Face Recognition using Eigenface

Author : Abhinav Pandey

1.1. Why is alignment necessary for Eigen-face?

Eigenfaces refers to an appearance-based approach to face recognition that seeks to capture the variation in a collection of face images and use this information to encode and compare images of individual faces. [2] The eigenface approach for face detection relies on pixel to pixel similarity for dimensionality reduction and feature extraction. Therefore, it performs best when the eyes, nose, lips, etc. in all images are approximately in the same pixel region and have the same orientation.

1.2. Facial Recognition using eigenfaces can be done by following a series of steps :

1. Read and store training in images in a 2D big data matrix. (Each image is represented as a single data point)
   - The dataset contains a total of 135 images of 15 individuals, consisting of 9 images of each of these individuals.

```
# List of filepaths for all training images
im_filepaths = glob.glob("trainingset/*")
# Add all image matrices to a numpy array to create a big data matrix of images
training_images = np.array([plt.imread(image).flatten() for image in im_filepaths])
print(training_images.shape)
>> (135, 45045)
```

Fig. 1. Sample images from the training data

2. Normalizing the data and performing dimensionality reduction (using Principal Component Analysis)

The steps involved are :

- Determine the **mean face** from all training images
- **Normalise the data** by removing common features (i.e. subtracting the mean face from training images)

- Computing a **variance-covariance** matrix using each of the 135 training images
- Computing **eigenvalues and eigenvectors** from the above variance-covariance matrix.

The implementation of these steps are as follows :

- Determine the **mean face** from all training images

```
# Determine the average of all the training face images
avg_face = training_images.mean(axis=0)
```

Fig. 2 Mean Face from Training Images

- **Normalise the data** by removing common features (i.e. subtracting the mean face from training images)

```
# Normalise the training data by subtracting the mean image from each one
normalised_training_images = training_images - avg_face
```

Fig. 3. Training Images after Normalising

1.2.2. Find a faster way to compute eigen values and vectors, explain the reason?

Each training image has a size of **231 x 195** = 45045. To determine the eigenvectors, we would require computing a 45045 x 45045 matrix and 45045 eigenvectors, which is computationally expensive.

A faster way to compute these eigenvectors is to first compute the variance-covariance matrix for all the 135 images and then determine the eigenvalues and eigenvectors of this matrix. Although, these eigenvectors are not of size 45045, they have a direct relationship with the 45045 size eigenvectors and thus can be used in their place to perform dimensionality reduction.

- Computing the **variance-covariance** matrix using each of the 135 training images

```
# Compute the covariance matrix for the normalised images
cov_matrix = np.cov(normalised_training_images)
print(cov_matrix.shape)
>> (135, 135)
```

