# **Código Documentado**

***Programa Principal***

//Detección de rostros aplicado en un sistema de seguridad y en la toma automatica de lista de asistentes

//Usando EmguCV cross platform .Net wrapper para C# .NET

//Integrantes : Jair Huaman Canqui

// : Miguel Ravelo Jove

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Windows.Forms;

using Emgu.CV;

using Emgu.CV.Structure;

using Emgu.CV.CvEnum;

using System.IO;

using System.Diagnostics;

using System.IO.Ports;

using System.Threading;

namespace DeteccionRostros

{

public partial class FrmPrincipal : Form

{

//Declaración de todas las variables, vectores and haarcascades

Image<Bgr, Byte> currentFrame;

Image<Bgr, Byte> currentFrameSecurity;

Capture grabber;

Capture grabberSecurity;

HaarCascade face;

HaarCascade eye;

MCvFont font = new MCvFont(FONT.CV\_FONT\_HERSHEY\_TRIPLEX, 0.5d, 0.5d);

Image<Gray, byte> result, TrainedFace = null;

Image<Gray, byte> gray = null;

List<Image<Gray, byte>> trainingImages = new List<Image<Gray, byte>>();

List<string> labels= new List<string>();

List<string> NamePersons = new List<string>();

List<int> CUIDetecteds = new List<int>();

int ContTrain, NumLabels, t;

string name,apellidos, names = null;

int CUI, rostrosDetectados;

BindingSource bindingSourceModems;

private int ContadorTiempo = 0;

private System.Windows.Forms.Timer timer = new System.Windows.Forms.Timer();

private bool primerRostroDetectado = true;

//Procedimiento para iniciar la lista de asistentes

private void initListView()

{

// Agregar las columnas

lvEstudiantes.Columns.Add("CUI", 150);

lvEstudiantes.Columns.Add("Nombres", 150);

lvEstudiantes.Columns.Add("Apellidos", 150);

}

private void iniciarTimer() {

timer.Interval = 1000;

timer.Tick += timer\_Tick;

//timer.Start();

}

//Procedimiento para obtener los Modems conectados a la PC y enlazarlos al combobox

private void cargarModems() {

GSMcom gsmControl = new GSMcom();

bindingSourceModems = new BindingSource();

bindingSourceModems.DataSource = gsmControl.ListarModemsConectados();

//Cargar la lista de puertos conectados

cbModems.DataSource = bindingSourceModems.DataSource;

cbModems.DisplayMember = "Description";

cbModems.ValueMember = "Name";

}

//Funcion para enviar un SMS, entra el numero de celular y el puerto al que esta conectado el GSM Modem

private void enviarMensaje(string celular, string COM) {

try

{

SerialPort sp = new SerialPort();

sp.PortName = cbModems.SelectedValue.ToString();

sp.Open();

sp.WriteLine("AT" + Environment.NewLine);

Thread.Sleep(200);

sp.WriteLine("AT+CMGF=1" + Environment.NewLine);

Thread.Sleep(100);

sp.WriteLine("AT+CMGS=\"" + celular + "\"" + Environment.NewLine);

Thread.Sleep(200);

sp.WriteLine(txtMensaje.Text + Environment.NewLine);

Thread.Sleep(100);

sp.Write(new byte[] { 26 }, 0, 1);

Thread.Sleep(100);

var response = sp.ReadExisting();

if (response.Contains("ERROR"))

{

MessageBox.Show("Error al enviar el mensaje!", "Mensaje", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else {

MessageBox.Show("Mensaje Enviado !", "Mensaje", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

sp.Close();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Mensaje", MessageBoxButtons.OK, MessageBoxIcon.Error);

throw;

}

}

//Evento del boton iniciar seguridad

private void btnInitSecurity\_Click(object sender, EventArgs e)

{

//Inicializar la captura de al WebCam

grabberSecurity = new Capture();

grabberSecurity.QueryFrame();

//Inicializar evento FrameGraber

Application.Idle += new EventHandler(FrameGrabberSecurity);

btnInitSecurity.Enabled = false;

}

//Evento del boton Exportar

private void btnExport2Excel\_Click(object sender, EventArgs e)

{

Export2Excel();

}

//Esta función se ejecuta cada 1 segundo y aumenta el contador para verificar que

//ya pasaron 5 minutos para enviar el siguiente SMS

private void timer\_Tick(object sender, EventArgs e)

{

ContadorTiempo++;

