ENGS211 - Final Project Report

Yeva Stepanyan, Mineh Nazarian, Ruzanna Karamyan

May 5, 2024

Abstract

Face recognition is a fundamental task in computer vision with numerous applications in security, surveillance, and human-computer interaction. Principal Component Analysis (PCA) is a popular technique used for dimensionality reduction and feature extraction in face recognition systems. In this project, we investigate the effectiveness of PCA for face recognition by implementing a complete face recognition pipeline. Our project involves preprocessing face images, projecting them onto a low-dimensional PCA subspace, and classifying them using nearest-neighbor classification. We evaluate the system's performance on a dataset of face images, analyzing factors such as recognition accuracy, computational efficiency, and robustness to variations in lighting, pose, and facial expressions. Our results demonstrate the usefulness of PCA-based face recognition systems highlighting their strengths and limitations.

1 Theory statement

Principal component analysis (PCA) is a dimensionality reduction method that is used to reduce the dimensionality of large data sets. So, we need to change a large set of variables into a smaller one. This has to be done in a way that it still contains most of the necessary information.The main idea of PCA is reducing the number of variables of a data set by ranking the principal components with the amount of variance captured, and selecting the top components allows for dimensionality reduction with minimal loss of information.

1.1 Geometric implication

Principal components represent the directions of the data that explain a maximal amount of variance, therefore, the lines that capture most information of the data. The relationship between variance and information here, is that the larger the variance carried by a line, the larger the dispersion of the data points along it, and the larger the dispersion along a line, the more information it has. To get a better idea we can just think of principal components as new axes that provide the best angle to see and evaluate the data, so that the differences between the observations are better visible

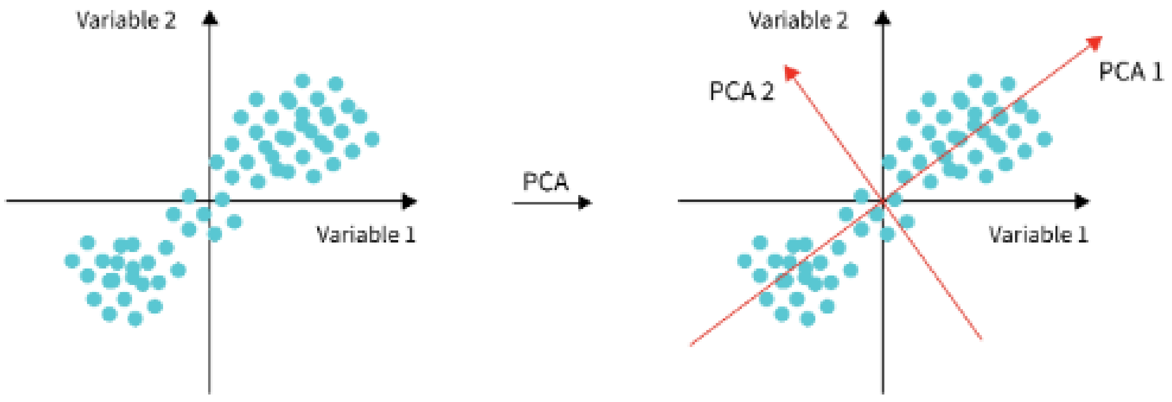


Figure 1: Finding principal components

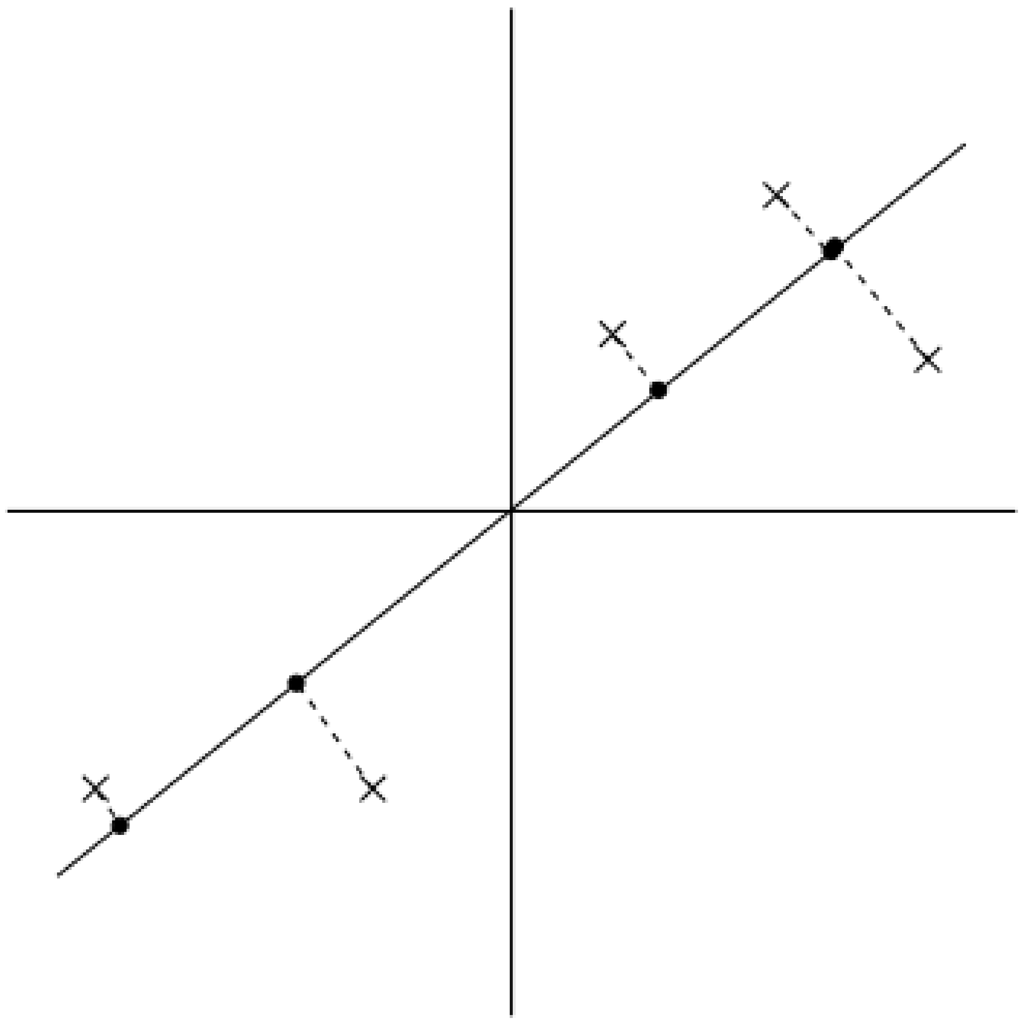


Figure 2: Projecting Data on PC1

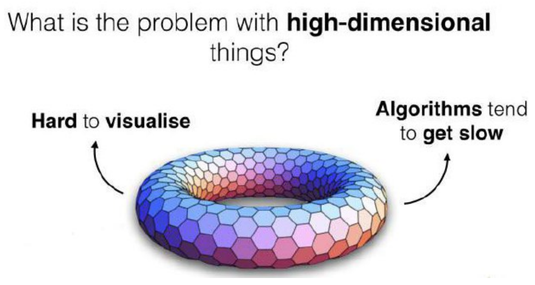

What is the problem with high-dimensional things?
Hard to visualise
Algorithms tend to get slow

Figure 3: Enter Caption

1.2 Motivation for PCA

Working directly with high-dimensional data, such as images, comes with some difficulties. High-dimensional data is often overcomplete, i.e., many dimensions are redundant and can be explained by a combination of other dimensions.Dimensions in high-dimensional data are often correlated so that the data possesses an intrinsic lower-dimensional structure. Dimensionality reduction exploits structure and correlation and allows us to work with a more compact representation of the data, ideally without losing much information. It can also be useful to detect potential patterns in high dimensional data, by visualizing the first 2-3 projections.

1.2.1 Problems with high dimensional things

1. Hard to visualise 2. Algorithms tend to get slow

2 Key aspects

2.1 Dimensionality Reduction

PCA helps manage high-dimensional datasets by extracting essential information and discarding less relevant features, simplifying analysis

2.2 Data Exploration and Visualization

This plays a significant role in data exploration and visualization, aiding in uncovering hidden patterns and insights.

2.3 Linear Transformation

PCA performs a linear transformation of data, seeking directions of maximum variance.

2.4 Feature Selection

Principal components are ranked by the variance they explain, allowing for effective feature selection.

2.5 Data Compression

PCA can compress data while preserving most of the original information.

2.6 Clustering and Classification

It finds applications in clustering and classification tasks by reducing noise and highlighting underlying structure.

2.7 Advantages

PCA offers linearity, computational efficiency, and scalability for large datasets.

2.8 Limitations

It assumes data normality and linearity and may lead to information loss

2.9 Matrix Requirements

PCA works with symmetric correlation or covariance matrices and requires numeric, standardized data.

2.10 Eigenvalues and Eigenvectors

Eigenvalues represent variance magnitude, and eigenvectors indicate variance direction.

2.11 Number of Components

The number of principal components chosen determines the number of eigenvectors computed.

3 Mathematical Explanation

Let's consider that we have a training set of M images each with size $N \times N$ pixels (same size).

Converting each image matrix in our training set to $N^2 \times 1$ vector:

This means reshaping each image matrix into a column vector, resulting in a face vector space with M face vectors.

Normalizing the face vectors:

We aim to remove common features shared by all training images. This includes

1. Calculating the average face vector:
$$\text{Average face vector} = \frac{1}{M}\sum_{i=1}^{M}\text{Face}_i$$
2. Computing the deviation vectors for each image:
$$\Phi_i = \Gamma_i - \psi \qquad\qquad i = 1,2,\ldots M$$
3. Forming the difference matrix A, preserving only distinguishing features:
$$A = [\ \Phi_1,\ \Phi_2,\ \ldots\ldots\ \Phi_M\ ]$$

Calculating the covariance matrix C of the training image vectors:

The covariance matrix captures the relationships between different features of the face vectors.

$$C = AA^T$$

The dimension of matrix C is

$$\begin{aligned}\text{Dimension of } (AA^T) &= (\text{Dimension of } A) \times (\text{Dimension of } A^T)\\ &= (N^2 \times M) \times (M \times N^2)\\ &= N^2 \times N^2\end{aligned}$$

For example, if N = 50, M = 100, then matrix C will be 2500 x 2500 which is a very large dimension. It will be very difficult, time and storage consuming to calculate N2 (2500 in this case) eigenvectors. That is why we need to reduce the dimension of our covariance matrix.

We can do it by calculating the covariance matrix by this formula instead:

$$C = A^T A$$

In this case the dimension of matrix C is

$$\begin{aligned}\text{Dimension of } (A^T A) &= (\text{Dimension of } A^T) \times (\text{Dimension of } A)\\ &= (M \times N^2) \times (N^2 \times M)\\ &= M \times M\end{aligned}$$

In case of our example the dimension of C will be 100 x 100. Calculating M (100 in this case) eigenvectors of size M x M (100 x 100) is easier than calculating N2 (2500) eigenvectors of size N2x N2.
$Facevectorspace \rightarrow Lowerdimensionalsubspace$

Selecting k best eigenfaces:

The eigenfaces represent the principal components of the face space, capturing the most significant variations in the data. We choose them such that k ¡ M and K can represent the whole training set.

Converting k lower-dimensional eigenvectors to the original dimensionality of the face vector space:

This step involves transforming the eigenvectors back to the original space to interpret their meaning in terms of facial features.

$$U_i = AV_i'$$

Representing each face image as a linear combination of all k eigenvectors:
Each picture of our training data set will be a weighted sum of the m eigenfaces + the average face vector. Our weight matrix is the following:

$$\Omega_i = \begin{bmatrix} w_1 & w_2 & \dots & w_k \end{bmatrix}^T$$

which shows how many percent of each eigenvector is our image made of. It is a key vector for our face images.

Comparing the testing image with the training dataset images and find the best match:

This is done by calculating the Euclidean distance between the weight vectors of the testing image and each training image.

$$\text{Average class projection } \Omega_\Psi = \frac{1}{X_i} \sum_{i=1}^{X_i} \Omega_i$$

The formula for the Euclidean distance is the following:

$$\delta_i = \|\Omega - \Omega_{\Psi i}\| = \sum_{k=1}^{M} (\Omega_k - \Omega_{\Psi iK})$$

If the distance is below a threshold, the testing image is considered a match with the closest training image. Otherwise we can conclude that the input face image is unknown.

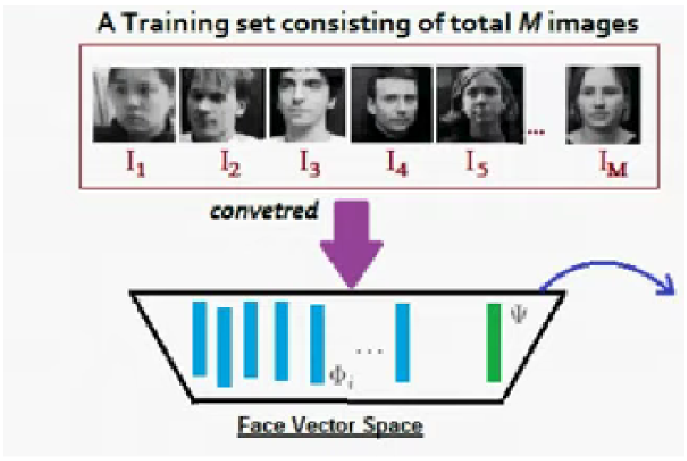
4 The Algorithm

4.1 Data Collection and Preprocessing

First, we collect datasets of face images to train and test our code with. Then, we make sure that the images are the same size and convert them to grayscale for a better and easier comparison process.

4.2 Feature Extraction

Then, since each image represents a matrix, we flatten them into one-dimensional vectors and construct a matrix that includes all such vectors.



4.3 Constructing Covariance Matrix

The next step is normalization, which will be done by calculating the average face (mean of all column vectors) and subtracting it from our matrix of faces. With our normalized matrix, we can now construct the covariance matrix, which we will use to extract eigenvalues and eigenvectors.

Let **X** be the normalized matrix of size $m \times n$, where m is the number of samples and n is the number of features.We

1. Calculate the mean vector

2. Subtract the mean vector from each row of $\mathbf{X}$ to obtain the centered matrix
3. Calculate the covariance matrix

4.4 Eigenvectors

After that, we

1. Compute the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ and eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ of $\mathbf{C}$:

$$\mathbf{C}\mathbf{v}_i = \lambda_i \mathbf{v}_i, \quad i = 1, 2, \ldots, n$$

2. Sort the eigenvalues in descending order along with their corresponding eigenvectors.
3. Choose the k most significant eigenvectors corresponding to the k largest eigenvalues to form the matrix $\mathbf{V}$.

4.5 Projected Data and Weights

Let **eigvectors_sort** be the sorted eigenvectors obtained from the previous step. We

1. Take the first k eigenvectors from and transpose them to obtain **reduced_data**:

$$\textbf{reduced\_data} = (\textbf{eigvectors\_sort})^T$$

2. Compute the projected data **projected** by multiplying the transposed standardized data $\textbf{X\_normalized}^T$ with **reduced_data** and then transposing the result:

$$\textbf{projected} = ((\textbf{X\_normalized})^T \cdot \textbf{reduced\_data})^T$$

3. Calculate the **weights** by multiplying the standardized data **X_normalized** with the transposed **projected**:

$$\textbf{weights} = \textbf{X\_normalized} \cdot (\textbf{projected})^T$$

4.6 Face Recognition

Finally, we use our testing images and try to recognize who they match in our training images. Let testing_images be a list of images to be recognized, proj_data be the projected data obtained from PCA, w be the weights calculated during PCA training, training_images be a list of training images, and threshold be the maximum distance threshold for a match. We

1. Initialize count to 0 to track the number of correct matches.

2. For each image img in testing_images:
    (a) Preprocess img by converting it to grayscale and flattening it into a feature vector (testing_face.)
    (b) Normalize testing_face by subtracting the average face average_face.
    (c) Project testing_face onto the PCA subspace using proj_data to obtain testing_weights.
    (d) Calculate the Euclidean distances between testing_weights and all training faces' weights w.
    (e) Find the index of the training face with the smallest distance as closest_face_index.
    (f) Retrieve the distance to the closest face as min_distance.
    (g) If min_distance is less than or equal to threshold, we check if the image corresponds to the correct person. and if correct, we increment count by 1.
    (h) Otherwise, set correctness to False.
    (i) Set recognized_face to the name of the recognized training image.
    (j) Display the input and recognized faces for visual inspection.
3. Return count i.e. the number of correct matches.

5 Conclusion

In this research, Principal Component Analysis (PCA) was the primary approach to developing a face recognition system. The algorithm showed encouraging results in correctly recognizing faces from a dataset of photos taken in various lighting, position, and occlusion scenarios. Our research revealed that PCA successfully preserved important information while reducing the dimensionality of the face picture data. This dimensionality reduction made computations faster and increased recognition process efficiency.

Additionally, by using PCA, we could recognize and extract the essential elements from the facial photos, which improved the system's ability to operate reliably and generalize to new faces. Even though the accuracy rates of our face recognition technology are satisfactory, there is still room for improvement. To improve the system's performance, for example, investigating deep learning-based methods and other sophisticated machine learning techniques may be beneficial, especially when handling larger datasets and more challenging scenarios.

Overall, this effort advances our knowledge of and ability to use PCA in face recognition, showcasing how useful it is for creating dependable and successful recognition systems. Our project offers a strong basis for future research and development efforts aimed at enhancing the capabilities of face recognition systems as the technology in this area continues to progress.

6 References

1 How PCA Recognizes Faces - Algorithm In Simple Steps
`https://www.youtube.com/watch?v=SaEmG4wcFfg&ab_channel=MahvishNasir`

2 Principal Component Analysis (PCA) with Scikit-Learn
`https://www.kdnuggets.com/2023/05/principal-component-analysis-pca-sc`
`html`

3 CS229 Lecture Notes - Andrew Ng and Tengyu Ma
`https://cs229.stanford.edu/main_notes.pdf`

4 A Step-by-Step Explanation of Principal Component Analysis (PCA)
`https://builtin.com/data-science/step-step-explanation-principal-comp`

5 A Step-by-Step Explanation of Principal Component Analysis (PCA)
`https://builtin.com/data-science/step-step-explanation-principal-comp`

6 What is principal component analysis (PCA)?
`https://www.ibm.com/topics/principal-component-analysis`