Unconstrained Contemporaneous Video-Based Human Profile Recognition

Under the Guidance of: **Dr. P Geetha**

Team Members:
Adhithya K - 2018103504 | Sharon Tincy BS - 2018103064 | Akash Murugesh K - 2018103510

Trunk-Branch Ensemble Convolutional Neural Networks for Video-based Face Recognition

Changxing Ding, *Student Member, IEEE*, Dacheng Tao, *Fellow, IEEE*

Abstract—Human faces in surveillance videos often suffer from severe image blur, dramatic pose variations, and occlusion. In this paper, we propose a comprehensive framework based on Convolutional Neural Networks (CNN) to overcome challenges in video-based face recognition (VFR). First, to learn blur-robust face representations, we artificially blur training data composed of clear still images to account for a shortfall in real-world video training data. Using training data composed of both still images and artificially blurred data, CNN is encouraged to learn blur-insensitive features automatically. Second, to enhance robustness of CNN features to pose variations and occlusion, we propose a Trunk-Branch Ensemble CNN model (TBE-CNN), which extracts complementary information from holistic face images and patches cropped around facial components. 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. Third, to further promote the discriminative power of the representations learnt by TBE-CNN, we propose an improved triplet loss function. Systematic experiments justify the effectiveness of the proposed techniques. Most impressively, TBE-CNN achieves state-of-the-art performance on three popular video face databases: PaSC, COX Face, and YouTube Faces. With the proposed techniques, we also obtain the first place in the BTAS 2016 Video Person Recognition Evaluation.

Index Terms—Video-based face recognition, video surveillance, blur- and pose-robust representations, convolutional neural networks.

1 Introduction

With the widespread use of video cameras for surveillance and mobile devices, an enormous quantity of video is constantly being captured. Compared to still face images, videos usually contain more information, e.g., temporal and multi-view information. The ubiquity of videos offers society far-reaching benefits in terms of security and law enforcement. It is highly desirable to build surveillance systems coupled with face recognition techniques to automatically identify subjects of interest. Unfortunately, the majority of existing face recognition literature focuses on matching in still images, and video-based face recognition (VFR) research is still in its infancy [1], [2]. In this paper, we handle the still-to-video (S2V), video-to-still (V2S), and video-to-video (V2V) matching problems, which are used in the most common VFR applications.

Compared to still image-based face recognition (SIFR), VFR is significantly more challenging. Images in standard SIFR datasets are usually captured under good conditions or even framed by professional photographers, e.g., in the Labeled Faces in the Wild (LFW) database [3]. In comparison, the image quality of video frames tends to be significantly lower and faces exhibit much richer variations (Fig. 1) because video acquisition is much less constrained. In particular, subjects in videos are usually mobile, resulting in serious motion blur, out-of-focus blur, and a large range of pose variations. Furthermore, surveillance and mobile cameras are often low-cost (and therefore low-quality) devices, which further exacerbates problems with video frame clarity [4].

Fig. 1. Video frames captured by surveillance or mobile devices suffer from severe image blur, dramatic pose variations, and occlusion. (a) Image blur caused by the motion of the subject, camera shake (for mobile devices), and out-of-focus capture. (b) Faces in videos usually exhibit occlusion and a large range of pose variations.

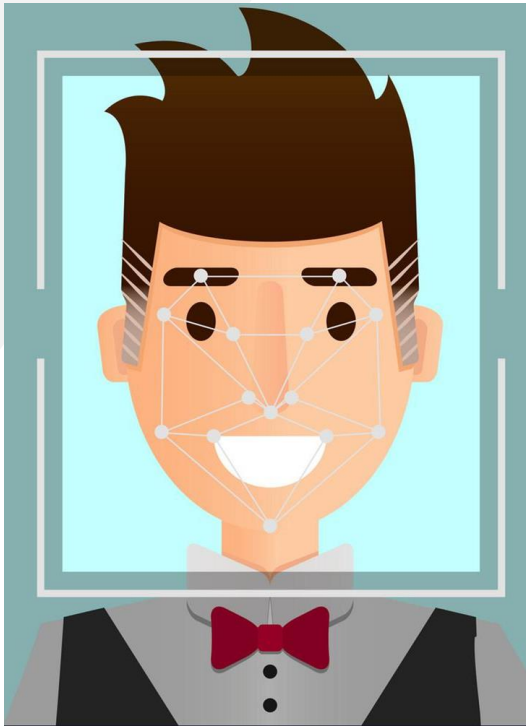
Recent advances in face recognition have tended to ignore the peculiarities of videos when extending techniques from SIFR to VFR [5], [6], [7], [8]. On the one hand, a major difficulty in VFR, such as severe image blur, is largely unsolved [9]. One important reason is that large amounts of real-world video training data are still lacking, and existing still image databases are usually blur-free. On the other hand, although pose variations and occlusion are partially solved in SIFR by ensemble modelling [6], [10], the strategy may not be directly ex-

C. Ding and D. Tao are with the Centre for Artificial Intelligence, and the Faculty of Engineering and Information Technology, University of Technology Sydney, 81 Broadway, Ultimo, NSW 2007, Australia (email: changxing.ding@student.uts.edu.au, dacheng.tao@uts.edu.au).

Introduction

- As the digital world and real world merge 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.
- Face recognition has many advantages over the other biometric traits, such as fingerprint, voice, iris, hand geometry and signature, as it is non-intrusive and easy to capture.

Face Recognition



- 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

  .

- Face recognition systems are developed in order to overcome shortcomings of image-based recognizers like sensitivity to low resolution, pose variations and partial occlusion.

- In this project, we develop a method of video-based face recognition which is fast, robust, not complex with relatively simple algorithms and techniques for real-time application.

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.
- Our project proposes an unconstrained video-based face recognition system to achieve the aforementioned trait.

Overall Objectives

1. To build a real-time capable face recognition system (FRS) for uncontrolled environments.

1. To achieve **unobtrusive recognition**, i.e. to create a system that operates in the background and does not need specific user interaction.

1. 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.

Literature Survey

S.No	Base Paper	Methodology	Merits	Demerits
1	Blur and Motion Blur Influence on Face Recognition Performance **Author**: Katarina Knezevic et al. **Publication**: NEUREL Conference **Year:** 2018	Haar Classifier, Local Binary Patterns Histogram algorithm (LBTH)	Established importance of Motion and Gaussian Blur in Face Recognition	Does not deal with occlusion.
2	Implementation of a Specified Face Recognition System based on Video **Author:** Chao Guo et al **Publication:** IEEE Journal **Year:** 2019	FFmpeg and SVM	Identifies faces even in low-quality videos	Not designed to handle unconstrained conditions.

S.No	Base Paper	Methodology	Merits	Demerits
3	A Framework for Real-Time Face Recognition **Author**: Samadhi Wickrama et al **Publication**: IEEE Journal **Year:** 2019	Deep Neural Networks	Generates FaceMaps along with Detection	No Approaches to improve robustness
4	Approaches On Partial Face Recognition: A Literature Survey **Author:** Reshma and Kannan **Publication:** ICOEI Conference **Year:** 2019	Dynamic Feature Matching (DFM)	Developed three approaches namely key-point based, region-based, CNN-based.	Does not deal with motion blur
5	Face Recognition from Video Using Deep Learning **Author:** Saibal Manna et al. **Publication:** ICCES Conference **Year:** 2020	Deep Learning	Mapping to Euclidean space to reduce computations	Does not deal with illumination problem.

Research Gaps

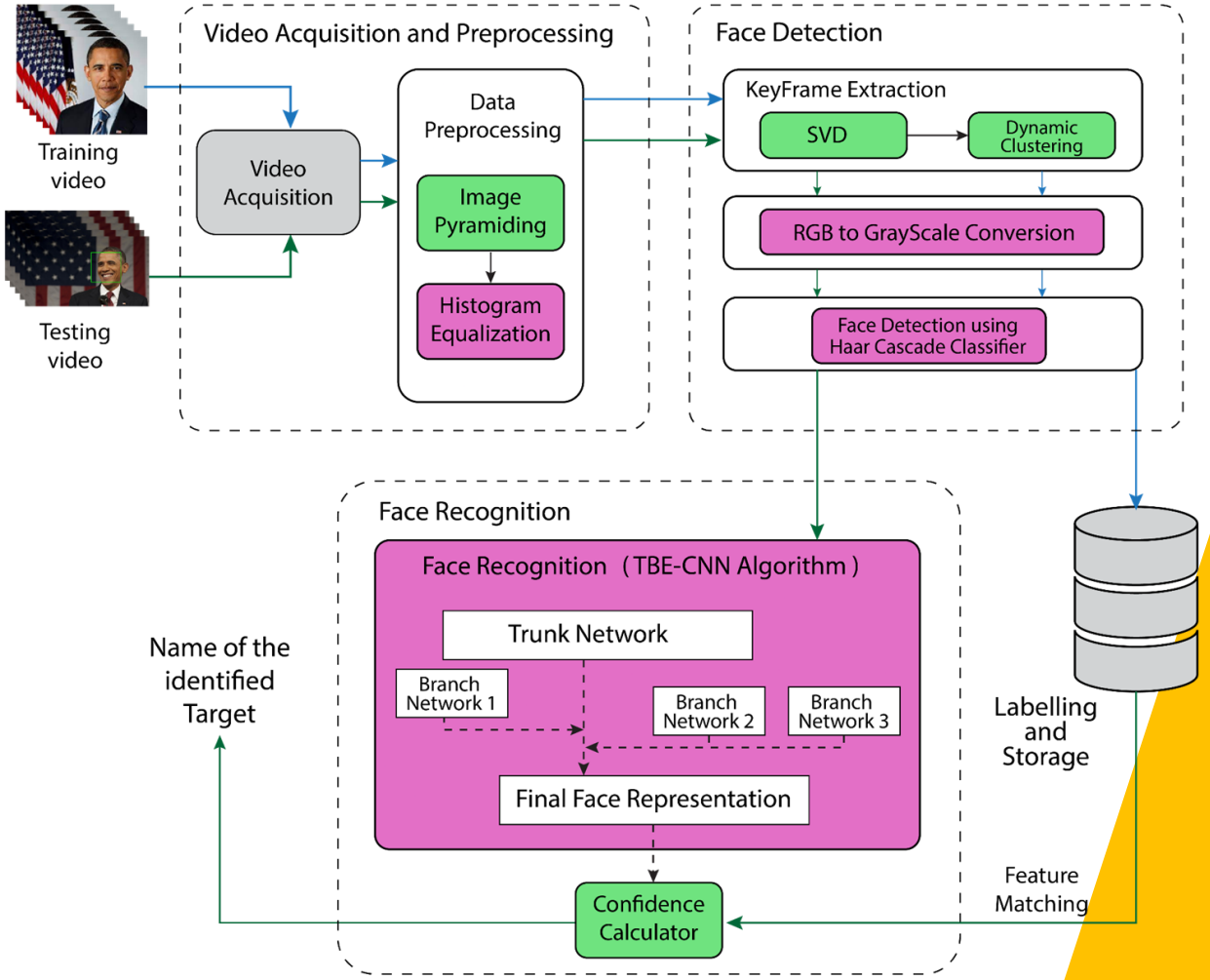
- The existing implementations don’t handle illumination variations, pose variations, and occlusions all together.
- There isn’t a comprehensive system that deals with all the aforementioned constraints seamlessly when real-time video is fed as input.
- The existing systems don’t recognize subjects who are far away in the background effectively.

📝 Proposed System

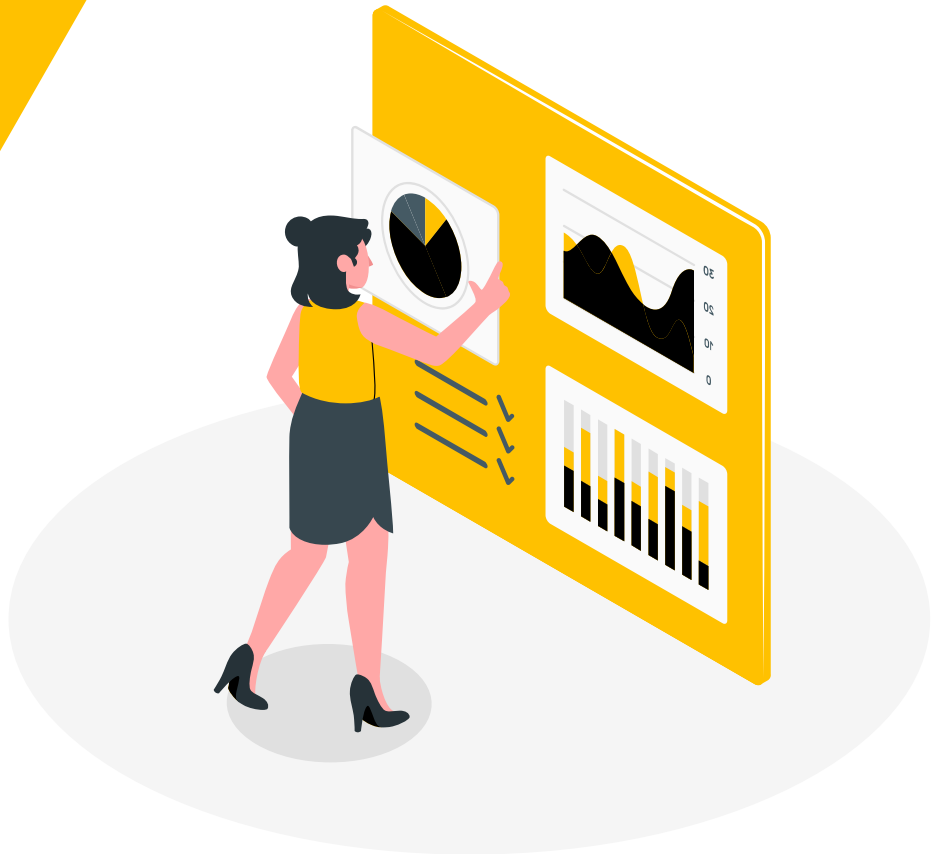
- We develop a method of video-based face recognition which is **fast, robust**, not complex and achieves greater accuracy with relatively simple and easy to comprehend algorithms and techniques for real-time application.

