Principle Component Analysis for Eigenfaces

Shen Liu

A53094622

3/22/2017

Computer Language:

Matlab R2017a

Objective

The goal of this project is to implement the ideas in the reference paper to understand the PCA for Eigenfaces. Principle Component Analysis(PCA) is a technique to transform or rotate the original dimension to a new coordinate system such that it emphasizes the variation of the dataset and reduces the correlation between datasets to achieve dimension reduction.

In this project, the given training dataset are the pixels of 190 people's face images (source: http://fei.edu.br/~cet/facedatabase.html), and the goal is to compute the Eigenfaces, or the principal components(PCs) from the training data so that we can reconstruct any faces using different number of PCs. Since the dataset in the new coordinate system will have less correlation between each pair dataset, we can discard the principal components or Eigenfaces with lower variance among dataset to achieve dimension reduction.

Background

The Eigenfaces is in fact a set of eigenvectors which are widely used in the area of human face recognition. This idea was first introduced by Sirovich and Kirby(1987,1990) and further implemented by Matthew Turk and Alex Penland in their paper.[1] Unlike a popular approaches for facial recognition which focus on detecting features such as noses, mouth and their relative position, the idea of Eigenfaces is to use Principal Component Analysis(PCA) on a set of training face images to form a set of face basis, and we can then linearly combine the face basis to achieve face reconstruction.[1] These face basis can be thought as a set of features that characterize the variation between face images rather than the intuitive notion of face features such as eyes and noses.[1] The Eigenfaces model can recognize new faces by training on a limited number of characteristic faces. This way pf approach in facial detection and recognition has proved to be fast and insensitive to small changes in the face image [1], and it also has low space complexity by reducing dimension.

Approach / Method

The following procedures describe how the Eigenfaces(PCs) are generated in the project [1]:

1. Given each NxM training face data, we first convert it into a NMx1 vector to reduce its dimensionality space.
2. Compute an average face among 190 training faces

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

   Γ_i denotes the NMx1 vector corresponding to the NxM face data and M=190.
3. Subtract the mean from each face vector: $\Phi_i = \Gamma_i - m$
4. Compute the covariance matrix C of training face vectors:

$$C = \frac{1}{M}\sum_{i=1}^{M}\Phi_i\Phi_i^T = AA^T$$

   Where $A = [\Phi_1, \Phi_2, ..., \Phi_M]$. However, the covariance matrix C will have a dimension of N^2xN^2 which involves a huge matrix computation. Instead, we can consider matrix A^TA (MxM) which usually is much smaller than N^2xN^2 matrix.
5. Compute the eigenvalues λ_i and the corresponding eigenvectors v_i of A^TA:

$$A^TAv_i = \lambda_i v_i$$

   This can be computed using Matlab built-in function *eig*.
6. We can then form the M Eigenfaces u_i as the following:

$$AA^TAv_i = \lambda_i Av_i$$
$$CAv_i = \lambda_i Av_i$$

   Then,

$$u_i = Av_i$$

$$u_i = \sum_{k=1}^{M} v_{ik}\Phi_k \quad i = 1, ..., M$$

   Notice that the M eigenvectors of A^TA are the first M out of N^2 eigenvectors of AA^T which is also known as the Eigenfaces.
   The Eigenfaces u_i are the principle components(PCs) of the training faces. We can perform dimension reduction by selecting PCs that have the largest variations among dataset along its dimension.

After computing the Eigenfaces u_i, we can reconstruct a test face Γ by conducting the following:

Suppose we select K best Eigenfaces which corresponds to K largest eigenvalues. Then we project the test face onto each selected Eigenfaces and linearly combine the Eigenfaces to reconstruct a face:

$$\omega_i = u_i^T(\Gamma - m)$$ (Normalize each Eigenface such that $\|u_i\| = 1$)

$$\Phi_f = \sum_{i=1}^{K} \omega_i u_i$$

Note that we need to add the average face back in order to obtain the correct reconstructed the face.

