Universite de Bourgogne

Applied Math

Face Recognition using PCA

Final Report

Authors:

Abdullah Thabit
Zohaib Salahuddin

January 13, 2019

Contents

1 Introduction

Face recognition is a technique of identifying a face in an image, which is one of the main applications of image analysis. It is widely used in many applications such as in security and surveillance systems. Face recognition using Principal Component Analysis (PCA) or the Eigenfaces algorithm, is an algorithm that uses eigenvalue decomposition to reduce the dimensionality of a face image by keeping only the eigenvectors with the largest eigenvalues, known as the eigenfaces. The linear combination of these eigenfaces (principal components) are said to be able to represent any face in the dataset, and therefore can identify it [1].

This project aims to implement and test the eigenfaces algorithm on a small dataset containing face images of the students in our class. The algorithm was implemented on multiple steps; first, all the face images were loaded and assigned labels, Then, a normalization step of the face images were performed to prepare the face images for recognition. The dataset was then divided into training and testing sets before moving on to the main identification step of finding the eigenfaces using PCA. Finally, the algorithm was evaluated on the testing set and the system's accuracy was recorded. Moreover, a Graphical User Interface (GUI) was created for interactively testing the system on different images.

2 Methodology

For face recognition using PCA, a normalization step is necessary to align the facial features in the image before applying PCA. Figure 1 shows the work pipeline followed in this project.

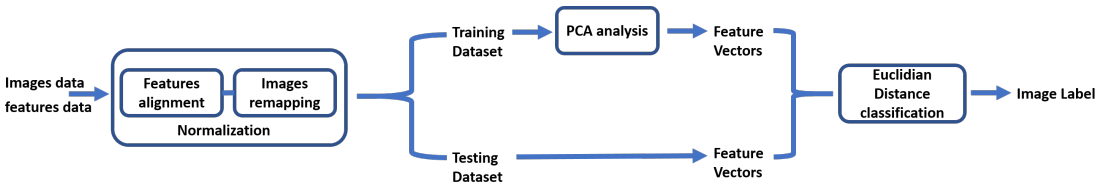

Figure 1: Face recognition pip-line using PCA

2.1 Normalization

Given the dataset of the images along with the xy-coordinates of their respective facial features (points of the left eye, right eye, nose tip, and left and right corners of the mouth), face normalization was applied to align these features and remap the images to a 64x64 face images. An iterative approach is performed to find the average (best) locations for the features to be mapped to. The following steps describe this approach:

1. A vector representing the average locations was defined and firstly initialized to a feature vector of one of the images.

2. This vector was then mapped to a predefined location using an affine transformation represented by the following formula:

$$f_i^p = A * f_i + b \tag{1}$$

   The formula was applied on two steps; firstly, the coefficients were determined by using Singular Value Decomposition, and then the affine transformation was evaluated to get the mapped average feature vector.

3. We iterate through every image, mapping its feature vector to the feature vector found in step 2.

4. We update the average feature vector by the averaging the vectors of aligned features in all the images.

5. We compare the final averaged feature vector with our previous average vector, if the difference is higher than a specific threshold, we go back to step 2. Otherwise we stop.

Once we get our final feature vector, we use it to map each face image to a 64x64 smaller image in an inverse mapping fashion as shown in the following steps:

1. We calculate the affine transformation coefficients from (1) for mapping every image facial features with the final average feature vector found previously.

2. We apply the inverse of this transformation to every pixel in the 64x64 image and therefore mapping the original face image to the new resized image.

Finally, the resized images are split into training dataset containing three images of each student (front, left and right sideviews), and testing dataset containing the rest of students' images.

2.2 Recognition

After preparing the dataset for training, the following steps are done to perform PCA and face classification:

1. Each face image KxK (64x64) should be converted into a column of K2 dimension. Columns of all images are then concatenated to form a feature matrix PxK2 as shown in Figure 2.

