RECOGNITION AND AGE PREDICTION WITH DIGITAL IMAGES OF MISSING CHILDREN

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Wallun Chan

December 2005

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Chris Pollett

Dr. Mark Stamp

Dr. Robert Chun

Abstract

Principal Components Analysis is a dimensionality reduction technique that determines eigenvectors (principal components) and corresponding eigenvalues from the covariance matrix of a data set. The eigenvectors that do not contribute much to scatter are truncated without excessive data loss. The remaining eigenvectors represent a new coordinate system in lower dimensional space, allowing a more compact and efficient representation of the original data. In this project, we age-progress digital face images of children by applying PCA to training images that maps young to aged faces. Once we compute the principal components from the training images, we project an input image onto the principal components to obtain a weight vector. The coefficients of the weight vector represent proportions of each corresponding principal component needed to approximately reconstruct the original input image by a weighted summation. We want the weight vector to be in close proximity to a cluster of projected training images. As a result, the reconstructed image should capture the aged features by resembling a weighted average result. In this report, we explain the mathematical derivation of PCA, and the eigenface approach that applies PCA to image data. We then cover the design and development of our age progression program that applies PCA to grayscale images of whole faces. Next, we discuss the extension of the program to support color images, and a feature-based approach to age progression. Finally, we show results of tests performed to evaluate the performance of the age progression program using various training image sets of young and aged face images.

Table of Contents

List of Figures and Tables

Chapter 1

Introduction

In this project, we use the eigenface technique as described in the classic paper by Turk and Pentland [11] to produce age-progressed face images of missing children. The eigenface approach is based on using PCA on image data. Principal Components Analysis (PCA or Karhunen-Loéve transformation) is a technique that has been used extensively for pattern recognition applications such as face detection and recognition. PCA determines a linear transformation of the coordinate system of multi-variate data such that the axes of the new coordinate system are in the directions of maximum point scatter. The data with respect to each new axis are then completely de-correlated. PCA has the advantage of allowing the reduction of high-dimensional data onto lower dimensional space by combining features in a least squares fashion. In other words, data points may be represented with fewer variables in this new coordinate system.

Mathematically, we wish to determine the eigenvectors of the covariance matrix of a sample data set that represents a new coordinate system in a lower dimensional space. The covariance matrix consists of covariances between all possible pairs of scalar elements in a vector, where covariance measures how closely two variables change in relation to each other. Therefore, the covariance matrix indicates the amount of correlation of data between axes. PCA minimizes

the amount of correlation between axes, and hence the amount of redundancy. If we assume a matrix that is real and symmetric, a set of orthonormal eigenvectors can always be found. The eigenvectors are then sorted by their respective eigenvalues. The eigenvectors with the largest eigenvectors are retained, such that the lesser eigenvectors that do not result in much scatter, may be truncated without sacrificing excessive accuracy of data reconstruction. Thus high-dimensionality data is reduced significantly in dimensions, allowing efficient algorithms that use only the required eigenvectors for the largest few corresponding eigenvalues. PCA has the property such that projecting data onto the eigenvectors results in a set of weights that represent linear proportions of the principal components. The weights can also be used to reconstruct a data point. PCA represents a sample data point with a proper linear combination of the eigenvectors (principal components) that minimizes the error between the reconstructed result and all sample data points. The goal of this project is to collect an adequately large training image set, compute the principal components, and reconstruct an input image by determining the proper linear combination of the principal components. In doing so, we attempt to capture the essence of aging for each reconstructed image by proportionally combining the variations across the set of training images represented by each principal component.

This project consists of several smaller projects that focus on research, demonstration, and evaluation of several key concepts and techniques used for

image reconstruction and age progression. First, we explore the feasibility of using PCA by implementing a program for image reconstruction and recognition of grayscale images. Second, we implement an image-retrieval and preprocessing tool to obtain and prepare a sufficient set of training and input images, and use the eigenface program to reconstruct aged grayscale images of missing children. Third, we begin exploring feature-based age progression by looking at shape contexts to solve the key problem of locating features on a face image (e.g. eyes, nose, and mouth). Finally, we take use these concepts and techniques to implement a program that takes color face images as inputs and locates major features (e.g. eyes, nose, mouth, and face) to be PCA trained. Prior to this, the images are preprocessed in terms of size, removal of in-plane rotation, and cropping to eliminate background and maximize face area. For each selected feature, each (RGB) color channel is extracted to produce three separate grayscale images that are concatenated to form an overall grayscale image. This is done for young and aged sets of images, results of which are also concatenated to form final training images for each feature. These training images provide a mapping between young and corresponding aged features. An input face image is preprocessed in a similar manner as mentioned above except that the selected features are concatenated with themselves. Each selected feature from the input image is projected onto a corresponding set of principal components to produce a set of weights, which are then used to reconstruct a predicted aged feature. The aged features are then blended back onto the aged faces at their respective locations. PCA is an unsupervised learning technique

that relies entirely on the training data, and therefore does not take advantage of specific target data. Selecting, removing, and training on specific features allows more control over how PCA performs image reconstruction. In addition, we rely on the preprocessing and normalization of training and input data to minimize any variants that may negatively affect the output results.

Chapter 2

Background and Theory

2.1 Principal Component Analysis (PCA)

In this section, we give a brief description and analysis of PCA. We provide more mathematical detail for PCA in the coming sections as it applies to images. Given *N* points in *d*-dimensional space, the eigenvectors of the covariance matrix for the data set form the new axes of the *d*-dimensional data points. Multi-variate data can be expressed more efficiently in terms of the top *d'* orthogonal eigenvectors. This is done by projecting the data points onto a vector passing through the mean of the data points in the direction of the eigenvectors with the largest eigenvalues. The eigenvectors are sorted such that the ones with the largest eigenvalues dominate in scatter in their respective directions.

Given a *d x d* covariance matrix, there are *d* eigenvectors that may be computed. PCA reduces this dimensionality by using a mapping process (Hotelling transform) that attempts to project vectors in *d*-dimensional space to vectors in *d'* dimensional space in a least squares fashion for $d' < d$. Specifically, instead of a *d x d* transformation matrix, we form a *d' x d'* transformation matrix from the top *d'* eigenvectors that performs the mapping process. However, since dimensionality reduction is a lossy process, reconstruction of each sample data point is no longer exact. To minimize the error between each reconstructed data point in *d'* dimensional space and original sample data point, we solve for the top

d' eigenvectors from the covariance matrix of the sample data set. In applications with high-dimensional data (e.g. digital image data), the first d' eigenvectors have significantly greater eigenvalues than the other *d* - *d'* eigenvectors.

PCA determines and uses the eigenvectors to represent patterns across all sample data points. Each eigenvector represents a principal component such that any sample data point may be reconstructed with a weighted linear combination of the principal components. Each principal component contributes more or less to each original data point. To reconstruct each training data point from the eigenvectors, the proper proportions or weights must be determined. This is done by projecting each sample data point onto the subset of principal components to form a weight vector.

2.2 PCA Theory

The following mathematical derivation of PCA is taken from Pattern Classification [3]. Given *N d*-dimensional sample data vectors $x_1, \ldots, x_n$, we want to find a *d*-dimensional vector x_0 that best represents $x_1, \ldots, x_n$ in a least squares sense. Specifically, $\overline{m} = \frac{1}{N}\sum_{i=1}^{N} x_i$ minimizes the error function

$E(x_0) = \sum_{i=1}^{N} \|x_0 - x_i\|^2$. However, the mean $\overline{m}$ does not show the variability of the data set. To do so, we project the data set onto a *d'*-dimensional representation, where *d'* ≤ *d*. Starting with a simpler and more intuitive example, we have e as a

unit vector passing through $\overline{m}$, and project the data onto a line given by $x = \overline{m} + ce$. Given $c = c_1, \ldots, c_n$, we redefine the error function to be

$$E(x) \rightarrow E(c,e) = \sum_{i=1}^{N} \left\| (\overline{m} + c_i e) - x_i \right\|^2 = \sum_{i=1}^{N} \left\| c_i e - (x_i - \overline{m}) \right\|^2$$

$$\sum_{i=1}^{N} {c_i}^2 \left\| e \right\|^2 - 2\sum_{i=1}^{N} c_i e^T (x_i - \overline{m}) + \sum_{i=1}^{N} \left\| x_i - \overline{m} \right\|^2 = \tag{1}$$

$$\sum_{i=1}^{N} {c_i}^2 - 2\sum_{i=1}^{N} c_i e^T (x_i - \overline{m}) + \sum_{i=1}^{N} \left\| x_i - \overline{m} \right\|^2, \; where \; \left\| e \right\| = 1$$

Solving for the partial derivative of the error function, we have

$$\frac{\partial E}{\partial c_i} = 2c_i - 2e^T x_i + 2e^T \overline{m} = 0$$

and obtain

$$c_i = e^T (x_i - \overline{m}) \tag{2}$$

Substituting equation (2) into (1), and noting that the covariance matrix is given by $\frac{1}{N-1} \sum_{i=1}^{N} (x_i - \overline{m})(x_i - \overline{m})^T$, we obtain

$$E(c,e) = \sum_{i=1}^{N} {c_i}^2 - 2\sum_{i=1}^{N} {c_i}^2 + \sum_{i=1}^{N} \left\| x_i - \overline{m} \right\|^2 = -\sum_{i=1}^{N} \left[e^T (x_i - \overline{m}) \right]^2 + \sum_{i=1}^{N} \left\| x_i - \overline{m} \right\|^2 =$$

$$-\sum_{i=1}^{N} e^T (x_i - \overline{m})(x_i - \overline{m})^T e + \sum_{i=1}^{N} \left\| x_i - \overline{m} \right\|^2 = -e^T S e + \sum_{i=1}^{N} \left\| x_i - \overline{m} \right\|^2$$

where the scatter matrix S is the covariance matrix times *N - 1*.

By maximizing $e^T S e$ the error function $E(c,e)$ is minimized. We use the Lagrange multiplier method given by $L(x,\lambda) = f(x) + \lambda \cdot g(x)$ subject to $g(x) = 0$ and derivative $\frac{\partial L}{\partial x} = \frac{\partial f}{\partial x} + \lambda \frac{\partial g}{\partial x} = 0$. We have $L(e,\lambda) = f(e) + \lambda\, g(e)$ where

$f(e) = e^T Se$, and since $\|e\| = e^T e = 1$, the constraint function becomes $g(e) = e^T e - 1 = 0$. We compute the partial derivative of $L(e, \lambda)$ yielding

$$\frac{\partial L(e,\lambda)}{\partial e} = \frac{\partial f(e)}{\partial e} + \lambda \frac{\partial (g(e))}{\partial e} = 2Se - 2\lambda e = 0 \rightarrow Se = \lambda e \qquad (3)$$

Equations (2) and (3) indicate that the eigenvector e with the largest eigenvalue λ of the scatter matrix points in the direction such that the projection of the data onto $x = \overline{m} + ce$ is maximized. Figure 1 and $E(x) = E(c, e) = \sum_{i=1}^{N} \|c_i e - (x_i - \overline{m})\|^2$ from equation (1) indicate that the subtraction of $\overline{m}$ centers the entire point distribution about $\overline{m}$, and c_i extends e such that $c_i e$ approaches $x_i - \overline{m}$. We

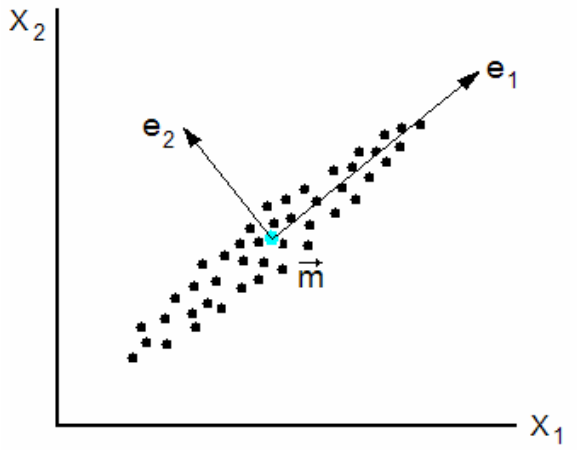


Figure 1: Principal components of 2-D data points

also note that e_1 in Figure 1 is clearly the dominant principal component. Therefore, the projection of data onto e_1 is well approximated without excessive loss of accuracy. We can extend $x = \overline{m} + ce$ and equation (1) in one dimension to obtain a least squares projection in *d'* dimensions with the following error function in which c_{ij} are coefficients that represent linear proportions of e_j

$$E_{D'}(e) = \sum_{i=1}^{N} \left\| \overline{m} + \sum_{j=1}^{D'} c_{ij} e_j - x_i \right\|^2 \qquad (4)$$

(principal components) to approximate the original sample data point. In general, PCA provides a mapping between *N* data vectors $(x_1, \ldots, x_D)^N$ of *d* dimensions to $(x'_1, \ldots, x'_{D'})^N$ of *d'* dimensions where *d'* ≤ *d*. The top *d'* eigenvectors of the covariance matrix represent the orthogonal axes of the new coordinate system.

