BÁO CÁO CÔNG VIỆC

**Công việc số:** 57

**Mô tả công việc:** Tính số phương tiện giao thông có trong khung hình video, camera (Tiếp tục công việc đã hoàn thành tại Job 51)

**Người thực hiện:** Hạ Quang Dũng

**Ngày bắt đầu:** 01/11/2024

**Ngày kết thúc:** 12/11/2024

**NỘI DUNG TÀI LIỆU**

[**1. Giới thiệu 2**](#_7rle0zq8bvwd)

[**2. Các công nghệ sử dụng 2**](#_g1q223pmxw09)

[**3. Cài đặt 2**](#_brksv8pp6xc0)

[3.1. Yêu cầu hệ thống 3](#_4j1qeq2uf40m)

[3.2. Các bước cài đặt 3](#_nbearyo8u8cs)

[**4. Cấu trúc dự án 3**](#_7joa0vt5rl70)

[**5. Chi tiết mã nguồn 4**](#_5ht1u6l0ivnf)

[5.1. VehicleDetection.src.CSharp.Models > DetectionResult.cs 4](#_uvzdwjzc23hd)

[5.2. VehicleDetection.src.CSharp.Services > Helper.cs 4](#_8mg1z4djepdq)

[5.3. VehicleDetection.src.CSharp.Services > ImageExtractor.cs 5](#_svkze6f2snr6)

[5.4. VehicleDetection.src.CSharp.Services > PythonExcutor.cs 7](#_2uwnsyqvd8we)

[5.5. VehicleDetection.src.CSharp.Services > YoloV8.cs 8](#_ah3b0m41kup)

[5.6. VehicleDetection.src.CSharp.Views > Form1.cs 16](#_xjeuw5grndj1)

[**5.7. Detect.py 22**](#_76y0vcsa7ix4)

[**6. Hướng dẫn sử dụng 23**](#_sibuj4l6imh1)

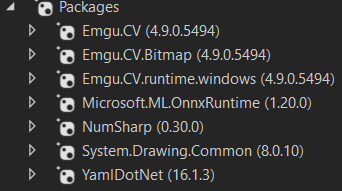
# 1. Giới thiệu

Dự án này có mục tiêu **trích xuất frame từ video** và **nhận dạng phương tiện** từ các hình ảnh đã được trích xuất. Hệ thống cung cấp hai lựa chọn nhận dạng: **PyTorch** và **ONNX**, cho phép lựa chọn giữa mô hình huấn luyện sử dụng PyTorch hoặc sử dụng ONNX để tăng tốc độ xử lý. Dự án có thể ứng dụng trong các lĩnh vực như giám sát giao thông, phân tích video giao thông, hoặc các dự án nghiên cứu liên quan đến nhận dạng và đếm phương tiện.

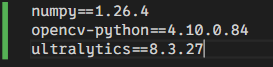
# 2. Các công nghệ sử dụng

* **C# và WinForms**: Xây dựng giao diện người dùng cho phép người dùng chọn video, trích xuất frame, và hiển thị kết quả nhận dạng phương tiện.
* **Emgu.CV**: Sử dụng thư viện xử lý hình ảnh dựa trên OpenCV trong C# để trích xuất frame từ video.
* **Python**: Chạy mô hình nhận dạng phương tiện.
* **PyTorch** và **ONNX Runtime**: Cung cấp hai cách lựa chọn nhận dạng:
  + PyTorch với mô hình YOLO (You Only Look Once) để nhận dạng phương tiện.
  + ONNX giúp chuyển đổi mô hình PyTorch sang định dạng ONNX để tăng tốc độ suy luận và khả năng tích hợp với các nền tảng khác.

Các thư viện được cài đặt trong C#:



Các thư viện chính được cài đặt trong Python



# 3. Cài đặt

## 3.1. Yêu cầu hệ thống

* **Windows OS** (đã cài đặt .NET Framework để chạy ứng dụng WinForms).
* **Python 3.7+** với các thư viện cần thiết (xem phần dưới).
* **ONNX Runtime**.