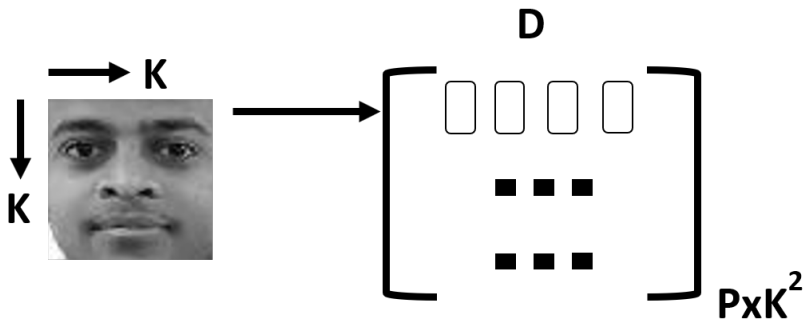

Figure 2: Converting face images into columns and concatinating them in a feature matrix

2. The mean of each column in matrix D is calculated and then deducted to center the intensity values in D.

3. The covariance matrix of matrix D is then calculated based on the formula:

$$\sum = \frac{1}{p-1} * D * D^T \tag{2}$$

4. An eigenvalue decomposition is performed on $\sum$ to get its eigenvalues and eigenvectors. We only keep N (usually less than 100) eigenvectors that correspond to the largest eigenvalues.

5. The kept eigenvectors are referred as the principal components, where they are of a dimension of K2; hence, by reshaping them back to KxK matrix we get their respective eigenfaces as shown in Figure 3.

Figure 3: samples of the resulted eignefaces (principal components

6. We Put the N principal component into a KxN projection matrix; Therefore, any image represented as a vector Xi in the K-dimensional space, can be projected into the PCA space by computing:

$$\phi_i = X_i * \phi \tag{3}$$

Once the projection matrix ϕ and all feature vectors in the training dataset were calculated, the actual recognition phase comes into place. Therefore, for one face image I_j in the testing dataset, the Euclidean distances between its feature vector ϕ_j and all feature vecotors in the training dataset is calculated. Then, the image I_j is identified by assigning the label of the feature vector of the smallest Euclidean distance to its feature vector.

2.3 Graphical User Interface

A simple graphical user interface was developed for this work. It aimed to simplify the testing phase by making it interactive. Figure 4 shows the systems' GUI and its main functionalities.

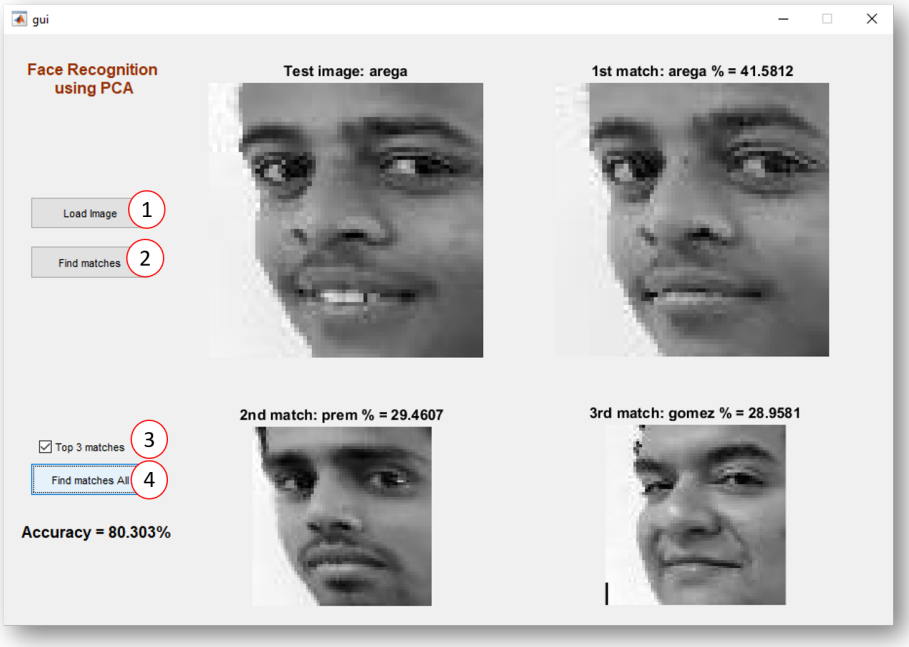

Figure 4: Graphical User Interface of the face recognition system

- ① **Load Image** button to load the image file needed to be matched
- ② **Find matches** button to run the algorithm and find best matches for the testing image
- ③ **Top 3 matches** tick box to calculate the accuracy with respect to the top 3 matches.
- ④ **Find matches all** button to check the system's accuracy in matching all images in the testing dataset.

