The paper was very interesting because it didn't talk only about how to implement eigenfaces with a controlled environment, but also showed how to deal with background and issues with the face not being centered. This will be very helpful for my group and I in our next QEA project!

It was actually very similar, and it was the first time I could look at some of their weird looking equations and understand what they were saying.

It didn't really change because I read the paper after I did the homework.

I got stuck when my brain got tired or I saw math, so I came to the reading at a later time.

Is PCA used in combination with deep learning or is PCA not able to be combined with those kinds of techniques?

1. Use PCA to compute the k principal components of the training face images (the k eigenvectors with largest eigenvalues).

```
clear;
load('class_data_train.mat')
load('class_data_test.mat')
```

Flattening the data, so that it becomes a 2d matrix instead of 3d

```
flatten_grayfaces_train = reshape(grayfaces_train, 64*64, 356);
flatten_grayfaces_test = reshape(grayfaces_test, 64*64, 356);
```

Calculate the mean of the training data in respect to pixel to pixel. If you cov face to face, then you don't create a "building block" for the face. It is supposed to be trained like a linear combination of vectors that will make the face, so we cov pixel to pixel

This means that the cov matrix will be 4096 by 4096 rather than 356 by 356.

```
mean_flatten_grayfaces_train = mean(transpose(flatten_grayfaces_train))
```

mean_flatten_grayfaces_train = *1×4096*

206.3205	201.1263	195.3155	188.5621	183.0787	178.3885	172.7266	167.0221	162.4965	158.968

```
A = transpose(flatten_grayfaces_train) - mean_flatten_grayfaces_train
```

A = *356×4096*

27.7187	32.9547	38.3928	44.2128	49.8504	54.6060	59.4500	63.8839	69.4919	72.831
26.8317	30.5334	36.9053	43.5533	49.1509	54.2081	58.3808	64.1046	68.5282	71.708
26.6368	32.6287	37.9718	44.4201	49.7035	53.8260	59.5269	65.0738	69.2736	71.760
27.7008	32.8027	38.2608	44.7105	49.8565	54.6043	60.0782	64.4457	68.9049	72.084
24.2393	29.2762	34.6726	41.4587	46.6163	50.8606	55.3171	61.0180	65.3578	68.296
24.6645	29.2827	34.6796	41.3580	45.6134	50.4465	55.1308	60.0666	64.3650	67.865
23.6814	28.3887	32.9197	39.9211	44.9901	49.2654	54.3413	59.5125	63.5923	66.986
22.1814	26.9362	32.7217	39.2189	43.9429	48.5553	54.1496	59.5674	63.7364	66.396
33.9204	38.7006	43.8334	50.1614	55.1376	56.6850	64.7803	72.2534	74.9263	77.509
35.0080	40.2529	46.7344	52.5591	56.4897	59.6003	65.8052	71.7036	75.1438	78.248

```
Atest = transpose(flatten_grayfaces_test) - mean_flatten_grayfaces_train
```

Atest = *356×4096*

27.9123	32.5339	37.6679	44.4552	49.9421	54.6866	59.5130	63.9126	68.5064	72.040
26.9680	31.9114	37.2975	44.2326	49.8006	54.2394	58.3281	63.7587	68.6921	72.193
26.8802	32.5557	38.7644	44.7091	49.9062	54.6712	59.9809	65.0381	68.7604	71.727
26.6703	31.7972	37.8292	45.3094	49.9598	54.2141	60.2303	65.1223	69.4215	72.087
24.8759	29.8202	35.0062	41.5898	46.1344	50.1889	55.1762	60.2522	64.5271	68.538
24.9739	29.0056	33.8640	41.4153	44.9720	48.8212	54.7041	60.9759	64.7674	68.037
23.7010	28.9058	34.6076	39.8787	44.9016	49.6628	54.6166	59.6042	63.5087	67.373
23.6785	28.8830	34.7591	39.6988	43.8284	48.6928	54.6309	59.9574	63.7188	67.088
33.7500	37.9407	43.5385	49.8635	54.6069	59.0107	64.8578	70.6340	74.4816	76.523
34.8007	39.2787	44.1763	50.3229	55.4512	60.6640	66.7164	73.0960	78.2234	76.929

```
% We technically do not need to mean it (because cov does it for us), but it helps me understand it
% better
cov_flatten_grayface_train = cov(A)
```

