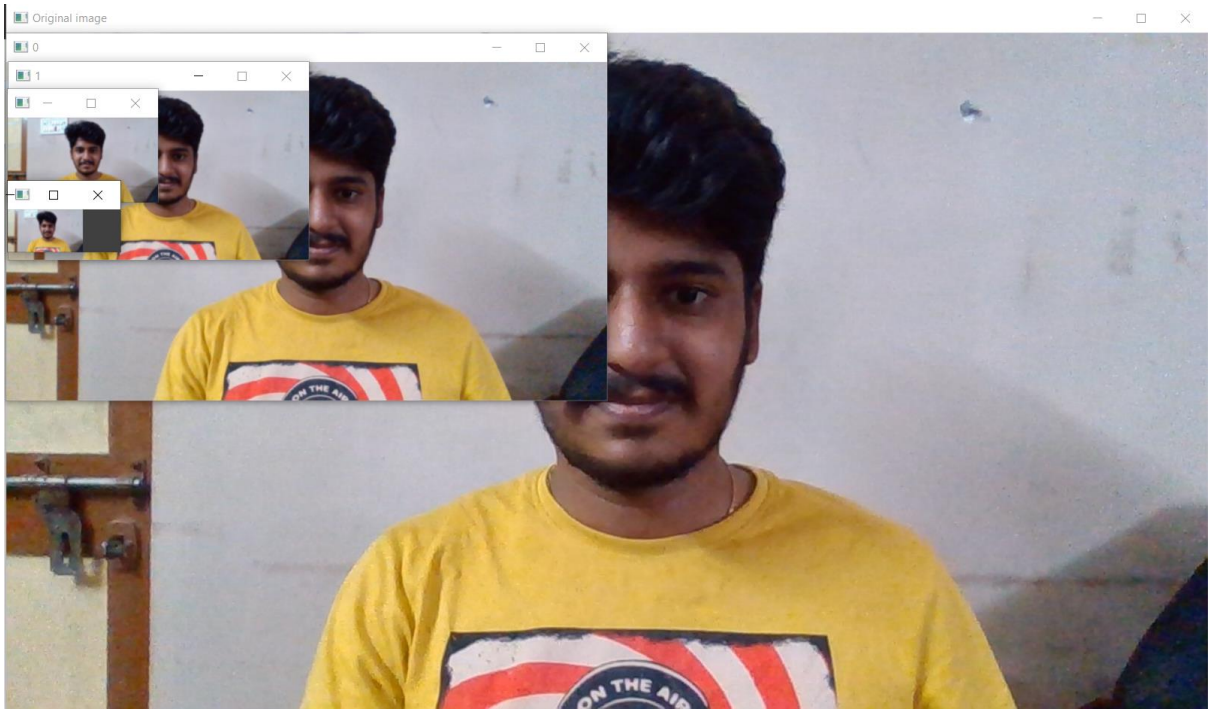


Figure 5.2.1.1 Sample output after applying Image Pyramiding over an input image

After Image Pyramiding, the next step called Histogram Equalization is applied.

```
for frame in pyramid_list:
    #BGR to RGB converion
    frame= cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)  #since cv reads frame in bgr order so rearraning to get frames in rgb

    #Histogram Equilization
    frame_rgb = hist_equilization(frame)

    D[count] = frame_rgb   #storing each frame (array) to D , so that we can identify key frames later
```

BGR stands for Blue(255, 0, 0), Green(0, 255, 0), Red(0, 0, 255). OpenCV uses BGR color as a default color space to display images, when we open an image or video in openCV using cv2. The main difference between RGB versus BGR is the arrangement of the subpixels for Red, Green, and Blue. But since we need the frames to be in RGB format for further processing, we use cvtColor() method to convert the frames from BGR format to RGB order.

Now, histogram equalization is performed over the frames to enhance the frames' contrast.

HISTOGRAM EQUALIZATION

```
In [5]: import cv2 as cv
        import numpy as np
        from matplotlib import pyplot as plt

        def hist_equilization(img):

            hist,bins = np.histogram(img.flatten(),256,[0,256])
            cdf = hist.cumsum()
            cdf_normalized = cdf * float(hist.max()) / cdf.max()
            plt.plot(cdf_normalized, color = 'b')
            plt.hist(img.flatten(),256,[0,256], color = 'r')
            plt.xlim([0,256])
            plt.legend(('cdf','histogram'), loc = 'upper left')
            plt.show()

            R, G, B = cv.split(img)

            output1_R = cv.equalizeHist(R)
            output1_G = cv.equalizeHist(G)
            output1_B = cv.equalizeHist(B)

            equ = cv.merge((output1_R, output1_G, output1_B))

            hist,bins = np.histogram(equ.flatten(),256,[0,256])
            cdf = hist.cumsum()
            cdf_normalized = cdf * float(hist.max()) / cdf.max()
            plt.plot(cdf_normalized, color = 'b')
            plt.hist(equ.flatten(),256,[0,256], color = 'r')
            plt.xlim([0,256])
            plt.legend(('cdf','histogram'), loc = 'upper left')
            plt.show()

            return equ
```

A histogram is the best way to visualise the frequency distribution of a dataset by splitting it into small equal-sized intervals called bins. Numpy has a built-in numpy.histogram() function which represents the frequency of data distribution in the graphical form. The rectangles having equal horizontal size

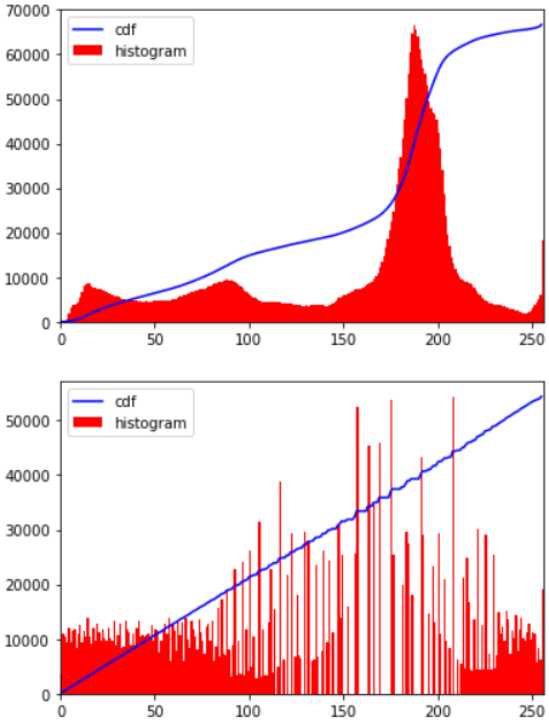
correspond to class intervals called bin and variable height corresponding to the frequency.

Flattening is a technique that is used to convert multi-dimensional arrays into a 1-D array. Each image frame is flattened into a 1-D array and given as a parameter to the histogram() function to find the frequency distribution of each frame. Since we use the RGB band, there'd be 256 intensities (0 to 255), so 256 bins are created representing each gray level on the X-axis. As the third parameter, the lower and upper range of bins are given.

Steps for Histogram Equalization

a) Find the running sum of the histogram values
b) Normalise the values from step1 by dividing by total number of pixels.
c) Multiply the values from step2 by the maximum gray level value and round off.
d) Map the gray-level values to the results from step 3, using a one-to one correspondence.

To implement these steps in Python, several functions are used. The cumsum() function is used to compute the cumulative sum of array elements over a given axis. The xlim() function in the pyplot module of matplotlib library is used to get or set the x-limits of the current axes. cv2.split() is used to split coloured/multi-channel images into separate single-channel images. Histogram equalization improves the contrast of an image, in order to stretch out the intensity range. You can equalise the histogram of a given image using the equalizeHist() function. A sample histogram transformation is shown in Figure 5.2.1.2



x axis - intensity
y axis - count

Figure 5.2.1.2 Transformation of Histogram of an image after applying Histogram Equalization over it

Then the histogram values are split into 3 bands and apply Histogram Equalization separately for them and finally merge them together. The histograms and CDFs of the frame before and after applying Equalization are shown above.

Sample Output for Histogram Equalization is shown in Figure 5.2.1.3 and Figure 5.2.1.4

ORIGINAL IMAGE

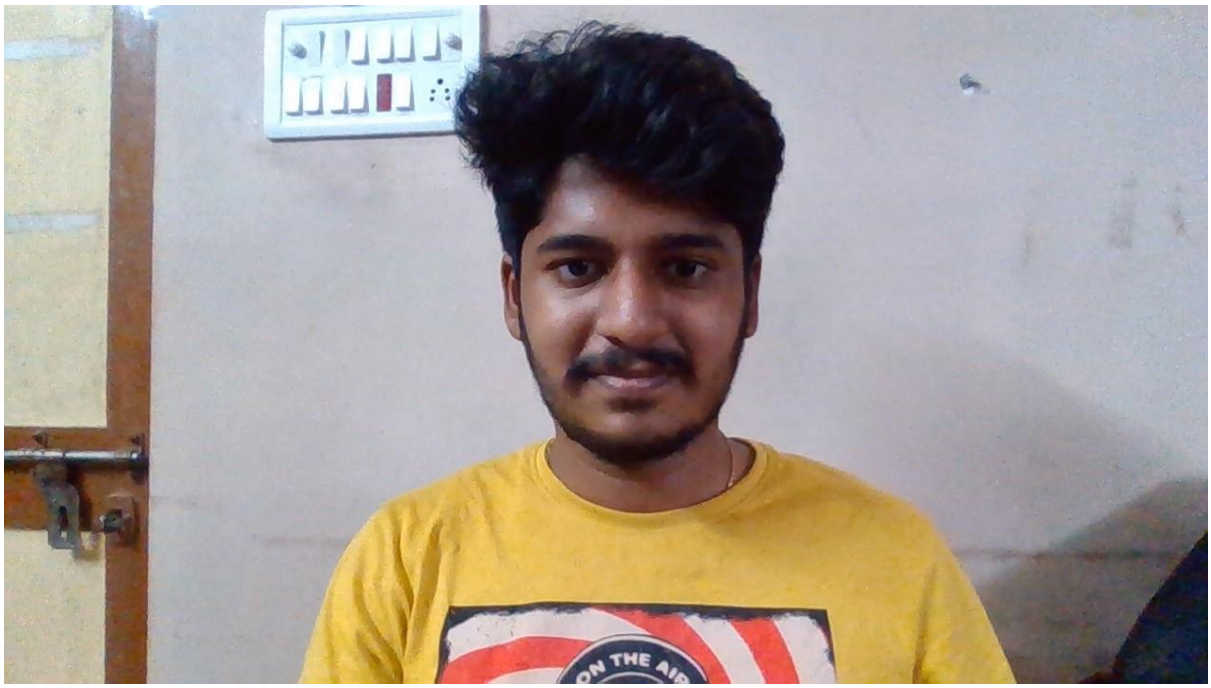


Figure 5.2.1.3 Input image before applying Histogram Equalization

HISTOGRAM EQUALIZED IMAGE

Figure 5.2.1.4 Image after applying Histogram Equalization

5.2.2 Module 2: Face Detection

KEY-FRAME EXTRACTION

```
    D[count] = frame_rgb   #storing each frame (array) to D , so that we can identify key frames later


    #dividing a frame into 3*3 i.e 9 blocks
    height, width, channels = frame_rgb.shape

    if height % 3 == 0:
        h_chunk = int(height/3)
    else:
        h_chunk = int(height/3) + 1
```

```
  if width % 3 == 0:
      w_chunk = int(width/3)
  else:
      w_chunk = int(width/3) + 1
```

