Xây dựng Winform nhận diện khuôn mặt sử dụng thư viện Face Recognition python

# 1. Giới thiệu thư viện Face Recognition

Face Recognition là một thư viện mã nguồn mở được sử dụng để thực hiện nhận diện khuôn mặt trong ảnh và video. Thư viện này được xây dựng trên nền tảng Python và cung cấp một loạt các công cụ mạnh mẽ để nhận diện và phân loại khuôn mặt dựa trên các thuật toán thị giác máy tính.

Dưới đây là một số điểm chính về thư viện Face Recognition:

* **Dễ sử dụng**: Thư viện cung cấp một API đơn giản và dễ sử dụng, giúp người phát triển nhanh chóng tích hợp chức năng nhận diện khuôn mặt vào ứng dụng của họ.
* **Tích hợp thuật toán mạnh mẽ**: Face Recognition sử dụng các thuật toán tiên tiến như dlib để thực hiện nhận diện khuôn mặt với độ chính xác cao. Điều này đảm bảo khả năng nhận diện tốt trong nhiều điều kiện ánh sáng và góc nhìn khác nhau.
* **Hỗ trợ nhiều định dạng ảnh**: Thư viện hỗ trợ nhiều định dạng ảnh khác nhau, bao gồm cả ảnh màu và ảnh xám.
* **Quản lý dữ liệu**: Face Recognition cung cấp các công cụ để quản lý cơ sở dữ liệu khuôn mặt, giúp người dùng lưu trữ và quản lý thông tin về các khuôn mặt đã được đăng ký.
* **Tuỳ chỉnh và mở rộng**: Thư viện cho phép người phát triển tuỳ chỉnh các tham số và mở rộng chức năng theo nhu cầu cụ thể của dự án.

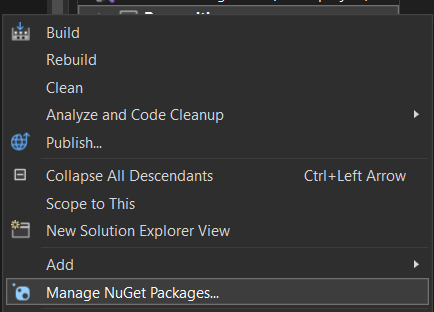
# 2. Cài đặt các thư viện cần thiết

## 2.1. Đối với Python

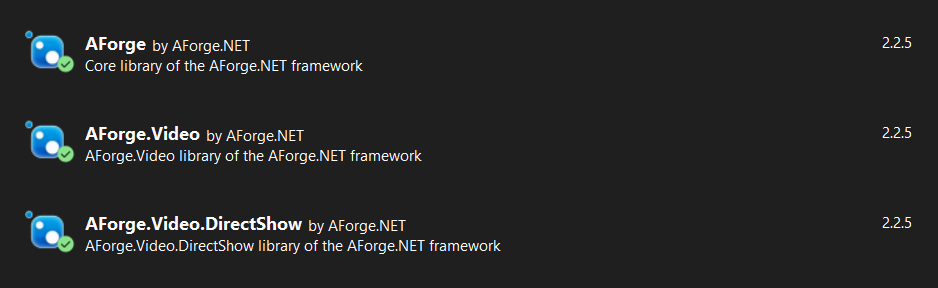
Bạn cần cài đặt thư viện Face Recognition thông qua câu lệnh sau đây: **pip install face\_recognition**

## 2.2. Đối với C#

Nhấn chuột phải vào dự án và chọn **Manage NuGet Packages** như hình bên dưới



Tiến hành cài đặt các thư viện như hình bên dưới để xử lý video từ camera



# 3. Giao diện Winform



# 4. Hướng dẫn chi tiết

## 4.1 Lõi chương trình (Python)

Trước tiên bạn cần import các thư viện và modules cần thiết

| import face\_recognition  import os  import sys |
| --- |

### 4.1.1. Hàm kiểm tra số lượng khuôn mặt trong bức ảnh truyền vào có phải 1 hay không?

| def is\_single\_face(image\_path):  # Đọc ảnh  image = face\_recognition.load\_image\_file(image\_path)  # Sử dụng HOG để nhận diện khuôn mặt  face\_locations = face\_recognition.face\_locations(image, model="hog")  # Kiểm tra số lượng khuôn mặt và xóa ảnh nếu không đúng 1 khuôn mặt  if len(face\_locations) == 1:  print(True)  else:  print(False)  # Xóa ảnh nếu không đúng 1 khuôn mặt  os.remove(image\_path) |
| --- |

### 4.1.2. Hàm mã hóa khuôn mặt dưới dạng vector trọng số

Hàm này nhận 3 tham số đầu vào đó là đường dẫn thư mục Dataset, đường dẫn file Models và dường dẫn Folder chứa các khuôn mặt đã huấn luyện.