}

//Evento que se ejecuta al iniciar el programa

public FrmPrincipal()

{

InitializeComponent();

initListView();

iniciarTimer();

cargarModems();

//Cargar haarcascades para la detección de rostros

face = new HaarCascade("haarcascade\_frontalface\_default.xml");

try

{

//Cargar los rostros previamente entrenados y labes para cada imagen

string Labelsinfo = File.ReadAllText(Application.StartupPath + "/TrainedFaces/TrainedLabels.txt");

string[] Labels = Labelsinfo.Split('%');

NumLabels = Convert.ToInt16(Labels[0]);

ContTrain = NumLabels;

string LoadFaces;

for (int tf = 1; tf < NumLabels+1; tf++)

{

LoadFaces = "face" + tf + ".bmp";

trainingImages.Add(new Image<Gray, byte>(Application.StartupPath + "/TrainedFaces/" + LoadFaces));

labels.Add(Labels[tf]);

}

}

catch(Exception e)

{

MessageBox.Show("En caso use la opción de tomar lista aun no hay Nada en la base de datos, Por favor agrege al menos un rostro(Agrege un estudiante con el boton Guardar).", "Cargar Rostros entrenados", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

}

//Iniciar la captura de la webcam e iniciar la detección de rostros

private void btnTomarLista\_Click(object sender, EventArgs e)

{

//Inicializar la captura de al WebCam

grabber = new Capture();

grabber.QueryFrame();

//Inicializar evento FrameGraber

Application.Idle += new EventHandler(FrameGrabber);

btnTomarLista.Enabled = false;

}

//Evento del boton Guardar para Guardar un rostro junto con su CUI y nombres y apellidos

private void btnGuardar\_Click(object sender, System.EventArgs e)

{

try

{

//Contador de Rostros entrenados

ContTrain = ContTrain + 1;

//Obtener un Frame en escala de grises del dispositivo de captura(Webcam)

gray = grabber.QueryGrayFrame().Resize(320, 240, Emgu.CV.CvEnum.INTER.CV\_INTER\_CUBIC);

//Detector de rostros

MCvAvgComp[][] facesDetected = gray.DetectHaarCascade(

face,

1.2,

10,

Emgu.CV.CvEnum.HAAR\_DETECTION\_TYPE.DO\_CANNY\_PRUNING,

new Size(20, 20));

//Acción para cada elemento detectados

foreach (MCvAvgComp f in facesDetected[0])

{

TrainedFace = currentFrame.Copy(f.rect).Convert<Gray, byte>();

break;

}

//Cambiar tamaño del rostro detectado para forzar a comprar el mismo tamaño con la

//imagen de prueba con el metodo de inperpolacion cubica

TrainedFace = result.Resize(100, 100, Emgu.CV.CvEnum.INTER.CV\_INTER\_CUBIC);

trainingImages.Add(TrainedFace);

labels.Add(txtCUI.Value + "," + txtNombres.Text + "," + txtApellidos.Text);

//Mostrar rostro agregado en escala de grises

imgBoxEstudianteNuevo.Image = TrainedFace;

//Escribir el numero de rostros entrenados en un archivo de texto para su posterior carga

File.WriteAllText(Application.StartupPath + "/TrainedFaces/TrainedLabels.txt", trainingImages.ToArray().Length.ToString() + "%");

//Escribir las etiquetas de los rostros entrenados en un archivo de texto para su posterior carga

for (int i = 1; i < trainingImages.ToArray().Length + 1; i++)

{

trainingImages.ToArray()[i - 1].Save(Application.StartupPath + "/TrainedFaces/face" + i + ".bmp");

File.AppendAllText(Application.StartupPath + "/TrainedFaces/TrainedLabels.txt", labels.ToArray()[i - 1] + "%");

}

MessageBox.Show(txtNombres.Text + "´s Rostros detectados y agregados", "Entrenamiento exitoso", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch

{

MessageBox.Show("No se ha detectado ningun Rostro", "Entrenamiento fallido", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

}

//Frame Grabber para la toma de lista de asistentes

void FrameGrabber(object sender, EventArgs e)

{

lblRostrosDetectados.Text = "0";

NamePersons.Add("");

//Obtener el actual Frame del dispositivo de captura

currentFrame = grabber.QueryFrame().Resize(320, 240, Emgu.CV.CvEnum.INTER.CV\_INTER\_CUBIC);