For simplicity and comparison purpose, we will compute MSE between pixels of original face and reconstructed face to test the accuracy of the model.

$$MSE = \frac{1}{MN}\left\|\Phi_f - \Phi\right\|^2$$

Discussions & Results

For Eigenfaces, we need to select Eigenfaces such that the variation of the dataset is the largest along its dimension. A shortcut is to select Eigenfaces with the largest eigenvalues.

The follow image shows the average face for 190 training data.

Figure2. The average face

1) **Reconstruct one of 190 people's neutral expression image using different number of PCs:**

Figure 3. One of 190 people's neutral expression image

Figure 4. Reconstructed face using best K PCs (K = 10,20,…,190 from top left to bottom right)

K	10	20	30	40	50	60	70	80	90	100
MSE	68.02	61.66	59.78	55.69	51.26	47.28	43.49	38.99	34.87	33.36
K	110	120	130	140	150	160	170	180	190	
MSE	30.29	26.95	20.33	14.79	12.85	9.77	4.03	2.31	1.13	

Table 1. MSE between original face and reconstructed face with K = 10,20…190

From figure 4, we can almost perfectly reconstruct one of 190 trained faces by selecting best K PCs (K = 10,20,…,190 from top left to bottom right). The more PCs we select, the more detailed and accurate we can reconstruct the face. When K = 80 will already give us a pretty decent result as shown in the table 1.

2) Reconstruct one of 190 people's smiling expression image using different number of PCs

Figure 5. One of 190 people's smiling expression image

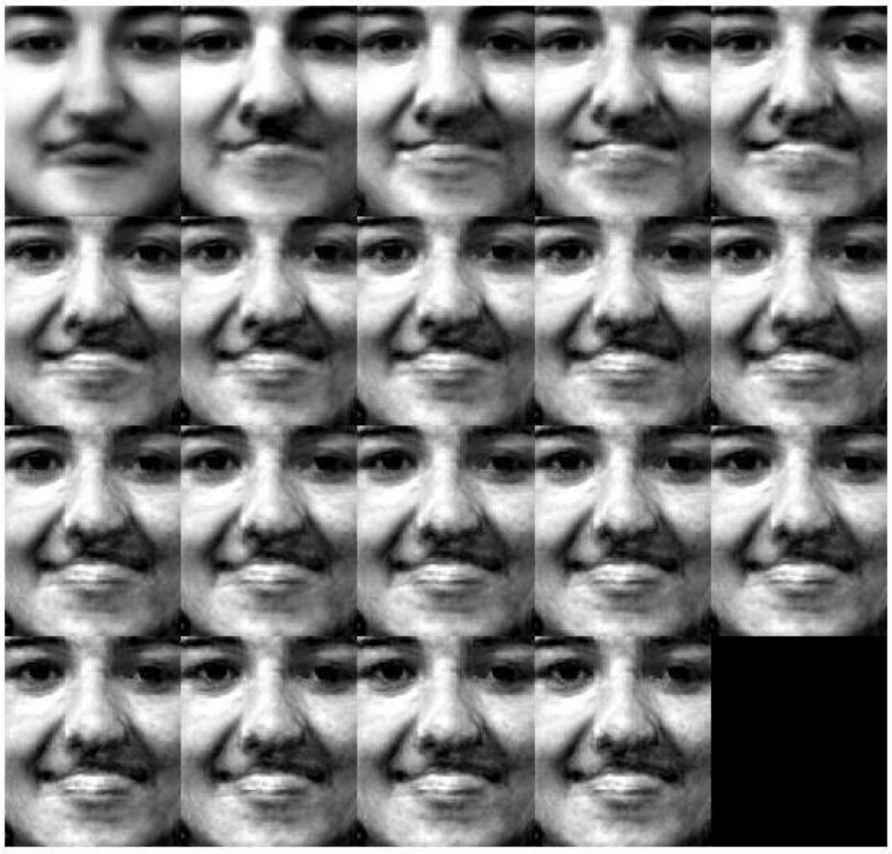


Figure 6. Reconstructed smiling face using best K PCs(K = 10,20,…,190 from top left to bottom right)

The result of reconstructing a smiling face in figure 6 is not as good as the result shown in figure 4. This is because we never train a smiling face during training, therefore the model had a hard time in recovering the mouth and the skin around it. And the less the K is, the less detailed the recovered face will be.