2.3 Solving for Eigenvalues and Eigenvectors

We first assume a symmetric real matrix S such that $S = S^T$ where $(s_{ij} = s_{ji})$. We use the Jacobi method with the following derivation from Numerical Recipes in C [8] that applies a sequence of planar rotational transformation T_{pq} to S until $S' = T_{pq}{}^T \cdot S \cdot T_{pq}$ is diagonal. Given the planar rotational matrix

$$T_{pq} = \begin{array}{c} \\ p \\ \\ q \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} & & p & & q & & \end{array} \\ \begin{bmatrix} 1 & & & & & & \\ & \cdots & & & & & \\ & & a & \cdots & b & & \\ & & \vdots & 1 & \vdots & & \\ & & -b & \cdots & a & & \\ & & & & & \cdots & \\ & & & & & & 1 \end{bmatrix} \end{array}$$

, subscripts *p* and *q* refer to the rows and columns of scalars $a = \cos\phi$ and $b = \sin\phi$. All diagonal elements are unity except for scalars a, and all off-diagonals are zero except for scalars b. Multiplying out the diagonalization of $S' = T_{pq}{}^T \cdot S \cdot T_{pq}$, the following equations are obtained:

$$s'_{rp} = a\,s_{rp} - b\,s_{rq} \qquad (r \neq p, r \neq q) \qquad (5a)$$
$$s'_{rq} = a\,s_{rq} + b\,s_{rp} \qquad (r \neq p, r \neq q) \qquad (5b)$$
$$s'_{pp} = a^2\,s_{pp} + b^2\,s_{qq} - 2\,a\,b\,s_{pq} \qquad (5c)$$
$$s'_{qq} = b^2\,s_{pp} + a^2\,s_{qq} + 2\,a\,b\,s_{pq} \qquad (5d)$$
$$s'_{pq} = s_{pq}\left(a^2 - b^2\right) + a\,b\left(s_{pp} - s_{qq}\right) \qquad (5e)$$

We see that only rows *p* and *q*, and columns *p* and *q* are changed for S'. The idea is to zero-out the non-diagonal elements of S' by applying the planar rotation matrix T_{pq} with θ iteratively. Since the subscript *pq* is off-diagonal, we set equation (5e) to zero and define $\theta \equiv \dfrac{a^2 - b^2}{2\,a\,b} = \dfrac{s_{qq} - s_{pp}}{2\,s_{pq}}$ and $t \equiv \dfrac{b}{a}$. We then have

$$\begin{aligned} &2\,\theta\,a^2\,t = a^2 - a^2\,t^2 \\ &a^2\,t^2 + 2\,\theta\,a^2\,t - a^2 = 0 \\ &t^2 + 2\,t\,\theta - 1 = 0 \end{aligned} \qquad (6)$$

Equation (6) may be solved with the quadratic formula for *t*. With equation (6) and setting $s'_{pq} = 0$, we substitute (5e) into (5c), separate s_{pp} from s_{qq}, and obtain

$$s'_{pp} = s_{pp} - t\,s_{pq} \qquad (7a)$$

Further substitutions with (5a – 5e) yield

$$s'_{qq} = s_{qq} + t\,s_{pq} \qquad (7b)$$
$$s'_{rp} = s_{rp} - b\left(s_{rq} + \tau\,s_{rp}\right) \qquad (7c)$$
$$s'_{rq} = s_{rq} - b\left(s_{rp} + \tau\,s_{rq}\right) \qquad (7d)$$

where $\tau = \dfrac{b}{1 + a} = \tan\dfrac{\phi}{2}$.

Therefore, for each off-diagonal element of S', the Jacobi method involves using equations (7a – 7d) iteratively to compute $S \rightarrow T_1^T \cdot S \cdot T_1 \rightarrow$

$T^T{}_2 \cdot T_1^T \cdot S \cdot T_1 \cdot T_2 \rightarrow T_N{}^T \cdot \ldots \cdot T_3^T \cdot T^T{}_2 \cdot T_1^T \cdot S \cdot T_1 \cdot T_2 \cdot T_3 \cdot \ldots \cdot T_N = S' = Z^T \cdot S \cdot Z$. Once S' becomes diagonal, the eigenvalues are located in the diagonals of S', and the eigenvectors are located in the columns of Z. Numerical Recipes in C [8] gives an algorithm that executes in $O(n^3)$.

2.4 PCA Algorithm and Eigenfaces

We begin with a digital image as an $N \times N$ array of 8-bit grayscale pixel values from 0 to 255. To prepare a set of *M* images for PCA analysis, we first stretch out each $N \times N$ image array so that it is $N^2 \times 1$, and concatenate *M* images to form image matrix $\Gamma = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_M\}$ of size $N^2 \times M$. We then calculate mean image $\Psi = \frac{1}{M}\sum_{i=1}^{M}\Gamma_i$ and obtain mean adjusted image matrix $A = \{\Phi_1, \Phi_2, \ldots, \Phi_M\}$ with $\Phi_i = \Gamma_i - \Psi$. The corresponding eigenvalues $\lambda_{i\ldots M}$ and eigenvectors $u_{i\ldots M}$ are determined from the covariance matrix $C = \frac{1}{M}\sum_{i=1}^{M}\Phi_i \Phi_i{}^T = A A^T$. Recall from Section 2.1.1 that we wish to find the top *d'* (*M* in this case) eigenvectors of the covariance matrix that best represents the data set with respect to the new coordinate system by solving equation (3). It follows that $Se = \lambda e \rightarrow e^T Se = \lambda e^T e \rightarrow \lambda_i = \frac{1}{M}\sum_{i=1}^{M} u_i{}^T (\Gamma_i - \Psi)(\Gamma_i - \Psi)^T u_i$, such that the *i*th largest eigenvalue λ_i and corresponding eigenvector u_i are subject to

maximizing $\lambda_i = \frac{1}{M}\sum_{i=1}^{M}\left(u_i^T \Phi_i\right)^2$, and $u_j^T u_i = \delta_{ji} = \begin{cases} 1 & i = j \\ 0 & otherwise \end{cases}$ so that eigenvectors $u_{i...M}$ must be orthonormal.

We note that since AA^T is $N^2 \times N^2$, computing for all N^2 eigenvalues and corresponding eigenvectors is infeasible for very large images. An optimized method given by Turk and Pentland [11] in computing the principal components first solves for the eigenvalues and eigenvectors of $A^T A$. By doing so, the total number of eigenvectors is reduced from N^2 to *M*. Mathematically, we solve for the eigenvectors v_i of $A^T A$ such that $A^T A v_i = \lambda v_i$, and multiply both sides by *A* yielding $AA^T A v_i = \lambda A v_i$. It is apparent that Av_i are eigenvectors of AA^T. To obtain the principal components u_i, we form the proper linear combinations of the mean adjusted images Φ by

$$u_i = \sum_{j=1}^{M} v_{ij}\Phi_j = Av_i,\ i = 1, \ldots, M \qquad (8)$$

Since $M << N^2$ for typical image sizes, only a relatively small number $M' < M$ of eigenvectors are needed. Therefore computation time to solve for the eigenvalues and corresponding eigenvectors is significantly reduced using this method. Using proper terminology, we refer to the set of eigenvectors computed from the covariance matrix of a set of images as eigenfaces, as they resemble ghostly images of faces. Finally, to project an input image onto the set of eigenvectors to obtain its corresponding weight vector we compute

$$\Omega^T = \{\omega_1, \ldots, \omega_{M'}\} = u^T\left(\Gamma_{input} - \Psi\right) \rightarrow \omega_i = u_i^T\left(\Gamma_{input} - \Psi\right) \qquad (9)$$

$for\ i = 1, \ldots, M'$. To reconstruct an image from Ω^T, we have

$$\left(u^T\right)^{-1}\Omega^T = \left(u^T\right)^{-1}u^T\left(\Gamma_{input} - \Psi\right) \rightarrow\ \Gamma_{input} = u\,\Omega^T + \Psi \qquad (10)$$

where $\left(u^T\right)^{-1} = \left(u^T\right)^T = u$ and $u\,u^T = I$ for orthonormal vectors $u_{1\ldots M'}$. We note that these equations are irreversible since the principal components u_i represent a lossy reduction in dimensionality such that some amount information is lost, but minimized with respect to equation (4).

Chapter 3

Initial Design and Implementation

3.1 Design and Evaluation

In this chapter and sections that follow, we describe, demonstrate, and evaluate key concepts and techniques that are used to support our main focus of face reconstruction and age progression.

3.2 Eigenface Program

The purpose of the eigenface program is to demonstrate the feasibility of the eigenface approach for face recognition and reconstruction. The eigenface method applies PCA to reduce a set of training images by representing them in terms of a more compact set of orthogonal vectors (principal components). An input image is then projected onto the set of principal components to obtain a weight vector that represents the proportions of each eigenface to reconstruct the input image.

3.2.1 Image Recognition

To extend the application of eigenfaces for face recognition, we use the face class and face space Euclidean distance metrics defined by $\varepsilon_{class\,k}{}^2 = \left\|\Omega - \overline{\Omega}_k\right\|^2$ and $\varepsilon_{face}{}^2 = \left\|\Phi - \Phi_{face}\right\|^2$ respectively. The Euclidean distance between two vectors $x = \{x_1, \ldots, x_N\}$ and $y = \{y_1, \ldots, y_N\}$ is determined by

$d(x,y)=\sqrt{\sum_{i=1}^{N}(x_i-y_i)^2}$. A face class is defined as a group of images (varying in orientation, lighting, and expression, etc.) of the same person (or object). The weight vector $\overline{\Omega}_k$ is the average of the weight vectors of images in face class k. For the face space metric, we project an input image onto face space consisting of *M'* eigenfaces, obtain its characteristic weight vector with equation (9), and compute $\Phi_{face}=u\,\Omega^T=\sum_{i=1}^{M'}\omega_i u_i$.

To perform face recognition, we first gather a set of *M* training images $\Gamma=\{\Gamma_1,\Gamma_2,\ldots,\Gamma_M\}$ for X face classes, and assign one or more images to each person such that each image of the person varies in facial expression, orientation, and lighting, etc. $L=A^TA$ is computed where $A=\{\Phi_1,\Phi_2,\ldots,\Phi_M\}$, $\Phi_i=\Gamma_i-\Psi$, and $\Psi=\frac{1}{M}\sum_{i=1}^{M}\Gamma_i$. The eigenvalues $\lambda_{i\ldots M}$ and corresponding eigenvectors $v_{i\ldots M}$ are computed from L , $v_{i\ldots M}$ sorted in the order of descending eigenvalues, and the top *M'* eigenvalues and corresponding eigenvectors are selected such that *M'* ≤ *M*. Next, the eigenfaces (principal components) are computed from equation (8). Then for each training image $\Gamma_{training}$, we compute the weight vector from equation (9). $\overline{\Omega}_k$ is obtained by averaging the weight vectors of training images in face class k. $\overline{\Omega}_{1\ldots X}$ represents the knowledge base that the PCA classifier uses to distinguish between the X face classes. Finally,

we specify face class and face space threshold values θ_{class} and θ_{face}, respectively, such that the classification rules in Table 1 apply.

Table 1: Eigenface classification rules

a) Image is a face and is recognized ($\varepsilon_{class\,k} < \theta_{class}$ and $\varepsilon_{face} < \theta_{face}$)
b) Image is a face and is not recognized ($\varepsilon_{class\,k} > \theta_{class}$ and $\varepsilon_{face} < \theta_{face}$)
c) Image is not a face and is recognized ($\varepsilon_{class\,k} < \theta_{class}$ and $\varepsilon_{face} > \theta_{face}$)
d) Image is not a face and is not recognized ($\varepsilon_{class\,k} > \theta_{class}$ and $\varepsilon_{face} > \theta_{face}$)

For input image Γ_{input}, we subtract Ψ from it to form mean adjusted input image Φ_{input}, and project Γ_{input} onto face space to obtain weight vector Ω^T. We then compute Φ_{face} and the face class $\varepsilon_{class\,k}$ and face space ε_{face} metrics, and classify the input image with respect to the rules in Table 1. Note that $\varepsilon_{class\,k}$ represents the smallest Euclidean distance between Ω^T and $\overline{\Omega}_k$ for face classes $k = 1, \ldots, X$.

The θ_{class} and θ_{face} threshold values are carefully chosen by trial and error, since they are critical to the recognition performance of the PCA classifier. In addition, one cannot be specified independently of the other in the case that input images are not in the same classification as the training images. For example, if the PCA classifier is trained with face images and input images are not faces, then specifying a high θ_{face} value causes the classifier to err for input images that are known not to be face images. On the other hand, specifying a low θ_{face} value

causes the classifier to be overly conservative and reject images that are known to be faces, regardless of whether a match is found for a face class. However, if we assume a priori that all images are face images, then θ_{face} may be disregarded by setting it to a sufficiently large value. We note that the optimal values for θ_{class} and θ_{face} to maximize the percentage of recognized faces is dependent on the set of input images. The classification rules in Table 1 essentially divide the input space into four classification groups. Therefore, the addition of new input images can change the values for θ_{class} and θ_{face} and shift the optimal division lines.

