Voila Jones Algorithm (Face Detection)

→ Face detection has several application.

→

Algorithmic Flow

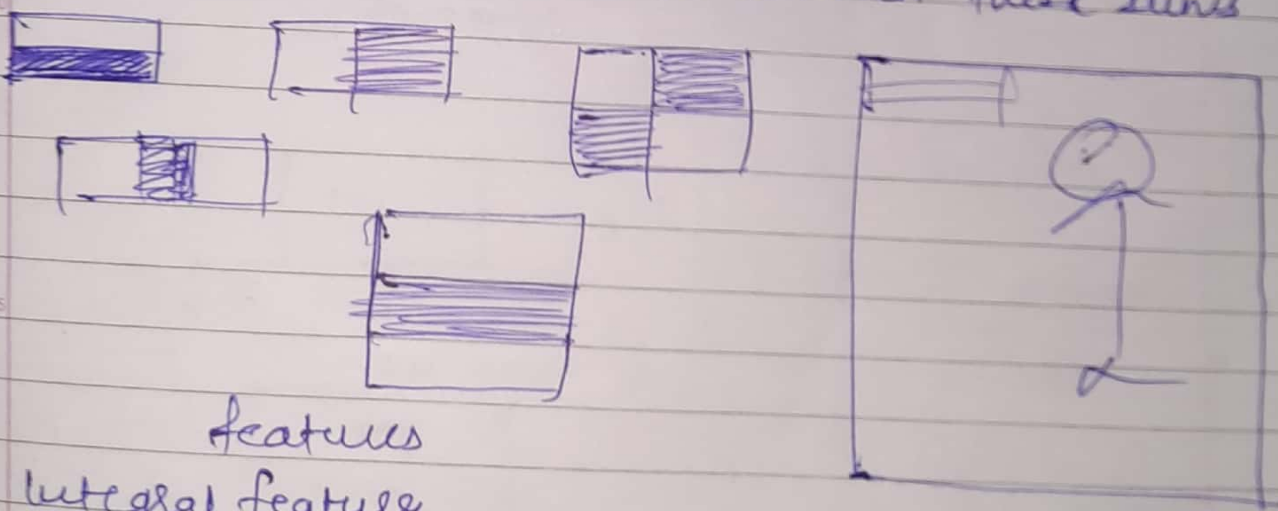
Input Image ⇒ Haar feature selection ⇒ Integral Image ⇒ Adaboost Training ⇓ Cascading Classifiers

Goal: Face detection not face recognition

Highly Robust.

Haar Features

- Adjacent rectangular regions at a specific location
- Sums up the pixel intensities in each region
- Calculate the difference between these sums



features

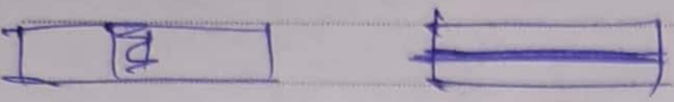
Integral feature

For each pixel add all the intensities in the previous coloumns and row

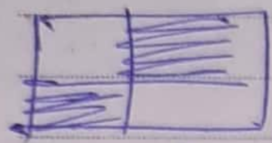
$$ii(x, y) = \sum_{x' \leq x,\; y' \leq y} i(x', y')$$

Boosting

→ Edge features

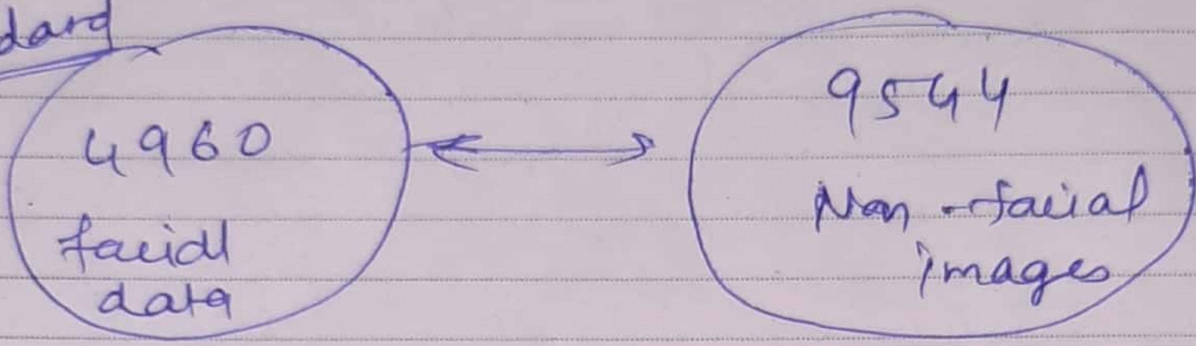


→ Line features



Four rectangle features

Standard



$$F(x) = a_1 f_1(x) + a_2 f_2(x) + a_3 f(x)$$

weights.

→ Not all features are created equal

→ Initializing each weight $w_i = \frac{1}{N}$

and then we can normalize weights

$$w_{t,i} = \frac{w_{t,i}}{\sum_{j=1}^{N} w_{t,j}}$$ → w_i is probability distribution

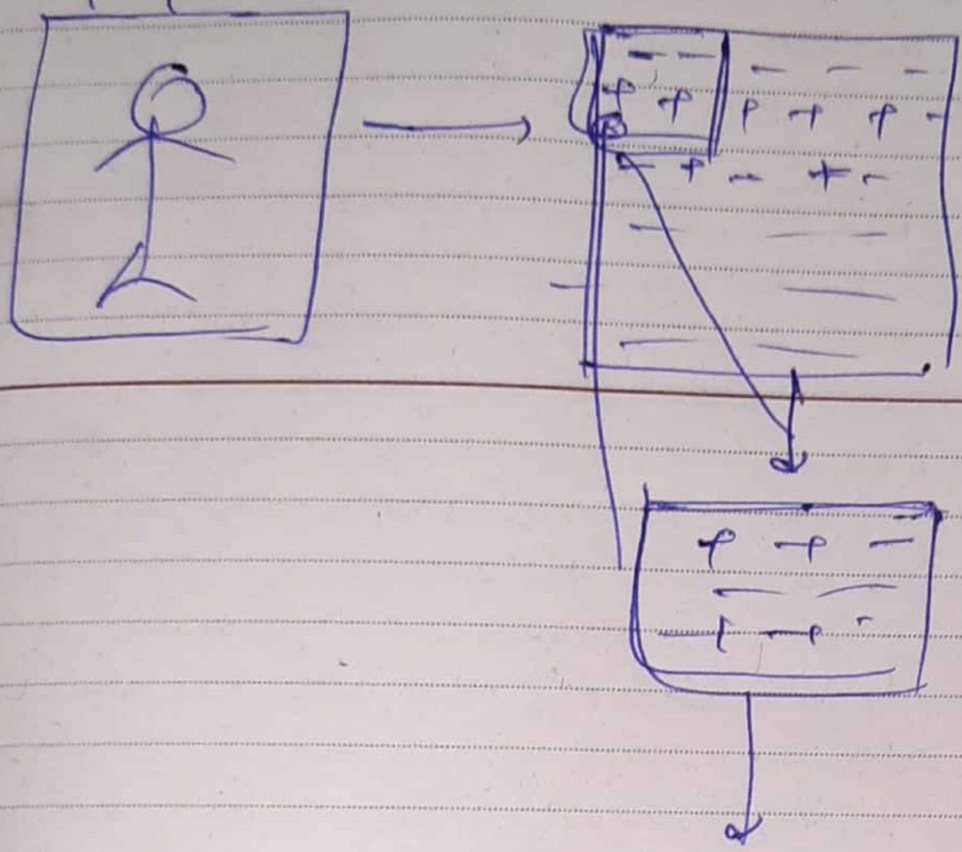
Cascading.

Trying all the features is time consuming but with cascading procedure get faster