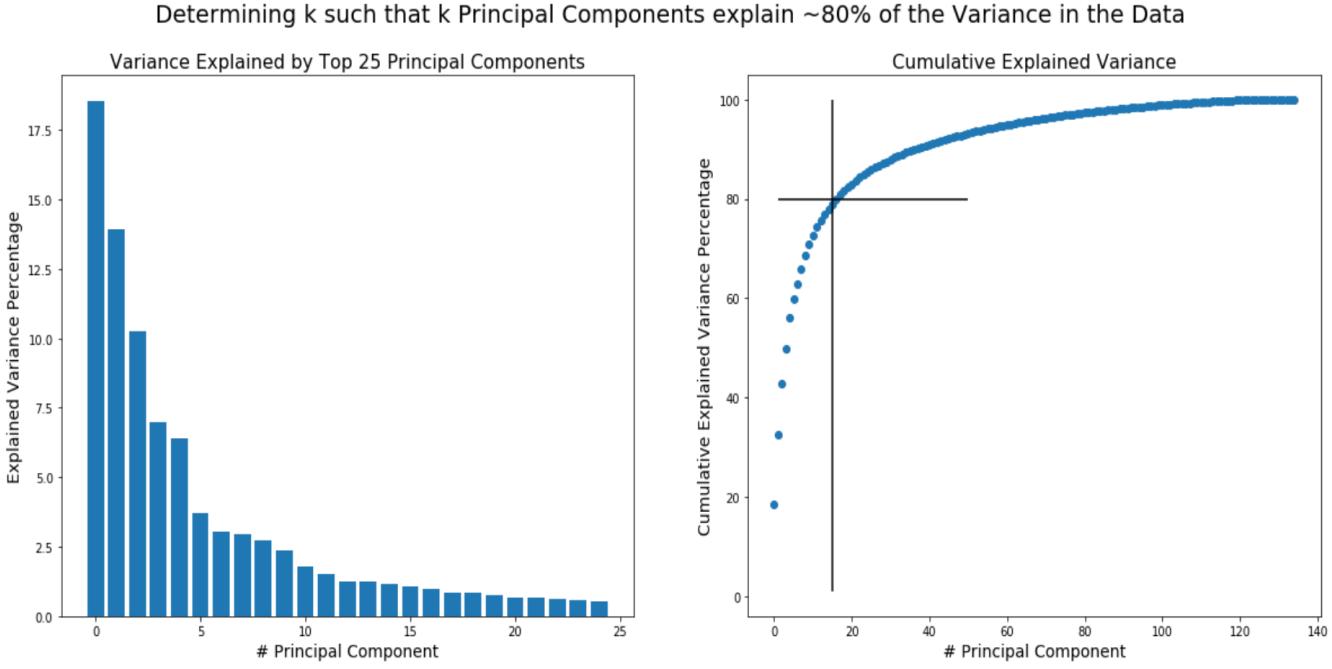
- Computing **eigenvalues and eigenvectors** from the above variance-covariance matrix.

```
# Calculate the eigen-values and eigen-vectors of the covariance matrix
eig_values, eig_vectors = np.linalg.eig(cov_matrix)
# Sort eigenvalues and eigenvectors (in descending order of eigenvalues)
desc_sorted_index = eig_values.argsort()[::-1]
eig_values = eig_values[desc_sorted_index]  # Sorted eigenvalues
eig_vectors = eig_vectors[:, desc_sorted_index] # Corresponding eigenvectors
```

3. Determine the top k principal components and visualize the top-k eigen- faces.

Here, I have selected **k** = 15. As we can see in the figure below, approximately 80% of the variance in the training data can be explained using the first **15 Principal Components** (eigenvectors), which is an acceptable threshold for information retention during dimensionality reduction. In other words, these 15 Principal Components alone capture 80% of the variance in the data. Therefore, we can use just these top 15 features for our Face Detection System and get the approximately the same recognition accuracy, with the added advantage of faster computation.

