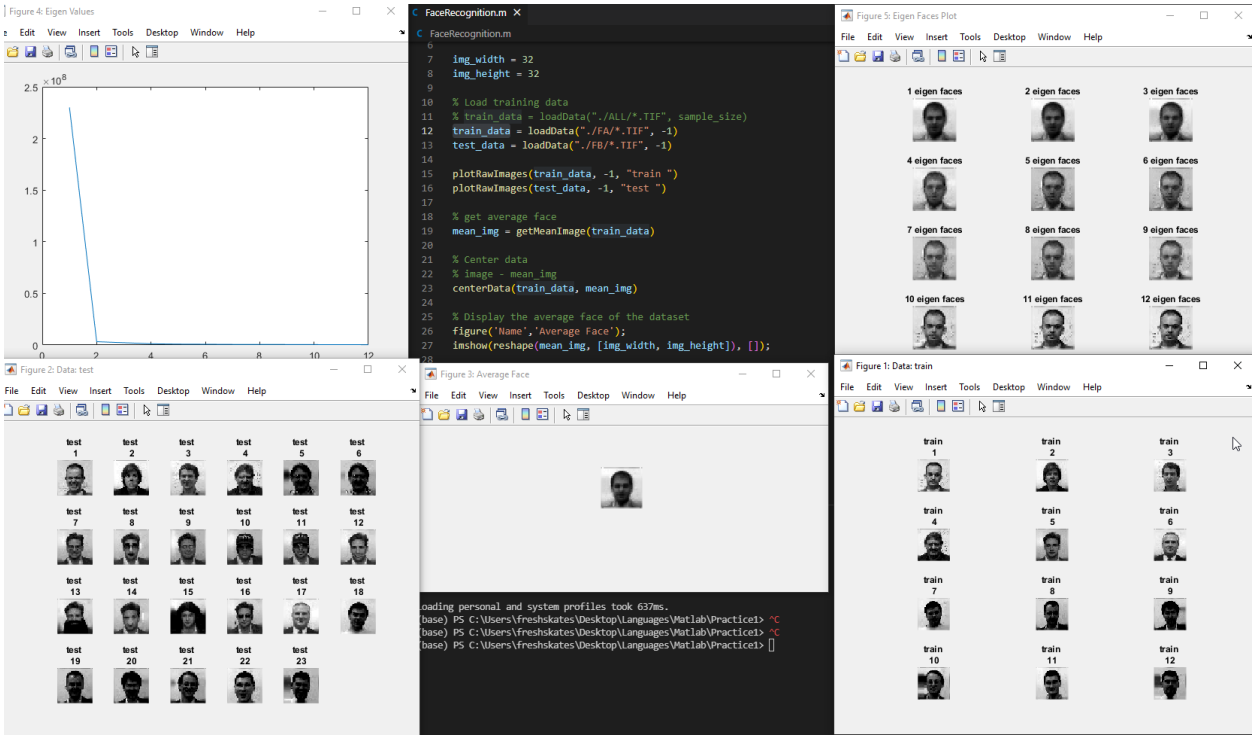
Robert Cacho

Dr. Okada

October 6, 2021

Fast Prototype 1: Eigen Faces

Most of my time was spent reading and modularizing the code so it doesn't look so bad.



Conceptual I followed:

Principal Component Analysis

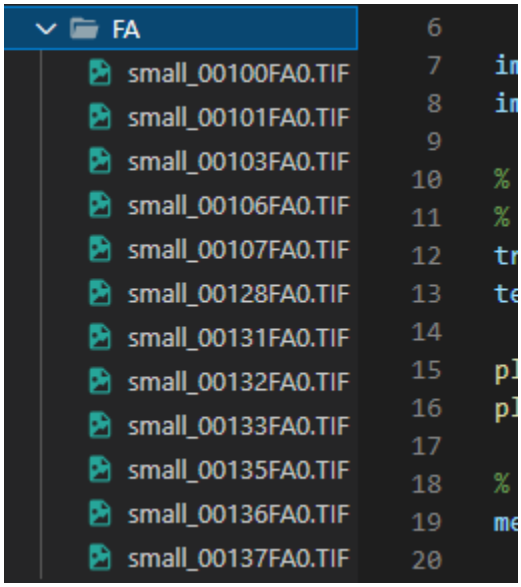
- Conceptual Steps
  1) Collect M Training Images (must be aligned, Nx by Ny matrix)
  2) Vectorize the Images: $X = \{x_1, .., x_M\}$ Each of M images is a column vector with N coefficients where $N = Nx$ times Ny
  3) Compute mean image: $\mu = mean(X)$; a vector of N coeffs
  4) Construct Covariance Matrix: $C = (X - \mu^T)(X - \mu^T)^T$ N by N mat
  5) Solve Eigenvalue Problem: $Cv_i = \lambda_i v_i$
  6) Sort resulting eigen vectors in decreasing order of corresponding eigen values.
  7) Select the top K Eigenvectors $W = \{v_1, .., v_K\}$, resulting in a face model $\{\mu, W\}$

