Faculty of Engineering
Cairo University

Computer Vision

Task 5

Face Detection and Recognition

Submitted by:

BN	Section	Name
45	1	Sohaila Mahmoud
10	1	Arwa Essam
32	2	Maryam megahed
46	1	Shrouq shawqy

Team 8

Submitted to:

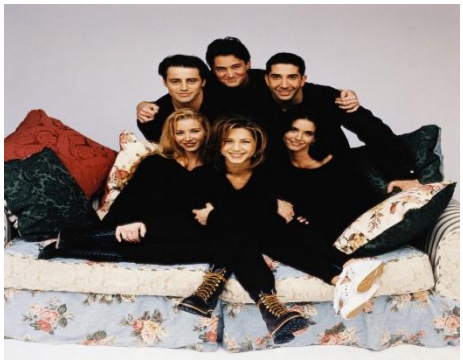
Eng. Laila Abbas

Face Detection

Face detection applications use algorithms and ML to find human faces within larger images, which often incorporate other non-face objects such as landscapes, buildings and other human body parts like feet or hands. Our face detection algorithm, extract face(image), aims to extract the face from an input image using the Haar cascade algorithm. Here are the steps involved in the function:

1. Convert the input image to grayscale
2. Detect faces in the grayscale image: The next step is to detect faces in the grayscale image using the Haar cascade classifier cascade_model. The cascade_model.detectMultiScale() function is used for this purpose. It takes several parameters:
    - gray: The grayscale image in which to search for faces.
    - scaleFactor: Parameter specifying how much the image size is reduced at each image scale. It helps in resizing the image for more efficient face detection. In this case, a scale factor of 1.4 is used.
    - minNeighbors: Parameter specifying how many neighbors each candidate rectangle should have to retain it. Higher values result in fewer detections but with higher quality. A value of 5 is used in this case.
    - minSize: Minimum possible object size. Objects smaller than this size are ignored. The minimum size is set to (32, 32) pixels in this case.
3. Extract the face from the image: If faces are detected (i.e., len(faces) > 0), the function proceeds to extract the face region from the original image. The last detected face coordinates are used for this purpose. The coordinates x, y, w, and h represent the top-left corner coordinates and the width and height of the detected face rectangle, respectively. The face region is extracted using array slicing: image[y:y+h, x:x+w]. This operation crops the image to include only the face region.
4. Return the extracted face

The Results:



Face Recognition

Eigen Faces

An eigenface is the name given to a set of eigenvectors when used in the computer vision problem of human face recognition.

The eigenvectors are derived from the covariance matrix of the probability distribution over the high-dimensional vector space of face images. The eigenfaces themselves form a basis set of all images used to construct the covariance matrix. This produces dimension reduction by allowing the smaller set of basis images to represent the original training images. Classification can be achieved by comparing how faces are represented by the basis-set.

How to Get Eigen Faces and Recognize the Face

First, we calculate the mean and get the mean face.

The Function :

- Initialize the mean face array of shape (1, height*width) filled with zeros
- Calculate the sum of all training images: The function then loops over each image and adds it to the mean_face.
- Calculate the mean face: After the loop, the function divides the mean_face array by the length of image_paths (which represents the number of training images). This step computes the average of all the images and flattens the resulting array. The variable mean_face now represents the mean face of the training images.
- Normalize the faces.
- Iterate over each image and subtract the mean face.
- Return the mean face and normalized faces.

2.-Calculate the covariance:

The covariance matrix represents the relationships between different features (pixels) of the normalized faces.

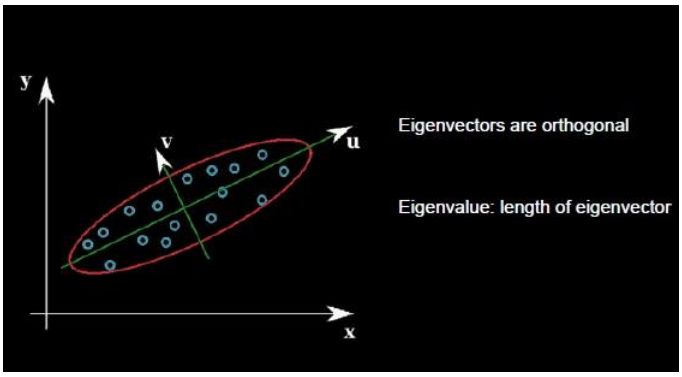
The function starts by:

- calculating the covariance matrix of the normalized_faces array using np.cov(normalized_faces).
- Normalize the covariance matrix: The function then divides the covariance matrix by the number of normalized faces (i.e., len(normalized_faces)). This step normalizes the covariance matrix by scaling it based on the number of samples.
- Return the covariance matrix.