3 Results and Discussion

Images of 33 students were involved in this work, with 3 images of each student in the training set summing up to a total of 98 images (only 2 images were included for one of the students), and 2 images in the

testing dataset summing up to a total of 66 images. Only one student (out of 34) was excluded from the study due to improper locating of the facial features in the preprocessing step.

All images were normalized together and then were split into training and testing datasets. The system was assessed by increasing an error counter every mismatch happens between the testing image and its true label. The accuracy was then calculated according to the formula:

$$accuracy = ((1 - \frac{\epsilon}{totalnumberoftestimages}) * 100) \tag{4}$$

The accuracy was also assessed for the top 3 matches, and the error counter was only increased whenever the label of the testing image was not found within the top 3 matches in the training dataset. Moreover, a percentage of confidence for each of the top 3 matches was calculated as shown in Figure 5.

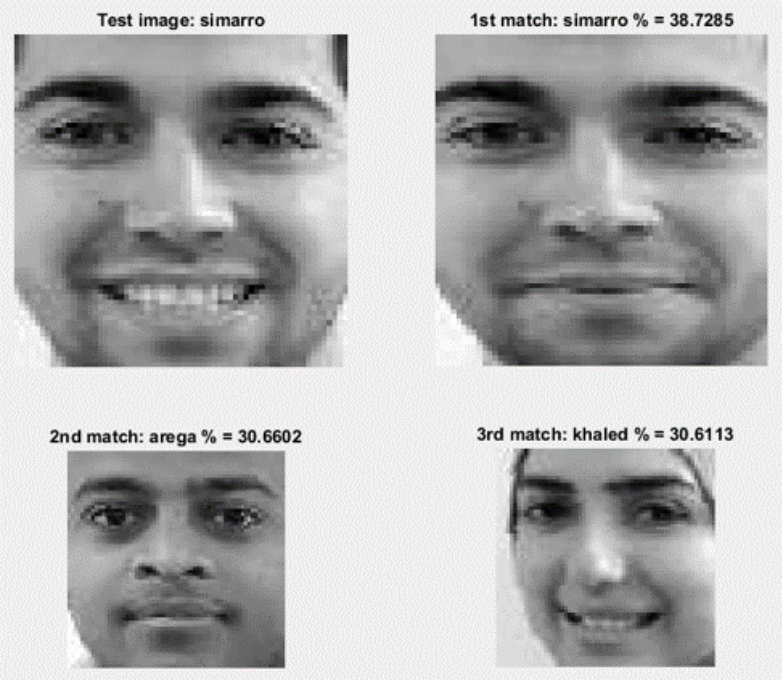

Figure 5: Sample test showing a test image and its 3 top matches along with their confidence percentages

Different parameters were noticed to affect the obtained overall accuracy, such as the chosen number of principal components and the dimension of the covariance matrix in the PCA analysis. Figure 6 shows the changes in accuracy as a result in changing the number of principal components. It can be seen from the figure that the system was able to reach an accuracy of 68.18% with number of principal components greater than 40, and 80.3% for the correct person to be identified among the top 3 matches.

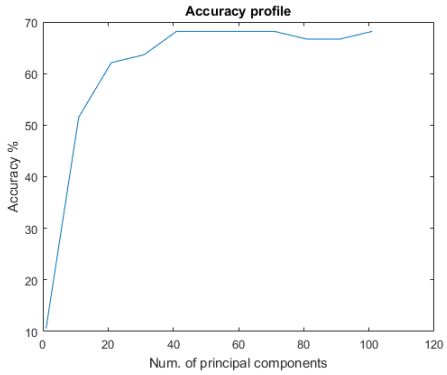

Figure 6: accuracy profile with respect the number of principal components

In the best-case scenario, our system was able to correctly identify the right person in the testing image 45 times out of 66 images in the testing dataset. 4 of these mismatched images were outliers with one of the feature points improperly located, leading to a deformed face image as shown in Figure 7.

