Analysis and Implementation of Eigenfaces Method of Facial Recognition

Vaghesan Sundaram

January 22, 2024

1 Introduction

Authentication online is becoming ever present when visiting web pages, or when handling sensitive digital information. Being able to use a computer to recognize a face, which can't be easily faked, is incredibly useful. One approach to this is to use eigenfaces, which are sets of eigenvectors calculated from a set of face images. These eigenfaces can be used to recognize faces in the set of face images. The method was developed by Sirovich and Kirby, and it was used by Turk and Pentland.

2 Eigenface Generation

A set of training images must be prepared. We will be utilizing 187 images from the Chicago Face Database for this, as well as inserting my own face into the set to compare with another image later. Each image will be cropped to align the eyes and mouth, as well as be converted to a 64x64 resolution. A matrix T will be created, with each image vector being a column in T. By performing Principal Component Analysis and Singular Value Decomposition, a set of eigenfaces can be generated.

1. Subtract the mean image from each image in T.
2. Calculate the eigenvalues and eigenvectors of the covariance matrix S. Using S in computations can be infeasible, so we will use SVD to take the eigenvalue decomposition without computing S.
3. Determine the principal components that represent 95% of variance in all faces and remove components that are not needed.

T will contain the eigenfaces that represent 95% of variance in all faces in the original set. For actually identifying faces, we need to find the weight of each eigenface for every image in the database, as well as for the image we want to identify. Weight vectors can be calculated from input vectors, which can be compared to the weight vectors of images in the database by calculating the Euclidean distance.

3 Finding Eigenfaces with Octave

Let us generate the eigenvectors and eigenvalues for T, containing 188 faces. To start, we will take our images and crop them to align the eyes and mouth, as well as and re-scale them to a 64x64 resolution.

We can use Octave code to convert each image to a 64 x 64 x 3 matrix, convert it to a 4096 x 1 matrix while gray-scaling it, and put it in a matrix T.

```
dirlist = glob("*.jpg");
T = []
for i = 1:length(dirlist) $Iterates through each image in image folder

  dirname = dirlist{i,1};
```

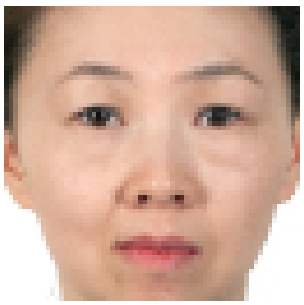

Figure 1: Image 1 in the database, before vector conversion

```
    img = double(imread(dirname)); $64 x 64 x 3 matrices

    vec = squeeze(mean(img,3)); $Gray-scales
    vec = vec(:)  $converts to 4096 x 1 matrix
    T = horzcat(T,vec)  $Appends each image vector to T

endfor
```

We can calculate the mean image using, and subtract the mean image from each column in T. We still need to keep the matrix of original matrices for later.

```
meanimage = mean(T, 2)
images = T
T = bsxfun(@minus, T, meanimage) $subtracts the mean image form each column in T
```

Figure 2: The mean face vector converted back to a gray-scale image

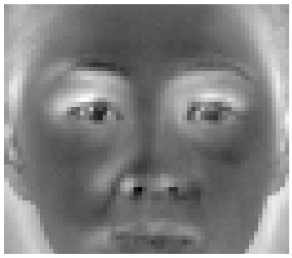

Figure 3: The mean image subtracted from image 1

Now we must calculate the eigenvalues and eigenvectors of the covariance matrix, which is equal to TT^T. Because this is a computationally expensive step, we will use SVD. Octave has a function that return the matrices U, S, and V. U contains the eigenvectors of TT^T as columns, and S is a diagonal matrix that contains the eigenvalues. V contains the eigenvectors of T^TT, which is not needed.

```
[U,S,V] = svd(T,0);
```

Now that we have our eigenvalues and eigenvectors, we can determine the principal components that make up 95% of the variance. This allows us to essentially throw out the eigenvectors that have little effect on variance. We can store the number of principal components in k95.

```
S = diag(S)'
eigsum = sum(S)
csum = 0;
for i = 1:columns(S)
  csum = csum + S(i)
  tv = csum/eigsum
  if tv > 0.95;
    k95 = i
    break

  end;
end;

S = S(1:k95)
U = U(:,1:k95)
```

4 Application in Facial Recognition

Images in the database can be saved as a collection of weights describing each eigenface's contribution. When identifying an image, it's weights are calculated and compared to every other face in the database. The comparison is done by calculating the euclidean distance between each weight vector. We will first make a matrix of the weight vectors for all 188 images.

```
weights = zeros(columns(S), columns(images))
for i = 1:columns(images)
  for j = 1:k95
    weights(j,i) = U(:,j)' * (images(:,i)-meanimage);
  end;
end;
```