**def encode\_images\_in\_dataset(datasetPath, output\_file, trainedPath):**

Trước tiên, hàm cần kiểm tra xem file chứa vector trọng số khuôn mặt hay **encoding.txt** đã tồn tại hay chưa, nếu chưa thì tạo mới tệp tại đường dẫn file Models được truyền vào.

| if not os.path.isfile(output\_file):  with open(output\_file, 'w') as file:  file.write('') |
| --- |

Tiếp theo, cần load tất cả các hình ảnh có định dạng PNG hoặc JPG trong thư mục dataset

| image\_files = [f for f in os.listdir(datasetPath) if f.endswith('.jpg') or f.endswith('.png')] |
| --- |

Sau đó, ta cần mở file encoding để ghi nối tiếp khuôn mặt được mã hóa. Ta lặp qua ảnh trong tập dataset. Đọc chúng và chuyển đổi các khuôn mặt được tìm thấy sang mã hóa thông qua phương thức face\_encoding của thư viện Face Recognition. Sau đó ta ghi mã hóa vào file **encoding.txt**. Di chuyển ảnh đã được xử lí vào thư mục Trained (Thư mục chứa ảnh đã huấn luyện). Do ta có thể huấn luyện nhiều ảnh của một người nên tránh bị trùng tên ảnh, ta cần kiểm tra thêm fileName của ảnh đó có trùng không bằng cách sử dụng biến đếm và gắn hậu tố cho ảnh bị trùng.

| with open(output\_file, 'a') as encoding\_file:  for image\_file in image\_files:  image\_path = os.path.join(datasetPath, image\_file)  image = face\_recognition.load\_image\_file(image\_path)  encoding = face\_recognition.face\_encodings(image)  if len(encoding) > 0:  encoding\_file.write(  f"{os.path.splitext(image\_file)[0]}: {encoding[0]}\n")  image\_name, image\_extension = os.path.splitext(  os.path.basename(image\_path))  new\_path = os.path.join(datasetPath, trainedPath, f"{image\_name}\_trained{image\_extension}")  counter = 1  while os.path.exists(new\_path):  new\_path = os.path.join(datasetPath, trainedPath, f"{image\_name}\_trained\_{counter}{image\_extension}")  counter += 1  os.rename(image\_path, new\_path) |
| --- |

Chi tiết mã nguồn đầy đủ của hàm:

| def encode\_images\_in\_dataset(datasetPath, output\_file, trainedPath):  # Kiểm tra xem file encodings.txt đã tồn tại chưa  if not os.path.isfile(output\_file):  # Nếu chưa tồn tại, tạo file mới và ghi đè nội dung  with open(output\_file, 'w') as file:  file.write('')  # Danh sách các hình ảnh trong thư mục  image\_files = [f for f in os.listdir(  datasetPath) if f.endswith('.jpg') or f.endswith('.png')]  # Mở file để ghi nối tiếp cho encodings.txt  with open(output\_file, 'a') as encoding\_file:  for image\_file in image\_files:  # Đường dẫn đầy đủ của hình ảnh  image\_path = os.path.join(datasetPath, image\_file)  # Đọc hình ảnh và chuyển đổi thành encoding  image = face\_recognition.load\_image\_file(image\_path)  encoding = face\_recognition.face\_encodings(image)  if len(encoding) > 0:  # Ghi encoding vào encodings.txt  encoding\_file.write(  f"{os.path.splitext(image\_file)[0]}: {encoding[0]}\n")  # Di chuyển file đã sử dụng vào thư mục 'Trained' với xử lý trùng tên  image\_name, image\_extension = os.path.splitext(  os.path.basename(image\_path))  new\_path = os.path.join(datasetPath, trainedPath, f"{image\_name}\_trained{image\_extension}")  # Kiểm tra xem tên file mới đã tồn tại hay chưa  counter = 1  while os.path.exists(new\_path):  new\_path = os.path.join(datasetPath, trainedPath, f"{image\_name}\_trained\_{counter}{image\_extension}")  counter += 1  os.rename(image\_path, new\_path) |
| --- |

