EE417 - Assignment 10 - Post-Lab Report

Baris Sevilmis
Lab Date: 25/12/2018
Due Date: 31/12/2018

Overview

Tenth lab assignment of EE417 focuses on Face Detection, Face Reconstruction and Face Recognition by the method of Principal Component Analysis, to be more specific Eigenfaces technique. Following sections will be according to Face Detection, Face Reconstruction and Face Recognition. Euclidean and Mahalonabis distances will be used as distance metrics, such that distance metrics could be compared in terms of their accuracy.

Face Detection

Face Detection is the first phase of face recognition problem. This step will be mainly based on eigenfaces algorithm over a dataset of 400 images. Each image is size of $56x46$ and in addition, images are flattened beforehand to a single dimensional array of 2576. Total data size will be $2576x400$ and first 360 columns will be used as our training set. Figure 1 provides some of the image within the dataset. Rest of the 40 columns will be used for face recognition and face reconstruction problems in the further sections of this report.

Figure 1: Portion of the Dataset

Algorithm starts by obtaining and flattening the training face images$(I_1, I_2, ..., I_M)$ such that all face images are centered and of the same size. As soon as these images are flattened into $1 - D$ every image I_i turn into vectors, namely Γ_i. Listing 1 ensures mentioned two consecutive steps of the eigenface algorithm as well as the code for visualizing training set images as in Figure 1.

Listing 1: MatLab Code Face Image Partitioning and Visualization

```
load('faces.mat');
face = faces;
[row, col, num] = size(face);
train_face = face(:,1:360);
[rowtr, coltr, numtr] = size(train_face);
test_face = face(:,361:400);
[rowte, colte, numte] = size(test_face);
vis = zeros(56,46,1,12);
count= 1;
for jj = 1:8:8*16
    vis(:,:,:,count)=uint8(reshape((train_face(:,jj)),[56, 46]))
        ;
    count = count + 1;
end
vis = uint8(vis);
figure; montage(vis, 'Size', [4 4]); title(Training set of Faces);
```

Algorithm proceeds by computing average face vector of the training set, which is denoted by Ψ as in Listing 2. Listing 2 provides also the normalization of the face vectors by simply subtracting average face vector from the rest of the images and normalized face vectors are denoted by Φ_i as i refers to the sample image. Vectorization of computation speeds up the algorithm. Recovered Ψ and Φ normally computed as follows:

$$\Psi = \frac{1}{M}\sum_{i=1}^{M}\Gamma_i$$

$$\Phi_i = \Gamma_i - \Psi$$

Figure 2 depicts the average face image of the 360 image training set. For simplicity, average face subtracted versions of face images will be called as Mean Faces.

Listing 2: MatLab Code for Average Face Computation and Mean Faces

```
%% Average Face and Mean Faces
train_face = double(train_face);
avg_face = sum(train_face,2)./coltr;
normtrain_face = train_face - avg_face;
figure; montage(uint8(reshape((avg_face-min(avg_face))/(max(
    avg_face)-min(avg_face)), [56, 46])*255));
title(Average Face);
```

Figure 2: Average Face

As the next step covariance matrix C is computed in the following manner:

$$C = \frac{1}{M}\sum_{n=1}^{M}\Phi\Phi^T = AA^T -> (N^2XN^2)$$

$$A = [\Phi_1, \Phi_2, ..., \Phi_M] -> (N^2XM)$$

However, C matrix has size of (N^2XN^2). Therefore, for efficiency transpose of this operation is taken into consideration, since $M << N^2$. To be more specific, following operation takes place, as the resulting M eigenvectors correspond to the largest eigenvectors of AA^T instead of N^2 eigenvectors:

$$C = A^TA -> (MXM)$$

Listing 3 ensures the Covariance Matrix calculation, as every column is again normalized.

Listing 3: MatLab Code for Covariance Matrix

```
%% Covariance Matrix
C = normtrain_face'*normtrain_face/coltr;
```

Largest K eigenvectors are picked out of M eigenvectors. Nonetheless, $K << M << N^2$ is a condition that is already determined. K is chosen according to the eigenvalue magnitudes, as can be seen in Figure 3. For the sake of convenience, K is picked as 50. Finally, we multiply Mean Faces with resulting KXK covariance matrix in terms of finding the eigenfaces in the following way:

$$\hat{\Phi}_i - mean = \sum_j = 1^K w_j u_j, (w_j = u_j^T\Phi_i)$$

Listing 4 provides the corresponding MatLab code of this phase including the normalization of eigenfaces. u_i denotes the eigenfaces.

