Faculty of Engineering
Cairo University

Computer Vision

Task 5

Face Detection and Recognition

Submitted by:

BN	Section	Name
10	1	Esraa Mohamed Saeed
12	1	Alaa Tarek Samir
15	1	Amira Gamal Mohamed
8	2	Fatma Hussain Wageeh
26	2	Mariam Mohamed Osama

Team 3

Submitted to:

Eng. Laila Abbas

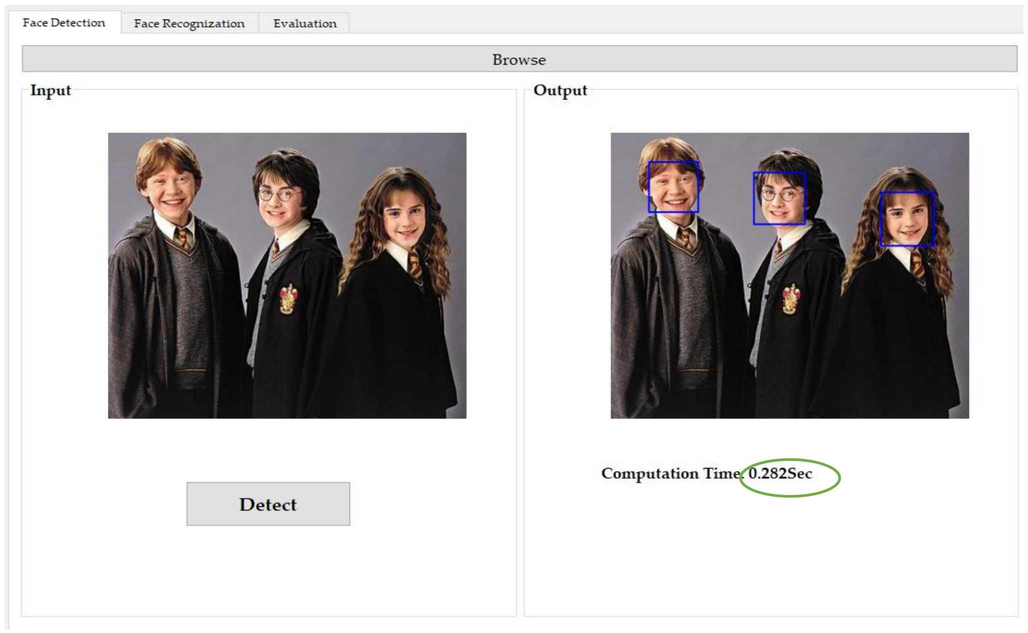
Contact Email: amira.gamal.ag22@gmail.com

Face Detection

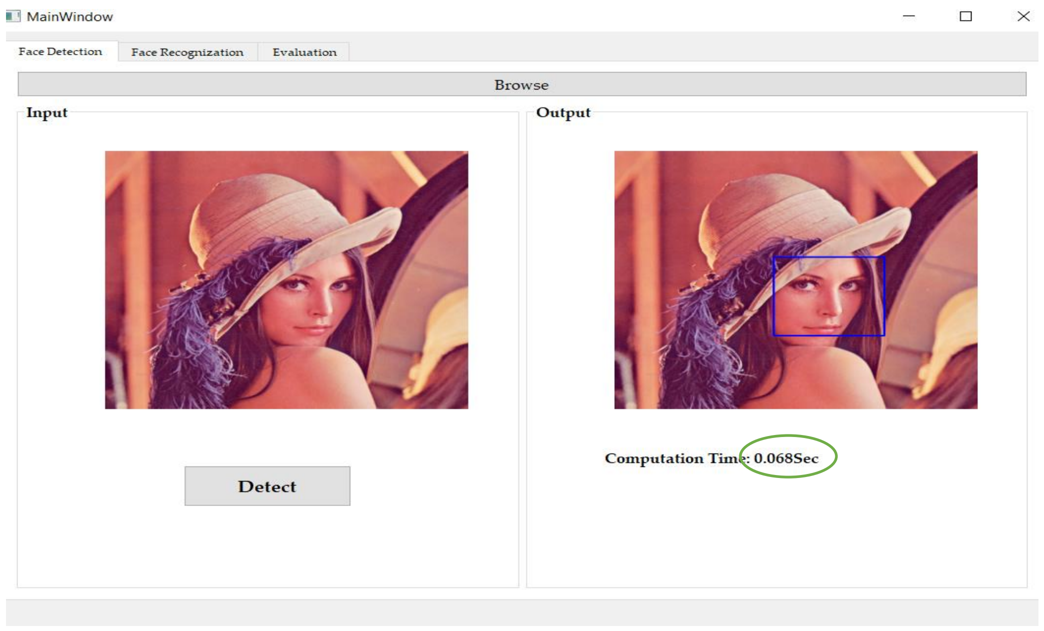
Face detection applications use algorithms and ML to find human faces within larger images, which often incorporate other non-face objects such as landscapes, buildings and other human body parts like feet or hands. Face detection algorithms typically start by searching for human eyes -- one of the easiest features to detect. The algorithm might then attempt to detect eyebrows, the mouth, nose, nostrils and the iris. Once the algorithm concludes that it has found a facial region, it applies additional tests to confirm that it has, in fact, detected a face.