window → F1 —Accept→ F2 —Accept→

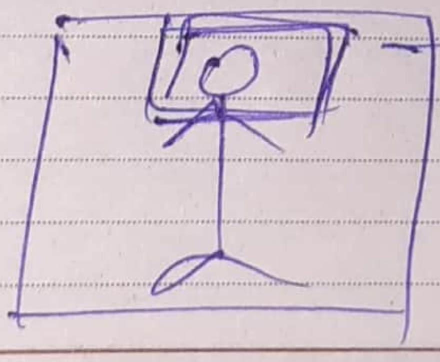
Reject ↓   ↓ Reject

Gray scale Image



Saturday 16

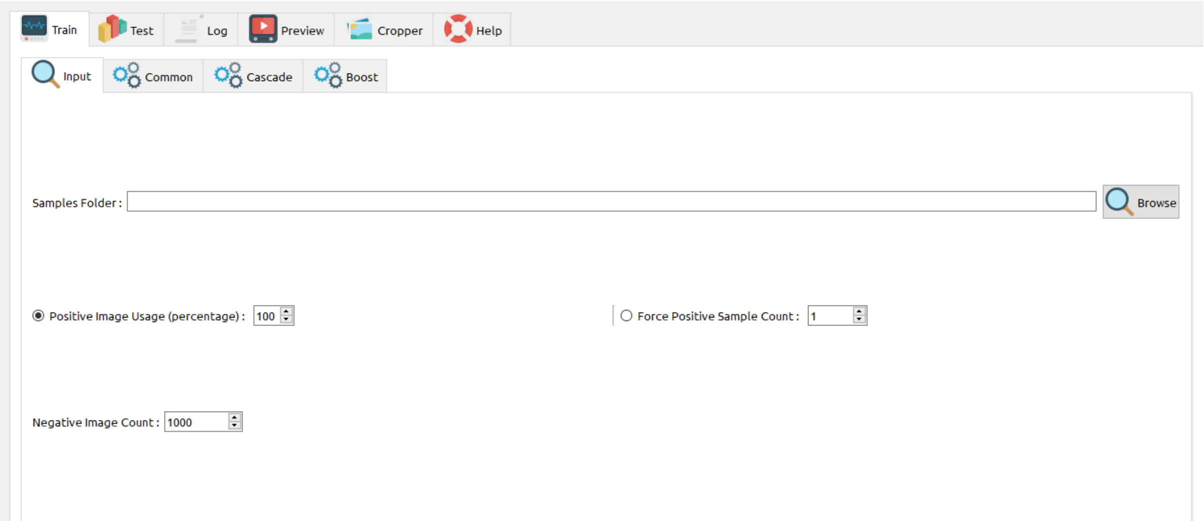
Face detected



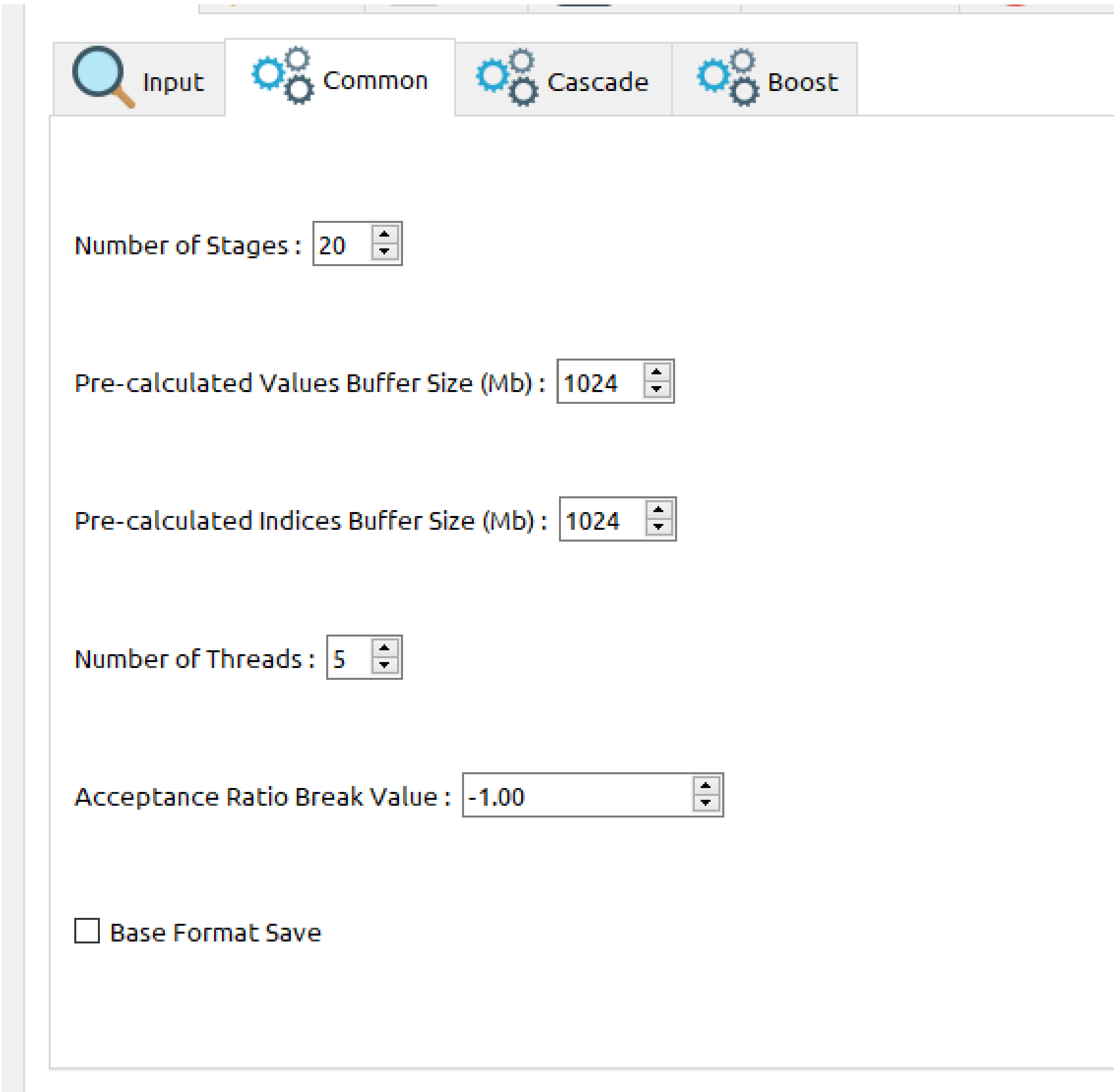
→ detection

For making cascade_final.xml document used following procedure used in viola jones algorithm :

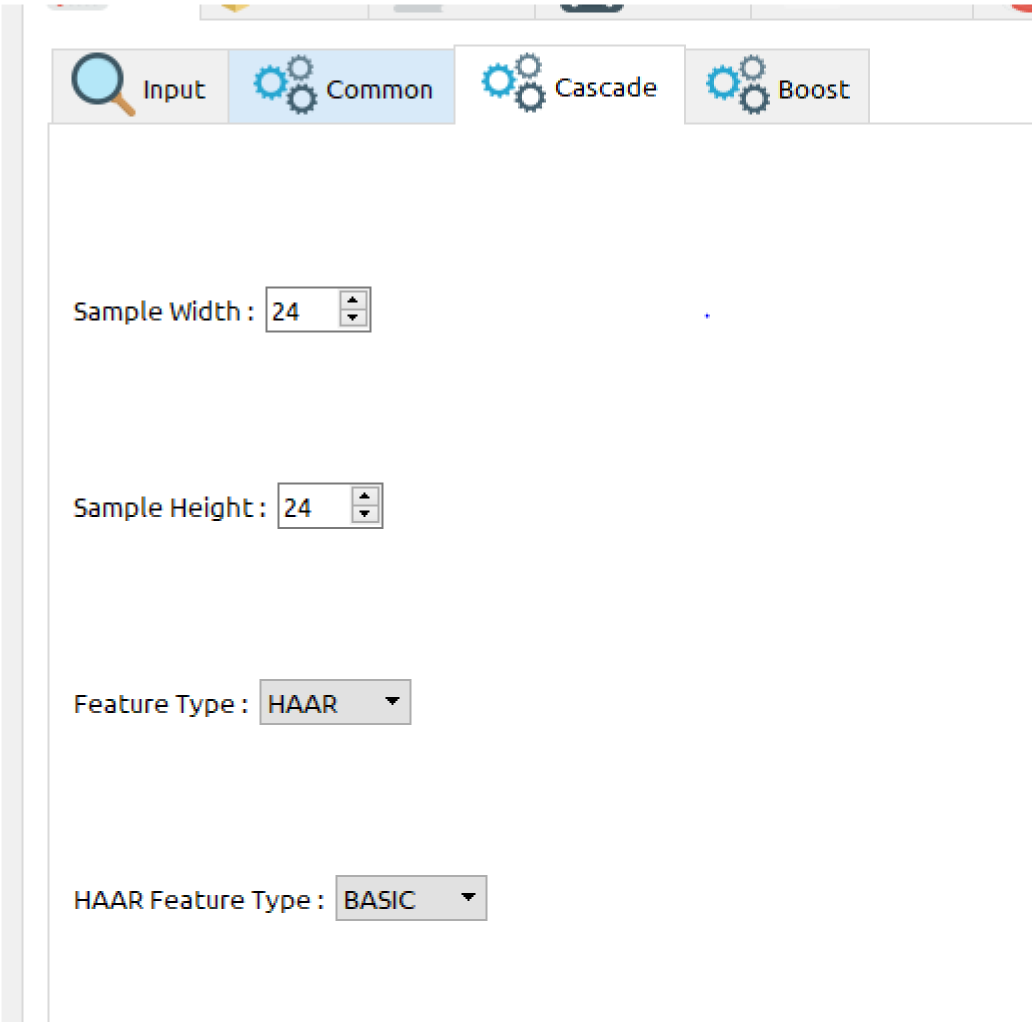
1) Created data set which had positive images (images which had the face) and negative images (images which don't have faces)
2) After creating dataset, included the sample in below window,



