Face Recognition using Eigen Faces: Report

Team Number: 11
Team Name: 21 din mein CG double
Team Members:

- Rutvij Menavlikar (2019111032)
- Suyash Vardhan Mathur (2019114006)
- Tejas Chaudhari (2019111013)
- Tushar Choudhary (2019111019)

Repository Link: https://github.com/Rutvij-1/Eigen-Faces-SMAI-M21-project

Problem Statement

Developing a computational model of face recognition is quite difficult, because faces are complex, multidimensional, and meaningful visual stimuli. Unlike most early visual functions, for which we may construct detailed models of retinal or striate activity, face recognition is a very high-level task for which computational approaches can currently only suggest broad constraints on the corresponding neural activity. Our aim is to develop a computational model of face recognition which is fast, reasonably simple, and accurate in constrained environments such as an office or a household.

Implemented PCA and got the Eigen Faces

- Flattened the image data in order to make manipulations on it as a vector.
- Performed Eigen Value Decomposition on the face matrix to get the Eigen Faces.
- Dimensionally reduced the face data using the top 20 Eigen Faces.

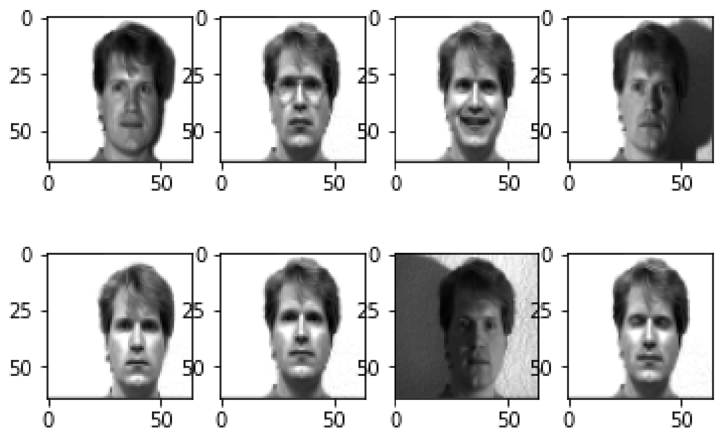
Implemented Face Recognition using Nearest Neighbour Classifier

- For any image from the testing dataset, we first reduce its dimensions with the transformation matrix calculated in PCA.

- Then with the coordinates of the projection of this image on face space, we find the image in the training dataset whose projection on face space is nearest to the projection of the test image, and assign the label of that image to the test image.

Dataset Used

- For the project, we use the Yale Face Database.
- The dataset consists of 165 grayscale images of 15 individuals (**classes**).
- There are 11 images per subject, one per different facial expression or configuration: center-light, w/glasses, happy, left-light, w/no glasses, normal, right-light, sad, sleepy, surprised, and wink.
- Following are a few sample face images of a single subject from the dataset:



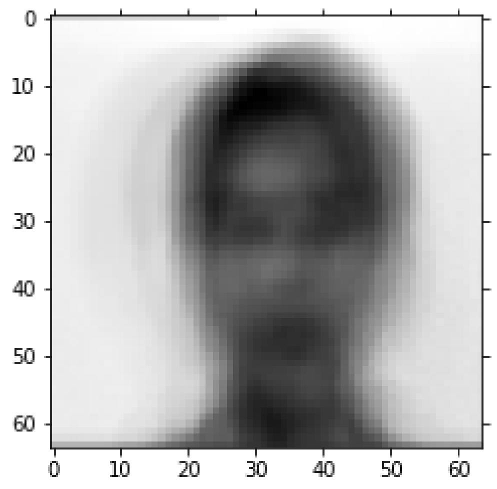
Procedure

- First, we load the images from the dataset and split them into training and testing data. The split is made such that 2 images of each of the 15 subjects go to testing data, while the others go to training data. Thus, **training data consists of 135 images while the testing data consists of 30 images**.
- From the face samples above, we can see that the facial features are aligned to a certain extent for all the faces. Further, we note that most of the faces here have similar lighting conditions and are aligned towards the centre.
- The variance of the faces is high only in certain directions. Thus, if we are able to find these directions, then we'll be able to recognise these faces with lesser dimensions. Therefore, we can use PCA for the task of dimensionality reduction.