//Convertirlo a Escala de grises

gray = currentFrame.Convert<Gray, Byte>();

//Detector de Rostros

MCvAvgComp[][] facesDetected = gray.DetectHaarCascade(

face,

1.2,

10,

Emgu.CV.CvEnum.HAAR\_DETECTION\_TYPE.DO\_CANNY\_PRUNING,

new Size(20, 20));

try

{

//Accion para cada elemento detectado

foreach (MCvAvgComp f in facesDetected[0])

{

t = t + 1;

result = currentFrame.Copy(f.rect).Convert<Gray, byte>().Resize(100, 100, Emgu.CV.CvEnum.INTER.CV\_INTER\_CUBIC);

//Dibujar el rostro detectado en el 0th (gray) channel con el color azul

currentFrame.Draw(f.rect, new Bgr(Color.Red), 2);

if (trainingImages.ToArray().Length != 0)

{

//TermCriteria para reconocimiento de rostros con numeros de imagenes entrenadas como maxIteration

MCvTermCriteria termCrit = new MCvTermCriteria(ContTrain, 0.001);

//Reconocedor Eigen face

EigenObjectRecognizer recognizer = new EigenObjectRecognizer(

trainingImages.ToArray(),

labels.ToArray(),

3000,

ref termCrit);

string[] tmpData = recognizer.Recognize(result).Split(',');

name = tmpData[1];

apellidos = tmpData[2];

//Manejar CUIs y Agregar a lista de asistentes.

CUI = int.Parse(tmpData[0]);

bool exists = false;

//verificar si el CUI ya fue detectado

foreach (int unitCUI in CUIDetecteds)

{

if (CUI == unitCUI)

{

exists = true;

}

}

if (!exists)

{

CUIDetecteds.Add(CUI);

string[] arr = new string[3];

ListViewItem itm;

//Agregar items a la lista de asistentes

arr[0] = CUI.ToString();

arr[1] = name;

arr[2] = apellidos;

itm = new ListViewItem(arr);

lvEstudiantes.Items.Add(itm);

}

//Dibujar la etiqueta para cada rostro detectado y reconocido

currentFrame.Draw(name, ref font, new Point(f.rect.X - 2, f.rect.Y - 2), new Bgr(Color.LightGreen));

}

NamePersons[t - 1] = name;

NamePersons.Add("");

//Establecer el numero de rostros detectados en la escena

lblRostrosDetectados.Text = facesDetected[0].Length.ToString();

}

t = 0;

}

catch (Exception)

{

}

//Concatenación de nombres de personas reconocidas

for (int nnn = 0; nnn < facesDetected[0].Length; nnn++)

{

names = names + NamePersons[nnn] + ", ";

}

//Mostrar los rostros procesados y reconocidos

imageBoxFrameGrabber.Image = currentFrame;

lblPresentesEscene.Text = names;

names = "";

//Limpiar la list(vector) de nombres

NamePersons.Clear();

}

//Frame Grabber para el mini sistema de seguridad con notificación

void FrameGrabberSecurity(object sender, EventArgs e)

{

try

{

//Obtener el actual Frame del dispositivo de captura

currentFrameSecurity = grabberSecurity.QueryFrame().Resize(320, 240, Emgu.CV.CvEnum.INTER.CV\_INTER\_CUBIC);

//Convertirlo a Escala de grises

gray = currentFrameSecurity.Convert<Gray, Byte>();

//Detector de Rostros

MCvAvgComp[][] facesDetected = gray.DetectHaarCascade(

face,

1.2,

10,

Emgu.CV.CvEnum.HAAR\_DETECTION\_TYPE.DO\_CANNY\_PRUNING,

new Size(20, 20));

bool messageSend = false;

//Accion para cada elemento detectado

foreach (MCvAvgComp f in facesDetected[0])

{

t = t + 1;

result = currentFrameSecurity.Copy(f.rect).Convert<Gray, byte>().Resize(100, 100, Emgu.CV.CvEnum.INTER.CV\_INTER\_CUBIC);

//Dibujar el rostro detectado en el 0th (gray) channel con el color azul

currentFrameSecurity.Draw(f.rect, new Bgr(Color.Red), 2);

//Establecer el numero de rostros detectados en la escena

rostrosDetectados = facesDetected[0].Length;

if (primerRostroDetectado && !messageSend)

{

timer.Start();

primerRostroDetectado = false;

enviarMensaje(txtCelular.Text, cbModems.SelectedValue.ToString());

messageSend = true;

}

//Envia un SMS al detectar algun rostro y espera 5 minutos antes de enviar el

//siguiente SMS en caso detecte algun rostro.