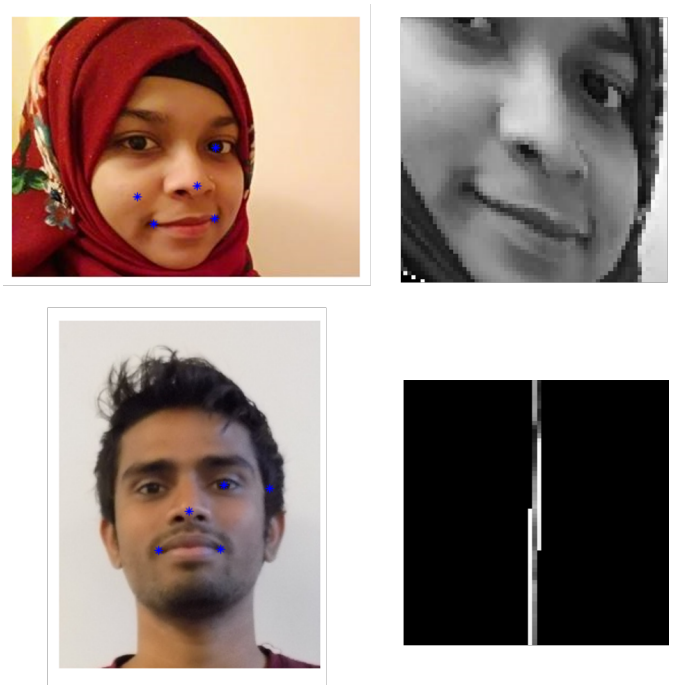

Figure 7: Sample of the outliers in the dataset. Blue stars indicate the position of the chosen facial features

By taking a closer look at the confidence percentages obtained for each matching result, as shown in Figure 5, we notice only small difference between the top 3 matches, which explains the not very satisfying accuracy of the system. Despite the normalization step performed to reduce variabilities in scaling and rotation between images, the system still misclassified about one third of the testing images. We think that is due to the low accuracy of the Euclidian distance classifier, which is not robust when it comes to small differences between objects to be identified.

A disadvantage of the presented algorithm include the need for precise alignment of the facial features before performing PCA analysis, which was accomplished through the normalization step, otherwise the algorithm will fail to correctly identify faces. This normalization step suffers from two sides: 1) it needed manual locating for facial features before remapping the images, which is a tedious task prone to mistakes

as shown in figure 2) features alignment could be a challenging task and hard to guarantee in real life; Figure shows an example of such misalignment, which could be more sever in an uncontrolled and less ideal dataset. Another limitation of this work was the small available dataset. 5 images per person were not enough for proper training and evaluation of the system. Moreover, a larger dataset is required incase more advanced classification techniques were intended to be used to enhance the classification accuracy, such as using Support Victor Machine (SVM).

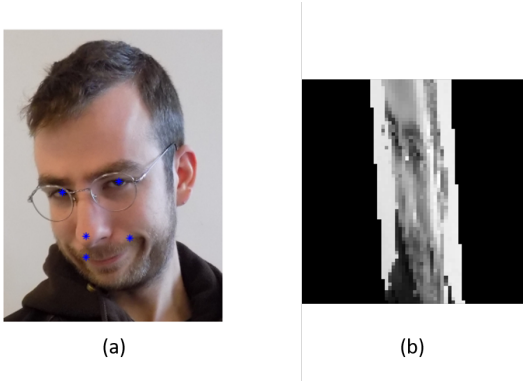

Figure 8: Misalignment sample due to close distance between facial features (blue stars)

4 Conclusion

Face recognition Using Principal Component Analysis is a very popular algorithm and one of the very first developed algorithms in this area. It clearly shows how powerful PCA is in reducing dimensionality while keeping critical information. However, this algorithm requires a strict alignment of the facial features, making it less robust in real life applications. Further improvements of this work could be done by testing it on a larger dataset and applying a more advanced machine learning classification techniques instead of relying on the Euclidian distance.

5 References

[1] Sirovich, Lawrence, and Michael Kirby. "Low-dimensional procedure for the characterization of human faces." Josa a 4.3 (1987): 519-524.