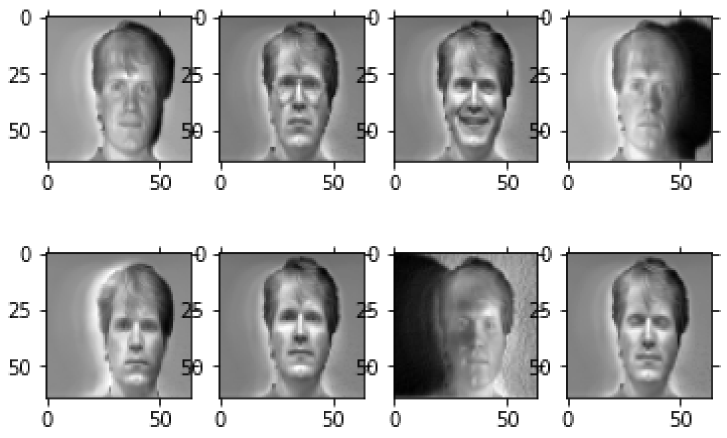
- For this, we stack all the images into a single column vector, calling it `training_tensor`. Considering we have M images of size N^2 then this training tensor has the dimension $M \times N^2$.
- In order to do PCA, we first need to normalize the images, for which we find out the mean face.
- Now, we find the mean/average face for all the faces as

  ```
  np.mean(training_tensor,axis=0)
  ```

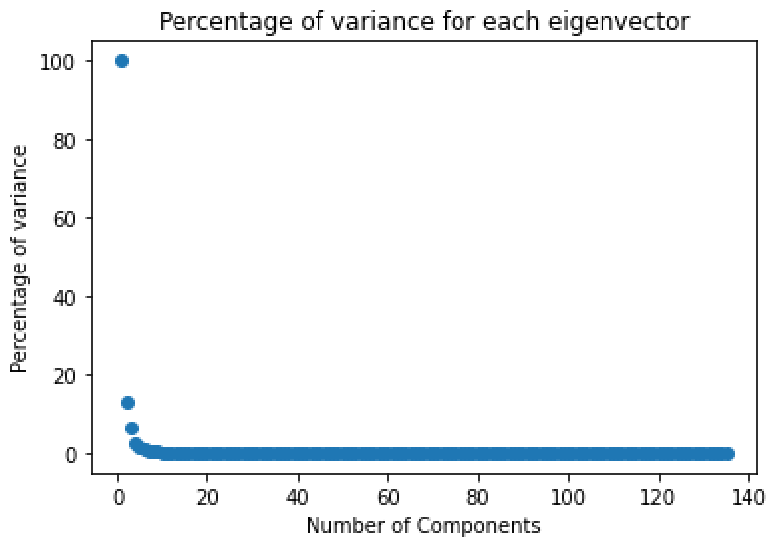
  and obtain the following mean face:



- Now, we normalize all the face images from our dataset using this mean face. These were normalized by taking the difference from the mean face, i.e. $\Gamma = \Gamma - \mu$, where μ represents the mean face. The normalized faces can be seen below:

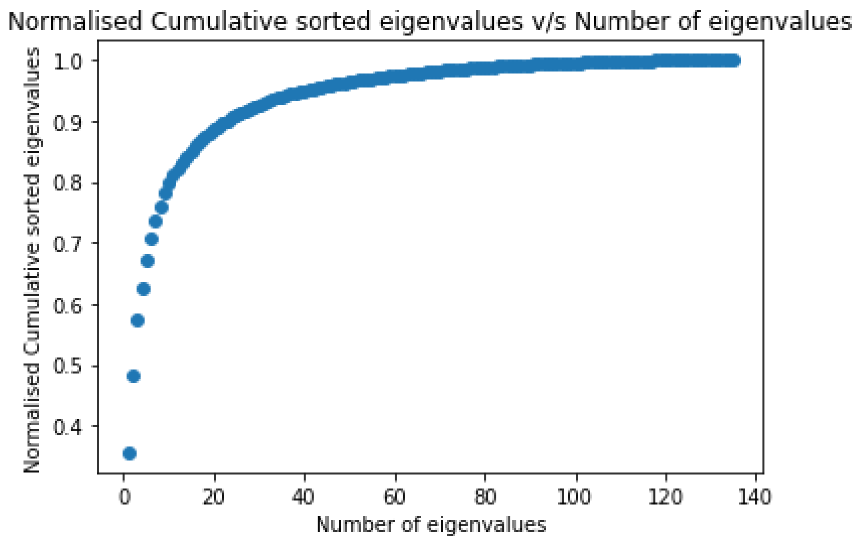


- Note that the background appears darker in the normalized images now.
- To do implement PCA, we have to calculate the eigenvectors. This should ideally be done by finding the covariance matrix $C = training\_tensor^T \times training\_tensor$. However, this covariance matrix will here have a dimension of $N^2 \times N^2$. This is not computationally efficient to calculate and we can do better. So instead we calculate our covariance matrix $C' = training\_tensor \times training\_tensor^T$, which will have dimensions of $M \times M$. It can be proved that both these covariance matrices have the same eigenvalues and their eigenvectors can be related by the relation $eigenvec(C) = training\_tensor^T \times eigenvec(C')$. Proof for this can be found here.
- Next, we sort the eigenfaces based upon their respective eigenvalues, since the greater the eigenvalue, the more is the contribution of the eigenvector to representing the set of faces. We have analyzed the variance of the data set on these eigenfaces using the plots shown below:



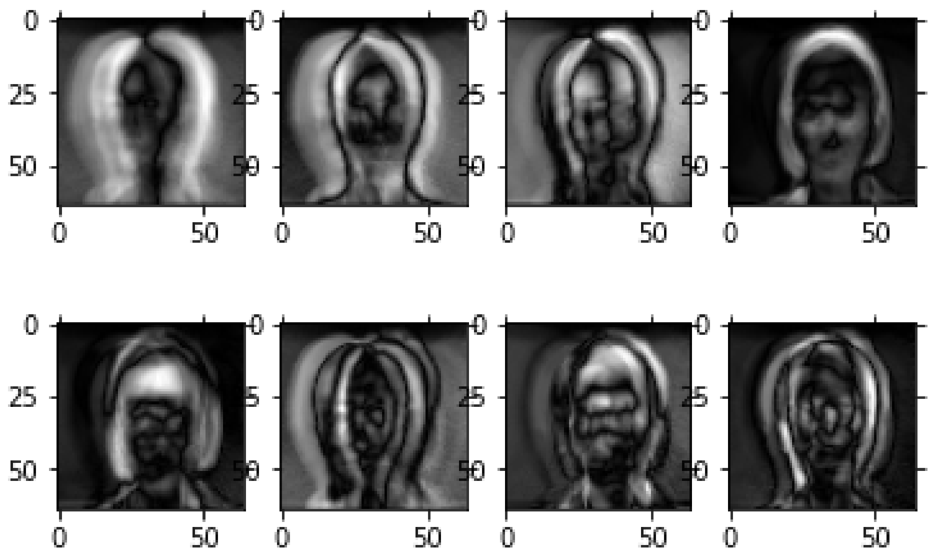
In this plot we see that the variance along the starting eigenfaces are high and it decreases as we move towards the eigenfaces with low eigenvalue.

- In the next plot we have plotted the Normalized Cumulative Sum of Eigenvectors against the sorted eigenfaces:



Thus, we can see that taking around 20 eigenfaces gives us that value of the Normalized Cumulative Sum close to 0.9. Hence, reducing them to 20 components should be sufficient.