3.-Extract eigenvectors from the covariance.

From the covariance we can extract the eigenvectors.

Note: The first eigenvector will describe more information than the second and so on. For this reason, later we have to pick the first eigenvectors generated (avoiding noise).

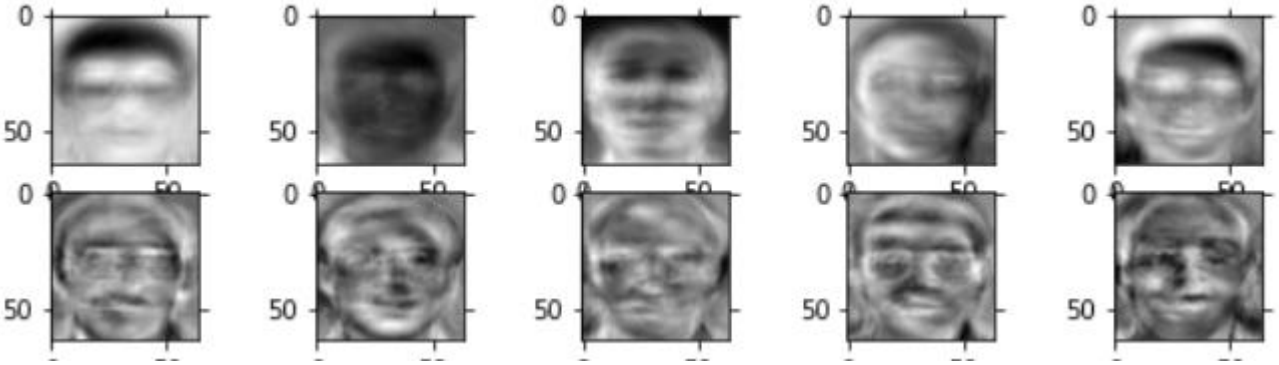


4.-Calculate eigenfaces: eigenvectors x normalized pictures.

Each eigenvector is multiplied by the whole normalized training set matrix and as a result, we will have the same amount of eigenfaces as images in our training set.

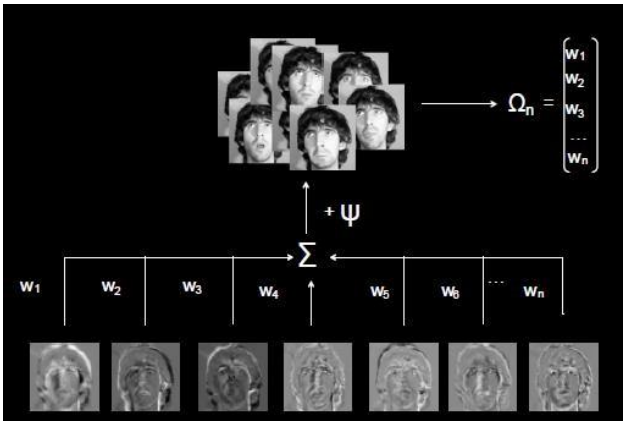
5.-Choose the most significant eigenfaces.
The first eigenfaces represent more information than the last eigenfaces. Actually, the last eigenfaces only add noise to the model, so it is necessary to avoid them.

Sample of them:



6.-Calculate weights: chosen eigenfaces x normalized pictures.

Each normalized face in the training set multiplies each eigenface. Consequently, there will be N set of weights with M elements (N = number of pictures in the training set, M = number of eigenfaces).



Second, we recognized the test face:

7.-Vectorize and normalize this picture: subtract the calculated mean from the picture.

