**CPEG 585 – COMPUTER VISION**

**HOMEWORK 8**

**Name:** Ivan Sangines Escrig

**ID#:** 968606

**Instructor:** Dr. Mahmood

**Date:** April 19, 2019

**TABLE OF CONTENT**

[**INTRODUCTION**](#_Toc446970371) 3

[**SCREENSHOTS:**](#_Toc446970373) 4

[**SOURCE CODE:**](#_Toc446970374) 7

[**CONCLUSION:**](#_Toc446970375) 25

**INTRODUCTION**

The purpose of this assignment is to get familiar and understand better how to use PCA for face recognition. In order to come up with the solution for this assignment, I will use different classes to create objects with the corresponding information needed for the PCA. In order to do all the computations, I will be using the properties of the corresponding objects (Eigen Vectors, Eigen Values, adjusted matrix etc.).

Classes used in this assignment:

* MyImage:
  + Contains all the img properties (name, width, height etc.).
  + ImgVector (initial pixels of the img).
  + ImgVectorAdj (Mean adjusted vector, subtracting mean img to the actual img).
  + FSV array (Face space vector, will contain the coefficients used in reconstruction).
  + Euclidean Error
* EvEvec:
  + EigenValues (One eigen value per img).
  + EigenVectors (array containing Eigen Vector corresponding to the current img).
  + Method to sort/compare EvEvec objects by Eigen Value.
* EigenFace:
  + EigenFace (array containing the Eigen Faces vector).
  + EigenValue.

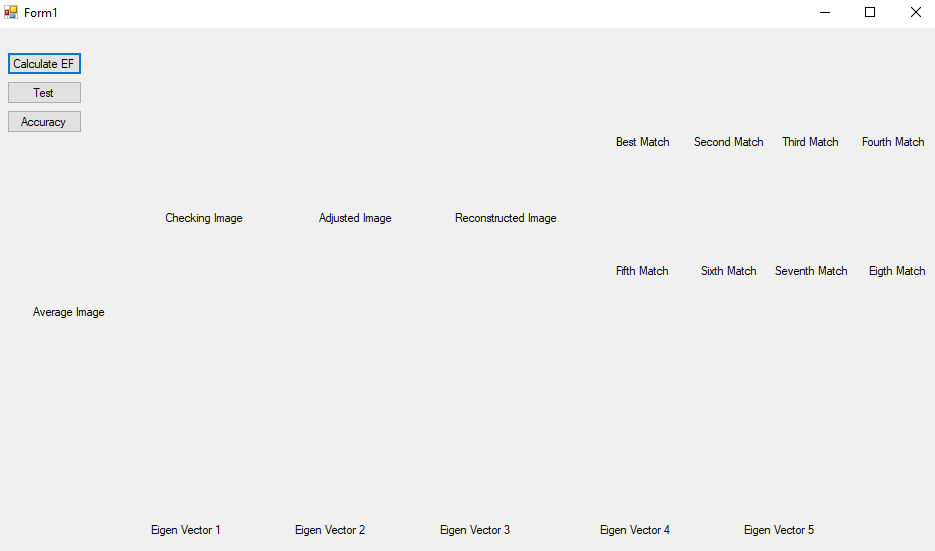
In order to apply the PCA algorithm, I will first get the images from the file and create a MyImage object for each img (store pixel by pixel into the ImgVector). I will create a List containing all the training images. Once I have a list storing all the images, I will compute the mean img vector by adding the ith pixel from each img and dividing it by total images in the training file. Once the average img is computed, I will create a new list with adjusted images vectors (subtracting mean img). The next step will be to create a matrix with the adjusted images to further compute the covariance matrix.

The next step will consist in computing the EigenValues and EigenVector (creating EvEvec objects) for each vector in the covariance matrix and store them into a new list. Then, I will sort the list by Eigenvalue, and create a matrix with the top n (input from user) EigenVectors.

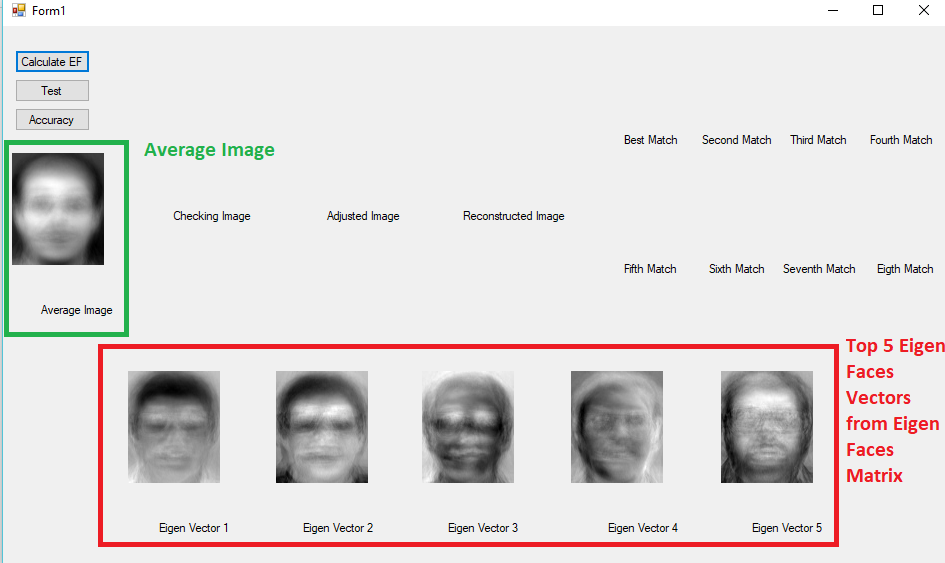
The final steps, will consist in creating a list of EigenFace objects using the EigenVector matrix and image adjusted matrix. After computing all the EigenFaces requested, we will create a projection img vector, using the adjusted img matrix (transposed) and the EigenFaces matrix.

