Face Classification and Recognition System Using C#

Table of Content

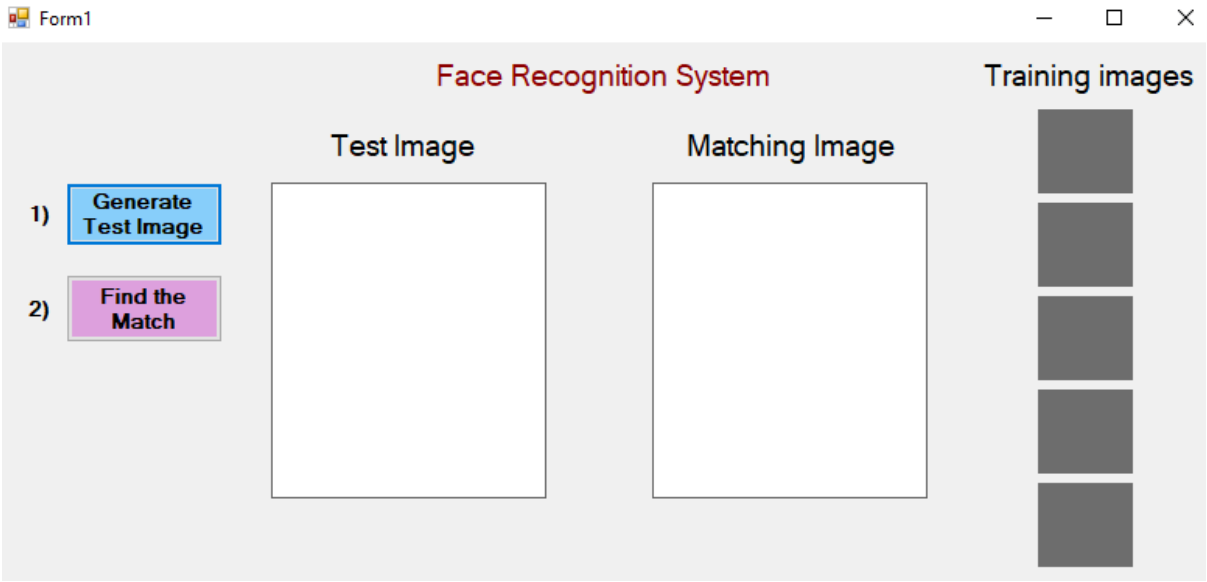
Description:

The aim of this project is to develop an automatic face recognition and classification system using Visual Studio C# to process images and recognize faces in the images. A total of 40 classes with 10 images in each class, was used to build this application. Five images from each class were used for training and the rest five were used for testing purpose.

The user first generates a random test image, and then triggers the training of a Linear regression model to process and predict the class of the test image and display the results.

Evaluation:

1) Front-End (GUI):



The GUI has been developed using WinForms in C# .Net:

- Generate Test Image button – To randomly select an image for processing and displaying
- Test Image box – To display the randomly selected test image
- Find the Match button – To initiate training based on the Linear regression algorithm, and displaying the results
- Matching Image box – To display the matching image after training
- Training images section – To display the list of images that were used for training

2) Test image generator:

An image is randomly selected from one of the 40 classes for testing purpose. The test image is resized, normalized and stored in a vector for further processing.

- Generating and using random numbers between certain ranges to select our test image –

```
Random rnd = new Random();
int randomTestingImg = rnd.Next(6, 10); //Selects a random image between 6 and 10 in a class
int randomDir = rnd.Next(1, 40);    //Selects a random Image class between 1 and 40
```

- Image was loaded using the following method –

```
Image testImg = Image.FromFile(dirs[randomDir] + "/" + randomTestingImg + ".jpg");
```