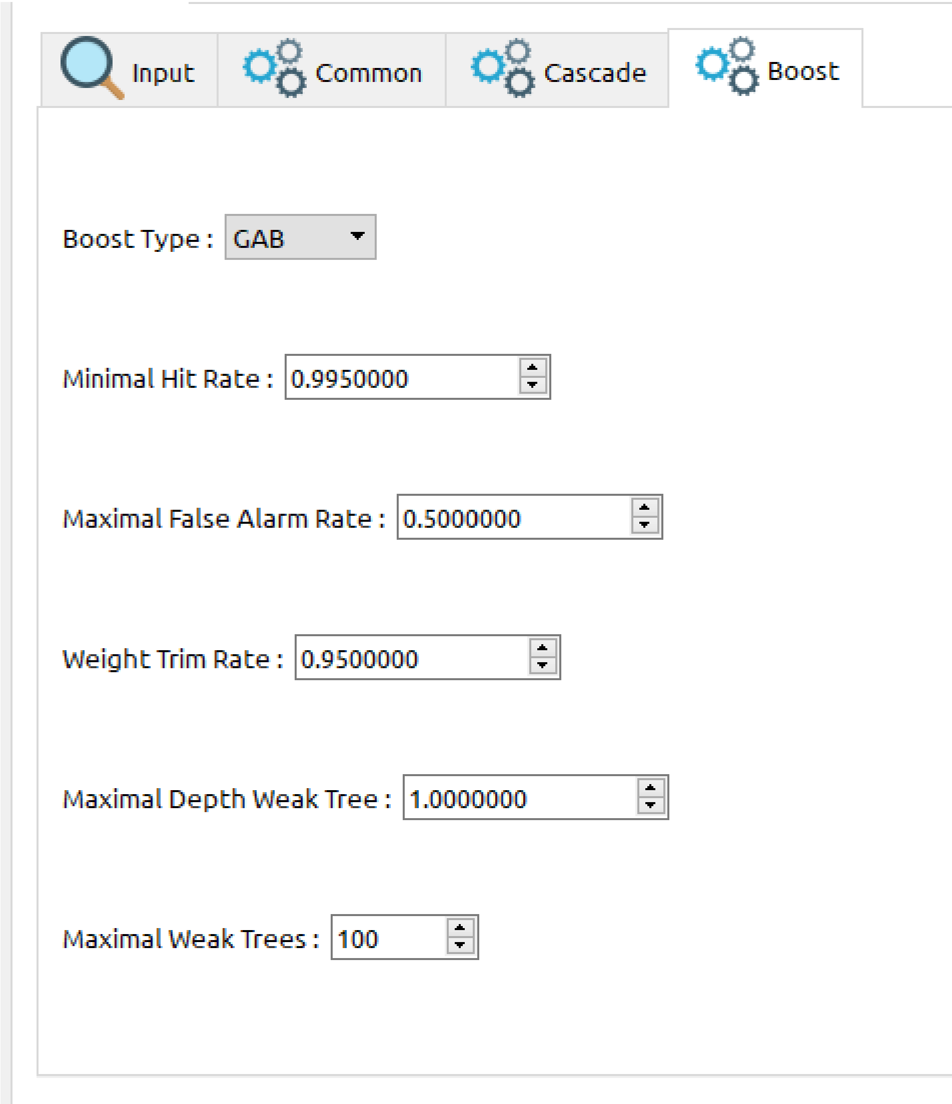
3) Selected some attributes



4) Now Cascaded the dataset in the following step in the software.



5) After filling the information in each steps



I have used dataset of having positive images value around 2000 images and negative images of around 4500 images, it required an overnight to make the final xml document.

Ran Viola – Jones Algorithm on the Avengers picture and result is as follows:

I further calculated the max_height and max_width for which the image can be cropped, actually resizing the image increases the accuracy in face recognition. But we can do the step that we crop the face only and hence it will further increase our model accuracy and also help us to reduce complexity of the model.

Credits for XML document : Cascade Trainer GUI

References: https://www.youtube.com/watch?v=uEJ71VlUmMQ

https://www.youtube.com/watch?v=LopYA64KmdE&t=602s

Mind Map

Detection $\rightarrow$ Classification of faces

Integrating face detection and Recognition system in real time.

1. We will do face detection using OpenCV and will provide cropped image to the training of the recognition system.

Now in training procedure;

Training

⇒ Suppose we have M images that are of dimension NXN, we first convert all images into vectors of dimension $N^2 \times 1$. Each of it is denoted by Γ_i

⇒ We next find 'mean face', i.e the mean of all the M vectors as Ψ

$$\Psi = \Sigma \Gamma_i$$

⇒ Now we will find offsets of image from the mean face by substracting every image as

$$\phi_i = \Gamma_i - \Psi$$

⇒ Next want to find the covariance matrix of A where

$$A = [\phi_1 \; \phi_2 \; --- \; \phi_M] \rightarrow \text{offsets of all the images.}$$

$$C = \frac{1}{M} \sum_{n=1}^{M} \phi_n \phi_n^T = AA^T \quad (N^2 \times N^2 \text{ matrix}).$$

where $A = [\phi_1 \; \phi_2 \; -- \; \phi_M]$ $(N^2 \times M \text{ matrix})$

⇒ There are procedures / methods here to find eigenvectors and eigen values of covariance matrix

Case 1 : when N is small
if is N is small, then for $N^2 \times N^2$ covariance matrix is not greater ~~[illegible]~~ and can be obtained in python

Case 2: When N is large
N↑ Computation power ↑
to calculate eignvector
Hence we calculate eigen vector of small covariance matrix

$$\boxed{C_S = A^T A \quad (M \times M) \text{ matrix.}}$$

Calculate eigen vector for this small matrix so to find eigen vector of covariance matrix. we can multiply that vector to A

$$\boxed{u_i = A v_i} \rightarrow \text{eigen vector of small covariance matrix}$$

eigen vector of covariance matrix

⇒ Now next step is to calculate weight matrix

$$\hat{\phi}_i - \text{mean} = \sum_{j=1}^{k} w_j u_j \quad (w_j^i = u_j^T \phi_j)$$

$$\Omega_i = \begin{bmatrix} w_1^i \\ w_2^i \\ | \\ w_k^i \end{bmatrix} \qquad i = 1, 2, \ldots M$$

Ω_i represents the weight matrix of M train images.

Camlin Page
Date

=> Now, Instead of taking all the weights we have taken best K weights.

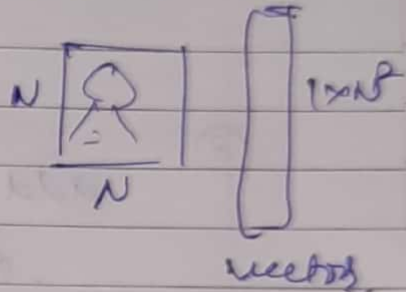
→ Corresponding to that K weights we have eigen vectors.

=> Now, training ends, here for testing

<u>Testing</u>

→ ① Read and convert the image into vector.

→ Subtract the mean from the vector



$$\Phi_i = \Gamma_i - \Psi$$

get this offset

→ Now from this offset calculate weight matrix

$$\hat{\Phi} = \sum_{i=1}^{K} w_i u_i \quad (w_i = u_i^T \Phi)$$

$$\Omega = \begin{bmatrix} w_1 \\ w_2 \\ | \\ w_N \end{bmatrix}$$

→ Calculate minimum distance bet Ω_{Train} and Ω_{test}

$$e_r = \min \|\Omega - \Omega^i\|$$

e_r we will get which image is close to which set of person.

Scanned with CamScanner

Defining $\boxed{\text{Accuracy} = \frac{\text{Correctly matched}}{\text{Total Test Images}} \times 100}$

Mind Map (Train)

(1)

N×N Image ⟹ Vector

$N^2 \times 1$

⇓

(2) add them continuously in a loop and obtain the average Image (Ψ)

$$\phi_i = \Gamma_i - \Psi$$

$$A = [\phi_1 \ \phi_2 \ \cdots \ \phi_M] \quad (N^2 \times M) \text{ Matrix}$$

(3)

$$C = \frac{1}{M} \sum_{n=1}^{M} \phi_n \phi_n^T = AA^T \quad (N^2 \times N^2) \text{ Matrix}$$

where $A = [\phi_1 \ \phi_2 \ \cdots \ \phi_m]$ $(N^2 \times M)$ Matrix

Calculate eigenvectors and eigenvalues directly

$C_s = A^T A$ $(M \times M)$ Matrix

$U_i = A v_i$ → small covariance matrix

↑ eigenvector

(4) $$\hat{\phi}_i - \text{mean} = \sum_{j=1}^{k} w_j^i u_j \quad (w_j^i = u_j^T \phi_i)$$

$$\Omega_i = \begin{bmatrix} w_1^i \\ w_2^i \\ \vdots \\ w_k^i \end{bmatrix} \quad i = 1, 2 \ldots M$$

Now <u>Test (Map)</u>

Read the Image ⇒ Convert into vector form ⇒ Subtract Mean from flattened Image

$\phi_i = \Gamma_i - \psi$

⇓

Calculated minimum distance (er) $e_r = \min \|\Omega - \Omega_i\|$ → Finally get Accuracy of overall model.

→ <u>Important Note :-</u>

Here value of k → complexity of Model
In Machine Learning Models, we have to keep complexity and accuracy both in mind. For increase in some amount of accuracy we can't increase our complexity.