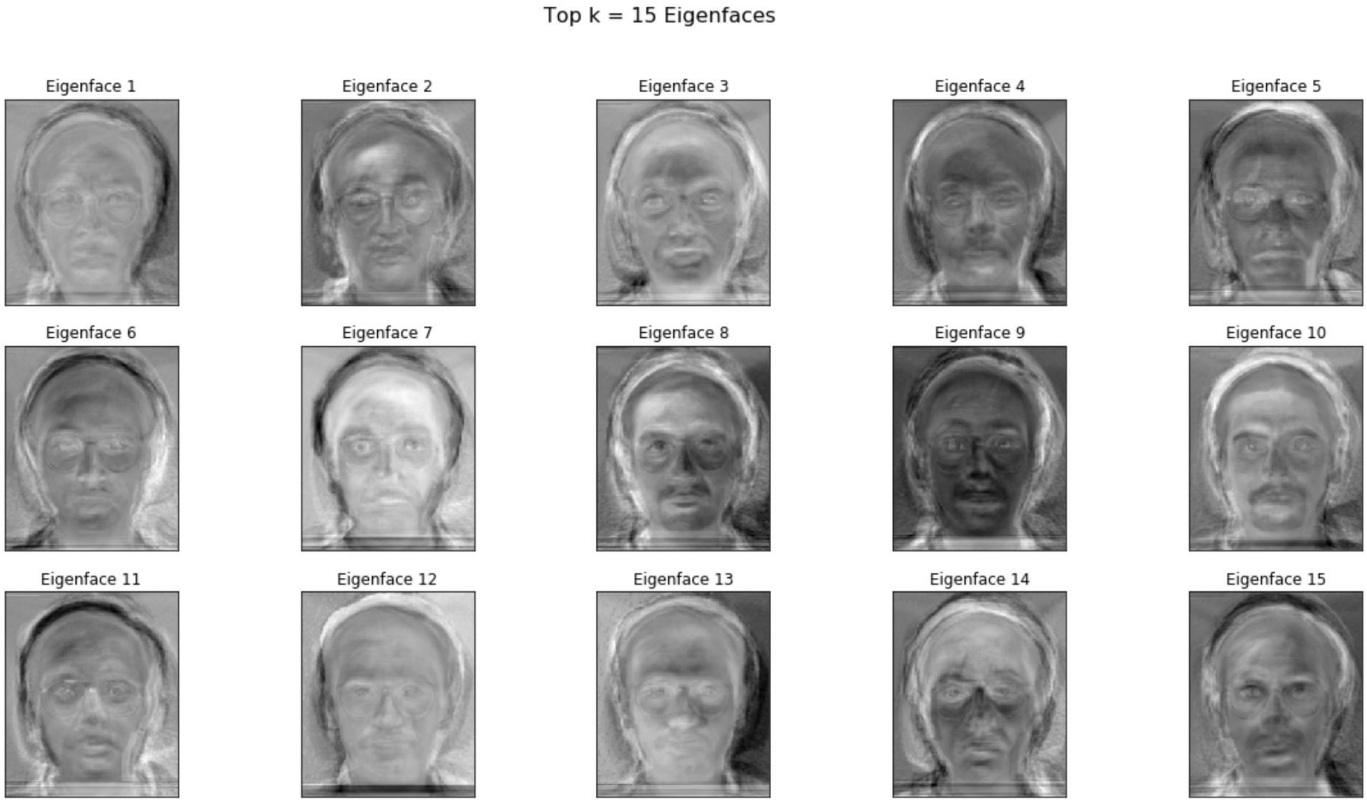
Fig. 4



The top k = 15 eigenfaces can computed by computing the projection of the normalised training images on the basis spanned by the first 15 Principal Components. The code is as follows :

```
# Assign k, where k is the number of Principal Components

k = 15

# Select the first k Principal Components (eigen-vectors)

k_components = eig_vectors[:k, :]

# Determine the eigen-faces using the first k Principal Components

eigen_faces = np.dot(k_components, normalised_training_images)
```

Fig. 5



The final step of the training process is to represent our training data as the weighted sum of the top k =15 eigenfaces. This enables us to create a *model*, which can be used to recognize unseen face images

```
# Project eigen-faces onto the subspace spanned by eigen-vectors

model_weights = np.dot(normalised_training_images, eigen_faces.T)
```

4. **For each of the 10 test images in Yale-Face find out which three face (training) images are the most similar.**
    - **Show these top 3 faces next to the test image.**

- **Analyse the recognition accuracy.**

Using the above model, we can determine which images in the training data have the most resemblance to each test image. We can do this by :

I. calculating the Euclidean distance of the normalised test image projections to the normalised training image projections on the basis spanned by the **k** Principal Components.
II. Identifying the 3 closest training images for each test image

The code for testing looks like this -

```
# Vectorise test images
test_images = np.array([plt.imread(image).flatten() for image in test_data_filepaths])
# Normalise the test data by removing the mean image features
normalised_test_images = test_images - avg_face
# Project test images on the k Principal Component subspace
test_weights = normalised_test_images @ eigen_faces.T
# Calculate distance of all test images from each training image in the new subspace
dist_from_train_images = np.linalg.norm(test_weights[:,None] - model_weights, axis=2)
# Determine the indexes of 3 "most similar" training images (those having least distances from each test image)
test_preds = dist_from_train_images.argsort(axis=1)
test_preds = [test_preds[i][:3] for i in range(len(test_preds))]
```

The figure (APPENDIX Figure 1(a) and (b) for prediction results can be found at the end of the report.