After applying Histogram Equalization to all frames, they are stored into D. The Python numpy module has a shape function, which helps us to find the shape or size of an array or matrix. That function is used to get the height, width and channels of each frame.

```
    h=0
    w= 0
    feature_vector = []
    for a in range(1,4):
        h_window = h_chunk*a
        for b in range(1,4):
            frame = frame_rgb[h : h_window, w : w_chunk*b , :]
            hist = cv2.calcHist(frame, [0, 1, 2], None, [6, 6, 6], [0, 256, 0, 256, 0, 256])#finding histograms for each bloc
            hist1= hist.flatten()  #flatten the hist to one-dimensinal vector
            feature_vector += list(hist1)
            w = w_chunk*b

        h = h_chunk*a
        w= 0


    arr =np.vstack((arr, feature_vector )) #appending each one-dimensinal vector to generate N*M matrix (where N is number of
      #and M is 1944)
    count+=1
else:
    break
```

Each frame is divided into 9 blocks and the histogram of each block is calculated for all 3 of its channels and flatten it into a one-dimensional vector. Here, cv2.calcHist()(in-built function in OpenCV) is used to find the histogram.

Syntax: cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])

where,
images : it is the source image of type uint8 or float32 represented as “[img]”.
channels : it is the index of channels for which we calculate histogram. For grayscale image, its value is [0] and color image, you can pass [0], [1] or [2] to calculate histogram of blue, green or red channels respectively.
mask : mask image. To find a histogram of full image, it is given as “None”.
histSize : this represents our BIN count. For full scale, we pass [256].
ranges : this is our RANGE. Normally, it is [0,256].

The numpy.vstack() function is used to stack the sequence of input arrays vertically to make a single array. Now, the one-dimensional vector of each frame is appended as new rows in the previously created feature-frame matrix.

```
    cv2.imshow('image',ret)
    k= cv2.waitKey(100) & 0xff

    if k == 27:
        break

print("--- %s seconds ---" % (time.time() - start_time))

final_arr = arr.transpose() #transposing so that i will have all frames in columns i.e M*N dimensional matrix
#where M is 1944 and N is number of frames
print(final_arr.shape)
print(count)
```

imshow() method is used to display an image in a window. The window automatically fits the image size. The ASCII Code for Escape key is 27. So, until Esc key is pressed, the webcam will be capturing video.

The number of seconds taken for the code to run is printed along with the shape of the transpose of the feature-frame matrix. In our sample, the number of frames in the captured video is 233.

```
--- 27.20482301712036 seconds ---
(1944, 233)
233
```

Now move to the next stage called “Singular Value Decomposition”,

```
In [12]: from scipy.sparse import csc_matrix
         from scipy.sparse.linalg import svds, eigs
         A = csc_matrix(final_arr, dtype=float)

         u, s, vt = svds(A, k = 63)
```

```
In [4]: print(u.shape, s.shape, vt.shape)
```

```
(1944, 63) (63,) (63, 1832)
```

```
In [5]: print(list(s))
```

```
[507.58633979103655, 513.3394019469036, 542.5885461980113, 557.461573026448, 581.1252578281178, 595.6523491133432, 625.8676251128798, 667.3543038445863, 703.9369306867552, 785.0033226440822, 824.124029717276, 830.2095332996873, 862.3243911201392, 962.167800588134, 993.188433535317, 1074.4879221595838, 1104.0363306894762, 1174.3795871198802, 1310.4024628743787, 1429.8059729265503, 1436.0497628725557, 1784.083546388292, 1920.5743747040262, 2075.5124822673397, 2235.9930038951484, 2450.8504358624887, 2789.861284247546, 3266.4378720353525, 3467.454332697814, 3703.2744228804345, 4026.5120288278313, 4160.600410916418, 4339.992564704211, 4608.340768433539, 4872.711772927273, 5124.862670258989, 5276.309542523109, 5668.146768603114, 5931.507922089641, 5952.291087578098, 6235.342567902848, 6697.298321083391, 6776.947714481707, 7065.201258998802, 7853.121480587885, 8067.823543248019, 8691.548708604729, 9140.661744359926, 9938.174572146694, 10316.757223295494, 10792.030201567124, 11187.863261213428, 11687.075188310106, 13425.115571459452, 13830.826759645013, 14644.218274242545, 15148.499757824604, 16712.59359000954, 17776.734430885106, 22769.412721247292, 30817.597542113654, 45793.98450615915, 76082.29505434036]
```

```
In [6]: v1_t = vt.transpose()

        projections = v1_t @ np.diag(s) #the column vectors i.e the frame histogram data has been projected onto the orthonormal basis
        #formed by vectors of the left singular matrix u .The coordinates of the frames in this space are given by v1_t @ np.diag(s)
        #So we can see that , now we need only 63 dimensions to represent each column/frame
        print(projections.shape)
```

```
(1832, 63)
```

SVD technique is used to decompose our feature-frame matrix into U, Sigma and V^T matrices. Later only the top 63 singular values are chosen and ignore the rest, thus achieving data reduction as intended.

```
#dynamic clustering of projected frame histograms to find which all frames are similar i.e make shots
f=projections
C = dict() #to store frames in respective cluster
for i in range(f.shape[0]):
    C[i] = np.empty((0,63), int)

#adding first two projected frames in first cluster i.e Initializaton
C[0] = np.vstack((C[0], f[0]))
C[0] = np.vstack((C[0], f[1]))

E = dict() #to store centroids of each cluster
for i in range(projections.shape[0]):
    E[i] = np.empty((0,63), int)

E[0] = np.mean(C[0], axis=0) #finding centroid of C[0] cluster

count = 0
for i in range(2,f.shape[0]):
    similarity = np.dot(f[i], E[count])/( (np.dot(f[i],f[i]) **.5) * (np.dot(E[count], E[count]) ** .5)) #cosine similarity
    #this metric is used to quantify how similar is one vector to other. The maximum value is 1 which indicates they are same
    #and if the value is 0 which indicates they are orthogonal nothing is common between them.
    #Here we want to find similarity between each projected frame and last cluster formed chronologically.


    if similarity < 0.9: #if the projected frame and last cluster formed  are not similar upto 0.9 cosine value then
                         #we assign this data point to newly created cluster and find centroid
                         #We checked other thresholds also like 0.85, 0.875, 0.95, 0.98
                        #but 0.9 looks okay because as we go below then we get many key-frames for similar event and
                        #as we go above we have lesser number of key-frames thus missed some events. So, 0.9 seems optimal.

        count+=1
        C[count] = np.vstack((C[count], f[i]))
        E[count] = np.mean(C[count], axis=0)
```

Through Dynamic Clustering, frames are segregated into clusters depending on their similarity. To achieve this, “Cosine Similarity Check.” is employed.

```
    else:  #if they are similar then assign this data point to last cluster formed and update the centroid of the cluster
        C[count] = np.vstack((C[count], f[i]))
        E[count] = np.mean(C[count], axis=0)
```

```
b = []  #find the number of data points in each cluster formed.

#We can assume that sparse clusters indicates
#transition between shots so we will ignore these frames which lies in such clusters and wherever the clusters are densely populc
#and we can take the last element of these shots to summarise that particular shot

for i in range(f.shape[0]):
    b.append(C[i].shape[0])

last = b.index(0)  #where we find 0 in b indicates that all required clusters have been formed , so we can delete these from C
b1=b[:last ] #The size of each cluster.
```

```
res = [idx for idx, val in enumerate(b1) if val >= 25] #so i am assuming any dense cluster with atleast 25 frames is eligible to
#make shot.
print(len(res))
```

```
25
```

It is assumed that sparse clusters indicate transition between shots so we will ignore these frames which lie in such clusters and wherever the clusters are densely populated indicate they form shots and the last element of these shots can be taken to summarise that particular shot.

```
GG = C #copying the elements of C to GG, the purpose of  the below code is to label each cluster so later
#it would be easier to identify frames in each cluster
for i in range(last):
    p1= np.repeat(i, b1[i]).reshape(b1[i],1)
    GG[i] = np.hstack((GG[i],p1))
```

```
#the purpose of the below code is to append each cluster to get multidimensional array of dimension N*64, N is number of frames
F=  np.empty((0,64), int)
for i in range(last):
    F = np.vstack((F,GG[i]))
```

```
#converting F (multidimensional array)  to dataframe

colnames = []
for i in range(1, 65):
    col_name = "v" + str(i)
    colnames+= [col_name]
print(colnames)

df = pd.DataFrame(F, columns= colnames)
```

```
['v1', 'v2', 'v3', 'v4', 'v5', 'v6', 'v7', 'v8', 'v9', 'v10', 'v11', 'v12', 'v13', 'v14', 'v15', 'v16', 'v17', 'v18', 'v19', 'v
20', 'v21', 'v22', 'v23', 'v24', 'v25', 'v26', 'v27', 'v28', 'v29', 'v30', 'v31', 'v32', 'v33', 'v34', 'v35', 'v36', 'v37', 'v3
8', 'v39', 'v40', 'v41', 'v42', 'v43', 'v44', 'v45', 'v46', 'v47', 'v48', 'v49', 'v50', 'v51', 'v52', 'v53', 'v54', 'v55', 'v5
6', 'v57', 'v58', 'v59', 'v60', 'v61', 'v62', 'v63', 'v64']
```

Now, each cluster is labelled to identify frames in them seamlessly. Finally, depending on the above mentioned criteria, we extract the frames of significance from different clusters successfully.

```
In [14]: df['v64']= df['v64'].astype(int)  #converting the cluster level from float type to integer type

In [15]: df1 =  df[df.v64.isin(res)]   #filter only those frames which are eligible to be a part of shot or filter those frames who are
         #part of required clusters that have more than 25 frames in it

In [16]: new = df1.groupby('v64').tail(1)['v64'] #For each cluster /group take its last element which summarize the shot i.e key-frame

In [17]:
         new1 = new.index #finding key-frames (frame number so that we can go back get the original picture)

In [18]: #output the frames in png format
         for c in new1:
             frame_rgb1 = cv2.cvtColor(D[c], cv2.COLOR_RGB2BGR) #since cv consider image in BGR order
             frame_num_chr = str(c)
             file_name = 'frame'+ frame_num_chr
             cv2.imwrite("dataset/User." + str(file_name) + '.' + str(count) + ".png")

In [19]: facesamples,id = haarcascades(dataset)

In [20]: getBranches(facesamples) #get branches from eyes,nose and mouth
```