- Image was resized using the following method –

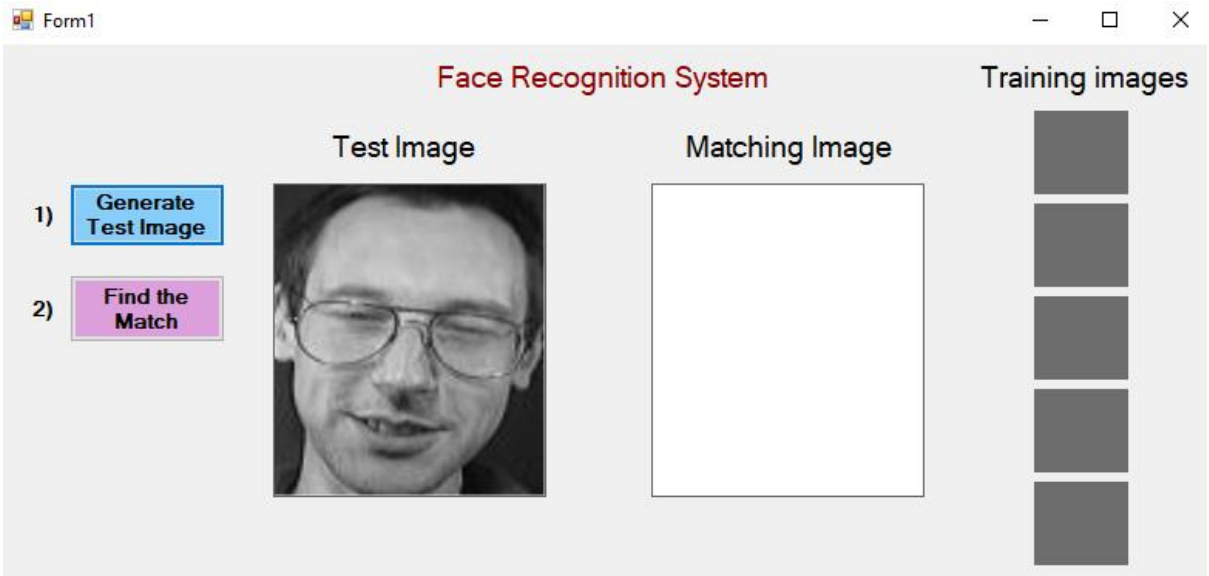
```
Bitmap resized = new Bitmap(testImg, new Size(30, 30));
```

- A vector was created to store the normalized test image –

```
Vector<float> NormalizedTestImgVector = Vector<float>.Build.Dense(900);
```

- The test image was displayed in the following way –

```
TestImageBox.Image = testImg;   //Displaying the randomly selected test image
```



3) Training and displaying results:

- In training phase, 5 images from each class are resized, normalized and stored in a *List* which is later converted to a *2d-array* –

```
Image original = Image.FromFile(CurrentImgFolder[counter]); //Reading an image
Bitmap resized = new Bitmap(original, new Size(30, 30));    //Resizing the image

List<float> ImgList = new List<float>();

ImgList = new List<float> (GetNormalizedImg(resized));  //Storing Normalized image in a List

ClassList.Add(new List<float>(ImgList));    //Storing each normalized image in a ClassList
```