6 Annex: MATLAB code printout

6.1 Main file: FindMyFace

```
close all, clear all
%% Load initial data (No need to run these functions)
% data = LoadData;
%% Normalize each raw image to a size of 64x64
% faces = NormalizeFaces(data);
%% Load training and testing datasets
[trainData,testData] = LoadTraining_TestingData;
%% PCA face recognition
train_output = PCArecognition(trainData,97);
%% Test the system: find the best match
Top3matchesFlag = 1;
accuracy =
    FindFace(trainData,testData,train_output,Top3matchesFlag);
accuracy = FindFace(trainData,testData,train_output,1);
```

6.2 Function: LoadData

```
function data = LoadData
    FeatureFolder = 'all_features'; % Determine where demo
        folder is (works with all versions).
    if ~isdir(FeatureFolder)
        errorMessage = sprintf('Error: The following folder
            does not exist:\n%s', myFolder);
        uiwait(warndlg(errorMessage));
        return;
    end
```

```
featurePattern = fullfile(FeatureFolder, '*.txt');
txtFiles = dir(featurePattern);
cd ('all_features')
for k = 1:length(txtFiles)
    baseFileNameFeature = txtFiles(k).name;
    fullFileNameFeature = fullfile(FeatureFolder,
        baseFileNameFeature);
    fprintf(1, 'Now reading %s\n', fullFileNameFeature);
    facef = load(fullFileNameFeature);
    dif1 = abs(facef(2,1) - facef(1,1));
    dif2 = abs(facef(2,2) - facef(1,2));
    if dif2 > dif1
        fixedFacef = [facef(:,2),facef(:,1)];
    else
        fixedFacef = facef;
    end
    featuresArray{k} = fixedFacef;

end
cd ..

myFolder = 'all_faces'; % Determine where demo folder is
    (works with all versions).
if ~isdir(myFolder)
    errorMessage = sprintf('Error: The following folder
        does not exist:\n%s', myFolder);
    uiwait(warndlg(errorMessage));
    return;
end
facePattern = fullfile(myFolder, '*.jpg');
jpgFiles = dir(facePattern);
cd ('all_faces')
for k = 1:length(jpgFiles)
    baseFileNameFaces = jpgFiles(k).name;
    fullFileNameFaces = fullfile(myFolder,
        baseFileNameFaces);
    fprintf(1, 'Now reading %s\n', fullFileNameFaces);
    imageArray{k} = imread(fullFileNameFaces);
    imshow(imageArray{k}); % Display image.
    drawnow; % Force display to update immediately.
```

```
    end
    cd ..
    for k = 1:length(jpgFiles)

        data.name{k} = jpgFiles(k).name;
        data.features{k} = featuresArray{k};
        data.faces{k} = imageArray{k};
    end
end
```

6.3 Function: NormalizeFaces

```
 function faces = NormalizeFaces(data)
%%
    Finit = data.features(1);
    Favg = round(cell2mat(Finit));
    Fpre = [13,20;50,20;34,34;16,50;48,50];
    count = 0;
    while(1)
        [A,b] = FindTransformation(Favg,Fpre);
        Fout = ApplyTransfomration(A,b,Favg);
        Favg = Fout;
        for i = 1: length(data.features)
            Fi = data.features(i);
            Fi = round(cell2mat(Fi));
            [A,b] = FindTransformation(Fi,Favg);
            Fout = ApplyTransfomration(A,b,Fi);
            Fall(:,:,i) = Fout;
        end
        %%
        F = mean(Fall,3);
        e = abs(F - Favg);
        thresh = 0.9 * ones(size(F));
        if (e<thresh)
            break;
        end
        if count >= 10
            break;
```

```
        end
        Favg = F;
        count = count + 1;
    end
    %%
    Ismall = zeros(64,64,length(data.features));
    for k = 1:length(data.features)

        Fi = data.features(k);
        Ii = cell2mat(data.faces(k));
        Ii = rgb2gray(Ii);
        m = size(Ii,1);
        n = size(Ii,2);
        Fi = round(cell2mat(Fi));
        [A,b] = FindTransformation(Fi,F);
        Ainv = inv(A);
        for i = 1:size(Ismall,1)
            for j = 1:size(Ismall,2)

                inxy = [j;i];
                outxy = Ainv * ( inxy - b);
                outxy = round(outxy);
                if (outxy(1) > 1 && outxy(1) <= n && outxy(2) >
                    1 && outxy(2) <= m)
                    Ismall(i,j,k) = Ii(outxy(2),outxy(1));
                end

            end
        end
    end

    for i = 1:size(Ismall,3)
        Ii = cell2mat(data.faces(i));
        Ii = rgb2gray(Ii);
        subplot(121),imshow(Ii,[])
        subplot(122),imshow(Ismall(:,:,i),[])
        drawnow;
    end
faces = Ismall;
end
```