We load the face cascade _which is just an XML file that contains the data to detect faces_ into memory so it’s ready for use.

Results and Computation Time

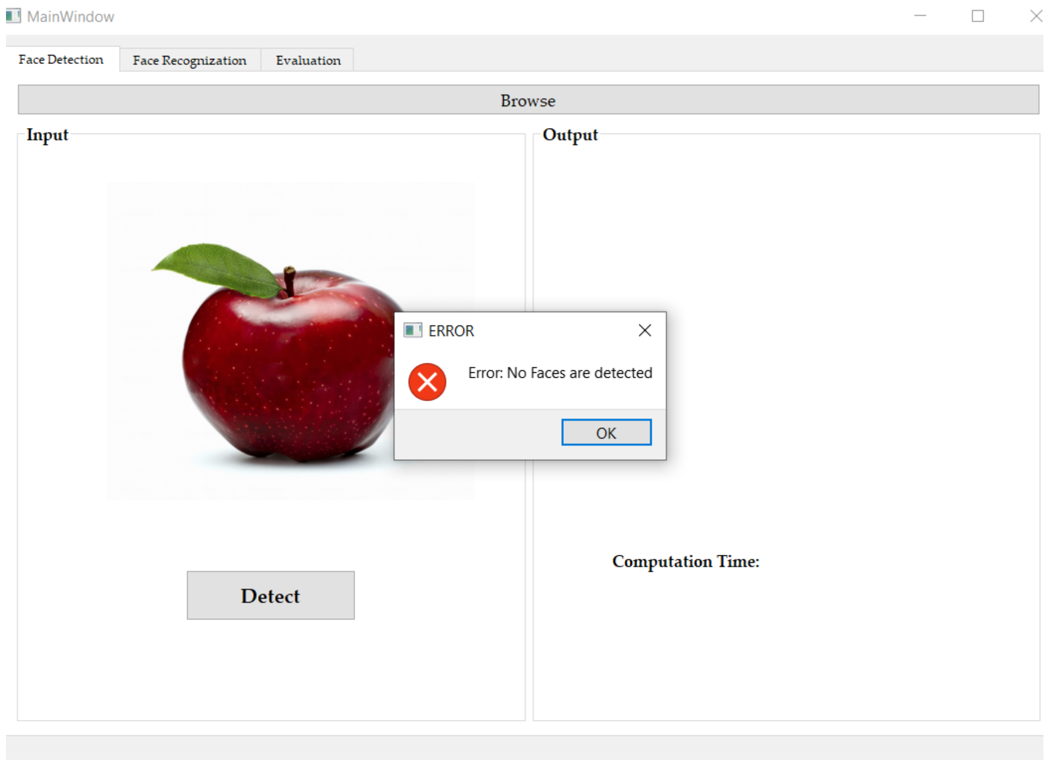


Computation Time: 0.282Sec



Computation Time: 0.068Sec

If there is no face:



Error: No Faces are detected

Face Recognition by Eigen Faces

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 trained the model:

