Applied Mathematics Final Project

University of Burgundy

Master's in Computer Vision and Robotics

(VIBOT & MsCV)

Face Recognition using Principle Component Analysis

Group members:
Hassa ZAAL
AbdelRahman ABUBAKR

Supervisor:
Prof. Desire Sedibe

Contents

1 Introduction

Facial recognition systems are commonly used for security purposes but are increasingly being used in a variety of other applications. One of the most famous application we see everyday is the Facebook face recognition system, in which the Facebook can recognize the person and his friends, then suggests to tag them in the image.

1.1 Objective

The objective of this project is to use Principle Component Analysis (PCA) algorithm, to achieve the goal of Face recognition of our class members. For this purpose, we are given a dataset containing different images for every person in our class in the size of 320 x 320 pixels, and we are asked to normalize and convert them to a fixed window of size 64 x 64 pixels, then apply the PCA algorithm that when given a face of any person from the training set, find the nearest matching face for this person in the training set.

2 Normalization

2.1 Overview

Given a face image, a normalization step is required prior to recognition process. The goal of normalization is to account for variations in location, scale, orientation, etc. In this project, the normalization step will be done by an iterative algorithm based on affine transformations.

The idea in this approach is to use a transformation to map certain facial features from face image to predetermined location in a fixed size window. In our project, we are using the following facial features: (1) Left eye center, (2) right eye center, (3) tip of nose, (4) left mouth corner, and (5) right mouth corner. An initial positions of these features are given as follows:

$$F^p = \begin{pmatrix} P_1 = (13, 20) \\ P_2 = (50, 20) \\ P_3 = (34, 34) \\ P_4 = (16, 50) \\ P_5 = (48, 50) \end{pmatrix}$$

2.2 The algorithm

After extracting the locations of the facial features from all images, we can compute the parameters of an affine transformation that maps them to the predetermined location in 64 x 64 window. The algorithm described below is used to compute the parameters of an affine transformation that maps the facial features of each image to the predetermined facial feature locations in the 64 x 64 window. this algorithm works as follows:

Step 1: Use a vector $\overline{F}$ to store the average locations of each facial feature over all face image i.e., $\overline{F} = (\overline{P}_1\ ,\ \overline{P}_2\ ,\ \overline{P}_3\ ,\ \overline{P}_4\ ,\overline{P}_5\ )$ where $\overline{P}_1$ and $\overline{P}_2$ correspond to the average positions of the eye centers, $\overline{P}_3$ corresponds to the average position of the noses tip, $\overline{P}_4$ and $\overline{P}_5$ correspond to the average positions of the mouth corners. For simplicity, we initialize $\overline{F}$ with the feature locations of just the first face image i.e., $\overline{F} = F_1$

Step 2: Compute the affine transformation that aligns $\overline{F}$ with the predetermined positions F^p in the 64 x 64 window. Since each pair of corresponding features must satisfy the affine transformation equations, we have the following equations:

$$F_1^p = A\overline{P}_1 + b$$
$$F_2^p = A\overline{P}_2 + b$$
$$F_3^p = A\overline{P}_3 + b$$
$$F_4^p = A\overline{P}_4 + b$$
$$F_5^p = A\overline{P}_5 + b$$

Where

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

$$b = \begin{bmatrix} b_{11} \\ b_{21} \end{bmatrix}$$

Since there are 10 equations (i.e., two for each feature in X and Y) in 6 unknowns, the system is overdetermined and can be solved using SVD. After calculating the transformation, compute the aligned features $\overline{F}' =$ A $\overline{F}$ + b and update

$\overline{F}$ by setting $\overline{F} = \overline{F}'$.

Step 3: For every face image F_i, use SVD to compute the affine transformation (A_i ,b_i) that aligns the facial features of F_i with the average facial features $\overline{F}$; lets call the aligned features F' where $F' = A_i \ F_i + b_i$.

Step 4: Update $\overline{F}$ by averaging the aligned feature locations F_i' for each face image i.