- We aim to build a system that effectively handles
    i) pose variations
    ii) occlusion
    iii) variation in illumination

Architecture Diagram:



Details Of Modules

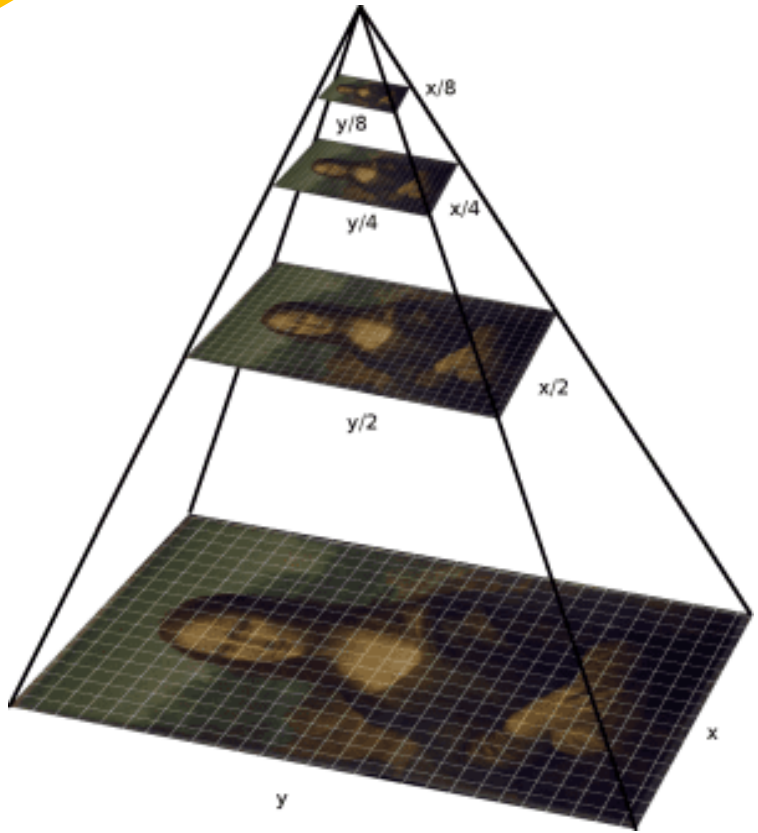


Module 1: Video Acquisition & Preprocessing

INPUT: Real-time Video
OUTPUT: Pre-processed data

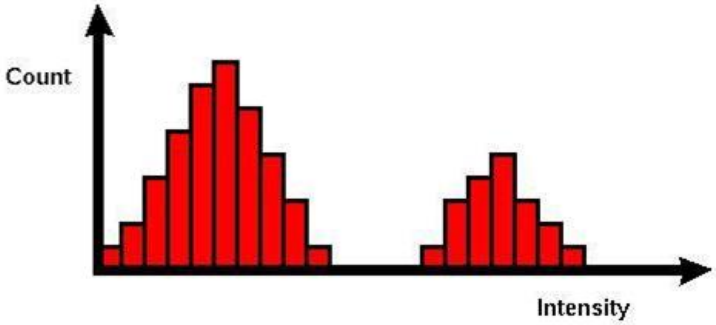
- The model acquires the video from livestream through the Video Acquisition submodule.
- To reduce the variability in the faces, the videos must be processed before they are fed into the network.
- To achieve this feat, we harness the Image Pyramiding technique and Histogram Equalization.

Image Pyramiding



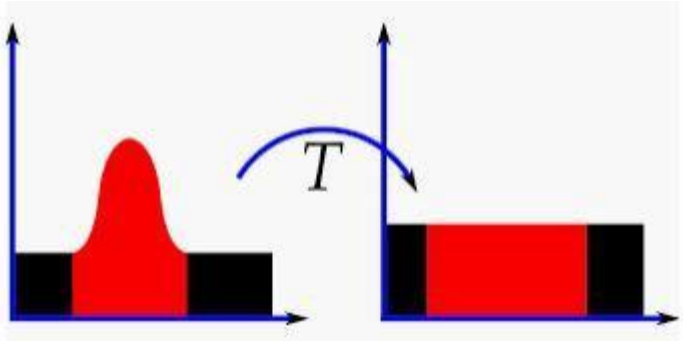
- The **pyrUp()** function increases the size to double of its original size and **pyrDown()** function decreases the size to half.
- If we keep the original image as a base image and go on applying pyrDown function on it and keep the images in a vertical stack, it will look like a **pyramid**
- The same is true for upscaling the original image by pyrUp function.

Histogram Equalization



- 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.
- As we can see in the image above, the X-axis represents the pixel intensity levels of the image. The Y-axis of the histogram indicates the frequency or the number of pixels that have specific intensity values.

Histogram Equalization Cont..



- 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.

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.

Module 2: Face Detection

INPUT: Pre-processed data
OUTPUT: Bounding boxes enclosing faces

- The pre-processed frames procured from Module 1 are fed into the Face Detection framework, to enclose the faces in them with bounding boxes.
- 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, we first extract key frames and detect faces only in those frames

Key Frames Extraction

- Key frame extraction is a powerful tool that selects a set of summary key frames to represent video sequences.

- We use Color Histogram, **SVD** and **Dynamic Clustering** Method to obtain Key-Frames from a video.

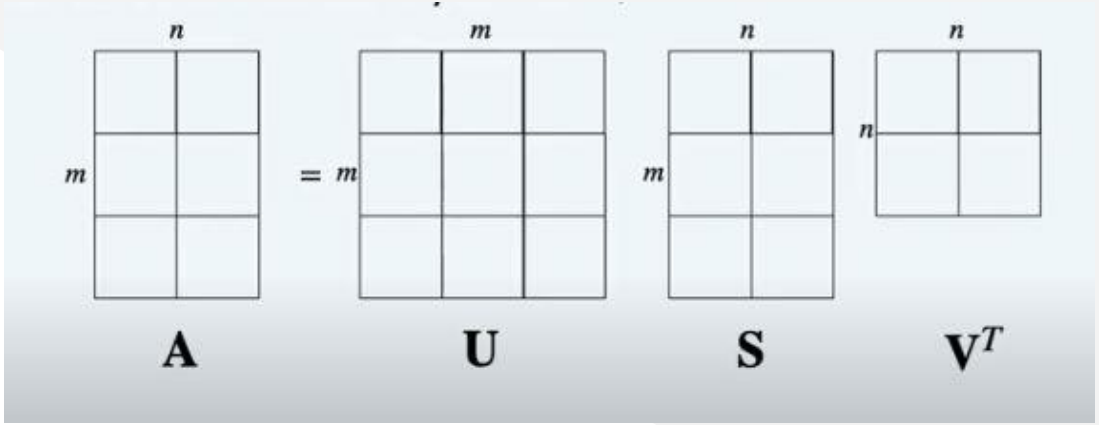
Singular Value Decomposition

- 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.

- We then perform **dimensionality reduction** on the matrix using **SVD**, thus reducing the dimensions of the feature-frame matrix.

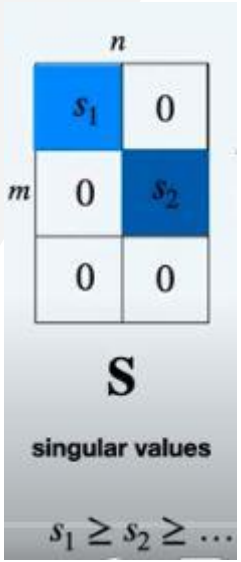
Algorithm

- The Singular Value Decomposition (SVD) of a matrix is a factorization of that matrix into three matrices.

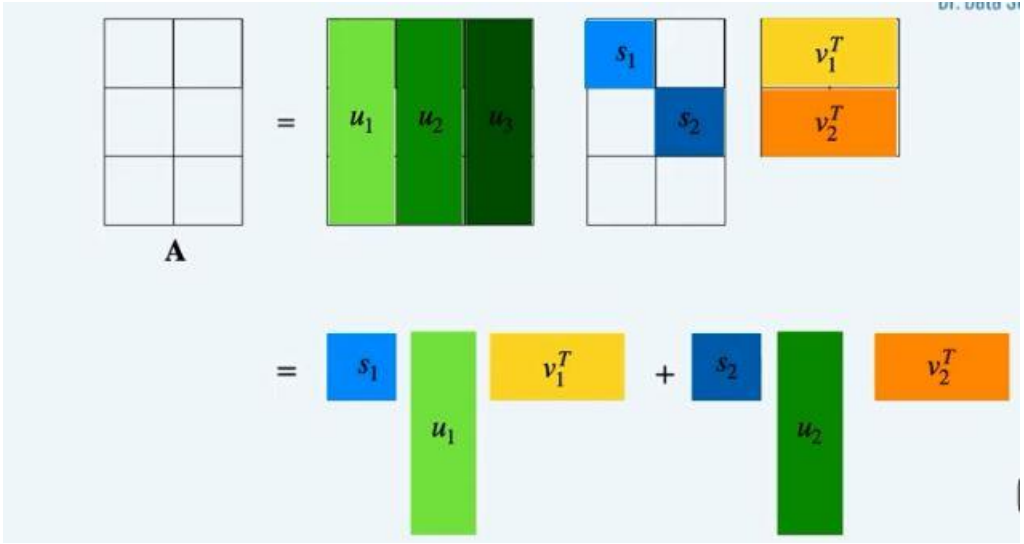


- Here, A denotes the matrix to be decomposed. SVD states that any matrix A can be decomposed into or written as a product of 3 matrices - **U**, **S** (Sigma) and $\mathbf{V^T}$

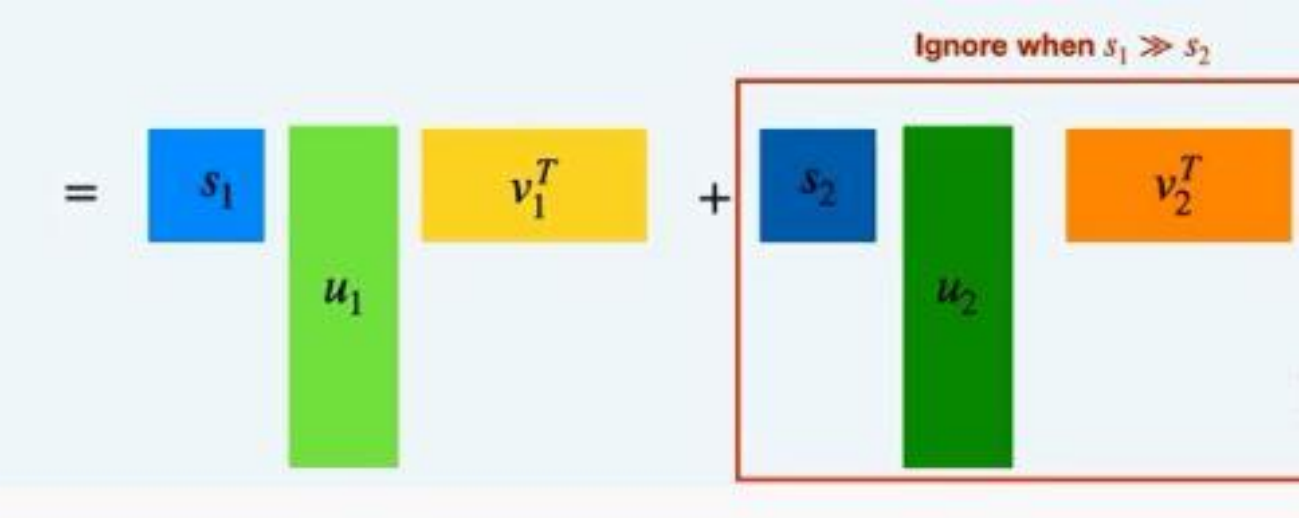
- The columns of matrix U are called "**Left Singular Vectors.**" Matrix U contains the orthonormal eigenvectors of AA^T. The columns of V (or the rows $\mathbf{V^T}$) are called "**Right Singular vectors"** Matrix $\mathbf{V^T}$ contains orthonormal eigenvectors of A^TA.



