

**ASSIGNMENT 1**

**Face Recognition**

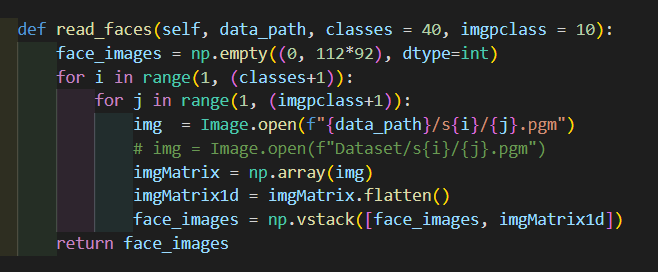
|  |  |
| --- | --- |
| Name | ID |
| Khaled Mohamed Mohamed Ahmed | 20010528 |
| Mohamed Aly Hassan Mahmoud | 20011662 |
| Mohamed El-Nady Mohamed Gomma | 20011513 |

Problem Statement

* We intend to perform face recognition. Face recognition means that for a given image you can tell the subject id. Our database of subject is very simple. It has 40 subjects. Below we will show the needed steps to achieve the goal of the assignment.

Generate the Data Matrix and the Label vector:

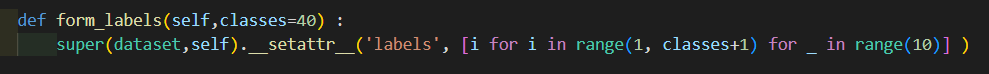
* We make a method called read\_faces() that read every pgm image in the form of 2D pixel matrix. Then we convert this 2D matrix to 1D vector using numpy.flatten()
* After converting every image into a vector, we concatenate this vector to a 2d matrix having all the images as 1D vectors of size 1x10304 and the matrix of size 400x10304



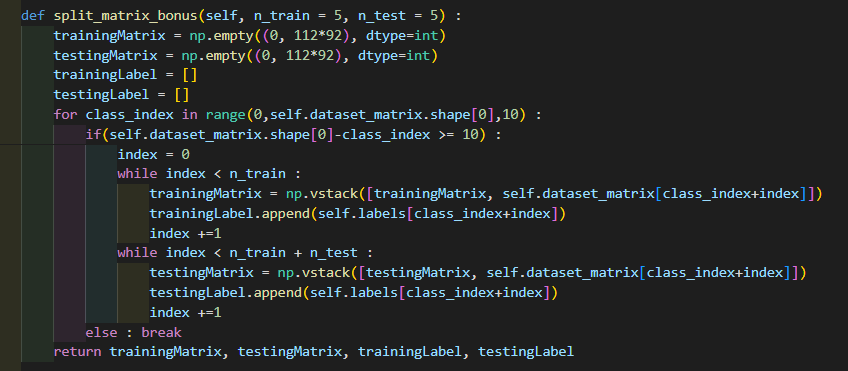
**Figure 1:Read pgm images and convert them to vector images**

Split the Dataset into Training and Test sets:

* After we made the 2D matrix of vector images (dataset\_matrix), we split this matrix to two matrices: training matrix and testing matrix. Each of 200 vector image (default) divided by parity by function called split\_matrix\_with\_bonus().
* split\_matrix\_with\_bonus() takes the number of instances of an image in training set and in the testing set and split the dataset\_matrix accordingly.
* We also had the label matrix constructed after reading the images by function called form\_labels().
* We have another function called read\_non\_faces() that read other non face images, resize each image to 92x112 to grayscale image.

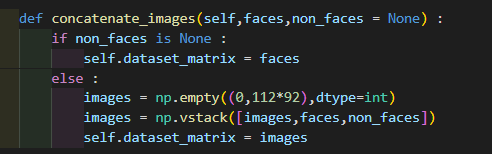


**Figure 2: method to form label vector based on the read faces**

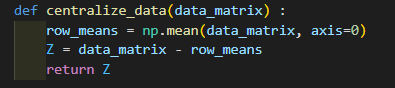
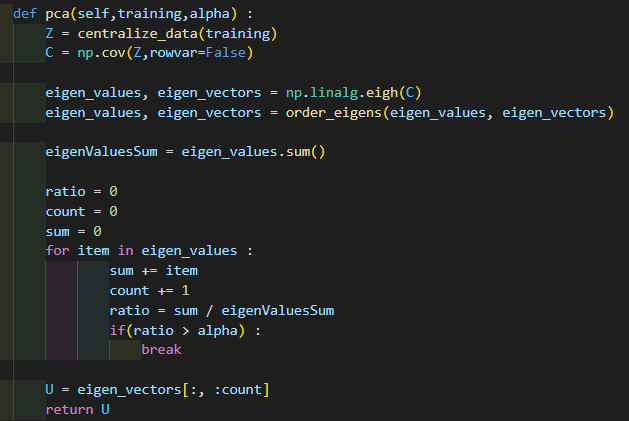


**Figure 3: split the dataset\_matrix to training, testing images and labels**

* For face recognition with non-face images, we use a method called concatenate\_images() used to concatenate the faces with non-faces before splitting.



PCA Algorithm:

* First we centralized the dataset using function centralize\_data.
* Second, we computed the covariance matrix by numpy.cov(Z) to give a matrix of size 10304x10304
* Compute the eigenvalues and eigenvectors of the covariance matrix
* Order the eigenvalues and eigenvectors in descending order.
* Get the number of needed eigenvectors (eigenfaces) by using different values of alpha.
* These eigenfaces returned used to project the data on the new space and predict the testing sets

**Faces/Accuracy comparison**

|  |  |
| --- | --- |
| Value of alpha | Accuracy |
| 0.8 | 88% |
| 0.85 | 93.5% |
| 0.9 | 94% |
| 0.95 | 93.5% |

**Conclusion:** By increasing the value of alpha, the number of eigenfaces will increase so the ability to detect the correct faces will increase.

**For non-faces with 1000 images:**

|  |  |
| --- | --- |
| Value of alpha | Accuracy |
| 0.8 | 97.86% |
| 0.85 | 98% |
| 0.9 | 97.5% |
| 0.95 | 96.57% |

**BONUS PART**

* 1. Use different Training and Test splits. Change the number of instances per subject to be 7 and keep 3 instances per subject for testing. compare the results you have with the ones you got earlier with 50% split.

|  |  |  |  |
| --- | --- | --- | --- |
| ALPHA | 7-training and 3-testing | | 5-training and 5-testing |
| 0.9 | 95% | 94% | |
| 0.8 | 95.83% | 88% | |
| 0.85 | 95.83% | 93.5% | |
| 0.95 | 94.17% | 93.5% | |

We see that when we changed the ratio between training and testing using faces only (not including non-faces) the accuracy is increased that happed due to the increased number of training sets that gives accurate numbers and facilitate the true prediction using nearest k neighbors