3) Reconstruct one of the other 10 people's neutral expression image using different number of PCs

Figure 7. One the other 10 people's neutral expression image

Figure 8. Reconstructed new face using best K PCs(K = 10,20,…,190 from top left to bottom right)

K	10	20	30	40	50	60	70	80	90	100
MSE	72.93	73.94	67.39	64.27	63.37	62.88	60.93	60.47	59.94	58.29
K	110	120	130	140	150	160	170	180	190	
MSE	58.11	57.73	56.79	56.49	56.04	55.82	55.60	55.11	55.08	

Table 2. MSE between original face and reconstructed face with K = 10,20…190

The result for reconstructing a new face using Eigenfaces is also decent as shown in the figure 8 and table 2. However, comparing table 1 & 2, we can conclude that our model performs better to a trained face than to a new face.

The follow figure shows the reconstructed face for all 10 new faces with 80 best Eigenfaces selected.

4) Use other non-human image (e.g., car image, resize and crop to the same size), and try to reconstruct it using all PCs

Figure 9. a car image(left); reconstruction of the car image using all PCs(right)

Since we use only human faces during training, our model react poorly to a test car image for reconstruction. As we observe from figure 9, our model tries to reconstruction a car image back to a human face which leads to a weird result.

5) **Rotate one of 190 people's neutral expression image with different degrees and try to reconstruct it using all PCs**

Figure 10. One of 190 people's neutral expression image with 10 degree of rotation(left); reconstruction of the rotated image with all PCs(right).

If we rotate a trained image by a degree of 10, we cannot get back our original image using the Eigenfaces. This is because we did not use any rotated face during training, and our model still treats the test face without any rotation which leads to an inaccurate reconstruction.

Denoising

6) **Adding noise to one of the other 10 people's neutral expression image, and reconstruct the image using different number of PCs. Try different level of Gaussian noise.**

Figure 11. One of the other 10 people's neutral expression image with Gaussian noise (variance = 0.01, 0.1 & 0.5 for left, middle and right)

Figure 12. Reconstruction of Gaussian noisy face with variance = 0.01

Figure 13. Reconstruction of Gaussian noisy face with variance = 0.1

Figure 14. Reconstruction of Gaussian noisy face with variance = 0.5

K	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
v = 0.01	93.34	93.81	92.14	91.64	91.36	90.43	89.85	89.76	89.27	88.74
v = 0.1	113.08	113.35	113.22	113.56	113.56	113.13	113.08	113.27	113.13	112.98
v = 0.5	119.63	119.78	119.91	120.50	120.64	120.39	120.60	120.77	120.71	120.77
K	**110**	**120**	**130**	**140**	**150**	**160**	**170**	**180**	**190**	**110**
v = 0.01	88.74	88.73	88.49	88.24	88.36	88.36	88.31	88.30	88.11	88.74
v = 0.1	112.96	112.89	112.98	113.01	112.96	112.89	112.98	113.13	113.12	112.96
v = 0.5	120.85	120.91	120.88	120.93	120.94	120.88	121.01	121.12	121.00	120.85

Table 3. MSE between original noisy face and reconstructed face with different variance level

With different level Gaussian noise added on the face image, we can reconstruct the original image by eliminating most of the noise with different number of PCs. However, the higher the noise level is, the harder we can get back our original image since we lose more pixel information as noise level increases.