**SCREEN SHOTS:**

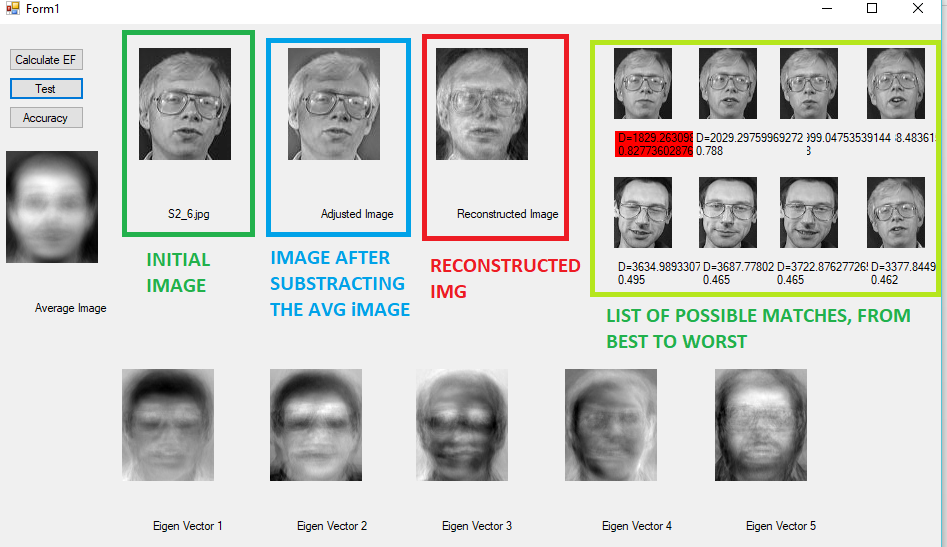
Initial Form, the first step will be to compute the Eigen Faces matrix to further compute the projected images in the dimensinality reduced space.



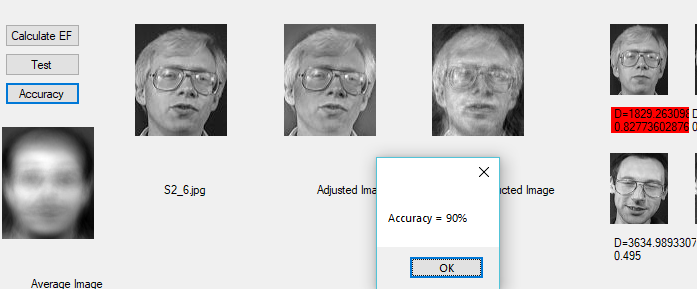
After computing the PCA algorithm, we will first obtain the Average Image, and finally we obtain the Eigen Faces matrix (I am just showing the first five Eigen Vectors).

****

In the following screenshot, I am trying to find the best match for a specific image. I will first pick a testing img and compute the corresponding projected img. Then I will use Eucledean or Correlation to find the match with less error.

****

Computing the accuracy for the best match:



**SOURCE CODE:**

**MyEnums:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp1

{

class MyEnums

{

public enum ImgFormat : int

{

EightBit,

TwentyFourBit

}

public enum ImgComparison : int

{

CORRELATION,

EUCLIDEAN

}

}

}

**MyImage:**

namespace WindowsFormsApp1

{

class MyImage : IComparable, ICloneable

{

public int Width { get; set; }

public int Height { get; set; }

public Bitmap BmpImg { get; set; }

public string Id { get; set; } // id of image

public string FileName { get; set; } // full path name

public string FileNameShort { get; set; } // short file name for image

public double ImgMean { get; set; }

public double[] ImgVector;// linearized pixel values

public double[] ImgVectorAdjM;// linearized pixel values minus mean image

public double[] FSV;// Face space vector, projection onto reduced Dimension

ImgComparison imgCompareMode;

public ImgComparison ImgCompareMode { get; set; }

public double CorrError { get; set; }

public double EuclideanError { get; set; } // L2 Norm

public MyImage()

{

}

public MyImage(int width, int height, string id)

{

this.Id = id;

this.ImgVector = new double[width \* height];

}

public MyImage(string fname, string id, int width, int height, ImgFormat imf, ImgComparison imc)

{

FileInfo finfo = new FileInfo(fname);

string dirName = finfo.Directory.Name;

this.FileNameShort = finfo.Name;

this.Id = id;

this.FileName = fname;

imgCompareMode = imc;