### 4.1.3. Hàm đọc thông tin từ file mã hóa

Hàm nhận vào đường dẫn tệp mã hóa và trả về 2 giá trị là tên và vecto mã hóa

| def load\_encoding\_file(file\_path):  names = []  encodings = []  with open(file\_path, 'r') as file:  lines = file.readlines()  current\_name = None  current\_encoding = []  for line in lines:  parts = line.strip().split(': ')  if len(parts) == 2:  if current\_name is not None:  names.append(current\_name)  encodings.append(current\_encoding)  current\_name = parts[0]  current\_encoding = [  float(value) for value in parts[1][1:-1].replace(']', '').split()]  elif current\_name is not None:  current\_encoding.extend(  [float(value) for value in line.strip().replace(']', '').split()])  if current\_name is not None:  names.append(current\_name)  encodings.append(current\_encoding)  return names, encodings |
| --- |

### 4.1.4. Hàm nhận diện giữa ảnh được truyền vào và file mã hóa

Hàm này sẽ tìm kiếm tất cả khuôn mặt trong bức ảnh được truyền vào, xác định vị trí khuôn mặt và mã hóa chúng, sau đó so sánh chúng với file mã hóa để tìm ra khuôn mặt giốn nhất. Nếu đúng thì sẽ in ra tên người đó còn không thì là Unknown.

| def recognize\_faces\_out(image\_path, names, encodings):  # Load the image  image = face\_recognition.load\_image\_file(image\_path)  # Find all face locations and face encodings in the image  face\_locations = face\_recognition.face\_locations(image)  face\_encodings = face\_recognition.face\_encodings(image, face\_locations)  results = []  # Loop through each face found in the image  for (top, right, bottom, left), face\_encoding in zip(face\_locations, face\_encodings):  # Compare the face encoding with the known encodings  matches = face\_recognition.compare\_faces(  encodings, face\_encoding, tolerance=0.5)  name = "Unknown" # Default name if no match is found  # Check if a match is found  if True in matches:  first\_match\_index = matches.index(True)  name = names[first\_match\_index]  # Add result to the list  results.append({  "top": top,  "right": right,  "bottom": bottom,  "left": left,  "name": name  })  # print(results)  if results:  recognized\_name = results[0]['name']  print(recognized\_name) |
| --- |

### 4.1.5. Hàm Main

Đoạn mã này dùng để chạy từ dòng lệnh, và nó kiểm tra các đối số dòng lệnh để xác định hàm nào sẽ được thực thi dựa trên lệnh cụ thể được chỉ định. Hãy phân tích đoạn mã từng bước:

if \_\_name\_\_ == "\_\_main\_\_":: Điều kiện này kiểm tra xem đoạn mã có đang chạy là chương trình chính (không phải là module được import). Nếu đúng, mã bên trong khối này sẽ được thực thi.

Trước tiên, ta cần kiểm tra xem có ít hơn hai đối số dòng lệnh hay không. Chương trình mong đợi ít nhất một đối số dòng lệnh (tên chương trình là đối số đầu tiên), và nếu có ít hơn, nó sẽ in một thông báo sử dụng và thoát với mã trạng thái là 1.

* command = sys.argv[1]: Lấy giá trị của đối số dòng lệnh thứ hai, giả định rằng đó là lệnh chỉ định chức năng nào sẽ được chạy.
* Sử dụng một loạt các câu lệnh if-elif để kiểm tra giá trị của biến command và thực thi hàm tương ứng:
  + if command == "is\_single\_face":: Kiểm tra xem lệnh có phải là "is\_single\_face" không và nếu đúng, gọi hàm is\_single\_face với đối số thứ ba của dòng lệnh (nếu nó tồn tại).
  + elif command == "encode\_images\_in\_dataset":: Kiểm tra xem lệnh có phải là "encode\_images\_in\_dataset" không và nếu đúng, gọi hàm encode\_images\_in\_dataset với các đối số cần thiết là đường dẫn dataset, đường dẫn encodings, và đường dẫn trainer.
  + elif command == "recognize\_faces":: Kiểm tra xem lệnh có phải là "recognize\_faces" không và nếu đúng, gọi hàm recognize\_faces với đường dẫn hình ảnh, đường dẫn encodings, và đường dẫn savefile.
  + elif command == "recognize\_faces\_out":: Kiểm tra xem lệnh có phải là "recognize\_faces\_out" không và nếu đúng, gọi hàm recognize\_faces\_out với đường dẫn hình ảnh và đường dẫn encodings.