As the figure 12-14 and table 3 shows, as the variance of Gaussian noise increases, MSE also increases.

7) **Recompute the PCs using first 190 people's neutral expression image, but with first person's image contaminated by noise. Try to reconstruct that person's noisy image using different number of PCs**

Figure 15. first person with noise added

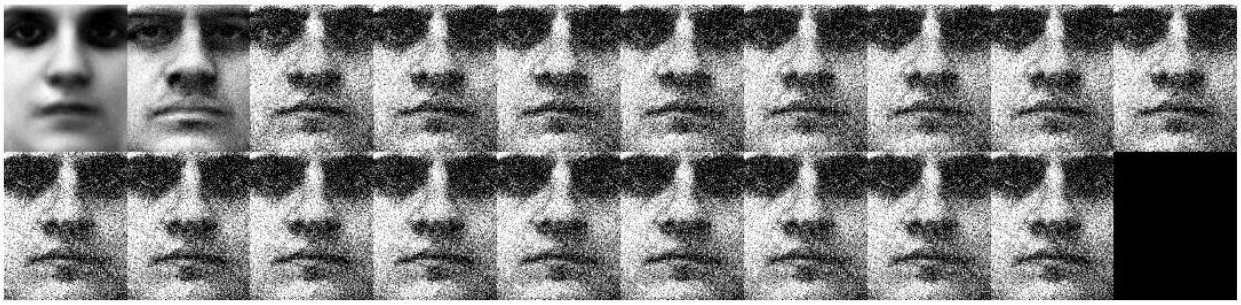


Figure 16. Reconstruction of first person with Gaussian noise using best K PCs(K = 10,20,…,190 from top left to bottom right)

As shown in figure 16, If we train the first person with noise and try to reconstruct the same noisy image, the noise is not removed at all when using all PCs. This is because when we are training this noisy image, the noise are uniformly distributed into all dimensions and it should not have a relative higher or lower variance in one dimension. Therefore, when we are try to recover the noisy face, since the noisy face is trained into all Eigenfaces, we will recover the same noisy face with all PCs.

SVM with Eigenfaces for smiling face detection

In addition, I tried to use Support Vector Machine combine with Eigenfaces reconstruction to detect whether the test face is a smiling face or a neutral expression face. The procedures show as follow:

1. Train 100 (1 to 100) neutral expression faces to obtain Eigenfaces.
2. Reconstruct 50 new faces with neutral expression (101a to 150a) using Eigenfaces, and reconstruct 50 new smiling faces(101b to 150b) using Eigenfaces.

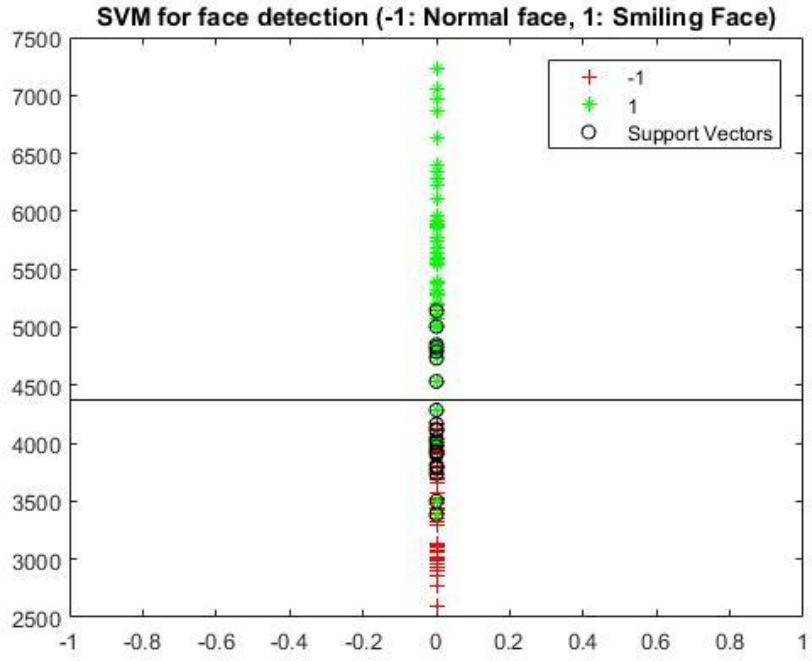
3. Compute the Euclidean distance between original faces and reconstructed faces:

$$\left\|\Phi_f - \Phi\right\|_2$$

There a total 100 Euclidean distances and we use those data to train our SVM model.

4. Use another 50 neutral expression faces (151a to 200a) and 50 smiling faces(151b to 200b) as the test data to compute the Euclidean distance and test our SVM model.

The follow figure shows how our SVM model and training data look like by selecting 100 Eigenfaces for test faces reconstruction:



The following table shows the accuracy percentage of our SVM model for testing over 100 new faces (50 neural expression face and 50 smiling face) with different number PCs selected.

Number of PCs	100	50	10	5
Accuracy percentage	85%	87%	83%	84%

As we can see the accuracy of detecting if a face is smiling or not does not vary a lot when number of PCs is decreased.

Summary & Discussion

From the experiment, we conclude that we can use Eigenfaces computed from training faces to reconstruct an old face or a new face nicely even with a large reduction of the dimension. The reconstruction of an old face has a better result than an new face (According to their MSE). As long as we choose a number of PCs with largest variance (e.g. K = 80), we can decently reconstruct the faces. The more PCs we choose, the more accurate the reconstruction is. Further, we can remove the noise from the image during reconstruction while maintain a good quality of the image. However, if the test image is rotated or a non-human image, our model cannot react properly since we train our model using only human faces without rotation. A solution to solve this issue is to include rotated human faces or non-human images during training session. Even though this would increase the computational cost during training, our model has advantage in detecting new faces for its speed and simplicity. Therefore, the idea of Eigenfaces can be also used further in facial recognition and detection while maintain a low computational cost and a low storage. The SVM model combine with the Eigenfaces reconstruction in the experiment has shown a decent result for smiling face detection.

Reference

Paper:

1. M. Turk and A. Pentland, 1991. Eigenfaces for recognition, Journal of Cognitive Neuroscience 3:71-86.

Dataset:

2. http://fei.edu.br/~cet/facedatabase.html

Code (Matlab R2017a)

Functions:

```
function [ eigenFaces_norm, average_face ] = trainFaces
%   Description:
%   This function reads and trains 190 people's neutral expression images
%   and generate the normalized eigenfaces(or Principle Components).
%
%   Input: null
%   Output: eigenFaces_norm: The normalized eigenFaces
%               average_face: The average face among all faces

%% Reading training data
num = 190; % number of training faces
[row,col] = size(imread('1a.jpg'));
faces_array = zeros(row*col,num); % store all faces as array

for i = 1:190 % read first 190 training faces into array
    temp_img = imread([num2str(i),'a.jpg']);

    % Adding noise to first person's face
    %if i == 1
    %  temp_img = origin_face;
    %end

    temp_arr = temp_img(:);
    faces_array(:,i) = temp_arr;
end

% Initializing faces data
average_face = (sum(faces_array') / num)'; % compute an average face
faces_data = faces_array - average_face*ones(1,num); % each face minus mean

%% Training faces to produce eigenfaces
C = cov(faces_data); % covariance of faces
[eigenVectors, eigenValues] = eig(C); % Compute eigenvalues and eigenvectors
eigenFaces = faces_data * eigenVectors; % construct eigenfaces(PCs)
```

```
% Normalize each eigenface
eigenFaces_norm = zeros(row*col,num);
for i = 1:num
    temp_face = eigenFaces(:,i);
    norm_factor = norm(temp_face); % compute the 2-norm of each eigenface
    temp_face_norm = temp_face / norm_factor; % normalize eigenface
    eigenFaces_norm(:,i) = temp_face_norm;
end

%% Plotting average faces
avg_face = reshape(average_face,[row,col]);

figure;
imshow(uint8(avg_face))
title('The average face')

end
```

```
function [ reconstruct_face ] = reconstructFace( origin_face, eigenFaces, K,
average_face )
%   Description:
%   This function reconstruct the input face as a matrix using eigenFaces
%
%   Input: origin_face:  The test face matrix for reconstruction
%          eigenFaces:  The principal components for face reconstruction
%                   K:  The number of PCs used to reconstruct the face
%        average_face:  The average face among all faces
%   Output: reconstruct_face: The reconstructed face in a matrix

%% Reconstructing the test face
num = 190; % number of training faces
[row,col] = size(origin_face);

% Preprocessing the test face data
origin_face_double = double(origin_face);
test_face = origin_face_double(:); % convert nxn image into n^2x1

% Uses only first K number of PCs for reconstruction (Dimension Reduction)
eigenFaces = eigenFaces(:,num-K+1:num);

% Reconstructing face
face_proj = (test_face - average_face)' * eigenFaces; % projection of data
onto each eigenfaces
reconstruct_array = eigenFaces * face_proj' + average_face; % reconstruct the
original face

% Convert reconstructed face from n^2x1 to nxn image
reconstruct_face = reshape(reconstruct_array, [row, col]);


end
```