cov_flatten_grayface_train = *4096×4096*

10^3 ×

3.1279	3.3322	3.3206	3.2886	3.1939	3.0562	2.9494	2.7908	2.6874	2.504
3.3322	3.7768	3.8736	3.8767	3.7785	3.6372	3.5335	3.3621	3.2509	3.045
3.3206	3.8736	4.2659	4.4402	4.3204	4.1726	4.0921	3.9566	3.8502	3.634
3.2886	3.8767	4.4402	4.8970	4.9191	4.8184	4.7686	4.6583	4.5592	4.320
3.1939	3.7785	4.3204	4.9191	5.2672	5.3397	5.3205	5.1903	5.1015	4.876
3.0562	3.6372	4.1726	4.8184	5.3397	5.6517	5.7462	5.6414	5.5440	5.332
2.9494	3.5335	4.0921	4.7686	5.3205	5.7462	6.0527	6.0941	6.0290	5.833
2.7908	3.3621	3.9566	4.6583	5.1903	5.6414	6.0941	6.4035	6.4540	6.290
2.6874	3.2509	3.8502	4.5592	5.1015	5.5440	6.0290	6.4540	6.6808	6.614
2.5047	3.0453	3.6346	4.3202	4.8760	5.3327	5.8339	6.2900	6.6140	6.814

```
[eigsV , eigsD] = eigs(cov_flatten_grayface_train, 100)
```

eigsV = *4096×100*

0.0022	-0.0179	-0.0066	0.0161	-0.0172	0.0097	0.0164	-0.0080	0.0018	0.014
0.0028	-0.0207	-0.0107	0.0193	-0.0176	0.0110	0.0209	-0.0083	0.0063	0.015
0.0031	-0.0228	-0.0153	0.0226	-0.0200	0.0163	0.0251	-0.0086	0.0084	0.019
0.0033	-0.0251	-0.0199	0.0268	-0.0237	0.0181	0.0300	-0.0070	0.0168	0.022
0.0030	-0.0267	-0.0238	0.0309	-0.0261	0.0172	0.0308	-0.0031	0.0240	0.025
0.0025	-0.0275	-0.0272	0.0340	-0.0272	0.0153	0.0325	-0.0031	0.0268	0.029
0.0017	-0.0284	-0.0311	0.0357	-0.0298	0.0117	0.0373	-0.0027	0.0275	0.031
0.0010	-0.0293	-0.0360	0.0375	-0.0310	0.0101	0.0389	-0.0017	0.0253	0.034
0.0007	-0.0300	-0.0394	0.0394	-0.0311	0.0092	0.0400	0.0008	0.0268	0.035
-0.0004	-0.0298	-0.0432	0.0407	-0.0305	0.0099	0.0375	0.0051	0.0298	0.036

eigsD = *100×100*

10^6 ×

3.6568	0	0	0	0	0	0	0	0	
0	1.7611	0	0	0	0	0	0	0	
0	0	0.8720	0	0	0	0	0	0	
0	0	0	0.6034	0	0	0	0	0	
0	0	0	0	0.3736	0	0	0	0	
0	0	0	0	0	0.3111	0	0	0	
0	0	0	0	0	0	0.2381	0	0	
0	0	0	0	0	0	0	0.2086	0	
0	0	0	0	0	0	0	0	0.2073	

```
0        0        0        0        0        0        0        0        0    0.1734
```

With these eig values we can see tha the top few eigenvalues are significantly larger than other other eigenvalues

2. Project the training and test face images onto the k principal components. We'll call this the facespace representation of our original images.

```
PCV2 = eigsV(:, 1:2)
```

```
PCV2 = 4096×2
    0.0022   -0.0179
    0.0028   -0.0207
    0.0031   -0.0228
    0.0033   -0.0251
    0.0030   -0.0267
    0.0025   -0.0275
    0.0017   -0.0284
    0.0010   -0.0293
    0.0007   -0.0300
   -0.0004   -0.0298
```