- We have hence taken the first 20 eigenfaces $A' = \begin{bmatrix} \phi'_1 & \phi'_2 & \dots & \phi'_{M'} \end{bmatrix}$ where $M' = 20$ and ϕ'_i is the ith eigenface, after sorting them. The first 8 eigenfaces have been visualized below:



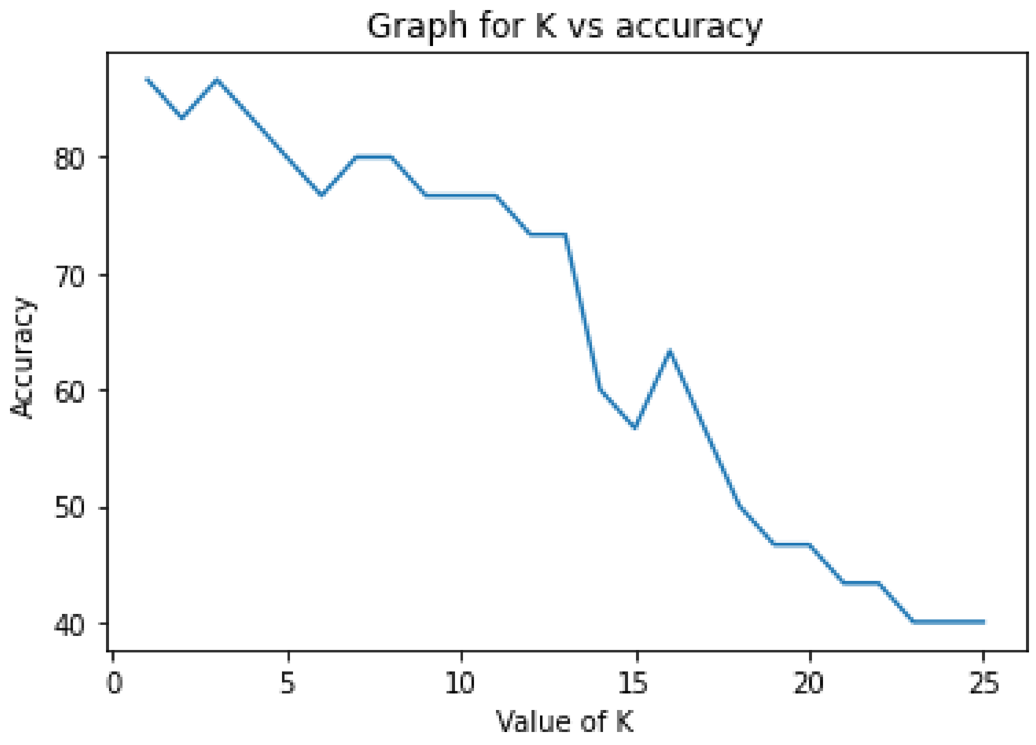
- Now, we obtain dimension reduction matrix $proj\_data$ such that $w_i = proj\_data \times \phi'_i$ gives the dimensionally reduced eigenface.

$A = \begin{bmatrix} I_1 \\ I_2 \\ . \\ . \\ I_M \end{bmatrix}$ where I_i is the i^{th} image flattened. Then, $proj\_data = (A^T \times A')^T = A'^T \times A$.

- Then, for each image I which is flattened into a vector I_v, we take its projection on face space by taking $T_v \times proj\_data$ where T_v is the normalised image vector I_v.
- Now, we obtain the weights for all the training images by taking the dot product of each normalized image matrix with the $proj\_data$ matrix.
- Now, for recognizing the unknown faces in the testing phase, we normalize the testing face using the mean face, and then find the weights associated with the testing image by taking the dot product with the $proj\_data$ matrix.
- Now, we find the closest training weight to the testing image's weights in terms of Euclidean distance. Here, we use the K-nearest neighbour classifier to determine the final label to be allotted to the testing image.
- Further, if the minimum euclidean distance exceeds a certain threshold t_0, then we can say that the face doesn't belong to the dataset, and if it exceeds another larger threshold t_1, then we can say that the image is not a face.