The keyframes are outputted along with the frame numbers as displayed below in Figure 5.2.2.1:

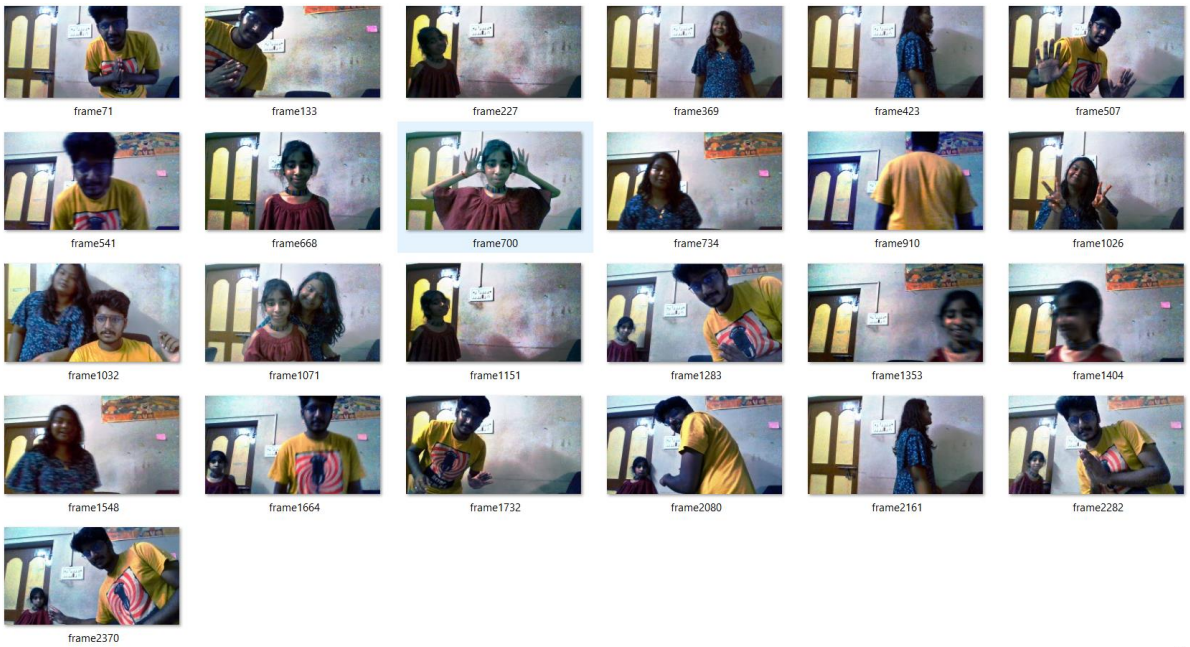


Figure 5.2.2.1 Sample Output for Keyframe Extraction

RGB to Gray-scale Conversion

Initially, the image is a three-layer image (i.e., RGB), So It is converted to a one-layer image (i.e., grayscale).

Haar Cascade Classifier

The CascadeClassifier method in the cv2 module supports the loading of haar-cascade XML files.

```
In [20]: import cv2
         import numpy as np
         from PIL import Image
         import os
         # Path for face image database
         path = 'dataset'

         recognizer = cv2.face.LBPHFaceRecognizer_create()
         detector = cv2.CascadeClassifier("haarcascade_frontalface_default.xml");
```

Here, “haarcascade_frontalface_default.xml” for face detection is needed.

```
# function to get the images and label data
def getImagesAndLabels(path):

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

    for imagePath in imagePaths:

        PIL_img = Image.open(imagePath).convert('L') # convert it to grayscale
        img_numpy = np.array(PIL_img,'uint8')

        id = int(os.path.split(imagePath)[-1].split(".")[1])
        faces = detector.detectMultiScale(img_numpy)
```

Detection is done using the cv2::CascadeClassifier::detectMultiScale method, which returns boundary rectangles for the detected faces (i.e., x, y, w, h). It takes two parameters namely, scaleFactor and minNeighbors. ScaleFactor determines the factor of increase in window size which initially starts at size “minSize”, and after testing all windows of that size, the window is scaled up by the “scaleFactor”, and the window size goes up to “maxSize”.

If the “scaleFactor” is large, (e.g., 2.0), there will be fewer steps, so detection will be faster, but we may miss objects whose size is between two tested scales. (default scale factor is 1.3). Higher the values of the “minNeighbors”, less

will be the number of false positives, and less error will be in terms of false detection of faces.

```
        for (x,y,w,h) in faces:
            faceSamples.append(img_numpy[y:y+h,x:x+w])
            ids.append(id)

    return faceSamples,ids

print ("\n [INFO] Training faces. It will take a few seconds. Wait ...")
faces,ids = getImagesAndLabels(path)
recognizer.train(faces, np.array(ids))
```

```
#Save model = '1trainer.yml'
recognizer.write('train.yml') # recognizer.save() worked on Mac, but not on Pi

# Print the numer of faces trained and end program
print("\n [INFO] {0} faces trained. Exiting Program".format(len(np.unique(ids))))
```

```
[INFO] Training faces. It will take a few seconds. Wait ...

[INFO] 2 faces trained. Exiting Program
```

In [21]:

```
import cv2
import numpy as np
import os

#recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer = cv2.face.LBPHFaceRecognizer_create()
#recognizer = cv2.face.createLBPHFaceRecognizer()
recognizer.read('C:/Users/Dr.K.Latha.MD/Downloads/project/train.yml')
cascadePath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath);

font = cv2.FONT_HERSHEY_SIMPLEX

#iniciate id counter
id = 0,1,2,3
```

```
# Initialize and start realtime video capture
cam = cv2.VideoCapture(0)
cam.set(3, 640) # set video widht
cam.set(4, 480) # set video height

# Define min window size to be recognized as a face
minW = 0.1*cam.get(3)
minH = 0.1*cam.get(4)

while True:

    ret, img =cam.read()
    # img = cv2.flip(img, -1) # Flip vertically

    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

```
    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor = 1.2,
        minNeighbors = 5,
        minSize = (int(minW), int(minH)),
       )

    for(x,y,w,h) in faces:
        cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)

    cv2.imshow('camera',img)

    k = cv2.waitKey(10) & 0xff # Press 'ESC' for exiting video
    if k == 27:
        break

# Do a bit of cleanup
print("\n [INFO] Exiting Program and cleanup stuff")
cam.release()
cv2.destroyAllWindows()
```

Rectangles are drawn around the detected faces by the rectangle method of the cv2 module by iterating over all detected faces.

```
[INFO] Exiting Program and cleanup stuff
```

```
In [22]: merged = getBranches(img) #get branches from eyes,nose and mouth
```

A sample output for Face Detection using Haar Cascade Classifier is shown below in Figure 5.2.2.2 and Figure 5.2.2.3

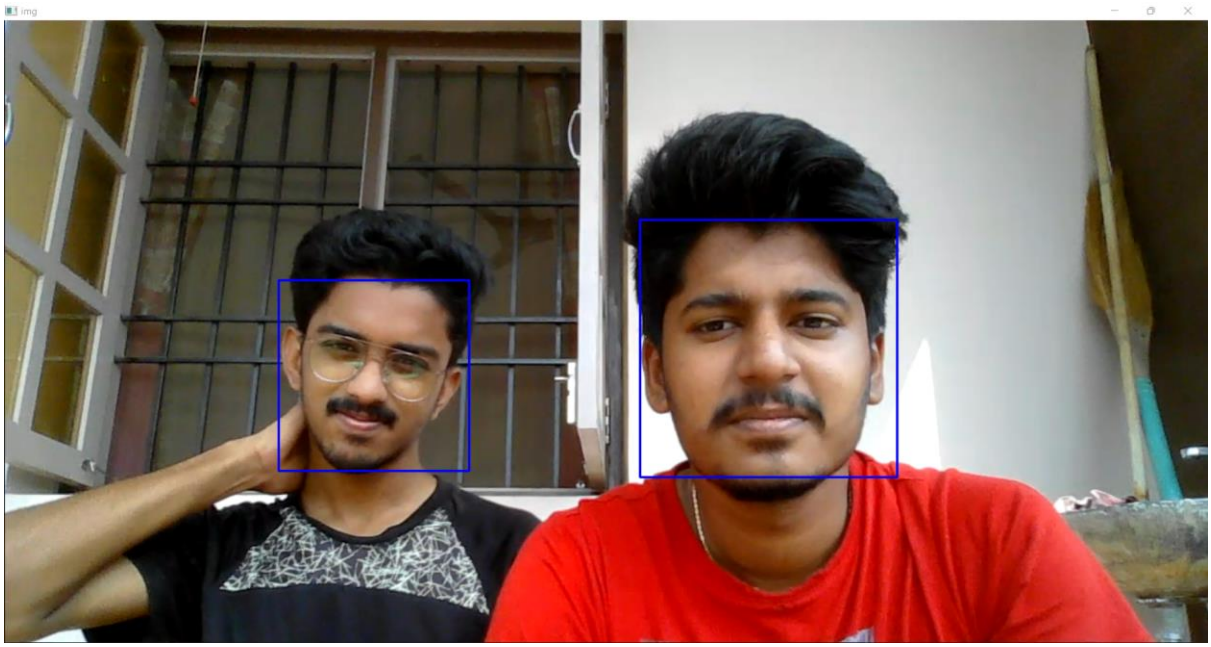


Figure 5.2.2.2 Faces enclosed within bounding boxes after applying Haar Cascade classifier over input key frame.

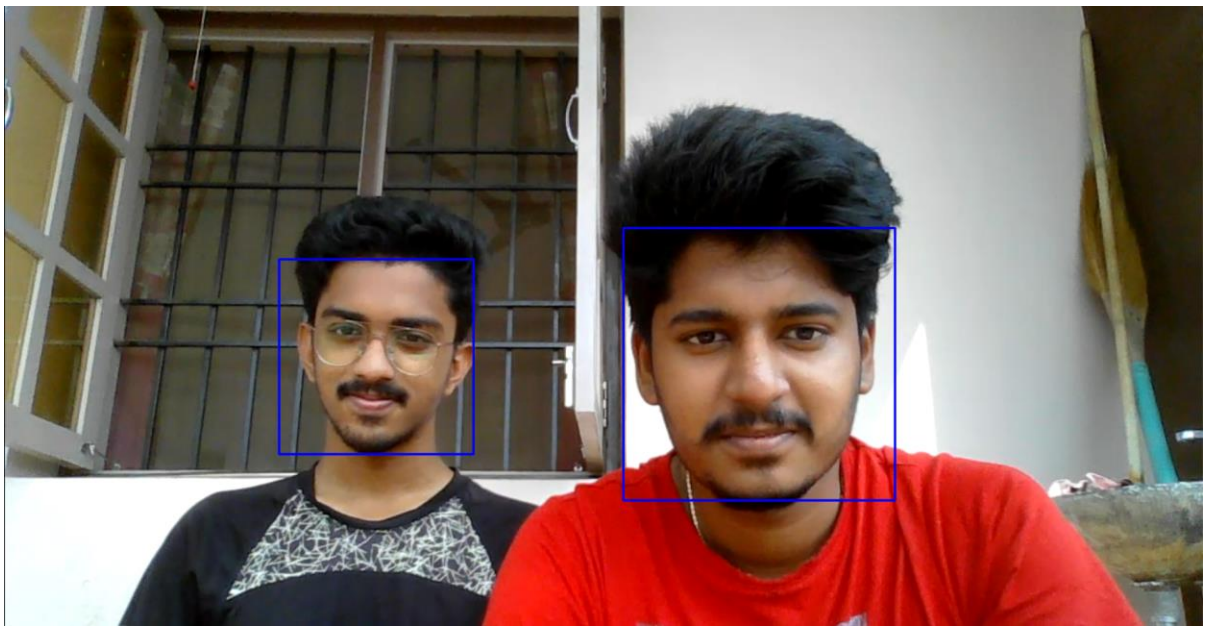


Figure 5.2.2.3 Faces enclosed within bounding boxes after applying Haar Cascade classifier over input key frame.

5.2.3 MODULE 3:- FACE RECOGNITION

Trunk Branch Ensemble CNN

The proposed trunk network extracts features from new pixels. On top of Inception of the trunk network, we reduce the size of its feature maps by max pooling. For each branch network, we directly crop feature maps from Conv2 and Inception 3 module outputs of the trunk network instead of computing low- and middle-level features from scratch.