In [*]:

```
import cv2

face_cascade = cv2.CascadeClassifier('cascade_final_1.xml')

img = cv2.imread('avengers-endgame.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray,1.1,1)

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

cv2.imshow('img',img)
cv2.waitKey()
```

In [9]:

```
Max_Width=0
Max_Height=0

images_train = 8
variants = 15

for i in range(1, variants+1):
    for j in range(1, images_train+1):
        if (i<10):
            face_image = cv2.cvtColor(cv2.imread("train/Subject0"+str(i)+" (" + str(j) + ")
        else:
            face_image = cv2.cvtColor(cv2.imread("train/Subject"+str(i)+" (" + str(j) + ").
        faces=face_cascade.detectMultiScale(gray,1.1,1)
        for (x,y,w,h) in faces:
            cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
            if (Max_Width<w):
                Max_Width=w
            if (Max_Height<h):
                Max_Height=h
for i in range(1, variants+1):
    for j in range(images_train, images_train+1+3):
        if (i<10):
            try:
                face_image = cv2.cvtColor(cv2.imread("test/Subject0"+str(i)+" (" + str(j) +
            except:
                continue
        else:
            try:
                face_image = cv2.cvtColor(cv2.imread("test/Subject"+str(i)+" (" + str(j) +
            except:
                continue
        faces=face_cascade.detectMultiScale(gray,1.1,1)
        for (x,y,w,h) in faces:
            cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
            if (Max_Width<w):
                Max_Width=w
            if (Max_Height<h):
                Max_Height=h
```

In [10]:

```
Max_Width
```

Out[10]:

```
104
```

In [11]:

```
Max_Height
```

Out[11]:

```
104
```

In [ ]:

```

```

Classifying Detected Faces using idea of Eigen Faces

In [5]:

```
# Importing all the important libraries which we will require in further steps
import numpy as np
import cv2
from matplotlib import pyplot as plt
import math
```

We need to resize our image as we take input, since from our analysis of the viola jones algorthim we got to know that maximum sized face in image lies between width and height of 104 and 104 respectively.

In [42]:

```
total_pixels = 10816
images_train = 8 # Training Images
variants = 15  # Total Number of Subjects
total_images = images_train*variants

face_vector = []
# Running two for loops for training all the variants and 8 images of each images
# Hence total images used for training =120
for i in range(1, variants+1):
    for j in range(1, images_train+1):
        if (i<10):
            face_image = cv2.cvtColor(cv2.imread("train/Subject0"+str(i)+" (" + str(j) + ")
            # Mention the width and height of the size of the images
            width = 104
            height = 104
            dim = (width, height)

            # resize image
            face_image = cv2.resize(face_image, dim, interpolation = cv2.INTER_AREA)
        else:
            face_image = cv2.cvtColor(cv2.imread("train/Subject"+str(i)+" (" + str(j) + ").
            width = 104
            height = 104
            dim = (width, height)

            # resize image
            face_image = cv2.resize(face_image, dim, interpolation = cv2.INTER_AREA)
        plt.imshow(face_image, cmap = 'gray', interpolation = 'bicubic')
        plt.show()
        # Plotting image for verification
        face_image = face_image.reshape(total_pixels,)
        # We also change our shape in above step
        face_vector.append(face_image)
        # Finally we store each image data into the face_vector

face_vector = np.asarray(face_vector)
face_vector = face_vector.transpose()

print(face_vector.shape)
print(face_vector)
```

Now taking all the images and appending them into the facevector we need to subtract each vector with it's mean. Hence we get normalized face vector

In [43]:

```
avg_face_vector = face_vector.mean(axis=1) # Taking mean along axis=1
avg_face_vector = avg_face_vector.reshape(face_vector.shape[0], 1) # Reshaping vector to su
normalized_face_vector = face_vector - avg_face_vector # Now subtracting each vector with m
print(normalized_face_vector)
```

```
[[  7.775        7.775        7.775       ... -34.225        7.775
    7.775      ]
 [  6.05         6.05         6.05        ... -26.95         6.05
    6.05       ]
 [  5.275        5.275        5.275       ...  -7.725        5.275
    5.275      ]
 ...
 [ -9.525       29.475       29.475       ...  26.475       28.475
   25.475      ]
 [ -5.91666667  28.08333333  28.08333333 ...  27.08333333  27.08333333
   19.08333333]
 [  0.55833333  31.55833333  31.55833333 ...  30.55833333  15.55833333
   32.55833333]]
```

Covariance Matrix Calculation from the Normalized Vectors of Each Image.

In [44]:

```
covariance_matrix = np.cov(np.transpose(normalized_face_vector))
print(covariance_matrix)
```

```
[[ 7135.55123252  1677.95284541  1007.63837433 ... -4568.29151024
    -66.70706089  -984.24448088]
 [ 1677.95284541  2724.07587853   448.87689574 ... -1218.61996653
   -473.10453375  -849.68322177]
 [ 1007.63837433   448.87689574  1850.42961998 ...  -710.31428453
     62.07375967   -21.44972611]
 ...
 [-4568.29151024 -1218.61996653  -710.31428453 ...  6354.81426635
    364.30136077   381.15223038]
 [  -66.70706089  -473.10453375    62.07375967 ...   364.30136077
   2093.30753961   367.91063954]
 [ -984.24448088  -849.68322177   -21.44972611 ...   381.15223038
    367.91063954  3448.51324958]]
```

Since Python can do ton's of calculation each second we don't need to go for the second step of the calculating smaller matrix.

In [45]:

```
eigen_values, eigen_vectors = np.linalg.eig(covariance_matrix)
```

In [47]:

```
# This is would the step called as PCA
# Here we take only top 60 eigen vectors for the further prediction of the test images
# If k is small we reduce the complexity of model heavily
print(eigen_vectors.shape)
k = 60
k_eigen_vectors = eigen_vectors[0:k, :]
print(k_eigen_vectors.shape)
```

```
(120, 120)
(60, 120)
```

In [48]:

```
# Now we need to lower the dimension and also we complete the step of having dot product wi
# face vector
eigen_faces = k_eigen_vectors.dot(np.transpose(normalized_face_vector))
print(eigen_faces.shape)
```

```
(60, 10816)
```

In [49]:

```
# Now the there are K eigenvector which carries information of all the images
# All the image normalized vector are the linear combination of the selcted K eigenvectors
# And hence we calculate the weights of such linear combination
weights = np.transpose(normalized_face_vector).dot(np.transpose(eigen_faces))
print(weights)
```

```
[[-19751293.08461218-512723.31719035j -30833963.95790347-512723.31719035j
   41071047.00050665-512723.31719038j ...
  -21148717.96916098-512723.31719036j  24041258.84470751-512723.31719038j
   41889351.93616537-512723.31719034j]
 [-13866656.53824024+118792.51405259j  -5290483.56858655+118792.51405259j
    8416135.42293709+118792.51405259j ...
    3614185.82647408+118792.51405258j  15671292.44183596+118792.51405259j
   19593882.12448596+118792.51405259j]
 [ -8923876.6041844 +233474.78611274j    663862.44374345+233474.78611274j
    6198499.72604148+233474.78611275j ...
    4428246.09014259+233474.78611274j  13062070.37996404+233474.78611275j
    8777386.41807092+233474.78611273j]
 ...
 [ 30226678.38135644-265850.98779615j  14296439.86852476-265850.98779616j
  -18471496.92830129-265850.98779616j ...
   -4085791.02974588-265850.98779614j -33746605.54521682-265850.98779615j
  -46864329.58240522-265850.98779615j]
 [-14818874.21051635+267675.23917903j   4166257.90704249+267675.23917903j
   -4749577.29935316+267675.23917904j ...
    2078559.88958225+267675.23917903j   6837154.2083489 +267675.23917904j
    1856196.98144588+267675.23917903j]
 [  7348099.2554031 +611312.90196432j   8600884.26643787+611312.90196432j
  -13883122.3688818 +611312.90196435j ...
   10299523.59405019+611312.90196432j  -4715473.37990118+611312.90196435j
   -1821684.20731755+611312.90196431j]]
```

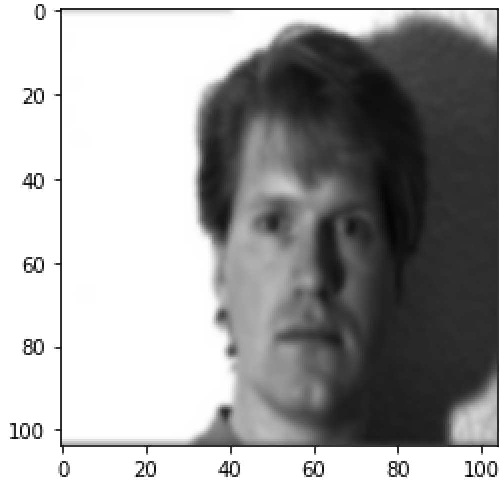
In [ ]:

```
# The below algorithm can be use for the manual looking that what would be predicted for ea

test_add = "test/Subject05 (11).jpg"
test_img = cv2.imread(test_add)
test_img = cv2.cvtColor(test_img, cv2.COLOR_RGB2GRAY)
# Again Redefining the Image
width = 104
height = 104
dim = (width, height)

# resize image
test_img = cv2.resize(test_img, dim, interpolation = cv2.INTER_AREA)


test_img = test_img.reshape(total_pixels, 1)
test_normalized_face_vector = test_img - avg_face_vector
test_weight = np.transpose(test_normalized_face_vector).dot(np.transpose(eigen_faces))


index =  np.argmin(np.linalg.norm(test_weight - weights, axis=1))

if(index>=0 and index <8):
    print("Prediction : Subject01")
if(index>=8 and index<16):
    print("Prediction : Subject02")
if(index>=16 and index<24):
    print("Prediction : Subject03")
if(index>=24 and index<32):
    print("Prediciton : Subject04")
if(index>=32 and index<40):
    print("Prediction : Subject05")
```