ReadPic(imf, width, height); // read the picture into an 1-D array

FindImageMean();

this.ImgVectorAdjM = new double[width \* height];

this.FSV = new double[width \* height];

}

//-----------READING AND CONVERTING IMG TO GREY SCALE-----------------

private void ReadPic(ImgFormat imf, int width, int height)

{

try

{

Bitmap b = new Bitmap(this.FileName);

this.BmpImg = new Bitmap(b, new Size(width, height));

ImgVector = new double[width \* height];

int k = 0;

int r1a, g1a, b1a, gray;

Color c1;

for (int i = 0; i < height; i++)

{

for (int j = 0; j < width; j++)

{

c1 = b.GetPixel(j, i);

r1a = c1.R;

g1a = c1.G;

b1a = c1.B;

if (r1a != b1a)

gray = (int)(.299 \* r1a + .587 \* g1a + .114 \* b1a); //CONVERTING PIXELS TO GREY

else

gray = b1a;

ImgVector[k++] = gray;

}

}

}

catch (Exception ex)

{

throw new Exception(ex.Message + "ImageArray Creation Error");

}

}

//---------------------------------------------------------------------

//-------------------FindImageMean---------------------------

void FindImageMean()

{

double imgSum = 0;

for (int i = 0; i < this.ImgVector.Length; i++)

imgSum = imgSum + this.ImgVector[i];

this.ImgMean = imgSum / this.ImgVector.Length;

}

//------------------------------------------------------------------

public object Clone()

{

MyImage clone = (MyImage)this.MemberwiseClone();

if (this.FSV != null)

clone.FSV = (double[])(this.FSV.Clone());

if (this.ImgVector != null)

clone.ImgVector = (double[])(this.ImgVector.Clone());

if (this.ImgVectorAdjM != null)

clone.ImgVectorAdjM = (double[])(this.ImgVectorAdjM.Clone());

return clone;

}

public int CompareTo(object obj)

{

MyImage im = (MyImage)obj;

if (imgCompareMode == ImgComparison.CORRELATION)

return im.CorrError.CompareTo(this.CorrError); // index 0 is best

else

return im.EuclideanError.CompareTo(this.EuclideanError);

}

}

}

**EvEvec**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp1

{

class EvEvec:ICloneable, IComparable

{

public double EigenValue; // Eigen value

public double[] EigenVec; // Eigen Vector Array

public int size; // Size of Eigen Vector array

public EvEvec()

{

}

public EvEvec(double Ev, double[] Evc, int sz)

{

EigenVec = new double[sz];

EigenValue = Ev;

size = sz;

for (int i = 0; i < sz; i++)

EigenVec[i] = Evc[i];

// EVecs are already normalized i.e., magnitude of 1

}

public int CompareTo(Object obj) // for sorting

{

EvEvec evv = (EvEvec)obj;

return evv.EigenValue.CompareTo(this.EigenValue); // highest to lowest sorting by Eigen value

}

public object Clone() // for making a copy of the EvEvec object

{

EvEvec clone = new EvEvec();

clone.EigenValue = this.EigenValue;

if (this.EigenVec != null)

clone.EigenVec = (double[])this.EigenVec.Clone();

clone.size = this.size;

return clone;

}

}

}

**EigenFaces:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp1

{

//An EigenFace is one of the vectors after we reduce the dimensionality

class EigenFace:ICloneable

{

public double[] EF;

public int size;

public double[] Xvar2;

public double EigenValue;

public EigenFace()

{

}

public EigenFace(int sz)

{

EF = new double[sz];

Xvar2 = new Double[sz];

size = sz;

}

public object Clone() // for in memory copy

{

EigenFace copy = new EigenFace();

if (this.EF != null)

copy.EF = (double[])this.EF.Clone();

copy.EigenValue = this.EigenValue;

copy.size = this.size;

if (this.Xvar2 != null)

copy.Xvar2 = (double[])this.Xvar2.Clone();

return copy;

}

}

}

**FaceRecognition:**

class FaceRecogByEF

{

public List<MyImage> ImageList; // list of stored images

public MyImage AvgImage = new MyImage();

public Matrix Cov = null; // Covariance Matrix

public Matrix I = null; // matrix of mean adjusted images

public double[] Evals; // Eigen Values

public List<EvEvec> EVList; // Eigen values and vectors

public List<EigenFace> EigenFaceList = new List<EigenFace>(); // list of Eigen faces

public Matrix EigenFaceMatrix;

public MyImage unkIm;

private int imageWidth;

private int imageHeight;

public int imageNumPixels;

private int NumEigenFaces; // significant number of EigenFaces

public FaceRecogByEF(int imWidth, int imHeight, int imPixels, int M)

{

imageWidth = imWidth;

imageHeight = imHeight;

imageNumPixels = imPixels;

this.NumEigenFaces = M;

}

//----------------------ComputeEFs()----------------

public void ComputeEFs(ImgComparison imgComparison, string imagesFolder)

//Euclidean or Corr

{

ImageList = new List<MyImage>();