Nếu không có lệnh nào khớp, chương trình in ra "Unknown command" và thoát với mã trạng thái là 1.

| if \_\_name\_\_ == "\_\_main\_\_":  if len(sys.argv) < 2:  print("Usage: python main.py command [options]")  sys.exit(1)  command = sys.argv[1]  if command == "is\_single\_face":  if len(sys.argv) > 2:  image\_path = sys.argv[2]  is\_single\_face(image\_path)  elif command == "encode\_images\_in\_dataset":  if len(sys.argv) > 2:  dataset\_path = sys.argv[2]  encodings\_path = sys.argv[3]  trainer\_path = sys.argv[4]  encode\_images\_in\_dataset(  dataset\_path, encodings\_path, trainer\_path)  elif command == "recognize\_faces":  if len(sys.argv) > 2:  image\_path = sys.argv[2]  encodings\_path = sys.argv[3]  savefile\_path = sys.argv[4]  names, encodings = load\_encoding\_file(encodings\_path)  recognize\_faces(image\_path, names, encodings, savefile\_path)  elif command == "recognize\_faces\_out":  if len(sys.argv) > 2:  image\_path = sys.argv[2]  encodings\_path = sys.argv[3]  names, encodings = load\_encoding\_file(encodings\_path)  recognize\_faces\_out(image\_path, names, encodings)  else:  print("Unknown command")  sys.exit(1) |
| --- |

## 4.2. Chương trình winform

Ta cần khai báo một số thư viện dùng trong chương trình

* **System.Drawing**: Thư viện này cung cấp các lớp để làm việc với đồ họa, chẳng hạn như hình ảnh và vẽ đồ họa.
* **System.IO**: Thư viện này cung cấp các lớp để thực hiện các thao tác đầu vào và đầu ra, chẳng hạn như đọc và ghi file.
* **System.Diagnostics**: Thư viện này cung cấp các lớp để quản lý và tương tác với quá trình và thông tin về hệ thống.
* **System.Linq**: Thư viện này hỗ trợ Language Integrated Query (LINQ), là một tập hợp các phương pháp mở rộng cho ngôn ngữ C#.
* **System.Text**: Thư viện này cung cấp các lớp để làm việc với chuỗi và dữ liệu văn bản.
* **System.Threading** và **System.Threading.Task**s: Cung cấp các lớp và phương thức để thực hiện đa luồng (multithreading) và công việc bất đồng bộ (asynchronous).
* **System.Windows.Forms**: Thư viện này cung cấp các lớp và thành phần để xây dựng giao diện người dùng trên nền tảng Windows.
* **System.Drawing.Imaging**: Cung cấp các lớp để làm việc với định dạng hình ảnh và thực hiện các thao tác xử lý hình ảnh.
* **AForge.Video.DirectShow** và **AForge.Video**: Là một thư viện AForge.NET, mà có thể được sử dụng để xử lý và làm việc với dữ liệu video, chẳng hạn như làm việc với webcam.

| using System;  using System.Collections.Generic;  using System.Drawing;  using System.IO;  using System.Diagnostics;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  using System.Drawing.Imaging;  using AForge.Video.DirectShow;  using AForge.Video;  using System.Threading; |
| --- |

Tiếp theo, cần khai báo một số biến sử dụng:

| private string selectedFilePathTrain = null;  private string selectedFilePathRecognize = null;  private string pythonExePath = @"C:/Users/Admin/AppData/Local/Microsoft/WindowsApps/python3.12.exe";  string pythonScriptPath = @"../../python/main.py";  string currentDirectory = Environment.CurrentDirectory;  private string tempFolderPath = Path.Combine(Environment.CurrentDirectory, "temp");  private bool isCameraRunning = false;  private bool isRecognizing = false;  private FilterInfoCollection videoDevices;  private VideoCaptureDevice videoSource; |
| --- |