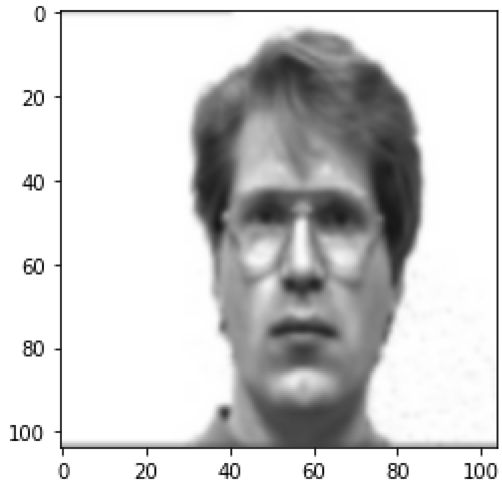
The below code is written for calculating the accuracy

In [51]:

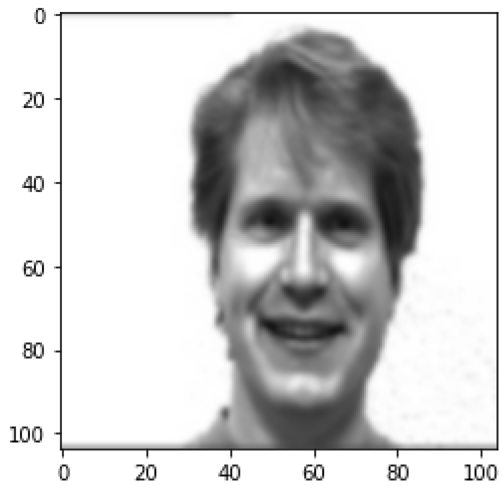
```
no_of_images_test=3
accuracy=0

for i in range(1, variants+1):
    for j in range(images_train+1,images_train+1+no_of_images_test):
        if (i<10):
            test_add="test/Subject0"+str(i)+" ("+str(j)+").jpg"
            test_img = cv2.imread(test_add)
            width = 104
            height = 104
            dim = (width, height)

            # resize image
            test_img = cv2.resize(test_img, dim, interpolation = cv2.INTER_AREA)
        else:
            test_add="test/Subject"+str(i)+" ("+str(j)+").jpg"
            test_img = cv2.imread(test_add)
            width = 104
            height = 104
            dim = (width, height)

            # resize image
            test_img = cv2.resize(test_img, dim, interpolation = cv2.INTER_AREA)
        plt.imshow(test_img, cmap = 'gray', interpolation = 'bicubic')
        plt.show()

        # The steps which are followed by the train images are same for the test images
        test_img = cv2.cvtColor(test_img, cv2.COLOR_RGB2GRAY)

        # Again we generate weights and normalized vectors by the mean vector of the traini
        test_img = test_img.reshape(total_pixels, 1)
        test_normalized_face_vector = test_img - avg_face_vector
        test_weight = np.transpose(test_normalized_face_vector).dot(np.transpose(eigen_face

        # In this step we obtain minimum of all the weights
        index =  np.argmin(np.linalg.norm(test_weight - weights, axis=1))

        predicted=math.floor(index/images_train)+1

        print("Real : "+str(i))
        print("Predicted : "+str(predicted))
        # After our Prediction matches with real scenario we increase value in accuracy var
        if predicted==i:
            accuracy=accuracy+1


accuracy=(accuracy*100)/(variants*no_of_images_test)
print("Accuracy : " ,accuracy)
```




```
Real : 1
Predicted : 1
```