```
In [6]: from tensorflow.python.keras.layers import Conv2D, MaxPooling2D, Concatenate, Input
        import os

        def getBranches(facesamples):
            # # Just disables the warning, doesn't enable AVX/FMA (no GPU)
            os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'


            def get_model(width, height):
                input_img = Input(shape=(width, height, 3))

                conv1_convolution = Conv2D(64, (7, 7), strides=2, data_format='channels_last', activation='relu', padding='same', name='
                conv1 = MaxPooling2D(data_format='channels_last', padding='same', strides=2, pool_size=(2, 2), name='conv1')(conv1_convo

                conv2_convolution = Conv2D(192, (3, 3), strides=2, data_format='channels_last', activation='relu', padding='same', name=
                conv2 = MaxPooling2D(data_format='channels_last', padding='same', strides=1, pool_size=(2, 2), name='conv2')(conv2_convo

                inception3a_activation = inception.with_dimension_reduction(conv2, 64, False, name='inception3a_activation')
                inception3 = inception.with_dimension_reduction(inception3a_activation, 120, True, name='inception3')
```

The size of the feature maps cropped from the Conv2 output is reduced by a half by max pooling and then concatenated with the feature maps cropped from the Inception 3 output. The concatenated feature maps form the input of the branch network.

```
        ######################################################## Branch 1 ########################################################

        inception4b_activation = inception.with_dimension_reduction(inception3, 128, False, name='inception4b_activation_branch_
        inception4e_activation = inception.with_dimension_reduction(inception4b_activation, 132, False, name='inception4e_activa
        inception4 = inception.with_dimension_reduction(inception4e_activation, 208, True, name='inception4_branch_1')
        inception5a_activation = inception.with_dimension_reduction(inception4, 208, False, name='inception5a_activation_branch_
        inception5b_2 = inception.with_dimension_reduction(inception5a_activation, 256, True, name='inception5b_2_branch_1')

        cv2.imwrite("branch1/User." + str(face_id) + '.' + str(count) + ".jpg", gray[y:y+h,x:x+w])
```

The output feature maps of the trunk network and branch networks are fused by concatenation to form an over-complete face representation, whose dimension is reduced to 512 by one fully connected layer.

```
        ################################################## Branch 2 ##################################################

        inception4c_activation = inception.with_dimension_reduction(inception3, 128, False, name='inception4c_activation_branch_
        inception4e_activation = inception.with_dimension_reduction(inception4c_activation, 132, False, name='inception4e_activa
        inception4 = inception.with_dimension_reduction(inception4e_activation, 208, True, name='inception4_branch_2')
        inception5a_activation = inception.with_dimension_reduction(inception4, 208, False, name='inception5a_activation_branch_
        inception5b_3 = inception.with_dimension_reduction(inception5a_activation, 256, True, name='inception5b_3_branch_2')

        cv2.imwrite("branch2/User." + str(face_id) + '.' + str(count) + ".jpg", gray[y:y+h,x:x+w])

        ################################################## Branch 3 ##################################################

        inception4d_activation = inception.with_dimension_reduction(inception3, 128, False, name='inception4d_activation_branch_
        inception4e_activation = inception.with_dimension_reduction(inception4d_activation, 132, False, name='inception4e_activa
        inception4 = inception.with_dimension_reduction(inception4e_activation, 208, True, name='inception4_branch_3')
        inception5a_activation = inception.with_dimension_reduction(inception4, 208, False, name='inception5a_activation_branch_
        inception5b_4 = inception.with_dimension_reduction(inception5a_activation, 256, True, name='inception5b_4_branch_3')

        cv2.imwrite("branch3/User." + str(face_id) + '.' + str(count) + ".jpg", gray[y:y+h,x:x+w])

        merged_branch = Concatenate(axis=1)([
            inception5b_3,
            inception5b_4,
        ])

        merged = Concatenate(axis=1)([
            inception5b_2,
            merged_branch
        ])