6.4 Function: FindTransformation

```
function [A,b] = FindTransformation(Fin,Fref)

    Ain = [Fin, ones(5,1)];
    b1 = Fref(:,1);
    b2 = Fref(:,2);

    [U,S,V] = svd(Ain);
    sv = find(S);

    c = U'*b1;
    for i = 1:length(sv)
        y(i) = c(i)/S(i,i);
    end
    y = y';
    x1 = V*y;

    c2 = U'*b2;
    for i = 1:length(sv)
        y2(i) = c2(i)/S(i,i);
    end
    y2 = y2';
    x2 = V*y2;

    A = [x1(1),x1(2);x2(1),x2(2)];
    b = [x1(3);x2(3)];

end
```

6.5 Function: LoadTrainingTestingData

```
function [train_images,test_images] = LoadTraining_TestingData
%% save croped faces lll
% cd('train_images');
% for i = 1:size(faces,3)
%     name = data.name{i};
%     image = mat2gray(faces(:,:,i));
```

```
%     imwrite(image,name);
% end
% cd ..
%% read training images
myFolder = 'train_images';
trainfile = fullfile(myFolder, '*.jpg');
jpgFiles = dir(trainfile);
cd(myFolder);
for k = 1:length(jpgFiles)
    baseFileNameFaces = jpgFiles(k).name;
    fullFileNameFaces = fullfile(myFolder, baseFileNameFaces);
    fprintf(1, 'Now reading %s\n', fullFileNameFaces);
    imageArray{k} = imread(fullFileNameFaces);
    imshow(imageArray{k}); % Display image.
    drawnow; % Force display to update immediately.
end
cd ..
for k = 1:length(jpgFiles)
    name = jpgFiles(k).name;
    name = name(1:end-5);
    train_images.name{k} = name;
    train_images.faces{k} = imageArray{k};
end

%% read testing images

myFolder = 'test_images';
trainfile = fullfile(myFolder, '*.jpg');
jpgFiles = dir(trainfile);
cd(myFolder);
for k = 1:length(jpgFiles)
    baseFileNameFaces = jpgFiles(k).name;
    fullFileNameFaces = fullfile(myFolder, baseFileNameFaces);
    fprintf(1, 'Now reading %s\n', fullFileNameFaces);
    imageArray{k} = imread(fullFileNameFaces);
    imshow(imageArray{k}); % Display image.
    drawnow; % Force display to update immediately.
end
cd ..
for k = 1:length(jpgFiles)
```

```
    name = jpgFiles(k).name;
    name = name(1:end-5);
    test_images.name{k} = name;
    test_images.faces{k} = imageArray{k};
end
```

6.6 Function: PCArecognition

```
function training = PCArecognition(trainData,k)

D = zeros(length(trainData.faces),64*64);
p = length(trainData.faces);
for i = 1:p

    face = trainData.faces{i};
    face = face';
    columnFace = face(:)';
    % creating the data matrix
    D(i,:) = columnFace;
    name = trainData.name{i};
    % creating a label matrix
    L{i} = name;
end
% calculating the mean
Dmean = mean(D);
Dc = zeros(size(D));
for i = 1:size(D,2)

    Dc(:,i) = D(:,i) - (Dmean(i)*ones(size(D,1),1));
end

Cov = (1/p-1)*(Dc * Dc');

[V,Dev] = eig(Cov);

Vsorted = V;

projM = Vsorted(:,1:k);
```

```
projMreduced = D' * projM;

for i=1:k
    projMreduced(:,i)=projMreduced(:,i)/norm(projMreduced(:,i));
end

Ft = D * projMreduced;

training.features = Ft;
training.lables = L;
training.projecM = projMreduced;
end
```

