Aim: Implementation of classic Viola-Jones algorithm for detecting the faces and then classify the detected images using the concept of Eigen-faces

Requirements :
NumPy: version 1.19.1
Opencv-python: version 4.2.0.34

Part A: Detecting face in images using Haar Cascade Classifier.
Main Concept:
Haar-like features:
Below figure shows the basic haar features proposed by Viola and Jones. Further to speed up the feature extraction process, the concept of integral image is used.

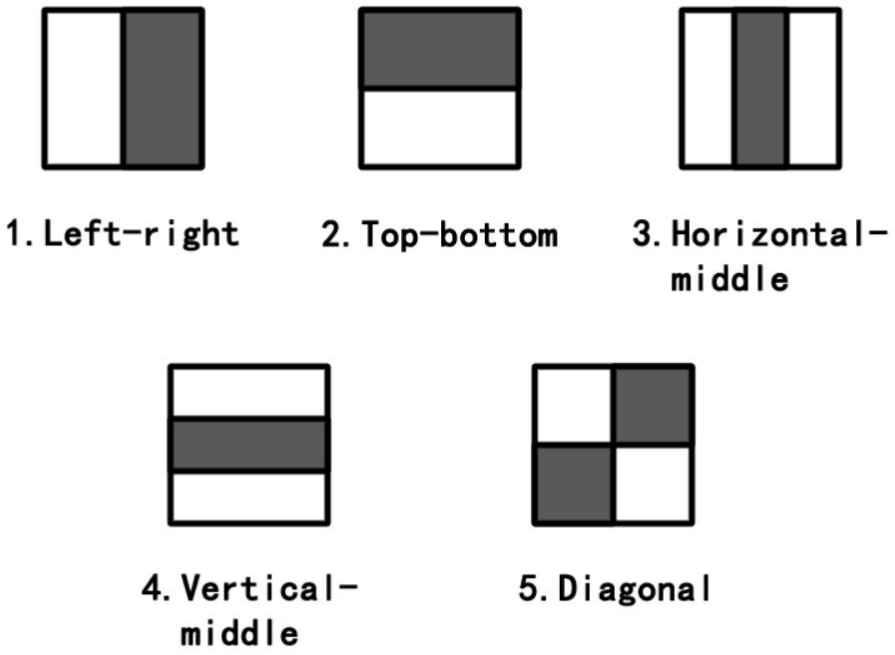


Image source: https://github.com/Donny-Hikari/Viola-Jones

Cascade classifiers:
Below figure shows the cascading of the classifiers. This is done to speed up the process as well as decrease the rate of false positives..