Experimenting with K-neighbours

As mentioned earlier, we have applied K-nearest neighbours to classify a test image. We tested with different K to check what value of K gives the highest accuracy on test data. The plot between K and accuracy is as follows:

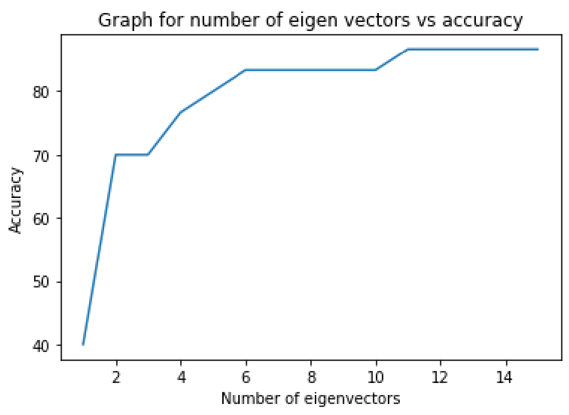


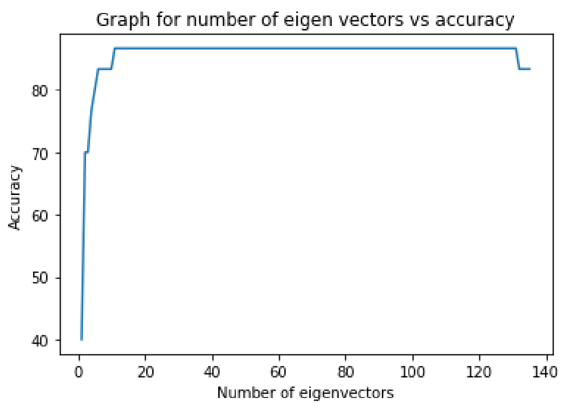
From this graph, we can see that accuracy is highest for K=1 and K=3 (86.66% for both). As the K increases to 10, 20, ... the accuracy keeps decreasing. This can be explained as the number of training images for each person is just 9, and if we increase the value of K beyond comparable numbers, the value sharply decreases due to inevitable interference from other incorrect classes.

This explains why the Accuracy peaks with K=3, and sharply falls after it.

Experimenting with number of eigenvectors

When we implemented PCA earlier, we saw that roughly 20 eigenfaces seemed sufficient to measure the majority of the variance. We implemented the algorithm using a different number of eigenvectors and plotted a graph between the number of eigenvectors and the accuracy on the test set. Here we see that the graph saturates after 11 eigenvectors (reaches 86.66% accuracy). An important thing to note here is that we get the same accuracy even with 135 eigenvectors. This explains that we will need much more features to correctly classify the remaining images. However, only 11 eigenvectors are sufficient if we can work with this accuracy.



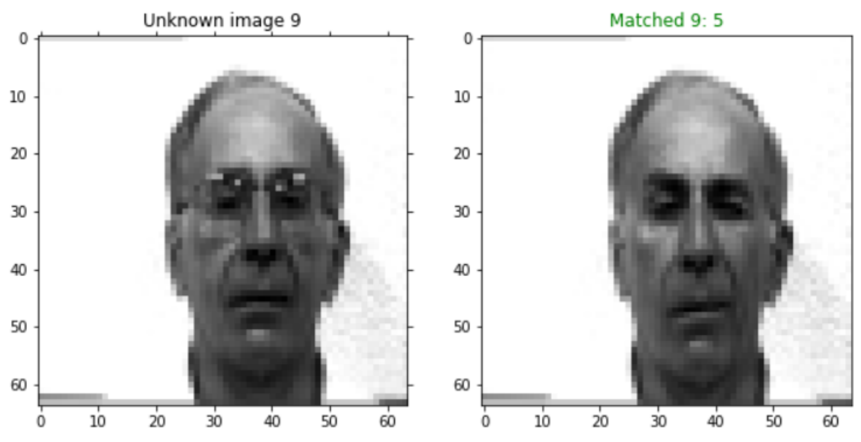


Face Detection and Recognition

The highest accuracy that we got for face recognition over our testing dataset of 30 images was 86.66%. The recognition outputs could be of 3 types based upon the Euclidean distance found using the weights - Face recognized with a match in the dataset, Face detected but not recognised, and Face not detected. This is done by finding the closest training weight to the testing image's weights in terms of Euclidean distance and using the K-nearest classifier to find the appropriate label. Further, if the minimum euclidean distance exceeds a certain threshold t_0, then we can say that the face doesn't belong to the dataset, and if it exceeds another larger threshold t_1, then we can say that the image is not a face.