CSC872: PAMI – Kazunori Okada (C) 2021 9

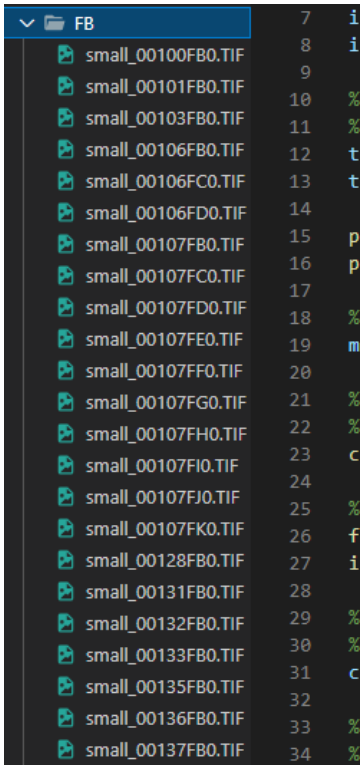
9

Collect M training images.

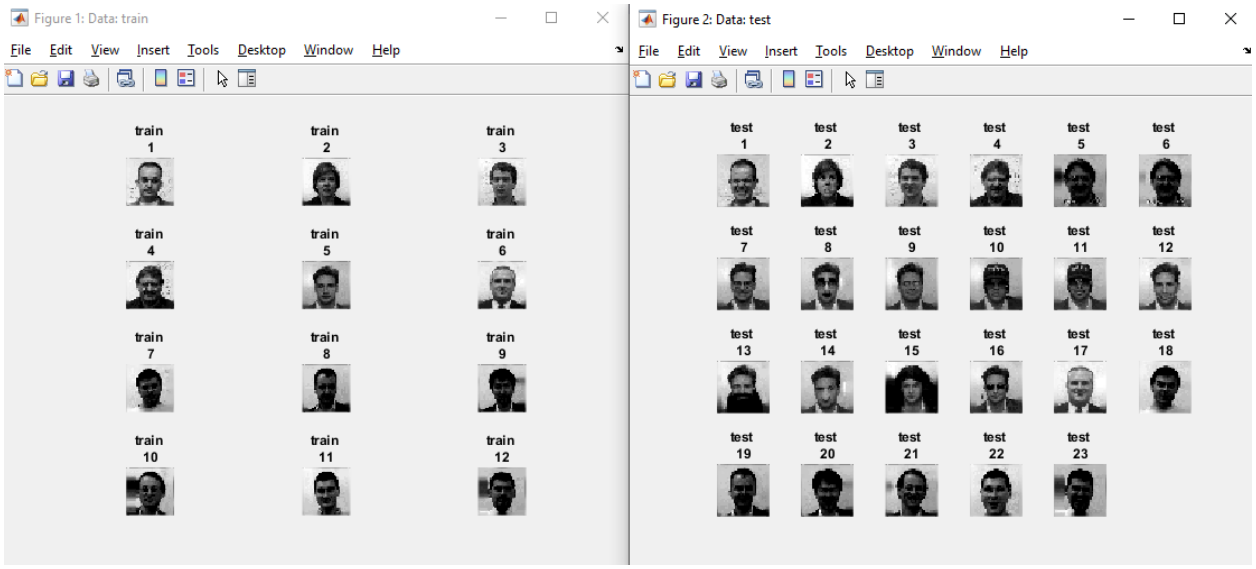
Training Images being the FA folder.