        return merged
```

The 512-dimensional feature vector is utilised as the final face representation of one video frame. The main goal of implementing this project is to find the thieves in the CCTV footage. Comparison is done to find out whether our query face belongs to an existing dataset or not. Dynamic Feature Matching (DFM) [11] algorithm takes face features, which are trained by the TBE-CNN model and finally gives the matched face from the gallery of face features. It calculates the total features matched in given two faces and the total features of the two faces. Finally calculates the percentage of match of both faces from both the features.

5.3 TEST CASES AND VALIDATION

Test Case for Keyframe Extraction

INPUT

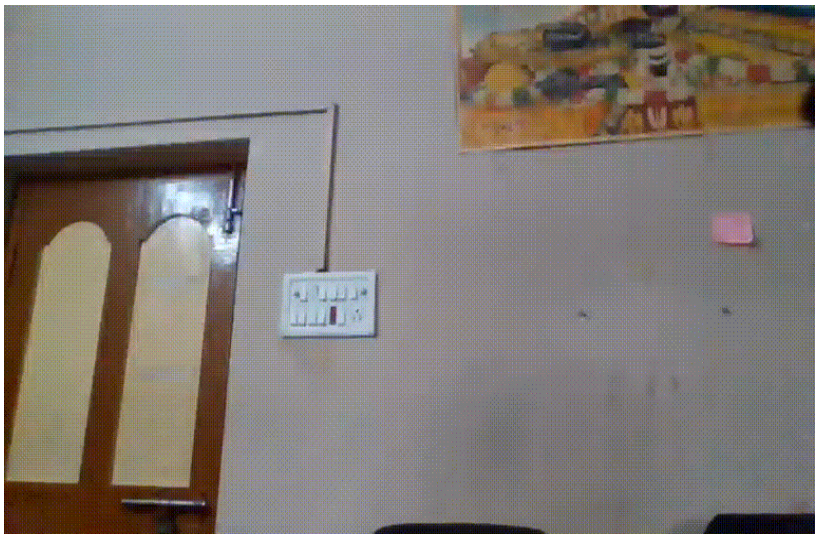


Figure 5.3.1 A dynamic input video

OUTPUT

Figure 5.3.2 Key frames extracted from a dynamic input video

Since the input video in Figure 5.3.1 is a dynamic one with several transitions and movements, Dynamic Clustering results in the formation of several clusters, and thus many keyframes get extracted as shown in Figure 5.3.2.

Test Case for Face Detection:

TESTE CASE 1

INPUT

Figure 5.3.3 A less dynamic input video

Output from Module 1

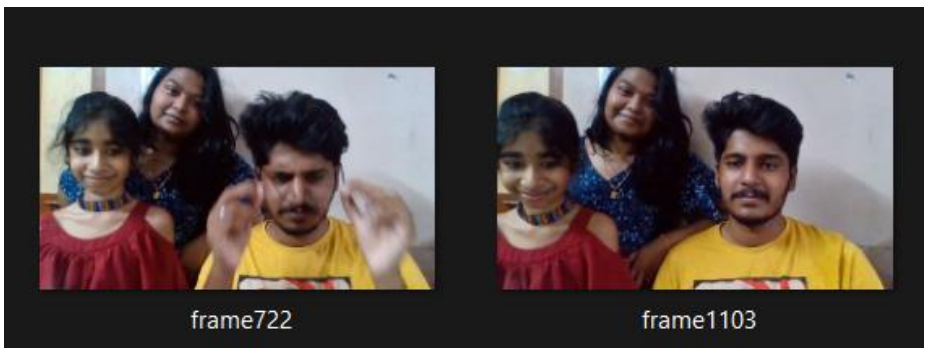


Figure 5.3.4 Keyframes extracted from a less-dynamic input video

Since the input video in Figure 5.3.3 is a less dynamic one, we have only a few transitions, hence only lesser number of keyframes get extracted as shown in Figure 5.3.4

Output from Module 2

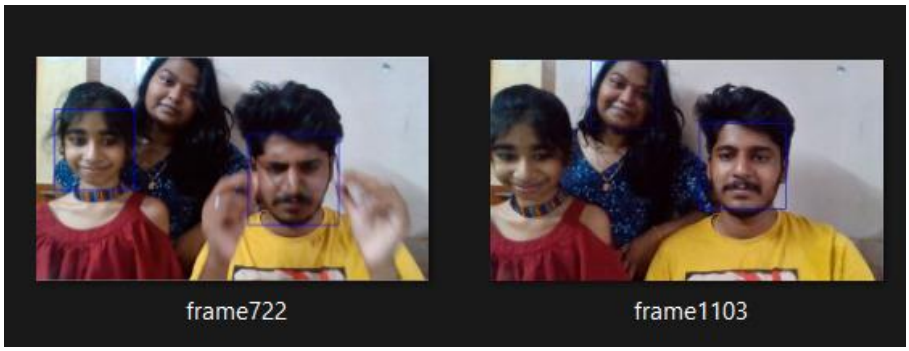


Figure 5.3.5 Bounding boxes enclosing Faces in the keyframes

Since the input video is a static one, the number of key frames extracted is comparatively lesser. The faces within the keyframes are identified and enclosed within bounding boxes as shown above in Figure 5.3.5.

TEST CASE 2

INPUT

Figure 5.3.6 An input video without any human faces

Output from Module 1

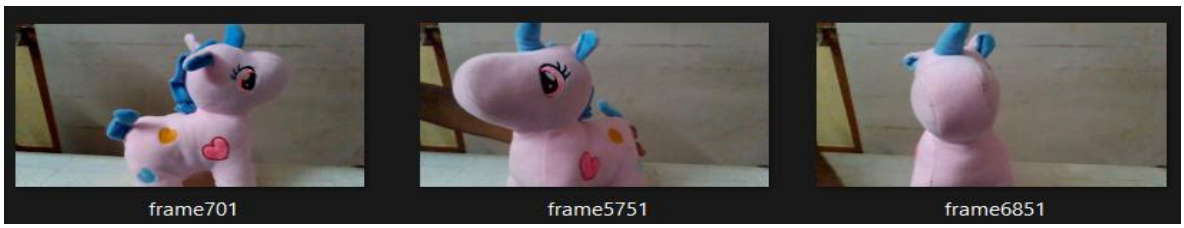


Figure 5.3.7 Keyframes extracted from the video without human faces

The keyframes extracted from input video is shown in Figure 5.3.7

Output from Module 2

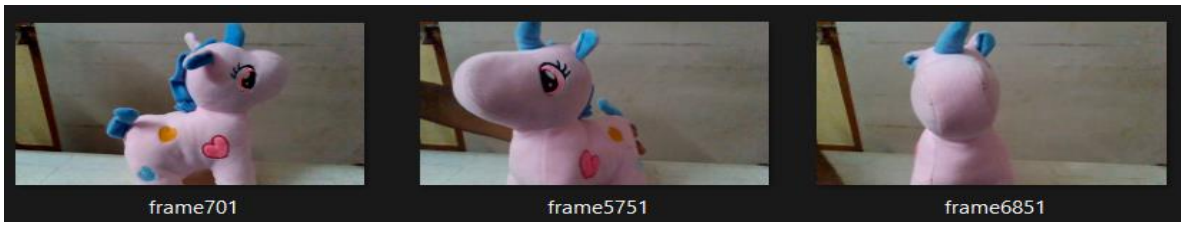


Figure 5.3.8 Keyframes with no bounding boxes as no faces get detected

Since there are no faces in the input video in Figure 5.3.6, there are no bounding boxes in the result as shown in Figure 5.3.8.

TEST CASE 3

INPUT

Figure 5.3.9 Input video taken in an outdoor environment

Output from Module 1

Figure 5.3.10 Keyframes extracted from video taken in outdoor environment

Output from Module 2

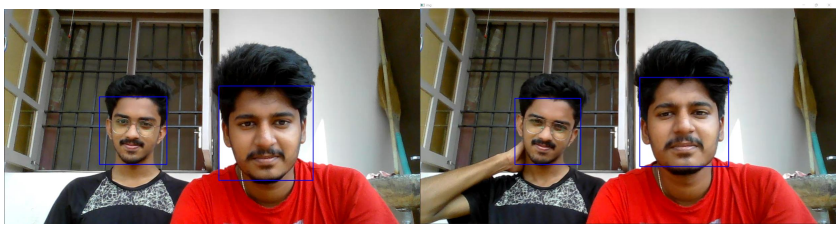


Figure 5.3.11 Faces detected in video taken in outdoor environment

The video in Figure 5.3.9 was taken in a well-lit outdoor environment; the keyframes extracted out of it are shown in Figure 5.3.10 and the faces detected in it are enclosed using bounding boxes in Figure 5.3.11

Test Cases for Face Recognition

TEST CASE 1

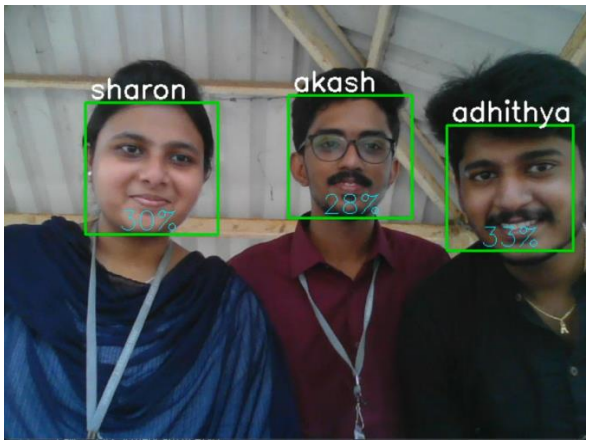


Figure 5.3.12 Output showing bounding boxes enclosing detected faces along with the names of the persons and the corresponding confidence values.

A sample output of the entire Face recognition network in shown in Figure 5.3.12

TEST CASE 2 (OCCLUSIONS IN OUTDOOR ENVIRONMENT)