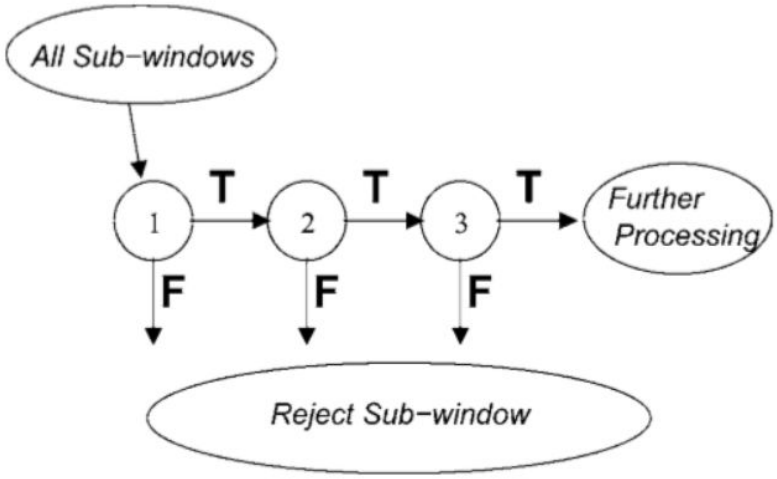


Image source: https://github.com/Donny-Hikari/Viola-Jones

Stepwise implementation in code:
Step 1: Prepare the dataset to be trained. Store positive images in “p” folder and negative images in “n” folder. Use this data for training and as the result, we get cascade.xml file. We will use this file to detect a face in images. Cascade.xml is prepared using Cascade-trainer-GUI.
The number of stages trained: 10
Step 2: Read the image and convert it to grayscale(takes lesser processing time).
Load the classifier with cascade.xml file as a parameter and create “face_cascade” object for detection.
Step 3: Detect faces using detectMultiScale() function and draw rectangle boxes on the obtained faces. We were expected to show results on the Yale database which contains 1 face per image. So, I **assumed** that the maximum one face is present in the image. So, if more than one rectangle is found in the image(say one over eyes and one over entire face), then the one with maximum area is chosen and drawn.
Important assumption: Maximum one face per image.

Part B: Recognizing faces in images using the concept of Eigenfaces.
Stepwise implementation in code along with concepts used:
Step 1: Read the images from the database and flatten them one by one.
By flattening I mean, an image of dimensions (N, M) gets flattened to ($N \times M$,1). Append all of these flattened images to an array A. A has dimensions of (NM, number of images).
In our case, the number of images is 50 and NM = 77760
So, dimensions of A are (77760, 50)
Step 2: There are 50 face vectors, on for each image. Calculate the average face vector and subtract it from each of the face vectors to make them normalised.

Step 3: Calculate the covariance matrix using the normalised face vectors.

$$Covariance\ matrix\ =\ AA^{T}$$

Step 4: Eigenvectors of the covariance matrix are the Eigen's faces and the eigenvalues of these eigenvectors act as weights to those eigenfaces.
Each of the eigenvectors acts as an eigenface and has a corresponding eigenvalue. Eigenface with maximum eigenvalue is considered as the best eigenface(has maximum features).
Step 5: Represent each of the database images as the linear combination of eigenfaces.

$$\mathbf{I_i = n_1F_1 + n_2F_2 + ...+n_mF_m}$$

Where I_i is an ith image written as a linear combination of eigenface vectors F_1 F_2….
And, m is the total number of images.
We store the coefficients n_1 n_2… in an array and append into weights array W.
Step 6: We will use these weights to represent images in smaller subspace(Dimensionality reduction- Using k best eigenvectors to represent images.)
Step 7: Load the test image and write it as a linear combination of best k Eigenfaces. Store the coefficients in an array say W_temp.
Step 8: Compare each of the arrays in W with W_temp. The one with the best matching is the recognized image. We find the minimum distance between training vectors and testing vectors.
Step 9: Define a threshold value for minimum distance, if minimum distance > threshold value then we say that image is not found in the database.

Results:
Accuracy analysis:
The number of testing images used:
Positive images: 50 (images which should be recognized)
Negative images: 5 (images which should not be recognized)