EVList = new List<EvEvec>();

// Scan Stored Image directory and add every jpg to

try //Storing each image in ImageList

{

DirectoryInfo dirInfo = new DirectoryInfo(imagesFolder);

if (!dirInfo.Exists)

throw new DirectoryNotFoundException(imagesFolder + "folder does not exist,");

foreach (FileInfo nextFile in dirInfo.GetFiles())

{

if (nextFile.Extension.ToUpper() == ".JPG")

this.ImageList.Add(new MyImage(nextFile.FullName, "II", imageWidth, imageHeight, ImgFormat.TwentyFourBit, imgComparison)); // Euclidean or Corr

else

if (nextFile.Extension.ToUpper() == ".GIF")

this.ImageList.Add(new MyImage(nextFile.FullName, "II",imageWidth, imageHeight, ImgFormat.EightBit, imgComparison));

}

}

catch (Exception ex)

{

throw new Exception(ex.Message + "Error creating the ImageList..");

}

AdjustAllImages(); // subtract mean image from each image

ComputeCovMatrix();

ComputeEigenValuesEigenVectors();

ComputeEigenFaces();

ComputeKnownFaceSpace(); // projection of images onto reduced dim

}

//--------------------FindAvgImageAndAdjustImages()------------

void AdjustAllImages()

{ //Computing average img (sum each pixel in same position, one per image, and div by total images)

try

{

// find average image

double[] sum = new double[imageWidth \* imageHeight];

//Initializing to 0 each index

for (int i = 0; i < sum.Length; i++)

sum[i] = 0;

//Sum of all pixels in same position (one per img)

for (int i = 0; i < sum.Length; i++)

{

foreach (MyImage img in ImageList)

{

sum[i] += img.ImgVector[i];

}

}

this.AvgImage.ImgVector = new double[sum.Length];

//Computing the mean, dividing the sum by total images

for (int i = 0; i < sum.Length; i++)

this.AvgImage.ImgVector[i] = sum[i] / ImageList.Count;

// subtract average image from each image

foreach (MyImage mimg in ImageList)

{

SubtractAvgImage(mimg); //Will call substractImg for each adjusted img in ImageList

}

int numImages = ImageList.Count;

I = new Matrix(imageWidth \* imageHeight, numImages); //Image Matrix with all adjusted images (i.e. 1000x200 = 1000pixels, 200img)

int count = 0;

// copy mean adjusted images into I matrix

foreach (MyImage mimg in ImageList)

{

for (int j = 0; j < mimg.ImgVectorAdjM.Length; j++)

I[j, count] = mimg.ImgVectorAdjM[j];

count++;

}

}

catch (Exception)

{

throw;

}

}

//-------------------------------------------------------------

//----------------SubtractAvgImage()---------------------

public void SubtractAvgImage(MyImage img1)

{

// subtract average image from given image

for (int i = 0; i < img1.ImgVector.Length; i++) //goes through length beause we have vectors images #pixes x 1

{

img1.ImgVectorAdjM[i] = img1.ImgVector[i] - this.AvgImage.ImgVector[i];

}

}

//-------------------------------------------------

//---------------------ComputeCovMatrix--------------------

private void ComputeCovMatrix() // covariance matrix

{

Cov = (Matrix)(((I.Transpose()).Multiply(I)));

}

//-------------------------------------------------

//---------------------ComputeEigenValues-------------------

private void ComputeEigenValuesEigenVectors()

{

int i, j;

int n = ImageList.Count; //Total number of images

Evals = new double[n]; //One EigenVal per Img

IMatrix Evecs = new Matrix(n, n);

IEigenvalueDecomposition eigen = Cov.GetEigenvalueDecomposition();

Evecs = eigen.EigenvectorMatrix;

Evals = eigen.RealEigenvalues;

//---Copy the Eigen values and vectors in EvEvec objects

//---for easier sorting by Eigen values.

double[] evcTemp = new double[n];

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

evcTemp[j] = Evecs[j, i];

EVList.Add(new EvEvec(Evals[i], evcTemp, n));

}

EVList.Sort(); // sorts, highest Eigen value in pos. 0

}

//----------------------------------------------------------

//-----------ComputeEigenFaces()-------------------------

private void ComputeEigenFaces()

{

int numEFs = 0;

if (ImageList.Count < NumEigenFaces)

numEFs = ImageList.Count;

else

numEFs = NumEigenFaces;

EigenFaceMatrix = new Matrix(imageWidth \* imageHeight, numEFs);

// copy EigenVectors into a Matrix

Matrix EV = new Matrix(ImageList.Count, NumEigenFaces); //CREATING A MATRIX WITH JUST THE TOP EIGENVECTORS

//The number of eigenvector we will use is prompted in the constructor

for (int i = 0; i < NumEigenFaces; i++)

{

for (int j = 0; j < ImageList.Count; j++)

EV[j, i] = EVList[i].EigenVec[j]; //Filling up the matrix with top EIGENVECTORS

}

EigenFaceMatrix = (Matrix)(I.Multiply(EV)); //Creating EigenFace Matrix, I \* EV

//normalize EigenFace(it is an eigen vector of orig.covar matrix)

for (int j = 0; j < NumEigenFaces; j++)

