FACE RECOGNITION - PCA

APPLIED MATHEMATICS

PRAMITA WINATA
RICHA AGARWAL

INTRODUCTION

The objective of this project is to apply PCA (Principal Component Analysis) in Face Recognition application. Face recognition has been one of the famous field in Computer Vision. In a nut shell, Face Recognition application will extract a set of a particular pixel in the image and try to find the best match with a database of images. Before we can do PCA, we need to detect a face in the image and make the normalized image from that. In this project, detecting the face is done manually. By manually defined the features location in the images, we can proceed to the next step. Normalization will be based on the affine transformation. After these steps, we can proceed to apply PCA for face recognition.

NORMALIZATION

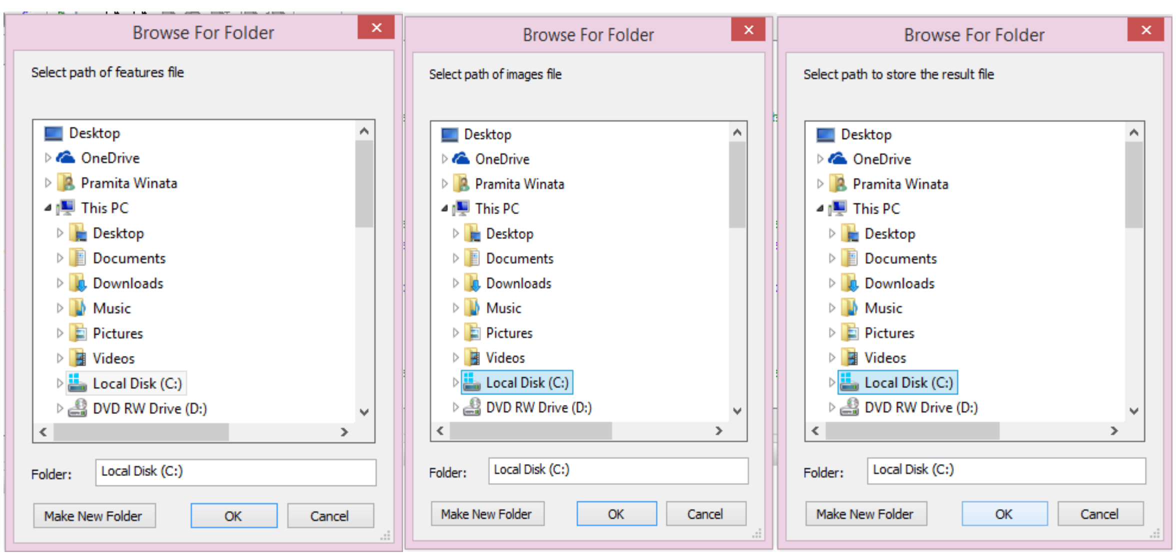
Normalization of the images is required before applying PCA because we need to consider the scale, orientation and location differences. So, we need to map the facial features to a predetermined location in a specific window. We choose 64x64 window in this case. We will use the affine transformation for this. Below, we will describe the implementation of the iterative algorithm in MATLAB. We follow the steps given in the Homework problem description to do this.

1. Initiation

   We need to have set of images with their corresponding features coordinate.

```
%Get the path where the features are and where the images will be stored
datapath = uigetdir('C:\','Select path of features file');
datapath3 = uigetdir('C:\','Select path of images file');
datapath2 = uigetdir('C:\','Select path to store the result file');
```

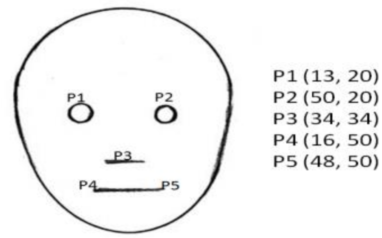
These will pop up 3 dialogs, where we can specifies the path of the corresponding requirement.



2. Use a vector $\bar{F}$ to store the average locations of each facial feature over all face images; initialize $\bar{F}$ with the feature locations in the first face image F_1. Initial value is given in below picture.

```
% set the predetermine location. We can set it to
the feature of first but
% need to rescale it to 64X64
F_predetermined = [13 20;
                   50 20;
                   34 34;
                   16 50;
                   48 50;]';

%1. Use a vector F to store the average locations of
each facial feature over all face images;
%initialize F with the feature locations in the
first image F1.
F_average = F_predetermined;
```