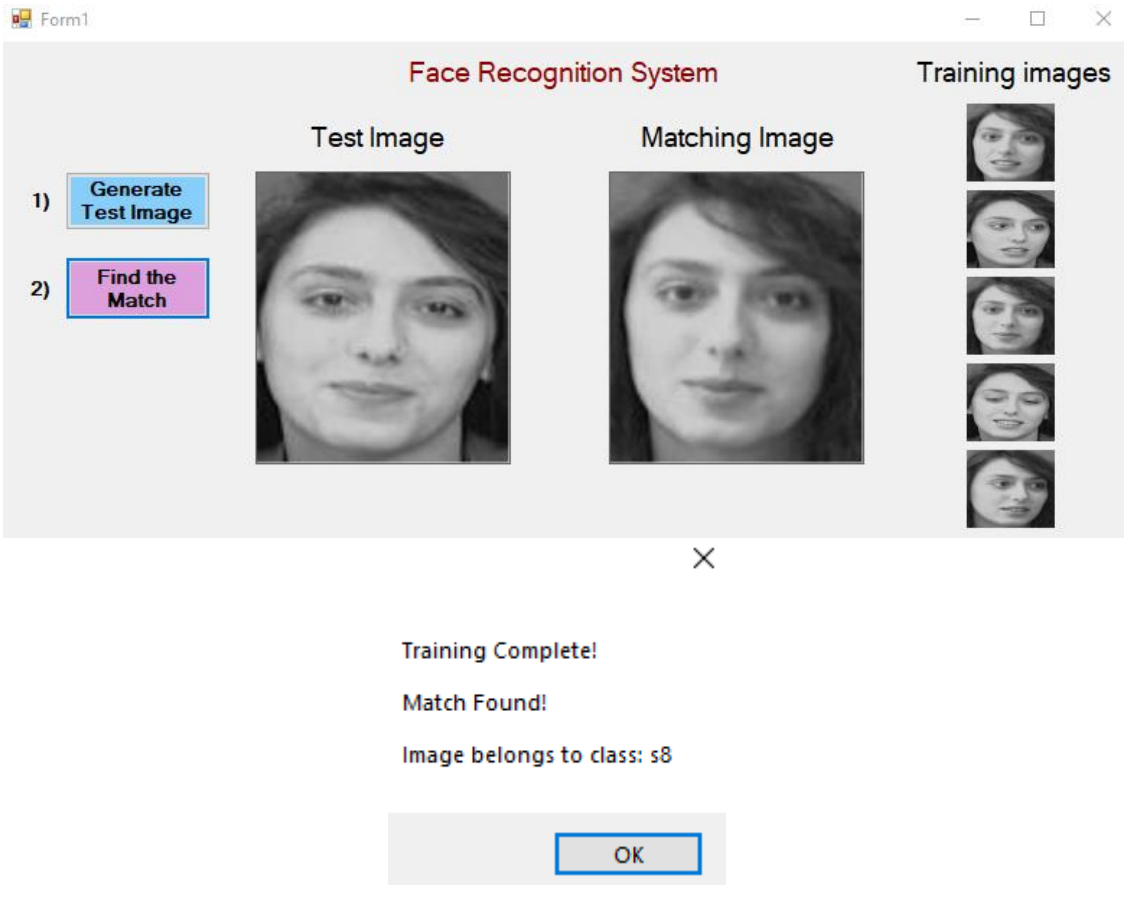
- The *ClassArray2d* and the Test image vector (*y*) are provided to a function to find the Beta value and the Euclidean distance –

```
  CalcBeta(ClassArray2d);   //Calculating final Beta value

Vector<float> BetaVector = Vector<float>.Build.Dense(5);    //Final value beta value

List<double> DistanceList = new List<double>(); //List tha contains Euclidean distance values
```

- Displaying results after finding the match –



4) How Normalisation was performed:

```
private List<float> GetNormalizedImg(Bitmap resized)   //Method to Normalize image
{
    List<float> theList = new List<float>();
    List<float> NormalizedList = new List<float>();

    for (int i = 0; i < resized.Width; i++)
    {
        for (int j = 0; j < resized.Height; j++)
        {
            theList.Add(resized.GetPixel(i, j).R);  //Storing pixel value in a list
        }
    }

    float max = theList.Max();

    for (int i = 0; i < theList.Count(); i++)
    {
        NormalizedList.Add((theList.ElementAt(i)) / max);   //Dividing each value in the list with the max value
    }

    return NormalizedList;
}
```

- The pixels of a resized image are stored in a list called *theList*
- Each pixel value in the list is divided by the largest pixel value in the list, in order to bring down the pixel range between 0 – 1 from 1-255.
- The normalized image is now stored in a new list called *NormalizedList.*

5) How Linear Regression was performed:

- To efficiently implement vectors and matrices in the solution, an additional package called Math.NET was installed. This package written in C# provides methods and algorithms for numerical computations, and gives rich support for vectors and matrices.
  The purpose of using such a tool was to demonstrate code reuse in C# programming.

```
using MathNet.Numerics.LinearAlgebra;
using MathNet.Numerics.LinearAlgebra.Double;
```

- Computation of Beta vector –

$$\hat{\beta}_i = (\mathbf{X}_i^T \mathbf{X}_i)^{-1} \mathbf{X}_i^T \mathbf{y}.$$

```
BetaVector = ((ClassMatrix.Transpose() * ClassMatrix).Inverse()) * (ClassMatrix.Transpose() * NormalizedTestImgVector);
```

- Computation of Predicted vector ŷ –

$$\hat{\mathbf{y}}_i = \mathbf{X}_i \hat{\beta}_i, \quad i = 1, 2, \ldots, N$$

```
var Predicted_y = ClassMatrix * BetaVector;  //Computing predicted 'y' value
```

- Computation of Euclidean distance using L2Norm –

L2Norm = the square root of the sum of the squared values.

$$d_i(\mathbf{y}) = \|\mathbf{y} - \hat{\mathbf{y}}_i\|_2, \quad i = 1, 2, \ldots, N$$

```
var y = NormalizedTestImgVector - Predicted_y;

var d = y.L2Norm(); //Performing L2Norm to find Euclidean distance

DistanceList.Add(d);
```