Sample outputs of Face recognition model dealing with occlusions are shown in Figure 5.3.13, 5.3.14 and 5.3.15

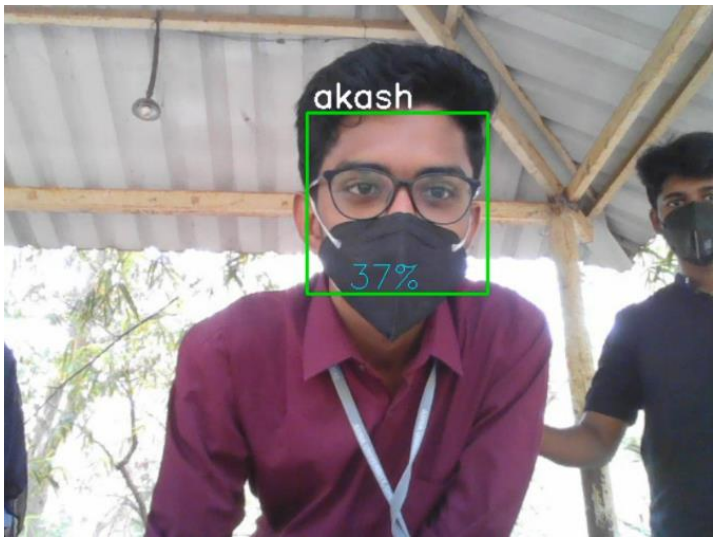


Figure 5.3.13 Face recognized even if occluded by mask

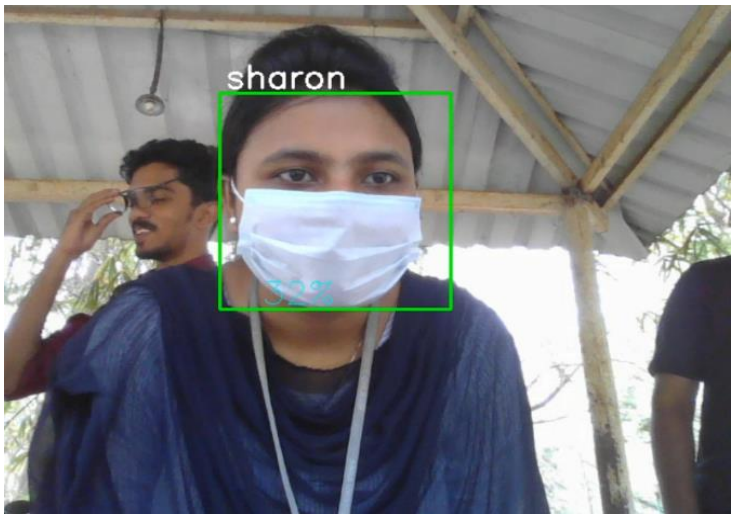


Figure 5.3.14 Face Recognition achieved even if face is partially occluded by mask

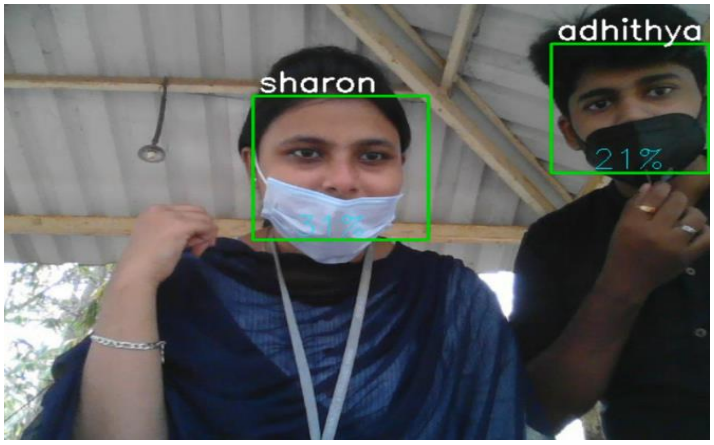


Figure 5.3.15 Face Recognition achieved even if faces are partially occluded by masks

This test case shows that our model recognizes faces even if they're occluded (i.e partially covered with mask in this case).

TEST CASE 3 (DARK ENVIRONMENT)

This test case shows sample outputs of the Face Recognition network that is able to recognize people even if they are present in a dark environment as shown in Figure 5.3.16, Figure 5.3.17 and Figure 5.3.18.

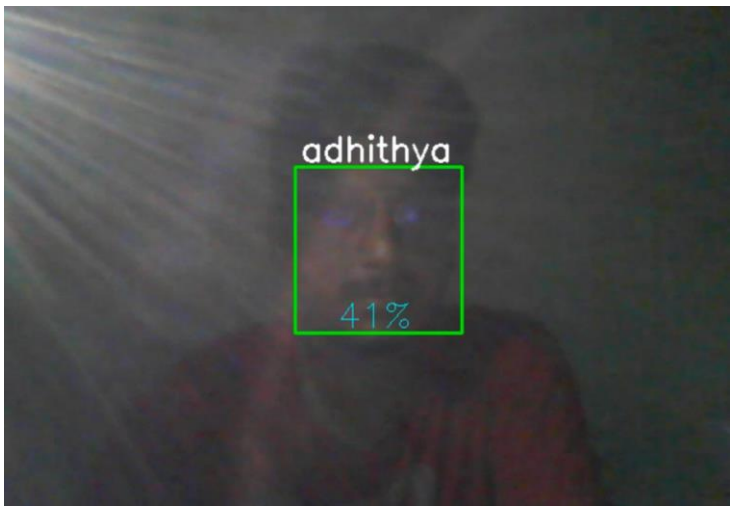


Figure 5.3.16 Face Recognition achieved even in Dark environment

Figure 5.3.17 Face Recognition achieved even in a Dark environment

Figure 5.3.18 Face Recognition achieved even in a Dark indoor environment

TEST CASE 4 (LIGHT ILLUMINATION CHANGES)

This testcase shows sample outputs of the Face Recognition network that is able to recognize people even if they are present in varying light illumination conditions as shown in Figure 5.3.19, Figure 5.3.20 and Figure 5.3.21.

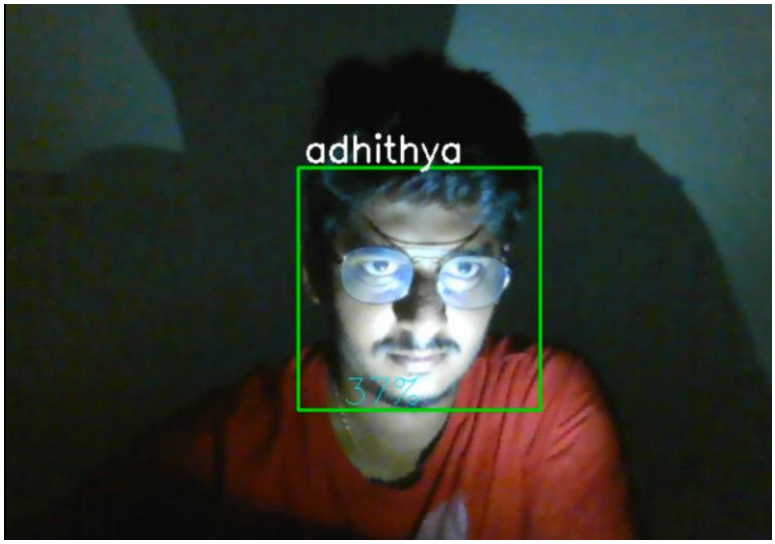


Figure 5.3.19 Face recognition achieved even in varying light illumination conditions.

Figure 5.3.20 Face recognition achieved in varying illumination conditions.

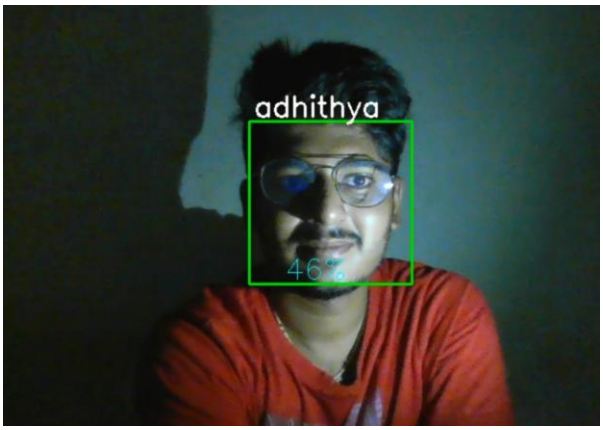


Figure 5.3.21 Face recognition achieved in varying illumination conditions.

TEST CASE 5 (EXPRESSION & POSE VARIATIONS)

This test case shows sample outputs of the Face Recognition network that is able to recognize people even if they are portraying varying poses and expressions as shown in Figure 5.3.22, Figure 5.3.23, Figure 5.3.24 and Figure 5.3.25.

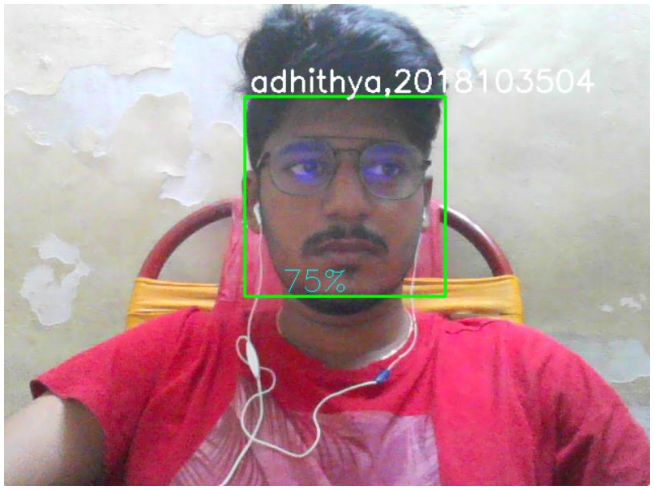


Figure 5.3.22 Face Recognition dealing with varying expressions and poses

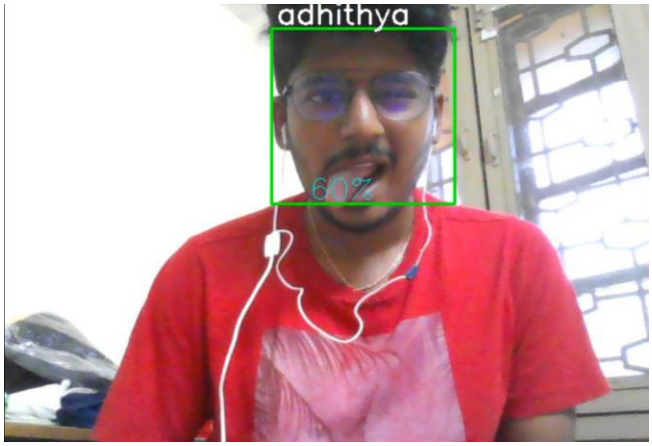


Figure 5.3.23 Face Recognition dealing with varying expressions and poses

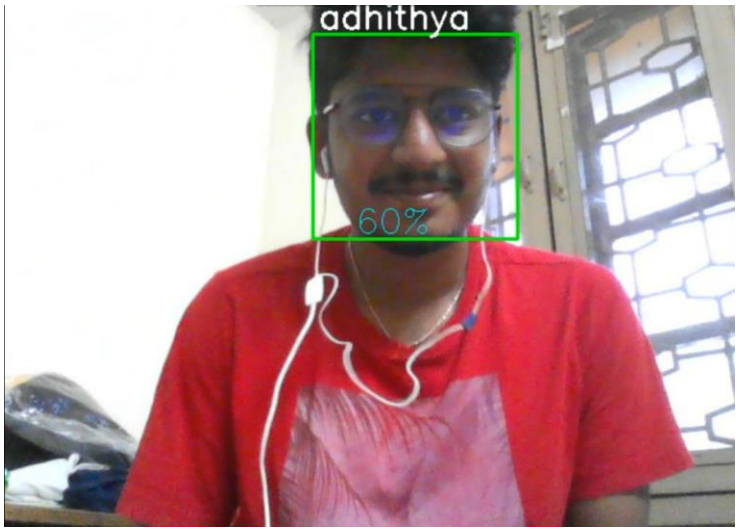


Figure 5.3.24 Face Recognition dealing with varying expressions and poses

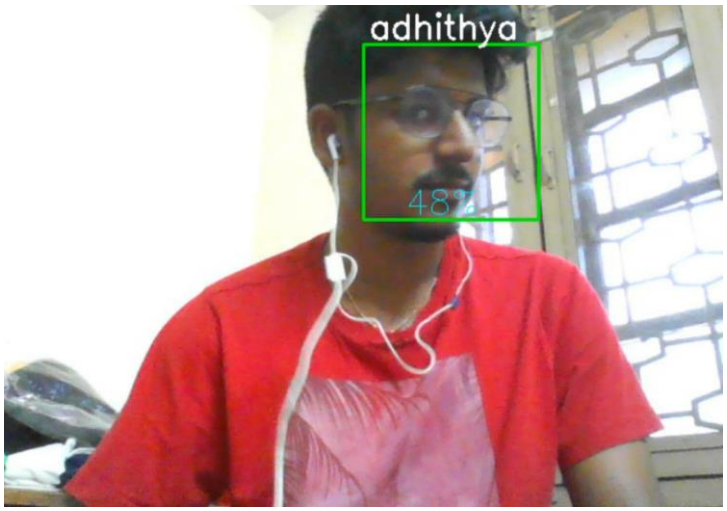


Figure 5.3.25 Face Recognition dealing with varying expressions and poses

TEST CASE 6 (OCCLUSIONS IN IN-DOOR ENVIRONMENT)

This test case shows sample outputs of the Face Recognition network that is able to recognize people even if their faces are partly occluded as shown in Figure 5.3.26, Figure 5.3.27, Figure 5.3.28, Figure 5.3.29, Figure 5.3.30, Figure 5.3.31, Figure 5.3.32, Figure 5.3.33 and Figure 5.3.34.

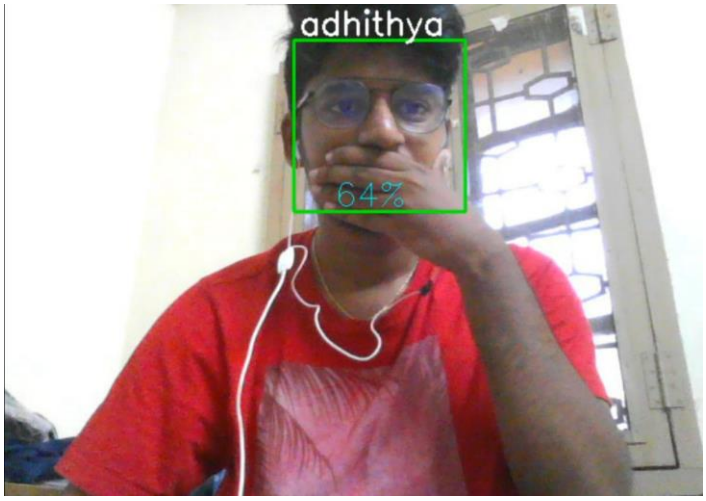


Figure 5.3.26 Face Recognition achieved even if face is partly occluded

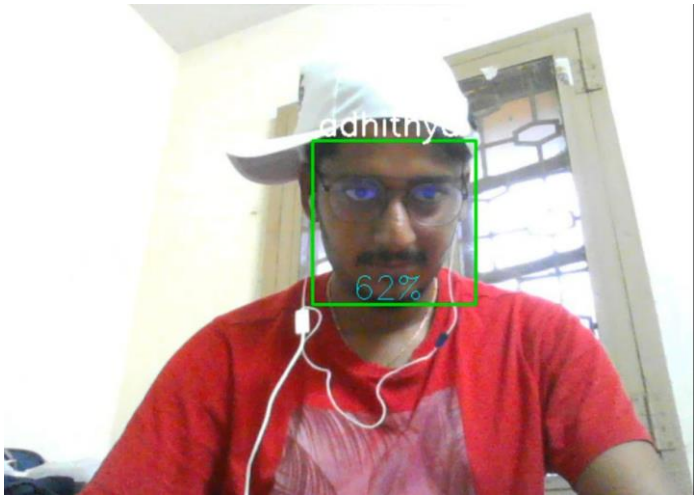


Figure 5.3.27 Face Recognition achieved even if additional materials like spectacles and cap are worn.

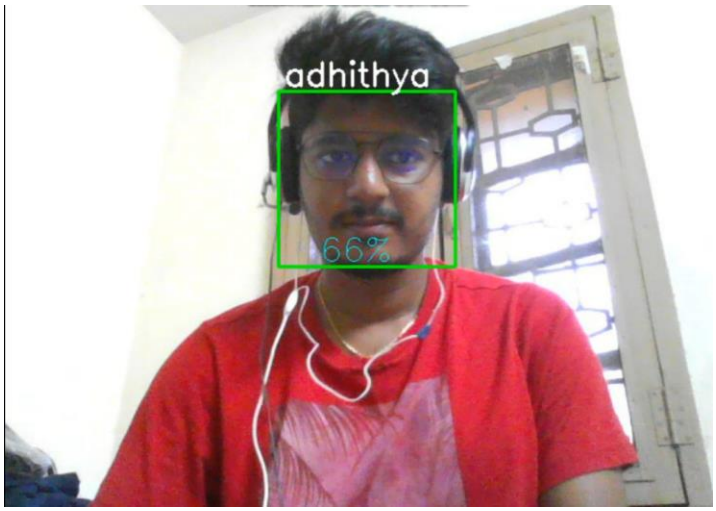


Figure 5.3.28 Face Recognition achieved even if additional objects like spectacles and headphones are worn

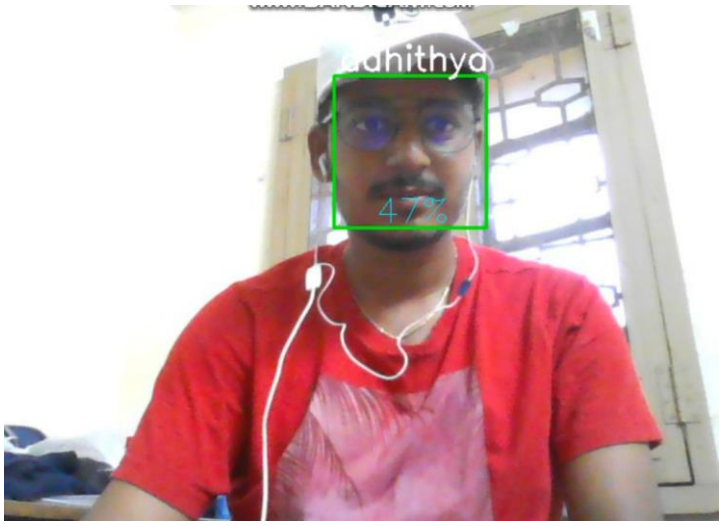


Figure 5.3.29 Face Recognition achieved even if additional materials like cap and spectacles are worn