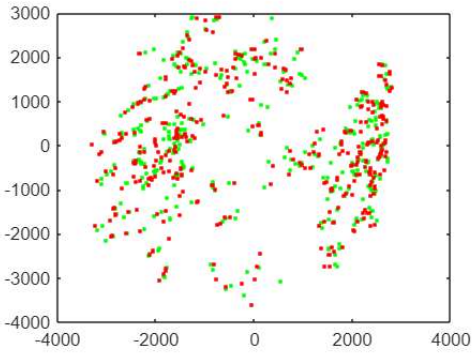
```
% 365 x 2          356 x 4096  4096 x 2
alphaAVisualize = A * PCV2
```

```
alphaAVisualize = 356×2
10^3 ×
   -0.7785   -0.9783
   -0.4016   -0.8350
   -0.8478   -0.8782
   -0.8354   -0.8756
    2.0993   -0.6499
    2.1232   -0.6381
    2.4411   -0.7091
    2.4722   -0.6046
    2.6592   -0.5438
    2.5817   -0.6348
```

```
alphaATestVisualize = Atest * PCV2
```

```
alphaATestVisualize = 356×2
10^3 ×
   -0.7945   -0.9971
   -0.5061   -0.8317
   -0.8222   -0.9292
   -0.8268   -0.8255
    2.0972   -0.6672
    2.0586   -0.6574
    2.4955   -0.6759
    2.4586   -0.6642
    2.6517   -0.5471
    2.5740   -0.7525
```

```
plot(alphaATestVisualize(:, 1), alphaATestVisualize(:, 2), '.', 'Color', 'g')
hold on
plot(alphaAVisualize(:, 1), alphaAVisualize(:, 2), '.', 'Color', 'r')
hold off
```



```
PCV2 = eigsV(:, 1:3)
```

```
PCV2 = 4096×3
    0.0022   -0.0179   -0.0066
    0.0028   -0.0207   -0.0107
    0.0031   -0.0228   -0.0153
    0.0033   -0.0251   -0.0199
    0.0030   -0.0267   -0.0238
    0.0025   -0.0275   -0.0272
    0.0017   -0.0284   -0.0311
    0.0010   -0.0293   -0.0360
    0.0007   -0.0300   -0.0394
   -0.0004   -0.0298   -0.0432
```

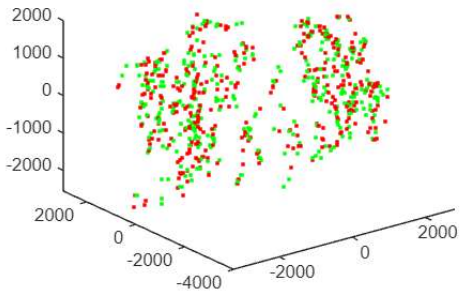
```
% 4096 x 2
alphaAVisualize = A * PCV2
```

```
alphaAVisualize = 356×3
10^3 ×
   -0.7785   -0.9783   -0.5968
   -0.4016   -0.8350    0.0232
   -0.8478   -0.8782   -0.4679
   -0.8354   -0.8756   -0.5686
    2.0993   -0.6499    0.5050
    2.1232   -0.6381    0.5194
    2.4411   -0.7091    0.4664
    2.4722   -0.6046    0.6526
    2.6592   -0.5438    0.8647
    2.5817   -0.6348    0.8415
```

```
alphaATestVisualize = Atest * PCV2
```

```
alphaATestVisualize = 356×3
10^3 ×
   -0.7945   -0.9971   -0.6148
   -0.5061   -0.8317   -0.0909
   -0.8222   -0.9292   -0.5227
   -0.8268   -0.8255   -0.4166
    2.0972   -0.6672    0.4367
    2.0586   -0.6574    0.4649
    2.4955   -0.6759    0.5211
    2.4586   -0.6642    0.5456
    2.6517   -0.5471    0.9781
    2.5740   -0.7525    1.0129
```

```
plot3(alphaATestVisualize(:, 1), alphaATestVisualize(:, 2), alphaATestVisualize(:,3), '.', 'Color', 'g')
hold on
plot3(alphaAVisualize(:, 1), alphaAVisualize(:, 2), alphaAVisualize(:,3), '.', 'Color', 'r')
hold off
```



Let k = 5 because I feel like it. This means that we are going to project 356 dimensions into a 5 dimensional space

```
k = 2
```

```
k = 2
```

```
PCV = eigsV(:, 1:k)
```

```
PCV = 4096×2
    0.0022   -0.0179
    0.0028   -0.0207
    0.0031   -0.0228
    0.0033   -0.0251
    0.0030   -0.0267
    0.0025   -0.0275
    0.0017   -0.0284
    0.0010   -0.0293
    0.0007   -0.0300
   -0.0004   -0.0298
```

```
% 4096 x k
alphaA = A * PCV % DONT USE THIS, just to see
```

```
alphaA = 356×2
10^3 ×
   -0.7785   -0.9783
   -0.4016   -0.8350
   -0.8478   -0.8782
   -0.8354   -0.8756
    2.0993   -0.6499
    2.1232   -0.6381
    2.4411   -0.7091
    2.4722   -0.6046
    2.6592   -0.5438
    2.5817   -0.6348
```

```
alphaATest = Atest * PCV
```

```
alphaATest = 356×2
10^3 ×
   -0.7945   -0.9971
   -0.5061   -0.8317
   -0.8222   -0.9292
   -0.8268   -0.8255
    2.0972   -0.6672
    2.0586   -0.6574
    2.4955   -0.6759
    2.4586   -0.6642
    2.6517   -0.5471
    2.5740   -0.7525
```

What does this info mean?