2. Compute the best transformation that aligns F (i.e., average locations of the features) with predetermined positions in the 64 × 64 window. The affine transformation can be defined by six parameters $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$, $and\ b \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$, with $F_i^P = A\, F_i + b$. We have 5 features; left eyes, right eyes, nose, left mouth and right mouth. We consecutively defined them as $\overline{X_1}, \overline{X_2}, \overline{X_3}, \overline{X_4}, \overline{X_5}$. We apply the transformation to this features, we get:

$$X_1^P = A\, \overline{X_1}, +b\ ;$$

$$X_2^P = A\, \overline{X_2}, +b\ ;$$

$$X_3^P = A\, \overline{X_3}, +b\ ;$$

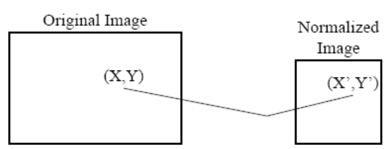
$$X_4^P = A\, \overline{X_4}, +b\ ;$$

$$X_5^P = A\, \overline{X_5}, +b\ .$$

Each features have X and Y coordinates, so we will have 10 equations. After computing the matrices, we can separate the x and y, so we will have a two new equations, $P\, v_1 = \ p_x\ and\ F\, v_2 = \ p_y,\ with:$

$$P = \begin{bmatrix} \overline{X_1} & \overline{Y_1} & 1 \\ \overline{X_2} & \overline{Y_2} & 1 \\ \overline{X_3} & \overline{Y_3} & 1 \\ \overline{X_4} & \overline{Y_4} & 1 \\ \overline{X_5} & \overline{Y_5} & 1 \end{bmatrix},\ p_x = \begin{bmatrix} X_1^P \\ X_2^P \\ X_3^P \\ X_4^P \\ X_5^P \end{bmatrix},\ p_y = \begin{bmatrix} Y_1^P \\ Y_2^P \\ Y_3^P \\ Y_4^P \\ Y_5^P \end{bmatrix},\ v_1 = \begin{bmatrix} a_{11} \\ a_{12} \\ b_1 \end{bmatrix},and\ \ v_2 = \begin{bmatrix} a_{21} \\ a_{22} \\ b_2 \end{bmatrix}$$

Need to be noted that when applying the affine transform, each pixel should be computed using the inverse of affine transformation to avoid gaps.



After getting this, we solve it using SVD then we apply it on $\bar{F}$; $\bar{F}'$. Update $\bar{F} = \bar{F}'$.

```
file = textread(str, '%s', 'delimiter', ' ', 'whitespace', '');
%store the read file in a matrix
x1 = str2double( file(1) ); y1 = str2double( file(2) );
x2 = str2double( file(3) ); y2 = str2double( file(4) );
x3 = str2double( file(5) ); y3 = str2double( file(6) );
x4 = str2double( file(7) ); y4 = str2double( file(8) );
x5 = str2double( file(9) ); y5 = str2double( file(10) );

% Get the best transformation of image F_i with the affine transformation parameter
% Affine transformation can be defined by six parameters A and b
%      F_predetermined_i = A*Fi , where A = [a11 a12 ; a21; a22 ], b = [b1 b2]
% P_i c1 = fx , P_i c2 = fy , affine = [c1;c2],where c1 = [a11; a12; b1] , c2 = [a21;
a22; b2]
% P_i * affine = F_i ,  with P_i is
P_i = [   x1 y1 1;
          x2 y2 1;
          x3 y3 1;
          x4 y4 1;
          x5 y5 1;
      ]';

%Solve the affine transformation with SVD
%Once the transformation is obtained, we apply it on F_average.
%3. For every face image F_i , compute the best transformation that aligns the facial
features of F_i with
%the average facial feature F_average; called the aligned features F'_i.
affine =  F_average * pinv(P_i);

F_i_aligned = affine * P_i;

% Noting the first aligned features F_average ,
% we update F_average by setting F_average = F_i_aligned.
if (  first_iteration == 0  && j == 0)
   F_average = F_i_aligned;
   first_iteration = first_iteration +1;
end
```

3. For every face image F_i, compute the best transformation that aligns the facial features of F_i with the average facial features $\bar{F}$; called the aligned features F'_i.

   We created new function to apply the affine transformation for the image.

```
function [  ] = affine_transform( fileName, facesPath, affine, resultPath )
imagePath = fullfile(facesPath ,fileName);
I = imread(imagePath);
I = rgb2gray(I);


% Ab - trasformation that aligns features of image F_i with F_predetermined
% A = [a11 a12 ; a21; a22 ],% b = [b1 b2]
% affine = [c1;c2] = [ a11 a21; a12 a22; b1 b2]
A = affine([1, 3; 2 4]);
b = affine([5; 6]);

result = zeros(64, 64);
for x = 1:64
    for y = 1:64
        %calculate the inverse of the affine transformation
        %int32(A \ ([x ; y] - b))
        out_pixel = int32(A \ ([x ; y] - b));
        if (out_pixel(1) <= 0 || out_pixel(1) >= 240 || out_pixel(2) >= 320 ||
out_pixel(2) <= 0)
            pixel_value = 254;
        else
            pixel_value = I(out_pixel(2), out_pixel(1));
        end
        result(x, y) = uint8(pixel_value);
    end
end
result = mat2gray(result');

testPath = fullfile(resultPath,fileName);
imwrite(result, testPath);

end
```

