Face Recognition Using Eigen Faces Algorithm

Arnav Kumar Agrawal **(201201075)**
Saurabh Kathpalia **(201201039)**

I. Objective

The main aim of the project is to recognize face of the input image, using Eigen Faces, of a person for whom we already have some other images in the training set database. This is done by reconstructing the face in the input image using a blend of training set images.

II. Database Used

The database used is the set of images in (.jpg format) of various individuals. It may be coloured or not. All images should have a face with sufficient light. We have collected around 27 images of different individuals gathered from Facebook.

III. Method

Training Part

First the images are converted into grayscale images and also the size of image is changed to that of images in the training set as we are dealing with images with some fixed size. Also the 2D matrix of image is converted into 1D matrix by simple concatenation of rows.

$$\mathrm{I_i} = \begin{bmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NN} \end{bmatrix} \xrightarrow{concatenation} \begin{bmatrix} a_{11} \\ . \\ . \\ . \\ a_{1N} \\ . \\ . \\ . \\ a_{NN} \end{bmatrix} = T_i$$

$$\mathrm{S} = \{\mathrm{T}_1,\ \mathrm{T}_2,\ \mathrm{T}_3, \ldots\ldots, \mathrm{T}_\mathrm{M}\}$$

Here M is the number of training images and each variable is a vector.

Then in the training part we normalize the images and also calculate the mean (**Ψ**) of all the images. This is done to reduce the error due to lightning conditions. In the mean part the mean of each row is taken and then each obtained mean is converted to a 8 bit unsigned integer (0-255). As the mean is taken in rows so for every image we have a set of some “n” numbers where n is the number of columns which is same for each image.

$$\Psi = \frac{1}{M}\sum_{n=1}^{M} T_n$$

Once we have the mean image then we calculate the difference between each image and the mean image.

$$\phi_i = T_i - \Psi$$

Then covariance Matrix C is calculated

$C = AA^T$ where $A = [\Phi_1, \Phi_2, \Phi_3, \ldots \Phi_M]$

C is an $N^2 x\ N^2$ Matrix

Also we have another matrix $L = A^T A$ which MxM Matrix.

In order to find the Eigen vectors of C Matrix we find the Eigen values of L Matrix which is easier to find compared to that of C matrix as M << N^2

Then Eigenvectors of C are calculated and then normalised.

Now each face in the training set (minus the mean), ϕ_i can be represented as a linear combination of these Eigen Vectors u_i.

$$\phi_i = \sum_{j=1}^{M} w_j u_j$$

where u_j's are Eigen Vectors and K is M only

Each normalized training image is represented in this basis as a vector.

$$\Omega_i = \begin{bmatrix} w_1 \\ w_2 \\ . \\ . \\ . \\ w_M \end{bmatrix}$$

Testing Part (Recognition Part)

We now take input from the user inform of image that has to be recognized and then the aim is to calculate the corresponding weight matrix.

Let T denote the 1D matrix of the input image. We take the difference between the input image and the mean image and then normalize it.

$$\Phi = T - \Psi$$

Then we find the weights corresponding to each image in the training set.

$$w_i = u_i^T \phi$$

Now we have the matrix of weights. So we can construct the image by using this matrix and the eigenvectors we have computed earlier.

This matrix of weights is then used to find Euclidean distance between the database images and input image. We can check if a heuristically defined variable lies between the minimum and maximum threshold and then conclude the recognition accordingly.

Pseudo Code

Number of images on your training set.

```
M=27;
```

Chosen standard deviation and mean for normalization.
It can be any number that it is close to the standard deviation and mean of most of the images.

```
um= 100.0569;
ustd=79.7172;
```

Read images

Here we change the mean and std of all images. We normalize all images.
This is done to reduce the error due to lighting conditions.

```
for i=1:size(S,2)
    temp=double(S(:,i));
    m=mean(temp);
    st=std(temp);
    S(:,i)=(temp-m)*ustd/st+um;
end
```

Mean image

```
m=mean(S,2);   %obtains the mean of each row instead of each column
tmimg=uint8(m);   %converts to unsigned 8-bit integer. Values range from 0 to 255
img=reshape(tmimg,icol,irow);
%takes the N1*N2x1 vector and creates a N2xN1 matrix
img=img';       %creates a N1xN2 matrix by transposing the image.
figure(3);
imshow(img);
title('Mean Image','fontsize',18)

% Change image for manipulation
dbx=[];   % A matrix
for i=1:M
    temp=double(S(:,i));
    dbx=[dbx temp];
end
```

Covariance matrix C=A'A, L=AA'

```
A=dbx';
L=A*A';
% vv are the eigenvector for L
% dd are the eigenvalue for both L=dbx'*dbx and C=dbx*dbx';
[vv dd]=eig(L);
% Sort and eliminate those whose eigenvalue is zero
v=[];
d=[];
for i=1:size(vv,2)
    if(dd(i,i)>1e-4)
        v=[v vv(:,i)];
        d=[d dd(i,i)];
    end
 end
```

```
 %sort,  will return an ascending sequence
 [B index]=sort(d);
 ind=zeros(size(index));
 dtemp=zeros(size(index));
 vtemp=zeros(size(v));
 len=length(index);
 for i=1:len
    dtemp(i)=B(len+1-i);
    ind(i)=len+1-index(i);
    vtemp(:,ind(i))=v(:,i);
 end
 d=dtemp;
 v=vtemp;
```