{

double rsum = 0;

for (int i = 0; i < imageNumPixels; i++)

{

rsum += EigenFaceMatrix[i, j] \* EigenFaceMatrix[i, j];

}

for (int i = 0; i < imageNumPixels; i++)

EigenFaceMatrix[i, j] = EigenFaceMatrix[i, j] / Math.Sqrt(rsum);

}

// Copy Eigen Faces to a List for easier display later

for (int i = 0; i < NumEigenFaces; i++)

{

EigenFace ef = new EigenFace(imageWidth \* imageHeight);

for (int j = 0; j < imageWidth \* imageHeight; j++)

ef.EF[j] = EigenFaceMatrix[j, i];

EigenFaceList.Add(ef);

}

}

//---------------------------------------------------------

//----------ComputeKnownFaceSpace()----------------

private void ComputeKnownFaceSpace()

{

Matrix projection = (Matrix)(I.Transpose().Multiply(EigenFaceMatrix));

for (int i = 0; i < ImageList.Count; i++)

{

for (int j = 0; j < NumEigenFaces; j++)

ImageList[i].FSV[j] = projection[i, j];

}

}

//-------------------------------------------------

//----------ComputeFaceSpace()---------------------

public void ComputeFaceSpace(MyImage im)

{

int i, j;

double rsum;

for (i = 0; i < EigenFaceList.Count; i++)

{

rsum = 0.0;

for (j = 0; j < imageNumPixels; j++)

{

EigenFace ef = (EigenFace)EigenFaceList[i];

rsum = rsum + im.ImgVectorAdjM[j] \* ef.EF[j];

}

im.FSV[i] = rsum;

}

}

//-------------------------------------------------

//-------------------NormalizeEigenFaces()-----------

void NormalizeEigenFaces()

{

// normalize the EigenFace to 0-255 range

foreach (EigenFace ef in EigenFaceList)

{

double max1 = (from n in ef.EF select n).Max();

double min = (from n in ef.EF select n).Min();

double diff = max1 - min;

for (int i = 0; i < ef.EF.Length; i++)

{

ef.EF[i] = ef.EF[i] - min;

ef.EF[i] = ef.EF[i] / diff \* 255;

if (ef.EF[i] < 0)

ef.EF[i] = 0;

if (ef.EF[i] > 255)

ef.EF[i] = 255;

}

}

}

//---------------------------------------------------

//--------------GetReconstructedFaceSpaceImage(MyImage img)------------

public MyImage GetMatchedAndReconstructedImages(MyImage inputImg,ref double selfReconstError, ref MatchResult[] bestMatches)

{

// assumes data is in Imagevector

MyImage recImage = new MyImage();

recImage.ImgVectorAdjM = new double[inputImg.ImgVectorAdjM.Length];

// subtract mean image from input image

SubtractAvgImage(inputImg);

//-------FSV for input image-----------------

ComputeFaceSpace(inputImg);

double[] fsvData = inputImg.FSV;

// Reconstruct the input image

double[] recData = new double[inputImg.ImgVectorAdjM.Length];

for (int j = 0; j < inputImg.ImgVectorAdjM.Length; j++)

{

recData[j] = 0;

for (int i = 0; i < NumEigenFaces; i++)

{

recData[j] += fsvData[i] \* ((EigenFace)EigenFaceList[i]).EF[j];

}

}

// normalize the reconstructed image to 255 range

double max1 = (from n in recData select n).Max();

double min = (from n in recData select n).Min();

double diff = max1 - min;

for (int i = 0; i < inputImg.ImgVectorAdjM.Length; i++)

{

recData[i] = recData[i] - min;

recData[i] = (recData[i] / diff) \* 255;

recData[i] = recData[i];

if (recData[i] < 0)

recData[i] = 0;

if (recData[i] > 255)

recData[i] = 255;

}

// add mean image

for (int i = 0; i < recData.Length; i++)

recImage.ImgVectorAdjM[i] = recData[i] + this.AvgImage.ImgVector[i];

// readjust the reconstructed image to 0-255 range

max1 = (from n in recImage.ImgVectorAdjM select n).Max();

min = (from n in recImage.ImgVectorAdjM select n).Min();

diff = max1 - min;

for (int i = 0; i < recImage.ImgVectorAdjM.Length; i++)

{

recImage.ImgVectorAdjM[i] = recImage.ImgVectorAdjM[i] - min;

recImage.ImgVectorAdjM[i] = (recImage.ImgVectorAdjM[i] / diff) \* 255;

if (recImage.ImgVectorAdjM[i] < 0)

recImage.ImgVectorAdjM[i] = 0;

if (recImage.ImgVectorAdjM[i] > 255)

recImage.ImgVectorAdjM[i] = 255;

}

selfReconstError = 0;

for (int i = 0; i < inputImg.ImgVectorAdjM.Length; i++)

selfReconstError = selfReconstError + (inputImg.ImgVector[i] -

recImage.ImgVectorAdjM[i]) \* (inputImg.ImgVector[i] -

recImage.ImgVectorAdjM[i]);

selfReconstError = Math.Sqrt(selfReconstError);