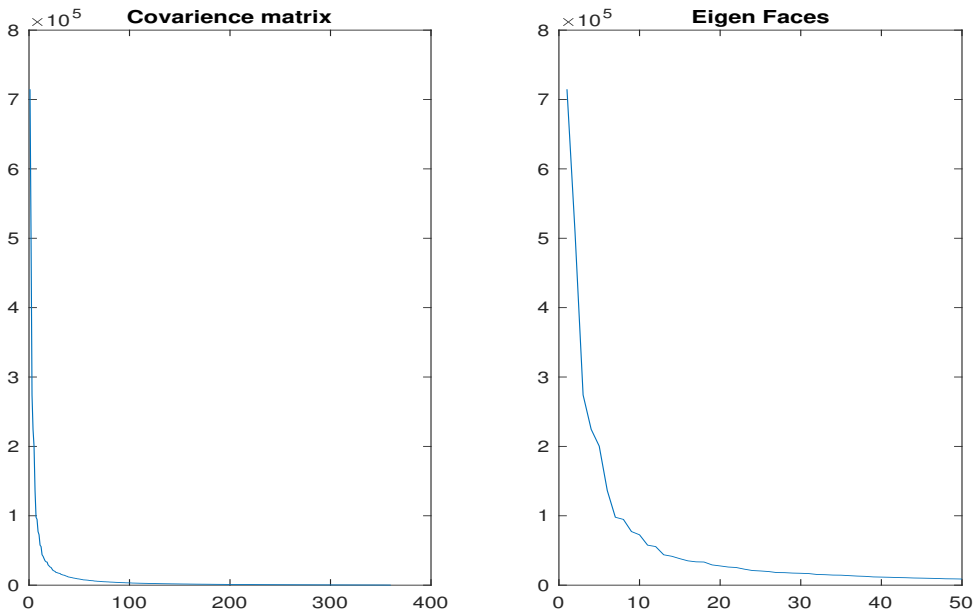

Figure 3: Covariance Matrix of M and K

Listing 4: MatLab Code for Eigenface Computation

```
%% Calculate K Eigenvalues and Eigenvectors
% Calculate Vectorwise Norm of a Matrix
% Test for K values
figure;
[V1, D1] = eigs(C, size(C,1));
subplot(1,2,1);plot(1:360, max(D1));
title(Covarience matrix);

% Choose K
K = 50;
[V2, D2] = eigs(C, K);
subplot(1,2,2);plot(1:K, max(D2));
title(Largest K of Covariance matrix);
new_V = (normtrain_face*V2);
N = vecnorm(normtrain_face*V2);
new_V = new_V ./ N;
```

For convenience, these face vectors are visualized with the code block in Listing 5. Figure 3 demonstrates the first 12 eigenfaces.

Listing 5: MatLab Code for Eigenface Visualization

```
%% Show first 12 eigenfaces
% Reshape a flattened image to show it
Image = zeros(56,46,1,12);
for jj = 1:12
    Image(:,:,:,jj)=uint8(reshape((new_V(:,jj)-min(new_V(:,jj)))
        /(max(new_V(:,jj))-min(new_V(:,jj))), [56, 46])*255);
end
Image = uint8(Image);
% Display multiple image frames as rectangular montage
figure; montage(Image, 'Size', [2 6]); title(Eigen Faces);
```

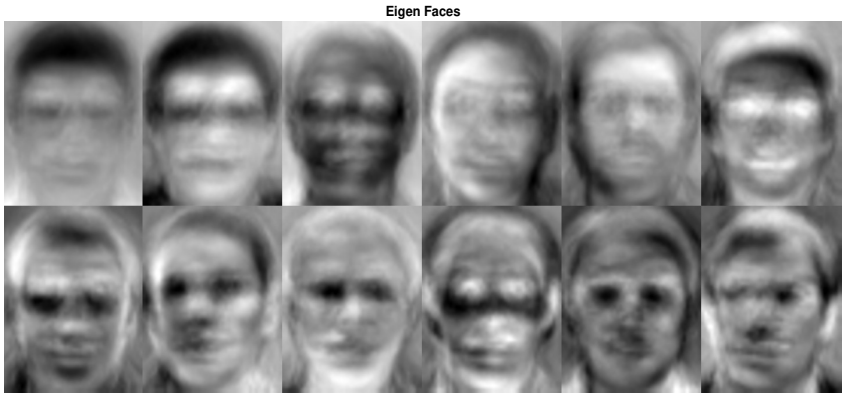

Figure 4: First 12 Eigenfaces

Face Reconstruction

Face reconstruction is applied on test set images as they were not used for the training, they will show reliability of the algorithm. As mentioned in the previous section, u_i denotes the computed eigenfaces and $\hat{x}$ indicates average face. On the other hand, x will be denoting our new test image. Following formula is used to construct reconstruction parameters a_i and second formula denotes the reconstruction error:

$$a_i = (x - \hat{x})^T u_i$$

$$||x - (\hat{x} + a_1 u_1 + a2u2 + ... + a_k u_k)|| < Threshold$$

Threshold error is picked by the user to eliminate non face images. Error value of these non face images will be relatively higher. Listing 6 ensures reconstruction of test faces with error detection.