3.2.2 Program Functionality

The program functionality follows the algorithm and details as outlined in the previous section. Figure 2 shows the user-interface for the eigenface program. Prior to running the program, a set of training images is loaded and processed by initially assigning to each person (face class) a directory (name of which is also the name of the face class) containing one or more images of the person in different orientations, lighting, or facial expressions, etc. The user selects the root directory path where the face class directories are stored, and the program loads into memory the names of the face classes and the paths of all the associated training images. PCA training is then performed by computing the eigenfaces (principal components) of all the training images in the selected root directory, and determining the average weight vector of each face class. After the training process, the user can match a single face image to its

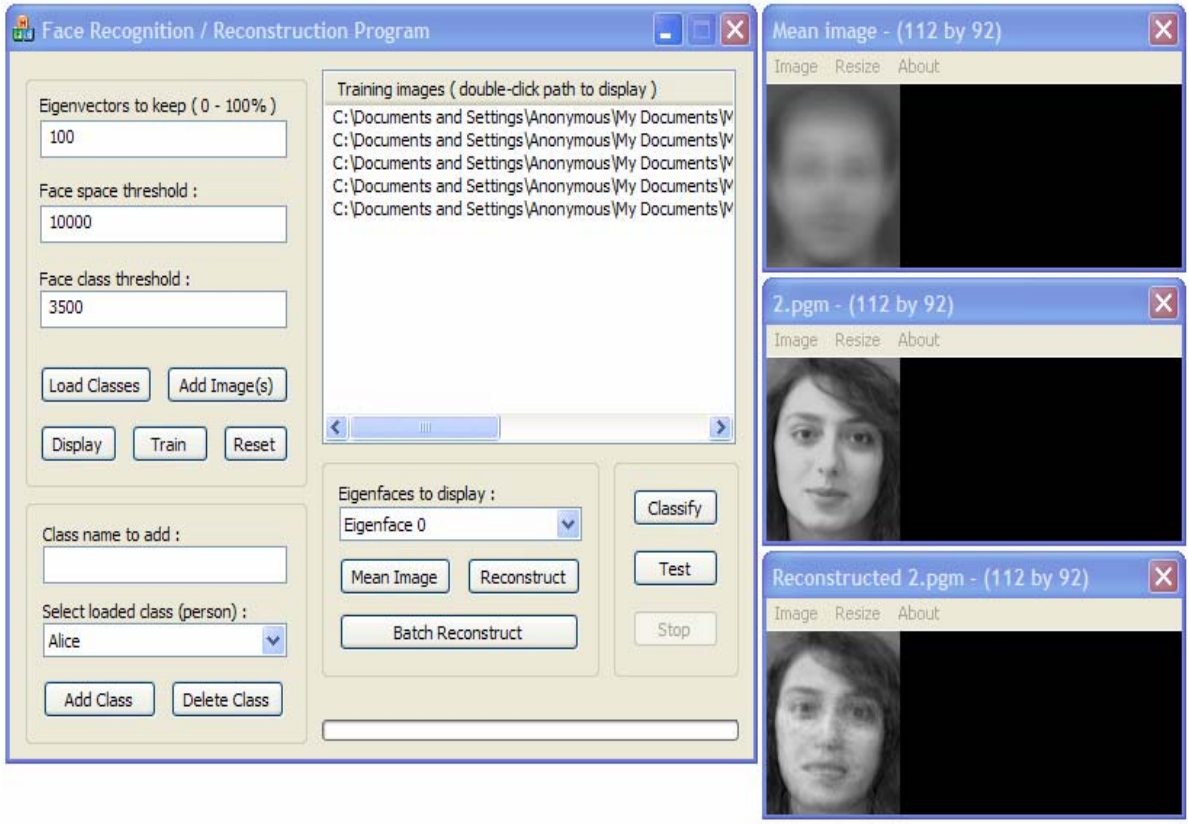


Figure 2: Eigenface program interface

corresponding face class, or classify a group of input images and determine the maximum face class and face space values. These values are then used to maximize the percentage of recognized faces from the set of input images. The eigenface program implementation also supports image reconstruction. Both image classification and reconstruction first projects the input image onto the set of eigenfaces to obtain its characteristic weight vector. For image classification, the input weight vector is used to compute the face class and face space metrics by $\varepsilon_{class\,k}{}^2 = \left\|\Omega - \overline{\Omega}_k\right\|^2$ and $\varepsilon_{face}{}^2 = \left\|\Phi - \Phi_{face}\right\|^2$, respectively. The classification rules of Table 1 are then used to classify the input image accordingly. For image reconstruction, the input weight vector is used to compute equations (9) and (10).

3.2.3 Program Design

The eigenface program is implemented in C++ and MFC (Microsoft Foundation Classes) for user-interface development in the Visual Studio environment. The class diagram of the eigenface program is shown in Figure 3. The CImageViewer and CEigenDlg classes are derived from the MFC class CDialog, which is used to display a dialog box on the screen. The CImageViewer class implements functionality to display grayscale images by using the MFC class CImage, which provides imaging support between the user and the windows device context of a dialog box. The CEigenDlg class implements functionality for all of the GUI controls. CStringToPtrMap is an MFC class that maps an MFC CString object to a pointer of type CObject. CFaceClassData inherits from CObject and maps face class names to CImageData objects, which encapsulate image pixel data stored in CMatrix objects. The CImageUtil class provides support for the reading, writing, and scaling of grayscale pgm images. The pgm image format represents pixel values in 8 or 16 bit P2 or P5 ASCII format. The P2 version represents each pixel value by an ASCII number string (e.g. 255 for white). The P5 version represents each pixel value by an ASCII character in binary format (e.g. character ÿ for white). The CWinUtil class supports the retrieval of path names of files or directories stored in a given directory, display of a directory search dialog box, and various path name manipulation functions. The CWinUtil class encapsulates code that is taken from the Windows Programming book [12]. CEigenSolver encapsulates functionality to solve and sort eigenvalues and eigenvectors of a symmetric real

matrix. The implementation is taken from Chapter 11 of Numerical Recipes in C [8]. The matrix class provides support for most matrix operations and is taken from the Matrix TCL website [19]. CMatrix extends the matrix class with additional functions specific to the computation of eigenfaces.

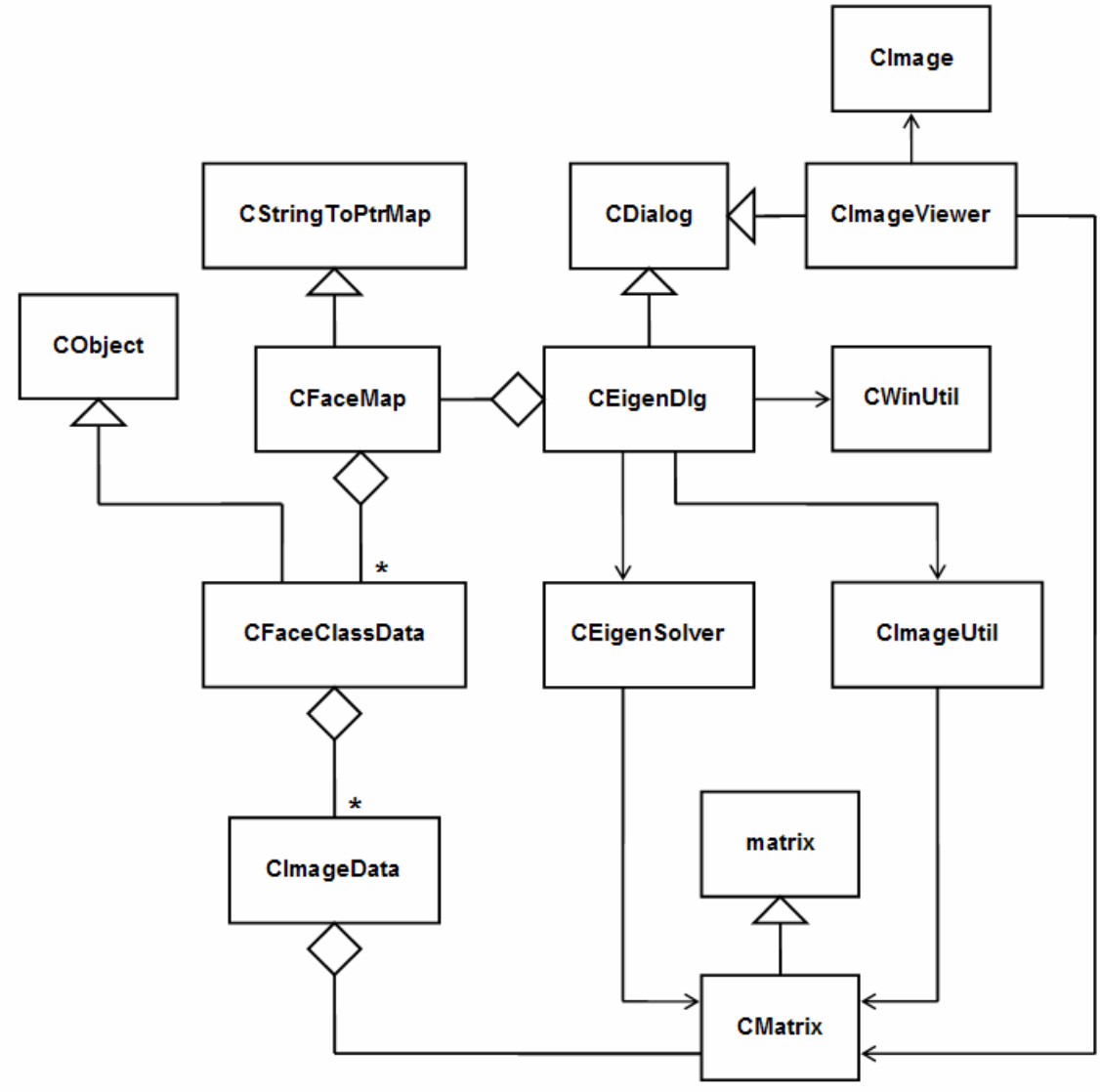


Figure 3: Eigenface program class design

3.2.4 Reconstruction and Recognition Results

To obtain image data for the eigenface program, we retrieve 400 pgm images from the Cambridge University Engineering Department Database of Faces [15]. These images consist of 40 face classes with 10 images for a given person in each face class that varies in lighting, facial expressions, as well as in-plane and out-of-plane face rotations, with no background details. To train the eigenface program, we create 40 face classes each assigned 5 images of the same person for a total of 200 training images. Figure 4 shows several training images. After training, we obtain eigenfaces and mean images as shown in Figure 5. To test the program, we reconstruct the remaining 200 non-training

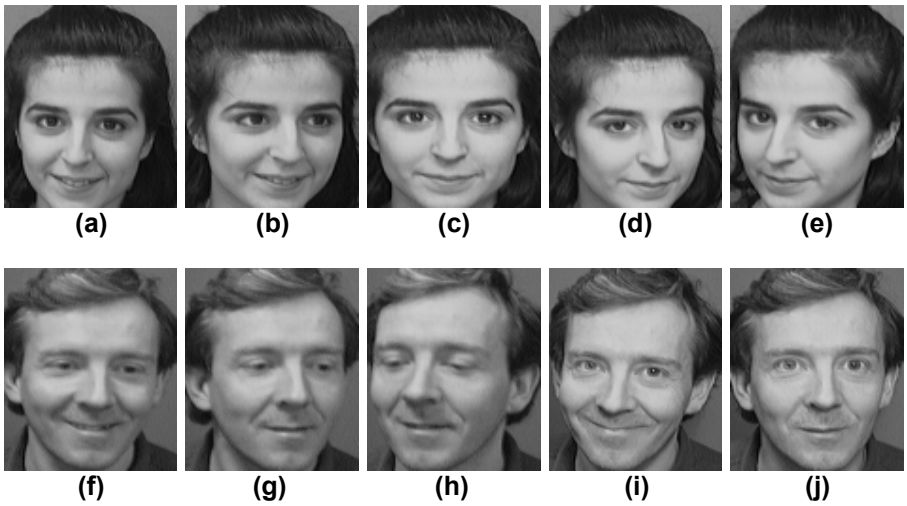


Figure 4: Training images of two face classes

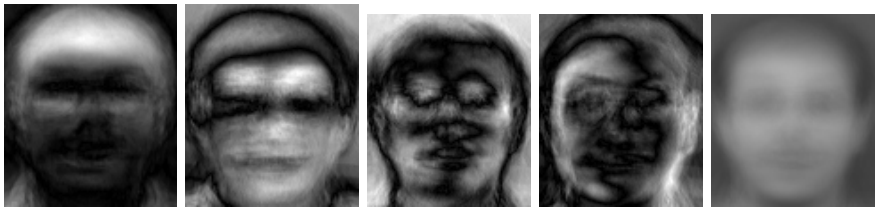


Figure 5: Example eigenfaces (left four) and mean image

images from the same face classes as the training images, and observe that about half are reconstructed such that they resemble human faces with reasonable clarity. For this test, we don't separate the input images by gender since they are from the same face classes as the training images. We also retain all eigenvectors to maximize reconstruction accuracy. Figure 6 shows several reconstructed images. We note that the lower images in Figures (6b) and (6d) look very similar to the images in Figures (4b) and (4i), respectively. Patterns observed from the test results demonstrate that if the input image looks similar to a subset of the training images, the reconstructed image also looks similar to this subset. Conversely, if an input image does not resemble a subset of training images, the quality of the reconstructed image suffers.

(a) **(b)** **(c)** **(d)**

Figure 6: Input (top) and reconstructed (bottom) images

To obtain recognition performance statistics, we classify the same 200 non-training images as in the reconstruction case. We obtain results in Table 2 by setting θ_{class} = 3602 and θ_{face} = 2680, just slightly larger than the maximum ε_{class} and ε_{face} values. This causes the classifier to regard all images as faces

and maximizes the percentage of correctly classified faces at the expense of increasing the percentage of incorrectly classified faces.

Table 2: Recognition results for 200 test images

a) Correctly classified faces	92%
b) Incorrectly classified faces	8%
c) Unrecognized faces (no face class matches)	0%
d) Not faces	0%
e) Minimum ε_{class}	1151.56
f) Maximum ε_{class}	3601.87
g) Minimum ε_{face}	1061.93
h) Maximum ε_{face}	2679.54

Figure 7 shows the reconstructed results of a training image using 100, 75, 50, 25, and 0 percent of top eigenvectors retained, respectively. As the percentage of retained eigenvectors approaches zero, the reconstructed image approaches the mean image, as predicted by equation (10). Even with only half of the eigenvectors, the reconstructed image is still discernible.

Figure 7: Eigenvector truncation results

3.3 Grayscale Age Progression

In this section, we describe the implementation of an image-retrieval and preprocessing tool that downloads applicable images from a missing children's web server, preprocesses the images in terms of size, orientation, format, and

creates a directory structure to organize and store the images. Specifically, this tool retrieves young and corresponding aged image pairs of a given person. The (young, aged) image pairs are then concatenated and input into the eigenface program to produce age-progressed grayscale images.

3.3.1 Image Retrieval and Preprocessing

The image retrieval and preprocessing tool consists of three command line Python scripts (*webcrawler.py*, *face_ops.py*, and *image_ops.py*) and a C helper program (trackface.exe) that calls a face detection library from http://vasc.ri.cmu.edu/NNFaceDetector [16]. Figure 8 illustrates a high-level overview of the image retrieval, preprocessing, and age progression process for grayscale images.