//-----------find best match----------------------------

MatchResult[] MR = new MatchResult[ImageList.Count];

for (int i = 0; i < ImageList.Count; i++)

{

MR[i] = new MatchResult(i, 0, (MyImage)((MyImage)ImageList[i]).Clone());

double dist = 0;

for (int j = 0; j < NumEigenFaces; j++)

dist += ((MR[i].mImage.FSV[j] - inputImg.FSV[j]) \* (MR[i].mImage.FSV[j] - inputImg.FSV[j]));

MR[i].EucledianDist = Math.Sqrt(dist);

}

//--------------find correlation--------------------

double[] corr = FindCorrelation(inputImg);

for (int i = 0; i < ImageList.Count; i++)

MR[i].Correlation = corr[i];

Array.Sort(MR);

bestMatches = MR;

return recImage;

}

public double[] FindCorrelation(MyImage unkIm)

{

int i, j, k;

double rsum = 0; double avgunk;

double[] avgi = new double[ImageList.Count];

//---------Calculates the unknown face space average---------//

//------------------ Y' ---------------------------------//

for (k = 0; k < EigenFaceList.Count; k++)

{

rsum = rsum + unkIm.FSV[k];

}

avgunk = rsum / EigenFaceList.Count;

//---------Calculates the known face space average---------//

//------------------ X' ---------------------------------//

for (i = 0; i < ImageList.Count; i++)

{

rsum = 0.0;

MyImage im = (MyImage)ImageList[i];

for (k = 0; k < EigenFaceList.Count; k++)//review

{

rsum = rsum + im.FSV[k];

}

avgi[i] = rsum / EigenFaceList.Count;

}

//----Calculate the Numerator of the Correlation Equation----//

double[] rtop = new double[ImageList.Count];

for (i = 0; i < ImageList.Count; i++)

{

rtop[i] = 0.0;

for (j = 0; j < EigenFaceList.Count; j++)

{

MyImage im = (MyImage)ImageList[i];

rtop[i] = rtop[i] + (im.FSV[j] - avgi[i]) \* (unkIm.FSV[j] - avgunk);

}

}

//---Calculate the Denominator of the Correlation Equation----//

double[] rbot1 = new Double[ImageList.Count];

double[] rbot2 = new Double[ImageList.Count];

for (i = 0; i < ImageList.Count; i++)

{

rbot1[i] = 0.0; rbot2[i] = 0.0;

for (j = 0; j < EigenFaceList.Count; j++)

{

MyImage im = (MyImage)ImageList[i];

rbot1[i] = rbot1[i] + (im.FSV[j] - avgi[i]) \* (im.FSV[j] - avgi[i]);

rbot2[i] = rbot2[i] + (unkIm.FSV[j] - avgunk) \* (unkIm.FSV[j] - avgunk);

}

}

//----Calculate the final Correlation Equation----//

double[] corr = new double[ImageList.Count];

for (i = 0; i < ImageList.Count; i++)

corr[i] = rtop[i] / Math.Sqrt(rbot1[i] \* rbot2[i]);

return corr;

}

//-------------------------------------------------

//-----------Finds the LSE ------------------------

public void FindLSE()

{

// ResultVec = new Double[ImageList.Count];

for (int i = 0; i < ImageList.Count; i++)

{

double rsum = 0.0;

MyImage im = (MyImage)ImageList[i];

for (int k = 0; k < EigenFaceList.Count; k++)//review

{

rsum = rsum + (im.FSV[k] - unkIm.FSV[k]) \* (im.FSV[k] - unkIm.FSV[k]);

}

}

}

//-------------------------------------------------

}

}

**MatResults:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using static WindowsFormsApp1.MyEnums;

namespace WindowsFormsApp1

{

class MatchResult:ICloneable, IComparable

{

public double Correlation;

public double EucledianDist;

public MyImage mImage;

public int ImageNum;

public string ImageName;

public MatchResult(int Imnum, double err, MyImage img)

{

ImageNum = Imnum;

if (img.FileNameShort.Length > 10)

ImageName = img.FileNameShort.Substring(img.FileNameShort.Length - 10,

9);

else

ImageName = img.FileNameShort;

mImage = img;

if (img.ImgCompareMode == ImgComparison.CORRELATION)

Correlation = err;

else

EucledianDist = err;

}

public int CompareTo(Object rhs) // for sorting

{

MatchResult mr = (MatchResult)rhs;

if (mImage.ImgCompareMode == ImgComparison.CORRELATION)

return mr.Correlation.CompareTo(this.Correlation); // high corr is best

else

return this.EucledianDist.CompareTo(mr.EucledianDist); // low dist isbest

}

public object Clone()

{

MatchResult clone = new MatchResult(0, 0, null);

clone.Correlation = this.Correlation;

clone.mImage = (MyImage)this.mImage.Clone();

return clone;