## 3.2. Các bước cài đặt

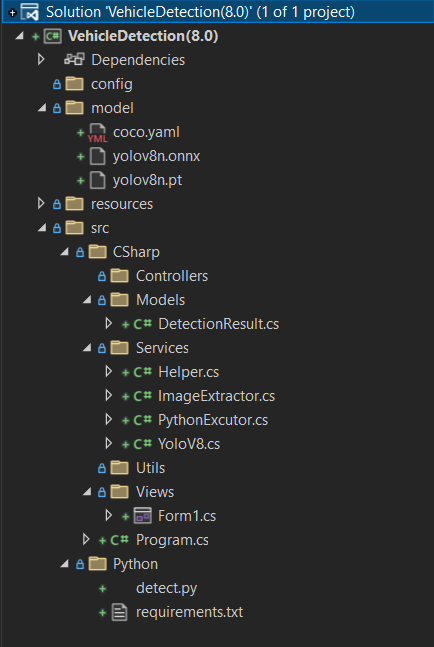
**Cài đặt thư viện cho Python:** pip install -r requirements.txt

requirements.txt bao gồm các plaintext như đã nêu trên mục 2

**Xây dựng ứng dụng WinForms (C#)**

* Mở dự án trong **Visual Studio**.
* Cài đặt các thư viện .NET cần thiết như đã nêu tại mục 2

# 4. Cấu trúc dự án



# 5. Chi tiết mã nguồn

## 5.1. VehicleDetection.src.CSharp.Models > DetectionResult.cs

| using System.Collections.Generic; using System.Text.Json.Serialization;  namespace VehicleDetection.src.CSharp.Models {  public class DetectionResult  {  [JsonPropertyName("total\_time")]  public double TotalTime { get; set; } // Thời gian xử lý tổng   [JsonPropertyName("total\_vehicles")]  public int TotalVehicles { get; set; } // Tổng số phương tiện   [JsonPropertyName("vehicle\_counts")]  public Dictionary<string, int> VehicleCounts { get; set; } // Số lượng các loại phương tiện  } } |
| --- |

## 5.2. VehicleDetection.src.CSharp.Services > Helper.cs

| namespace VehicleDetection.src.CSharp.Services {  public static class Helper  {  public static void DeleteImagesInDirectory(string directoryPath)  {  if (!Directory.Exists(directoryPath))  {  Console.WriteLine("Thư mục không tồn tại.");  return;  }   var imageExtensions = new[] { ".jpg", ".png", ".jpeg", ".gif" };  foreach (var file in Directory.GetFiles(directoryPath))  {  if (Array.Exists(imageExtensions, ext => file.EndsWith(ext, StringComparison.OrdinalIgnoreCase)))  {  File.Delete(file);  Console.WriteLine($"Đã xóa: {file}");  }  }  Console.WriteLine("Hoàn thành việc xóa ảnh.");  }  } } |
| --- |

## 5.3. VehicleDetection.src.CSharp.Services > ImageExtractor.cs

| using Emgu.CV; using Emgu.CV.CvEnum;  namespace VehicleDetection.src.CSharp.Services {  public class ImageExtractor  {  private readonly string \_outputImagePath;   // Constructor nhận đường dẫn để lưu các ảnh trích xuất được  public ImageExtractor(string outputImagePath)  {  \_outputImagePath = outputImagePath;  }   // Phương thức trích xuất ảnh từ video  public void ExtractImages(string videoPath, int frameSkip = 1)  {  // Kiểm tra video có tồn tại không  if (!File.Exists(videoPath))  {  Console.WriteLine("Video file not found!");  return;  }   // Mở video  using (var capture = new VideoCapture(videoPath))  {  // Lấy tổng số frame trong video  int totalFrames = (int)capture.Get(CapProp.FrameCount);  int frameCount = 0;   // Đảm bảo thư mục lưu ảnh đã tồn tại  if (!Directory.Exists(\_outputImagePath))  {  Directory.CreateDirectory(\_outputImagePath);  }   // Lặp qua từng frame của video  while (frameCount < totalFrames)  {  // Đọc một frame từ video  Mat frame = new Mat();  capture.Read(frame);   // Kiểm tra nếu không có frame nào để đọc (end of video)  if (frame.IsEmpty)  {  break;  }   // Nếu frameCount chia hết cho frameSkip thì lưu frame  if (frameCount % frameSkip == 0)  {  SaveFrame(frame, frameCount);  }   frameCount++;  }   Console.WriteLine("Complete!");  }  }   // Phương thức lưu frame dưới dạng ảnh  private void SaveFrame(Mat frame, int frameCount)  {  // Tạo tên tệp cho ảnh  string fileName = Path.Combine(\_outputImagePath, $"frame\_{frameCount:D6}.jpg");   // Lưu frame thành ảnh  frame.Save(fileName);  Console.WriteLine($"Frame {frameCount} saved to {fileName}");  }  } } |
| --- |