The function of the *webcrawler.py* script is to communicate with a specific missing children's web server and download all missing children's images by state (e.g. California). Each missing child is represented by one young image and a corresponding digitally aged image. This script interacts with a Java servlet object by sending the appropriate parameters. For example, the url http://www.missingkids.com/missingkids/servlet/PubCaseSearchServlet?act=usMapSearch&missState=CA&searchLang=en_US returns a web-page that contains images of all missing children in the state of California. The *webcrawler.py* script parses this web-page for all the relevant image tags and uses the url's to download the images. Each (young, aged) image pair is stored

in a directory assigned to a given missing person. All image directories are then stored in a specified root directory.

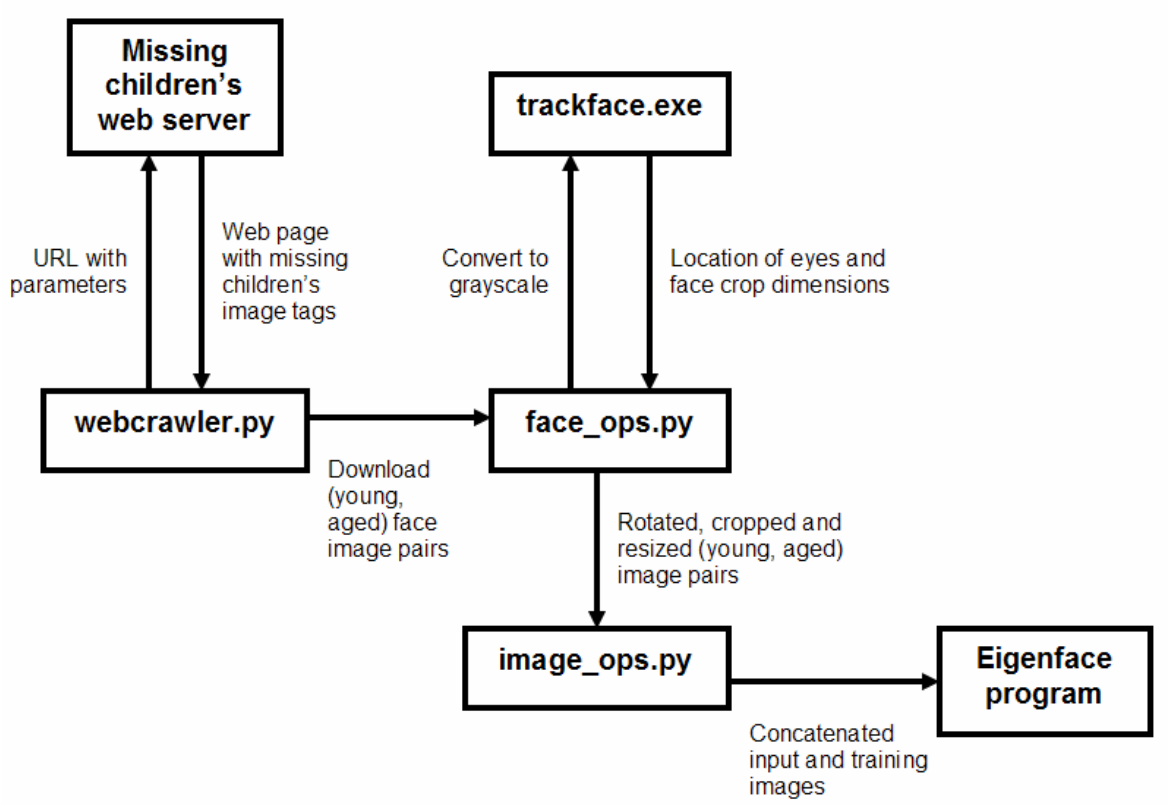


Figure 8: Image retrieval, preprocessing, and age progression process

The *face_ops.py* script traverses the directory structure created by *webcrawler.py* and preprocesses all images stored in it. First, the script converts each image to grayscale pgm image file format. It then invokes a helper program trackface.exe to read the pgm image. The helper program uses a neural network based face detection library [16] to locate the boundary of a face and center points of the eyes. It then saves the dimensions to a file that *face_ops.py* parses. The *face_ops.py* script reads the dimensions and then crops a face in each image to maximize the face area (i.e. reduce pixels covered by hair and

background), and corrects any in-plane rotation so that faces are vertical. The in-plane rotation angle is determined between a line through the centers of the eyes and the horizontal axis, and is applied to the center of an image. Each rotated and cropped (young, aged) face pair is resized, concatenated, and stored in a specific directory structure. The *image_ops.py* script traverses the directory structure created by *face_ops.py,* and creates a new directory structure that contains concatenated input or training grayscale pgm images as illustrated in Figures 11 and 13, respectively. We rely on the Python programming environment and imaging library to provide the necessary image processing tools and operations as needed.

3.3.2 Age Progression Test Results

We use the eigenface program to train on a number of concatenated images created by *face_ops.py*. Then we input a concatenated image of the same young face on both sides of the image into the eigenface program. We wish to obtain a reconstructed image with one side of the image aged-progressed and extracted. We first download a number of images from http://www.missingkids.com by using the *webcrawler.py* python script. An example pair of young and aged images is shown in Figure 9. The Python scripts *face_ops.py* and *image_ops.py* are then invoked to produce a set of preprocessed training images as shown in Figure 10. Given a concatenated input image as shown in Figure 11, the eigenface program produces a

reconstruction of the input image that is shown in Figure 12. The corresponding training image is shown in Figure 13. Note that the input image in Figure 11 is

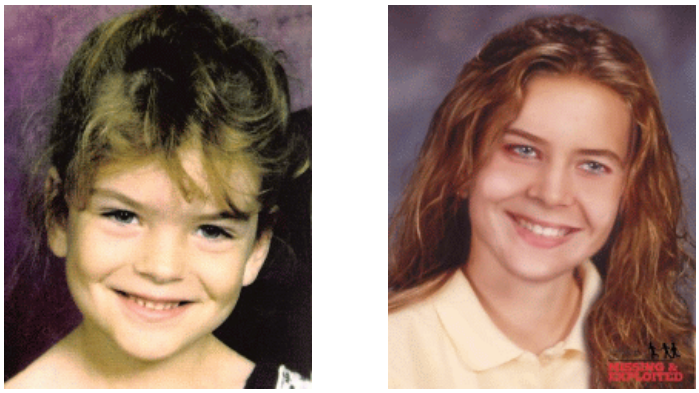


Figure 9: Young and corresponding aged image pair

the same left face of Figure 13, and therefore the resultant image should be well reconstructed. In general, however, we want to reconstruct input images that are not from the training image set.

Figure 10: Concatenated training image

Figure 11: Concatenated input image

Figure 12: Reconstructed input image

Figure 13: Concatenated training image

We retrieve and preprocess 300 female (young, aged) face pairs (100 by 100 pixels) with the Python scripts *webcrawler.py* and *face_ops.py*. We assign 200 of them as concatenated training images as in Figure 10, and the rest as young concatenated input images as in Figure 11. For this test we do not separate the training images according to any criteria, and essentially gather them in a single group. We use the eigenface program to reconstruct the 100 input images without truncating any eigenvectors, and visually determine that 60 out of the 100 resultant images are reconstructed successfully, and 15 of the 60 successfully reconstructed images are adequately age-progressed. Doing the same for 150 male training images and 50 male input images, we observe 30 out of the 50 resultant images are successfully reconstructed, and 10 of the 30 successfully reconstructed images are adequately age-progressed. Figure 14 shows successful grayscale reconstructions of aged female and male grayscale images for this test. Of the successfully reconstructed images that are not well age-progressed, the eigenface program is not able to discernibly age the images.

Figure 14: Grayscale reconstructions of aged (right side) female and male

3.4 Feature Matching Program

In this section, we discuss the feasibility of shape contexts as described in the paper by Belongie and Malik [2] to provide a robust means of quantifying and representing the shape of an object. This representation is then used in conjunction with other techniques to locate features on face images. The idea is to manually select features on a template image, perform edge detection and sample points from the edges, and then find points on a test image that best correspond to sampled points of features on the template image.

3.4.1 Shape Context and Bipartite Matching

Shape context is characterized by the spatial relationship between a given point and all other points in a shape. Specifically, it is defined by a set of points sampled from the internal or external edges of a shape. The edges are determined by using an edge detection algorithm that searches for regions of changing image intensity and localizes the edges between these regions. This application uses the Boie-Cox edge detector from Practical Algorithms for Image Analysis [10]. Figure 15 shows an example of an edge-detected image.

Figure 15: Edge detected image

Since face images are inherently random, it is generally difficult to register points on the edges of an edge-detected face for exact spatial correspondence and ordering of points between two images. Therefore, we use a random sampling of points for this application. For a given sampled point, a shape context descriptor is defined by determining the set of vectors from the point to all other sampled points on the shape. Specifically, the shape context for a point is a log-polar histogram that sorts all vectors for a given point by relative distance and angular orientation. Therefore, for *N* sampled points, each point has *N - 1* vectors to all other points. And the corresponding log-polar histogram that has *x* radial and *y* angular separations has *x * y* bins. The log-polar plot can be visualized as a series of concentric circles enclosing a number of wedge bins. The wedge bins are uniform in angular spacing, but vary logarithmically in the radial direction. In other words, in Cartesian space, the inner circles are closer together than the outer circles. But in log space, the circles are spaced uniformly in the radial direction. Figure 16 below illustrates this. The second picture shows

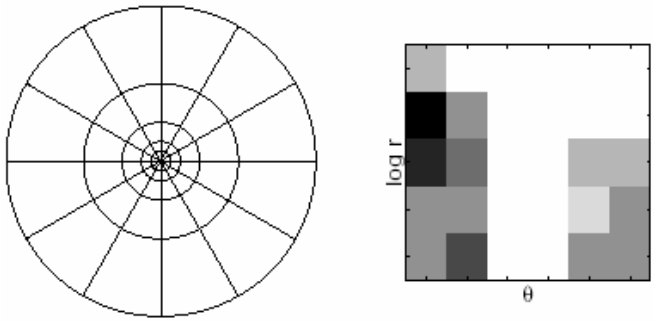


Figure 16: Log-polar bins and corresponding flattened histogram [2]

a flattened shape context histogram with the relative darkness of the bins indicating point density. The advantage of using shape contexts is that several geometric invariances are inherent with this method. Specifically, invariance to translation is built-in since vectors are between all points on a given shape. Invariance to scaling is achieved by normalizing all vectors with respect to the mean vector for all *N * (N - 1)* pairs of points. Rotational invariance is achieved by determining the average vector from a point to all other points and using this vector as the axis to measure the angle from for each point.

Qualitatively, the shape context for each point gives a precise description of the relative position of a point to all other points. Thus, the shape context can be used as an effective measurement of shape similarity for a corresponding point on another shape to be compared. This implies that there must be a point-to-point correspondence between points on two shapes. To obtain point-to-point correspondence, the log-polar histogram representing the shape context for each point on a shape, may be used to calculate the measurement cost between points on two shapes. Specifically, the χ^2 (chi-squared) distance metric is used. Given two shapes each with N sampled points, a cost matrix of size $N \times N$ is set up where each matrix element is the χ^2 cost between points on the two shapes. The cost equation is defined by $C_{ij} = \frac{1}{2}\sum_{k=1}^{K}\frac{\left[h_i(k) - h_j(k)\right]^2}{h_i(k) + h_j(k)}$ for *K* bins, where C_{ij} represents the cost between points p_i on one shape and p_j on the other. The term $h_i(k)$ represents the k^{th} bin of the normalized histograms at p_i and p_j. In

essence, we are comparing the similarity between histograms at two points. The denominator for C_{ij} may go to zero, in which case we just assign a zero to the ratio for a particular k.

Therefore, the matrix represents the cost between all possible pairs of points, and is an instance of the weighted bipartite matching problem. This problem may be cast in the more familiar form of the machine scheduling problem, where the goal is to assign one task to every machine so that every task will be tended to. The cost elements of the matrix measure the effectiveness of a machine to perform a specific task, while the objective value aims to maximize the value for all machine and task pairs. In modeling the bipartite matching problem with respect to shape contexts, we wish to minimize the total cost for all point pairs. The bipartite matching problem may be solved in $O(n^3)$ by using the Hungarian algorithm. This application uses a more efficient algorithm given by Jonker and Volgenant [14] in solving the linear assignment problem.

By solving the bipartite matching problem, we have the optimal correspondence between sampled points from features on the template and test images. We then determine the centroid of each group of points, which approximately locates corresponding features on the test image. For each group of corresponded points on the test image, we remove outliers greater than a

number of standard deviations from the mean radial distance between the group center and each point.