}

}

}

**Main:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using static WindowsFormsApp1.MyEnums;

namespace WindowsFormsApp1

{

public partial class Form1 : Form

{

int NumberOfEigenFaces = 100; // 100 most significant EigenVctors

int width = 92;

int height = 112;

FaceRecogByEF ef = null;

public Form1()

{

InitializeComponent();

}

private void btn\_EF\_Click(object sender, EventArgs e)

{

ef = new FaceRecogByEF(width, height, width \* height, NumberOfEigenFaces);

ef.ComputeEFs(ImgComparison.CORRELATION, @"C:\Users\ivans\_000\Desktop\MASTER\Spring2019\Computer Vision\Assignment8\_ComputerVision\_SANGINES\ATTFaceDataSet\Training");

// show average image

double[] avgData = new double[ef.AvgImage.ImgVector.Length];

for (int i = 0; i < avgData.Length; i++)

avgData[i] = ef.AvgImage.ImgVector[i];

NormalizeDataAndShowFace(avgData, width, height, picAvg);

// show top five Eigen Faces

double[] eData0 = new double[((EigenFace)ef.EigenFaceList[0]).EF.Length];

for (int i = 0; i < eData0.Length; i++)

eData0[i] = ((EigenFace)ef.EigenFaceList[0]).EF[i];

NormalizeDataAndShowFace(eData0, width, height, picEF1);

double[] eData1 = new double[((EigenFace)ef.EigenFaceList[1]).EF.Length];

for (int i = 0; i < eData1.Length; i++)

eData1[i] = ((EigenFace)ef.EigenFaceList[1]).EF[i];//+ef.AvgImg.ImgVector[i]);

NormalizeDataAndShowFace(eData1, width, height, picEF2);

double[] eData2 = new double[((EigenFace)ef.EigenFaceList[2]).EF.Length];

for (int i = 0; i < eData2.Length; i++)

eData2[i] = ((EigenFace)ef.EigenFaceList[2]).EF[i];//+ef.AvgImg.ImgVector[i]);

NormalizeDataAndShowFace(eData2, width, height, picEF3);

double[] eData3 = new double[((EigenFace)ef.EigenFaceList[3]).EF.Length];

for (int i = 0; i < eData3.Length; i++)

eData3[i] = ((EigenFace)ef.EigenFaceList[3]).EF[i];//+ef.AvgImg.ImgVector[i]);

NormalizeDataAndShowFace(eData3, width, height, picEF4);

double[] eData4 = new double[((EigenFace)ef.EigenFaceList[4]).EF.Length];

for (int i = 0; i < eData3.Length; i++)

eData4[i] = ((EigenFace)ef.EigenFaceList[4]).EF[i];//+ef.AvgImg.ImgVector[i]);

NormalizeDataAndShowFace(eData4, width, height, picEF5);

}

private void btn\_Test\_Click(object sender, EventArgs e)

{

OpenFileDialog ofd = new OpenFileDialog();

if (ofd.ShowDialog() == DialogResult.OK)

{

FileInfo fi = new FileInfo(ofd.FileName);

Bitmap bmp = new Bitmap(ofd.FileName);

bmp = new Bitmap(bmp, new Size(width, height));

MyImage mimg = null;

if (fi.Extension.ToUpper() == ".GIF")

mimg = new MyImage(ofd.FileName, fi.Name, width, height, ImgFormat.EightBit, ImgComparison.CORRELATION);

if (fi.Extension.ToUpper() == ".JPG")

mimg = new MyImage(ofd.FileName, fi.Name, width, height, ImgFormat.TwentyFourBit, ImgComparison.CORRELATION);

NormalizeDataAndShowFace(mimg.ImgVector, width, height, pic0);

lblCheck.Text = mimg.FileNameShort;

ef.SubtractAvgImage(mimg); // mean adjusted image

NormalizeDataAndShowFace(mimg.ImgVectorAdjM, width, height, picAdjust);

double selfReconsError = 0;

MatchResult[] bestMatches = new MatchResult[ef.ImageList.Count];

MyImage reconImg = ef.GetMatchedAndReconstructedImages(mimg, ref selfReconsError,ref bestMatches);

NormalizeDataAndShowFace(reconImg.ImgVectorAdjM, width, height, picRecons);

//lblReconstructedError.Text = selfReconsError.ToString() + ":" + bestMatches[0].ImageName.ToString();

if (bestMatches[0].EucledianDist < 50)

lblBest.BackColor = System.Drawing.Color.Green;

else

lblBest.BackColor = System.Drawing.Color.Red;

NormalizeDataAndShowFace(((MyImage)ef.ImageList[bestMatches[0].ImageNum]).ImgVector, width, height, picBest1);

lblBest.Text = "D=" + bestMatches[0].EucledianDist.ToString() + "\n" + bestMatches[0].Correlation.ToString();//.Substring(0, 5);

NormalizeDataAndShowFace(((MyImage)ef.ImageList[bestMatches[1].ImageNum]).ImgVector,width, height, picBest2);

lblBest2.Text = "D=" + bestMatches[1].EucledianDist.ToString() + "\n" + bestMatches[1].Correlation.ToString().Substring(0, 5);