We will also make a weight vector of the face image we want to identify.

```
im2 = double(imread(facetobeidentified));
idenFace = squeeze(mean(im2,3));
idenFace = idenFace(:);

weightsIden = zeros(k95,1)
for i = 1:k95
  weightsIden(i) = U(:,i)' * (idenFace-meanimage);

end
```

The final step is to calculate the euclidean distances between the weight vectors of the image we want to identify and the images in the database. The image with the least distance in the database is the one most likely to be the same person in the image we want to identify. However, if the minimum euclidean distance is not less than 2000, there is a high change that the face is not in the database. If the euclidian distance is more than 8000, there is a high chance that the image is not a face.

```
distances = zeros(columns(images),1)
for i = 1:columns(images)
  distances(i) = norm(weightsIden - weights(:,i));
end

[minval, idx] = min(distances)
if minval > 8000
    idx = -1;
end
if minval > 2000
    idx = 0;
end
```

The variable idx will store the index of the most similar database picture, corresponding to which column it represents in the images matrix. A system for pairing a name to each image can be created, outputting a name instead of an index. We can test by inputting an image to be tested.

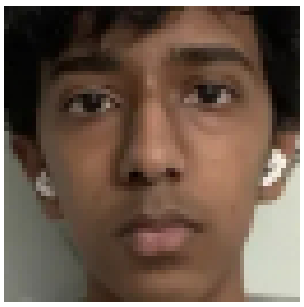

Figure 4: Image 188 in the database

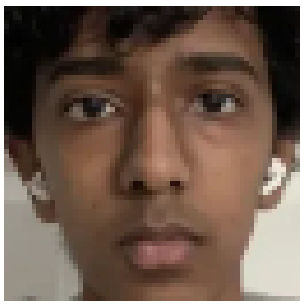

Figure 5: Image to be identified

After calculating our eigenvectors and values, finding weight vectors, and calculating euclidean distances, our 188 x 1 distances vector looks like:

$$distances = \begin{bmatrix} 6501.2 \\ 5911.7 \\ 5230.6 \\ 5671.1 \\ \vdots \\ 5655.5 \\ 5726.4 \\ 1197.1 \end{bmatrix}$$

The last element in the distances vector represents the distance between image 188 and the image to be identified. Since we can verify that it is the lowest element in the vector, and with range to be

considered a candidate, we can identity the face in the image to be identified as the same face in image 188.

5 Analysis

One of the main merits of the eigenfaces approach is its ability to handle large face databases. Additionally, adding a picture to the database only requires adding the picture and it's weight vector; no other calculations are needed. This makes it very efficient computationally, especially with the implementation of SVD. Furthermore, the actual eigenface calculation process is simple, with SVD being the most complicated step.

However, the eigenfaces approach does have some shortcomings. It requires all images to have the eyes and mouth aligned, as well as have similar lighting. More often than not the most significant eigenfaces mainly have illumination encoding. Because of this, the first 3 eigenfaces are usually discarded in practice. Eigenfaces is also very sensitive to expression changes, and has difficulty capturing them. The Chicago Face Database contains faces with multiple emotions, so only faces with a neutral expression were used. Because of the difficulty in controlling the environment, other facial recognition methods are preferred, such as the fisherface method, which uses Linear Discriminant Analysis and handles lighting differences better. Approaches utilizing AI and Machine Learning have also shown remarkable accuracy.

6 Materials

The faces used for the eigenface generation and image identification, as well as the full Octave code, are located in the folder below:
Link to Google Drive.

References

[1] Jaadi, Zakaria. “A Step-by-Step Explanation of Principal Component Analysis (PCA).” Built In, 29 Mar. 2023, builtin.com/data-science/step-step-explanation-principal-component-analysis.

[2] Navarrete, Pablo, and Javier Ruiz-del-Solar. Analysis and Comparison of Eigenspace-Based Face ... - Uchile.Cl, www.cec.uchile.cl/ aabdie/jruizd/papers/ijprai2002.pdf. Accessed 22 Jan. 2024.

[3] Raj, Bhiksha. “Machine Learning for Signal Processing.” Assignment 2: Face Detection, 2012, www.cs.cmu.edu/ pmuthuku/mlsp_page/assignments/assignment2_hints.html.

[4] Wang, Yue. “SVD and Eigenfaces.” Nextjournal, 22 Feb. 2021, nextjournal.com/yuewangpl/svd-and-eigenfaces.