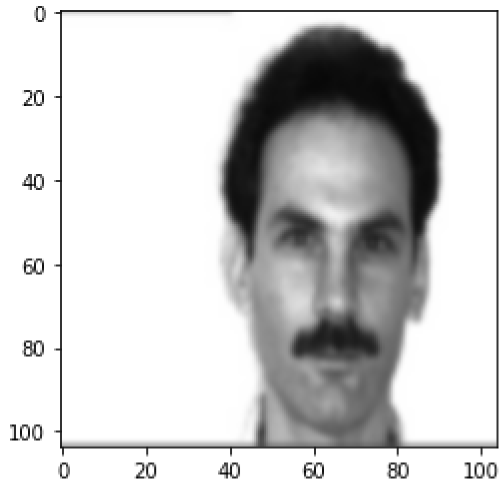



```
Real : 1
Predicted : 1
```

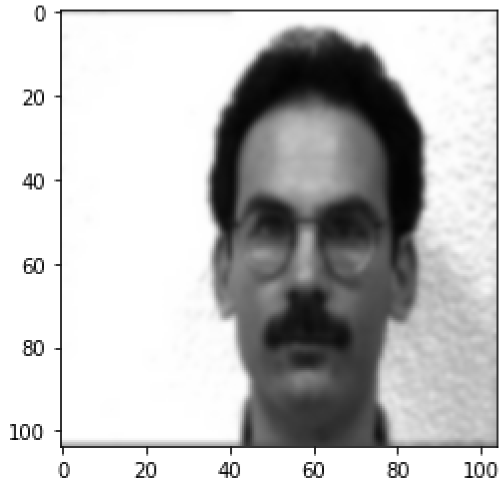



```
Real : 1
Predicted : 1
```

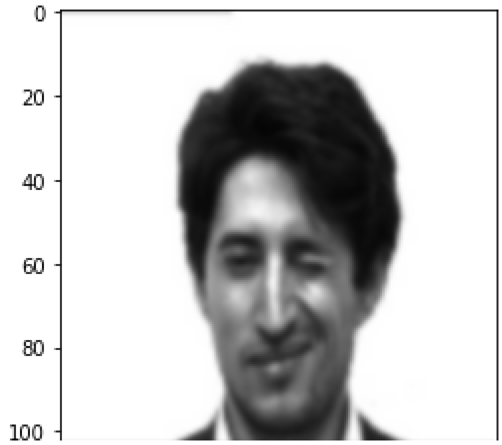

```
Real : 2
Predicted : 2
```



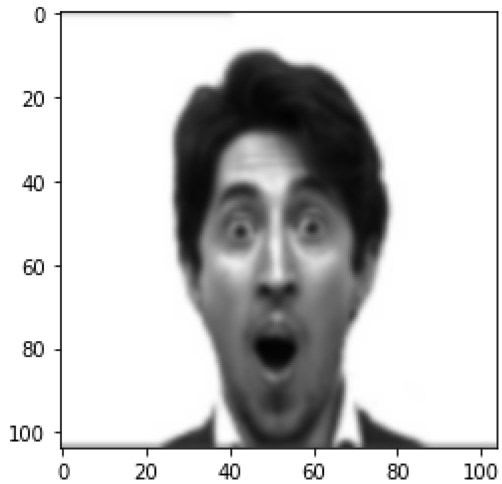

```
Real : 2
Predicted : 1
```



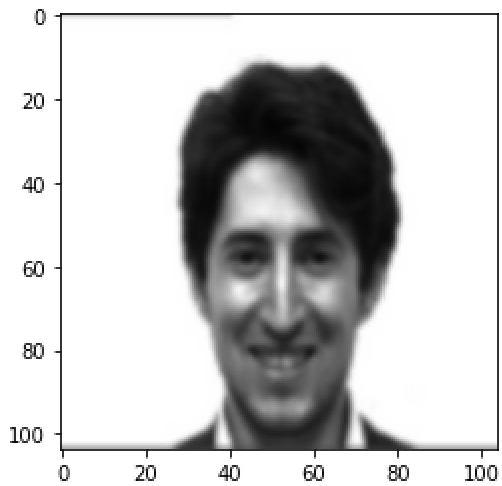

```
Real : 2
Predicted : 4
```



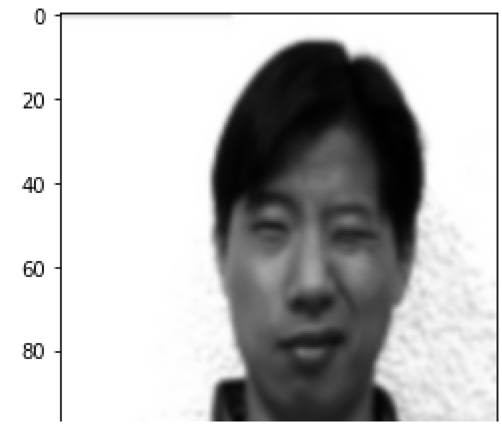

```
Real : 3
Predicted : 3
```



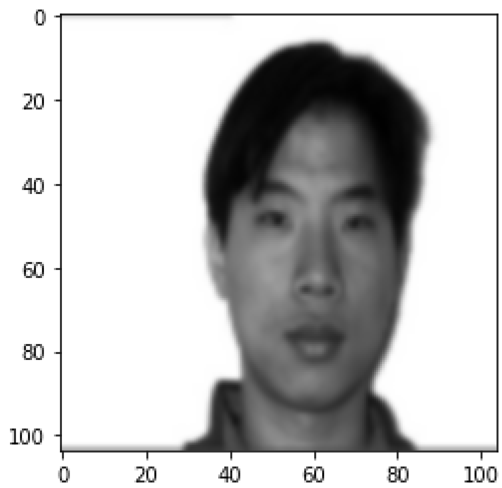

```
Real : 3
Predicted : 3
```



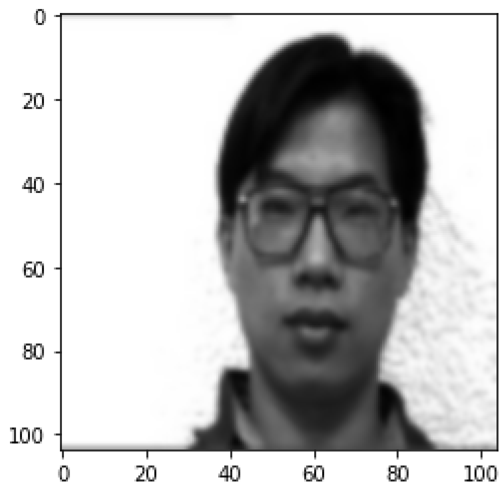

```
Real : 3
Predicted : 3
```



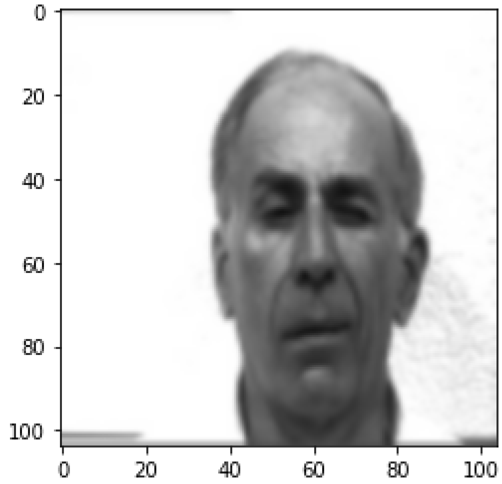
Real : 4
Predicted : 4



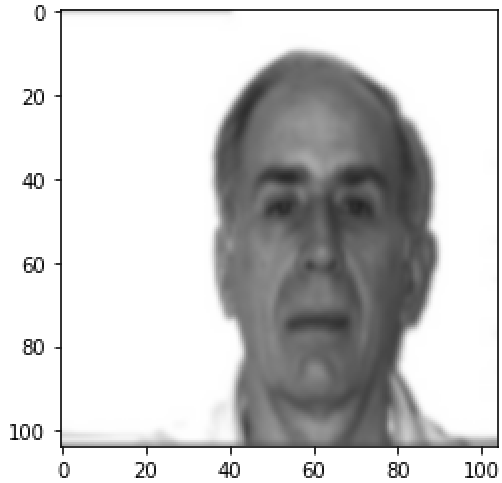
Real : 4
Predicted : 4



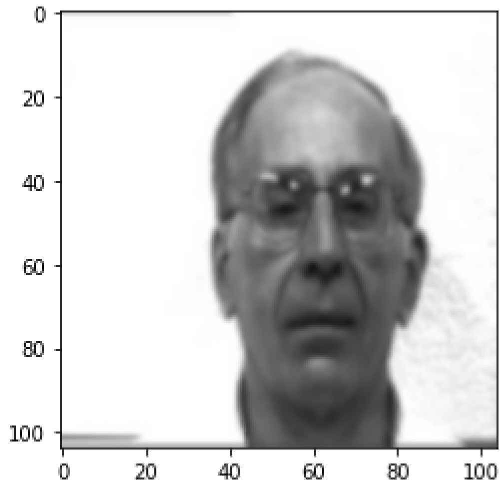
Real : 4
Predicted : 4



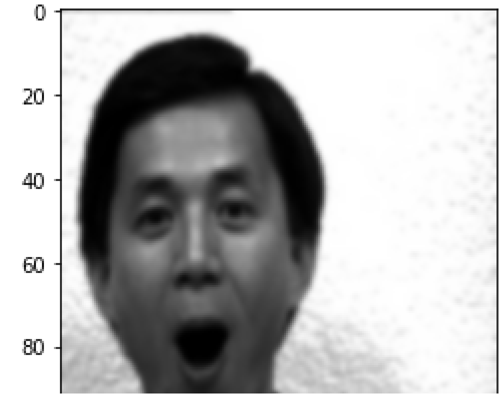
Real : 5
Predicted : 5



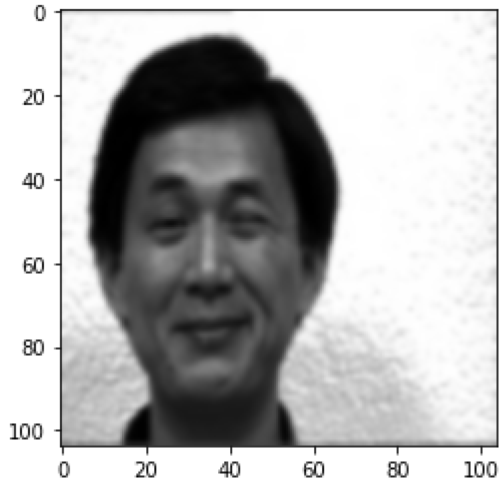
Real : 5
Predicted : 5



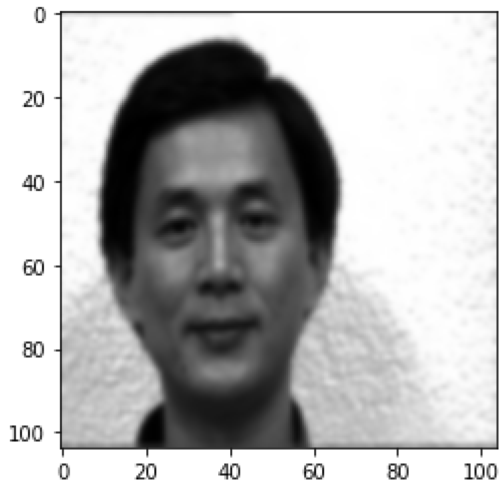
Real : 5
Predicted : 5



```
Real : 6
Predicted : 6
```