Normalization of eigenvectors

```
 for i=1:size(v,2)        %access each column
   kk=v(:,i);
   temp=sqrt(sum(kk.^2));
   v(:,i)=v(:,i)./temp;
end

%u is the Eigenvectors/eigenfaces of C matrix
u=[];
for i=1:size(v,2)
    temp=sqrt(d(i));
    u=[u (dbx*v(:,i))./temp];
end
```

Normalization of eigenvectors

```
for i=1:size(u,2)
   kk=u(:,i);
   temp=sqrt(sum(kk.^2));
    u(:,i)=u(:,i)./temp;
end
```

Display the eigenfaces

```
figure(4);
for i=1:size(u,2)
    img=reshape(u(:,i),icol,irow);
    img=img';
    img=histeq(img,255);

subplot(ceil(sqrt(M)),ceil(sqrt(M)),i)
    imshow(img)
    drawnow;
    if i==3
```

```
title('Eigenfaces','fontsize',18)
    end
end
```

Find the weight of each face in the training set.
omega are weight of images

```
omega = [];
for h=1:size(dbx,2)
    WW=[];
    for i=1:size(u,2)
        t = u(:,i)';
        WeightOfImage =
dot(t,dbx(:,h)');
        WW = [WW; WeightOfImage];
    end
    omega = [omega WW];
end
```

Acquire new image

```
img=InputImage;
tempsize = size(img);
l= size(tempsize);
if l(2) == 3
    img = rgb2gray(img);
    newvar = imresize(img, [292
376]);
    img=newvar;
    mystring =
strcat(int2str(i),'.jpg');
end
InputImage = img;
imshow(InputImage);
colormap('gray');title('Input
image','fontsize',18)
InImage=reshape(double(InputImage)'
,irow*icol,1);
temp=InImage;
me=mean(temp);
st=std(temp);
temp=(temp-me)*ustd/st+um;
NormImage = temp;
Difference = temp-m;

p = [];
aa=size(u,2);
for i = 1:aa
    pare = dot(NormImage,u(:,i));
    p = [p; pare];
end
ReshapedImage = m + u(:,1:aa)*p;
%m is the mean image, u is the
eigenvector
ReshapedImage =
reshape(ReshapedImage,icol,irow);
ReshapedImage = ReshapedImage';
```

We have the reconstructed image.

```
subplot(1,2,2)
imagesc(ReshapedImage);
colormap('gray');
title('Reconstructed
image','fontsize',18)

InImWeight = [];
for i=1:size(u,2)
    t = u(:,i)';
    WeightOfInputImage =
dot(t,Difference');
    InImWeight = [InImWeight;
WeightOfInputImage];
end

ll = 1:M;
figure(68)
subplot(1,2,1)
stem(ll,InImWeight)
title('Weight of Input
Face','fontsize',14)
```

Find Euclidean distance

```
e=[];
for i=1:size(omega,2)
    q = omega(:,i);
    DiffWeight = InImWeight-q;
    mag = norm(DiffWeight);
    e = [e mag];
end

kk = 1:size(e,2);
subplot(1,2,2)
stem(kk,e)
title('Eucledian distance of input
image','fontsize',14)

MaximumValue=max(e) %max value of
eculidean distance from database
images
MinimumValue=min(e) %min value of
euclidean distance from database
images
```

IV. Results

Input Image

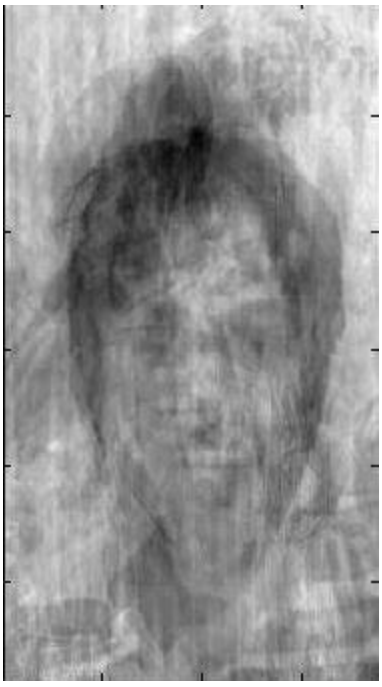
Reconstructed Image

Input Image

Reconstructed Image

Input Image

Reconstructed Image



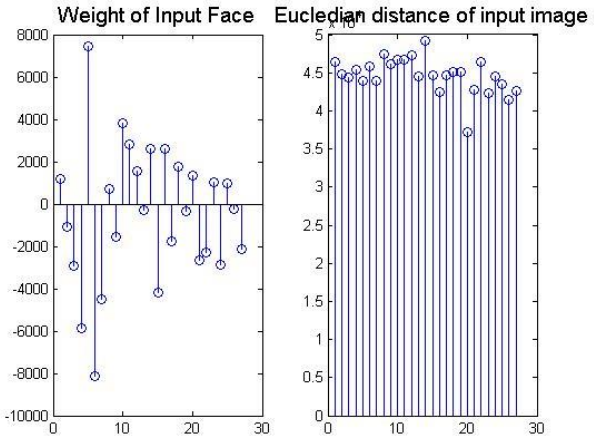
Input Image

Reconstructed Image

V. Explanation/Analysis

In the first, second and third case the top image is the input image and the bottom one is the reconstructed image. In this some other image of the person is there in the training set. The more similar the input

image is similar to the one in dataset the better the reconstructed image is. In the last case the input image is same as that of an image that is present in the training set, so we almost get the same image in the output. For the first image we plot the weights of the input image and the Euclidean distance. The result can be seen as follows:



VI. References

[1] http://jmcspot.com/Eigenface/Default

[2] http://onionesquereality.wordpress.com/2009/02/11/face-recognition-using-eigenfaces-and-distance-classifiers-a-tutorial/