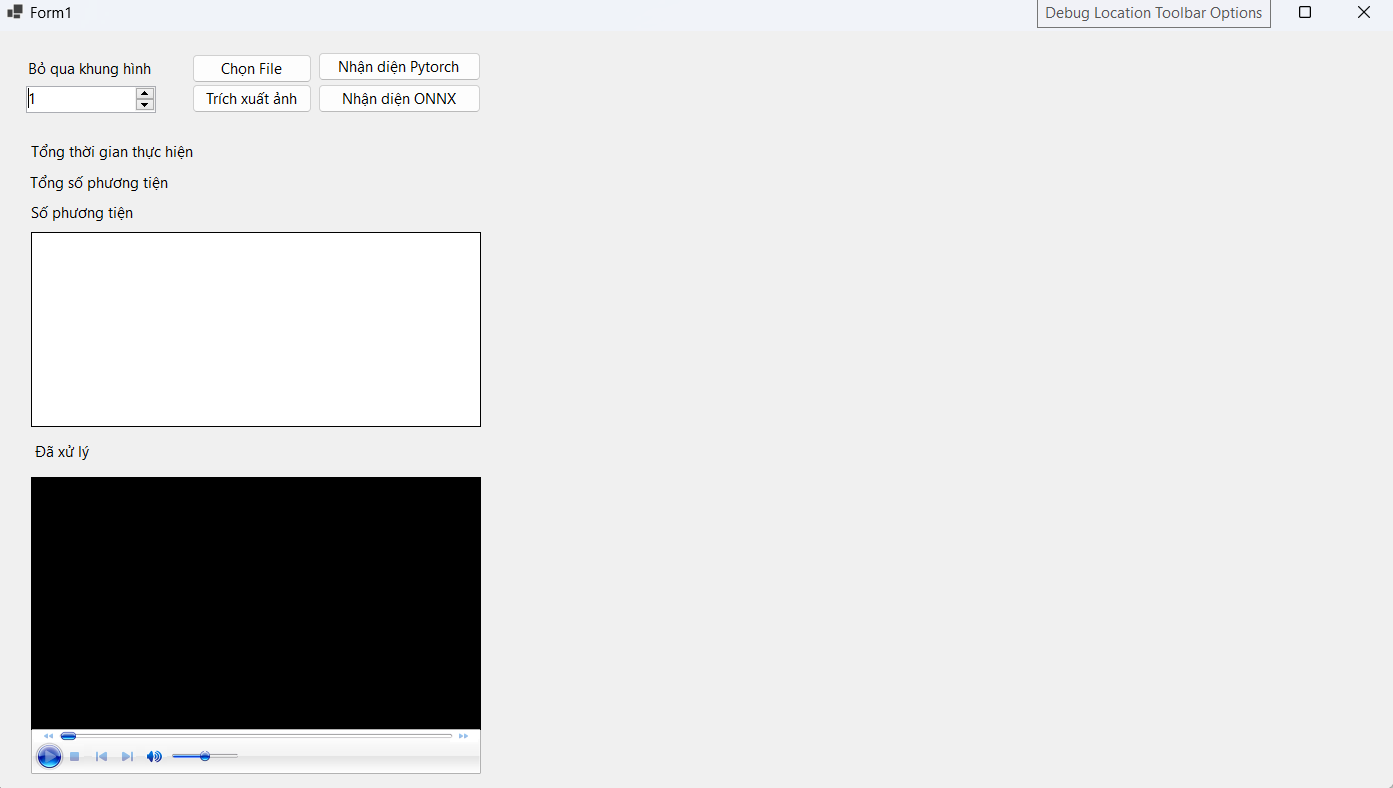
## 5.4. VehicleDetection.src.CSharp.Services > PythonExcutor.cs