Figure 5.3.30 Face Recognition even if face is occluded by mask and spectacles

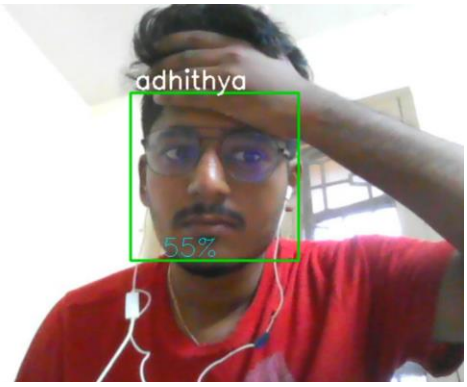


Figure 5.3.31 Face Recognition achieved even if face is partly occluded

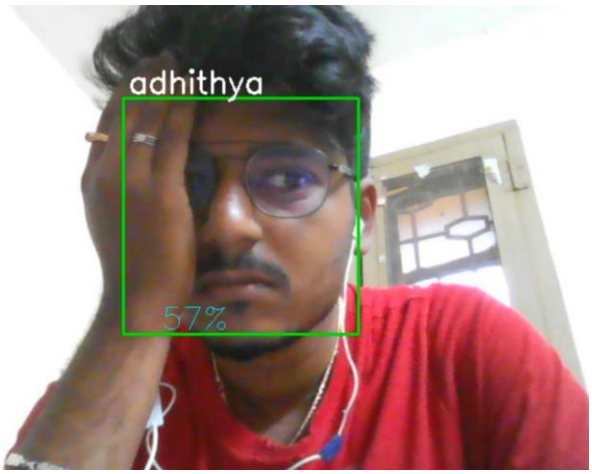


Figure 5.3.32 Face Recognition achieved even if face is partly occluded

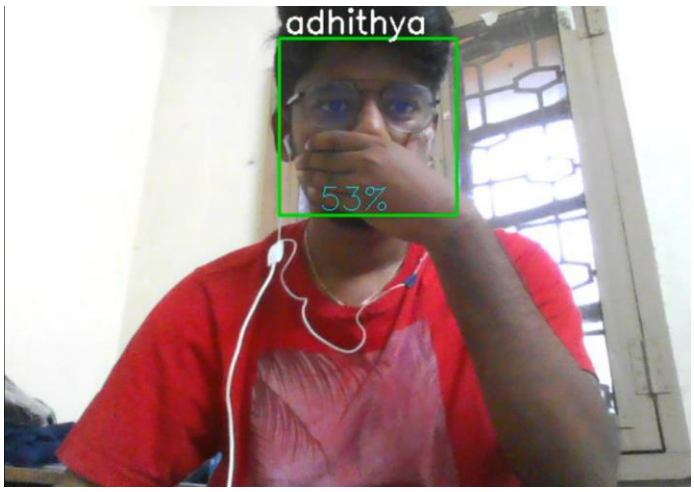


Figure 5.3.33 Face Recognition achieved even if face is partly occluded

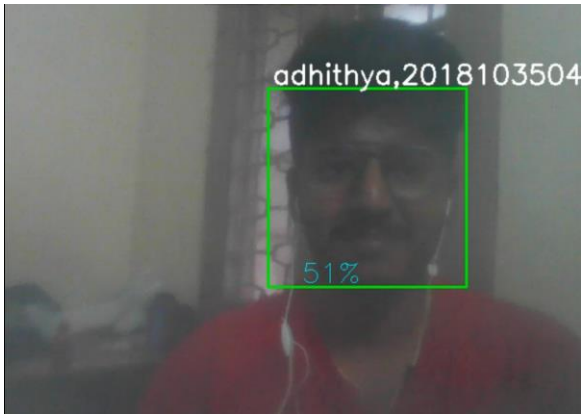


Figure 5.3.34 Face Recognition achieved even if camera is covered by a translucent material

TEST CASE COMPARISON

Table 5.3.1 shows the comparison of different test cases with regards to parameters like number of keyframes, number of faces detected, duration of the video and actual number of frames present in the input video.

Table 5.3.1 Comparison of different test cases

Techniques Experimented	**Test Cases**	**Time Duration (In minutes)**	**Actual number of frames extracted**	**Number of key frames detected**	**No. of Face Detected**
Singular Value Decomposition & Dynamic Clustering	Sample Test Case	1 min 22 secs	25	2370	3
	Test Case-1	1 min 13 secs	2	1803	3
	Test Case-2	22 sec	3	685	0
	Test Case-3	28 sec	2	750	2

Table 5.3.2 Comparison of different constraints

Constraints Handled	**No. of frames**	**No. of frames identified correctly**	**No. of frames identified in-correctly**	**Accuracy**
Occlusions in outdoor environment	8963	8781	182	**97.96**
Dark Environment	5189	5084	105	**97.97**
Light illumination changes	1565	1530	35	**97.76**
Expression & Pose variations	9486	9290	196	**97.93**
Occlusions in in-door Environment	6572	6430	142	97.83

5.4 PERFORMANCE METRICS

Performance evaluation method is the yardstick to analyse the efficiency of any face recognition system. The assessment is essential for understanding the quality of the model or the technique, for refining parameters in the iterative process of learning and for selecting the most adequate model or strategy from a given set of models or techniques. Several criteria are used to evaluate models for different tasks.

5.4.1 TRIPLET LOSS FUNCTION

Triplet loss is a loss function for machine learning algorithms where a reference input is compared to a matching input and a non-matching input. The distance from the anchor to the positive is minimised, and the distance from the anchor to the negative input is maximised as shown in Figure 5.4.1

$$\sum_{i}^{N} \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]$$

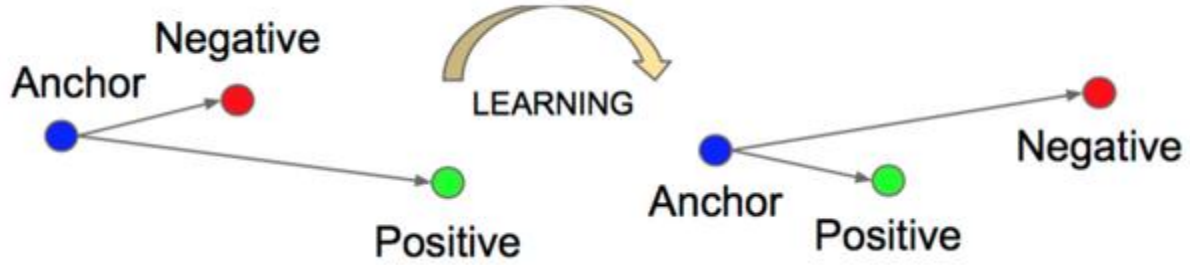


Figure 5.4.1 Learning of Anchor, Positive and Negative images

TRIPLET LOSS FUNCTION OUTPUT

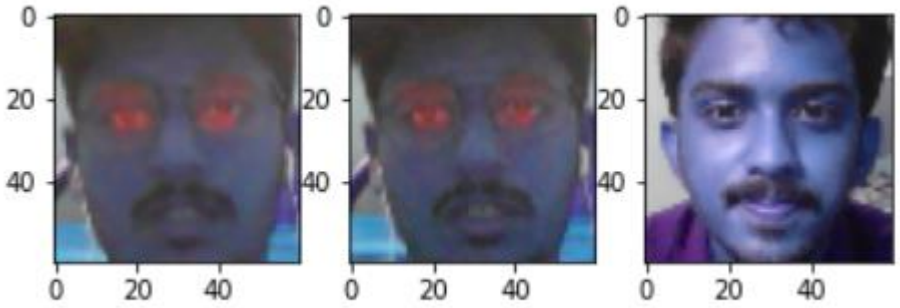


Figure 5.4.2 Sample output displaying the anchor, positive and negative images

The model is trained using the Triplet Loss function to minimise the distance between Anchor and Positive image and maximise the distance between Anchor and Negative image. Sample images of Anchor, positive and negative entities are shown in Figure 5.4.2

5.4.2 CONFUSION MATRIX

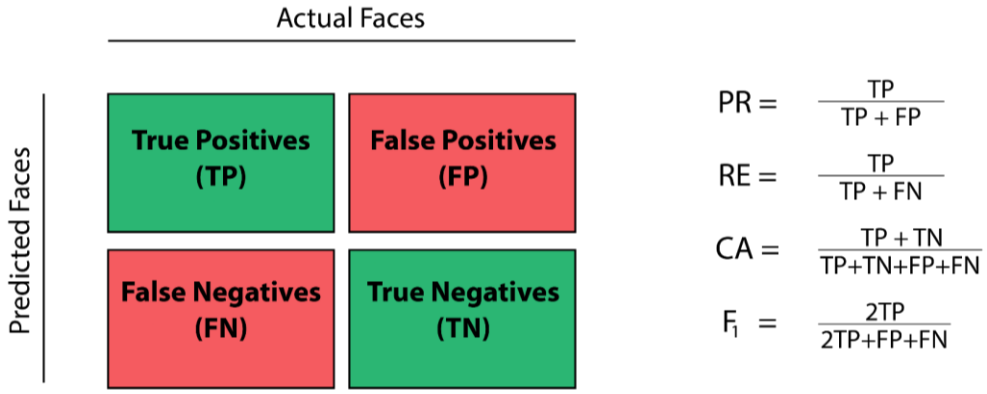


Figure 5.4.3 Representation of a Confusion matrix along with the formulae

Where, PR denotes Precision
RE denotes Recall
CA denotes Accuracy
F1 denotes F1 score.

Representation and formulae defining a confusion matrix are shown in Figure 5.4.3

CONFUSION MATRIX OUTPUT