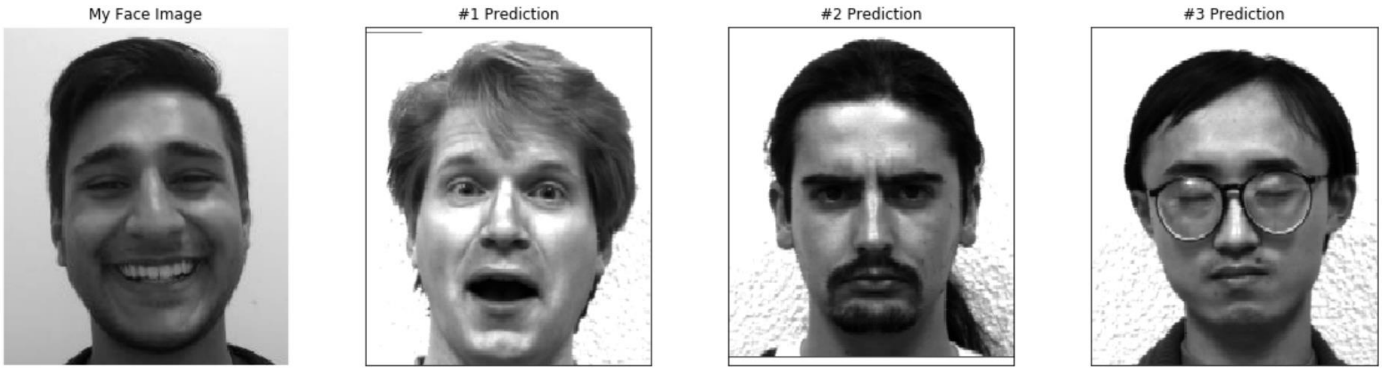
Table. 1. Overview of the recognition accuracy

	# Correct Predictions	# Incorrect Predictions	# Total	Accuracy
1st Prediction	10	0	10	100 %
2nd Prediction	9	1	10	90 %
3rd Prediction	9	1	10	90 %
OVERALL	**28**	**2**	**30**	**93.34 %**

5. **Read in one of your own frontal face images. Then run your face recognizer on this new image. Display the top 3 faces in the training folder that are most similar to your own face.**

The image below shows the predictions for my own face. We can see that the results are unsatisfactory, which is not surprising since the predictions can only be face images which were included during the training of the face recognition system.

Fig. 6. Predictions for new face image (when system is not trained on similar images)



6. **Repeat the previous experiment by pre-adding the other 9 of your face images into the training. Display the top 3 faces that are the closest to your face.**

Steps 1 to 5 were exactly repeated on an updated dataset (containing 9 new images).

<u>The figure (APPENDIX Figure 2(a) and (b)) for prediction results can be found at the end of the report.</u>

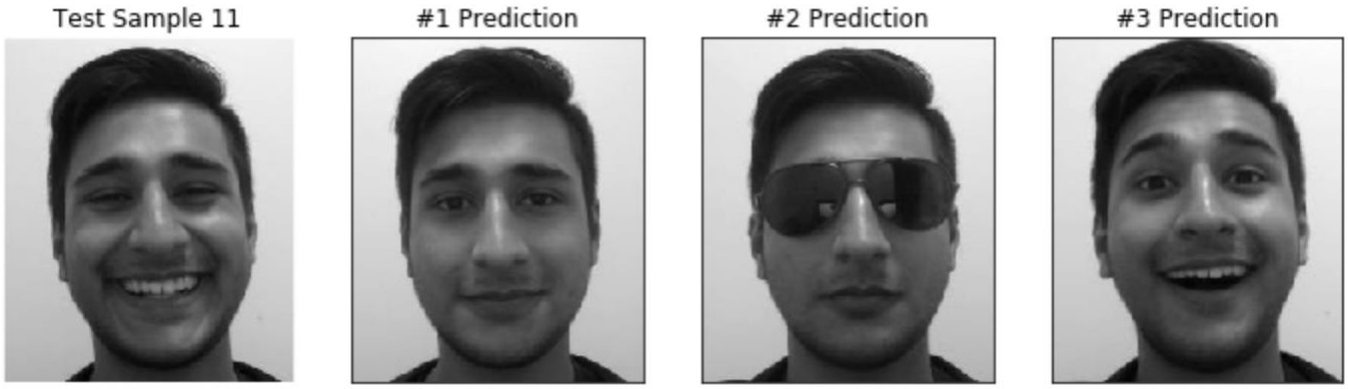
Table 2 contains an overview of the performance of the new face detection system.

Table. 2. Overview of the recognition accuracy (with UPDATED training set)

	# Correct Predictions	# Incorrect Predictions	# Total	Accuracy
1st Prediction	11	0	11	100 %
2nd Prediction	10	1	11	90.91 %
3rd Prediction	9	2	11	81.81 %
OVERALL	**30**	**3**	**33**	**90.91 %**

This new system is correctly able to recognise the new face image with 100% accuracy (for top 3 predictions).