NormalizeDataAndShowFace(((MyImage)ef.ImageList[bestMatches[2].ImageNum]).ImgVector,width, height, picBest3);

lblBest3.Text = "D=" + bestMatches[2].EucledianDist.ToString() + "\n" + bestMatches[2].Correlation.ToString().Substring(0, 5);

NormalizeDataAndShowFace(((MyImage)ef.ImageList[bestMatches[3].ImageNum]).ImgVector,width, height, picBest4);

lblBest4.Text = "D=" + bestMatches[3].EucledianDist.ToString() + "\n" + bestMatches[3].Correlation.ToString().Substring(0, 5);

NormalizeDataAndShowFace(((MyImage)ef.ImageList[bestMatches[4].ImageNum]).ImgVector,width, height, picBest5);

lblBest5.Text = "D=" + bestMatches[4].EucledianDist.ToString() +"\n" + bestMatches[4].Correlation.ToString().Substring(0, 5);

NormalizeDataAndShowFace(((MyImage)ef.ImageList[bestMatches[5].ImageNum]).ImgVector,width, height, picBest6);

lblBest6.Text = "D=" + bestMatches[5].EucledianDist.ToString() +"\n" + bestMatches[5].Correlation.ToString().Substring(0, 5); NormalizeDataAndShowFace(((MyImage)ef.ImageList[bestMatches[6].ImageNum]).ImgVector,width, height, picBest7);

lblBest7.Text = "D=" + bestMatches[6].EucledianDist.ToString() + "\n" + bestMatches[6].Correlation.ToString().Substring(0, 5);

NormalizeDataAndShowFace(((MyImage)ef.ImageList[bestMatches[7].ImageNum]).ImgVector,width, height, picBest8);

lblBest8.Text = "D=" + bestMatches[7].EucledianDist.ToString() + "\n" + bestMatches[7].Correlation.ToString().Substring(0, 5);

}

}

private void btn\_Accuracy\_Click(object sender, EventArgs e)

{

try

{ // loop through test images folder to see if the

// test image matches correctly to known images

string testImagesFolder = @"C:\Users\ivans\_000\Desktop\MASTER\Spring2019\Computer Vision\Assignment8\_ComputerVision\_SANGINES\ATTFaceDataSet\Testing";

DirectoryInfo dirInfo = new DirectoryInfo(testImagesFolder);

if (!dirInfo.Exists)

throw new DirectoryNotFoundException(testImagesFolder +" folder does not exist,");

int accuracyCount = 0;

int count = 0;

foreach (FileInfo nextFile in dirInfo.GetFiles())

{

if (nextFile.Extension.ToUpper() == ".JPG")

{

MyImage mimg = new MyImage(nextFile.FullName, nextFile.Name,width,height, ImgFormat.TwentyFourBit, ImgComparison.CORRELATION);

//ef.SubtractAvgImage(mimg);

MatchResult[] bestMatches = new MatchResult[ef.ImageList.Count];

double reconstError = 0;

MyImage reconImg = ef.GetMatchedAndReconstructedImages(mimg, ref reconstError, ref bestMatches);

if (bestMatches[0].ImageName.ToString().Substring(0, 3) == mimg.FileNameShort.Substring(0, 3))

accuracyCount++;

else

Console.WriteLine(mimg.FileNameShort + ":" + bestMatches[0].ImageName);

count++;

}

}

MessageBox.Show("Accuracy = " + ((double)accuracyCount / count \* 100).ToString() + "%");

}

catch (Exception ex)

{

throw new Exception(ex.Message + "Error creating the ImageList..");

}

}

Bitmap GetBitmapFromByteArray(byte[] bdata)

{

Bitmap bmp = new Bitmap(width, height);

int k = 0;

for (int i = 0; i < height; i++)

{

for (int j = 0; j < width; j++)

{

bmp.SetPixel(j, i, Color.FromArgb(bdata[k], bdata[k], bdata[k]));

k++;

}

}

return bmp;

}

void NormalizeDataAndShowFace(double[] imData, int width, int height,PictureBox pb)

{

// normalize the reconstructed image to 0-255 range

double max1 = (from n in imData select n).Max();

double min = (from n in imData select n).Min();

double diff = max1 - min;

for (int i = 0; i < width \* height; i++)

{

imData[i] = imData[i] - min;

imData[i] = imData[i] / diff \* 255;

if (imData[i] < 0)

imData[i] = 0;

if (imData[i] > 255)

imData[i] = 255;

}

Bitmap bmp = new Bitmap(width, height);

int k = 0;

for (int i = 0; i < height; i++)

{

for (int j = 0; j < width; j++)

{

bmp.SetPixel(j, i, Color.FromArgb((int)imData[k], (int)imData[k],

(int)imData[k]));

k++;

}

}

pb.Image = bmp;

}

}

}

**Conclusion:**

After concluding this assignment, I have to say I understand better all the PCA steps. This assignment helped me to understand the concepts explained in class as well as translating them to code.

This assignment also helped me to realize how OOP architecture can make most of the problems easier by creating lists of objects with needed properties. It makes the code easier to read and more organized.