This means that we disregard 4096 - 20 = 4076 eigenvectors because there are 5 faces that can carry a majority/enough of the variance in order to predict a face.

3. For each of the test images, compute the closest match between the test image (represented as a k-dimensional vector in facespace) and the training images (again, in facespace). The notion of “closest match” here can be described in a few different ways, but the easiest thing to do is to use the the Euclidean distance to define how far apart two points are.

(Euclidean distance is just to find how far away two points, squaring them, then squarerooting the sum.)

We can see how far away by solving for euclidean distance

```
% Euclidean distance
%distance1 = sqrt(sum((reconstructedData - flatten_grayfaces_test).^2))
% n^2 complexity time

alphaA % 4096x k
```

```
alphaA = 356×2
10^3 ×
   -0.7785   -0.9783
   -0.4016   -0.8350
   -0.8478   -0.8782
   -0.8354   -0.8756
    2.0993   -0.6499
    2.1232   -0.6381
    2.4411   -0.7091
    2.4722   -0.6046
    2.6592   -0.5438
    2.5817   -0.6348
```

```
alphaATest % 4096x k
```

```
alphaATest = 356×2
10^3 ×
   -0.7945   -0.9971
   -0.5061   -0.8317
   -0.8222   -0.9292
   -0.8268   -0.8255
    2.0972   -0.6672
    2.0586   -0.6574
    2.4955   -0.6759
    2.4586   -0.6642
    2.6517   -0.5471
    2.5740   -0.7525
```

```
answer = zeros(356, 1);
lowestDistance = 10000000;
for test = 1 : 356 % 356 faces
    for train = 1: 356 % 356 faces
        % project the train data into a 5d space
        % 1 x 356 356 x k = 1 x k

        % project the test data into a k dimensional space
        projectedTest = alphaATest(test,:);
        projectedTrain = alphaA(train,:);
        %calculate the distance
        distance = sqrt(sum((projectedTrain-projectedTest ).^2));
        % if distance is less than lowestDistance
        if distance < lowestDistance
            lowestDistance = distance;
            trainPoint = train;
        end

    end
    answer(test) = trainPoint; % training data with lowest distance being linked with the test point
    lowestDistance = 100000000;
    trainPoint=-1;
end

% Calculate the euclidean distance (is literally just distance) from the one test point and find the
% N^2 complexity, compare each test point to every train point and the test
% point will guess that it is the closest one.
```

In this way, you would look for the training point that has the smallest Euclidean distance for a particular test point and predict the identity of the test point to be the same as the identity of this closest training point. This method of classification is known as nearest neighbor classification and it is the one new concept you need to implement Eigenfaces

```
num_matches = sum(ceil(answer/4) == subject_test) % The reason for the ceil(ans/4) is because the 4 photos are of the same person but ans
```

```
num_matches = 270
```

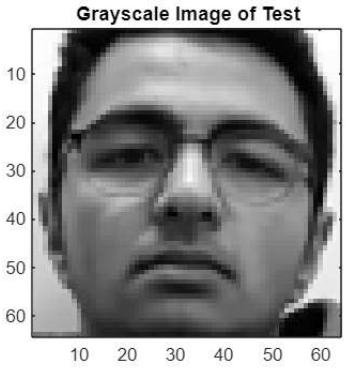
```
% with 20 pcv, it has 100% match. but also i think the photos are the same
accuracy = num_matches/356
```

```
accuracy = 0.7584
```

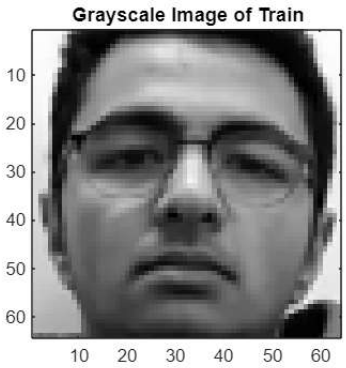
```
% Create a sample 64x64 matrix with values ranging from 0 to 255
image = 1
```

```
image = 1
```

```
matrixTest = grayfaces_test(:,:,image);

% Display the matrix as a grayscale image
figure;
imagesc(matrixTest);
% Set the colormap to grayscale
colormap(gray);


% Adjust axis properties
axis image; % Ensures that pixels are square
title('Grayscale Image of Test');
```




```
% Create a sample 64x64 matrix with values ranging from 0 to 255
matrixTrain = grayfaces_train(:,:,answer(image));

% Display the matrix as a grayscale image
figure;
imagesc(matrixTrain);
% Set the colormap to grayscale
colormap(gray);


% Adjust axis properties
axis image; % Ensures that pixels are square
title('Grayscale Image of Train');
```




```

```

```
num_matches = 356
accuracy = 1
```

```
for
```