### 4.2.1. Nhóm hàm liên quan đến việc đóng mở form, hiển thị lên pictureBox, gọi lõi python

#### **4.2.1.1 Hàm MainForm\_Load (Mở form)**

| private void MainForm\_Load(object sender, EventArgs e)  {  // Tạo một FilterInfoCollection để lưu danh sách các thiết bị đầu vào video  videoDevices = new FilterInfoCollection(FilterCategory.VideoInputDevice);  // Kiểm tra nếu không có camera nào được tìm thấy  if (videoDevices.Count == 0)  {  // Hiển thị thông báo và tắt nút Camera nếu không có camera  MessageBox.Show("Không tìm thấy camera.");  btnCamera.Enabled = false; // Tắt nút nếu không tìm thấy camera  return; // Thoát khỏi sự kiện nếu không có camera  }  // Kiểm tra xem thư mục tạm có tồn tại không  if (!Directory.Exists(tempFolderPath))  {  // Tạo thư mục tạm nếu không tồn tại  Directory.CreateDirectory(tempFolderPath);  }  } |
| --- |

#### **4.2.1.2. Hàm MainForm\_Closing (Đóng form)**

| private void MainForm\_FormClosing(object sender, FormClosingEventArgs e)  {  // Dừng camera khi đóng form  if (videoSource != null && videoSource.IsRunning)  {  videoSource.SignalToStop();  videoSource.WaitForStop();  }  } |
| --- |

#### **4.2.1.3. Hàm RunPythonScript (Chạy biên dịch python)**

| private string RunPythonScript(string cmd, string args)  {  ProcessStartInfo start = new ProcessStartInfo  {  FileName = cmd,  Arguments = args,  UseShellExecute = false,  RedirectStandardOutput = true,  CreateNoWindow = true // Không hiển thị cửa sổ dòng lệnh  };  // Khởi chạy tiến trình  using (Process process = Process.Start(start))  {  // Đọc đầu ra tiêu chuẩn của tiến trình  using (StreamReader reader = process.StandardOutput)  {  string result = reader.ReadToEnd();  // Hiển thị đầu ra trên màn hình console  Console.WriteLine(result);  return result;  }  }  } |
| --- |

#### **4.2.1.4. Hàm DisplayImageInPictureBox (Hiển thị ảnh lên hộp hình ảnh)**

| private void DisplayImageInPictureBox(string file\_path, PictureBox PicBox)  {  // Đọc ảnh từ tệp  Image originalImage = Image.FromFile(file\_path);  // Kích thước của picBoxTrain  int picBoxWidth = PicBox.Width;  int picBoxHeight = PicBox.Height;  // Tính toán chiều rộng và chiều cao mới dựa trên kích thước của picBoxTrain  int newWidth = originalImage.Width;  int newHeight = originalImage.Height;  if (newWidth > picBoxWidth)  {  newWidth = picBoxWidth;  newHeight = (int)((double)newWidth / originalImage.Width \* originalImage.Height);  }  if (newHeight > picBoxHeight)  {  newHeight = picBoxHeight;  newWidth = (int)((double)newHeight / originalImage.Height \* originalImage.Width);  }  // Tạo ảnh mới với kích thước mới  Image resizedImage = ResizeImage(originalImage, newWidth, newHeight);  // Tính toán vị trí để căn giữa ảnh trong picBoxTrain  int x = (picBoxWidth - newWidth) / 2;  int y = (picBoxHeight - newHeight) / 2;  // Tạo một Bitmap mới với kích thước của picBoxTrain  Bitmap centeredBitmap = new Bitmap(picBoxWidth, picBoxHeight);  // Tạo một đối tượng Graphics để vẽ ảnh mới vào vị trí giữa  using (Graphics g = Graphics.FromImage(centeredBitmap))  {  g.DrawImage(resizedImage, x, y, newWidth, newHeight);  }  // Hiển thị ảnh trong PictureBox  PicBox.Image = centeredBitmap;  } |
| --- |