//Es necesario para que no envie SMS a cada segundo

if (ContadorTiempo >= 300 && !messageSend)

{

enviarMensaje(txtCelular.Text, cbModems.SelectedValue.ToString());

messageSend = true;

ContadorTiempo = 0;

}

}

t = 0;

//Mostrar los rostros procesados y reconocidos

imgBoxFrameGrabberSecurity.Image = currentFrameSecurity;

}

catch (NullReferenceException nullex)

{

MessageBox.Show("No se ha detectado un puerto COM o un Celular");

primerRostroDetectado = true;

ContadorTiempo = 0;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

//Función para exportar ListView a Excel

private void Export2Excel()

{

try

{

string[] st = new string[lvEstudiantes.Columns.Count];

DirectoryInfo di = new DirectoryInfo(@"c:\DeteccionRostros\");

if (di.Exists == false)

di.Create();

StreamWriter sw = new StreamWriter(@"c:\DeteccionRostros\" + "Asistentes" + ".xls", false);

sw.AutoFlush = true;

for (int col = 0; col < lvEstudiantes.Columns.Count; col++)

{

sw.Write("\t" + lvEstudiantes.Columns[col].Text.ToString());

}

int rowIndex = 1;

int row = 0;

string st1 = "";

for (row = 0; row < lvEstudiantes.Items.Count; row++)

{

if (rowIndex <= lvEstudiantes.Items.Count)

rowIndex++;

st1 = "\n";

for (int col = 0; col < lvEstudiantes.Columns.Count; col++)

{

st1 = st1 + "\t" + "'" + lvEstudiantes.Items[row].SubItems[col].Text.ToString();

}

sw.WriteLine(st1);

}

sw.Close();

FileInfo fil = new FileInfo(@"c:\DeteccionRostros\" + "Asistentes" + ".xls");

if (fil.Exists == true)

MessageBox.Show("Proceso Completo", "Exportar a Excel", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

}

}

***GSM***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Management;

using System.Text;

namespace DeteccionRostros

{

public class GSMcom

{

public string Name { get; set; }

public string Description { get; set; }

public GSMcom()

{

}

override

public string ToString()

{

return $"{Description} {Name}";

}

public GSMcom[] ListarModemsConectados()

{

List<GSMcom> gsmCom = new List<GSMcom>();

ConnectionOptions options = new ConnectionOptions();

options.Impersonation = ImpersonationLevel.Impersonate;

options.EnablePrivileges = true;

string connString = $@"\\{Environment.MachineName}\root\cimv2";

ManagementScope scope = new ManagementScope(connString, options);

scope.Connect();

ObjectQuery query = new ObjectQuery("SELECT \* FROM Win32\_POTSModem");

ManagementObjectSearcher search = new ManagementObjectSearcher(scope, query);

ManagementObjectCollection collection = search.Get();

foreach (ManagementObject obj in collection)

{

string portName = obj["AttachedTo"].ToString();

string portDescription = obj["Description"].ToString();

if (portName != "")

{

GSMcom com = new GSMcom();

com.Name = portName;

com.Description = portDescription;

gsmCom.Add(com);

}

}

return gsmCom.ToArray();

}

}

}

***EigenFace***

using System;

using System.Diagnostics;

using Emgu.CV.Structure;

namespace Emgu.CV

{

/// <summary>

/// An object recognizer using PCA (Principle Components Analysis)

/// </summary>

[Serializable]

public class EigenObjectRecognizer

{

private Image<Gray, Single>[] \_eigenImages;

private Image<Gray, Single> \_avgImage;

private Matrix<float>[] \_eigenValues;

private string[] \_labels;

private double \_eigenDistanceThreshold;

/// <summary>

/// Get the eigen vectors that form the eigen space

/// </summary>

/// <remarks>The set method is primary used for deserialization, do not attemps to set it unless you know what you are doing</remarks>

public Image<Gray, Single>[] EigenImages

{

get { return \_eigenImages; }

set { \_eigenImages = value; }

}

/// <summary>

/// Get or set the labels for the corresponding training image

/// </summary>

public String[] Labels

{

get { return \_labels; }

set { \_labels = value; }

}

/// <summary>

/// Get or set the eigen distance threshold.