- 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 below:



- If s2<<s1, the second term in the addition can be ignored, as the first term is much more important than the second term. 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).



Dynamic Clustering

- Then, we create clusters of consecutive frames 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 –

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

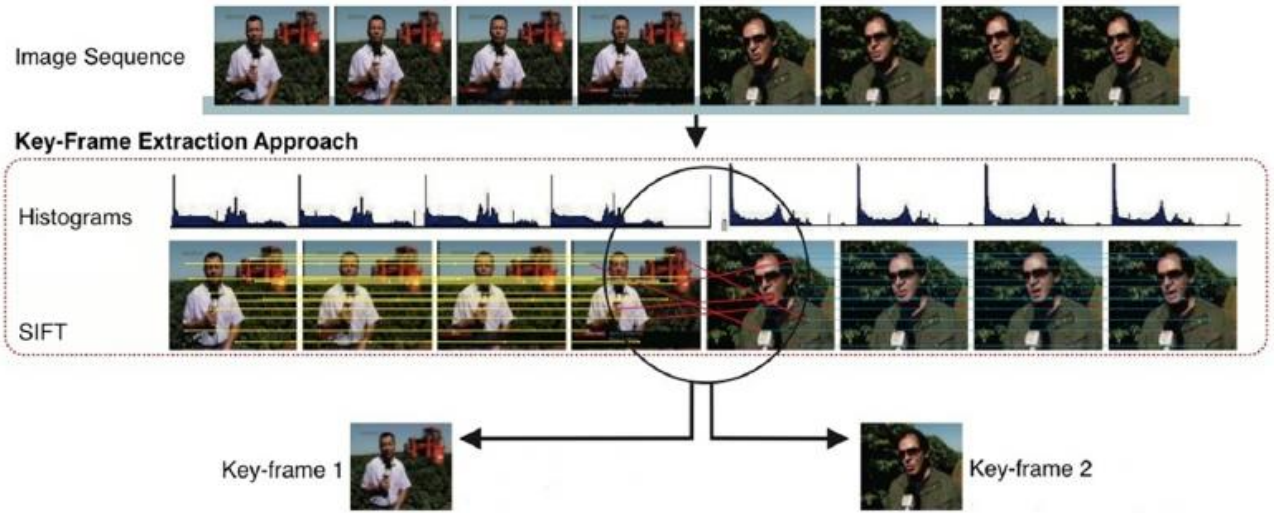
where,

- **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’.

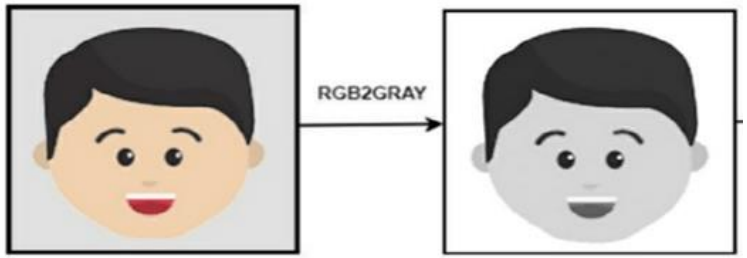
- If θ = 0°, the ‘x’ and ‘y’ vectors overlap, thus proving they are similar.
- If θ = 90°, the ‘x’ and ‘y’ vectors are dissimilar.

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.



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.
- The reason for differentiating such images from any other sort of color image is that less information needs to be provided for each pixel.



Haar Cascade Classifier:

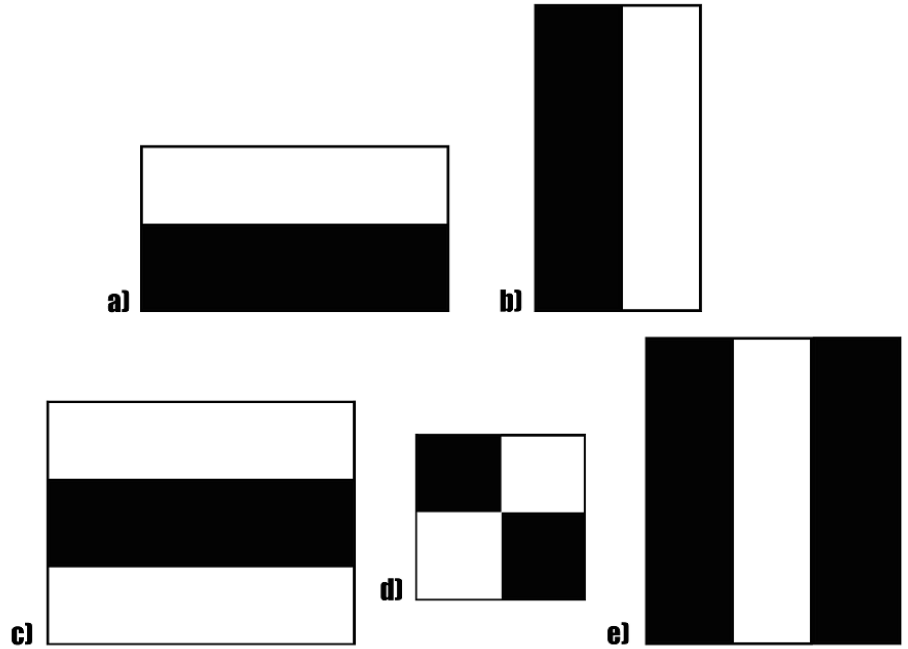
- Now, we use the Haar Cascades Classifier (or) Viola Jones Face Detection Technique to detect faces in the key frames.

- 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.

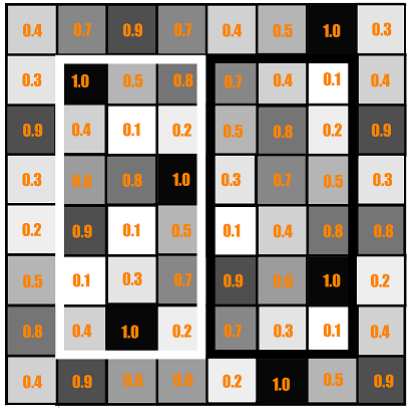
- The algorithm is given a lot of positive images consisting of faces, and a lot of negative images not consisting of any face to train on them.

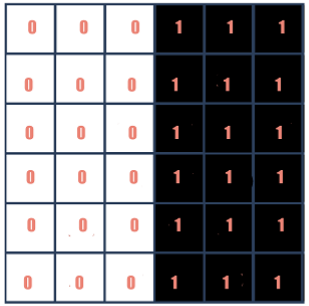
Haar Cascade Classifier Contd..

The first contribution to the research was the introduction of the **haar features** shown below. 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.



A sample calculation of Haar value from a rectangular image section has been shown here.





=

SUM OF THE DARK PIXELS/NUMBER OF DARK PIXELS - SUM OF THE LIGHT PIXELS/NUMBER OF THE LIGHT PIXELS

(0.7 + 0.4 + 0.1 + 0.5 + 0.8 + 0.2 + 0.3 + 0.7 + 0.5 + 0.1 + 0.4 + 0.8 + 0.9 + 0.6 + 1.0 + 0.7 + 0.3 + 0.1)/18

-

(1.0 + 0.5 + 0.8 + 0.4 + 0.1 + 0.2 + 0.6 + 0.8 + 1.0 + 0.9 + 0.1 + 0.5 + 0.1 + 0.3 + 0.7 +0.4 + 1.0 + 0.2)/18

0.51-0.53 = -0.02

- The rectangle on the left is a sample representation of an image with pixel values 0.0 to 1.0. The rectangle at the center 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.

Attentional Cascade

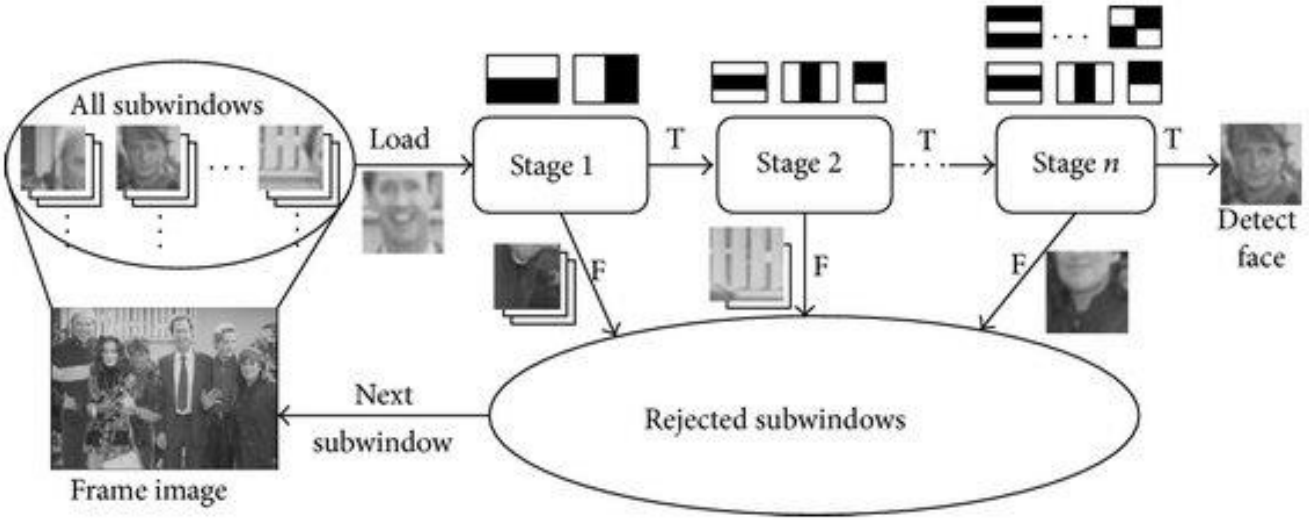
- The idea behind this is, not all the features need to run on each and every window.
- If a feature fails on a particular window, then we can say that the facial features are not present there.
- Hence, we can move to the next window where there can be facial features present.
- This way a lot of processing time will be saved, as the irrelevant windows will not be processed.
- 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.

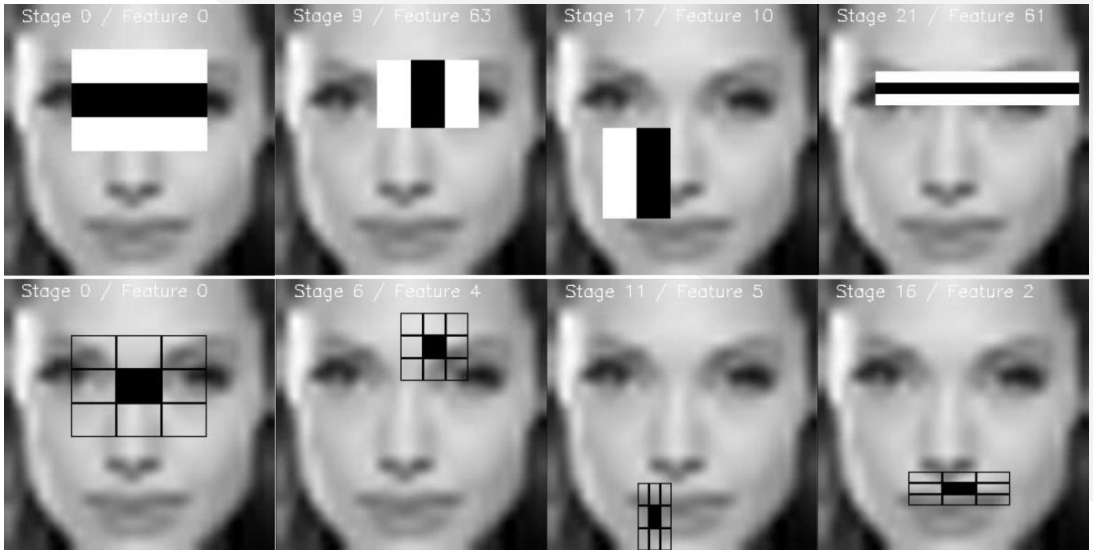
- The stages in the beginning contain simpler features, in comparison to the features in a later stage which are complex, complex enough to find the nitty gritty details on the face.

- 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.



- 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.



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.