3.4.2 Program Functionality

As detailed above, the application first performs edge detection to retrieve the edge data of images. These edges are then sampled for both test and target point-sets for the two images to be mapped. Next, point-to-point correspondence is performed between the two point-sets to obtain a minimum overall cost. The

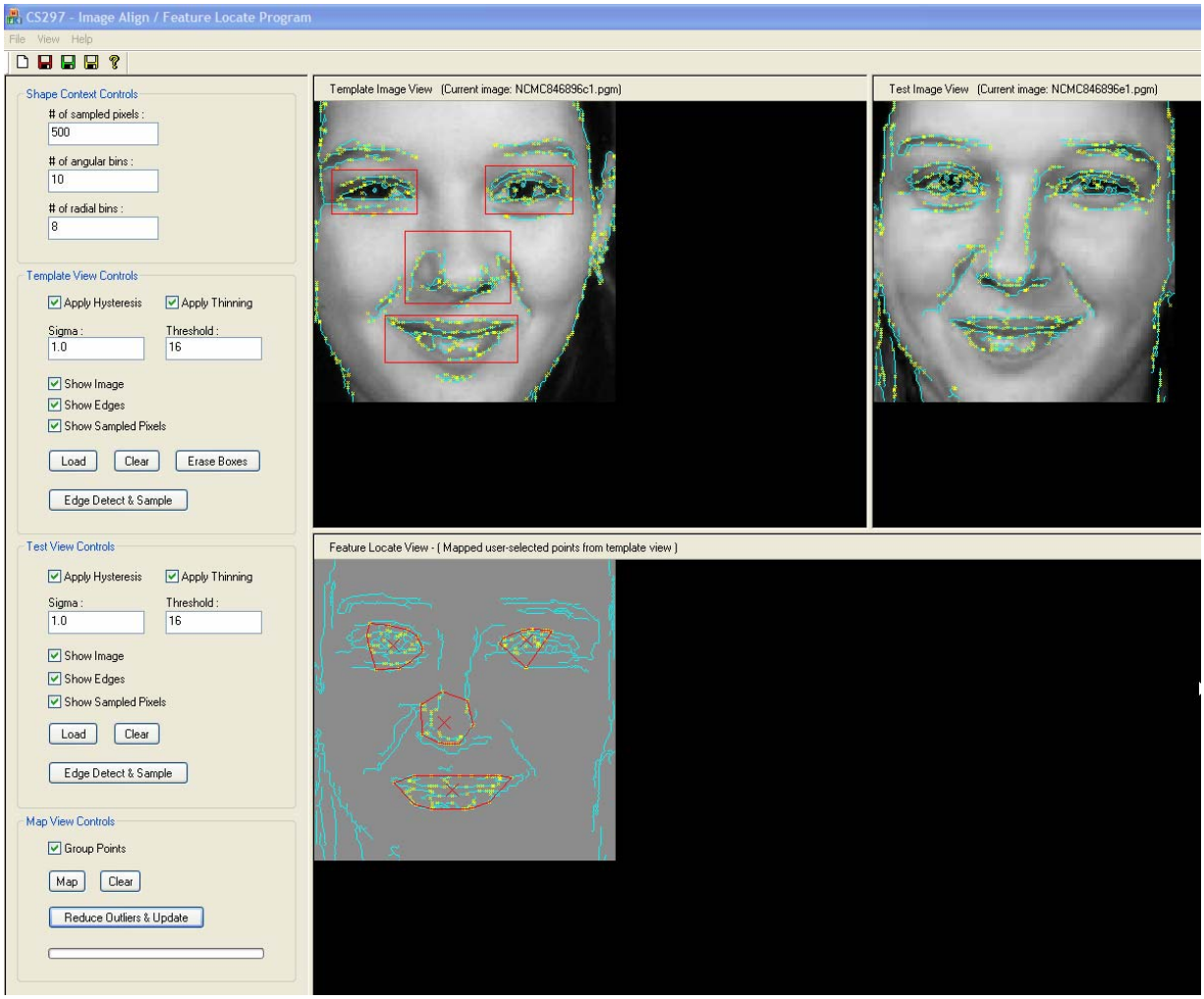


Figure 17: Feature Matching Program Interface

user-interface shown in Figure 17 consists of four (control, template, test, and feature-locate) views. The control view allows the user to load, display, edge detect, and sample points from images in the template and test views. After determining the optimal mapping of points, the user selects features in the template view to be corresponded to features in the test view, which are then displayed in the feature-locate view. The program takes all the user-selected points from the template view, maps it to the test image, and removes outliers that are not within a specified number of standard deviations from the group center. Next, the convex hull of each group of matched and culled points are computed, and displayed along with the group centers in the feature-locate view. The user can fine-tune overall performance by modifying sigma and threshold values for edge detection sensitivity, angular and radial bin count for point matching performance, and sample point count from the edge data.

3.4.3 Program Design

The feature matching program is implemented in C++ and MFC and supports the display of pgm images only. A class diagram of the feature-matching program is shown in Figure 18. The CControlView class implements functionality for all of the GUI controls, and is derived from MFC class CFormView, which is a view with embedded GUI controls. The scroll-enabled CTemplateView, CTestView, and CFeatureView classes provide support for the display of grayscale template, test, and feature-matched images, respectively. The CPGMUtil class implements functionality to read, write, and superimpose

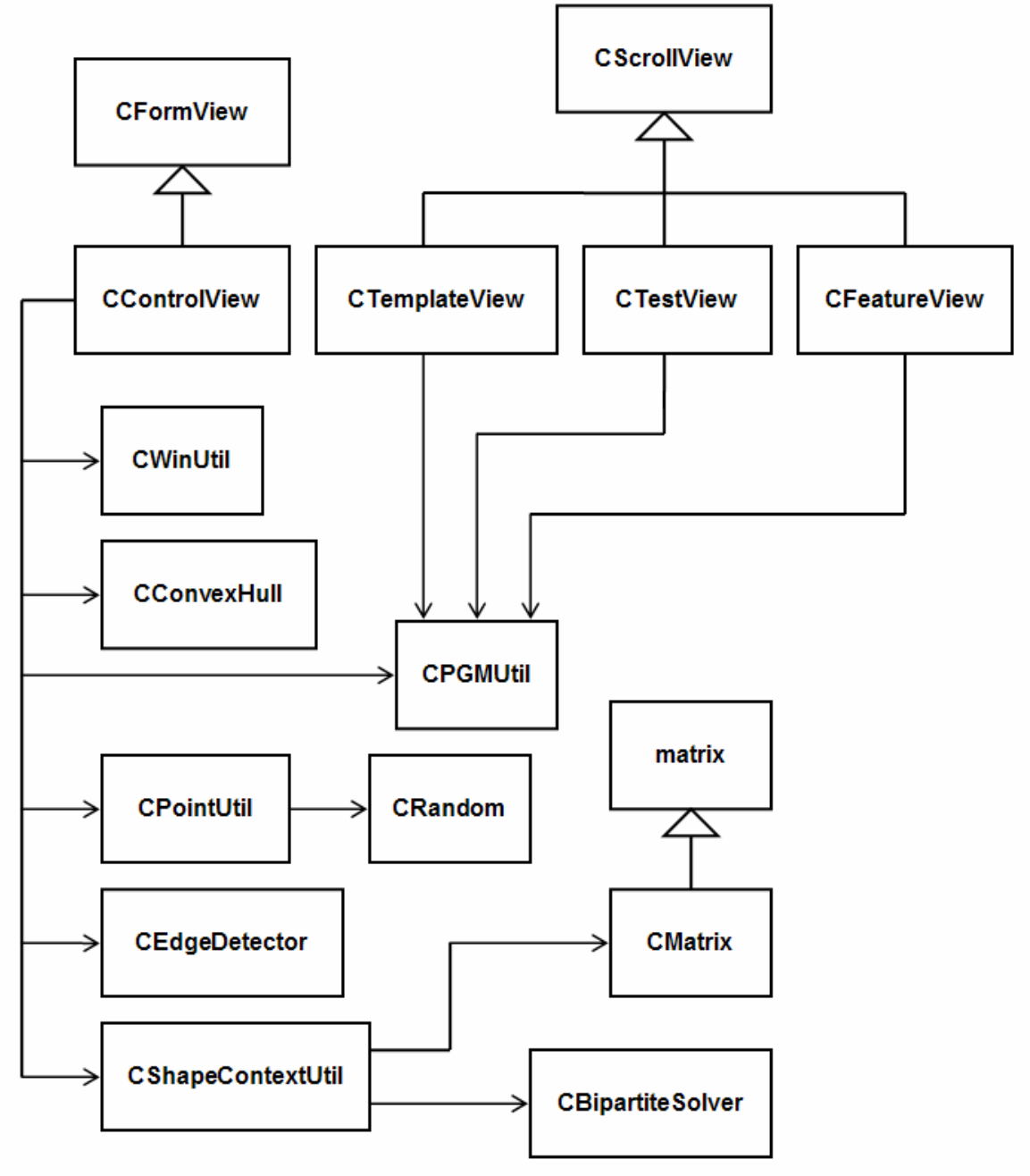


Figure 18: Feature matching program class design

different images (e.g. sampled points and edges overlaid on top of a face image). The CPointUtil class implements functions for random point sampling from edges, and determining the centroid and removing outliers from a group of points. CShapeContextUtil implements the shape context descriptor as detailed in Section 3.4.1. The implementation takes two sets of points and rearranges one of them such that each pair of points is (least-cost) optimal in accordance with a cost matrix determined by the shape context descriptor. CShapeContextUtil relies on the CBipartiteMatcher class to solve the point correspondence problem represented by a cost matrix. The CBipartiteSolver class encapsulates functionality and source code taken from www.magiclogic.com/assignment.html. CConvexHull implements Graham's scan algorithm and encapsulates source code taken from Computational Geometry [6]. The CEdgeDetector class encapsulates edge detection functionality and source code taken from Practical Algorithms for Image Analysis [10]. The CWinUtil and CMatrix classes are described in Section 3.2.3.

3.4.4 Feature Matching Test Results

We test the program with three sets of data consisting of shape and face images, and obtain test results by experimenting with various values of detector sensitivity, angular and radial bin counts, and number of sampled points. We select values for detector sensitivity, bin count, and sampled point count that avoid excessive detected edges, decrease in point matching performance, and increase in computation time, respectively. Excessive detected edges imply a

higher probability that points may be sampled from non-feature edges, which biases the cost matrix towards non-features. A large bin count causes the cost matrix to approach a zero matrix and degenerate into random point matching, assuming a random sampling of points from the edges. Increasing the number of sampled points also increases the size of the cost matrix, which is solved by the linear assignment algorithm [14] in $O(n^3)$.

To test the program with shapes, we use pgm images of alphanumeric characters for a total of 36 pairs of test images (A – Z and 0 – 9). With this test set, the program successfully locates features on 32 of the 36 character images. Figure 19 shows several feature-located images for various shapes. Next, to test

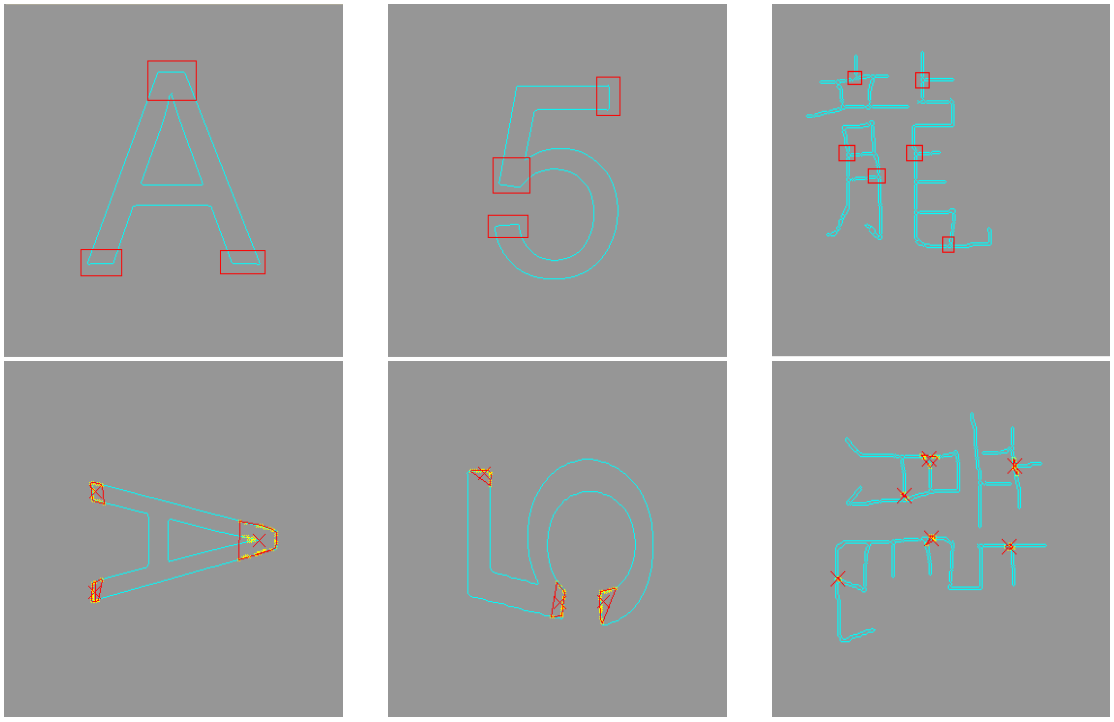


Figure 19: Shape matching results (selected boxes – top, matched features – bottom)

the program for face images of the same person, we use 50 (young, aged) image pairs retrieved and preprocessed by the Python scripts *webcrawler.py* and *face_ops.py* described in Section 3.3.1. The program successfully locates user-selected features (e.g. eyes, nose, and mouth) for 30 of the 50 (young, aged) pairs. Figure 20 shows a successfully feature-located test image. We then test 50 pairs of face images of different persons and successfully match features for 15 face pairs. Figure 21 illustrates an unsuccessful attempt at feature location for two different faces.