Step 5: If $\|\overline{F}(t) - \overline{F}(t-1)\|$ is less than a threshold, then stop; otherwise, go to Step 2.

For our code, using a threshold of 0.0001, this algorithm converged with 3 iterations. For each face image, it yields an affine transformation that maps the face image to the 64 x 64 window. The final $\overline{F}$ we get at the end is as follows:

$$\overline{F} = \begin{bmatrix} 15.588 & 19.320 \\ 48.017 & 19.350 \\ 32.848 & 37.134 \\ 14.238 & 49.138 \\ 50.307 & 49.0552 \end{bmatrix}$$

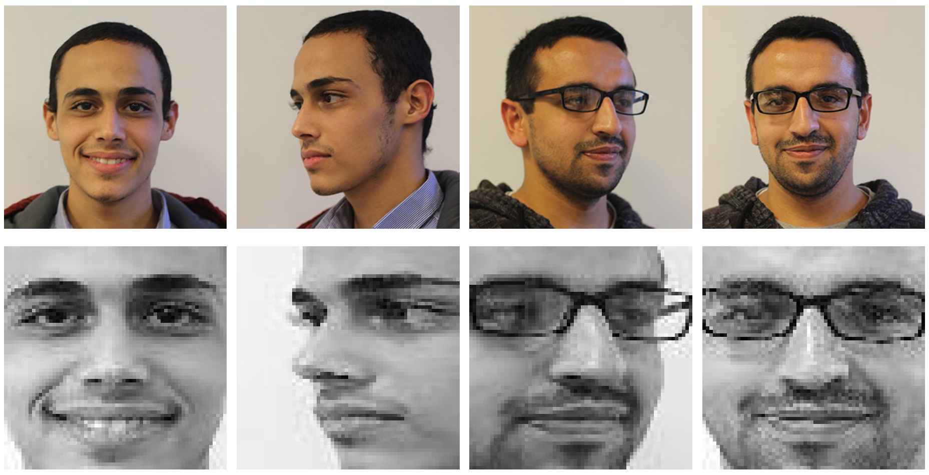

Figure 1: A sample of the images used, before and after the normalization step

To avoid gaps when computing the normalized images, each point in the normalized images is determined by applying the inverse affine transformation, using the final $\overline{F}$. So, the output images are got by iterating through every point in the output image, then find the corresponding point in the input image, and copy it over in the output image.

Figure 1 shows samples of the dataset images before and after the normalization step, and the code of the normalization algorithm can be found in Appendix A.

3 Principle Component Analysis

3.1 Overview

In PCA, faces are represented as a linear combination of weighted eigenvectors called as Eigen faces. These eigenvectors are obtained from covariance matrix of a training image set. These eigenvectors defined a new face space where the images are represented. After the face space has been constructed, the feature vectors are formed as a linear combination of the eigenvectors of the covariance matrix. The recognition accuracy of the system is computed as the ratio of the faces recognized correctly from the test set over the total number of faces in the test set.

3.2 The algorithm

We divide the face images after normalization into two groups: a training set and a test set. Then we applied PCA algorithm.

Reshaping the images: We convert each of the images into MN dimensional column vector (1 x MN) by appending the rows of the image. The shape of each image has been changed in order to create a data matrix of all the images.

Create the data matrix: The system will insert all the reshaped images into a new matrix (data matrix) as rows.

Creating a mean matrix: The proposed system creates a mean matrix of all the different image matrices.The mean matrix will be calculated by adding all the rows of data matrix divided by the total number of rows.

Subtract the mean from each column: The algorithm then subtracts the mean from all the columns to get the mean subtracted data matrix.

Create the covariance matrix: The system will make the covariance matrix by multiplying($\frac{1}{length(trainingimages)-1}$) by the mean subtracted matrix by its transpose to make it a square matrix in the next phase.

Find the eigen values and vectors: Then the system finds the Eigen vectors and Eigen values.