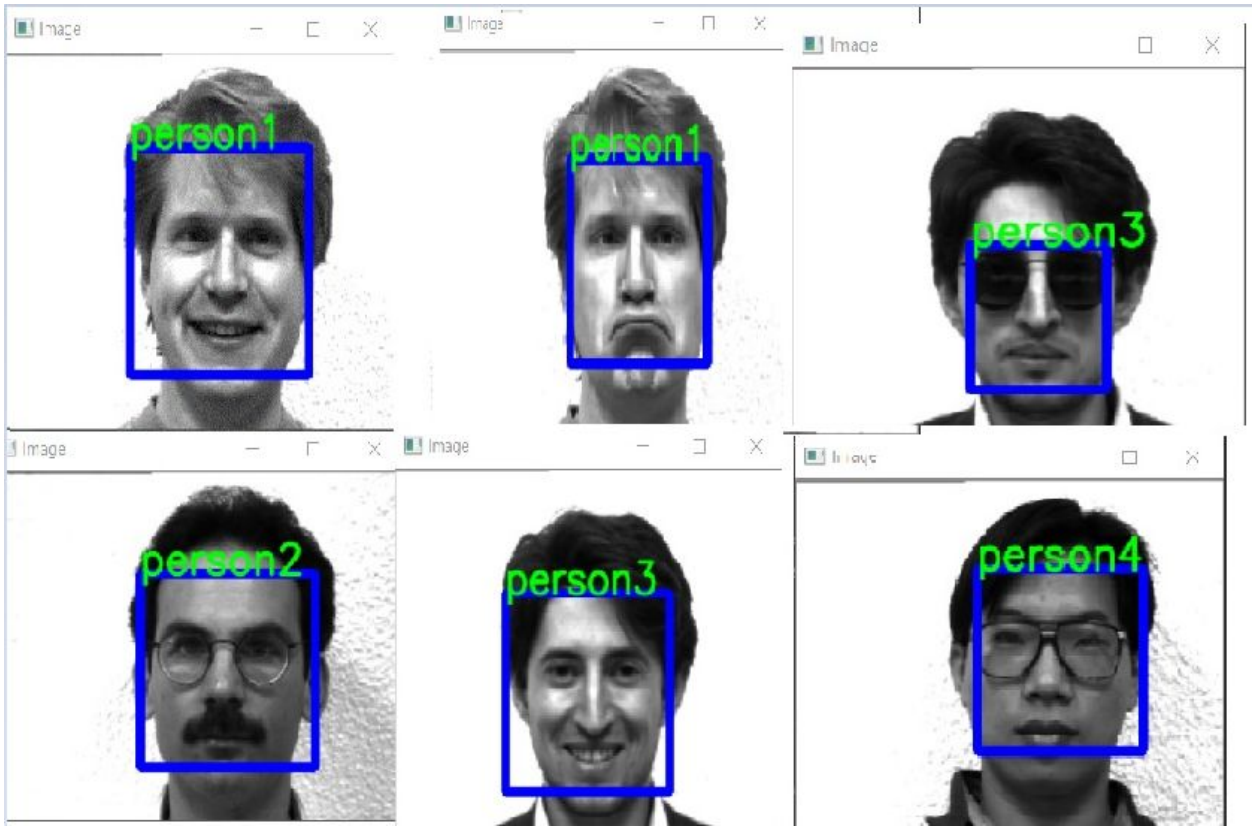
Accuracy = $\frac{Correctly\_matched}{Total\ number\ of\ images}$ * 100

For K = 5, Accuracy = 80.0
For K = 10, Accuracy = 85.45454545454545
For K = 15, Accuracy = 98.18181818181819
For K = 20, Accuracy = 98.18181818181819

Conclusion: As we consider more and more eigenfaces, the accuracy of the result increases.

Output images:
Testing of the algorithm was performed using 50 images of 10 people(5 each). Below images show the result. The face has been detected using the cascade.xml file and rectangle has been drawn on the face. Then, recognition of the face is performed using Eigenafaces concept.
Face recognition algorithm finds the best matching of test image in the database(if any) and then using the index_number of the file we find the person name.
For simplicity, the 1st person has been named Person1, 2ns as Person2 and so on.

The same result is obtained after running the code. For detailed output refer to the video that is the result of code output.