Main

```
% ---------------------------- PCA ---------------------------- %
clc;close all;clear all;
addpath('faces')
%% Reading test faces
neutral_train_face = imread('1a.jpg');
smile_train_face = imread('123b.jpg');
neutral_test_face = imread('197a.jpg');
neutral_train_face_rotate = imrotate(neutral_train_face,10,'crop');

[x, y] = size(neutral_train_face);
car = rgb2gray(imread('car.jpg'));
car = imresize(car, [x,y]); % resize image

%% Training Faces to obtain Eigenfaces
% compute eigenFaces and an average face
[eigenFaces, average_face] = trainFaces();


%% 1) Reconstructing trained netural expression face using k PCs
for k = 10:10:190
    % call reconstructFace function
    reconstruct_face = reconstructFace(neutral_train_face, eigenFaces, k,
average_face);
    [row,col] = size(reconstruct_face);

    % combine all reconstructed faces together
    i = floor((k-10)/10/5);
    j = mod((k-10)/10,5);
    combi_face1(row*i+1:row*(i+1),col*j+1:col*(j+1)) = reconstruct_face;

    % compute the MSE between original face and reconstructed face
    mse(1,k/10) = sum(sum((uint8(reconstruct_face) -
neutral_train_face).^2))...
        /(col*row);
end

figure;
imshow(neutral_train_face)
title('One of 190 faces with neutral expression')
figure;
imshow(uint8(combi_face1))
title('Reconstructed face')


%% 2) Reconstructing smiling face using k PCs
for k = 10:10:190
    % call reconstructFace function
    reconstruct_face = reconstructFace(smile_train_face, eigenFaces, k,
average_face);
    [row,col] = size(reconstruct_face);

    % combine all reconstructed faces together
    i = floor((k-10)/10/5);
```

```
    j = mod((k-10)/10,5);
    combi_face2(row*i+1:row*(i+1),col*j+1:col*(j+1)) = reconstruct_face;
end

figure;
imshow(smile_train_face)
title('One of 190 faces with smile')
figure
imshow(uint8(combi_face2))
title('Reconstructed smiling face')


%% 3) Reconstructing non-trained face using k PCs
for k = 10:10:190
    % call reconstructFace function
    reconstruct_face = reconstructFace(neutral_test_face, eigenFaces, k,
average_face);
    [row,col] = size(reconstruct_face);

    % combine all reconstructed faces together
    i = floor((k-10)/10/5);
    j = mod((k-10)/10,5);
    combi_face3(row*i+1:row*(i+1),col*j+1:col*(j+1)) = reconstruct_face;

    % compute the MSE between original face and reconstructed face
    mse(1,k/10) = sum(sum((uint8(reconstruct_face) -
neutral_test_face).^2))...
        /(col*row);
end

figure;
imshow(neutral_test_face)
title('One of 10 other faces with neutral expression')
figure;
imshow(uint8(combi_face3))
title('Reconstructed face')


%% 4) Reconstructing a car image using all PCs
k = 190;
% call reconstructFace function
reconstruct_car = reconstructFace(car, eigenFaces, k, average_face);

figure;
subplot(1,2,1)
imshow(car)
title('Reconstructed car image')
subplot(1,2,2)
imshow(uint8(reconstruct_car))
title('A car image')


%% 5) Reconstructing rotate trained face using all PCs
% call reconstructFace function
k = 190;
```

```
reconstruct_face_rot = reconstructFace(neutral_train_face_rotate, eigenFaces,
k, average_face);

figure;
subplot(1,2,1)
imshow(neutral_train_face_rotate)
title('An rotated face')
subplot(1,2,2)
imshow(uint8(reconstruct_face_rot))
title('Reconstructed rotated image')