Algorithm:

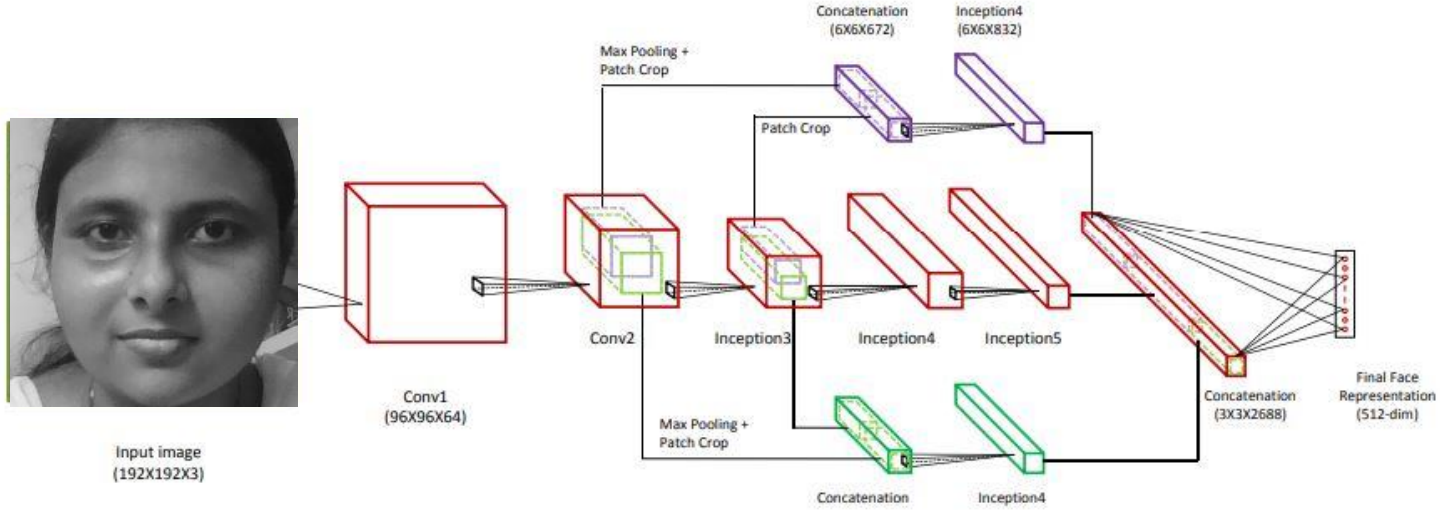
```
for i <- 1 to num of scales in pyramid of images do
        Downsample image to create image_i
        Compute integral image, image_ii
        for j <- 1 to num of shift steps of sub-window do
                for k <- 1 to num of stages in cascade classifier do
                        for l <- 1 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
```

Module 3: Face Recognition

INPUT: Detected faces
OUTPUT: Identification of target in video

- The detected faces obtained from Module 2 are fed into the Face Recognition module.
- To enhance robustness of CNN features to pose variations and occlusion, we propose a Trunk-Branch Ensemble CNN model (TBE-CNN).

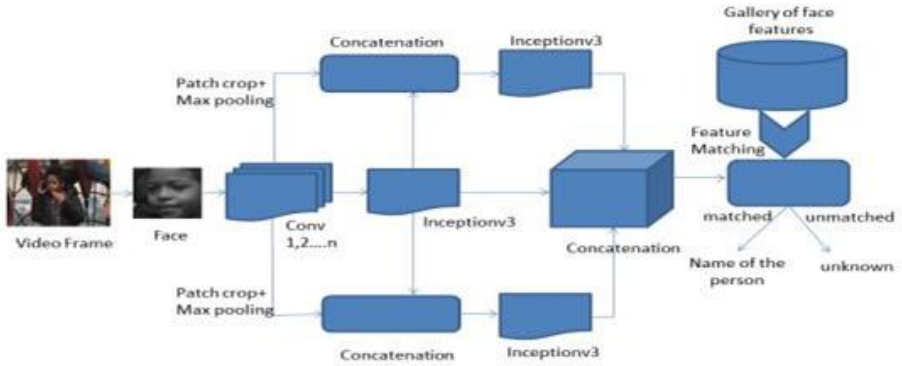
- The architecture consists of one trunk network and several branch networks.
- The trunk network will study global features, whereas the branch networks will study the local features in image input.
- The output feature maps of the trunk network and branch networks are fused by concatenation to form an over-complete face representation.
- The feature vector is utilized as the final face representation of one video frame.



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,

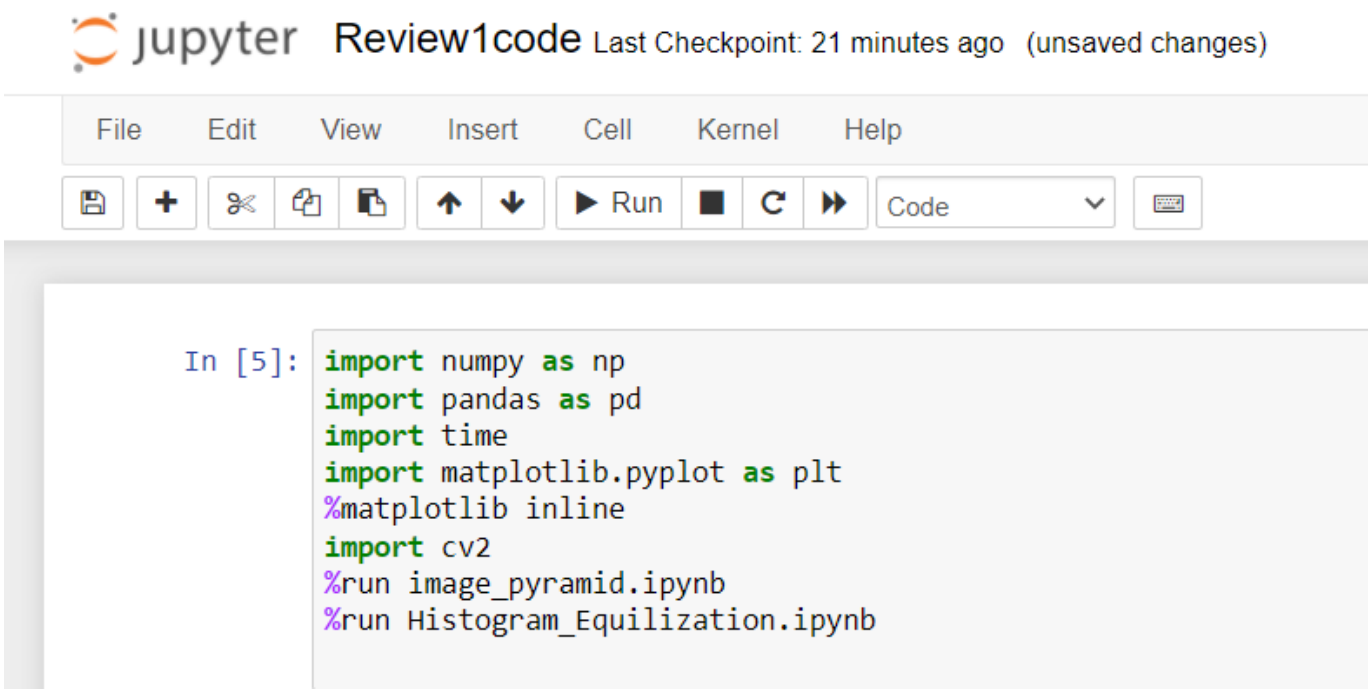
Dist = sqrt(sum((I1 - I2) .^ 2))
Where, I1 is the test face feature map, and I2 is the face feature map from the database.
Step5: Display result
.



Model Architecture of TBE-CNN and Feature Matching

IMPLEMENTATION DETAILS (80%):

Module 1: Video Acquisition and Pre-processing

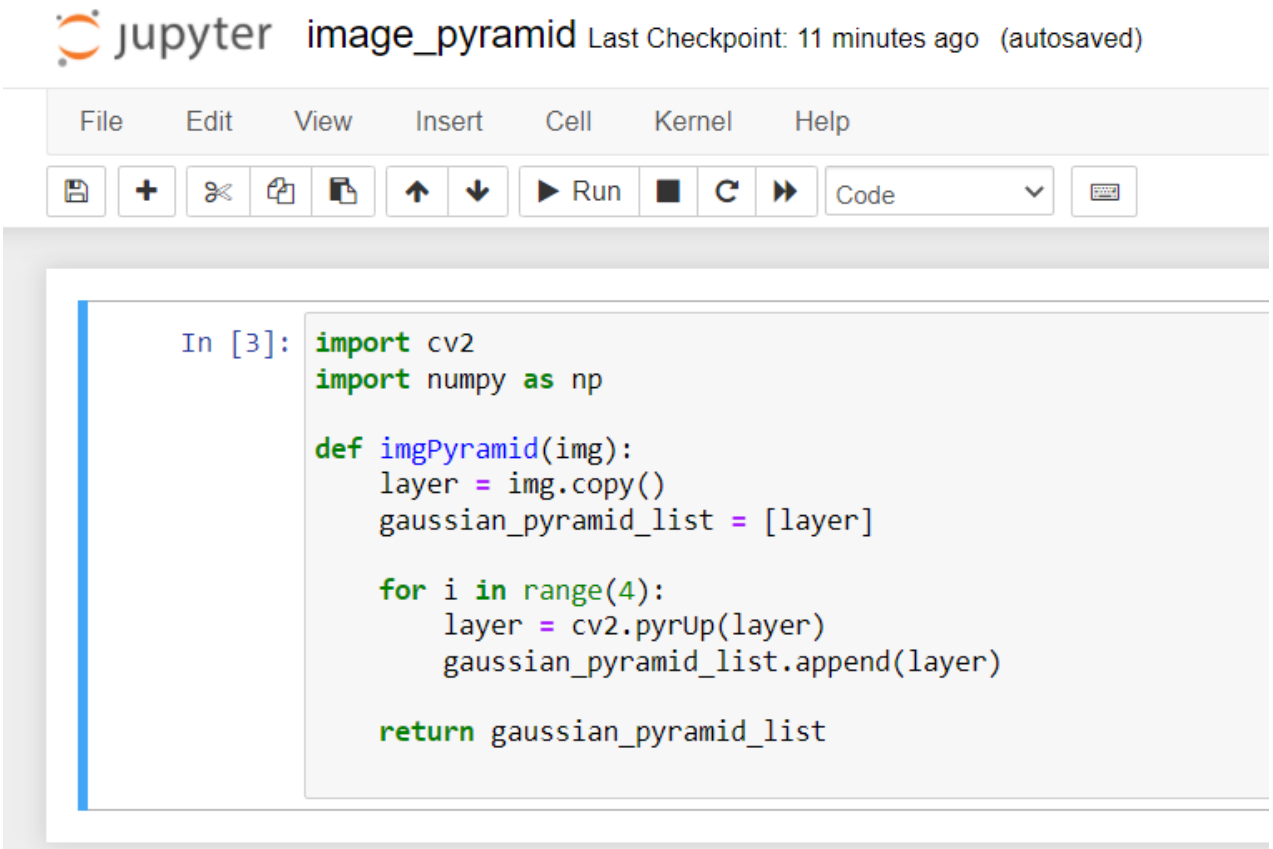


In [3]:
```
cap = cv2.VideoCapture(0)

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

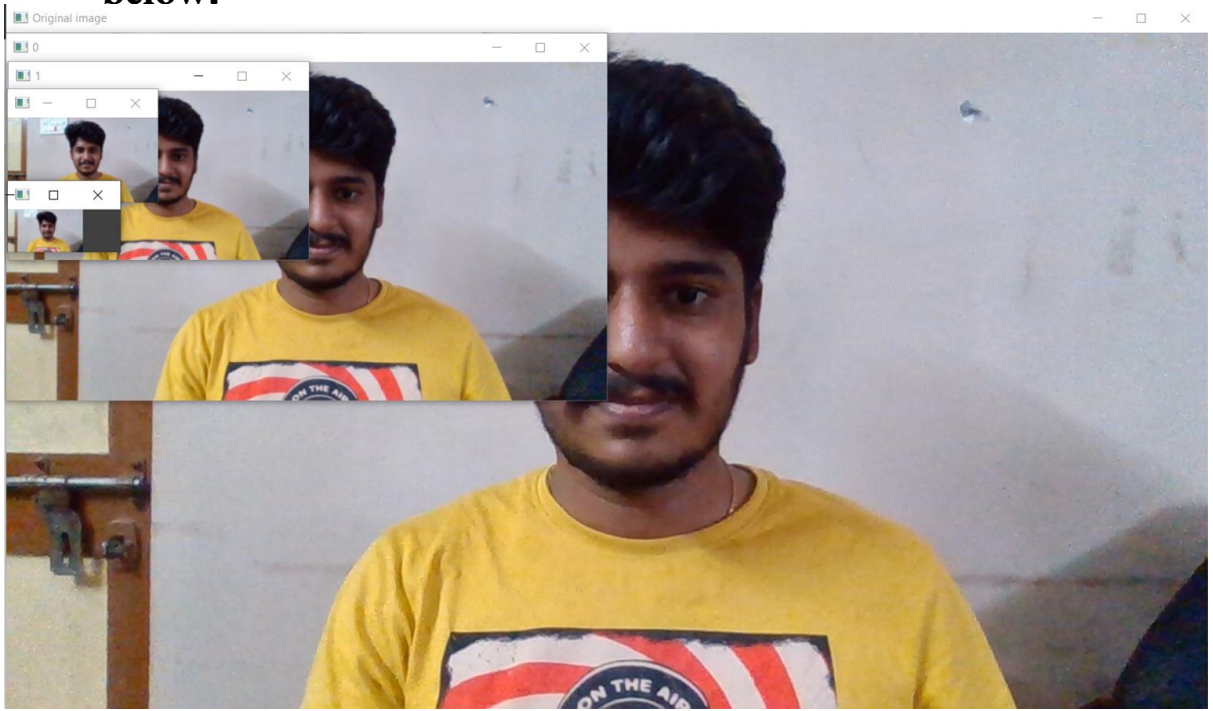
```
D=dict()    #to store the original frame (array)
count=0     #counting the number of frames
start_time = time.time()
```

```
while (True):

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

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

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

IMAGE PYRAMIDING:



A Sample Output for Image Pyramiding is shown below:



After Image Pyramiding, we move to the next step called Histogram Equalization.