Create an eigen image: The proposed system then creates an Eigen image by multiplying transpose the mean subtracted data matrix with the Eigen vectors.

Choosing highest eigen vectors: The proposed system then choose the highest eigen vectors by looking for highest eigen values and then picking up the corresponding rows of the eigen image as they have the highest eigen vectors. Eigen vectors with the highest eigen values are the principle components of the data set having maximum information. This will also reduce the size of final data set.

Creating the weighted matrix: The main data set known as the weighted matrix has been created which is used for the identification process. After this step the system can recognize any face image by comparing it with the main weighted matrix.

Finding the best match: The best match image is identified as the person with the smallest value among all the Euclidean values between the test image and images present in the training set. Euclidean distance provides similarity measure of feature between test and images present in the training set. Less Euclidean distance shows that maximum feature of input image and image to be recognized is matched. Less Euclidean distance indicates the maximum feature of input image and recognized image is matched.

Accuracy: We Check if the labels of best two matched images are the same. Otherwise, increment an error e count by 1.The recognition accuracy of the system is computed as the ratio of the faces recognized correctly from the test set over the total number of faces in the test set.
$accuracy = (1 - \frac{e}{total\ number\ of\ test\ images}) * 100$

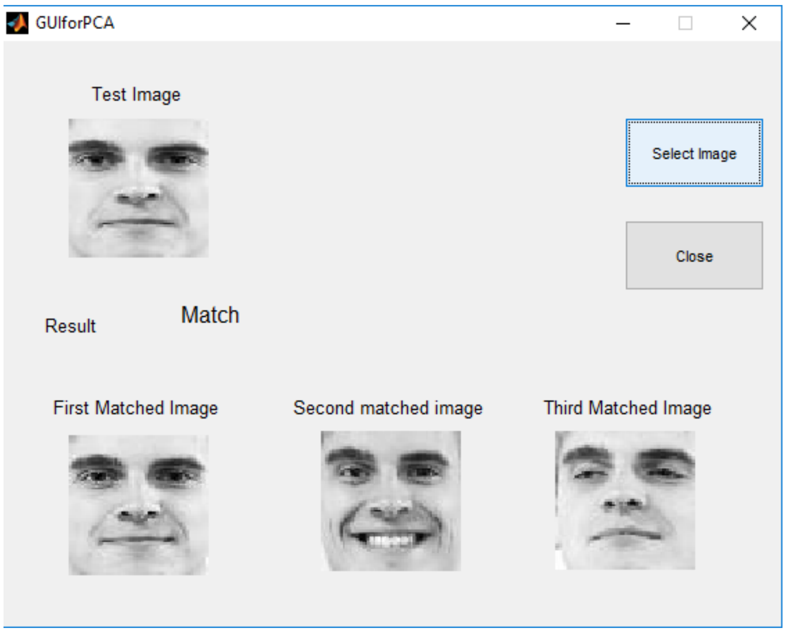

Figure 2

3.3 GUI

We created Graphical User Interface. A pushbutton to select an image to test it and the best three matched images will appear.

3.4 Results

Figure 2 shows that the three images are matched. Figure 3, Figure 4 and Figure 5 show that the first image is just matched. Figure 6 shows that the first three images are not matched but they match in orientation. Figure 7 shows that the first three images are not matched.

The final overall accuracy is 75%, and we can improve the accuracy by taking for each person more different images in different orientation and conditions.