6.7 Function: FindFace

```
function accuracy =
    FindFace(trainData,testData,train_output,top3FLAG)
    testD = zeros(length(testData.faces),64*64);
    p = length(testData.faces);
    for i = 1:p
        face = testData.faces{i};
        face = face';
        columnFace = face(:)';
        % creating the data matrix
        testD(i,:) = columnFace;
        name = testData.name{i};
        % creating a label matrix
        Ltest{i} = name;
    end
    errorCount = p;
    for q = 1:p
        testF = testD(q,:)*train_output.projecM;
        dist = zeros(size(train_output.features,1),1);
        for i = 1:size(train_output.features,1)
            trainF = train_output.features(i,:);
            dist(i) = sqrt(sum((trainF - testF).^2));
        end
        [sorted,idxmin] = sort(dist);
```

```
        top1match = trainData.name{idxmin(1)};
        top2match = trainData.name{idxmin(2)};
        top3match = trainData.name{idxmin(3)};
        confidence = (abs(sorted(1) -
            sorted(2)))/sorted(1)*100;
        personTrue = Ltest{q};
        matchTrue = top1match(1:3) == personTrue(1:3);
        match2True = top2match(1:3) == personTrue(1:3);
        match3True = top3match(1:3) == personTrue(1:3);
        matchFalse = matchTrue(matchTrue == 0);
        match2False = match2True(match2True == 0);
        match3False = match3True(match3True == 0);

        trueFace = cell2mat(testData.faces(q));
        matched1Face = cell2mat(trainData.faces(idxmin(1)));
        matched2Face = cell2mat(trainData.faces(idxmin(2)));
        matched3Face = cell2mat(trainData.faces(idxmin(3)));

        if(top3FLAG)
            check = (~isempty(matchFalse)&&
                ~isempty(match2False) && ~isempty(match3False));
        else
            check = ~isempty(matchFalse);
        end
        if (check)
        errorCount = errorCount - 1;
        end
    end
    accuracy = 100 * (errorCount/p) ;
    fprintf(1, 'accuracy = %4.2f %%\n', accuracy);
end
```