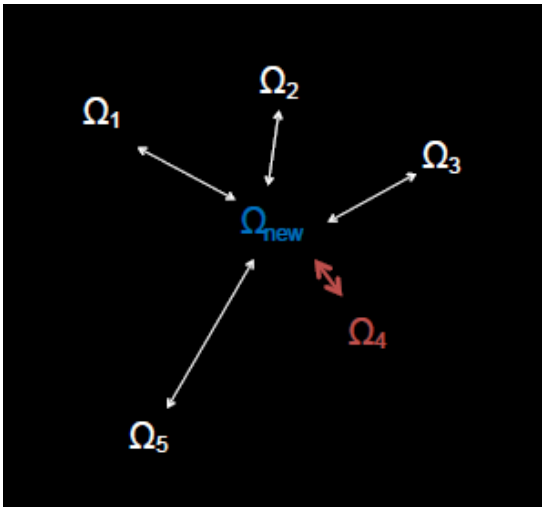
Reshape the test picture into a vector and subtract the mean calculated in 2) from it.

8.-Calculate the weights: multiply the eigenfaces x normalized picture.

The same as 8) but with the test picture.

9.-Interpret the distance of this weight vector in the face space and get the matched one (the closest distance).

Second, we recognized the test face:



Results:

Original

Best match

Performance Evaluation.

To report any system's performance, many metrics (Accuracy, AUC, etc.) can be used. In our case, we chose two performance metrics:

1. ROC curve:

A ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- True Positive Rate
- False Positive Rate

True Positive Rate (TPR) is a synonym for recall and is defined as follows:

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

False Positive Rate (FPR) is defined as follows:

$$FPR = \frac{FP}{FP + TN}$$

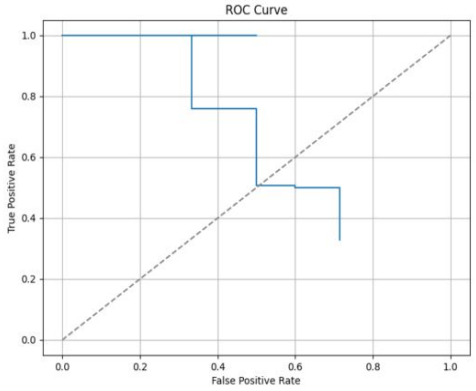
Now, we need to define the following:

- TP: True positives are the images that have been agreeing with the true label
- FP: False Positives are the images that have been wrongly disagreeing with the true label
- TN: True negatives are the images that have been agreeing with the true label in denying a claim
- FN: False negatives are the images that have been wrongly disagreeing with the true label by claiming a wrong claim.

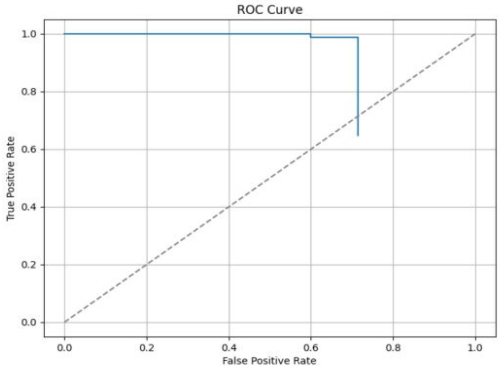
ROC curve plots TPR vs. FPR at different classification thresholds.

ROC results:

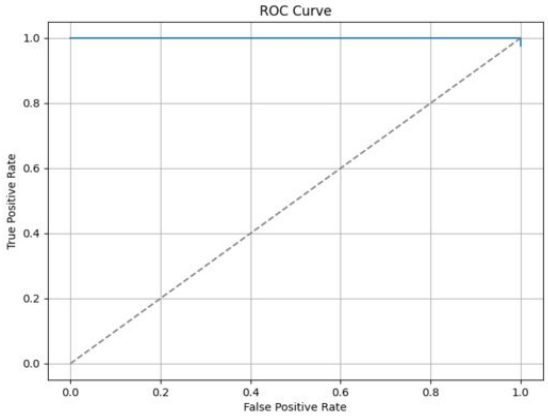
Threshold=100



Threshold=200



Threshold=300



2. Accuracy:

Accuracy is given by: $\frac{TP+TN}{TP+TN+FP+FN}$

Our output accuracy for threshold=100:

32%

Our output accuracy for threshold=200:

62%

Our output accuracy for threshold=300:

92%