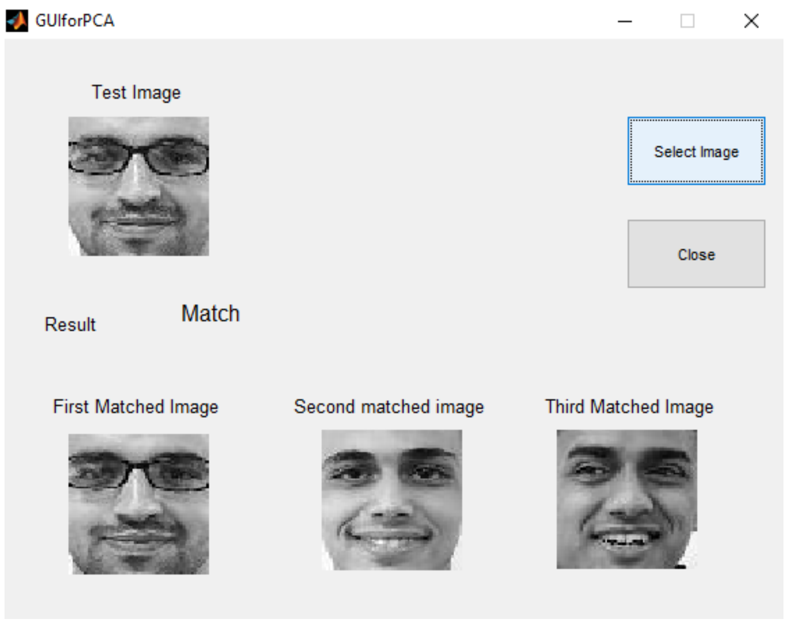

Figure 3

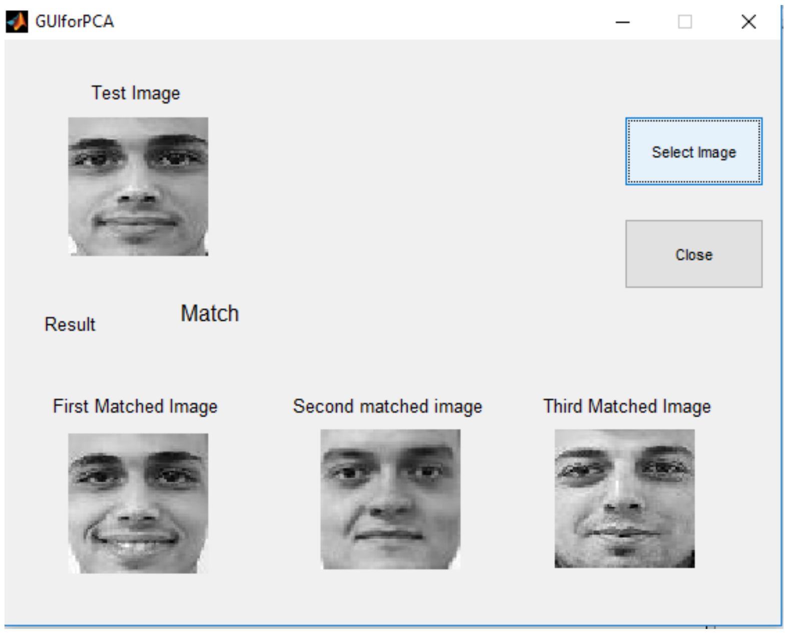

Figure 4

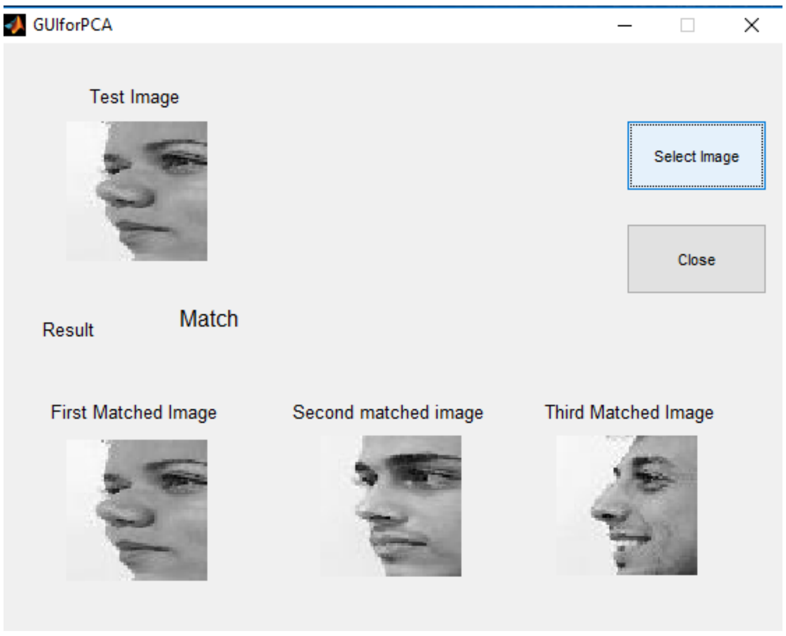

Figure 5

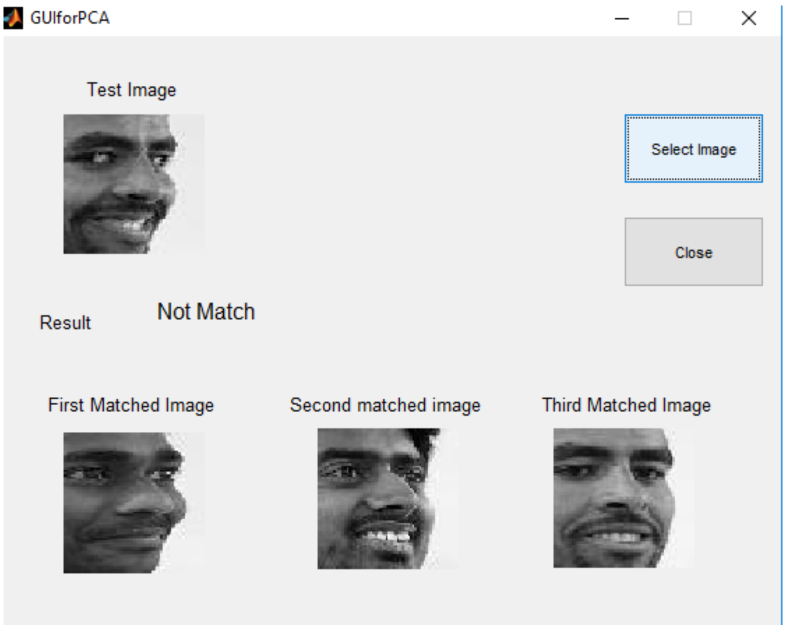

Figure 6

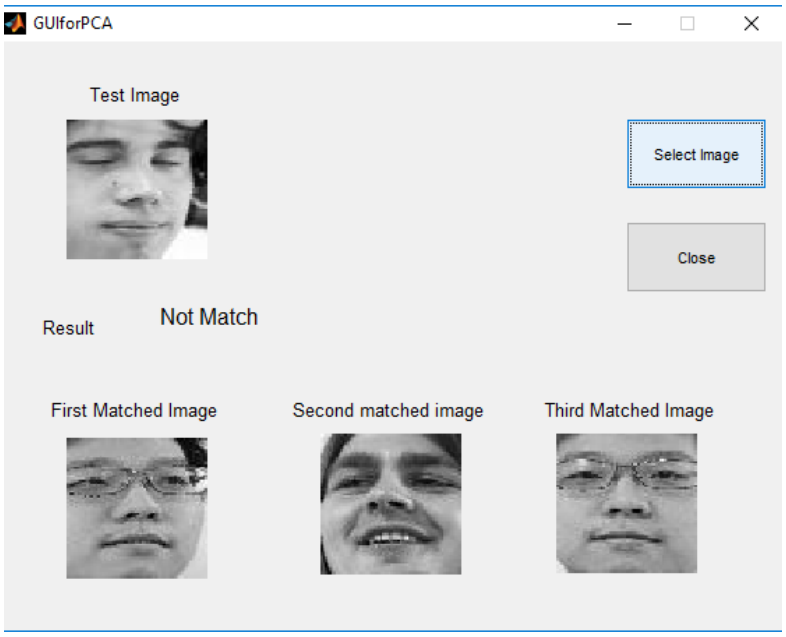

Figure 7

Appendices

A Normalization code

In this appendix, we show the code of the normalization step. this code is tested with all the face images, and the results are attached with the code.

```
% Normalization for all images
% Reading from excel file
filename = 'Dataset_Face_Recognition.xlsx';

% Makin 6 arrays to store the 5 features location and the names
[num,Names,raw] = xlsread(filename, 'A2:A151');
Eye1 = xlsread(filename, 'B2:C151');
Eye2 = xlsread(filename, 'D2:E151');
Nose = xlsread(filename, 'F2:G151');
Mouth1 = xlsread(filename, 'H2:I151');
Mouth2 = xlsread(filename, 'J2:K151');

% Step 1: Initialization
Fb = [13 20; 50 20; 34 34; 16 50; 48 50];
% F initialized with first image coordinates
F = [Eye1(1,:) ; Eye2(1,:); Nose(1,:) ; Mouth1(1,:); Mouth2(1,:)];
```

```
% The threshold to stop the loop and get the final F