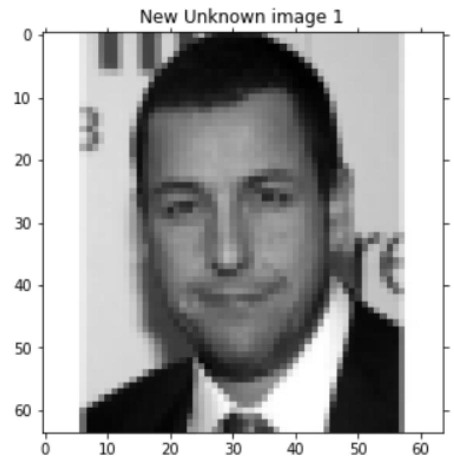
Below, we can see a correctly recognised face:

```
Detected image from training images index 44 and distance value 1762407.5204594964
Face recognised and found a match in the dataset

The prediction is correct
```



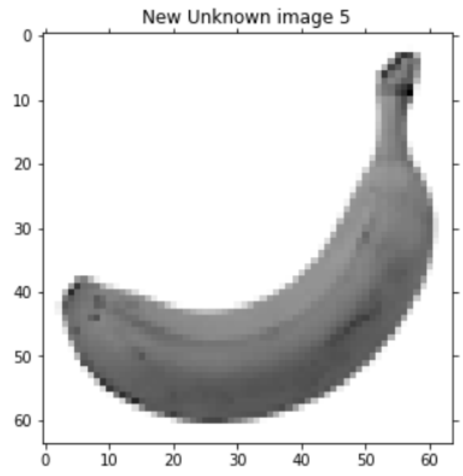
Also, the following is an unknown face that was detected by the model:

```
Detected image from training images index 14 and distance value 53890902.467777275
Face detected, but did not find it in the dataset

The prediction is correct
```



Below is an example of an unknown object's image, which wasn't detected as a face:

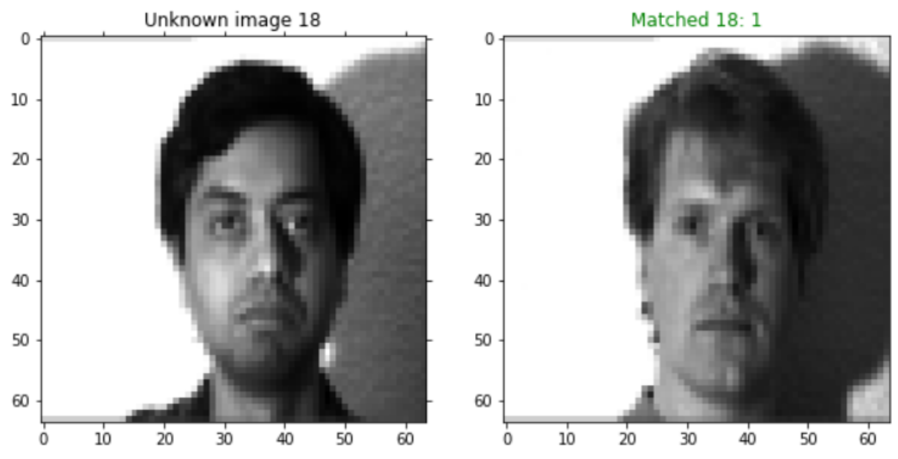
```
Detected image from training images index 42 and distance value 55457249.54113859
Face not detected

The prediction is correct
```



Analyzing the Incorrect Cases

We know that one disadvantage of the eigenfaces method is that it is sensitive to lighting, shadows and also the scale of the face in the image. In this case, one possible explanation as to why these two images are a match is that both the images have a similar shadow. And hence, the shadow and the lighting on the face may be getting highlighted rather than the facial structures of the two unseen images.

```
Detected image from training images index 3 and distance value 23802670.33221401
Face recognised and found a match in the dataset

The prediction is incorrect
```