HISTOGRAM EQUALIZATION

```
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
```

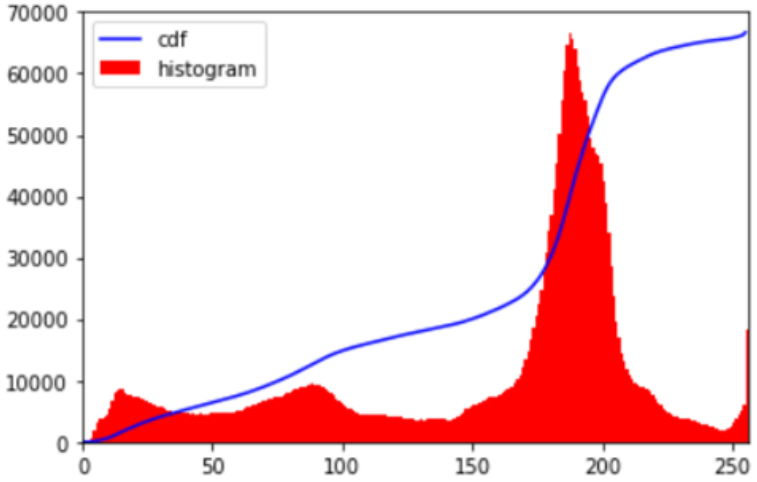
In [5]:

```
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()
```

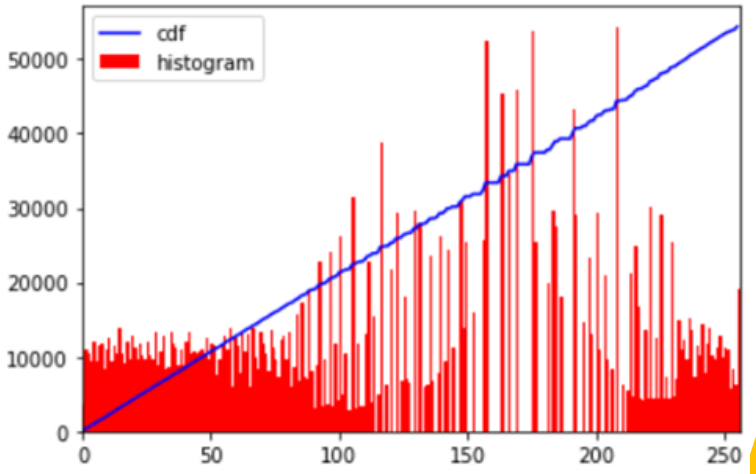
```
    R, G, B = cv.split(img)

    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
```

HISTOGRAM REPRESENTATION OF THE FRAME

Before Histogram Equalization

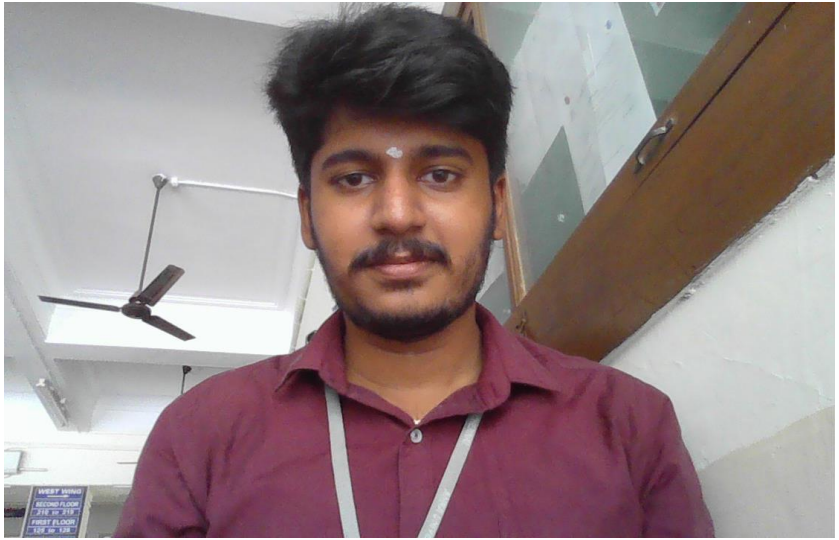


After Histogram Equalization



Sample Output for Histogram Equalization:

Original Image:



Histogram Equalized Image:

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
```

```
    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
```

```
    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)
```

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

Singular Value Decomposition

In [3]:
```
from scipy.sparse import csc_matrix
from scipy.sparse.linalg import svds, eigs
A = csc_matrix(final_arr, dtype=float)

#top 63 singular values from 76082 to 508
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)
```

We choose only the top 63 singular values 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, we segregate frames into clusters depending on their similarity. To achieve this, we employ “Cosine Similarity Check.”

```
    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 populo
#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)) #so total 25 shots with 46 (71-25) cuts
```

```
25
```

We assume 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 we can take the last element of these shots to summarise that particular shot.

```
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)) #so total 25 shots with 46 (71-25) cuts
```

```
25
```

```
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, we label each cluster 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:

EXTRACTED KEYFRAMES


frame71


frame133


frame227


frame369


frame423


frame507


frame541


frame668


frame700


frame734


frame910


frame1026


frame1032


frame1071


frame1151


frame1283


frame1353


frame1404


frame1548


frame1664


frame1732


frame2080


frame2161


frame2282


frame2370

RGB to Grayscale 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

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");
```

```
# 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)
```

```
        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
```

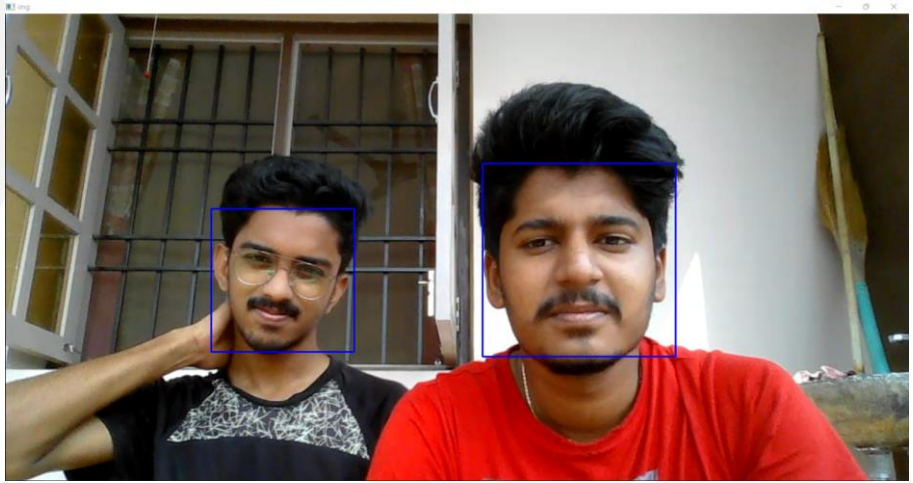
In [21]:

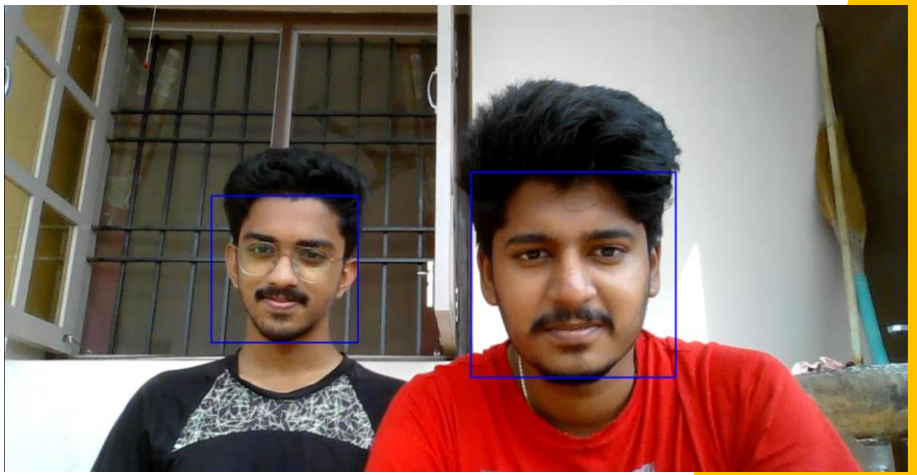
```
import cv2
import numpy as np
import os

#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()
```

A sample output for Face Detection using Haar Cascade Classifier is shown below:





Trunk Branch Ensemble CNN

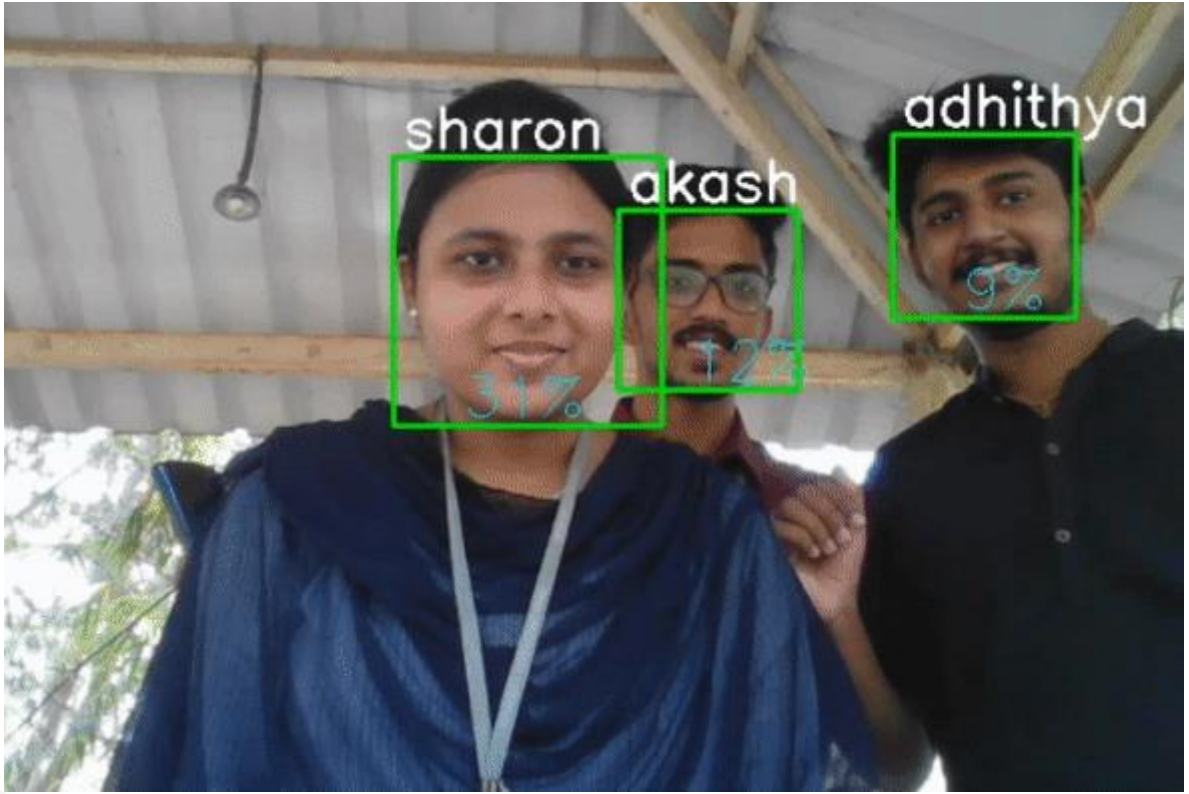
In [2]:
```
from tensorflow.python.keras.layers import Conv2D, MaxPooling2D, Concatenate, Input
import os
import inception

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='c
        conv1 = MaxPooling2D(data_format='channels_last', padding='same', strides=2, pool_size=(2, 2), name='conv1')(conv1_convol