Threshold = 0.0001 * ones(size(F));
Fn = zeros(size(F));
count = 0;

while (abs(F - Fn) > Threshold)
% Step 2
Fn = F; % prepare Fn to the next step
% putting ones in the last column of F, to calculate A & B
    together
P = [Fn ones(5,1)];
A = pinv(P)*Fb;  % pinv() uses SVD to get the inverse of P
F_ = P * A;  % Getting F_
F = F_; % updating F
% intialize accumulation to calculate average of all Fi at the end
Fi_acc = zeros(size(F));

% Step 3
 for i = 1 : size(Names) % loop to all images
   % form of Fi
   Fi = [Eye1(i,:) ; Eye2(i,:); Nose(i,:) ; Mouth1(i,:); Mouth2(i
       ,:)];
   Pi = [Fi ones(5,1)];

   Ai = pinv(Pi)*F; % Affine transformation for each image
   Fi_ = Pi * Ai; % apply the affine transfomation to get Fi_
   Fi_acc = Fi_acc + Fi_; %
 end

% Step 4
% update F by averaging the aligned features for all images
F = Fi_acc/length(Names);
count = count +1 ;

end

% Paths to input and output folders
path_source = 'faces_320x320/';
path_output = 'faces_64x64/';
NewIm = zeros(64,64); % initialize the new iamge matrix

for k = 1 : length(Names)
 % prepare the name of the image _ its path to read it
