Faculty of Engineering
Cairo University

Computer Vision

Face Recognition

Team: 8

By:

Ahmed Mohamed,

Ahmed Mahmoud,

Eman Emad,

Rawan Shoap

Algorithms

1. Compute the mean face

This algorithm is typically the first step in Eigenfaces or PCA-based face recognition.

Inputs:

- training tensor: A list or array of 2D arrays, where each element is a flattened grayscale image.
- height, width: the dimensions of the original face images. Each image has been reshaped into a 1D array of size height * width.
- train_image_names: a list that contains the names of the training images (used here only for its length to compute the average).

Mathematical Notation:

Let:

- N be the number of training images (len(train_image_names)).
- xi be the flattened image vector of the i-th image.

Then the mean face μ is computed as:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Implementation Steps:

Initialize the accumulator. This is a row vector of zeros, used to accumulate the sum of all training images. Accumulate pixel values from each training image: For each image add it to mean face. Divide by number of images to get the mean. The mean vector is reshaped back into a 2D image and displayed in grayscale. Turns off all axes and tick labels to focus on the image content.

Output

- Visual output: The algorithm shows the mean face image, which is the average of all training face images.

- Data output: The variable mean face holds the 1D array of average pixel intensities. It can be reused in further processing like subtracting from original faces in PCA-based face recognition.

2. Normalized faces

This operation centers each face around the origin in the pixel space. It ensures that the first principal component captures the direction of maximum variance, not simply the mean offset. The normalized vectors are now suitable for computing the covariance matrix, eigenvectors, and eventually projecting any new faces into the PCA subspace.

Inputs

- training_tensor: A list or array of flattened grayscale images, each of shape (1, height*width), as before.
- mean_face: A 1D array of shape (height*width,), computed earlier as the average of all training images.
- train_image_names: A list of identifiers for each training image; used here to define how many images exist.
- height, width: Dimensions of each face image.

Mathematical Notation

Let:

- xi be the flattened vector of the iii-th training image.
- μ be the mean face vector.

Then the normalized face is:

$$\bar{x}_i = x_i - \mu$$

Implementation Steps

Initialize an array to store normalized images. This creates an empty 2D array to store all normalized face vectors. Normalize each image: This subtracts the mean face from each image, storing the result in normalized tensor. Visualize normalized faces to reshape each normalized vector back into a 2D image and displays it using matplotlib. The subplot layout is 2x4, which means this is built for visualizing 8 normalized faces. Axes and ticks are removed for clarity.

Outputs

- Visual output: A subplot displaying 8 normalized faces (mean-subtracted). These faces may look darker or have more contrast since pixel values may now be negative or centered around zero.
- Data output: The normalised_training_tensor holds all zero-mean face vectors, ready for further analysis.

3. Covariance Matrix

A key step in Principal Component Analysis. computing the covariance matrix of the normalized face data and finding its eigenvectors and eigenvalues.

Inputs

- normalised_training_tensor: A 2D NumPy array of shape (N, h*w).

Mathematical Notation:

Let:

- X be the data matrix of normalized face vectors (each row is a sample).
- Then the covariance matrix C is:

$$C = \frac{1}{N} XX^T$$

Then we perform:

- Eigen decomposition:

$$Cv_\iota = \lambda_i v_i$$

where:

  - λ_i : eigenvalues (variance captured along eigenvector v_i).
  - v_i: eigenvectors.

Implementation Steps

Compute the covariance matrix. Scale it down by dividing each element by 8, assuming we want the average covariance. Compute eigenvalues and eigenvectors. Create eigenvalue–eigenvector pairs. Sort the pairs by eigenvalue (descending order). Separate sorted eigenvalues and eigenvectors:

Outputs

- Covariance matrix (cov_matrix): Shows the linear relationships between face samples in the mean-centered space.
- Eigenvalues (eigenvalues): Each represents the amount of variance explained by its corresponding eigenvector.
- Eigenvectors (eigenvectors): These are in the image-sample space. They will be projected later into the image space to obtain Eigenfaces.
- eigvalues_sort: A list of eigenvalues sorted in descending order from the most significant to least significant components.
- eigvectors_sort: A list of eigenvectors reordered accordingly.

4. Find cumulative variance of each principal component

now implementing key PCA steps: evaluating how much variance is explained by each principal component, and then projecting the original data into a reduced PCA space (Eigenfaces).

Inputs

- eigvalues_sort: List of eigenvalues sorted in descending order. Each represents variance explained by a principal component.
- eigvectors_sort: Corresponding sorted list of eigenvectors.
- training_tensor: Original training images in flattened vector form, shape: (N, D) or (N, h × w).

Mathematical Notation

Let:

- λ_i be the i-th eigenvalue.
- V_k be top k eigenvectors.
- X: original image data (as column vectors).
- Z: projected data in eigenface space.

Then Cumulative Proportion of Variance Explained:

$$var\_comp\_sum_i = \frac{\sum_{j=1}^{i} \lambda_j}{\sum_{j=1}^{N} \lambda_j}$$

Implementation Steps

Compute cumulative variance. Plot cumulative variance vs. number of components shows a scree plot to determine how many principal components are enough. Select top-k eigenvectors to form reduced subspace. Take the top 7 eigenvectors. To project into a 7D eigenface space. Find weights for each traning image

Outputs

- var_comp_sum: Cumulative variance explained by components.
- Plot: Scree plot showing variance explained per number of components.
- proj_data: Projected training faces in the reduced 7D Eigenface space.

5. Face Recognition using PCA (Eigenfaces)

final step of a face recognition system using PCA (Eigenfaces) evaluating whether a test image matches any trained face by comparing projections into the PCA subspace.

Inputs

- img: Name of the test image file (string).
- train_image_names: List of training image filenames.
- proj_data: PCA projection matrix (shape (k, D)), where each row is an Eigenface.
- w: Matrix of projected training faces into PCA space, shape (N, k).
- mean_face: Mean face vector (shape (D,)), shared globally.

Mathematical Notation

Let:

- x_u: test face image (flattened and normalized).
- μ: mean face.

- W: PCA projection matrix (Eigenfaces).
- w_u: test face in Eigenface space.
- w_i: projected training image.
- Use Euclidean distance:

$$\text{distance } (w_u, w_i) = \|wu - wi\|2$$

- Find:

$$\arg\min_i \|w_u - w_i\|$$

Implementation Steps

Read and preprocess the test image. Normalize the test image by subtracting the mean face. Project into PCA (Eigenface) space. Compare against all training projections to Compute distances to all training samples. And find the index of the nearest match. Classification based on thresholds using two thresholds: t1: max allowable distance to even consider it a face at all. t0: tighter threshold if passed, it's an exact match. If distance is less than t1, it's likely a face. If distance less than t0, check if the predicted class matches the true label. If not a known class (or marked as 'apple'), consider it unknown. Else, it's classified as not a face. The algorithm uses plotting to display the input image and matched face. Uses color-coded titles: green for correct prediction. red for incorrect or unrecognized to Increments correct predictions if prediction is correct and track number of test images.

Output

- A visual grid of test images and their matching result.
- A final printed accuracy.