Fig. 7. Predictions for new face image (when system is trained on similar images)



Observations :

- It is interesting to note that the overall recognition accuracy is worse for this new system compared to earlier version. One probable reason for this could be incorrect alignment of the new images compared to the previously existing images in the training set.
- Both the systems provide incorrect 2nd and 3rd predictions for Test Sample 5. The loss of information during PCA might be causing this failure. This is a scenario where an individual has to weigh the trade-off between reduced dimensionality (hence, faster computations) and information loss.

References -

1. Sheng Zhang and Matthew Turk (2008) Eigenfaces. Scholarpedia, 3(9):4244.

APPENDIX

Figure 1. a) Face Recognition Results

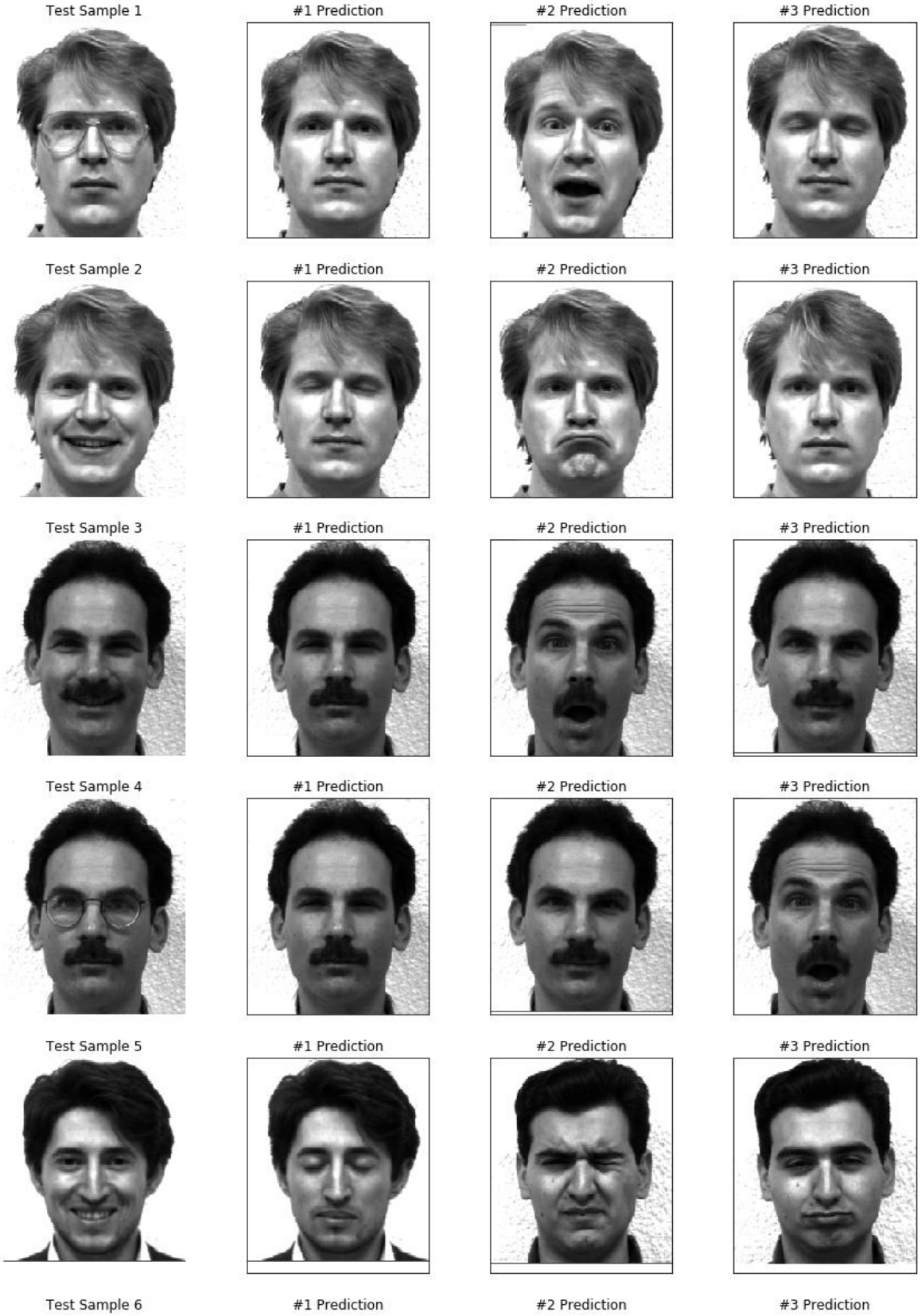


Figure 1. b) Face Recognition Results

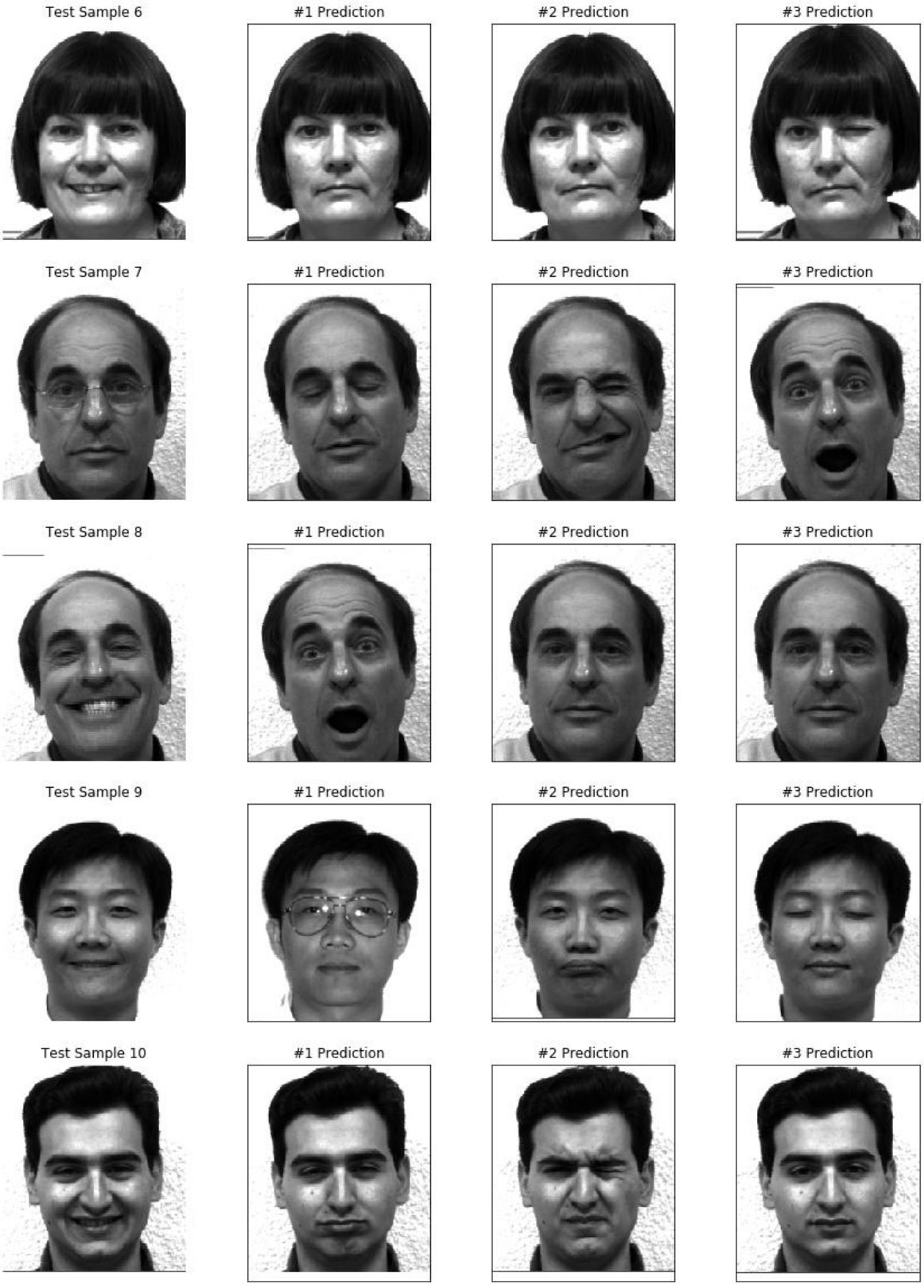


Figure 2. a) Face Recognition Results (with updated Training data)