 ImgName = char(strcat(path_source, Names(k)));
 RGBIm = imread(ImgName, 'jpg');
 Im = rgb2gray(RGBIm);
 %imshow(Im)

 Fk = [Eye1(k,:) ; Eye2(k,:); Nose(k,:) ; Mouth1(k,:); Mouth2(k,:)
    ];
 Pk = [Fk ones(5,1)];
 Ak = pinv(Pk)*F;  % Get Affine transformation for each image

 Ak_ = [Ak [0;0;1]];
 A_inv = pinv(Ak_);
% Apply the inverse affine transformation from output to input.
 for x = 1: 64  % nested loop to look at all pixels in 64 x 64
     for y = 1: 64
         X_ = [x; y; 1];
         X_orig = A_inv' * X_ ;
         X_orig = ceil(abs(X_orig));
         % To deal with pixels of indeces > 320
         if (X_orig(1) > 320)
             X_orig(1) = 320;
         end
         if(X_orig(2) > 320)
             X_orig(2) = 320;
         end
         % NewIm(x,y) = Im(X_orig(1), X_orig(2));
         NewIm(y,x) = Im(X_orig(2), X_orig(1));
     end
 end

 I = mat2gray(NewIm); % convert the matrix to image
 imshow(I); % show all images after normalization
 path_out = char(strcat(path_output, Names(k), '.jpg'));
 imwrite(I, path_out); % save all images to the output folder
end
```