#### 

### 4.2.2. Nhóm hàm liên quan đến việc huấn luyện

#### **4.2.2.1. Hàm RunIsSingleFaceFunction (Kiểm tra có đúng 1 khuôn mặt trong bức ảnh hay không?)**

| private bool RunIsSingleFaceFunction(string imagePath)  {  string fullPathPythonScript = Path.Combine(currentDirectory, pythonScriptPath).Replace('\\', '/');  string result = RunPythonScript(pythonExePath, $"{fullPathPythonScript} is\_single\_face \"{imagePath}\"");  if (Convert.ToBoolean(result) == false)  {  MessageBox.Show($"Vui lòng chọn bức ảnh chỉ có 1 khuôn mặt");  return Convert.ToBoolean(result);  }  else  {  // Tiến hành lưu vào Dataset  string DatasetPath = Path.Combine(currentDirectory, "../../Datasets").Replace('\\', '/');  string filePath = Path.Combine(DatasetPath, $"{txbName.Text}.png");  Bitmap originalImage = new Bitmap(ofdTrain.FileName);  originalImage.Save(filePath, System.Drawing.Imaging.ImageFormat.Png);  return Convert.ToBoolean(result);  }  } |
| --- |

#### **4.2.2.2. Hàm RunEncodeImagesInDataset (Huấn luyện khuôn mặt trong Dataset)**

| private void RunEncodeImagesInDataset()  {  string fullPathPythonScript = Path.Combine(currentDirectory, pythonScriptPath).Replace('\\', '/');  string datasetPath = Path.Combine(currentDirectory, "../../Datasets").Replace('\\', '/');  string encodingFilePath = Path.Combine(currentDirectory, "../../Models/encodings.txt").Replace('\\', '/');  string trainedPath = $"{datasetPath}/Trained";  RunPythonScript(pythonExePath, $"\"{fullPathPythonScript}\" encode\_images\_in\_dataset \"{datasetPath}\" \"{encodingFilePath}\" \"{trainedPath}\"");  MessageBox.Show("Đã huấn luyện xong", "Thông báo", MessageBoxButtons.OK, MessageBoxIcon.Information);  } |
| --- |

### 4.2.3. Nhóm hàm liên quan đến việc nhận diện

#### **4.2.3.1. Hàm videoSource\_NewFrame (Xử lý sự kiện khi một khung hình mới được tạo ra từ Video)**

| private void videoSource\_NewFrame(object sender, NewFrameEventArgs eventArgs)  {  // Lấy khung hình từ camera  Bitmap frame = (Bitmap)eventArgs.Frame.Clone();  // Lật ngang khung hình  frame.RotateFlip(RotateFlipType.RotateNoneFlipX);  // Hiển thị khung hình đã lật ngang trên PictureBox với chế độ Zoom  picBoxRecognize.SizeMode = PictureBoxSizeMode.Zoom;  picBoxRecognize.Image = frame;  } |
| --- |

#### **4.2.3.2. Hàm ResizeImage (Chỉnh sửa kích thước ảnh đầu vào)**

| private Image ResizeImage(Image image, int maxWidth, int maxHeight)  {  float aspectRatio = (float)image.Width / image.Height;  // Tính toán kích thước mới dựa trên tỷ lệ và giới hạn kích thước  int calculatedWidth = Math.Min(maxWidth, (int)(maxHeight \* aspectRatio));  int calculatedHeight = Math.Min(maxHeight, (int)(maxWidth / aspectRatio));  // Kiểm tra nếu kích thước mới vượt quá kích thước ban đầu  if (calculatedWidth > image.Width || calculatedHeight > image.Height)  {  calculatedWidth = image.Width;  calculatedHeight = image.Height;  }  // Tạo một Bitmap mới với kích thước mới  Bitmap resizedBitmap = new Bitmap(calculatedWidth, calculatedHeight);  // Tạo một đối tượng Graphics để vẽ ảnh mới  using (Graphics g = Graphics.FromImage(resizedBitmap))  {  // Vẽ ảnh mới với kích thước mới và giữ nguyên tỷ lệ  g.DrawImage(image, 0, 0, calculatedWidth, calculatedHeight);  }  return resizedBitmap;  } |
| --- |