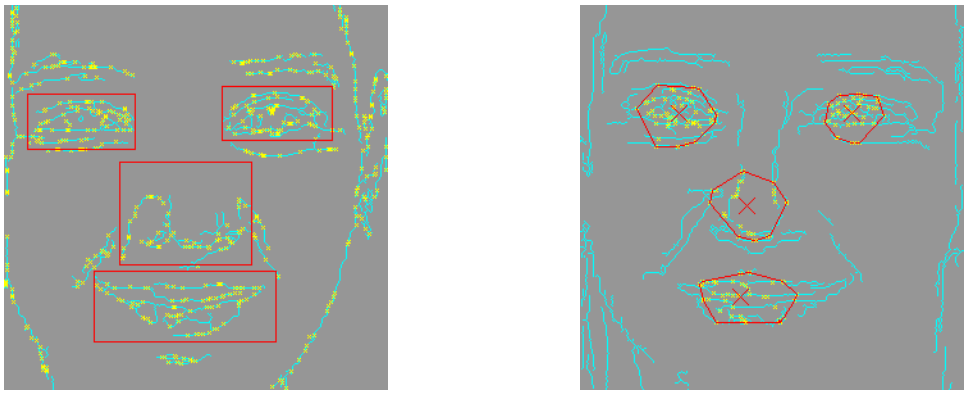


Figure 20: Successful feature matching of (young, aged) pair of same person

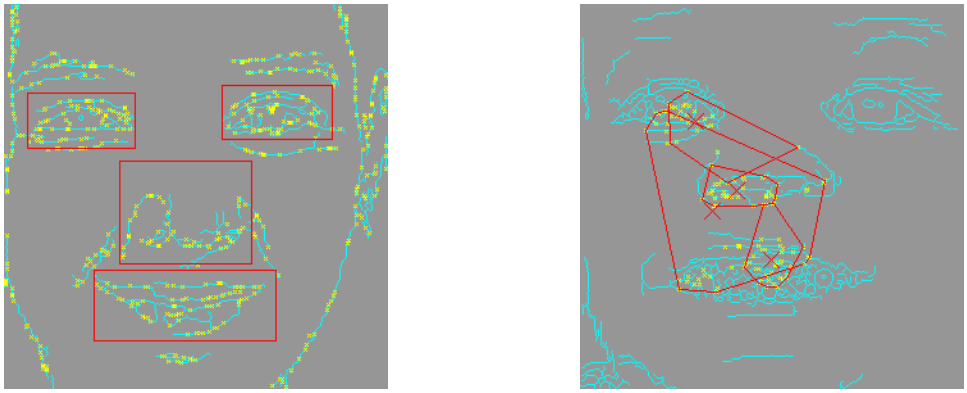


Figure 21: Unsuccessful feature matching of different faces

3.5 Discussion of Initial Results

The test results from Section 3.2.3 suggest that an input image must be similar to a subset of the training images to be accurately reconstructed. More specifically, we want the projected input image onto the transformed coordinate system to be in close proximity to a cluster of training images with specific aged features. However, PCA is an unsupervised technique such that the outputs are dependent solely on the training data with no direct control over how the clusters are formed. Therefore, the optimal training image set is one that contains a sufficient number of faces of different people, as well as adequate variation in expression, orientation, lighting, and other parameters for a given person, etc. This increases the probability that the projected input image is positioned near a cluster of training images that captures the desired age traits, and with the reconstructed image resembling the input image, since PCA minimizes the error between the two.

Image dimensionality is another factor that must be considered. By increasing the dimensions of the training images, we increase the number of parameters that may be used to discriminate the different clusters. As a result, we increase the dimensionality and descriptiveness of the set of principal components. Also, the dimensionality reduction process is more effective since for high-dimensional data, the number of eigenvectors that may be truncated far exceeds the dimensions needed to accurately represent a data point in the new

coordinate system. We conclude that image reconstruction for smaller images is more sensitive to differences between input and training image data.

To mitigate the limitations of inadequate range and size of the training data in terms of the number of available images, Section 3.4 begins exploring feature-based reconstruction and age progression by evaluating a solution for locating features on a face image. The results of the feature matching program indicate that using shape contexts and solving the bipartite matching problem, provide a robust solution for locating features for similar shapes with an exact or similar ordering and configuration of vertices. However, this solution does not work well for images that do not have a well registered set of vertices. Given that face images are inherently random, it is difficult to obtain a precise registration of points between images. We look to a more robust method of locating features by using the face detection library from [16], and to explore the feasibility of using color images to incorporate RGB data and increasing reconstruction performance.

Chapter 4

Final Design and Implementation

4.1 Feature-Based Age Progression

In this chapter and following sections, we discuss the extension of the eigenface program to support feature-based face reconstruction and age progression for color images. Since PCA is an unsupervised training technique, there is no direct control over the training process. By extracting and analyzing individual image features, this gives some indirect control over how individual features are trained and reconstructed. In addition, there is less variation for individual features as compared to an entire face, and as a result the probability of accurately reconstructing a feature increases. Next, we conclude from the results of Section 3.4.4 that a more reliable method of locating features on a face image is needed. Therefore, we use a neural network based face detection library [16] that is described in Section 3.3.1. Also, since this application supports color images, and eigenfaces are derived from grayscale images, we use an image matrix encoding to incorporate the (red, green, blue) values of a 24-bit color image in a format that supports matrix operations.

4.2 Feature-Based Overview

In feature-based age progression, the process first searches for key features on a face image (i.e. eyes, nose, mouth, and face). Then, features are extracted and encoded in a specific image matrix format as in Figure 25. This is

done for each young and corresponding aged image, results of which are concatenated to form a training image that separates the RGB intensity values and creates a mapping between each (young, aged) image pair. Next, eigenfeatures (principal components) are computed from the sets of training images for each feature on a face image. To age progress an input image, the features are extracted, projected onto their corresponding eigenfeatures, and reconstructed to obtain age-progressed features (Sections 2.1.3 and 3.1.2). These age-progressed features are then blended back to respective locations to form an overall age-progressed face. Figure 22 illustrates this feature-based aging process.

4.2.1 Feature Detection and Extraction

To locate features on a face image, we use the neural network based face detection library from [16]. However, this face detection library only locates the center of the eyes and face crop boundary of a head image. Therefore, we use the position of the eyes and face to locate and bound the rest of the features (i.e. nose and mouth). More specifically, we use the distance between the left and right eye as a base metric from which to estimate the locations and sizes of the other features. Figure 23 illustrates a simple dimension scheme to parameterize all major features on a face.

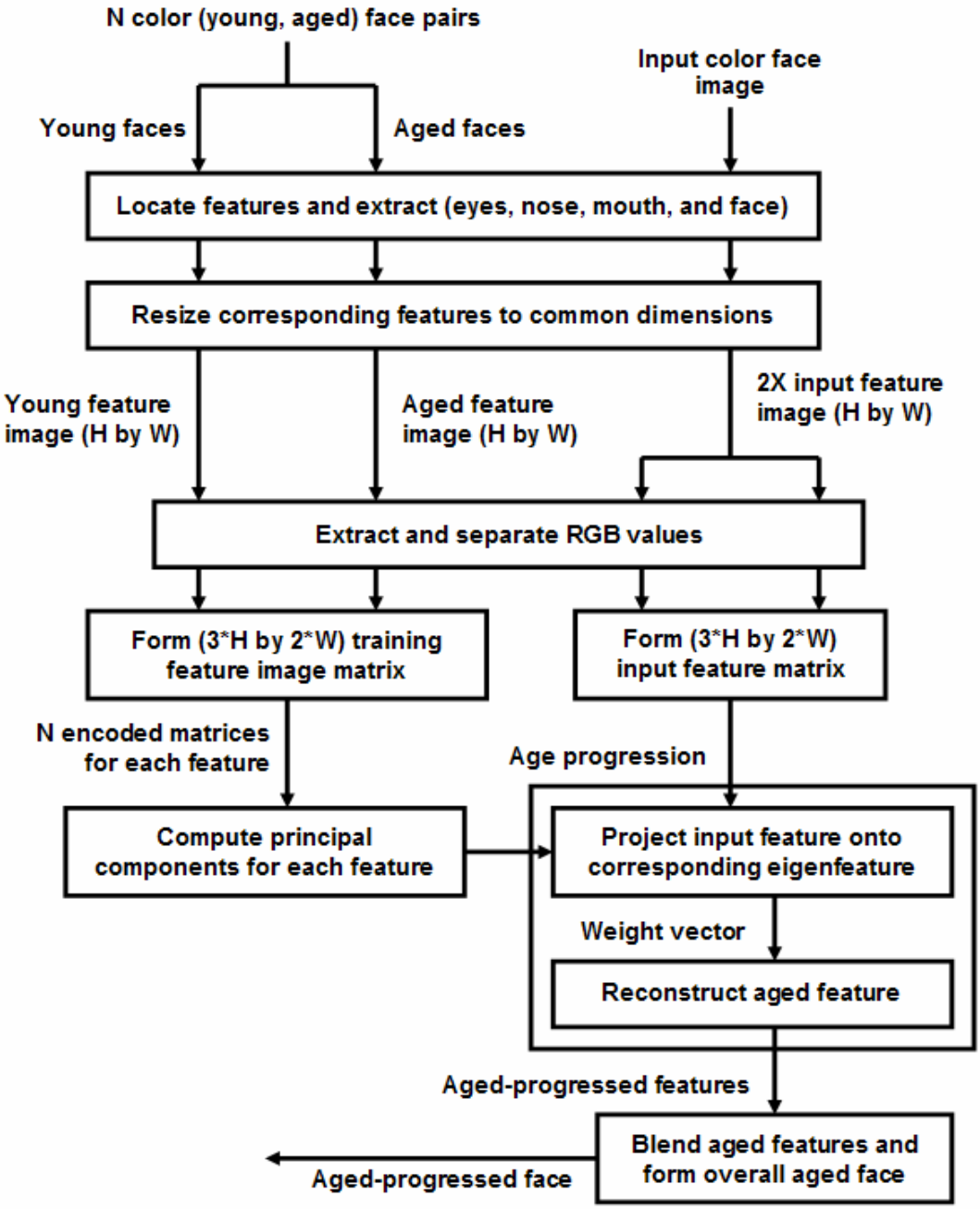


Figure 22: Feature-based age progression process

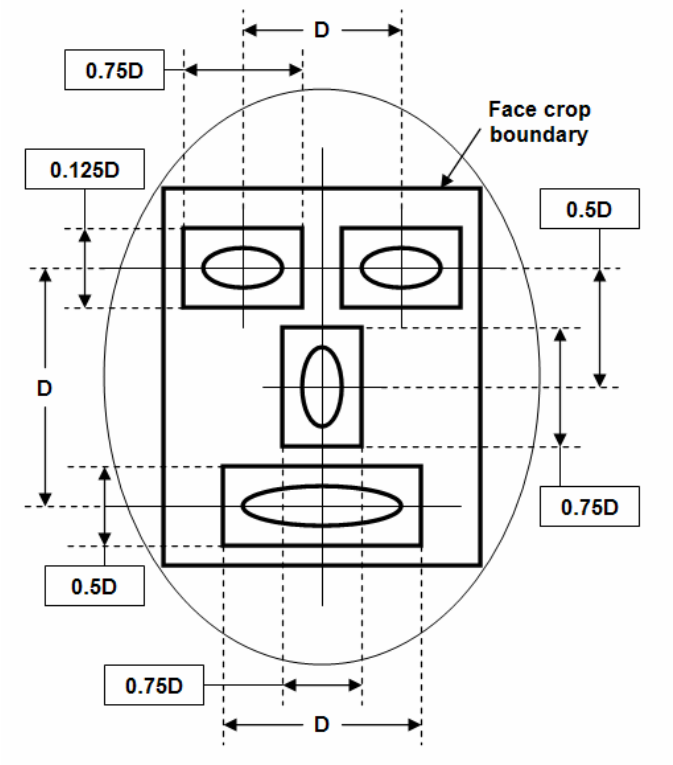


Figure 23: Parameterized face dimensions (not to scale)

This dimension scheme, obtained by trial and error, only gives rough estimates of the dimensions, assumes that certain face feature ratios do not vary significantly between face images, and works best for frontal faces (no out-of-plane rotations). To refine the locations and sizes of the face feature boundaries, we use a simple bounding box algorithm. The algorithm first re-centers a rough bounding box by performing edge detection on the bounded feature, and using the edge map within the rough boundary box to approximate the new center. It then determines the extremities of the edge map within the re-centered box and uses them to set the borders of the new bounding box. Figure 24 shows several feature-detected face images.

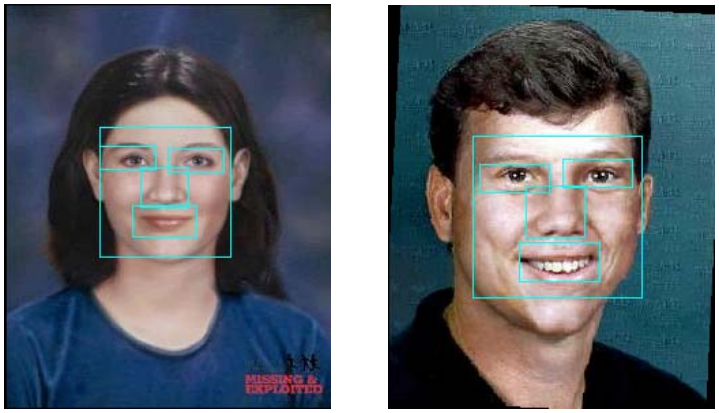


Figure 24: Feature detected faces

As Figure 25 illustrates, the RGB matrix format is a concatenation of the red, green, and blue color channels for a young face image that is concatenated with the same results of an aged face image. Therefore, it is essentially six

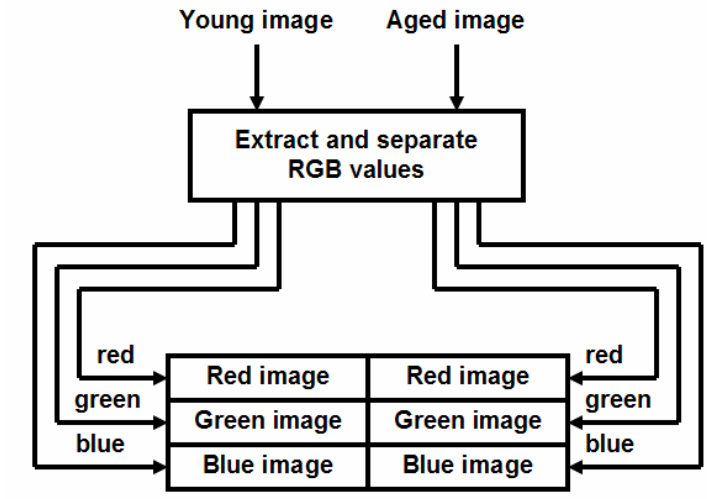


Figure 25: RGB image matrix format

images concatenated into a larger one. Since PCA calculates the eigenvalues and eigenvectors of a real matrix for this application, this is a simple method of incorporating the RGB values into a format that supports matrix operations. It also allows an aged-progressed image to be easily extracted by truncating the

proper half a reconstructed image, and then multiplexing the RGB values into a color image.

