CS 580 Homework 7

Kevin Mao

April 30, 2015

Background

The task of reliable and robust facial recognition represents a number of challenging computational, engineering, and practical problems. *Eigenfaces for Recognition* by Matthew Turk and Alex Pentland proposes a novel solution to this problem by deriving common vectors between a set of training faces, named Eigenfaces. These Eigenfaces are used to recognize new faces by calculating the magnitude of distance between the new image's representative image vector and the image vectors of the Eigenfaces.

This assignment recreates Turk and Pentland's original facial recognition algorithm, trains it using a randomly selected number of facial images from a well known database, and evaluates recognition against a second set of randomly selected facial images.

Generating Eigenfaces

Let each image be a training set of M distinct N by N pixel resolution images. For the purposes of simplicity, we convert each of these images into greyscale intensity values. Accounting for color during Eigenface generation is an area of future enhancment. Each of these N by N images are represented as a one dimensional array of values of length N^2.

Let the set of one dimensional arrays representing the greyscale training face images be represented as (1).

$$\tau = \tau_1, \tau_2, ..., \tau_M \tag{1}$$

The average face image (2) can be defined as a one dimensional array of size N with the value at each index equal to the average of pixel value of the training faces at that index.

$$\psi = \frac{1}{M}\sum_{n=1}^{M}(\tau_n) \tag{2}$$

We can then calculate a set of nomralized pixel vectors (3) representing each training face image

$$\phi_i = \tau_i - \psi \tag{3}$$

The naive approach to generating Eigenfaces from these normalized vectors would be to generate the covariance matrix (4) and derive the resultant N^2 eigenvalue/eigenvector pairs.

$$C = \frac{1}{M}\sum_{n=1}^{M}(\phi_n\phi_n^T) \tag{4}$$

$$= AA^T \tag{5}$$

This approach, however, becomes intractable as the generated covariance matrix is of size N^2 by N^2. For example, given a set of 256 by 256 greyscale training face images, where each pixel intensity value is represented as a 4-byte integer, the covariance matrix alone would require 16 gigabytes of memory. Turk and Pentland propose a more scalable solution based on the assumption that if $M << N^2$, only $M - 1$ out of N eigenvectors will be meaningful. Consider the eigenvalues μ_i and eigenvectors v_i of A^TA:

$$A^TAt_i = \mu_i t_i \tag{6}$$

Premultiplying both sides of the equation yields (6), which is equivalent to the the eigenvalue/eigenvector pairs of (4), where the eigenvector is equal to Av_i.

$$AA^TAv_i = \mu_i Av_i \tag{7}$$

The set of eigenvalue/eigenvector pairs (μ_i, Av_i) has a size of N^2. However only $(M-1)$ pairs have a nonzero eigenvalue. From these $(M-1)$ we select the M' pairs containing the largest eigenvalues, where $M' < (M-1)$ is some number of eigenfaces to use for identification/recognition. These form the set of eigenfaces $u_1, u_2, ..., u'_M$.

Classifying a Face Image

Recognizing a Face Image

Any input image, τ, can be transformed into a set of weights representing the contribution of each Eigenface image towards forming the original input image.

$$\omega_k = u_k^T(\tau - \psi) \tag{8}$$

The weights form the vector ω^T that represents the contribution of each Eigenface image towards forming the original input image.

$$\Omega^T = \{\omega_1, \omega_2, ..., \omega_{M'}\} \tag{9}$$

For each training face image, we derive its corresponding vector of Eigenface weights, Ω_k, to be used for recognition. Turk and Pentland refer to this as a "face class".

$$\Omega_k = \{\omega_{k,1}, \omega_{k,2}, ..., \omega_{k,M'}\} \tag{10}$$

$$\omega_{k,i} = u_i^T(\tau - \Psi) \tag{11}$$

Given the input image vector τ, we map it to the face class that provides the best description of the input image by calculating the Euclidean distance between its weight vector Ω and each of the face classes and select the face class yielding the minimum distance.

$$\epsilon_k^2 = ||(\Omega - \Omega_k)||^2 \tag{12}$$

Given this minimum distance and some preconfigured threshold Θ_k, we can determine whether the input image maps closely to one of the faces in our training set.

Even though we have identified which face in the training set most closely maps to the input image, many images will map to the training faces to some degree under Θ_k. The second part of classification, identification of faces, is described in the next section.

Identifying Faces

In order to ensure that the input image that we are classifying is an actual face (as opposed to Bart Simpson), Turk and Pentland propose a comparison

between the original input image and a reconstructed projection image built from the vector of Eigenfaces u_k and their corresponding weights, ω_k. From this, the squared distance between the mean-adjusted input image and the reconstructed projection can determines how closely the input maps to our set of training faces (13).

$$\epsilon^2 = ||\Psi - \Psi_f||^2 \tag{13}$$

For some preconfigured threshold Θ, we can determine whether the input image projects onto "face space", allowing us to determine whether the image we are analyzing is actually a face. If the distance ϵ is above Θ, we should assume that the input image is not a face.

Implementation

The facial recognition engine is implemented in Scala 2.10.4 running on Java 1.7.0_79. Image processing logic uses common code pulled from the open source project Tinderbox (https://github.com/crockpotveggies/tinderbox). Eigenface generation and facial identification matrix computation uses the Apache Commons Math library.

Testing

Testing for both verification and identification

Results

Overall error for each phase and how this changes with changing certain parameters of the algorithm