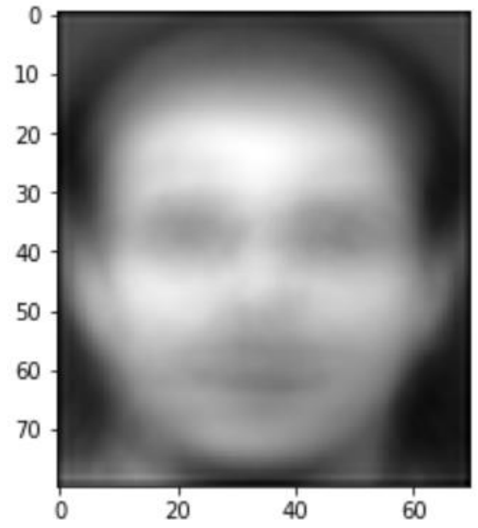
1.-Flat the black and white images of the training set (from matrices (2D) to vectors (1D))

1	2	3
4	5	6
7	8	9

1	4	7	2	5	8	3	6	9

2.-Calculate the mean and get the mean face.

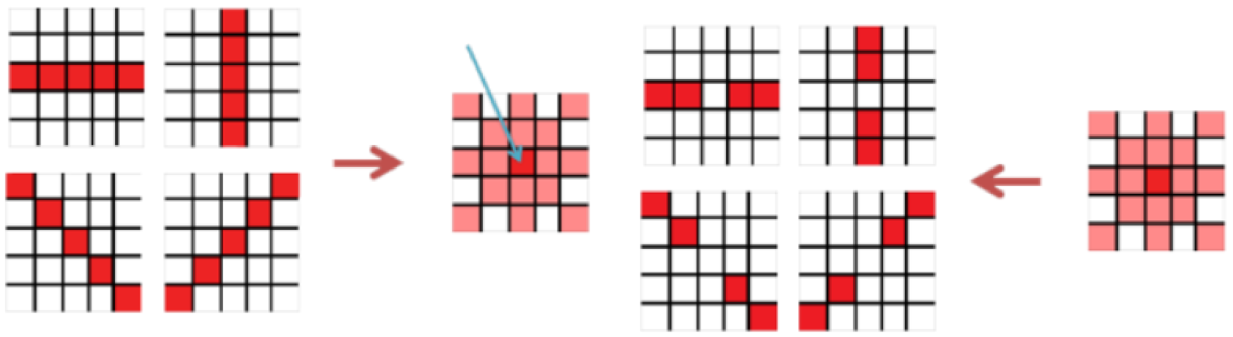
The mean is just the sum of all of the pictures divided by the number of pictures. As a result, we will have a “mean face”.



3.-Normalize the training set: for each image, subtract the mean.

To normalize the training set, we just simply need to subtract for each picture in the training set the mean that was calculated in the previous step.

The reason why this is necessary is because we want to create a system that is able to represent any face. Therefore, we calculated the elements that all faces have in common (the mean). If we extract this average from the pictures, the features that distinguish each picture from the rest of the set are visible.



4.-Calculate the covariance: multiply all images by themselves.

The covariance represents how two variables change together. After the previous step, we have a set of images that have different features, so now we want to see how these features for each individual picture change in relation to the rest of the pictures.

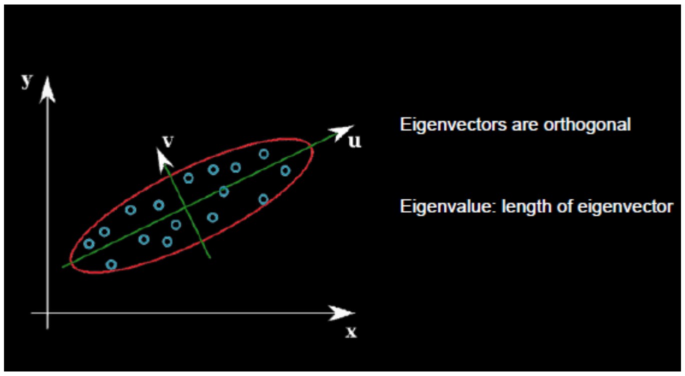
For this purpose, we put all the flat normalized pictures together in a vector. My training set consists of 328 pictures whose dimensions are 80×70 pixels. Therefore, the resulting matrix will be 5600×328. The covariance is the multiplication of this matrix by itself, and if we transpose it correctly, the resulting matrix will be 16×16:

328x5600 * 5600x328 = 328x328

5.-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).