        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_convol

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

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

    inception4b_activation = inception.with_dimension_reduction(inception3, 128, False, name='inception4b_activation_branch_1
    inception4e_activation = inception.with_dimension_reduction(inception4b_activation, 132, False, name='inception4e_activat
    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_1
    inception5b_2 = inception.with_dimension_reduction(inception5a_activation, 256, True, name='inception5b_2_branch_1')
```

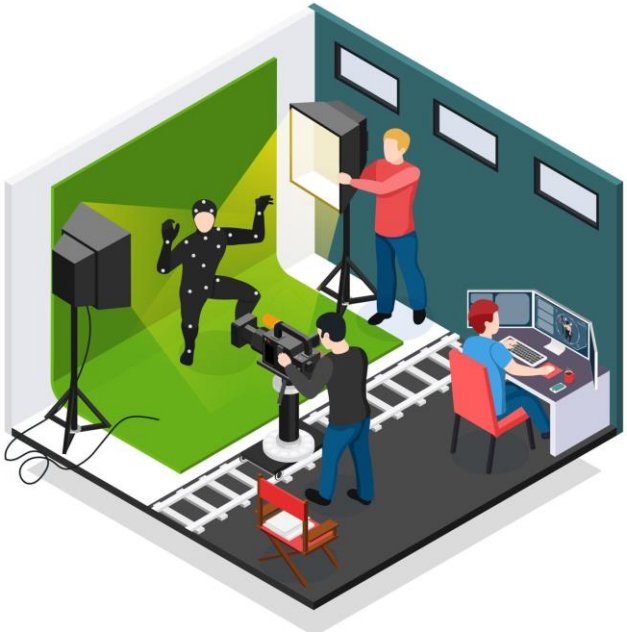
```
######################################################## Branch 2 ########################################################

inception4c_activation = inception.with_dimension_reduction(inception3, 128, False, name='inception4c_activation_branch_2
inception4e_activation = inception.with_dimension_reduction(inception4c_activation, 132, False, name='inception4e_activat
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_2
inception5b_3 = inception.with_dimension_reduction(inception5a_activation, 256, True, name='inception5b_3_branch_2')

############################################## Branch 3 --- addition ##############################################

inception4d_activation = inception.with_dimension_reduction(inception3, 128, False, name='inception4d_activation_branch_3
inception4e_activation = inception.with_dimension_reduction(inception4d_activation, 132, False, name='inception4e_activat
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_3
inception5b_4 = inception.with_dimension_reduction(inception5a_activation, 256, True, name='inception5b_4_branch_3')

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

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

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

A sample output where TBE-CNN stores the face images of the subjects along with the cropped patches around facial components is shown in next slide:

RESULTS



DATA SET DESCRIPTION



- Since our project revolves around recognizing individuals from a livestream, we have to compose our **own dataset**.
- 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.

INNOVATION

1. The base paper uses **PaSC, COX Face, and YouTube Faces datasets**, which are predefined ones, but we devise our **own dataset** where we capture videos and chip them into individual frames which are later labelled and used for recognition purposes.
2. To achieve image enhancement while capturing videos, we use **Histogram Equalization** which would in turn help achieve seamless Face Recognition by improving the contrast of frames.
3. There aren't ample resources and documentation available for **TBE-CNN Implementation**. But, making use of the base paper, we are trying to build the TBE-CNN model for facial recognition.
4. Generally most face recognition models do not have any exclusive features to identify and recognize small faces in videos. But we make use of **Image Pyramiding** technique to resize the frames to identify even the small faces present in the video.

CHALLENGES

1. For video-based face recognition, test data are from videos where each video contains tens of thousands of frames and each frame may have several faces. This makes the **scalability** of video-based face recognition a challenging problem. In order to make the face recognition system to be operationally effective, each component of the system should be **fast**, especially **face detection**, which is often the **bottleneck** in recognition.

1. Since faces are mostly from **unconstrained videos**, they have significant variations in pose, expression, illumination, blur, occlusion and video quality. Thus, any face representation we design **must be robust** to these variations and to **errors in face detection** and **association** steps.

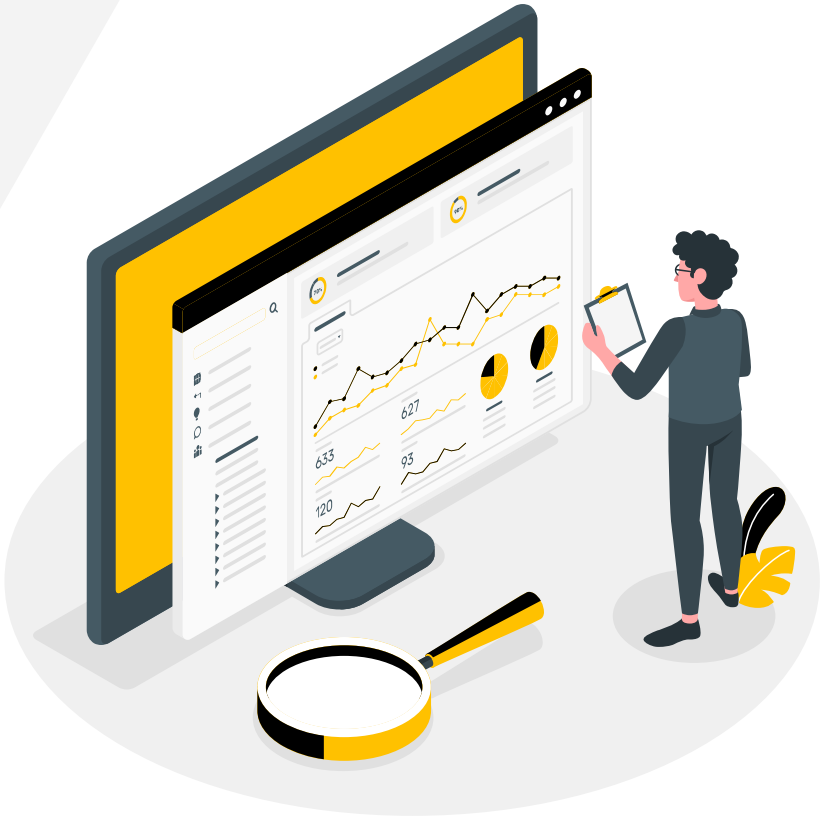
APPLICATIONS

1. **Automated surveillance**, where the objective is to recognise and track people.
2. Monitoring closed circuit television (**CCTV**), the facial recognition capability can be embedded into existing CCTV networks, to look for **lost children** or other missing persons or tracking known or **suspected criminals**.
3. **Airplane-boarding** gate, the face recognition may be used in places of random checks merely to screen passengers for further investigation.

APPLICATIONS - CONTD.

GOVERNMENT USE	COMMERCIAL USE
• Law Enforcement • Counter Terrorism • Immigration • Legislature	• Day Care • Gaming Industry • Residential Security • E-Commerce • Voter Verification • Banking

PERFORMANCE METRICS:



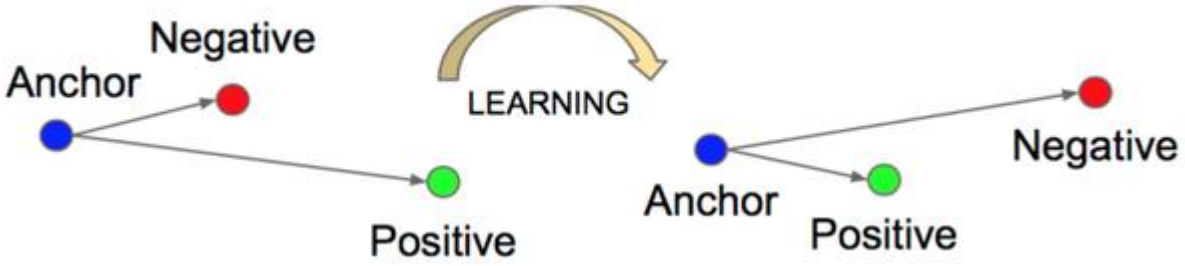
- Performance evaluation method is the **yardstick** to analyse the efficiency of any face recognition system.
- The assessment is essential for understanding the **quality of the model** or the technique, for refining parameters in the iterative process of learning and for selecting the most adequate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

TRIPLET LOSS FUNCTION

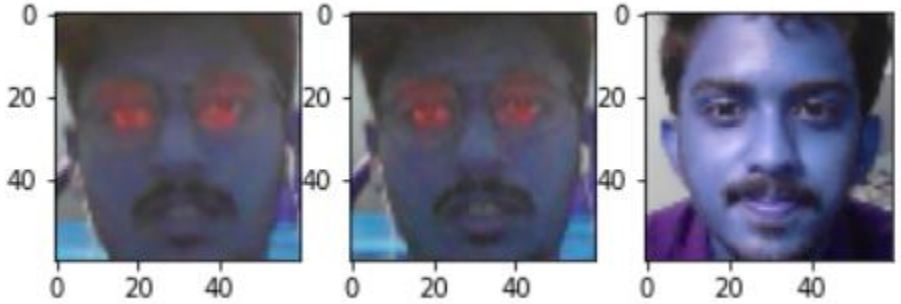
- Triplet loss is a loss function for machine learning algorithms where a reference input is compared to a matching input and a non-matching input.

- The distance from the anchor to the positive is minimized, and the distance from the anchor to the negative input is maximized.

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



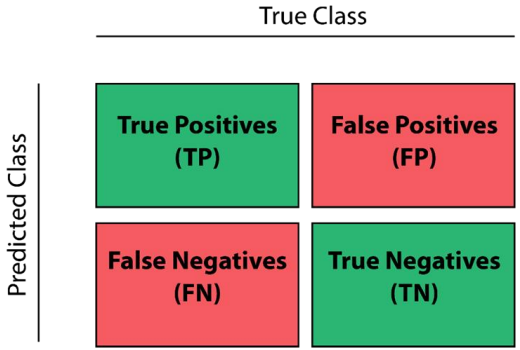
TRIPLET LOSS FUNCTION OUTPUT:



ANCHOR POSITIVE NEGATIVE

We train our model using Triplet Loss function to minimize the distance between Anchor and Positive image and maximize the distance between Anchor and Negative image.

CONFUSION MATRIX:



$$PR = \frac{TP}{TP + FP}$$

$$RE = \frac{TP}{TP + FN}$$

$$CA = \frac{TP + TN}{TP+TN+FP+FN}$$

$$F_1 = \frac{2TP}{2TP+FP+FN}$$

Where, PR denotes Precision

RE denotes Recall

CA denotes Accuracy

F1 denotes F1 score.

CONFUSION MATRIX OUTPUT:

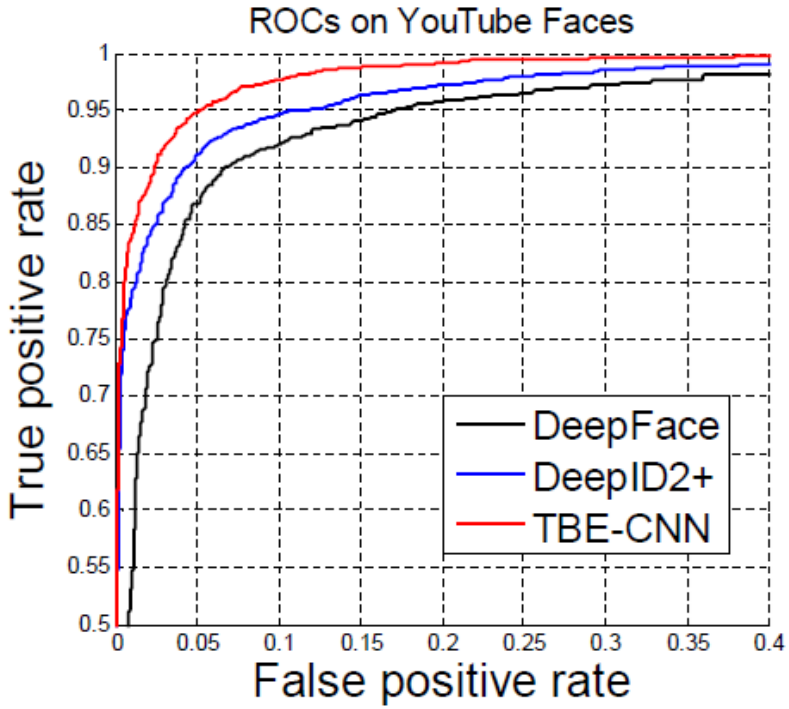
```
Classes:
Adhithya
Sharon
Akash
Karthikeyan
Latha
Vijay
Suruthi
Aniritha
Deepika
Sriram

[[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
```

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

COMPARATIVE ANALYSIS

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**

ROC curves of TBE-CNN and state-of-the art methods on the YouTube Faces database



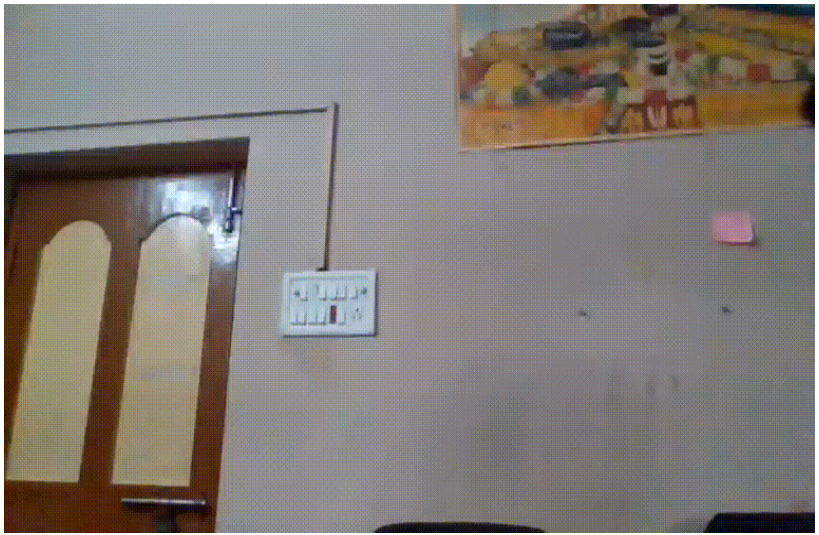
EIGEN FACE METHOD SAMPLE

```
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]]
```

We have displayed the output of the Results of EigenFace method implementation for Face Recognition here. This technique resulted in an accuracy of 78%.