Listing 6: MatLab Code for Face Reconstruction

```
%% Reconstruction
test_face = double(test_face);
thr = 5.0;

normtest_face = test_face - avg_face;
a = normtest_face' * new_V;
recon_face = avg_face + (new_V*a');
recon_final = recon_face - test_face;
er = sum(sqrt((recon_final).^2),1);

%Test sample m
m = 42;
if er(m) < thr
    figure; montage(uint8(reshape(test_face(:,m), [56, 46])));
    figure; montage(uint8(reshape((recon_face(:,m)-min(
        recon_face(:,m)))/...
        (max(recon_face(:,m))-min(recon_face(:,m))), [56, 46])
        *255));
else
    figure; montage(uint8(reshape(test_face(:,m), [56, 46])));
    figure; montage(uint8(reshape((recon_face(:,m)-min(
        recon_face(:,m)))/...
        (max(recon_face(:,m))-min(recon_face(:,m))), [56, 46])
        *255));
end
```

Figure 5 & 6 provides reconstruction of face images. On the other hand, Figure 7 provides reconstruction of a flower object into a face, therefore reconstruction error relatively is high.

(a) Original Image (b) Reconstructed Image

Figure 5: Reconstruction of a flower image in terms of face images

(a) Original Image

(b) Reconstructed Image

Figure 6: Reconstruction of a flower image in terms of face images

(a) Flower Image

(b) Reconstructed Image

Figure 7: Reconstruction of a flower image in terms of face images

Face Recognition

Last phase is the face recognition phase, in which test images are detected and reconstructed beforehand. Firstly, test image is turned into Mean Face form, where it is being normalized. Test image is denoted as Φ. Following formula is used:

$$\Phi = \Gamma - \Psi$$

Φ is projected on the eigen space as follows:

$$\hat{\Phi} = \sum_{i=1}^{K} w_i u_i, (w_i = {}^{T}_{i}\Phi)$$

Φ is represented in the following form:

$$\Phi : \Omega = [w_1, w_2, ..., w_k]$$

Error is calculated in the following form:

$$e_r = \min_{l} |||\Omega - \Omega^l||$$

If $e_r < Threshold_r$, than the test image is recognized as face l from the training set. This error calculation is normally done with euclidean distance metric, but for better results Mahalanobis distance will be used also, since Mahalanobis distance treats variations along all axes equally. Mahalanobis distance uses the following formula:

$$||\Omega - \Omega^k|| = \sum_{i=1}^{K} \frac{1}{\lambda_i}(w_i - w_i^k)^2$$

Listing 7 provides the MatLab code for Face Recognition. Figure 8, Figure 9 and Figure 10 demonstrates results of the algorithm on the provided dataset using euclidean distance metric.

Listing 7: MatLab Code for Face Recognition

```
%% Face Recognition
a_tr = normtrain_face' * new_V;
train_db = avg_face + new_V*a_tr';
euclidean = sqrt(sum((train_db-test_face(:,m)).^2));
%mahalanobis = sum(sqrt(1/new_V(m,m))*((train_db-test_face(:,m))
    .^2));
minInd = find(euclidean == min(euclidean(:)));
figure;subplot(1,2,1);montage(uint8(reshape((test_face(:,m)),
    [56, 46])));
title(Test Image);
subplot(1,2,2);montage(uint8(reshape((train_db(:,minInd)-min(
    train_db(:,minInd)))/...
     (max(train_db(:,minInd))-min(train_db(:,minInd))), [56, 46])
        *255));
title(Recognized Image);
```

Figure 8: Recognition on Test Face-1-Euclidean

Figure 9: Recognition on Test Face-2-Euclidean

Figure 10: Recognition on Test Face-3-Euclidean

Figure 11, 12 and 13 demonstrate the same test images above but uses Mahalanobis distance. Unfortunately, euclidean distance metric works as good as Mahalanobis distance metric, therefore resulting images are same. However, this implies that for our image dataset both of the distance metrics provide decent results, but as said normally mahalanobis distance provides better recognition than euclidean.

Figure 11: Recognition on Test Face-1-Mahalanobis

Figure 12: Recognition on Test Face-2-Mahalanobis

Figure 13: Recognition on Test Face-3-Mahalanobis

Lastly, my own images were provided to algorithm to test algorithms reliability. Figure 14 depicts my own images that have been allocated into train set. In addition, there is an additional image that is allocated into the test set for testing.

Figure 14: My Own Training Set Images

Figure 15 depicts recognition result with euclidean distance metric and Figure 16 shows the result of mahalanobis distance metric. Euclidean distance metric ensures exact result in our case, therefore advantage of the mahalanobis distance can not be visualized. However, mahalanobis distance metric provides better approximation for variations.

Figure 15: Recognition on Test Face-Euclidean

Figure 16: Recognition on Test Face-Mahalanobis