```
Classes:
Adhithya
Sharon
Akash
Karthikeyan
Latha
Vijay
Suruthi
Aniritha
Deepika
Sriram

[[210   0   2   0   2   0   0   0   0   0]
 [  0 105   0   0   0   0   0   1   1   0]
 [  0   0 210   1   0   1   0   0   2   0]
 [  0   0   0 107   0   0   0   0   0   0]
 [  0   0   0   0 106   1   0   0   0   0]
 [  0   0   0   2   1 207   2   0   0   2]
 [  0   1   0   1   1   0 209   0   1   1]
 [  0   1   1   0   0   0   0 105   0   0]
 [  0   0   0   0   0   0   0   0 107   0]
 [  0   0   1   0   0   0   0   0   0 106]]
              precision    recall  f1-score   support

           1       1.00      0.98      0.99       214
           2       0.98      0.98      0.98       107
           3       0.98      0.98      0.98       214
           4       0.96      1.00      0.98       107
           5       0.96      0.99      0.98       107
           6       0.99      0.97      0.98       214
           7       0.99      0.98      0.98       214
           8       0.99      0.98      0.99       107
           9       0.96      1.00      0.98       107
          10       0.97      0.99      0.98       107

    accuracy                           0.98      1498
   macro avg       0.98      0.99      0.98      1498
weighted avg       0.98      0.98      0.98      1498
```

Table 5.4.1. Comparison of Precision, Recall and F1 Score of various classes in our dataset

Classes	**Precision**	**Recall**	**F1-Score**
Adhithya	1.00	0.98	0.99
Sharon	0.98	0.98	0.98
Akash	0.98	0.98	0.98
Karthikeyan	0.96	1.00	0.98

Latha	0.96	0.99	0.98
Vijay	0.99	0.97	0.98
Suruthi	0.99	0.98	0.98
Aniritha	0.99	0.98	0.99
Deepika	0.96	1.00	0.98
Sriram	0.97	0.99	0.98

5.5 COMPARATIVE ANALYSIS

The size of training data along with the accuracy achieved by different face recognition approaches like DeepID [16], DeepFace [18], VGG Face [12], FaceNet [17] on the predefined YoutTube Faces dataset are tabulated in Table 5.4.1 The proposed TBE-CNN approach achieves an accuracy of 94.96%, which is relatively higher than DeepID, DeepFace and VGG Face techniques.

Table 5.4.2. Comparison of Accuracies achieved for various methods on Youtube Faces Dataset

Method	**Number of images**	**Accuracy for YTF**
DeepID [16]	0.2M	93.20
Deep Face [18]	4.4M	91.4
VGG Face [12]	2.6M	97.30
FaceNet [17]	200M	95.10
TBE-CNN	**0.49M**	**94.96**

The base paper uses a predefined dataset called “YouTube Faces” to train the model to recognize faces present in the dataset. This project is capable of implementing both real-time face recognition and recognition of datasets. It can capture live videos and train the model to recognize faces in that video; it can also

make use of a predefined dataset and train the model to recognize faces in that dataset.

The project also uses additional Preprocessing techniques like Image Pyramiding and Histogram Equalisation which weren't used in the base paper. With the YouTube faces dataset, the base paper achieved an accuracy of nearly 95%. When the project was implemented using the YouTube faces dataset, it achieved nearly 98% accuracy.

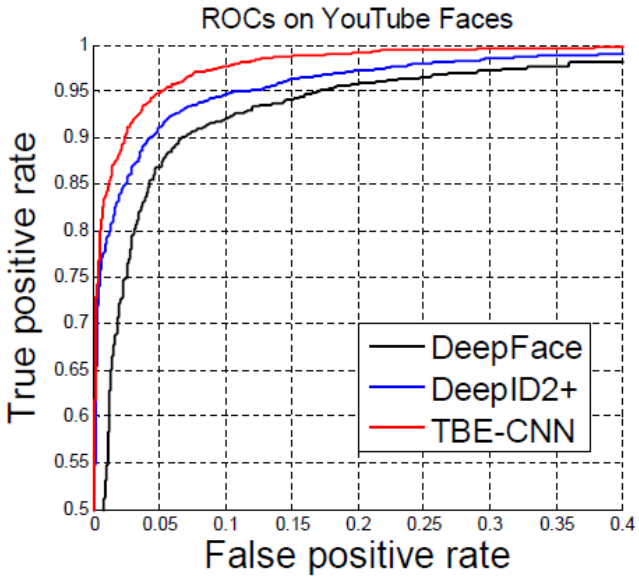


Figure 5.4.4 ROC curves of TBE-CNN and state-of-theart methods on the YouTube Faces database

An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters: True Positive Rate and False Positive Rate. Figure 5.4.4 displays the ROC Curves of DeepFace, DeepId and our proposed TBE-CNN approaches for different threshold values.

EigenFace Method Sample

```
Fitting the classifier to the training set
done in  17.571s
Best estimator found by grid search:
SVC(C=1000.0, class_weight='balanced', gamma=0.005)
Predicting people's names on the test set
done in  0.067s
                   precision    recall  f1-score   support

     Ariel Sharon       0.80      0.62      0.70        13
     Colin Powell       0.84      0.82      0.83        60
  Donald Rumsfeld       0.68      0.48      0.57        27
    George W Bush       0.78      0.96      0.86       146
Gerhard Schroeder       0.80      0.64      0.71        25
      Hugo Chavez       0.83      0.33      0.48        15
       Tony Blair       0.73      0.61      0.67        36

         accuracy                           0.79       322
        macro avg       0.78      0.64      0.69       322
     weighted avg       0.78      0.79      0.77       322

Confusion Matrix is:
[[  8   0   0   5   0   0   0]
 [  0  49   3   8   0   0   0]
 [  0   4  13   8   0   0   2]
 [  0   2   1 140   0   0   3]
 [  0   0   0   7  16   0   2]
 [  0   1   0   5   3   5   1]
 [  2   2   2   6   1   1  22]]
```

The output of the Results of EigenFace method implementation for Face Recognition has been displayed above. This technique resulted in an accuracy of 78%.

CHAPTER 6
CONCLUSION

6.1 SUMMARY

VFR is a challenging task due to severe image blur, rich pose variations, and occlusion. Compared to SIFR, VFR also has more demanding efficiency requirements. Here we address these problems via a series of contributions. To extract pose- and occlusion- robust representations efficiently, we propose a novel CNN architecture named TBE-CNN. TBE-CNN efficiently extracts representations of the holistic face image and facial components by sharing the low- and middle level layers of different CNNs. It improves on single CNN model performance with only marginal increases in time and memory costs.

6.2 FUTURE WORK

I. This system can be deployed for verification and attendance tracking at various government offices and corporations.
II. For access control verification and identification of authentic users it can also be installed in bank lockers and vaults.
III. For identification of criminals the system can be used by the police force.

CHAPTER 7
REFERENCE

1. A. RoyChowdhury, T.-Y. Lin, S. Maji, and E. Learned-Miller, “Face identification with bilinear cnns,” arXiv preprint arXiv:1506.01342, 2015.
2. C.-H. Chen, J.-C. Chen, C. D. Castillo, and R. Chellappa, “Video-based face association and identification,” in Proc. 12th IEEE Int. Conf. Autom. Face Gesture Recognit. (FG), 2017, pp. 149–156.
3. C. H. Chan, M. A. Tahir, J. Kittler, and M. Pietikainen, “Multiscale local phase quantization for robust component-based face recognition using kernel fusion of multiple descriptors,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 35, no. 5, pp. 1164–1177, 2013.
4. C. Whitelam et al., “IARPA janus benchmark-B face dataset,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW), 2017, pp. 592–600.
5. D. Chen, X. Cao, D. Wipf, F. Wen, and J. Sun, “An efficient joint formulation for bayesian face verification,” IEEE Trans. Pattern Anal. Mach. Intell., 2016.
6. H. Li, G. Hua, X. Shen, Z. Lin, and J. Brandt, “Eigen-pep for video face recognition,” in Proc. Asian. Conf. Comput. Vis., 2014.
7. J.-C. Chen, W.-A. Lin, J. Zheng, and R. Chellappa, “A real-time multitask single shot face detector,” in Proc. IEEE Int. Conf. Image Process. (ICIP), 2018, pp. 176–180.
8. J. Hu, J. Lu, and Y.-P. Tan, “Discriminative deep metric learning for face verification in the wild,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2014, pp. 1875–1882.
9. J. Lu, G. Wang, W. Deng, P. Moulin, and J. Zhou, “Multi-manifold deep metric learning for image set classification,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2015, pp. 1137–1145.
10. M. Nishiyama, A. Hadid, H. Takeshima, J. Shotton, T. Kozakaya, and O. Yamaguchi, “Facial deblur inference using subspace analysis for recognition of blurred faces,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 33, no. 4, pp. 838–845, 2011.

11. N. D. Kalka et al., “IJB–S: IARPA janus surveillance video benchmark,” in Proc. IEEE 9th Int. Conf. Biometrics Theory Appl. Syst. (BTAS), 2018, pp. 1–9.

12. O. M. Parkhi, K. Simonyan, A. Vedaldi, and A. Zisserman, “A compact and discriminative face track descriptor,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2014, pp. 1693–1700.

13. P. J. Phillips et al., “Overview of the multiple biometrics grand challenge,” in Advances in Biometrics (LNCS 5558), M. Tistarelli and M. S. Nixon, Eds. Heidelberg, Germany: Springer, 2009, pp. 705–714.

14. R. Gopalan, S. Taheri, P. Turaga, and R. Chellappa, “A blurrobust descriptor with applications to face recognition,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 34, no. 6, pp. 1220–1226, 2012.

15. T. Ahonen, E. Rahtu, V. Ojansivu, and J. Heikkila, “Recognition of blurred faces using local phase quantization,” in Int. Conf. Pattern Recognit., 2008, pp. 1–4.

16. Y. Taigman, M. Yang, M. Ranzato, and L.Wolf, “Deepface: Closing the gap to human-level performance in face verification,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2014, pp. 1701–1708.

17. Y. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2015, pp. 815–823.

18. Y. Sun, X. Wang, and X. Tang, “Deeply learned face representations are sparse, selective, and robust,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2015, pp. 2892–2900.