TEST CASE for KEY FRAME EXTRACTION

INPUT:

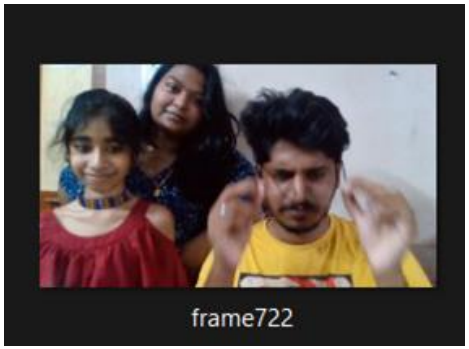


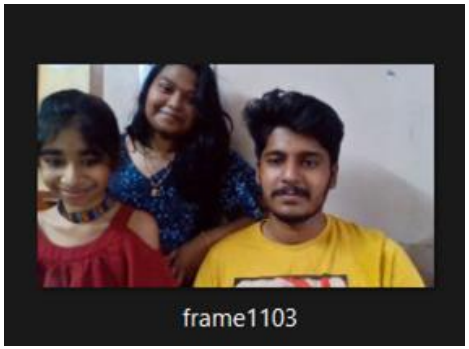
OUTPUT:

TEST CASE - 1 (FACE DETECTION)

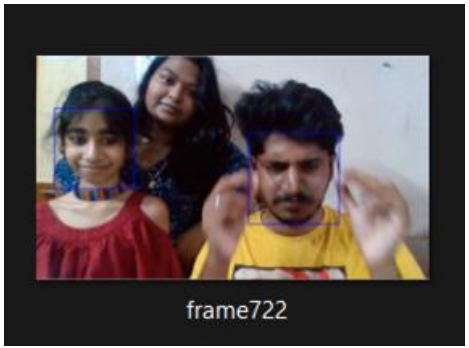
INPUT:

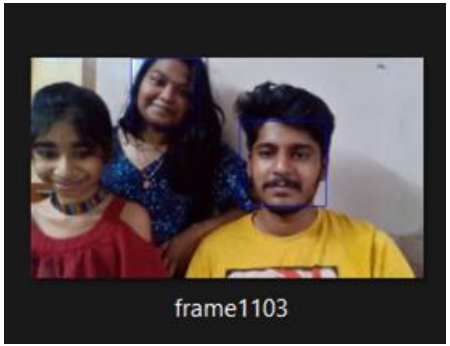
OUTPUT for Module 1:





OUTPUT for Module 2:

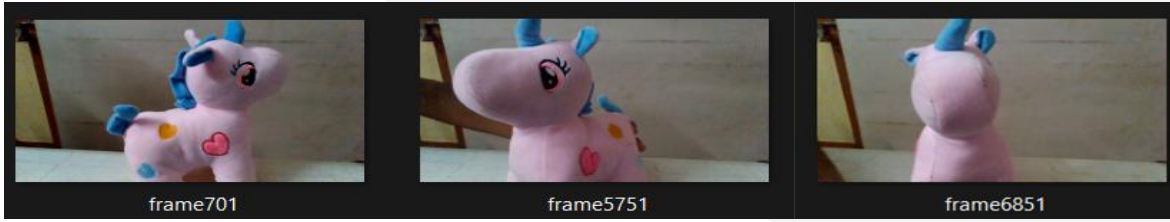




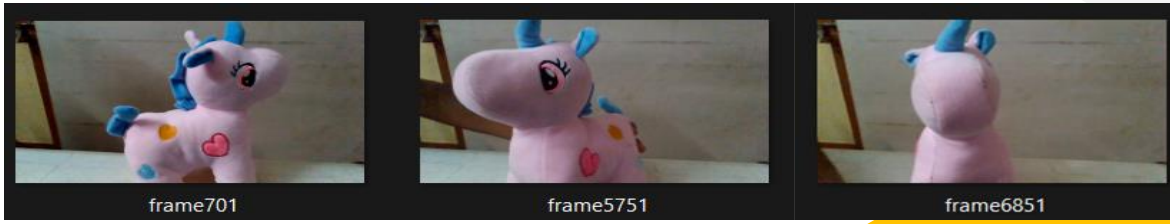
TEST CASE -2

INPUT:

OUTPUT for Module 1:

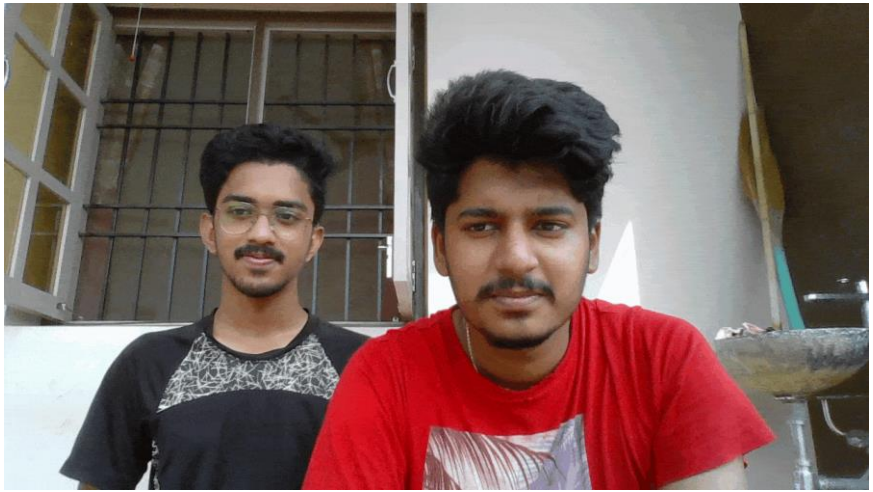


OUTPUT for Module 2:

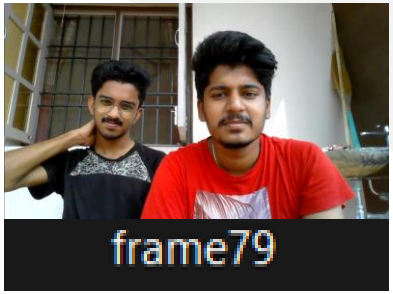


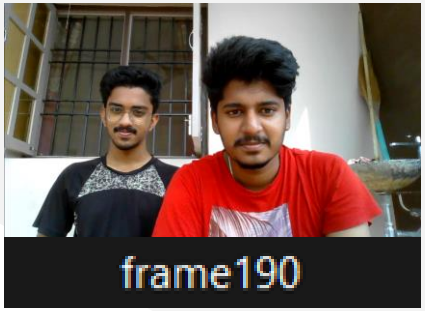
TEST CASE -3

INPUT:

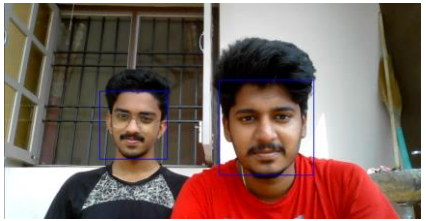


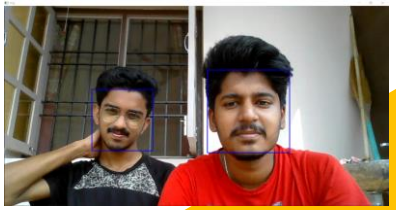
OUTPUT of Module 1:





OUTPUT of Module 2:





TEST CASE COMPARISON

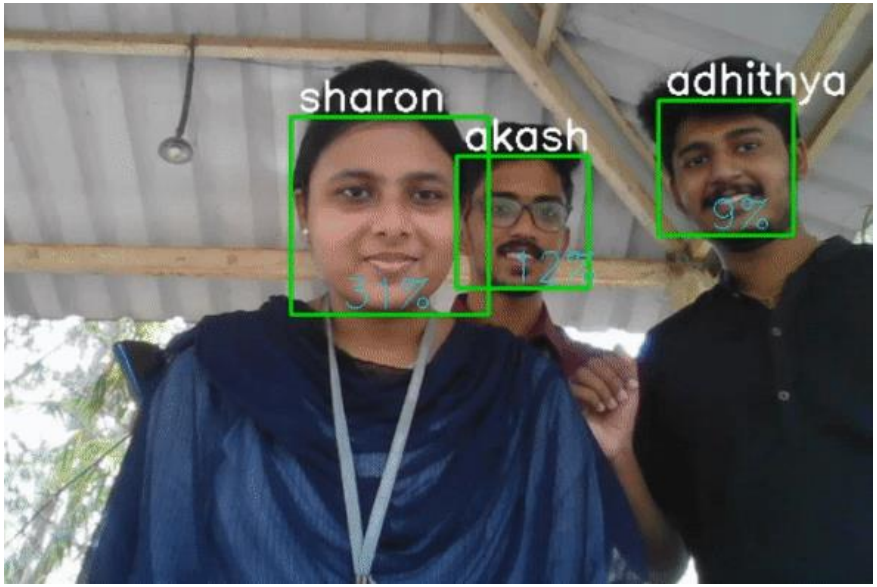
Techniques Experimented	Test Cases	Time Duration (In minutes)	Actual number of frames extracted	Number of key frames detected	No. of Face Detected
Singular Value Decomposition & Dynamic Clustering	Sample Test Case	1 min 22 secs	25	2370	3
	Test Case-1	1 min 13 secs	2	1803	3
	Test Case-2	22 sec	3	685	0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