B PCA Code

In this appendix, we show the code of the PCA step. this code is tested with test set images, and the results are attached with the code.

```

% give To choose Test Image and display it
path = 'test_images\*';
```

```
selectedFile = uigetfile(fullfile(path ));
axes(handles.TestImage);

ImgName = char(strcat('test_images\',selectedFile));
Q=ImgName;
test_image_selected1=imread(ImgName,'jpg');

imshow(test_image_selected1);

%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%reading training images and assigning a label to each training
    image
training_images=dir('train_images\*.jpg');

Lt=[];% to assign image label
T=[]; % to create matrix each row is an image

for k = 1 : length(training_images)

ImgName = training_images(k).name;
Lt=[Lt ; ImgName(:,1:3)];% assigning label for images by taking
    first 3 letters from each image
ImgName = char(strcat('train_images\',ImgName(1:end-4)));

TrainIm = imread(ImgName, 'jpg');
Img=reshape(TrainIm,1,64*64);
T=[T ; Img];
end
%%%%%%%%%%%%%

%subtract off the mean for each dimension
mT=mean(T,2);%mean of rows
D=double(T)-repmat(mT,1,size(T,2));

%covariance matrix
sigma=(1/(length(training_images)-1))*D*(D');


%eigenvalues and eigenvectors
[V,DD]=eigs(sigma,length(training_images)-2); % V eigenvectors and
     DD eignvalues
```

```

phai=D'*V;

Ft=[]; % features matrix
for k = 1 : length(training_images)

ImgName = training_images(k).name;
ImgName = char(strcat('train_images\',ImgName(1:end-4)));
TrainIm = imread(ImgName, 'jpg');
Img=reshape(TrainIm,1,64*64);
feature_vector_Train=double(Img)*phai;
Ft=[Ft;feature_vector_Train];% each row a feature vector for an
    image
end


test_image_selected=reshape(test_image_selected1,1,64*64);
proj=double(test_image_selected)*phai;



for k=1:length(training_images)
x(k)=norm(proj-Ft(k,:));
end

[a z]=min(x); % minimum distance between one test image and all
    training images. z is the index and a the value.
[b g]=sort(x);  %the distances between one image and all training
    images. It sorts from the smallest to biggest.  g is the index
    and b the value.



if Lt(z,1:end)~=Q(:,13:15) % comparing the labels to check the
    matching

set(handles.ResultM,'string','Not Match');
elseif Lt(z,1:end)==Q(:,13:15)

set(handles.ResultM,'string','Match');
end
% axes(handles.ResultImage);
% MatchedImage=T(z,:);
% MatchedImage=reshape(MatchedImage,64,64);
% imshow(MatchedImage);

axes(handles.axes3);
MatchedImage=T(g(1),:); % first matched image
MatchedImage=reshape(MatchedImage,64,64);
imshow(MatchedImage);
MatchedImage=reshape(MatchedImage,64,64);
axes(handles.axes4);
MatchedImage=T(g(2),:);%second matched image
MatchedImage=reshape(MatchedImage,64,64);
imshow(MatchedImage);
axes(handles.axes5);
MatchedImage=T(g(3),:);%third matched image
MatchedImage=reshape(MatchedImage,64,64);
imshow(MatchedImage);


% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% To calculate the accuracy we will read all the test images and search for
% matching and if not match we will increase the error

%reading test images

e=0; % error

test_images=dir('test_images\*.jpg');
for k = 1 : length(test_images)

ImgName = test_images(k).name;
ImgName = char(strcat('test_images\',ImgName(1:end-4)));

TestIm = imread(ImgName, 'jpg');


test_image_selected=reshape(TestIm,1,64*64);
feature_vector_test=double(test_image_selected)*phai;

for k=1:length(training_images)
x(k)=norm(feature_vector_test-Ft(k,:)); % calculate the distances
end

```

```
[a z]=min(x);% minimum distance between one test image and all training images. z is the index and a the value.

if Lt(z,1:end)~=ImgName(:,13:15) % comparing the labels to check the matching and calculate the error
e=e+1;% increase the error if not match

end

end
% Accuracy
accuracy=(1-(e/length(test_images)))*100


% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```