4. Update $\bar{F}$ by averaging the aligned feature locations F'_i for each face image F_i

5. Compute the error. If the error between F_t and F_{t-1} is less than a threshold, then stop; otherwise, go to step 2.

```
%4. Update F_average by averaging the aligned feature locations F'i for each
face image Fi.
    F_average = temp / totalFile;

%find difference between current and previous average, if its small stop
%5. If the error between F_average_t and F_average_t+1 is less than a
threshold, then stop;  otherwise, go to step 2.
difference_matrix = abs( (temp / totalFile) - F_average);
%maximum value in the difference matrix
difference = max(difference_matrix(:));
```

The result of applying the code can be seen below :

PART 2 : PCA [PRINCIPAL COMPONENT ANALYSIS]

PCA to transform our correlated data set into smaller uncorrelated data set. So PCA helped us on reducing the large dimensions of data set into smaller dimensions of it.

After finding F, we followed these steps:
We converted the 64 by 64 normalized images X_i into row vector sized 1x4096 and constructed matrix Ipxd(p = number of images, d = 64x64) that each row corresponds to Xi. Then we defined k between 30 an d 35 to calculated k_{th} largest eigen values.

Before computing the PCA, we computed the covariance matrix of D as follows:

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

Then we used matlab's "eig ()" function to get the eigen vectors from $\sum$ matrix. After that we calculated the covariance matrix of eigen values and normalized it before projection, now we get the projection matrix using the first k eigen vectors . so we are reducing the big space to less number of principal components.

$$\Phi_i = X_i * \Phi$$

So PCA space represent the linear combination of eigenfaces of training images, that is for any training image, we just need know what is its linear combination of eigenfaces. So as we have created the database of all the images in projection matrix so we can get the matching image using the Euclidian distance of each test image and projection matrix which is discussed in next part.

To get the PCA we made the following matlab function:

```
function [Proj_tarinData,Labels,firstEig_vec,meanX] = MyPca(K_pca,trainData)
%Pca_recognition Summary of this function goes here
% we need to match the normalized images in the training set to the
% images in test set.
%clear all; close all; clc;



file = dir(trainData);
% file is a Lx1 structure with 4 fields as: name,date,byte,isdir of all L files
present in the directory 'data_train'

Labels = [];            %to define labels of each image to use for regognition part

%each row of X represents one image in training set
X = [];

%to get training images and have them in matrix X,TO store the name of each
%image in label
for i = 1:length(file)
    if (isempty(strfind(file(i).name, '.png'))) == 0 || (isempty(strfind(file(i).name,
'.jpg'))) == 0
        imageName = fullfile(trainData,file(i).name);
        imageread = imread(imageName);
        X = [X; imageread(:)'];
        Labels = [Labels; cellstr(file(i).name)];
    end
end

%caluculating the mean and removing it from the images
meanX=double(mean(X));
%covariance matrix
for i = 1:size(X,1)
    Xmean(i,:)=double(X(i,:))-meanX;
end
 %Normalizing covariance
sigma=(1/size(X,1))*Xmean*Xmean';

%eigenvectors eigenvalues
[eig_vec, eig_val] = eig(sigma);

%covariance matrix of eigen values
eig_vec = Xmean' * eig_vec;

%normalizing the covariance matrix before projection
for i = 1:size(eig_vec, 2)
    eig_vec(:, i) = eig_vec(:, i)/norm(eig_vec(:, i));
end

%projection  matrix using top k principal components
firstEig_vec = eig_vec(:, 1:K_pca);
%creating the training datbase
Proj_tarinData = Xmean* firstEig_vec;
```

RECOGNITION ::