y
v
u
x
Eigenvectors are orthogonal
Eigenvalue: length of eigenvector

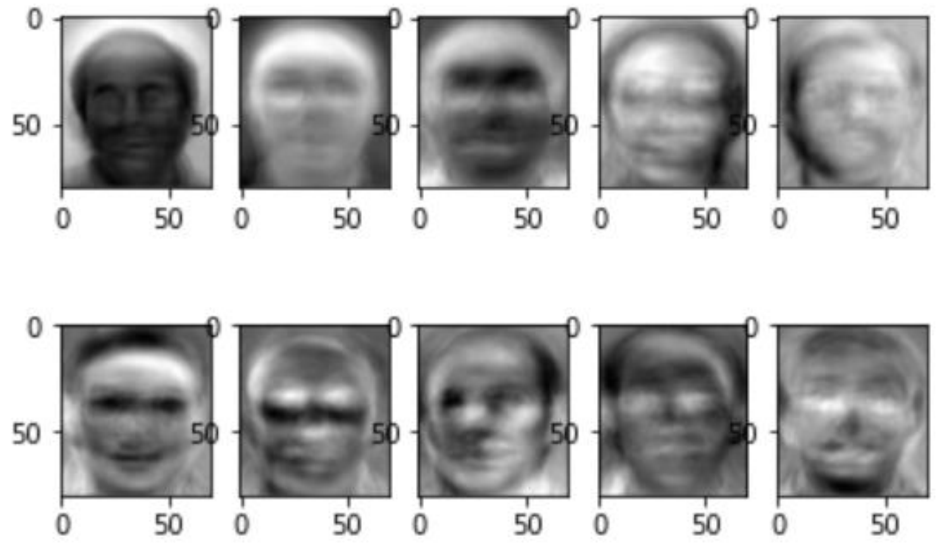
6.-Calculate eigenfaces: eigenvectors x normalized pictures.

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.

7.-Choose the most significant eigenfaces.

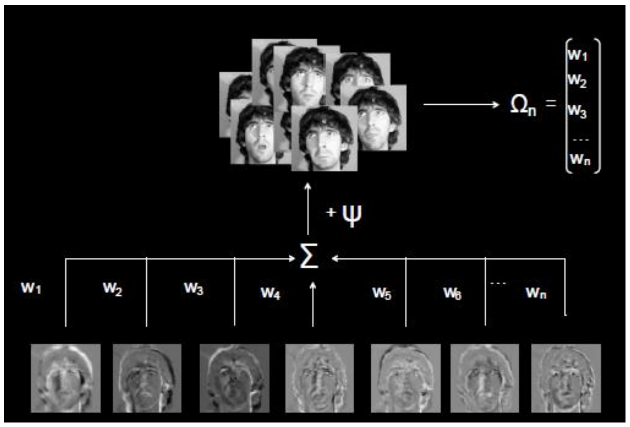
The first eigenfaces represent more information than the last eigenfaces. Actually, the last eigenfaces only add noise to the model, so it is necessary to avoid them.

Sample of them:



8.-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).



$\Omega_n = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \cdots \\ w_n \end{bmatrix}$
$+\Psi$
Σ
w_1
w_2
w_3
w_4
w_5
w_6
...
w_n

Second, we recognized the test face:

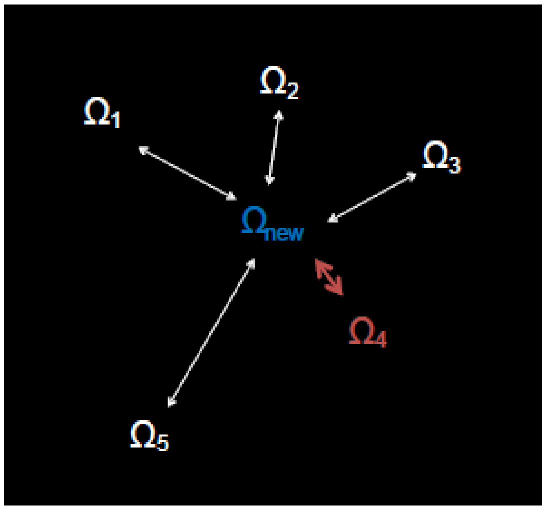
9.-Vectorize and normalize this picture: subtract the calculated mean from the picture.