/// The smaller the number, the more likely an examined image will be treated as unrecognized object.

/// Set it to a huge number (e.g. 5000) and the recognizer will always treated the examined image as one of the known object.

/// </summary>

public double EigenDistanceThreshold

{

get { return \_eigenDistanceThreshold; }

set { \_eigenDistanceThreshold = value; }

}

/// <summary>

/// Get the average Image.

/// </summary>

/// <remarks>The set method is primary used for deserialization, do not attemps to set it unless you know what you are doing</remarks>

public Image<Gray, Single> AverageImage

{

get { return \_avgImage; }

set { \_avgImage = value; }

}

/// <summary>

/// Get the eigen values of each of the training image

/// </summary>

/// <remarks>The set method is primary used for deserialization, do not attemps to set it unless you know what you are doing</remarks>

public Matrix<float>[] EigenValues

{

get { return \_eigenValues; }

set { \_eigenValues = value; }

}

private EigenObjectRecognizer()

{

}

/// <summary>

/// Create an object recognizer using the specific tranning data and parameters, it will always return the most similar object

/// </summary>

/// <param name="images">The images used for training, each of them should be the same size. It's recommended the images are histogram normalized</param>

/// <param name="termCrit">The criteria for recognizer training</param>

public EigenObjectRecognizer(Image<Gray, Byte>[] images, ref MCvTermCriteria termCrit)

: this(images, GenerateLabels(images.Length), ref termCrit)

{

}

private static String[] GenerateLabels(int size)

{

String[] labels = new string[size];

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

labels[i] = i.ToString();

return labels;

}

/// <summary>

/// Create an object recognizer using the specific tranning data and parameters, it will always return the most similar object

/// </summary>

/// <param name="images">The images used for training, each of them should be the same size. It's recommended the images are histogram normalized</param>

/// <param name="labels">The labels corresponding to the images</param>

/// <param name="termCrit">The criteria for recognizer training</param>

public EigenObjectRecognizer(Image<Gray, Byte>[] images, String[] labels, ref MCvTermCriteria termCrit)

: this(images, labels, 0, ref termCrit)

{

}

/// <summary>

/// Create an object recognizer using the specific tranning data and parameters

/// </summary>

/// <param name="images">The images used for training, each of them should be the same size. It's recommended the images are histogram normalized</param>

/// <param name="labels">The labels corresponding to the images</param>

/// <param name="eigenDistanceThreshold">

/// The eigen distance threshold, (0, ~1000].

/// The smaller the number, the more likely an examined image will be treated as unrecognized object.

/// If the threshold is &lt; 0, the recognizer will always treated the examined image as one of the known object.

/// </param>

/// <param name="termCrit">The criteria for recognizer training</param>

public EigenObjectRecognizer(Image<Gray, Byte>[] images, String[] labels, double eigenDistanceThreshold, ref MCvTermCriteria termCrit)

{

Debug.Assert(images.Length == labels.Length, "The number of images should equals the number of labels");

Debug.Assert(eigenDistanceThreshold >= 0.0, "Eigen-distance threshold should always >= 0.0");

CalcEigenObjects(images, ref termCrit, out \_eigenImages, out \_avgImage);

/\*

\_avgImage.SerializationCompressionRatio = 9;

foreach (Image<Gray, Single> img in \_eigenImages)

//Set the compression ration to best compression. The serialized object can therefore save spaces

img.SerializationCompressionRatio = 9;

\*/

\_eigenValues = Array.ConvertAll<Image<Gray, Byte>, Matrix<float>>(images,

delegate(Image<Gray, Byte> img)

{

return new Matrix<float>(EigenDecomposite(img, \_eigenImages, \_avgImage));

});

\_labels = labels;

\_eigenDistanceThreshold = eigenDistanceThreshold;

}

#region static methods

/// <summary>

/// Caculate the eigen images for the specific traning image

/// </summary>

/// <param name="trainingImages">The images used for training </param>

/// <param name="termCrit">The criteria for tranning</param>

/// <param name="eigenImages">The resulting eigen images</param>

/// <param name="avg">The resulting average image</param>

public static void CalcEigenObjects(Image<Gray, Byte>[] trainingImages, ref MCvTermCriteria termCrit, out Image<Gray, Single>[] eigenImages, out Image<Gray, Single> avg)

{

int width = trainingImages[0].Width;

int height = trainingImages[0].Height;