Subsequent to feature-detection, in-plane face / head rotation is removed. The corrective angle θ is measured between a line passing through the center of the eyes and the horizontal axis, and the corresponding rotation is applied at the center of the face image. We obtain the new centers of the eyes by applying the 2D rotation matrix $\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$. By applying the rotation angle θ, all features are then centered about the mid vertical axis. The dimension scheme in Figure 23 is then used in conjunction with edge detection to determine the location and best fit bounding box of each feature. Next, since PCA derives principal components from data with the same dimensionality, corresponding features extracted from each face image are resized to the common dimensions to ensure valid matrix computations.

4.2.2 Feature Blending and Reconstruction

Subsequent to the computation of the principal components (eigenfeatures) of each feature, we extract and age-progress features from an input image by projecting them onto their corresponding eigenfeatures by equation (9). Age progression is performed by reconstructing the features from the weight vectors by equation (10). An overall age-progressed face is then formed from the constituents as illustrated in Figure 26. To blend in the individual aged features, edge detection is first applied to determine the

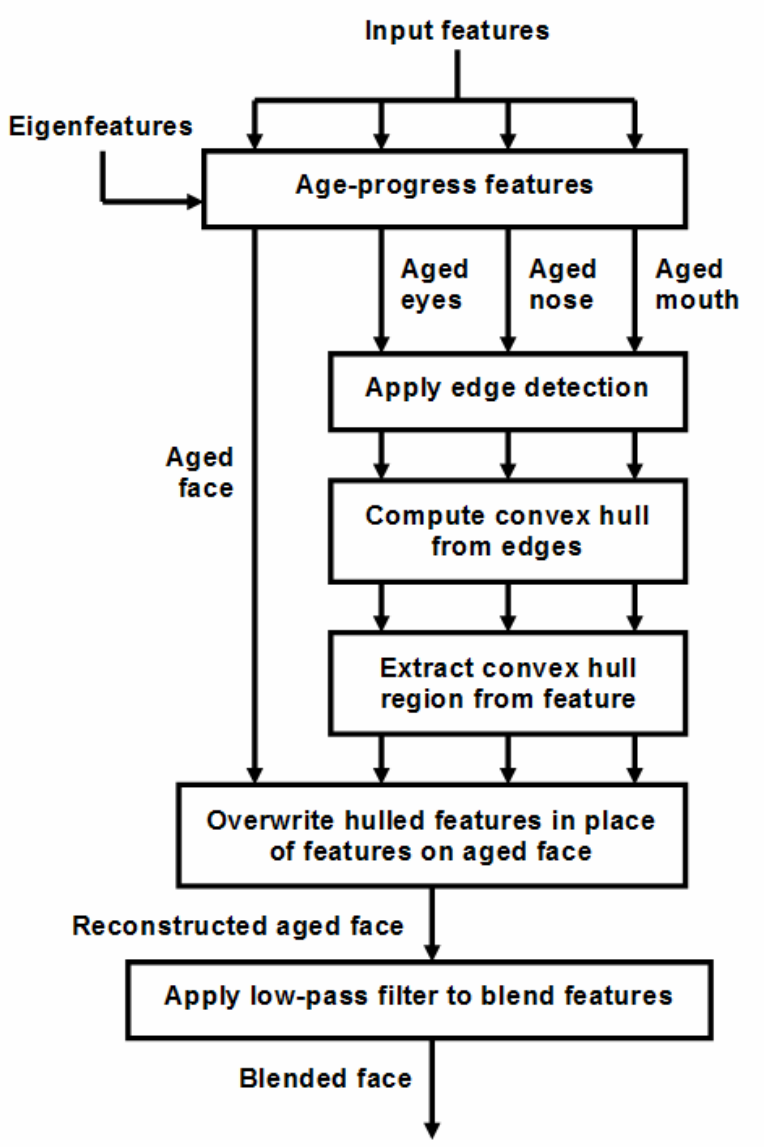


Figure 26: Reconstruction and blending process

extremities of each feature. The convex hull is then computed from the edges to obtain a rough contour. By doing so, each feature can more readily blend into the face without leaving behind demarcation lines of the bounding boxes around each feature. We then substitute the hulled features onto the aged face in their respective locations, essentially overwriting the features on the aged face. The final post-processing step involves applying a low-pass Gaussian filter to remove

high-frequency effects, resulting in the slight blurring of the image to further reduce the lines of demarcation left from the blending process. This filter convolves a square kernel matrix with a Gaussian profile to each pixel of the input matrix, essentially applying a neighborhood averaging effect that emphasizes the center kernel values. The Gaussian profile provides a more effective filter as compared with a uniform filter by removing noise while preserving detail. Figure 27 shows features that are isolated to highlight the approximated contour, and the corresponding aged and blended image.

Figure 27: Input image, aged and contoured features, and blended image

4.3 Program Functionality

The feature-based age progression program is implemented in the Microsoft Visual Studio environment using C++ and MFC, and incorporates most of the functionality illustrated in Figure 8 except for the retrieval of images, thereby eliminating the need for multiple modules. Figure 28 shows a screenshot of the user-interface. The program's interface consists of four (control, training,

input, and output) views. The left control view allows a user to load, train, and reconstruct color images, manually modify the in-plane rotation angle of an input

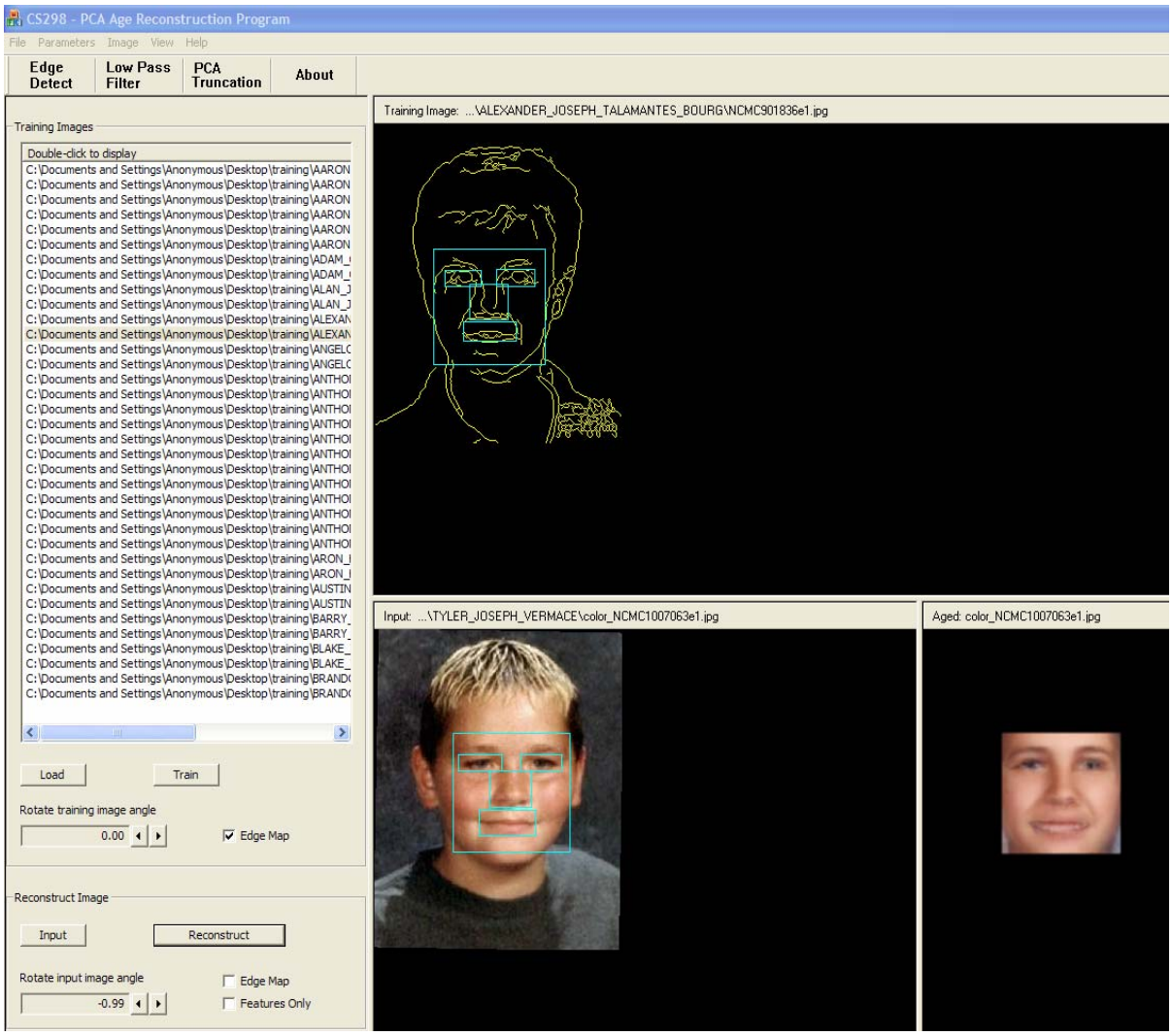


Figure 28: Feature-based age progression program interface

or training image, display the edge map of an image, and select a feature-detected training image to display. The top-right training view displays feature-detected training images, and allows the user to modify the feature detection results by manually dragging and selecting new feature-bounding boxes in the

view area. The lower-middle input view displays the image to be age-progressed, and also allows user to drag and select bounding boxes around features to be age-progressed. And the lower-right output view displays the reconstructed and age-progressed image. The user may also fine-tune parameters for edge detection and low-pass filter sensitivity, and percentage of eigenvectors to truncate. Modifying edge detection sensitivity affects the location of the bounding box of each detected feature. Increasing filter sensitivity causes the output image to be more blurred. And truncating eigenvectors decreases execution time at the expense of reconstruction quality.

To run the program, we use the Python script *webcrawler.py* described in Section 3.3.1 to retrieve the needed training (young, aged) image pairs. Each pair is stored in a separate directory, with all directories stored in a root directory. The user selects the root directory and the program loads all the training (young, aged) image pairs into memory. PCA training is then performed by first invoking the face detection library to locate the features (i.e. eyes, nose, mouth, and face) for all training images. The extracted features from the young and aged images are formed into training images in the format illustrated in Figure 25, and the principal components are computed for each respective feature. Next, the user selects an input image to load into memory to be reconstructed and age-progressed as illustrated in Figure 22.

4.4 Program Design

A class diagram of the feature-based age progression program is shown in Figure 29. The CControlView class implements functionality for all of the GUI controls. The CImageView class supports the display of 24-bit RGB images. CSelectView extends CImageView by implementing functionality that allows user to drag selection boxes in the view area. COutputView extends CImageView by implementing a pop-up context menu in the view area. The CInputView and CTrainingView classes provide support for both selection boxes and pop-up menus. The CImageMatrix class implements functionality for various image processing operations such as rotation, scaling, low-pass filtering, and conversion from a 24-bit RGB image to the image format as illustrated in Figure 25. It encapsulates MFC class CImage to provide low-level imaging support. The source code for image rotation, scaling, and low-pass filtering are taken from Practical Algorithms for Image Analysis [10]. The CFaceMap class maps the path name of each training image with a CTrainingData object. Each CTrainingData object contains two CImageData objects, one to hold data for a young image, and the other for an aged image. The CFaceParams class stores various face parameters such as bounding box dimensions of detected features and in-plane face rotation angle. The CFeatureLocate class implements functionality that locates features on a face image (eyes, nose, mouth, and face), and determines the best-fit bounding box around each feature by using edge detection provided by CEdgeDetector. It invokes the face detection library [16] to find the center positions of the eyes and boundaries of the face, and uses the

