Ozan, Ran
January 22, 2014

Applied Mathematics

Home work 4
Face Recognition using PCA

⌞ Chapter 1: Introduction ⌝

In this project, we implemented a face recognition system by using principal component analysis, which is known as *PCA*. *PCA* method provide a mathematical way to reduce the dimension of problem.

Since the most elements of a facial image are highly correlated, it is better to extract a set of interesting and discriminative feature of a facial image. Mathematically speaking, we transform the correlated data to independent data. To implement the transform, we employed some linear algebra method such as SVD. The main idea is to obtain *Eigenfaces* that every face can be regard as a linear combination of these eigenfaces. Then the face recognition problem convert to a mathematic problem: what is the linear combination of a face? In other words, it simplify a problem from 2D to 1D.

⌞ Chapter 2: Normalization ⌝

How to normalized images

Normalization actually is a projection process. Because we detected the face images with different angle, we hope that we can use a same coordinate system for different angle faces. This coordinate system is a 64×64 window which we predetermined five positions as facial features' location. These five locations is $\overline{F}$. Once we have $\overline{F}$, we can obtain affine transformation coefficients for each image as following figure shown.

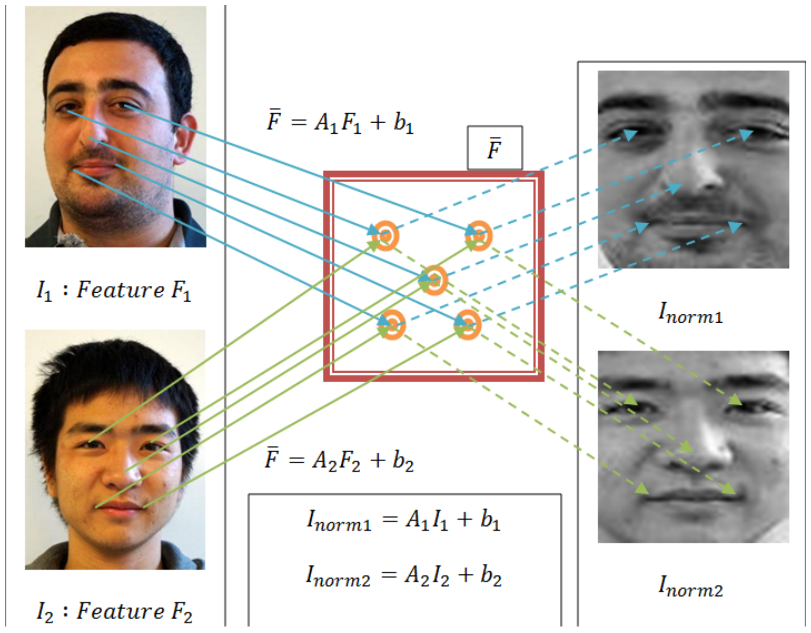

Figure 1:

In *Figure*1, A_i and b_i are affine transformation matrix. Where A and b are given by:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

How to compute A and b

For each feature, the affine transformation is given by:

$$f_i^P = Af_i + b$$

Where f_i is the feature location in original image and f_i^P is the position in normalization image. Since we already have f_i and f_i^P is also predetermined, let $f = \begin{bmatrix} x \\ y \end{bmatrix}$, then the transformation can be written into:

$$x_i^P = a_{11}x_i + a_{12}y_i + b_1$$
$$y_i^P = a_{21}x_i + a_{22}y_i + b_2$$

We apply it on five features, the transformation is given by:

$$\begin{bmatrix} x1_i & y1_i & 1 \\ x2_i & y2_i & 1 \\ x3_i & y3_i & 1 \\ x4_i & y4_i & 1 \\ x5_i & y5_i & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ b_1 & b_2 \end{bmatrix} = \begin{bmatrix} x1_i^P & y1_i^P \\ x2_i^P & y2_i^P \\ x3_i^P & y3_i^P \\ x4_i^P & y4_i^P \\ x5_i^P & y5_i^P \end{bmatrix}$$

Until now, the problem convert to solve $Ax = b$ in linear algebra. Since we got 10 equations for 6 unknowns, we employ SVD ($function\ pinv()\ in\ matlab$) to solve it.

How to find $\overline{F}$

It is very important to get an appropriate $\overline{F}$. We calculate $\overline{F}$ by following steps:

- Step 1: We initialize $\overline{F}$ with first image's feature to find average of all.
- Step 2: Finding transformation that project $\overline{F}$ to predefined position in 64x64 window. Then we can find this affine transformation as A and b matrices that has 6 parameters, and we apply this transformation to $\overline{F}$.
- Step 3: For each image feature F_i, we computing the A and b matrices for affine transformation that projecting $\overline{F}$ to F_i.
- Step 4: We change $\overline{F}$ to averaged projected feature locations that we calculated previous step.
- Step 5: If error between $\overline{F}_t$ - $\overline{F}_{t-1}$ is less than our threshold, we go back to step 2.

Chapter 3: Principal Component Analysis

We used PCA to transform our correlated data set into smaller uncorrelated data set. So PCA helped us on reducing the large dimensions of data set into smaller dimensions of it. After finding $\overline{F}$, we followed these steps:

We converted our 64 by 64 normalized images X_i into row vector sized 1x4096 and constructed matrix D_{pxd}(p = number of images, d = 64x64) that each row corresponds to X_i. Then we defined k between 50 and 100 to kth largest eigen values.

Before computing the PCA, we computed the covariance matrix of D as follows:

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

Then we used matlab's "svd()" function to get eigen values from Σ matrix. This process in our training is taking the most time. We put biggest k eigen values to Φ.
We calculated PCA space by;

$$\phi_i = X_i.\Phi$$

After all, now we can store all our training images in the PCA space and easily search them to find closest matches.

⌞ Chapter 4: Recognition ⌝

blablabla...

⌞ Chapter 5: Experimental Result and Analysis ⌝

blablabla...

⌞ Chapter 6: Summary ⌝

blablabla...

⌞ References ⌝