IntPtr[] inObjs = Array.ConvertAll<Image<Gray, Byte>, IntPtr>(trainingImages, delegate(Image<Gray, Byte> img) { return img.Ptr; });

if (termCrit.max\_iter <= 0 || termCrit.max\_iter > trainingImages.Length)

termCrit.max\_iter = trainingImages.Length;

int maxEigenObjs = termCrit.max\_iter;

#region initialize eigen images

eigenImages = new Image<Gray, float>[maxEigenObjs];

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

eigenImages[i] = new Image<Gray, float>(width, height);

IntPtr[] eigObjs = Array.ConvertAll<Image<Gray, Single>, IntPtr>(eigenImages, delegate(Image<Gray, Single> img) { return img.Ptr; });

#endregion

avg = new Image<Gray, Single>(width, height);

CvInvoke.cvCalcEigenObjects(

inObjs,

ref termCrit,

eigObjs,

null,

avg.Ptr);

}

/// <summary>

/// Decompose the image as eigen values, using the specific eigen vectors

/// </summary>

/// <param name="src">The image to be decomposed</param>

/// <param name="eigenImages">The eigen images</param>

/// <param name="avg">The average images</param>

/// <returns>Eigen values of the decomposed image</returns>

public static float[] EigenDecomposite(Image<Gray, Byte> src, Image<Gray, Single>[] eigenImages, Image<Gray, Single> avg)

{

return CvInvoke.cvEigenDecomposite(

src.Ptr,

Array.ConvertAll<Image<Gray, Single>, IntPtr>(eigenImages, delegate(Image<Gray, Single> img) { return img.Ptr; }),

avg.Ptr);

}

#endregion

/// <summary>

/// Given the eigen value, reconstruct the projected image

/// </summary>

/// <param name="eigenValue">The eigen values</param>

/// <returns>The projected image</returns>

public Image<Gray, Byte> EigenProjection(float[] eigenValue)

{

Image<Gray, Byte> res = new Image<Gray, byte>(\_avgImage.Width, \_avgImage.Height);

CvInvoke.cvEigenProjection(

Array.ConvertAll<Image<Gray, Single>, IntPtr>(\_eigenImages, delegate(Image<Gray, Single> img) { return img.Ptr; }),

eigenValue,

\_avgImage.Ptr,

res.Ptr);

return res;

}

/// <summary>

/// Get the Euclidean eigen-distance between <paramref name="image"/> and every other image in the database

/// </summary>

/// <param name="image">The image to be compared from the training images</param>

/// <returns>An array of eigen distance from every image in the training images</returns>

public float[] GetEigenDistances(Image<Gray, Byte> image)

{

using (Matrix<float> eigenValue = new Matrix<float>(EigenDecomposite(image, \_eigenImages, \_avgImage)))

return Array.ConvertAll<Matrix<float>, float>(\_eigenValues,

delegate(Matrix<float> eigenValueI)

{

return (float)CvInvoke.cvNorm(eigenValue.Ptr, eigenValueI.Ptr, Emgu.CV.CvEnum.NORM\_TYPE.CV\_L2, IntPtr.Zero);

});

}

/// <summary>

/// Given the <paramref name="image"/> to be examined, find in the database the most similar object, return the index and the eigen distance

/// </summary>

/// <param name="image">The image to be searched from the database</param>

/// <param name="index">The index of the most similar object</param>

/// <param name="eigenDistance">The eigen distance of the most similar object</param>

/// <param name="label">The label of the specific image</param>

public void FindMostSimilarObject(Image<Gray, Byte> image, out int index, out float eigenDistance, out String label)

{

float[] dist = GetEigenDistances(image);

index = 0;

eigenDistance = dist[0];

for (int i = 1; i < dist.Length; i++)

{

if (dist[i] < eigenDistance)

{

index = i;

eigenDistance = dist[i];

}

}

label = Labels[index];

}

/// <summary>

/// Try to recognize the image and return its label

/// </summary>

/// <param name="image">The image to be recognized</param>

/// <returns>

/// String.Empty, if not recognized;

/// Label of the corresponding image, otherwise

/// </returns>

public String Recognize(Image<Gray, Byte> image)

{

int index;

float eigenDistance;

String label;

FindMostSimilarObject(image, out index, out eigenDistance, out label);

return (\_eigenDistanceThreshold <= 0 || eigenDistance < \_eigenDistanceThreshold ) ? \_labels[index] : String.Empty;

}

}

}