- Finding least Euclidean distance –

```
double minVal = DistanceList.Min();
int index = DistanceList.IndexOf(minVal);
```

- Resource referred: https://numerics.mathdotnet.com/Matrix.html

How to Run the Application:

Pre-requisite:

The application requires an additional framework Math.NET to be installed, the installation steps are as follow:

Navigate to:

Tools -> NuGet Package Manager -> Package Manager Console

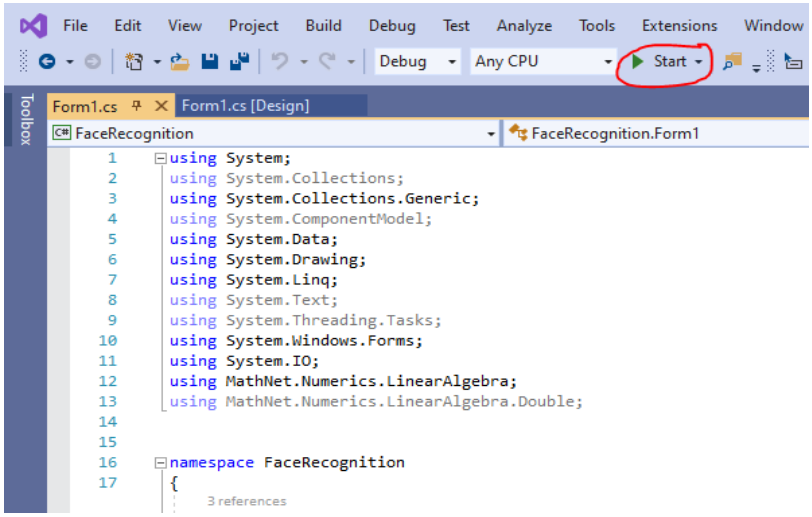
Type the following command to install the framework:

Install-Package MathNet.Numerics -Version 4.8.1

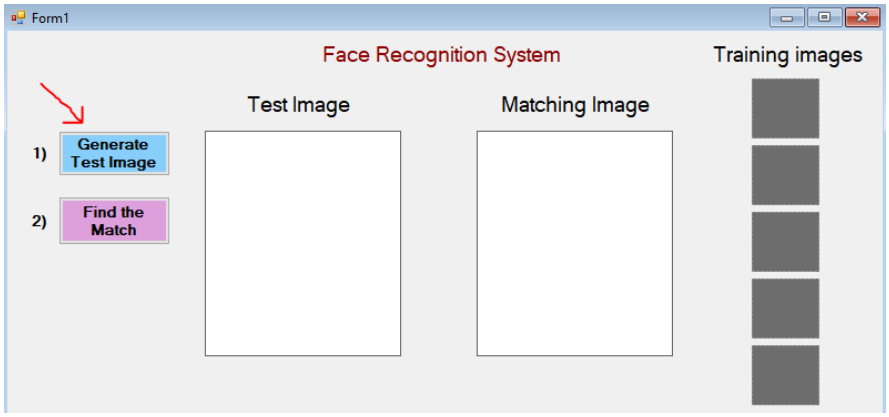
source of information: https://www.nuget.org/packages/MathNet.Numerics/

Running the application:

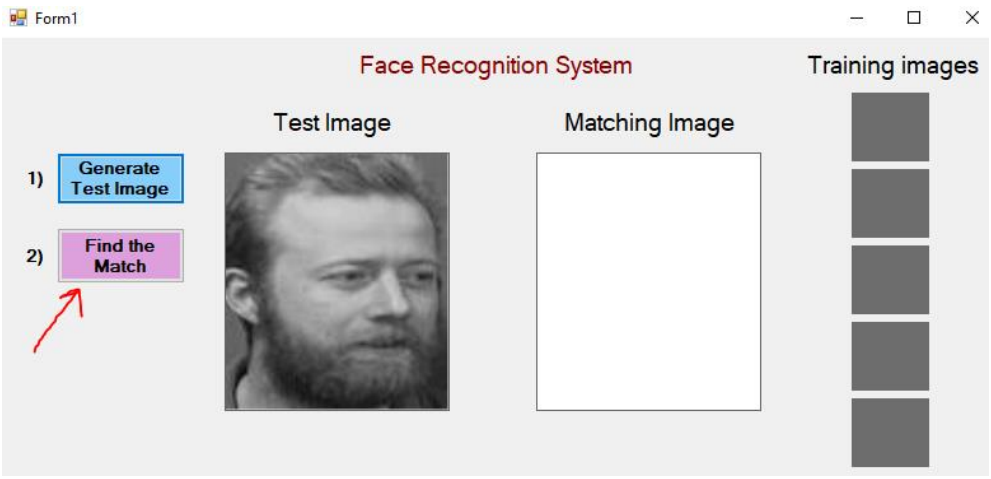
1) Press Start button to build and run the application –