And the testing images coming from the FB folder



Each image is a 32 * 32



Step 2:

Vectorize the Images:

1024x23 double

	1	2	3	4	5	6	7	8	9	10	11	12
1007	170	172	172	162	121	152	188	177	197	189	186	18
1008	171	172	172	162	126	154	185	175	194	186	182	18
1009	171	171	172	161	132	158	181	175	191	182	178	18
1010	171	171	171	163	140	160	178	174	188	179	175	17
1011	170	171	169	162	148	162	174	174	185	176	172	17
1012	169	171	168	162	154	163	171	173	182	173	169	17
1013	168	170	167	162	159	163	169	172	179	170	168	16
1014	167	168	165	162	161	162	167	170	176	165	166	16
1015	167	166	164	163	161	161	165	168	173	156	164	16
1016	166	159	163	153	160	160	163	164	169	141	158	16
1017	166	144	162	130	158	156	157	156	166	123	146	16
1018	167	122	159	105	156	151	144	141	160	106	128	15
1019	167	96	155	81	153	140	126	120	150	91	106	14
1020	164	72	149	72	151	132	106	100	135	81	88	12
1021	155	56	141	63	150	126	90	84	116	75	78	10
1022	143	49	133	57	151	125	79	76	97	74	75	9
1023	134	58	132	62	143	125	84	83	93	83	84	8
1024	141	97	147	94	152	141	120	118	123	121	120	11

All the images are stored in a vector 1024 * 23

Each column has 1024 elements, this is because when you reshape from 1024 to a [32, 32] then it becomes the original image.

```
61  function train_data = loadData(img_path, quantity)
62      global img_width
63      global img_height
64
65      train_data = [];
66      data = imageDatastore(img_path)
67      if quantity == -1
68          quantity = length(data.Files);
69      end
70
71      temp_train_data = zeros(quantity, img_width * img_height);
72      for i = 1:quantity
73          train_data_temp_1 = imread(data.Files{i})
74          temp_train_data(i,:) = reshape(train_data_temp_1, [1, img_width*img_height]);
75      end
76
77      train_data = [train_data; temp_train_data];
78
79      train_data = train_data'
80  end
```

Function to load data from path

Step 3:

Compute the mean image and subtract it from the vector.

Compute mean image: $\mu = mean(X)$; a vector of N coeffs

This is the function I created to get and return the mean.

```
81
82  function mean_img = getMeanImage(data)
83      mean_img = mean(data, 2)
84  end
```

Step 4:

Construct Covariance Matrix, use the mean to subtract from each dataset and multiply by the transpose of the data.

4)Construct Covariance Matrix: $C = (X - \mu^T)(X - \mu^T)^T$ N by N mat

```
85
86  function [] = centerData(data, mean_img)
87      for i = 1:size(data, 2)
88          data(:, i) = data(:, i) - mean_img
89      end
90  end
91
92  function cov_mat = covarianceMatrix(data)
93      cov_mat = data' * data;
94  end
95
```

These are the two functions that will complete this step, constructing the covariance matrix first involved centering the data, which means subtracting the mean. Then we data * data(transpose).

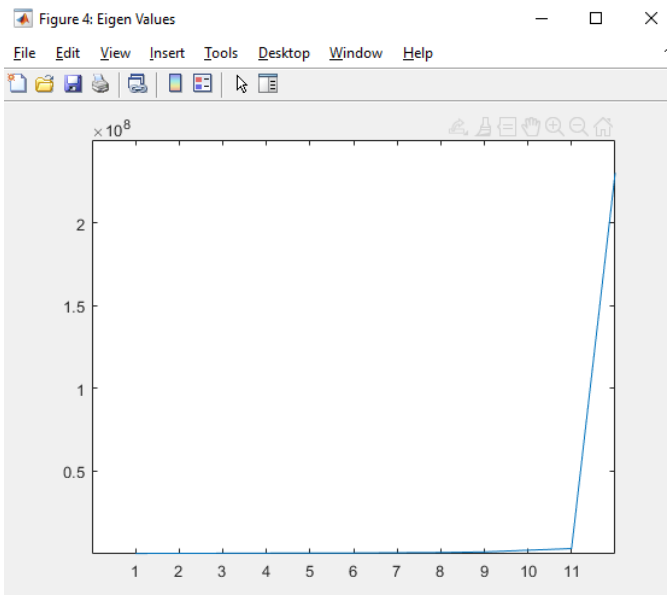
Step 5:

5)Solve Eigenvalue Problem: $Cv_i = \lambda_i v_i$

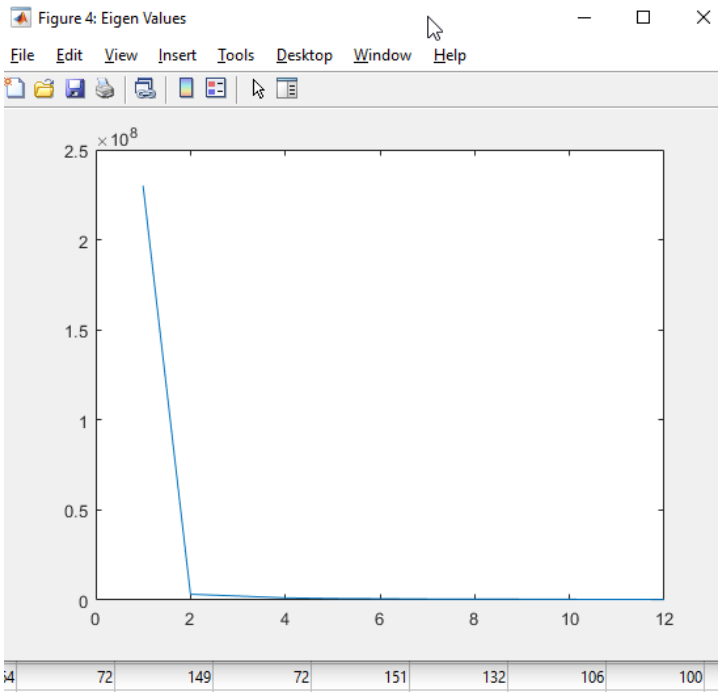
Step 6:

This one was tricky for me, I had trouble identifying and sorting the eigenvectors in decreasing the order of corresponding values. At first, I had to plot out to see that to arrange the eigenvectors in decreasing order I would have to reverse the vector.

```
99
00      eigValues = eigValues(end:-1:1);
01      V = V(:,end:-1:1);
02
```



Incorrect Figure because not in decreasing order

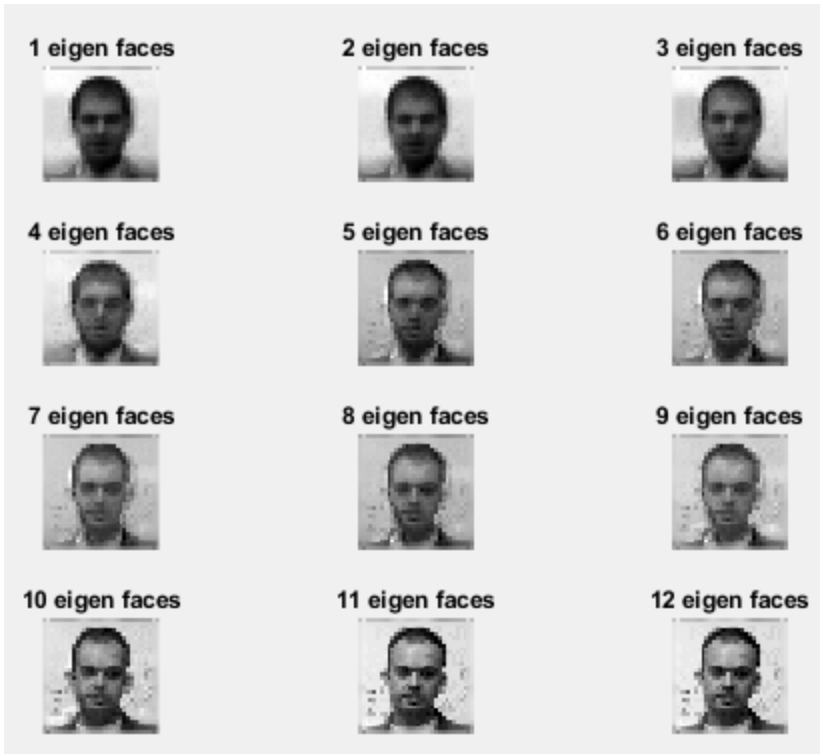


Correct figure

Step 7:

7)Select the top K Eigenvectors $W = \{v_1, .., v_K\}$, resulting in a face model $\{\mu, W\}$

This one was confusing to me, but I believe after reading I am close or got it.