To find a image matching in the training set we the project this image into the projection data we created in the above step so we get Φ_j and then we calculate the Euclidean distance between Φ_j and the projection matrix. We sort the Euclidian distance we got Euclidian distance values tells us how much similar the test image is in the projection set so we are interested in the least n values of Euclidean distance. Now we can get the index for each least distance and find the corresponding image in the training database. Then we can do the matching images for a particular test image. Below is the MATLAB function's section:

```
function accuracies =
face_recognition_accuracy(k_pca,testImage,similar_images,testData,trainData)

% Projection matrix from traing datbase
[Proj_tarinData, Labels, firstKEig_vec, mean_train] = MyPca( k_pca ,trainData );
% Face recognition%%
%test image's principal component

phi_j = (double(testImage(:)') - mean_train) * firstKEig_vec;

%on the basis of eucladian distance finding the most similar face in the
%traing database

% finding most similar image from training set
        for j=1:size(Proj_tarinData,1)
            euc_dis(j) = sqrt(sum((Proj_tarinData(j,:)-phi_j).^2));
        end
sorteuc_dis = sort(euc_dis);

%find m most similar images to test one
face_index = find(euc_dis <= sorteuc_dis(similar_images));
selected_images = Labels(face_index);

recognised_images = [];

for i = 1:length(selected_images)
    Image = char(selected_images(i));
    similarImg = fullfile(trainData,Image);
    recognised_images = [recognised_images imread(similarImg)];
end
```

In testing the algorithm we found that if the test image in angular image of a face and training set has front view of the face we are less likely to get better result for n similar images.

Also number if we increase more number of similar images we get better results. We checked the accuracy on the algorithm which is approx 81 % for 20 principal components for 4 similar images.

```
accuracies = [];
for n_images = 1:5

    files_read = dir(testData);

Errors = 0;

%to read all the traing images and test images similar to above
%procedure...
for k = 1:length(files_read)
    readFile = files_read(k).name;

    if (isempty(strfind(readFile, 'jpg'))) == 0 || (isempty(strfind(readFile, 'png')))
== 0
        imagek = fullfile(testData,readFile);
        test_image = imread(imagek);

        phij = (double(test_image(:)') - mean_train) * firstKEig_vec;
         for j=1:size(Proj_tarinData,1)
            eucliDist(j) = sqrt(sum((Proj_tarinData(j,:)-phi_j).^2));
        end
        sortedeucliDist = sort(eucliDist);

        ind = find(eucliDist <= sortedeucliDist(n_images));

        rec_image = Labels(ind);

        p1 = strfind(readFile, '.') - 2;

        matched = 0;
        for i = 1:length(rec_image)
           read_im = char(rec_image(i));
           p2 = strfind(read_im, '.') - 2;

           if strcmp(readFile(1:p1),read_im(1:p2)) == 1
               matched = 1;
           end
        end

        if matched == 0
            Errors = Errors + 1;
        end
    end
end

accuracy = (1 - Errors/size(Proj_tarinData, 1)) * 100;

accuracies = [accuracies accuracy];

end
```

PART 3: CLASSIFICATION:

We used Nearest Neighbor Algorithm to check the gender of the image. We already have labeled each image so we create two different sets , one having the female index as we can explicitly initialize and another the male set which have all the images except the females. Now we go with the previous logic and calculate the Euclidean distance between the test image to male set and female set. Now the probability of the female image is more if the Euclidean distance with the female set is less than the male set.
As shown below:

```
% Sex recognition using Nearest neighbour we craete two sets male and
% female
set_F = {'Flavia','Richa', 'Pramita'};
f_index = [];

for i = 1:length(set_F)
    f_index = [f_index, strmatch(char(set_F(i)), Labels)];
end

female_set = Proj_tarinData(f_index, :);
% male_set = Proj_tarinData(setdiff(1:size(Proj_tarinData, 1), f_index), :);
index_set=1:length(Labels);
index_set(f_index)=[];
male_set = Proj_tarinData(index_set,:)
%finding average distance between test image and male/female
%training set
for k=1:size(female_set,1)
            euclDisF(k) = sqrt(sum((female_set(k,:)-phi_j).^2));
end

for l=1:size(female_set,1)
            euclDisM(l) = sqrt(sum((female_set(l,:)-phi_j).^2));
        end

dis_fem = mean(mean(euclDisF))
dis_m = mean(mean(euclDisM))

sex = 'male';
if (dis_fem < dis_m)
    sex = 'female';
end
```

EXPERIMENTAL RESULTS::

We checked the normalization and face recognition on the data base of 36 images and with a test image we got the following result:

CHAPTER 6: SUMMARY

This project help us understand a great idea that PCA provide: transform the data from correlated to independent. From image processing point of view, Face recognition based on PCA actually represent a face by the combination of eigenfaces. It is a powerful method to reduce the dimension of problem.
We also got some deeper understanding of the application of some linear algebra tools such as eigenvalue, eigenvector, SVD, etc. Specifically, we got the point that why and how we use SVD.

REFERENCES

1. Desire's Notes - http://mscvision.u-bourgogne.fr/wp-content/uploads/2011/09/applied_maths1.pdf
2. https://en.wikipedia.org/wiki/Principal_Component_Analysis
3. https://en.wikipedia.org/wiki/Eigenface
4. https://en.wikipedia.org/wiki/Eigenvector