#### **4.2.3.3. Hàm RunRecognition (Chạy nhận diện)**

| private void RunRecognition(Image imgPath)  {  DateTime startTime = DateTime.Now;  string fullPathPythonScript = Path.Combine(currentDirectory, pythonScriptPath).Replace('\\', '/');  string encodingFilePath = Path.Combine(currentDirectory, "../..", "Models", "encodings.txt").Replace('\\', '/');  Image resizedImage = ResizeImage(imgPath, 512, 300);  // Lưu ảnh đã resize vào một tệp tin tạm thời  string resizedImagePath = Path.Combine(currentDirectory, "resized\_image.jpg");  resizedImage.Save(resizedImagePath, ImageFormat.Jpeg);  string text = RunPythonScript(pythonExePath, $"\"{fullPathPythonScript}\" recognize\_faces\_out \"{resizedImagePath}\" \"{encodingFilePath}\"");  // Sử dụng phương thức Invoke để thay đổi thuộc tính Text của txbRecognize trên UI thread  if (txbRecognize != null && !txbRecognize.IsDisposed && !txbRecognize.Disposing)  {  // Kiểm tra xem có cần phải Invoke để thực hiện trên UI thread không  if (txbRecognize.InvokeRequired)  {  // Nếu cần Invoke, thực hiện thay đổi trên UI thread  txbRecognize.Invoke(new MethodInvoker(delegate  {  // Kiểm tra lại trạng thái của txbRecognize để tránh lỗi  if (txbRecognize != null && !txbRecognize.IsDisposed && !txbRecognize.Disposing)  {  // Thay đổi thuộc tính Text của txbRecognize  txbRecognize.Text = text;  }  }));  }  else  {  // Nếu không cần Invoke, thực hiện thay đổi trực tiếp trên luồng hiện tại  txbRecognize.Text = text;  }  }  DateTime endTime = DateTime.Now;  TimeSpan processingTime = endTime - startTime;  // MessageBox.Show($"Thời gian xử lý: {processingTime.TotalSeconds} giây", "Thông tin", MessageBoxButtons.OK, MessageBoxIcon.Information);  } |
| --- |

### 4.2.4. Nhóm hàm xử lý sự kiện click

#### **4.2.4.1. Hàm btnOpenImageTrain\_Click (Nhấn nút mở chọn ảnh huấn luyện)**

| private void btnOpenImageTrain\_Click(object sender, EventArgs e)  {  // Thiết lập các thuộc tính cho OpenFileDialog  ofdTrain.Title = "Chọn ảnh đào tạo"; // Tiêu đề của hộp thoại  ofdTrain.Filter = "Ảnh|\*.jpg;\*.jpeg;\*.png;\*.gif";  ofdTrain.InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.MyPictures); // Thư mục mặc định khi mở hộp thoại  // Hiển thị hộp thoại mở tệp và xác nhận rằng người dùng đã chọn một tệp  if (ofdTrain.ShowDialog() == DialogResult.OK)  {  // Lấy đường dẫn đầy đủ của tệp đã chọn  selectedFilePathTrain = ofdTrain.FileName;  try  {  DisplayImageInPictureBox(selectedFilePathTrain, picBoxTrain);  }  catch (Exception ex)  {  MessageBox.Show("Không thể mở ảnh: " + ex.Message, "Lỗi", MessageBoxButtons.OK, MessageBoxIcon.Error);  }  }  } |
| --- |

#### **4.2.4.2. Hàm btnTrain\_Click (Nhấn nút huấn luyện)**

| private void btnTrain\_Click(object sender, EventArgs e)  {  // Kiểm tra xem txbName đã có dữ liệu hay không  if (string.IsNullOrEmpty(txbName.Text))  {  MessageBox.Show("Vui lòng nhập tên trước khi tiếp tục.", "Lỗi", MessageBoxButtons.OK, MessageBoxIcon.Error);  return; // Ngừng thực hiện khi txbName không có dữ liệu  }  // Kiểm tra xem người dùng đã chọn ảnh hay không  if (selectedFilePathTrain == null)  {  MessageBox.Show("Vui lòng chọn ảnh trước khi tiếp tục.", "Lỗi", MessageBoxButtons.OK, MessageBoxIcon.Error);  return; // Ngừng thực hiện khi người dùng không chọn ảnh  }  else  {  bool checkSingleFace = RunIsSingleFaceFunction(selectedFilePathTrain);  if (checkSingleFace)  {  RunEncodeImagesInDataset();  }  }  } |
| --- |