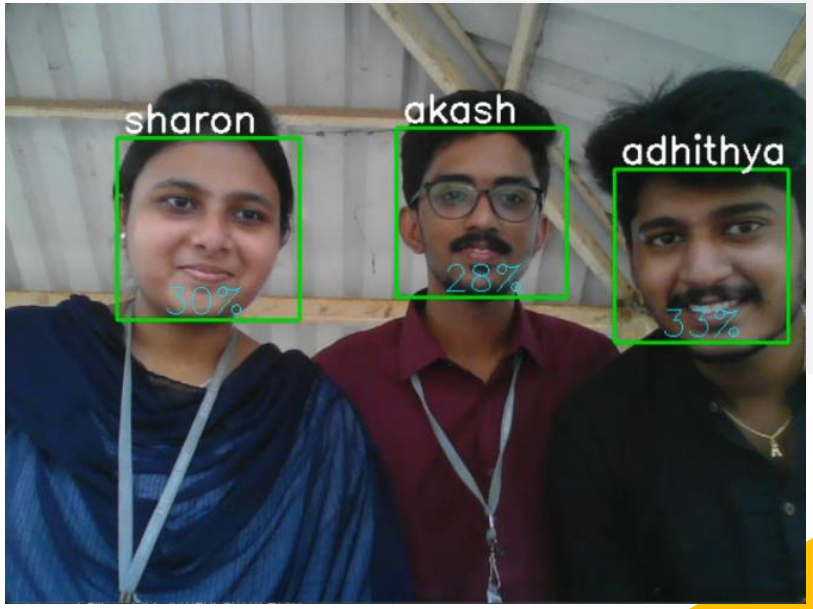
COMPARISON OF DIFFERENT CONSTRAINTS

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**

TEST CASES FOR FACE RECOGNITION

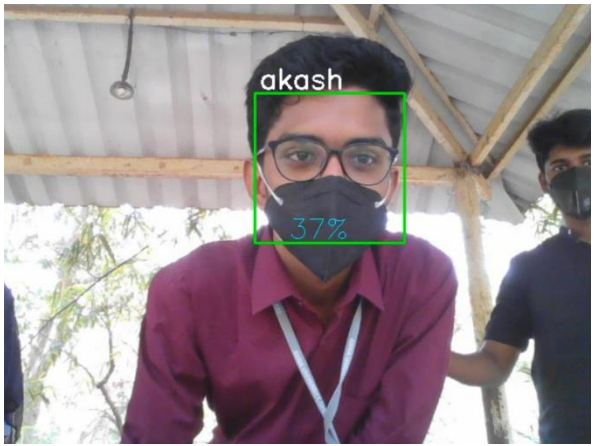
TEST CASE 1

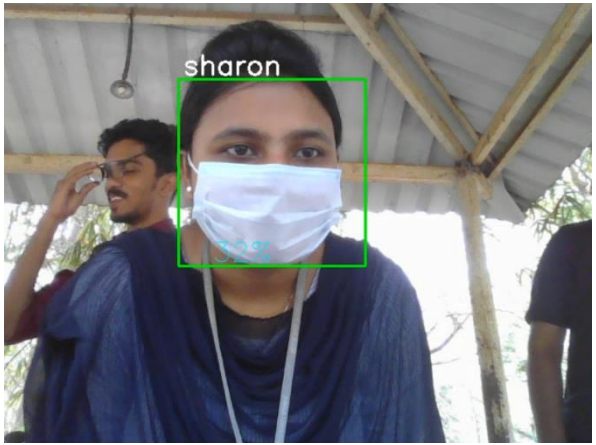


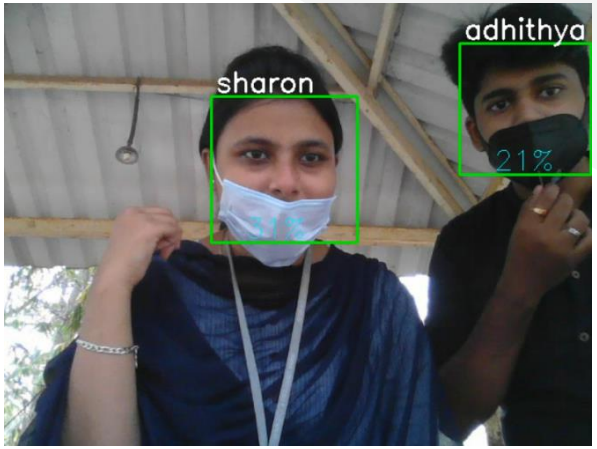


FACE RECOGNITION: TEST CASE 2

OCCLUSIONS

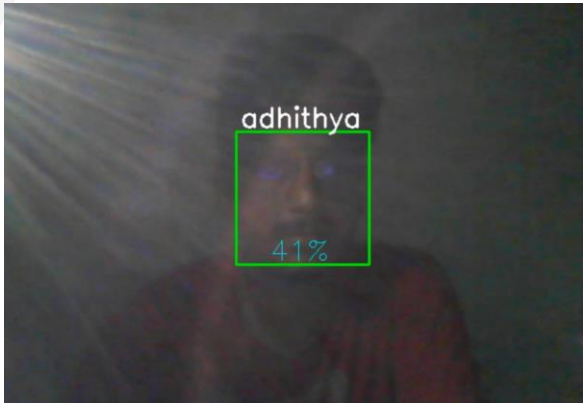






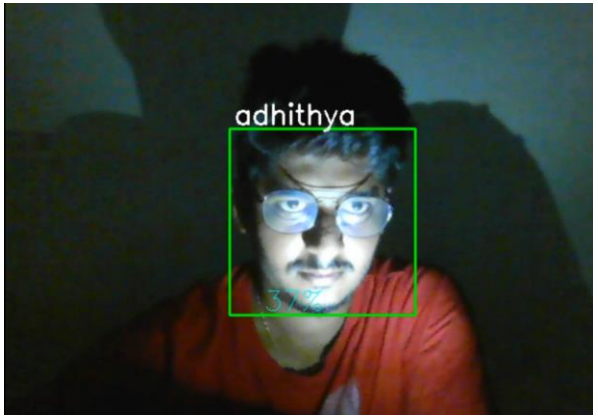
FACE RECOGNITION: TEST CASE 3

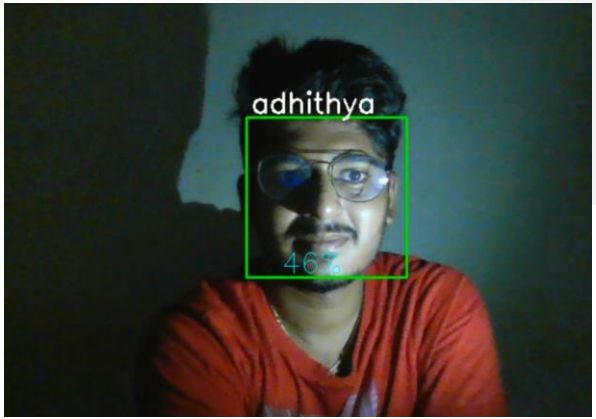
DARK ENVIRONMENT



FACE RECOGNITION: TEST CASE 4

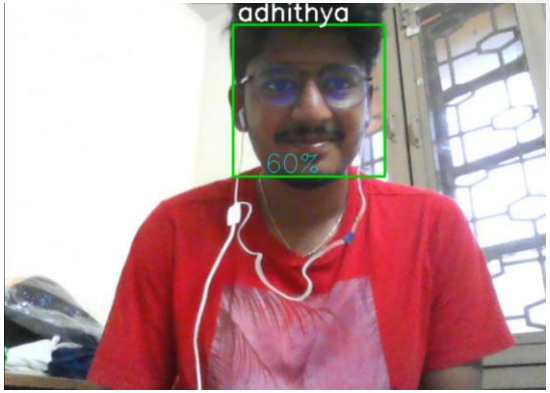
LIGHT ILLUMINATION CHANGES

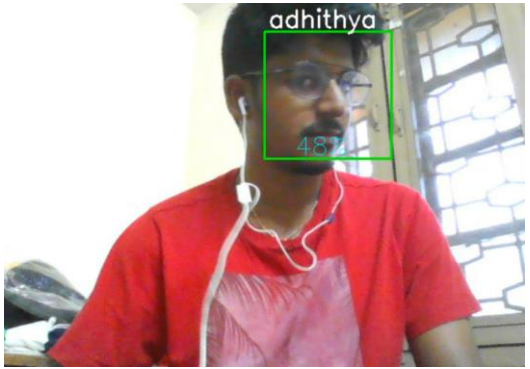




FACE RECOGNITION: TEST CASE 5

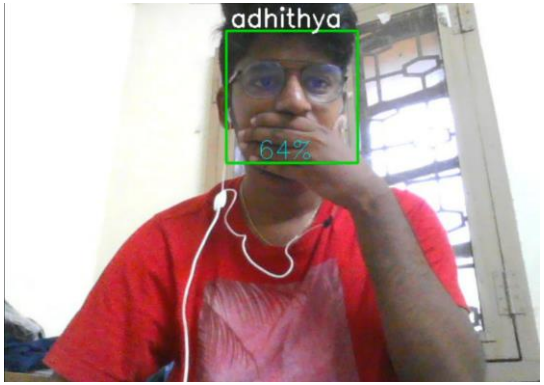
EXPRESSION AND POSE VARIATIONS

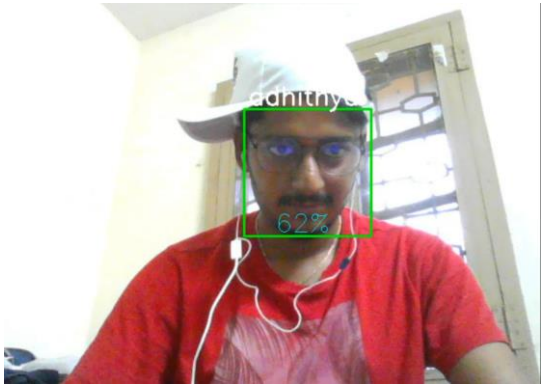


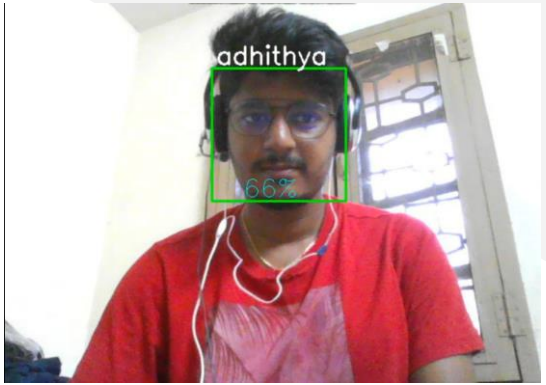


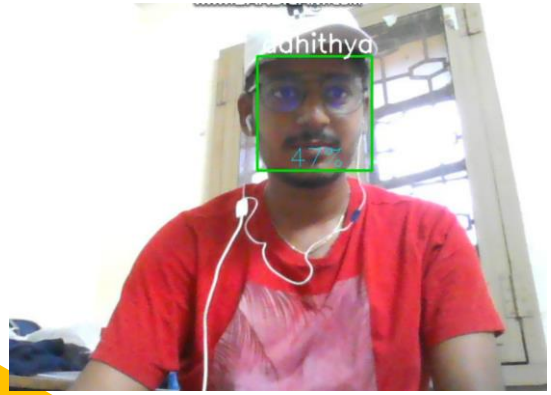
FACE RECOGNITION: TEST CASE 6

OCCLUSIONS IN INDOOR ENVIRONMENT

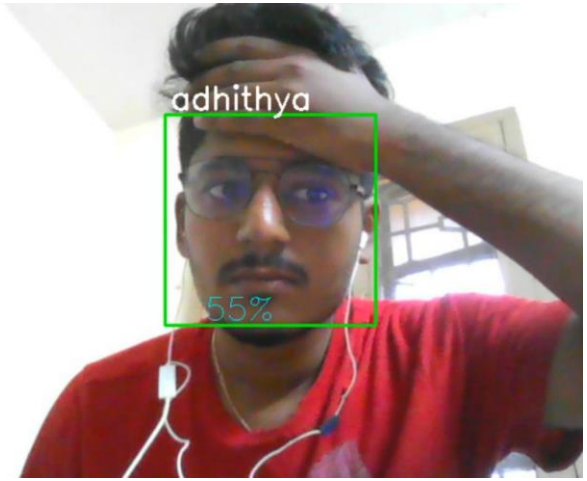


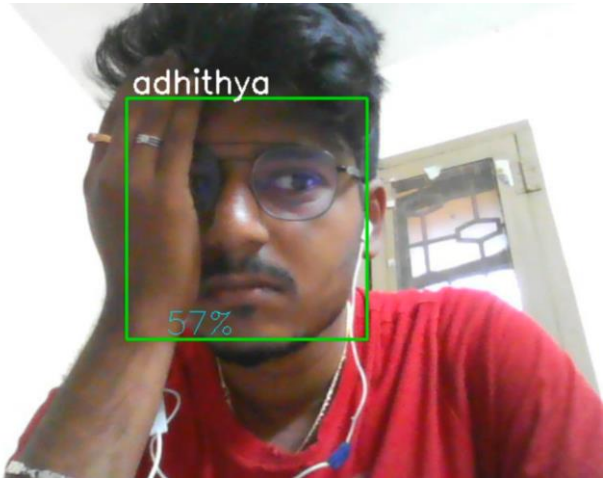






OCCLUSIONS IN INDOOR ENVIRONMENT CONTD..



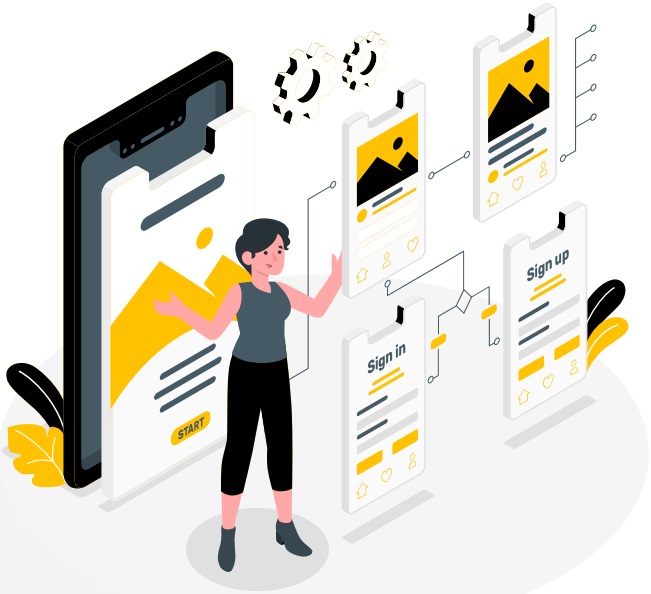


Remarks on Second Review

1. We were instructed to include pseudo codes for the algorithms involved in the project.

1. More references were asked to be included in the document.

REFERENCES:



1. C. Whitelam et al., “IARPA janus benchmark-B face dataset,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW), 2017, pp. 592–600.
2. 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.
3. 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.
4. 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.
5. 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] H. Li, G. Hua, X. Shen, Z. Lin, and J. Brandt, “Eigen-pep for video face recognition,” in Proc. Asian. Conf. Comput. Vis., 2014.

[12] 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.

[13] A. RoyChowdhury, T.-Y. Lin, S. Maji, and E. Learned-Miller, “Face identification with bilinear cnns,” arXiv preprint arXiv:1506.01342, 2015.

[14] 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.

[15] 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.

[16] 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.

Thank You!!!