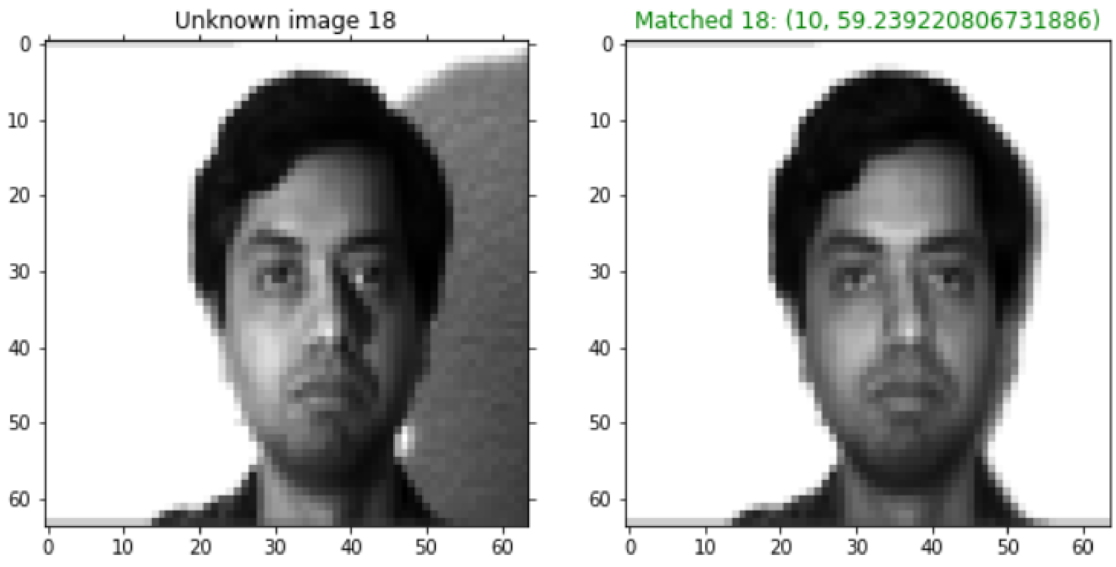


Comparison with other methods

For setting a benchmark to compare the performance of our model with, we use the built-in `LBPHFaceRecognizer` to train a model. This classifier takes into account the neighbouring pixels in a certain radius, and thresholds them to consider the results as binary numbers. The model was trained as below, using the same training data used in the EigenFaces model:

```
# Training the model
model = cv2.face.LBPHFaceRecognizer_create();
model.train(training_im, training_label)
```

Using the same testing data as the EigenFaces model, we get an accuracy of **90.0%** with the built-in module. This shows that our model gives comparable performance to inbuilt modules, with the inbuilt module just getting one more face recognition right out of the 30 in the testing data (4 of the misclassified images here were misclassified by eigenfaces approach as well).



Above is the only image that was corrected by the LBPH Face Recognizer. Clearly this is the case where eigenfaces was making comparisons based on the shadow and lighting in the image. The LBPH classifier does not suffer from this problem and thus, can be seen as an improvement over the eigenfaces classifier.

Points to note

Based on the above experiments and observations, we can say the advantages and limitations of the eigenfaces method as:

- Advantages:
    - Easy to implement and computationally less expensive.
    - No knowledge (such as facial features) of the image required (except id).
- Limitations:
    - A proper centred face is required for training/testing.
    - The algorithm is sensitive to lighting, shadows and also the scale of face in the image .
    - A front view of the face is required for this algorithm to work properly.

Work Distribution

- **Rutvij**: Code for generation of eigenfaces and dimension reduction, Illustrative graph plotting.

- **Suyash**: Documentation, code for face recognition, comparing with inbuilt library, Illustrative graph plotting.
- **Tejas**: Documentation, Code commenting and cleaning, Dataset Analysis, hyperparameter tuning, Illustrative graph plotting.
- **Tushar**: Documentation, code for face recognition, K-Nearest Neighbour Classifier implementation, Illustrative graph plotting.

Everyone devoted time and contributed equally to the project.