Feauture Detection using Eigenfaces

Nalin Ranjan and Sacheth Sathyanarayanan

May 14, 2019

1 Introduction

Computer vision has become an important part of our world, with numerous applications in security, navigation, analysis of scientific data, privacy, and industrial development. One of the most important problems in computer vision has been facial recognition, which similarly has been used in a wide variety of contexts - from securing personal devices to criminal recognition.[1]

In this project, we implement the concept of eigenfaces[2] to achieve this goal. The basic idea is to represent an image of a face as a vector, and form an M-dimensional subspace of $\mathbb{R}^N$, where N represents the number of pixels in the images and M is the number of samples in the data set. By quantifying the distance of test images to this M-dimensional subspace, our program can decide to decent accuracy which images of a specific format are human faces.

2 Theory

2.1 Input Data and Preprocessing

The first step of the process is to find a suitable training set which will be used to abstractly determine what patterns in the pixels are most indicative of facial features. These characteristics (elaboration to come) will be referred to as eigenfaces. For the purpose of illustration, let us have M images in our training set, all containing N pixels.

We will work with grayscale images, to simplify our input (this has been shown to be a valid simplification and does not detract from facial recognition capabilities). Let our training set of faces, represented as vectors in $\mathbb{R}^N$, be $\mathbf{F_1}, \mathbf{F_2}, \cdots, \mathbf{F_M}$. To normalize the data, we first find the mean of these eigenfaces $\overline{\mathbf{F}}$, and vector of standard deviations $\mathbf{S}$ (of each individual pixel). They are defined as

$$\overline{\mathbf{F}} = \frac{1}{M}\sum_{i=1}^{M} \mathbf{F_i},\ \mathbf{S} = \begin{bmatrix}\sigma_1 & \sigma_2 & \cdots & \sigma_N\end{bmatrix}^t \text{ with } \sigma_i = E(P_i^2) - E(P_i)^2$$

with P_i representing the i-th pixel in the images. We then proceed to define $\mathbf{\Gamma_i} = \text{div}(\mathbf{F_i} - \overline{\mathbf{F}}, \mathbf{S})$, where the operation div is element-wise division of the its arguments. The vector $\mathbf{\Gamma_i}$ can essentially be seen as a vector of z-scores of each pixel. From now on, we will be using these as our data points, as they have been standardized and can be compared to each other. They will allow succinct and efficient calculation of subsequent matrices.

2.2 Calculating Eigenfaces from Data

Treating each pixel as an independent random variable, we can calculate a covariance matrix between these random variables. We know that the covariance matrix can be represented as

$$C = \frac{1}{M}\sum_{i=1}^{M} \mathbf{\Gamma_i \Gamma_i}^t = FF^t$$

with $F = \begin{bmatrix}\mathbf{\Gamma_1} & \mathbf{\Gamma_2} & \cdots & \mathbf{\Gamma_M}\end{bmatrix}$. Using the fact that this covariance matrix is symmetric ($FF^t = (F^t)^t F^t = (FF^t)^t$), the spectral theorem for real matrices allows us to construct an orthonormal eigenbasis of faces that spans our subspace of faces. Though we would like to do so, even a relatively low-resolution image (200 pixels $\times$ 200 pixels) will yield a large covariance matrix (40000×40000 in this case). However, calculating the eigenvectors and eigenvalues of such a matrix is computationally infeasible.

We can use a clever observation to find some of the eigenvectors of $C = FF^t$, however. Let us consider the eigenvectors of F^tF, which is a significantly smaller $M \times M$ matrix. Note that if $F^tF\vec{v} = \lambda\vec{v}$, premultiplying our equation by F yields that $FF^tF\vec{v} = \lambda(F\vec{v})$. In other words, the $F\vec{v}$ are λ-eigenvectors of FF^t if the $\vec{v}$ are λ-eigenvectors of F^tF. We calculate these vectors $F\vec{v}_i$, and normalize them, forming the matrix $U = \begin{bmatrix}\vec{u}_1, \cdots, \vec{u}_M\end{bmatrix}$. ***The eigenvectors*** $\vec{u}_i = F\vec{v}_i/||F\vec{v}_i||$ ***are the eigenfaces calculated from the data set.***

We can take these observations a step further to realize that we can perform Principal Component Analysis (PCA) on our data set. As $C = FF^t$ is a symmetric matrix that is a product of a matrix and its transpose, we know that all of its eigenvalues are greater than or equal to zero. The corresponding eigenfaces $\vec{u}_i$ also are orthonormal:

$$\vec{u}_i \cdot \vec{u}_j = \frac{(F\vec{v}_i)\cdot(F\vec{v}_j)}{||F\vec{v}_i||||F\vec{v}_j||} = \frac{\vec{v}^tF^tF\vec{v}}{||F\vec{v}_i||||F\vec{v}_j||} = \frac{\lambda(\vec{v}_i\cdot\vec{v}_j)}{||F\vec{v}_i||||F\vec{v}_j||} = 0$$

as the $\vec{v}_i$ are orthogonal eigenvectors of the symmetric matrix F^tF. This gives rise to a key property: the variance in the direction of one eigenface is independent of the variance of all other eigenfaces. The eigenfaces here represent the principal components across which variation of key features is occurring, with importance given by the corresponding eigenvalue. Various eigenfaces will represent different "features," which may be easy to understand (e.g. the presence of eyes) or incredibly abstract and only intelligible to the computer.

We can further simplify our calculations without losing significant accuracy by choosing the k most significant eigenvectors (defined as the k eigenvectors whose eigenvalues are greatest). This is valid as variation in the direction of certain eigenvectors is much more important than others (in fact, choosing as little as $k = 10$ can yield pretty good results). Let these k eigenvectors be the columns of the matrix U_k.

2.3 Calculating the Distance of a Test Image

For a new image, we are interested in the linear combination of eigenfaces that represents the projection of its normalized form $\mathbf{\Gamma'} = \mathbf{F'} - \overline{\mathbf{F}}$ onto the subspace spanned by them. We know that the projection onto the subspace will be of the form

$$\text{proj}_{\mathcal{V}}(\mathbf{\Gamma'}) = \sum_{i=1}^{k}\omega_i\vec{v}_i = \sum_{i=1}^{k}(\vec{v}_i\cdot\mathbf{F'})\vec{v}_i$$

Figure 1: Example Training Data (sample of the images used to train the model)

Similarly, let us define the characteristic vector of data point i to be $\Omega_i = \begin{bmatrix}\omega_1 & \omega_2 & \cdots & \omega_k\end{bmatrix}^t$, where $\omega_i = \vec{v}_i \cdot \mathbf{\Gamma_i}$ (these are the coefficients of $\vec{v}_i$ in the projection of $\mathbf{\Gamma_i}$ onto the eigenface-subspace). We calculate the characteristic vector of each data point. To compare a new data point, we simply take the minimum Euclidean distance between its characteristic vector Ω and the Ω_i, i.e. $\min||\Omega - \Omega_i||$.

To minimize random error that arises when taking the simple minimum of this value, we choose to omit outliers. Specifically, we collect all the values $\delta_i = \min||\Omega - \Omega_i||$ and calculate the mean μ and standard deviation σ of this set. If $\delta_i - \mu > 2\sigma$, we omit the data point (equivalent to omitting all data points more than 2 standard deviations from the mean). This prevents probabilistic matching of an image, and is a better indicator of whether the given image is a face. Given an arbitrarily chosen threshold value Θ, we can compare the minimum Euclidean distance of to the threshold, accepting the image if the distance is less than the threshold distance.

3 Code Structure and Functionality

Our code client consists of two Python scripts, *calculate.py* and *test.py*. Our *calculate.py* is written to take in the test data and calculate the value of the eigenfaces and characteristic vectors Ω_i of all of the data points. Since this is a computationally expensive process, we store these values in auxiliary files that can be read in by the *test.py* script. In this way, the training (calculation of eigenfaces and characteristic vectors) must only be done once. The *test.py* script takes in a new image and compares its characteristic vector to the characteristic vectors of the data points (as outlined in section 2.3). It will ultimately output whether or not, based on its judgement, the image can be classified as similar to the data set (i.e. is it a human face?).

The folder **Training_Data** contains 40 images of people that we used to train our model. The folder **Test_Images_Random** contains 10 random images and the folder **Test_Images_People** contains 10 images of people that we used to test our model. The folder **Trained_Data** contains the data that we compute in *calculate.py* and that is read in by *test.py*.

The codes are all attached in our zip-file repository, and can be consulted for further information.

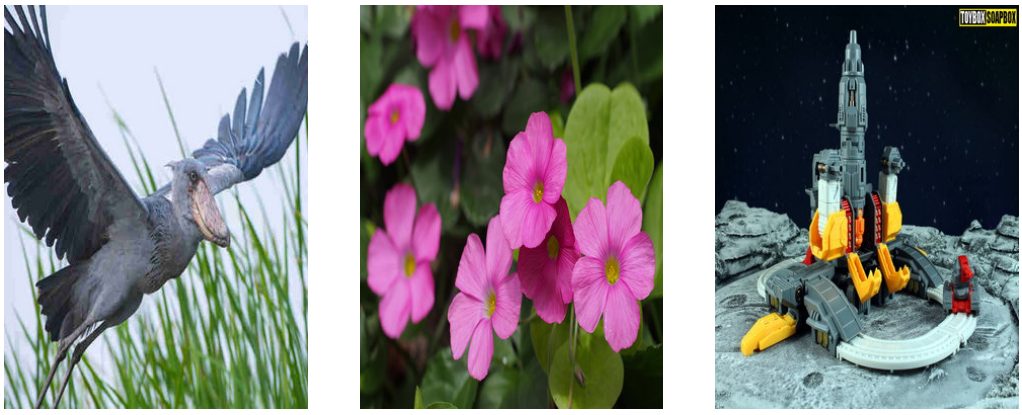

Figure 2: Example Test Data (sample of the random images used to test our model)

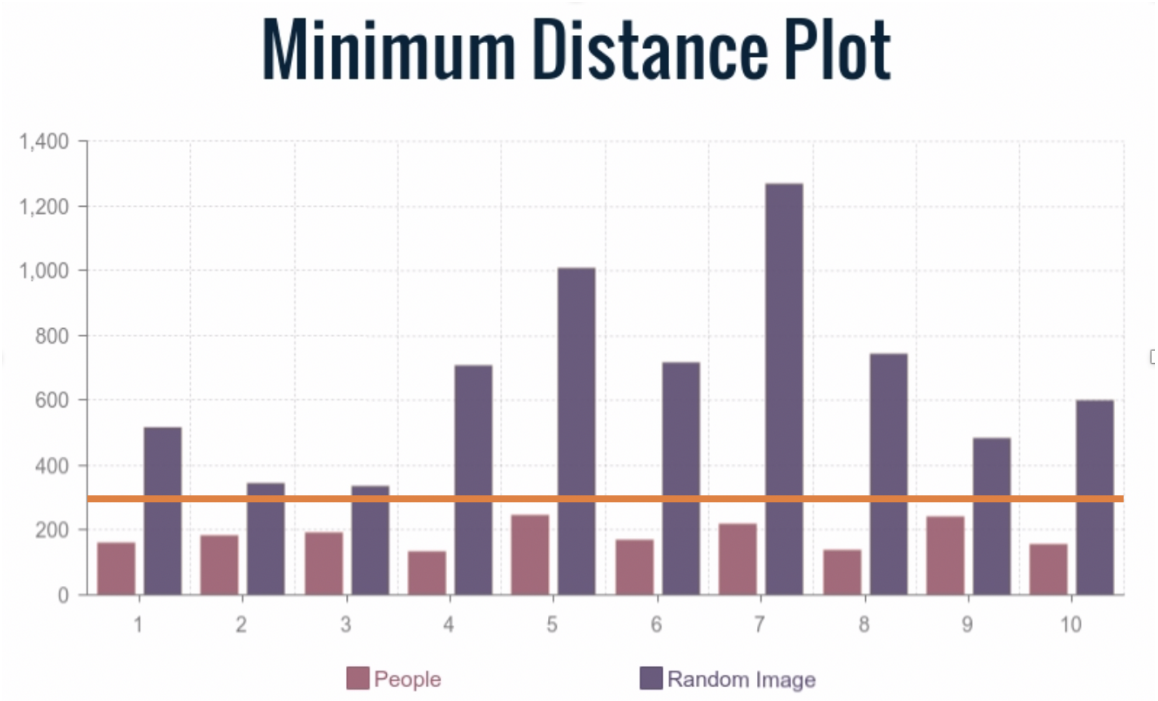

Figure 3: Results of the test set. The $y-$axis of this graph represents the minimum distance of the image from the eigenface space. The $x-$axis represents the index of the image used to test. We tested a total of 20 images, of which 10 were of people and 10 were of random images. The orange line represents the threshold $\Theta = 300$. It was found for our testing set that the distance of all human faces was below the threshold, while distances of non-faces was above the set threshold.

4 Results and Analysis

To ensure consistency in the images of the data set, we used Princeton's TigerBook app (which contains ID-quality pictures of Princeton students) as a source of randomly chosen photos. The portraits were each 300 pixels by 400 pixels, and a threshold distance $\Theta = 300$ was chosen (after some initial experimentation). Examples of the training images are shown in Figure 1.

To test the model, we input both sets of images that were also faces of Princeton students (also obtained from TigerBook) and images that were random objects (see Figure 2). The training sets and both testing sets have been included in our code repository for purpose of demonstration.

Using these parameters, there is a significant difference of distance scores between faces (mean distance score X) and non-faces (mean distance score Y), as can be observed in Figure 3. Though there is variation amongst the pictures, there does appear to be a distinctive gap between faces and non-faces, occurring at the chosen $\Theta = 300$.

5 Conclusions and Further Projects

A key further adjustment would be the collection of a better data set to train the model. Currently, the training data set requires a very specific format: a white background picture whose foreground has the subject's face and torso. Our facial recognition client currently only recognizes faces in images of this format. The data set could be expanded to account for different lighting, positions, and backgrounds. Another further adjustment could be changing the way that the distance from the data set is measured.[2] Specifically, it has been reported that the Mahalanobis distance may characterize the similarities of test images better than the currently used Euclidean distance.

6 References

[1] - Peregrud, Irina. "The Most Exciting Applications of Computer Vision across Industries." *InData Labs*, 13 February 2018, https://indatalabs.com/blog/data-science/applications-computer-vision-across-industries.

[2] - M. Turk and A. Pentland, "Eigenfaces for Recognition", Journal of Cognitive Neuroscience, vol. 3, no. 1, pp. 71–86, 1991.