6.8 GUI m-file: gui

```
function varargout = FaceRecognitionGui(varargin)
% GUI MATLAB code for gui.fig
%      GUI, by itself, creates a new GUI or raises the existing
%      singleton*.
```

```
%
%      H = GUI returns the handle to a new GUI or the handle to
%      the existing singleton*.
%
%      GUI('CALLBACK',hObject,eventData,handles,...) calls the
    local
%      function named CALLBACK in GUI.M with the given input
    arguments.
%
%      GUI('Property','Value',...) creates a new GUI or raises
    the
%      existing singleton*. Starting from the left, property
    value pairs are
%      applied to the GUI before gui_OpeningFcn gets called. An
%      unrecognized property name or invalid value makes
    property application
%      stop. All inputs are passed to gui_OpeningFcn via
    varargin.
%
%      *See GUI Options on GUIDE's Tools menu. Choose "GUI
    allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help gui

% Last Modified by GUIDE v2.5 13-Jan-2019 12:15:52

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',    mfilename, ...
                   'gui_Singleton', gui_Singleton, ...
                   'gui_OpeningFcn', @gui_OpeningFcn, ...
                   'gui_OutputFcn', @gui_OutputFcn, ...
                   'gui_LayoutFcn', [] , ...
                   'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
```

```
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State,
        varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before gui is made visible.
function gui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject   handle to figure
% eventdata reserved - to be defined in a future version of
    MATLAB
% handles   structure with handles and user data (see GUIDATA)
% varargin  command line arguments to gui (see VARARGIN)

% Choose default command line output for gui
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes gui wait for user response (see UIRESUME)
% uiwait(handles.figure1);



% --- Outputs from this function are returned to the command
    line.
function varargout = gui_OutputFcn(hObject, eventdata,
    handles)
% varargout cell array for returning output args (see
    VARARGOUT);
% hObject   handle to figure
% eventdata reserved - to be defined in a future version of
    MATLAB
% handles   structure with handles and user data (see GUIDATA)
```

```
% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata reserved - to be defined in a future version of
    MATLAB
% handles    structure with handles and user data (see GUIDATA)
global fileName;global testImage;
global trainData;global train_output;global testData
[fileName,path] = uigetfile('test_images\*.jpg');
cd(path)
testImage = imread(fileName);
cd ..
axes(handles.axes1);
imshow(testImage)
title(['Test image: ' fileName(1:end-6)])
data = load('train_test_data.mat');
PCAtrain = load('train_output.mat');
trainData = data.trainData;
testData = data.testData;
train_output = PCAtrain.train_output;

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata reserved - to be defined in a future version of
    MATLAB
% handles    structure with handles and user data (see GUIDATA)
global fileName;global testImage;
global trainData;global train_output;
[m1face,m2face,m3face] =
    FindMatchedFace(testImage,fileName,trainData,train_output);
axes(handles.axes2);
imshow(m1face.image,[])
title(['1st match: ' m1face.name ' % = '
    num2str(m1face.percentage)])
```

```
axes(handles.axes3);
imshow(m2face.image,[])
title(['2nd match: ' m2face.name ' % = '
    num2str(m2face.percentage)])
axes(handles.axes4);
imshow(m3face.image,[])
title(['3rd match: ' m3face.name ' % = '
    num2str(m3face.percentage)])
```

```
% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata reserved - to be defined in a future version of
    MATLAB
% handles    structure with handles and user data (see GUIDATA)
data = load('train_test_data.mat');
PCAtrain = load('train_output.mat');
trainData = data.trainData;
testData = data.testData;
train_output = PCAtrain.train_output;
top3flag = get(handles.checkbox2,'value');
accuracy = FindFace(trainData,testData,train_output,top3flag);
a = ['Accuracy = ' num2str(accuracy) '%'];
set(handles.text2,'String',a);
```

```
% --- If Enable == 'on', executes on mouse press in 5 pixel
    border.
% --- Otherwise, executes on mouse press in 5 pixel border or
    over text2.
function text2_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to text2 (see GCBO)
% eventdata reserved - to be defined in a future version of
    MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% --- Executes on button press in checkbox1.
function checkbox1_Callback(hObject, eventdata, handles)
% hObject   handle to checkbox1 (see GCBO)
% eventdata reserved - to be defined in a future version of
    MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox1


% --- Executes on button press in togglebutton1.
function togglebutton1_Callback(hObject, eventdata, handles)
% hObject   handle to togglebutton1 (see GCBO)
% eventdata reserved - to be defined in a future version of
    MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
    togglebutton1


% --- Executes on button press in checkbox2.
function checkbox2_Callback(hObject, eventdata, handles)
% hObject   handle to checkbox2 (see GCBO)
% eventdata reserved - to be defined in a future version of
    MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox2
```

6.9 GUI Function: FindMatchedFace

```
function [m1face,m2face,m3face] =
    FindMatchedFace(testFace,nameFace,trainData,train_output)

    Face = testFace';
    columnFace = Face(:)';
    columnFace = double(columnFace);
```

```
    testF = columnFace * train_output.projecM;
    dist = zeros(size(train_output.features,1),1);
    for i = 1:size(train_output.features,1)
         trainF = train_output.features(i,:);
         dist(i) = sqrt(sum((trainF - testF).^2));
    end
    [sorted,idxmin] = sort(dist);
    top1match = trainData.name{idxmin(1)};
    top2match = trainData.name{idxmin(2)};
    top3match = trainData.name{idxmin(3)};
    totalDistance = sorted(1) + sorted(2) + sorted(3);
    top1fraction = 50*(1 - (sorted(1)/totalDistance));
    top2fraction = 50*(1 - (sorted(2)/totalDistance));
    top3fraction = 50*(1 - (sorted(3)/totalDistance));
    matchTrue = top1match(1:3) == nameFace(1:3);

    matched1Face = cell2mat(trainData.faces(idxmin(1)));
    matched2Face = cell2mat(trainData.faces(idxmin(2)));
    matched3Face = cell2mat(trainData.faces(idxmin(3)));

    m1face.name = top1match;
    m1face.image = matched1Face;
    m1face.percentage = top1fraction;

    m2face.name = top2match;
    m2face.image = matched2Face;
    m2face.percentage = top2fraction;

    m3face.name = top3match;
    m3face.image = matched3Face;
    m3face.percentage = top3fraction;
end
```