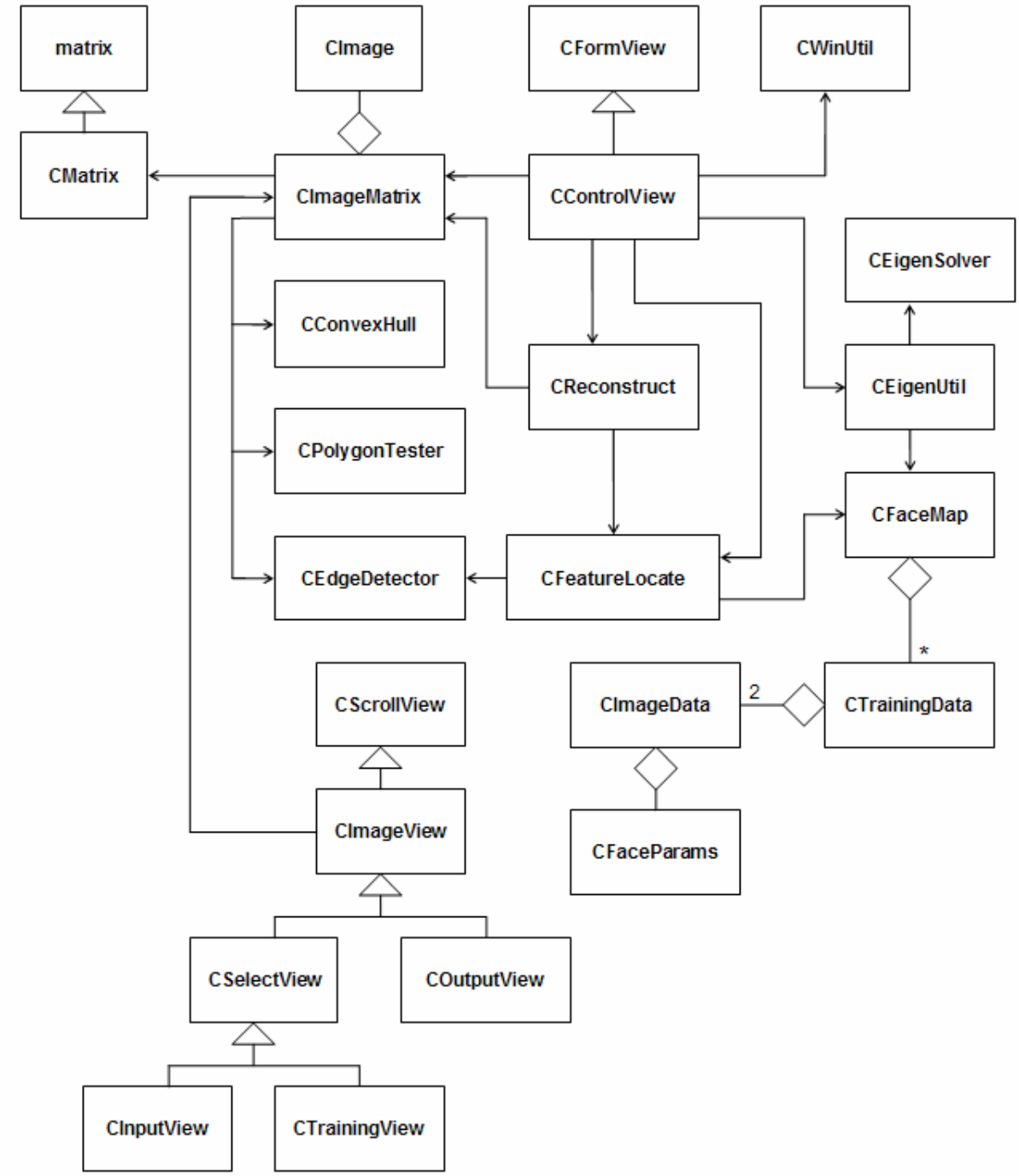


Figure 29: Feature-based program class design

dimension scheme in Figure 23 to determine the locations of the other features. The dimensions of the face features are then mapped in CFaceMap and stored in CImageData objects for each training image. The CEigenUtil class implements functionality that traverses a CFaceMap object, uses the in-plane rotation angle and bounding box dimensions stored in each CImageData object, and extracts the pixel values in bounding boxes from each training image. It then forms a set of training images for each feature, and uses each set to compute the principal components for a given feature. CEigenUtil uses CEigenSolver to compute the eigenvalues and eigenvectors from the covariance matrix of each feature. The CReconstruct class implements functionality to extract features from an input face image by using the CFeatureLocate class, performs feature age-progression by projecting each input feature onto a corresponding set of principal components, and reconstructs an overall aged face using the image processing functionality provided by CImageMatrix. The CPolygonTester class implements functionality to test whether a point is in a given polygon. This class is used to extract pixels from a convex hull region of a face image as illustrated in Figure 27. CMatrix and CWinUtil are discussed in Section 3.2.3. And CConvexHull and CEdgeDetector are discussed in Section 3.4.3.

4.5 Feature-Based Test Results

We use the same test images as in Section 3.3.2, but of different dimensions. This includes 300 female (young, aged) face pairs that are 240 by 300 pixels of various age ranges. We assign 200 of them as concatenated

training images as in Figure 10, and the rest as young concatenated input images as in Figure 11. We use the feature-based aging program to reconstruct the 100 input images without truncating any eigenvectors, and visually determine that 85 out of the 100 resultant images are reconstructed successfully, and 30 of the 85 successfully reconstructed images are adequately age-progressed. Doing the same for 150 male training images and 50 male input images also 240 by 300 pixels, we observe 35 out of the 50 resultant images are successfully reconstructed, and 15 of the 35 successfully reconstructed images are adequately age-progressed. There is a marked improvement in the number of successfully reconstructed and age-progressed images as compared to the results from Section 3.3.2. Specifically, we get a 100% and 50% improvement for female and male age progression results, respectively. This is attributed to the use of color images and increase in image dimensions, as discussed in Section 3.5.

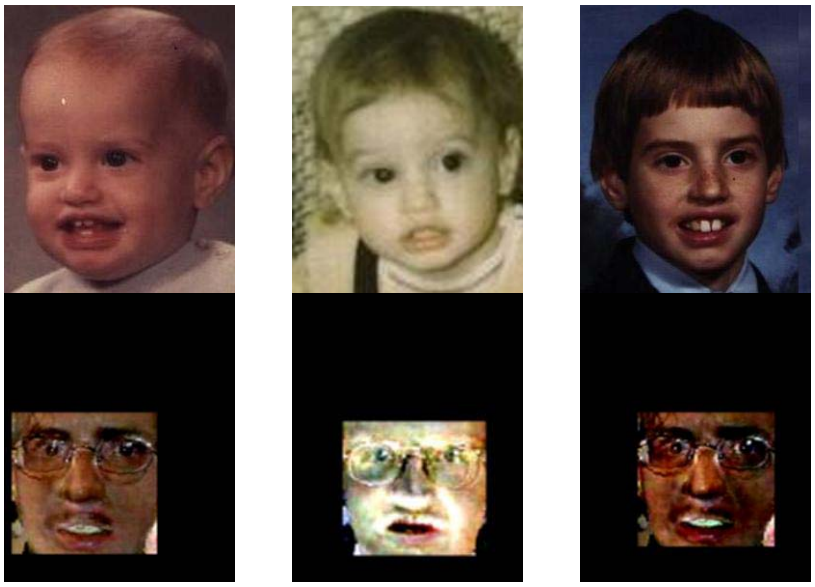
We attempt to improve on the feature-based age progression results by including more (young, aged) training image pairs for a given person. Due the limited number of (young, aged) pairs per person that can be retrieved from www.missingkids.com, we obtain images from a family album to obtain create more training images for a given person. The idea is to gather a number of images to represent four distinct age groups: baby, toddler, adolescent, and adult. We then produce a set of training images concatenating all possible pairs between images in the baby toddler, and adolescent group with images in the

adult group. Specifically, we use the baby, toddler, adolescent, and adult images as shown in Figure 30 to produce a total of (7 + 6 + 4) x 6 = 102 concatenated training images.

Figure 30: Training images for clustering test (permission from Chris Pollett)

The idea behind this test, as discussed in Section 3.5, is to have the projected input image be in close proximity to a cluster of projected training images with the desired age traits. We input a baby, toddler, and adolescent image and obtain corresponding reconstructed images as shown in Figure 31, with feature extraction disabled due to poor extraction results. The results demonstrate that the close clustering of training images causes the reconstructed

result to resemble a weighted averaging of all the adult training images. The averaging depends on the input image and where it lies in reference to the training images. We note that the reconstructed results of (5a) and (5c) are quite similar, which implies that the projection of the baby and adolescent input images onto the set of principal components has a smaller Euclidean distance value as compared to the projection of the toddler input image. Therefore, the PCA classifier regards the baby and adolescent input faces as being more similar.



(a) **(b)** **(c)**

Figure 31: Reconstructed results (bottom) of baby, toddler, and adolescent

We then add 100 (young, aged) image pairs into the training image set retrieved by the Python script *webcrawler.py* to observe how clustering performs with images from different people. We add a sufficient number of additional training images and hope that excessive skewing from the intended cluster of training images is prevented by an averaging effect. Figure 32 shows the

corresponding reconstructed results of the same input images using the extended training image set with feature extraction disabled. We see from Figure 32 that the addition of training images of different people diminishes the quality of the reconstruction such that the images no longer resemble the adult images in Figure 30. This is because as more training images are added, a reconstructed image is formed with a greater number of principal components due to the increase in dimensionality of the transformed coordinate system needed to represent the extra data. As a result, the reconstructed result also captures unwanted features from the added training images.

Figure 32: Reconstructed results with extended training set

4.6 Colorization Application and Results

To perform further testing of the feature-based aging program, we create training image pairs to consist of a gray and color image of the same person, as opposed to a young and aged image pair. By doing so, we attempt to convert the application from one that ages a color face image, to one that colorizes a grayscale face image by changing the training image data. Figure 33 shows two grayscale images and corresponding colorized images.

Figure 33: Example colorization results

For the colorization test, we use *webcrawler.py* to retrieve 300 young face images, and copy and convert 200 of them to grayscale images. Each grayscale and corresponding color image is concatenated to form a training image. Colorization is then attempted on the remaining 100 grayscale images. Results show that 90 of the 100 images are successfully reconstructed to reasonably resemble the input images in color. Visual inspection of the colorization results shows that the colorized images demonstrate consistent usage of color for skin tone and features. However, some of the results tend to exhibit regions of extreme white blending into areas of color as shown in Figure 34. Also, as

Figure 34: Colorized face with bright regions

previously mentioned, aside from color, the reconstructed faces do not look exactly like the input faces, as there are some slight variations (e.g. shape of features), therefore this may not be acceptable for a practical application.

Chapter 5

Conclusion

The test results from Section 4.5 demonstrate that PCA image reconstruction is highly sensitive to the inclusion of other training images, particularly those that are not in close proximity to existing projected training image clusters in multi-dimensional space. The results also show that it is difficult to manipulate the training images to obtain the desired clustering such that the projection of an input image is near a cluster of projected training images in the transformed coordinate system with the desired aged features. In addition, since image reconstruction in PCA involves a linear combination of all the principal components, the reconstructed result may capture features from training images that are not desired. One possible solution is to manually group images by face classes as described in Section 3.2.1 on eigenface recognition. The idea is to project an input image to obtain its weight vector, and determine the smallest Euclidean distance between the input weight vector and average weight vector for all face classes. Essentially, we classify the input image with respect to one of the groups of images (face class) representing a given person, and reconstruct the corresponding average weight vector. This has the effect of reconstructing a weighted average of the images belonging to the matched face class.

Another evident issue from the test results of Section 4.5 is program runtime. We traced the bottlenecks to two locations for the computation of eigenfaces. Specifically, the algorithm that solves for the eigenvalues and eigenvectors from a symmetric matrix runs in $O(n^3)$. This bottleneck may be reduced by truncating eigenvectors with negligible eigenvalues at the expense of reconstruction accuracy, and therefore is not a robust solution. The other significantly larger bottleneck occurs for the computation of principal components using equation (8), or $u_i = \sum_{j=1}^{M} v_{ij}\Phi_j = Av_i$, $for\ i = 1, \ldots, M$, where matrix $A = \{\Phi_1, \Phi_2, \ldots, \Phi_M\}$ and $\Phi_i = \Gamma_i - \Psi$ (input minus mean image). Since each column of A is an image stretched out to an N^2 by 1 vector (assuming $N \times N$ images), increasing the size of the images by multiplying the width or height by a constant factor also increases the size of A by the same factor. This observation points to the inefficient image encoding illustrated in Figure 24. Here, we essentially increase the size of an image by a factor of six. This increases A by the same factor, and in turn increases execution time by a factor of six, in computing a set of principal components. To improve runtime, we need an alternative image format that is more efficient in terms of the size of a matrix required to store the RGB image data, and also supports closed matrix operations.

The final issue that needs to be addressed is more accurate feature detection and location of the eyes, nose, and mouth. Currently, feature detection relies on the accurate location of the centers of the eyes, and on the assumption

that certain key face ratios do not vary greatly between images (e.g. ratio of distance between eyes and vertical distance between eyes and nose). This technique works poorly for faces that are skewed out-of-plane, or have face ratios that are not typical of the norm. A more robust solution would require a major undertaking in applying, for example, image segmentation or a customized neural network application. For image segmentation, the idea is to first convert images to HSI (hue, saturation, and intensity) color space. This color model decouples intensity from the color components. This allows color to be represented by hue and saturation values, where hue is a measure of the dominant color within the color spectrum, and saturation is a measure of strength or purity of a certain color. Image segmentation relies on the fact that human skin typically exhibits hue and saturation values that fall within relatively narrow bands. Faces can then be segmented by throwing out pixels that do not fall within these bands. Further image processing are performed, such as feature erosion / dilation, noise elimination, and region detection, to isolate the major features on a face that are not of skin tone (i.e. eyes and mouth). Region detection is then applied to label and identify the remaining processed features. For a neural network application, the training process would be similar to that of PCA training. However, coming up with a viable architecture (e.g. number of nodes per layer) requires a time-consuming trial and error process to determine a good balance between accuracy and generalization. In addition, given that images are typically of high dimensionality, there would be an inordinate number of weights that must be trained.

References

[1] Christopher M. Bishop. Neural Networks for Pattern Recognition. Oxford University Press, 1995.

[2] Serge Belongie and Jitendra Malik. Matching Shapes. Eighth IEEE International Conference on Computer Vision (July 2001).

[3] Richard O. Duda, Peter E. Hart, and David G. Stork. Pattern Classification. John Wiley & Sons, 2001.

[4] Rafael Gonzalez and Richard E. Woods. Digital Image Processing. Prentice Hall, 2002.

[5] R. Jonker and A. Volgenant. A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems. Computing 38, 325-340, 1987.

[6] Joseph O'Rourke. Computational Geometry. Cambridge University Press, 1998.

[7] Maria Petrou and Panagiota Bosdogianni. Image Processing: The Fundamentals. Wiley & Sons, 1999.

[8] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. Numerical Recipes in C. Cambridge University Press, 1988.

[9] Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Approach. Pearson Education, Inc., 2003.

[10] Michael Seul, Lawrence O'Gorman, and Michael J. Sammon. Practical Algorithms for Image Analysis: Description, Examples, and Code. Cambridge University Press, 2000.

[11] Matthew Turk and Alex Pentland. Eigenfaces for Recognition. Journal of Cognitive Neuroscience, Vol. 3, No. 1, 1991.

[12] Mario Giannini and Jim Keogh. Windows Programming: Programmer's Notebook. Prentice Hall, 2001.

Web References

[13] Lindsay I. Smith. A Tutorial on Principal Components Analysis. http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf, 2002.

[14] R. Jonker and A. Volgenant. Linear Assignment Source Code. http://www.magiclogic.com/assignment.html.

[15] Cambridge University Engineering Department Database of Faces. http://www.uk.research.att.com/facedatabase.html.

[16] Henry A. Rowley, Shumeet Baluja, and Takeo Kanade. Face Detection Library. http://vasc.ri.cmu.edu/NNFaceDetector/.

[17] Missing Children's Website. http://www.missingkids.com.

[18] Dmitri Pissarenko. Eigenface-Based Facial Recognition. http://dapissarenko.com/resources/2002_12_01_eigenfaces.pdf, 2002.

[19] Matrix TCL Lite Source Code. http://www.techsoftpl.com.