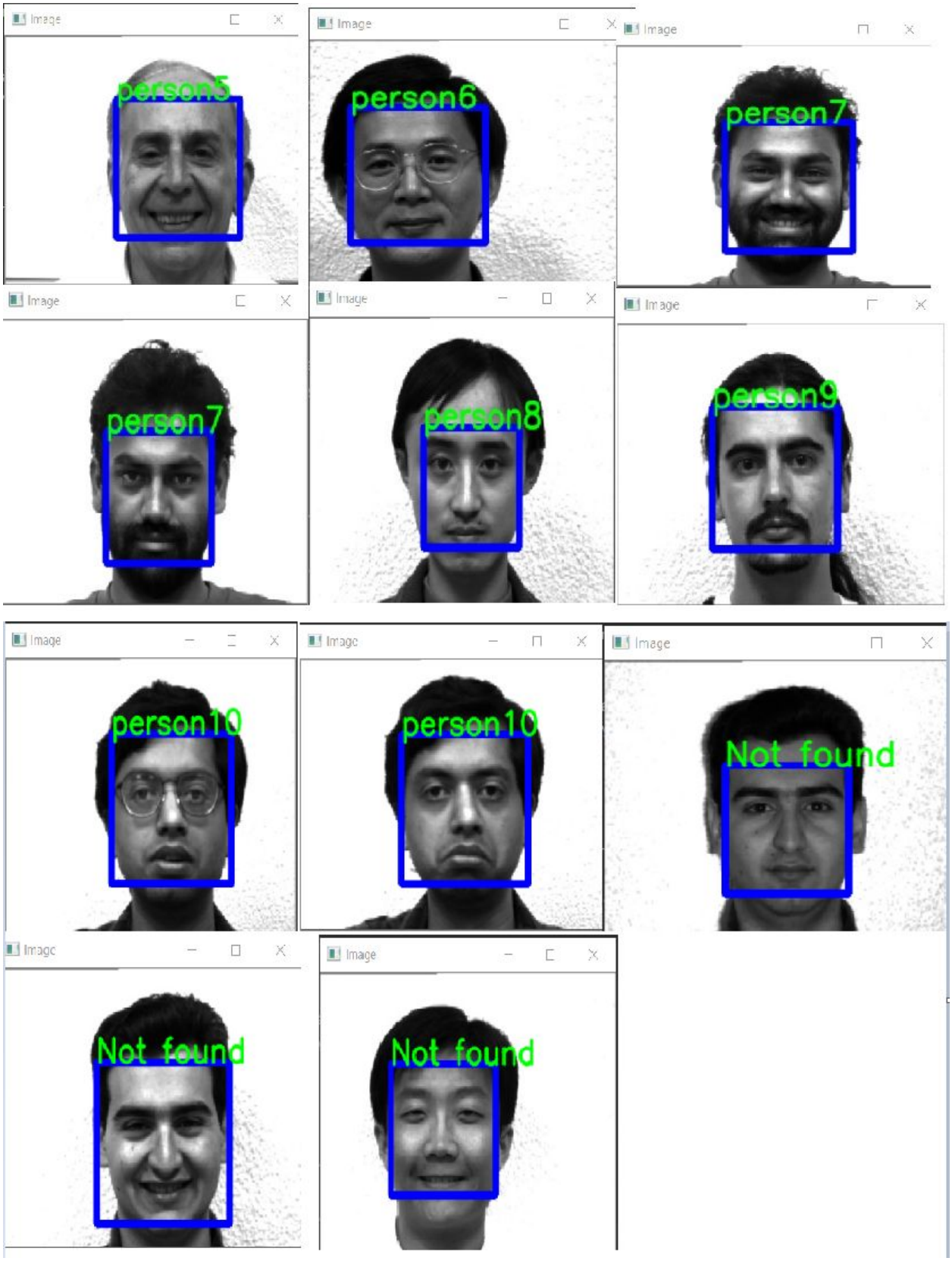


Image
person5
Image
person6
Image
person7
Image
person7
Image
person8
Image
person9
Image
person10
Image
person10
Image
Not found
Image
Not found
Image
Not found

References:

[1]https://github.com/ranriy/Face-Recognition-using-Eigenfaces
[2]https://www.geeksforgeeks.org/ml-face-recognition-using-eigenfaces-pca-algorithm/
[3]https://en.wikipedia.org/wiki/Eigenface
[4]https://github.com/xanmolx/FaceDetectorUsingPCA
[5]https://medium.com/@devalshah1619/face-recognition-using-eigenfaces-technique-f221d505d4f7
[6]https://www.youtube.com/watch?v=g4Urfno4aTc