%% 6) Reconstructing noisy trained face using k PCs
% Gaussian noise with variance = 0.01
neutral_test_face_noisy1 = imnoise(neutral_test_face,'gaussian',0,0.01);
for k = 10:10:190
    % call reconstructFace function
    reconstruct_face = reconstructFace(neutral_test_face_noisy1, eigenFaces,
k, average_face);
    [row,col] = size(reconstruct_face);

    % combine all reconstructed faces together
    if k <= 100
        combi_face5(1:row,((k/10-1)*col+1):(k/10)*col) = reconstruct_face;
    else
        combi_face5(row+1:2*row,(((k-100)/10-1)*col+1):((k-100)/10)*col) =
reconstruct_face;
    end

    % compute the MSE between original face and reconstructed face
    mse(1,k/10) = sum(sum((uint8(reconstruct_face) -
neutral_test_face_noisy1).^2))...
        /(col*row);
end

figure;
imshow(neutral_test_face_noisy1)
title('One of 10 other people with noise (v = 0.01)')
figure;
imshow(uint8(combi_face5))
title('Reconstructed noisy image with v = 0.01')

% Gaussian noise with variance = 0.1
neutral_test_face_noisy2 = imnoise(neutral_test_face,'gaussian',0,0.1);
for k = 10:10:190
    % call reconstructFace function
    reconstruct_face = reconstructFace(neutral_test_face_noisy2, eigenFaces,
k, average_face);
    [row,col] = size(reconstruct_face);

    % combine all reconstructed faces together
    if k <= 100
        combi_face6(1:row,((k/10-1)*col+1):(k/10)*col) = reconstruct_face;
    else
```

```
        combi_face6(row+1:2*row,(((k-100)/10-1)*col+1):((k-100)/10)*col) =
reconstruct_face;
    end

    % compute the MSE between original face and reconstructed face
    mse(2,k/10) = sum(sum((uint8(reconstruct_face) -
neutral_test_face_noisy2).^2))...
        /(col*row);
end

figure;
imshow(neutral_test_face_noisy2)
title('One of 10 other people with noise (v = 0.1)')
figure;
imshow(uint8(combi_face6))
title('Reconstructed noisy image with v = 0.1')

% Gaussian noise with variance = 0.5
neutral_test_face_noisy3 = imnoise(neutral_test_face,'gaussian',0,0.5);
for k = 10:10:190
    % call reconstructFace function
    reconstruct_face = reconstructFace(neutral_test_face_noisy3, eigenFaces,
k, average_face);
    [row,col] = size(reconstruct_face);

    % combine all reconstructed faces together
    if k <= 100
        combi_face7(1:row,((k/10-1)*col+1):(k/10)*col) = reconstruct_face;
    else
        combi_face7(row+1:2*row,(((k-100)/10-1)*col+1):((k-100)/10)*col) =
reconstruct_face;
    end

    % compute the MSE between original face and reconstructed face
    mse(3,k/10) = sum(sum((uint8(reconstruct_face) -
neutral_test_face_noisy3).^2))...
        /(col*row);
end

figure;
imshow(neutral_test_face_noisy3)
title('One of 10 other people with noise (v = 0.5)')
figure;
imshow(uint8(combi_face7))
title('Reconstructed noisy image with v = 0.5')


%% Reconstruct all 10 other people's face
for i = 191:200
    temp = imread([num2str(i),'a.jpg']);
    temp_reconstruct = reconstructFace(temp, eigenFaces, 80, average_face);
    [row, col] = size(temp_reconstruct);

    combi_face8(1:row,col*(i-191)+1:col*(i-190)) = temp;
    combi_face8(row+1:2*row,col*(i-191)+1:col*(i-190)) = temp_reconstruct;
```

```
end

figure;
imshow(uint8(combi_face8))
title('All 10 other people''s face(up) and their reconstructed face(bottom)
with nPCs = 80')
```