#### **4.2.4.3. Hàm btnOpenCamera\_Click (Nhấn nút mở camera)**

| private void btnOpenCamera\_Click(object sender, EventArgs e)  {  if (isCameraRunning)  {  // Stop the camera  if (videoSource != null && videoSource.IsRunning)  {  videoSource.Stop();  picBoxRecognize.Image = null; // Clear the PictureBox  }  isCameraRunning = false;  btnCamera.Text = "Mở Camera"; // Change button text  }  else  {  // Start the camera  if (videoDevices == null || videoDevices.Count == 0)  {  MessageBox.Show("Không tìm thấy camera hoặc videoDevices chưa được khởi tạo.");  return;  }  // Select the first camera  videoSource = new VideoCaptureDevice(videoDevices[0].MonikerString);  videoSource.NewFrame += new NewFrameEventHandler(videoSource\_NewFrame);  // Start displaying the camera  videoSource.Start();  isCameraRunning = true;  btnCamera.Text = "Tắt Camera"; // Change button text  }  } |
| --- |

#### **4.2.4.4. Hàm btnOpenImageRecognize\_Click (Nhấn nút mở ảnh nhận dạng)**

| private void btnOpenImageRecognize\_Click(object sender, EventArgs e)  {  txbRecognize.Text = null;  ofdRecognize.Title = "Chọn ảnh đào tạo";  ofdRecognize.Filter = "Ảnh|\*.jpg;\*.jpeg;\*.png;\*.gif";  // Hiển thị hộp thoại mở tệp và xác nhận rằng người dùng đã chọn một tệp  if (ofdRecognize.ShowDialog() == DialogResult.OK)  {  // Lấy đường dẫn đầy đủ của tệp đã chọn  selectedFilePathRecognize = ofdRecognize.FileName;  try  {  DisplayImageInPictureBox(selectedFilePathRecognize, picBoxRecognize);  }  catch (Exception ex)  {  MessageBox.Show("Không thể mở ảnh: " + ex.Message, "Lỗi", MessageBoxButtons.OK, MessageBoxIcon.Error);  }  }  } |
| --- |

#### **4.2.4.5. Hàm btnRecognize\_Click (Nhấn nút nhận dạng)**

| private void btnRecognize\_Click(object sender, EventArgs e)  {  if (isCameraRunning)  {  // Bắt đầu quá trình chụp và nhận diện liên tục  isRecognizing = true;  // Tạo một luồng mới để thực hiện chụp ảnh và nhận diện liên tục  Task.Run(() =>  {  while (isRecognizing && isCameraRunning)  {  // Tạo tên file ngẫu nhiên trong thư mục temp  string tempFilePath = Path.Combine(tempFolderPath, Guid.NewGuid().ToString() + ".jpg");  // Chụp ảnh từ camera và lưu vào thư mục temp  if (videoSource != null && videoSource.IsRunning)  {  // Lấy frame từ camera và tạo một bản sao  Bitmap frame = null;  if (picBoxRecognize.InvokeRequired)  {  picBoxRecognize.Invoke(new MethodInvoker(() =>  {  frame = (Bitmap)picBoxRecognize.Image.Clone();  }));  }  else  {  frame = (Bitmap)picBoxRecognize.Image.Clone();  }  // Lưu frame vào thư mục temp  if (frame != null)  {  frame.Save(tempFilePath, ImageFormat.Jpeg);  // Giải phóng frame sau khi sử dụng  frame.Dispose();  }  }  // Nhận diện từ ảnh vừa chụp  if (!string.IsNullOrEmpty(tempFilePath))  {  Image img = Image.FromFile(tempFilePath);  RunRecognition(img);  }  // Thêm thời gian nghỉ để tránh quá tải hệ thống (có thể điều chỉnh)  Thread.Sleep(1000);  }  });  }  else if (selectedFilePathRecognize != null)  {  Image img = Image.FromFile(selectedFilePathRecognize);  RunRecognition(img);  }  else  {  MessageBox.Show("Vui lòng mở camera hoặc chọn ảnh trước khi nhận diện.", "Lỗi", MessageBoxButtons.OK, MessageBoxIcon.Error);  // Dừng quá trình nhận diện liên tục khi camera đóng  isRecognizing = false;  }  } |
| --- |

#### 

#### 