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

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

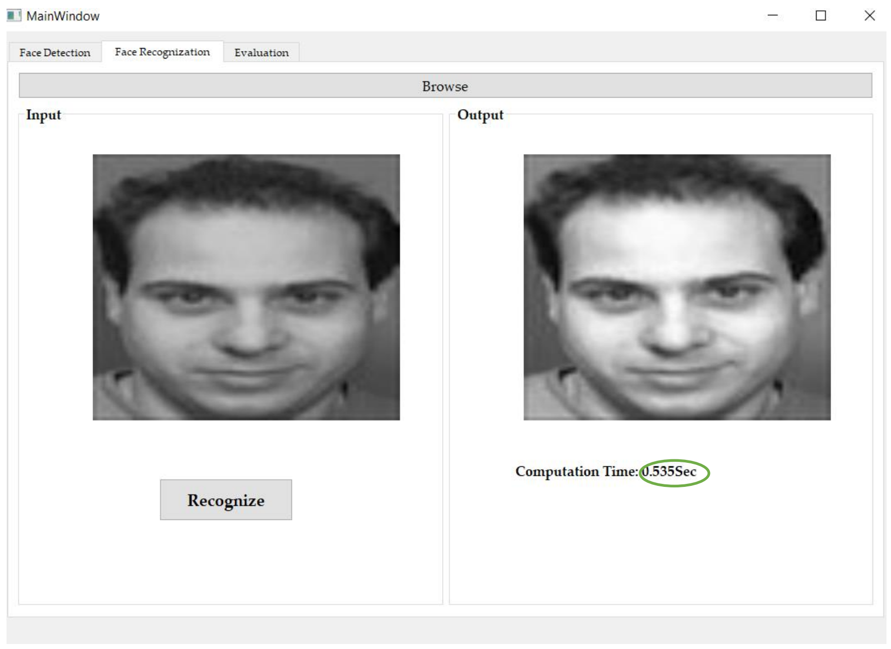
The same as 8) but with the test picture.