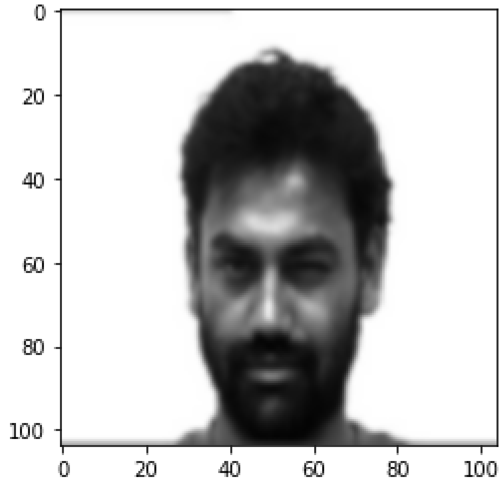


```
Real : 6
Predicted : 6
```

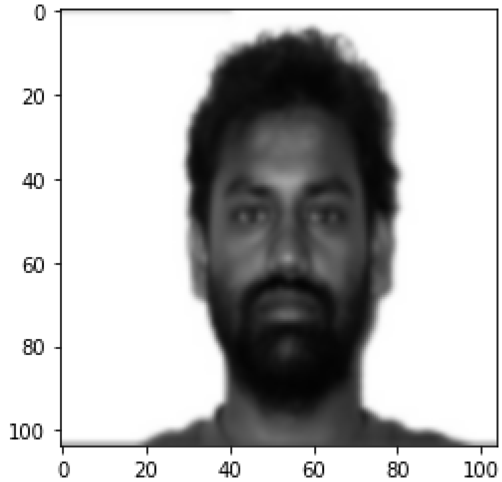


```
Real : 6
Predicted : 6
```

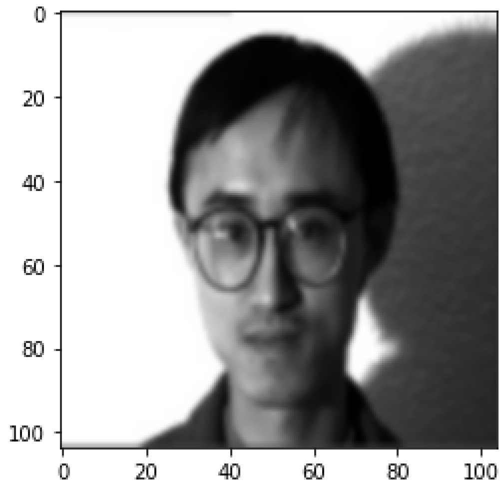

```
Real : 7
Predicted : 7
```



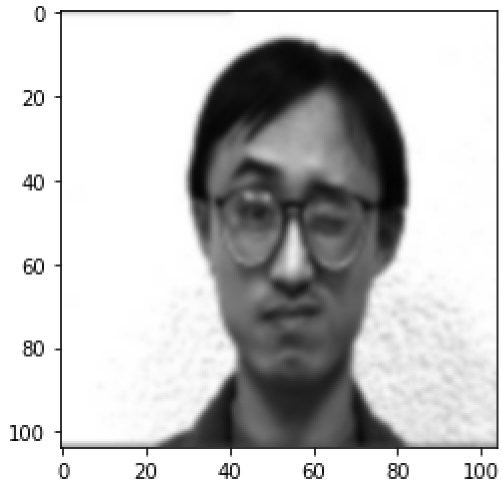

```
Real : 7
Predicted : 7
```



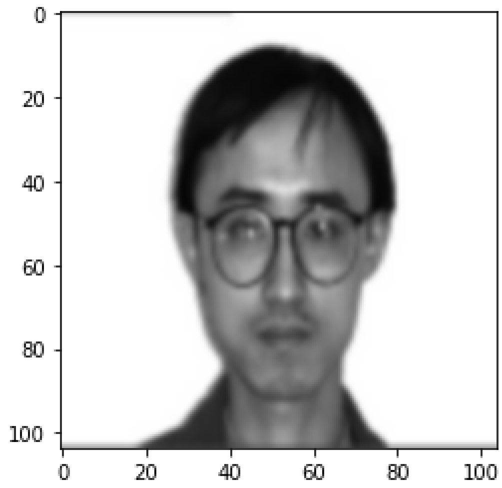

```
Real : 7
Predicted : 10
```



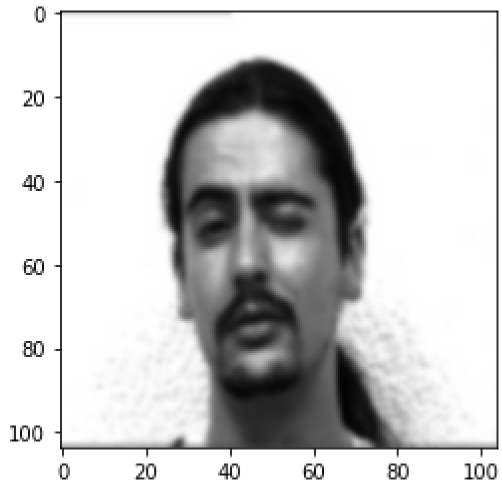
```
Real : 8
Predicted : 8
```



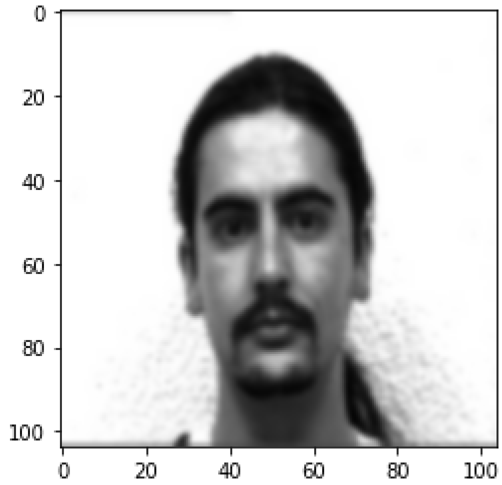

```
Real : 8
Predicted : 10
```



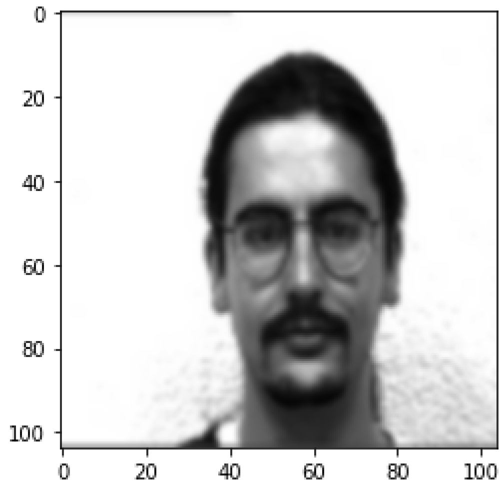

```
Real : 8
Predicted : 8
```



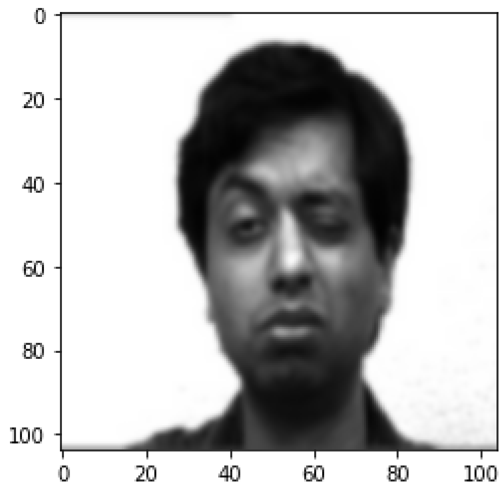

```
Real : 9
Predicted : 9
```



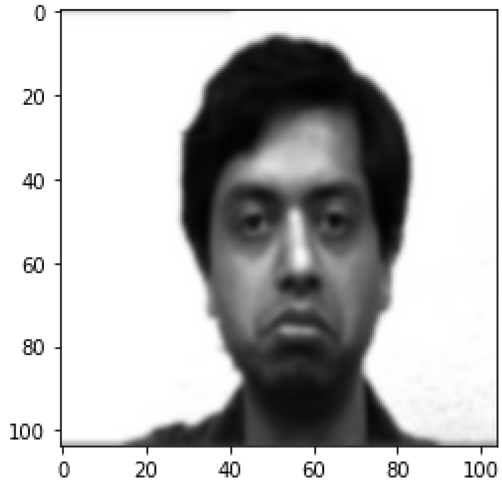
```
Real : 9
Predicted : 9
```



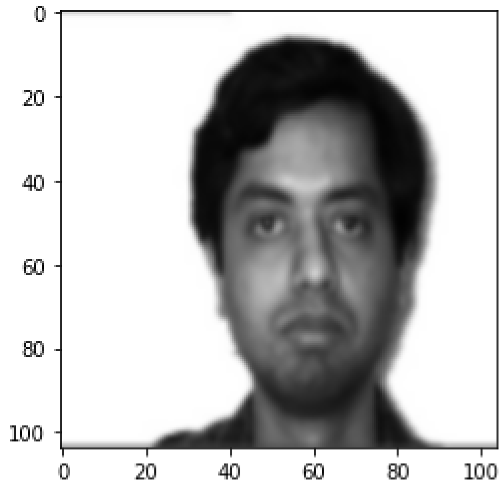
```
Real : 9
Predicted : 8
```



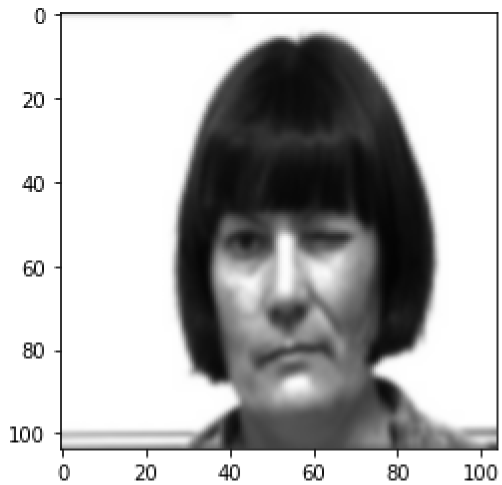
```
Real : 10
Predicted : 10
```




```
Real : 10
Predicted : 10
```