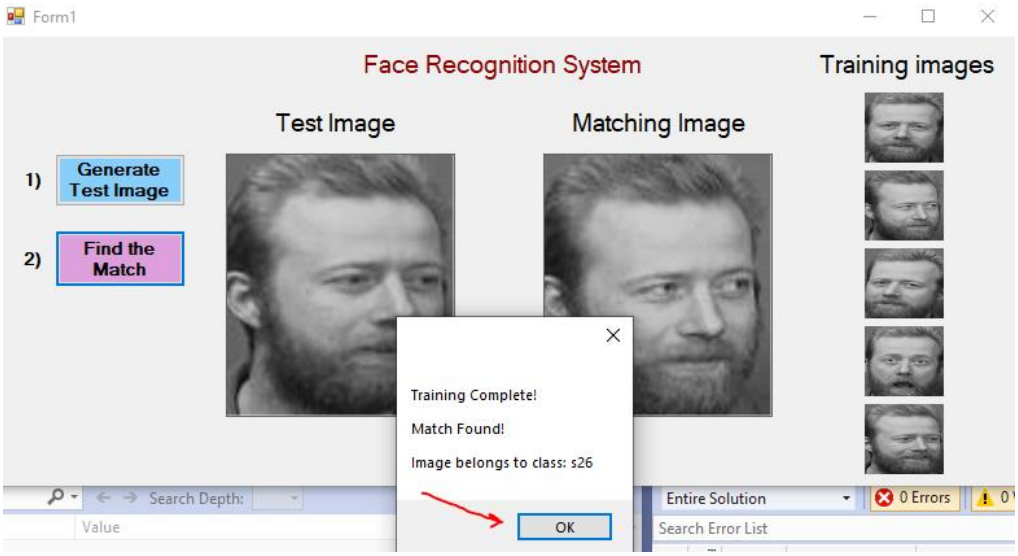
2) Press “Generate Test Image”



2) Press “Find the Match”



3) Press “OK” to continue using the application:



Self-Diagnosis:

- Different storage options in C# were explored, hence, there are multiple instances of transferring data between different data structures, for example, from jagged array to 2d-arrays, from list to vectors etc.
- Code can be made more modular.
- No button click function implemented before displaying training images list, as it was felt needless.
- There is a 1/20 chance of a mismatch between test image and result image.
- The aim of this project was not to develop a full-fledged AI solution; hence, the design of this application is unlike the way a typical AI model is.

Features	Implementation
Reading images from dataset	Yes
WinForms based GUI	Yes
Linear Regression algorithm	Yes
Displaying test image	Yes
Displaying matching image after training	Yes
Code Reuse	Yes
Use of frameworks	Yes
Implementing theory concepts	Yes
Acknowledgement of reused code	Yes

Challenges Faced:

- Initially, there was difficulty in reading images that were in PGM format, this issue was resolved when it was discovered that C# .Net does not support certain image formats that well. So, images in JPG format were used instead.

- Selecting the appropriate data structure for storage was a challenge at every step throughout the project. Certain storage structures were deemed suitable initially but had to be converted to a different format later on due to specific design requirements.

- Finding the right tool to go about with the Linear regression algorithm was challenging. But after finding the right resources and framework to use, there was a significant progress in the computations involved in the project.

Online code that was reused:

Reading images: https://docs.microsoft.com/en-us/dotnet/api/system.drawing.image.fromfile?view=netframework-4.8

Resize images: https://stackoverflow.com/questions/1922040/how-to-resize-an-image-c-sharp

Using Lists in C#: https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=netframework-4.8

Converting List to 2d-Array: https://stackoverflow.com/questions/39484503/transform-list-into-2d-array

Substring in C#: https://www.c-sharpcorner.com/UploadFile/mahesh/substring-in-C-Sharp/

Installing Math.NET framework package: https://www.nuget.org/packages/MathNet.Numerics/

Using Math.NET: https://numerics.mathdotnet.com/Matrix.html