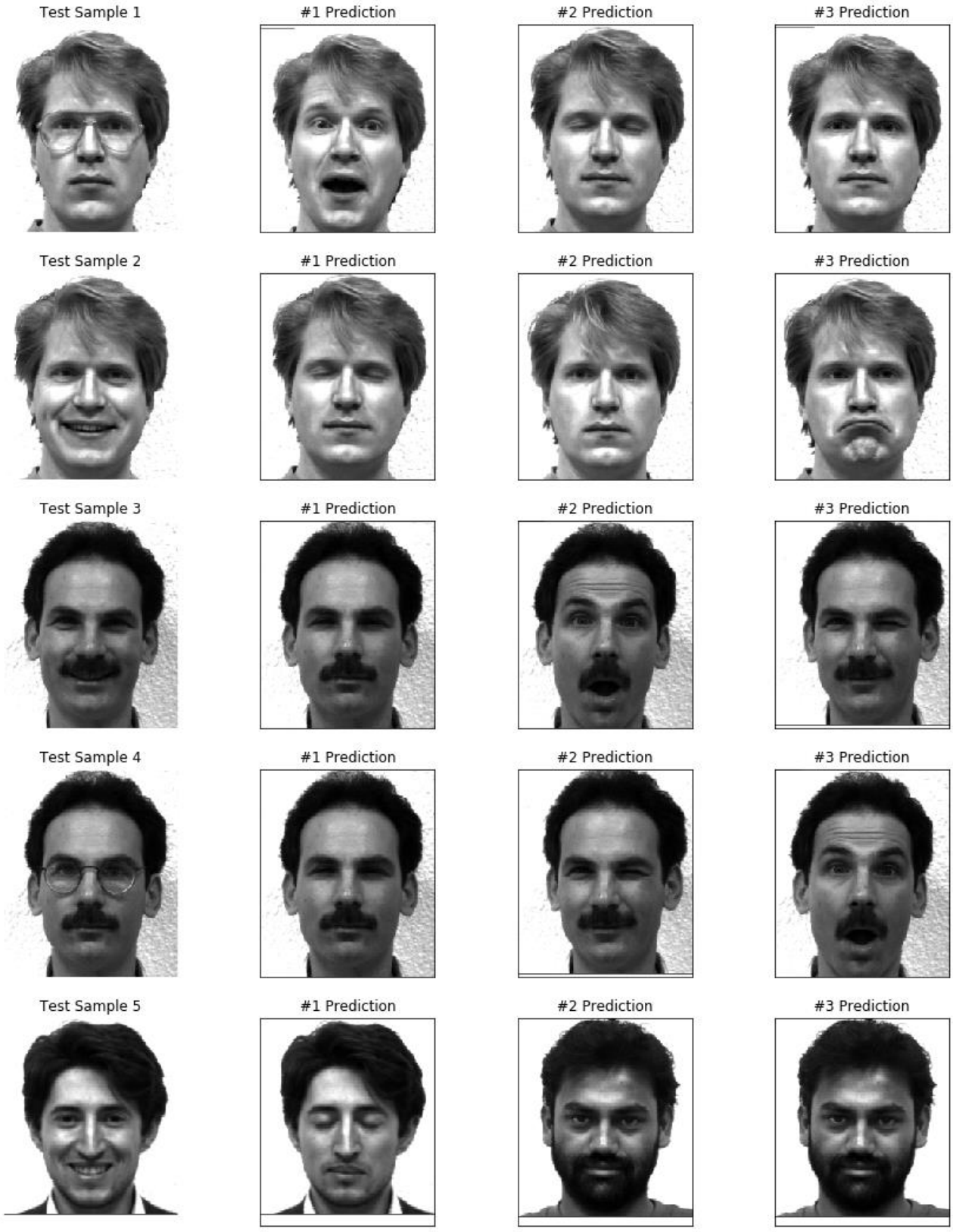
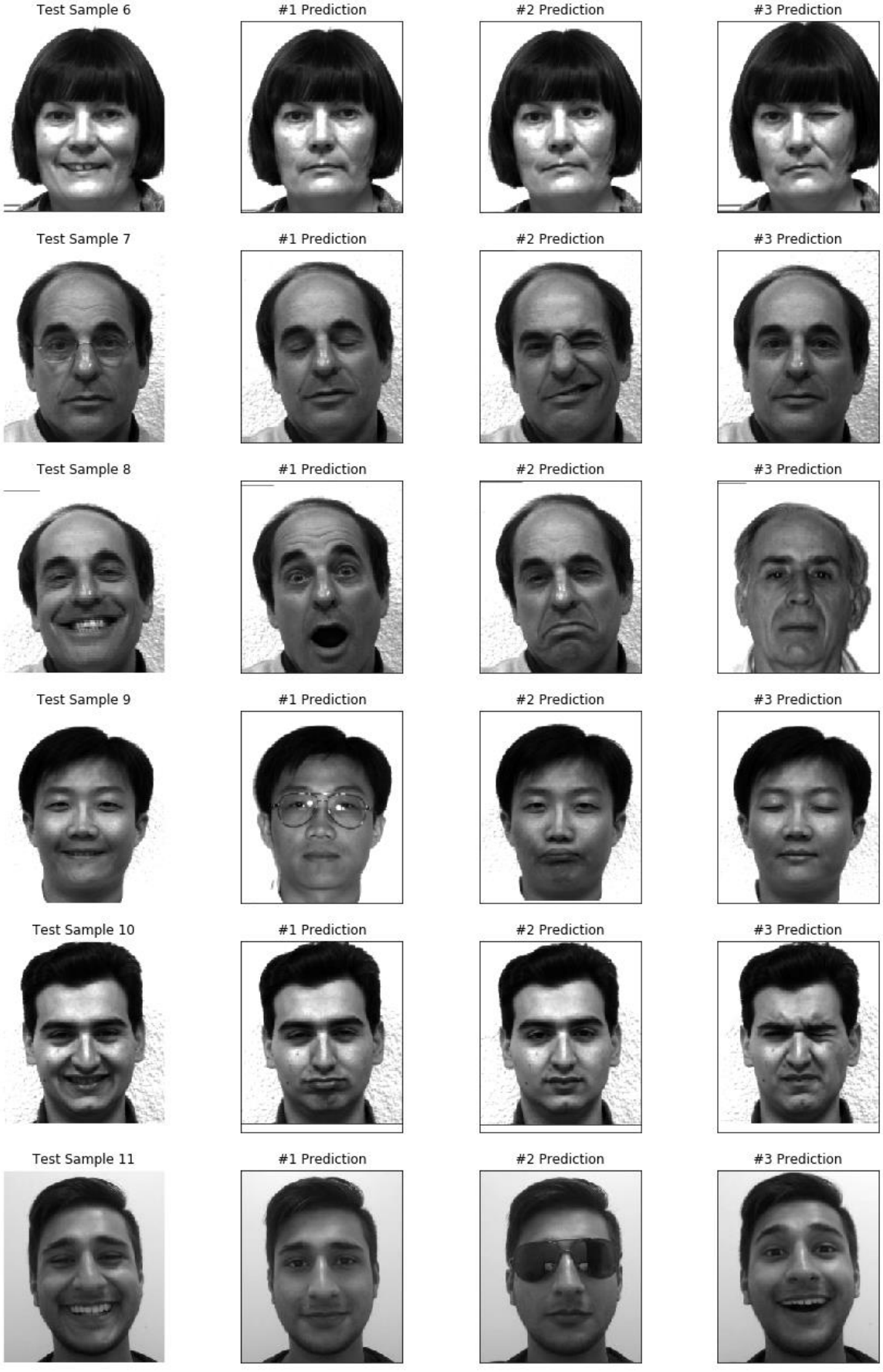


Figure 2. b) Face Recognition Results (with updated Training data)


Test Sample 6
#1 Prediction
#2 Prediction
#3 Prediction
Test Sample 7
#1 Prediction
#2 Prediction
#3 Prediction
Test Sample 8
#1 Prediction
#2 Prediction
#3 Prediction
Test Sample 9
#1 Prediction
#2 Prediction
#3 Prediction
Test Sample 10
#1 Prediction
#2 Prediction
#3 Prediction
Test Sample 11
#1 Prediction
#2 Prediction
#3 Prediction