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

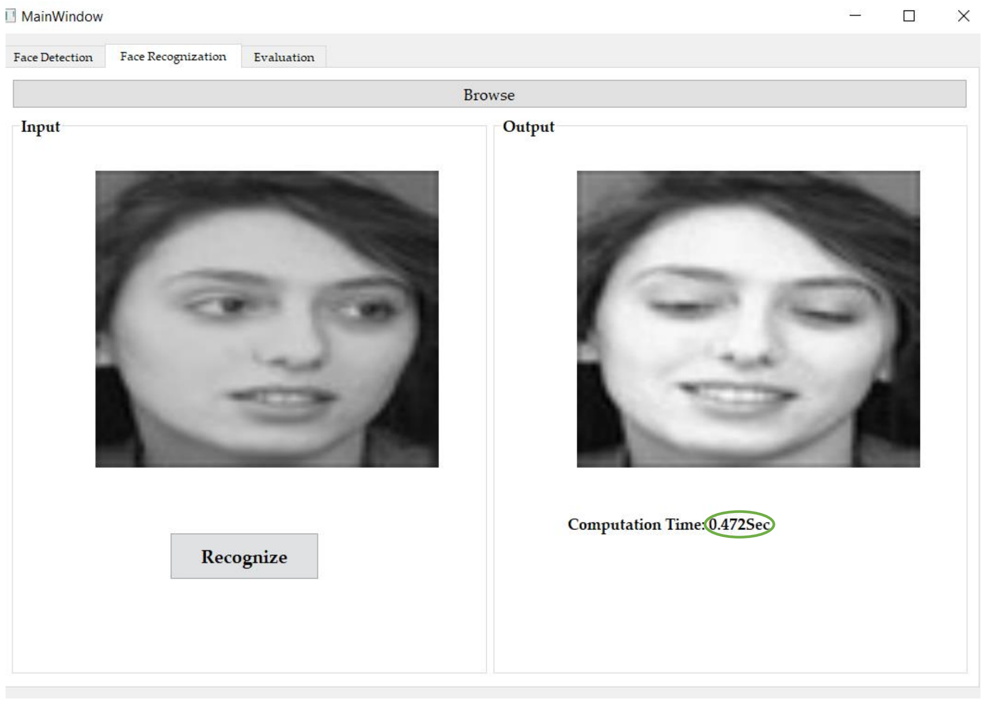


Results ana Computation Time

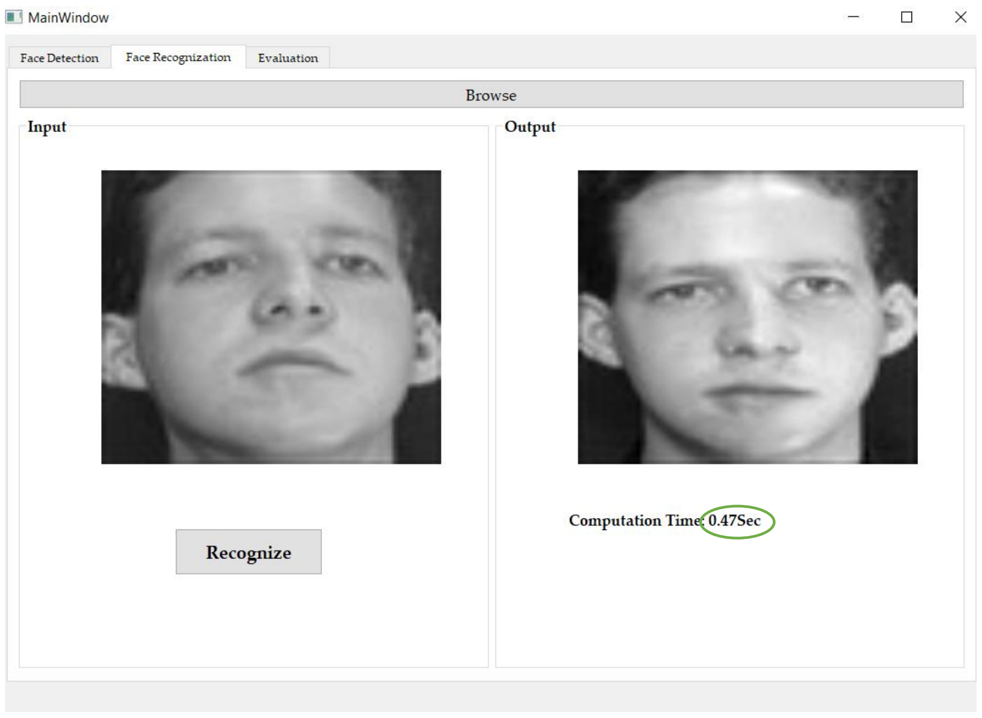
There are some images with their best matched one



Computation Time: 0.535Sec



Computation Time: 0.472Sec



Computation Time: 0.47Sec

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. Accuracy:

Testing the accuracy is given by: $\frac{Number\ of\ truely\ recognized\ images}{number\ of\ all\ images} * 100\%$

Our Eigen Faces systems delivered an accuracy of: **95.6%**

For better accuracy, we employed a RandomForest machine learning model and it reached an accuracy of: 96.5% which is not much better than our model.

2. 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:

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

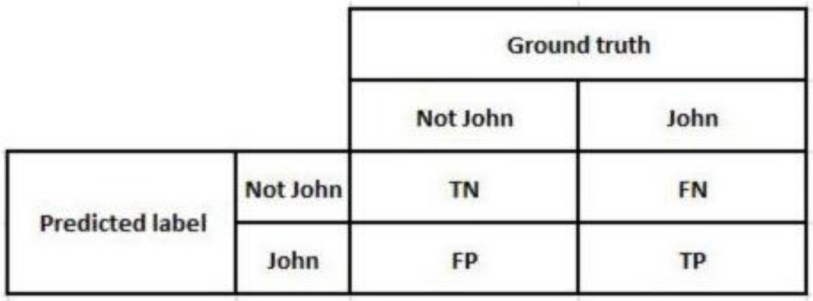
False Positive Rate (FPR) is defined as follows:

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

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

The image below may give a better insight of what these variables are



		Ground truth	
		Not John	John
Predicted label	Not John	TN	FN
	John	FP	TP

Example of classification terms used in context of face recognition.