1 eigen faces
2 eigen faces
3 eigen faces
4 eigen faces
5 eigen faces
6 eigen faces
7 eigen faces
8 eigen faces
9 eigen faces
10 eigen faces
11 eigen faces
12 eigen faces

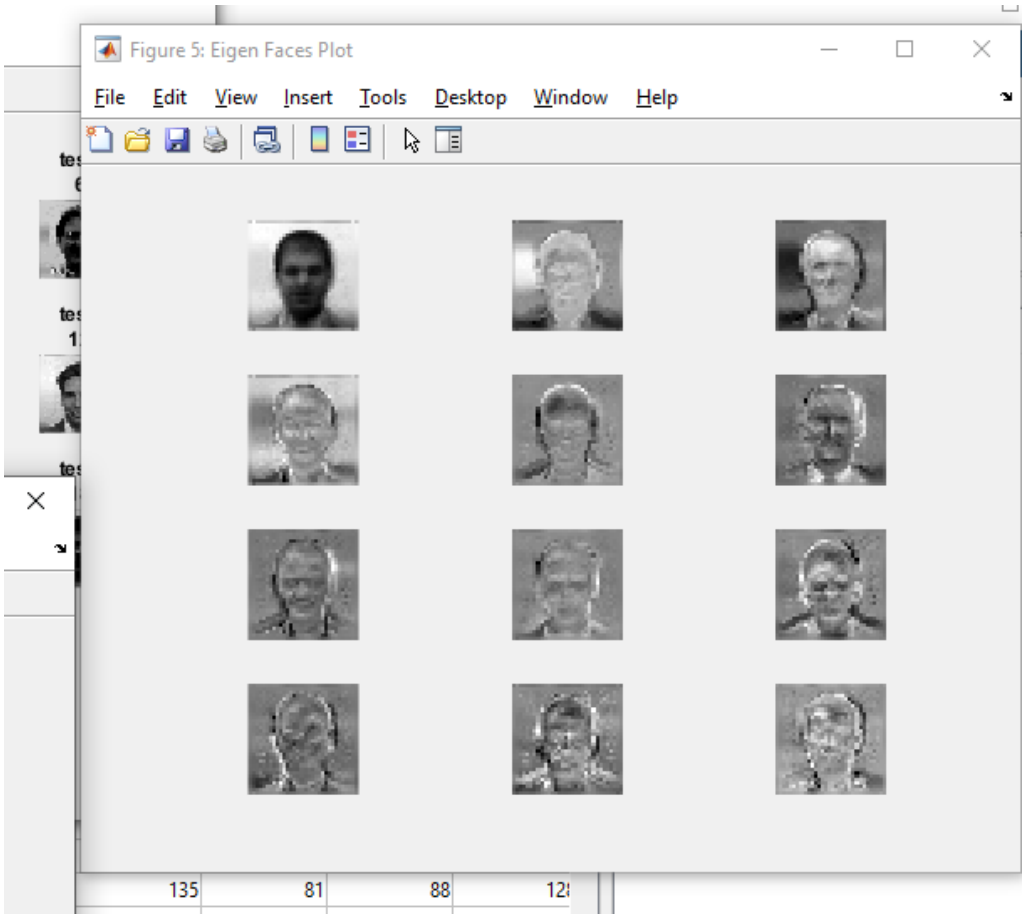


Figure 5: Eigen Faces Plot
File Edit View Insert Tools Desktop Window Help
135 81 88

Nearest Neighbor Recognition

- Learning & Database Construction
  1) Do PCA, yielding a face model $\{\mu, W\}$
  2) Construct DB of known faces with codes $y_j = W^T(x_j - \mu^T)$ for all known faces $\{x_j\}$ in FA

- Face Recognition by NN Classification
  1) Test face z is also projected to the model $W^T(z - \mu^T) = y_z$
  2) Nearest neighbor classification of y_z with $\{y_i\}$ by picking the index "i" that best match to y_z according to Euclidean distance

10

I was not able to implement the rest of this, a huge barrier was not knowing Matlab, however, I understand most of the concepts like PCA and the distances. Which actually I was wondering what the difference would be if we used manhattan distance instead of euclidean. It's just something I was curious about and would have been nice to test out.