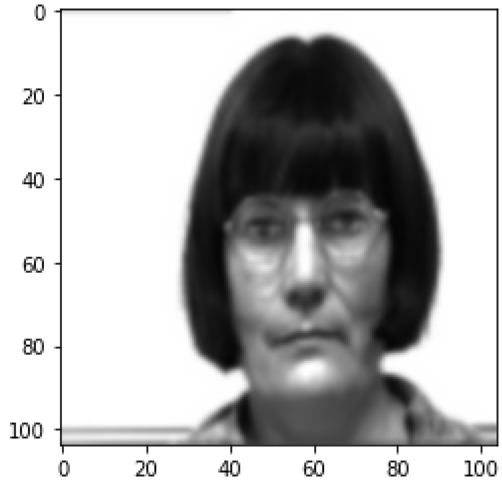



```
Real : 10
Predicted : 10
```

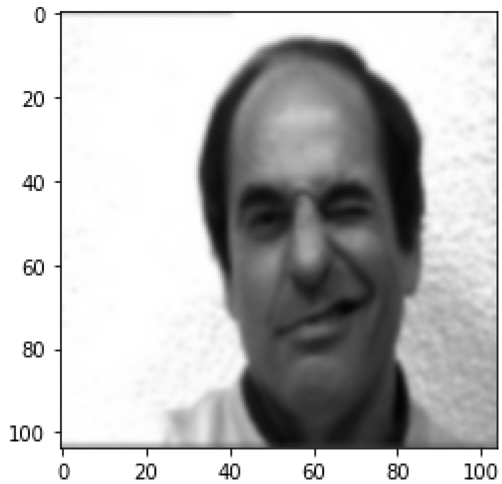



```
Real : 11
Predicted : 11
```

Real : 11
Predicted : 11

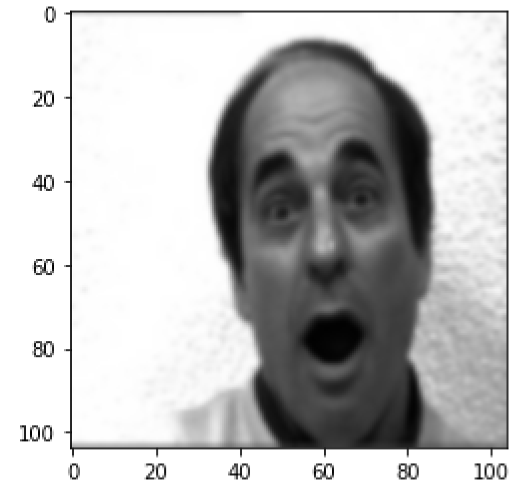


Real : 11
Predicted : 11

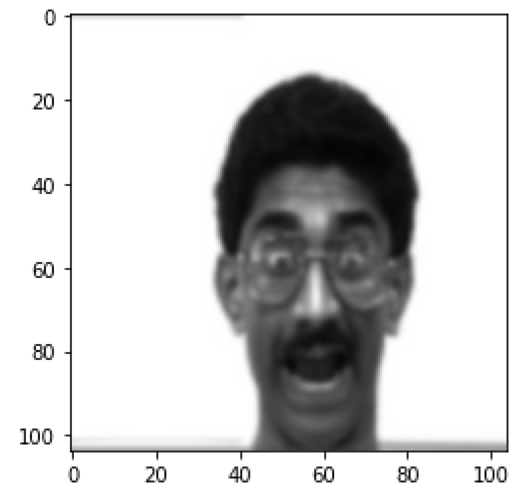


Real : 12
Predicted : 12

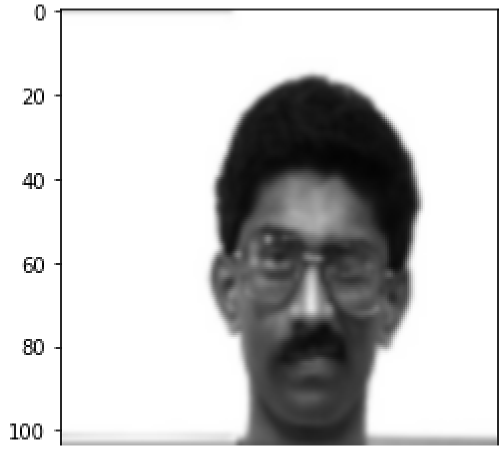

```
Real : 12
Predicted : 12
```



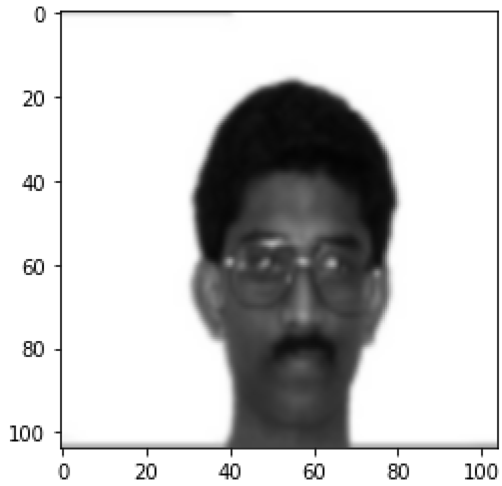

```
Real : 12
Predicted : 12
```



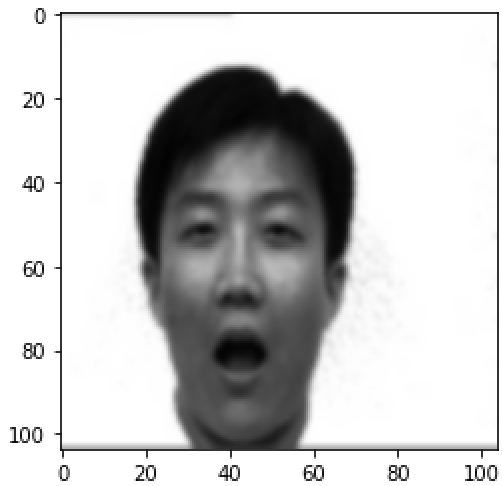

```
Real : 13
Predicted : 13
```



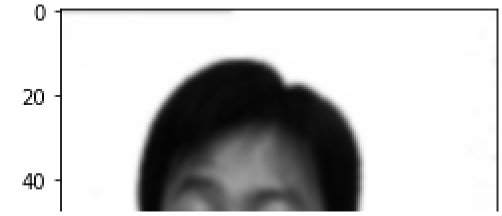
Real : 13
Predicted : 13



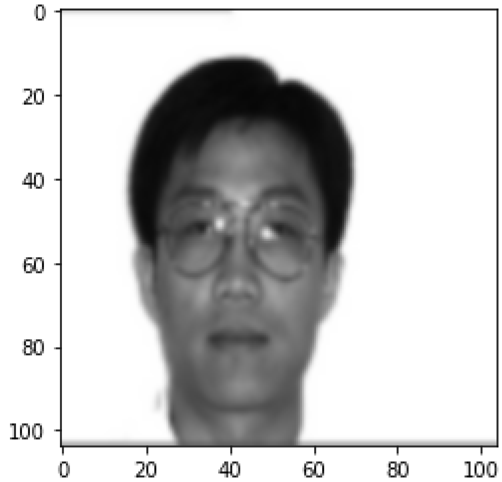
Real : 13
Predicted : 7



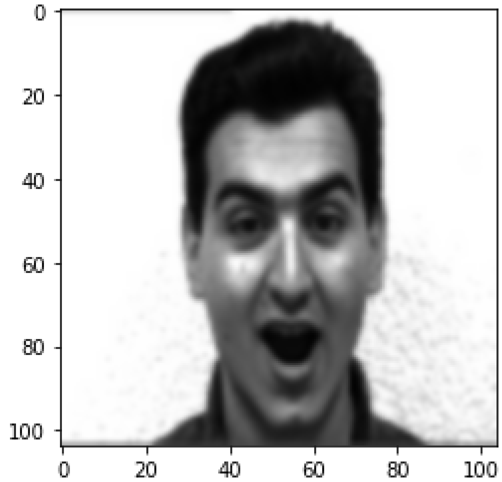
Real : 14
Predicted : 14



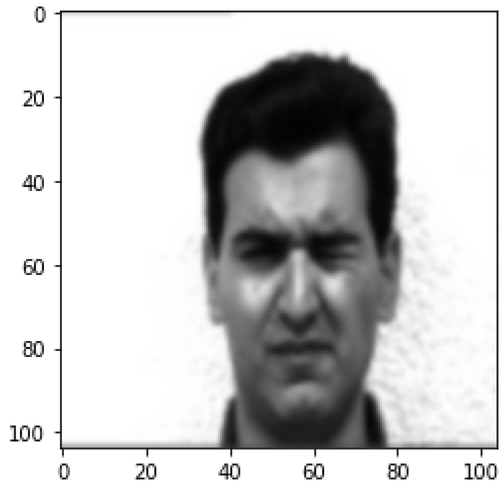
Real : 14
Predicted : 14



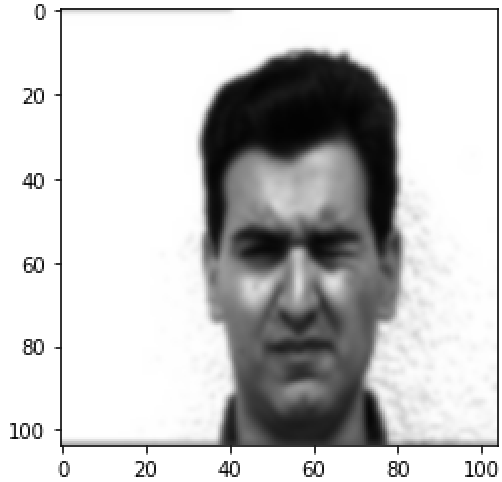
Real : 14
Predicted : 14



Real : 15
Predicted : 15



```
Real : 15
Predicted : 15
```




```
Real : 15
Predicted : 15
Accuracy :  86.66666666666667
```

References :
https://github.com/xanmolx/FaceDetectorUsingPCA/blob/master/PCA_Face_Recognition_IIT2016040.ipyn
(https://github.com/xanmolx/FaceDetectorUsingPCA/blob/master/PCA_Face_Recognition_IIT2016040.ipyı

https://www.youtube.com/watch?v=g4Urfno4aTc&t=1657s (https://www.youtube.com/watch?v=g4Urfno4aTc&t=1657s)

Discussion Collabarators : Kamlesh Sawadekar ; Unnat Dave

Ideas are influenced by the above sources, so the variable names or the steps might look similar but each step is being understood in written by me while reading the paper

In [ ]: