MSc Vibot & Computer Vision

Applied Mathematics

Homework 4

Désiré Sidibé
dro-desire.sidibe@u-bourgogne.fr

Face Recognition using PCA

NOTE
The goal of this homework is to help you familiarize yourselves with SVD and PCA through an intertesting computer vision application.

You can discuss the problems with each other, but your solutions must be independently written with your own words and formulations.

I. Face Recognition Problem

Face recognition (FR) is one of the most known/famous applications of Image Analysis and Computer Vision. You certainly have seen some examples in various movies (when investigators try to identify a person for instance). Ok, on TV they often achieve performances that are only possible in fiction! However, several methods have been developped that really work quite efficiently.
The basic idea of any FR system is to extract a set of interesting and discriminative features from the face images with the goal of reducing the number of variables. Let consider the 2D case where we have an input image I_q, and we wish to compare it with a set of database images $I_1, \ldots, I_p$ to find the best match. Each pixel can be considered a variable, so we have a very high dimensional space (every image is a point in a space of dimension $d = MN$, M and N being the image size). Using PCA can simplify the recognition problem by reducing dimensionality.

II. Normalization

Once a face has been detected in an image, a normalization step is required prior to recognition to account for scale, orientation, and location variations. To address this issue, we will use an iterative scheme based on affine transformations.
The main idea is using a transformation to map certain facial features from a face image to predetermined locations in a fixed size window. In this assignment, we will use the following facial features: (1) left eye center, (2) right eye center, (3) tip of nose, (4) left mouth corner, and (5) right mouth corner. Normally, the locations of these features should be extracted automatically. However, we will set them manually here in a 64×64 window, as shown in Figure 1.
Once we have extracted the locations of the facial features from all images, we need to compute the parameters of an affine transformation that maps them to the predetermined locations in the 64×64 window. Here we describe an iterative algorithm that work well in practice:

1. Use a vector $\overline{F}$ to store the average locations of each facial feature over all face images; initialize $\overline{F}$ with the feature locations in the first image F_1.

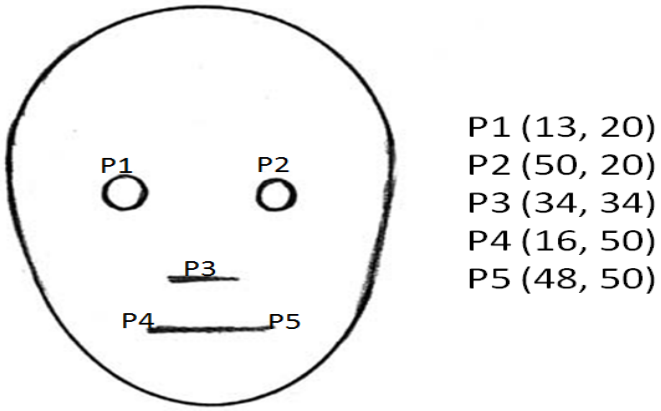

Figure 1: The five facial features used.

2. Compute the best transformation that aligns $\overline{F}$ (i.e., average locations of the features) with predetermined positions in the 64×64 window.

   The affine transformation can be defined by six parameters $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ and $b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$.

   A feature $f_i = \begin{bmatrix} x \\ y \end{bmatrix}$ is mapped to the corresponding predetermined location f_i^P by the equation

   $$f_i^P = Af_i + b$$

   Since we have 10 equations (two for each feature) and 6 unknowns, we have an over-determined system which can be solve using SVD.

   Once the transformation is obtained, we apply it on $\overline{F}$. Noting the aligned features $\overline{F}'$, we update $\overline{F}$ by setting $\overline{F} = \overline{F}'$.

3. For every face image F_i, compute the best transformation that aligns the facial features of F_i with the average facial feature $\overline{F}$; called the aligned features F_i'.

4. Update $\overline{F}$ by averaging the aligned feature locations F_i' for each face image F_i.

5. If the error between $\overline{F}_t$ and $\overline{F}_{t-1}$ is less than a threshold, then stop; otherwise, go to step 2.

The above algorithm typically converges within seven iterations. For each face image, it yields an affine transformation that maps the face to the 64×64 window.

NOTE:
In order to avoid gaps when computing the normalized images, we have to iterate through every pixel in the output image, find the corresponding point in the input image, and copy it over in the output image.

III. Recognition

An image I_i is treated as a vector X_i in the d-dimensional space ($d = MN$). This vector is obtained by concatenating the rows of the image I_i:

$$X_i = [I_i(1,1), I_i(1,2), \ldots, I_i(1,N), \ldots, I_i(M,1), I_i(M,2), \ldots, I_i(M,N)],$$

where $I_i(x, y)$ is the pixel intensity at position (x, y) in I_i.
We can see that a small size image, say 64×64, will results in a vector of dimension 4096. Clearly, most of the pixels will be highly correlated, so we can reduce this high dimension to a small number of variables.

Computing eigenfaces from a training set
If we have p training images, we can write our entire training data set as a $p \times d$ matrix D, where each row of D corresponds to one training image.

$$D = \begin{bmatrix} I_1(1,1) & I_1(1,2) & \dots & I_1(1,N) & \dots & I_1(M,1) & I_1(M,2) & \dots & I_1(M,N) \\ I_2(1,1) & I_2(1,2) & \dots & I_2(1,N) & \dots & I_2(M,1) & I_2(M,2) & \dots & I_2(M,N) \\ & & \vdots & & & & \vdots & & \\ I_p(1,1) & I_p(1,2) & \dots & I_p(1,N) & \dots & I_p(M,1) & I_p(M,2) & \dots & I_p(M,N) \end{bmatrix}_{p \times d}$$

To perform PCA, we have to compute the covariance matrix of D which is given by

$$\Sigma = \frac{1}{p-1} D^T D.$$

Note that Σ is a $d \times d$ matrix since we have d variables.
We then compute the eigenvalues and eigenvectors of Σ and keep the k eigenvectors corresponding to the k largest eigenvalues. Typically, we take $k \ll d$, then the dimensionality of the problem is reduced (for face recognition, $k < 100$).
Note that each principal component is a vector of dimension $d = MN$. So, we can convert principal components to images by reversing the concatenation operation. The converted images are called *eigenfaces*, since they are similar to human faces.

The k principal component are put into a $d \times k$ projection matrix Φ (columns of Φ are the principal components). Thus, any image I_i represented as a vector X_i in the d-dimensional space, can be projected into the PCA space by computing

$$\phi_i = X_i.\Phi$$

This new k-dimensional vector ϕ_i is the feature vector that represents the initial image I_i.
We can then store all our training images in the PCA space and easily search the database to find the closest match to a test image, comparing the feature vectors.

Few samples and many variables
In face recognition, we have a very large number of variables ($d = MN$) but a fewer number of images p. This means that the number of nonzeros eigenvalues of the covariance matrix is limited to p. Therefore, instead of computing the eigenvectors of the very large covariance matrix $\Sigma = \frac{1}{p-1} D^T D$ which is a $d \times d$ matrix, we can just compute the eigenvectors of $\Sigma' = \frac{1}{p-1} D D^T$ which is a $p \times p$ matrix.
Then putting the eigenvectors of Σ' as columns of a matrix Φ', it is easy to show that the eigenvectors of Σ, what we want to find, are given by $D^T \Phi'$.
This is a very useful trick, if we have to deal with high resolution images (just imagine computing the eigenvectors of a $10^6 \times 10^6$ matrix with Matlab!)

Recognizing a face
In the recognition phase, we are given an image I_j of a known person. Let α_j be the identity (label) of this person.
First, image I_j is represented by a vector X_j as in the training phase and this vector X_j is projected into the PCA space to get a corresponding feature vector $\phi_j = X_j \Phi$.

To identity I_j, we have to compute a similarity, for example Euclidean distance, between ϕ_j and all feature vectors ϕ_i's in the training set. The image I_i whose feature vector is most similar to ϕ_j is the output of the face recognition system. If $\alpha_i = \alpha_j$, then we have correctly identified the person. Otherwise, we have misclassified the person.

What You Have to Do?

In this assignment, you will use a small face dadaset to build a recognition system based on eigenfaces. The work can be divided in two main parts.

- **Part 1**

  You first have to apply the methods described in Section 1 to normalize all your face images. The recognition system will work with the normalized 64×64 images only.

  Then, divide the face images into two groups: a training set and a test set. For the training set use 3 different images for each subject: one frontal view and two side views. Put the other image(s) of each subject in the test set.

  You can create two directories `\train_images` and `\test_images`.

- **Part 2**

  Write a PCA based recognition system. A pseudo-algorithm is gigen bellow:

---

1. Set a PCA parameter N_{pca}, i.e. the number of principal components.
2. Read training images and form a training data matrix D.
3. Assign a label to each training image and form a labels matrix L_t. The label can just be the name of the person.
4. Perform PCA of D. (Don't forget to remove the mean prior to computing covariance matrix). This gives you a PCA transformation matrix Φ.
5. Compute feature vectors of all training images using Φ and store them in a matrix F_t.
6. Read test images, and for each test image I_q
   - Calculate the feature vector ϕ_q using the PCA transformation matrix Φ
   - Compute the distance between ϕ_q and all training feature vectors, which are stored in F_t
   - Find the best match I_z (minimun distance between feature vectors)
   - Check if the label of I_q is the same as the label of I_z. Otherwise, increment an error count ϵ by 1.
7. Output the correct recognition accuracy:

$$\text{accuracy} = (1 - \frac{\epsilon}{\text{total number of test images}}) * 100$$

You can also check if the correct face is among the first k faces returned by your algorithm. Show the accuracy for different values of k.

What you have to turn in?

A brief report including a print-out of your source code. In your report you should:

- Explain what you're doing (methodology)
- Give a description of the experiments
- Show and analyse the results (include figures, graphics, etc)
- A brief summary of what you have learned

You must also provide a Matlab script which, when given a test image I_j, find the corresponding best match image I_i in the training set, and show both images on the screen.

IMPORTANT

1. *PCA based face recognition has been around for years, so you can easily find some code on the Internet. However, the goal of this assignment is to make you understand how what we have learned can be used in practice. Therefore, doing it yourself is the best way for understanding how it works!*
2. *You have to write your own PCA code. The use of any Matlab built-in PCA function is strictly forbidden.*
3. *Contact me if you need any assistance.*
4. *Enjoy!*