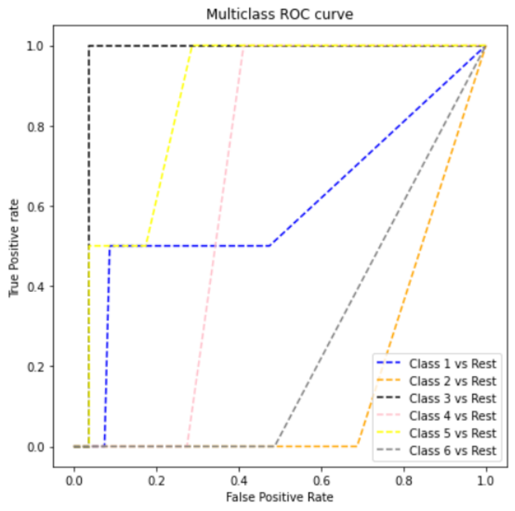
In the above scenario the prediction and ground truth (or actual outcome) can result in one of four outcomes:

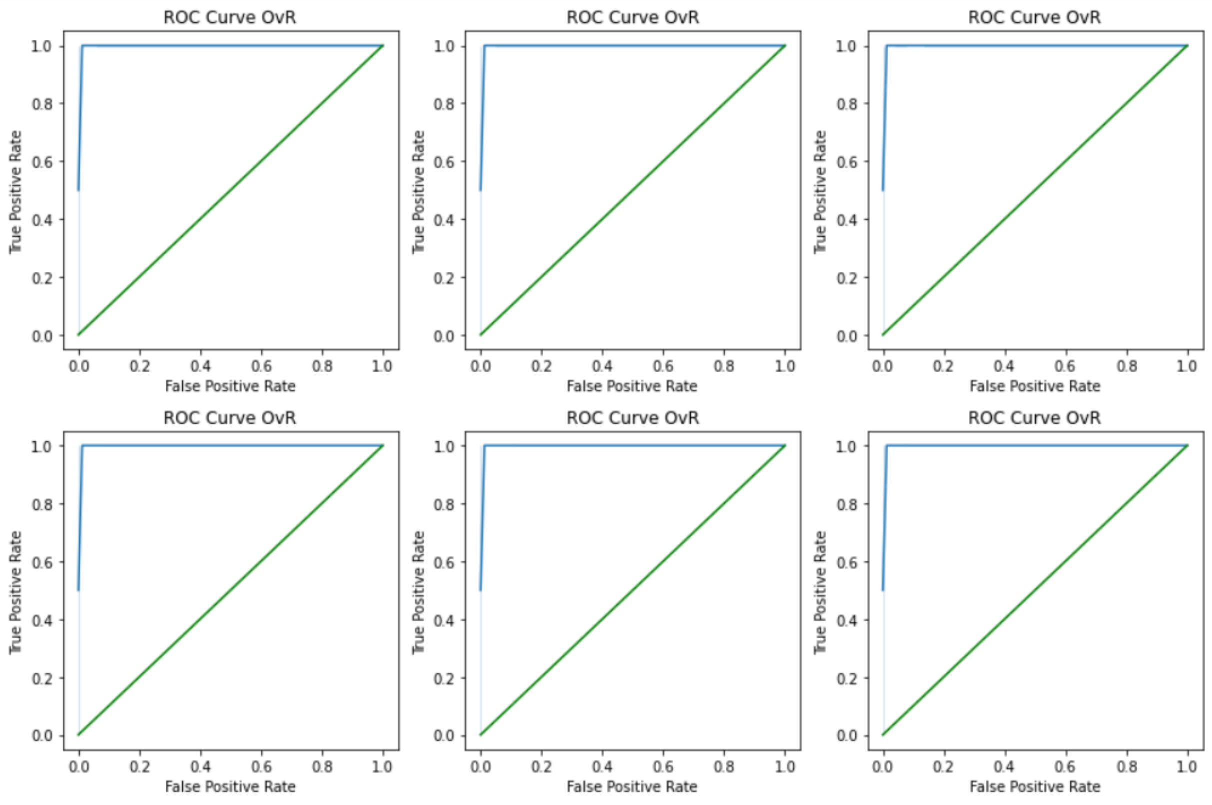
True Positive (TP) — the model identifies John as John.
False Positive (FP) — the model identifies another person as John.
False Negative (FN) — the model identifies John as someone else.
True Negatives(TN) — the model identifies some other person as 'Not John'.

ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows a typical ROC curve.

In our code, we represent each subject (person) in a separate ROC curve.

The following curves are some examples of the output ROC curves for some subjects:




ROC Curve OvR
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC Curve OvR
True Positive Rate
False Positive Rate
ROC Curve OvR
True Positive Rate
False Positive Rate
ROC Curve OvR
True Positive Rate
False Positive Rate
ROC Curve OvR
True Positive Rate
False Positive Rate
ROC Curve OvR
True Positive Rate
False Positive Rate