Face Detection

```
% ------------------------ SVM with Eigenfaces ---------------------- %
clc;close all; clear all;
addpath('faces');
%% Training Faces to obtain Eigenfaces
% compute eigenFaces and an average face
num = 100; % number of training faces
[eigenFaces, average_face] = trainFaces(num);

%% SVM for detection
k = 100; % number of PCs that are selected
feature = []; % sample data for training

% Reading training data
for i = 101:150
    temp_original = imread([num2str(i),'a.jpg']);
    temp_reconstruct = reconstructFace(temp_original, eigenFaces, k,
average_face, num);

    % Calculate the euclidean distance between original face and
    % reconstructed face
    dist = sqrt(sum((double(temp_original(:)) - temp_reconstruct(:)).^2));
    feature = [feature; 0 dist -1];
end
for i = 101:150
    temp_original = imread([num2str(i),'b.jpg']);
    temp_reconstruct = reconstructFace(temp_original, eigenFaces, k,
average_face, num);

    % Calculate the euclidean distance between original face and
    % reconstructed face
    dist = sqrt(sum((double(temp_original(:)) - temp_reconstruct(:)).^2));
    feature = [feature; 0 dist 1];
end

% Training SVM
feature = feature(randperm(size(feature,1)),:); % shuffle the sample data
figure;
svm = svmtrain(feature(:,1:2), feature(:,3), 'Showplot',true);
title('SVM for face detection (-1: Normal face, 1: Smiling Face)')

% Reading test data
feature_test = []; % sample data for testing;
label = []; % The true label
for i = 151:200
```

```
    temp_original = imread([num2str(i),'a.jpg']);
    temp_reconstruct = reconstructFace(temp_original, eigenFaces, k,
average_face, num);

    % Calculate the euclidean distance between original face and
    % reconstructed face
    dist = sqrt(sum((double(temp_original(:)) - temp_reconstruct(:)).^2));
    feature_test = [feature_test; 0 dist];
    label = [label -1];
end
for i = 151:200
    temp_original = imread([num2str(i),'b.jpg']);
    temp_reconstruct = reconstructFace(temp_original, eigenFaces, k,
average_face, num);

    % Calculate the euclidean distance between original face and
    % reconstructed face
    dist = sqrt(sum((double(temp_original(:)) - temp_reconstruct(:)).^2));

    feature_test = [feature_test; 0 dist];
    label = [label 1];
end

% Testing SVM model
group = svmclassify(svm, feature_test);

% Compute predict accuracy rate
num = 0;
for i = 1:length(group)
    num = num + (group(i) == label(i));
end
accuracy = num / length(group);
```