| using System.Diagnostics; namespace VehicleDetection.src.CSharp.Services {  public class PythonExecutor  {  private string pythonPath; // Đường dẫn đến interpreter Python  private string scriptPath; // Đường dẫn đến script Python   public PythonExecutor(string pythonPath, string scriptPath)  {  this.pythonPath = pythonPath;  this.scriptPath = scriptPath;  }   public string Execute(string args)  {  var startInfo = new ProcessStartInfo  {  FileName = pythonPath,  Arguments = $"{scriptPath} {args}", // Thêm các đối số cần thiết  RedirectStandardOutput = true,  UseShellExecute = false,  CreateNoWindow = true  };   using (var process = Process.Start(startInfo))  {  using (var reader = process.StandardOutput)  {  string result = reader.ReadToEnd();  process.WaitForExit();  return result;  }  }  }  } } |
| --- |

## 5.5. VehicleDetection.src.CSharp.Services > YoloV8.cs

| using System; using System.Collections.Generic; using System.Drawing.Imaging; using System.Drawing; using System.IO; using System.Linq; using System.Text; using System.Threading.Tasks; using Emgu.CV; using Emgu.CV.Dnn; using Emgu.CV.Structure; using Microsoft.ML.OnnxRuntime; using Microsoft.ML.OnnxRuntime.Tensors; using NumSharp; using YamlDotNet.RepresentationModel;  namespace VehicleDetection.src.CSharp.Services {  public class YoloV8  {  private Dictionary<int, string> classNames;  private static int test = 0;  private static int imgWidth, imgHeight;  private string \_modelPath;  private InferenceSession session;  private List<int> \_modelInput = new List<int>();  private List<int> \_modelOutput = new List<int>();  private float \_confidenceThreshold;  private float \_iouThreshold;  Dictionary<string, int> objectKeyValue = new Dictionary<string, int>();   public YoloV8(string modelPath, string yamlFilePath, float confidenceThreshold = 0.2f, float iouThreshold = 0.5f)  {  \_modelPath = modelPath;  session = new InferenceSession(modelPath);  foreach (var input in session.InputMetadata)  {  var inputInfo = input.Value;  var inputShape = inputInfo.Dimensions.Select(d => d.ToString()).ToArray(); // Chuyển thành mảng để dễ xử lý   // Duyệt qua từng phần tử của mảng inputShape và thêm vào list  foreach (var shapeElement in inputShape)  {  if (int.TryParse(shapeElement, out int dimension))  {  \_modelInput.Add(dimension);  }  }  }  foreach (var output in session.OutputMetadata)  {  var outputInfo = output.Value;  var outputShape = outputInfo.Dimensions.Select(d => d.ToString()).ToArray();  // Duyệt qua từng phần tử của mảng inputShape và thêm vào list  foreach (var shapeElement in outputShape)  {  if (int.TryParse(shapeElement, out int dimension))  {  \_modelOutput.Add(dimension);   }  }  }  // Tải các tên class từ file YAML  classNames = LoadClassNamesFromYaml(yamlFilePath);  \_confidenceThreshold = confidenceThreshold;  \_iouThreshold = iouThreshold;  }   public Dictionary<string, int> Detect(string imagePath, string outputImagePath)  {  objectKeyValue.Clear();   // Tiền xử lý ảnh  int inputWidth = \_modelInput[\_modelInput.Count - 2];  int inputHeight = \_modelInput[\_modelInput.Count - 1];  int outputdata1 = \_modelOutput[0];  int outputdata2 = \_modelOutput[1];  int outputdata3 = \_modelOutput[2];  var ndArray = PreprocessImage(imagePath, inputWidth, inputHeight);    // Chuyển NDArray thành DenseTensor<float>  var tensor = ConvertToTensor(ndArray);   // Tạo một Dictionary để chứa tensor đầu vào  var inputs = new[] { NamedOnnxValue.CreateFromTensor("images", tensor) };   // Chạy mô hình và nhận kết quả  using var results = session.Run(inputs);   Tensor<float> outputTensor = results[0].AsTensor<float>();   // Chuyển đổi sang ndarray sử dụng NumSharp  var npOutput = np.array(outputTensor).reshape(outputdata1, outputdata2, outputdata3);   // Thực hiện squeeze để loại bỏ chiều đầu tiên (1)  var squeezedOutput = np.squeeze(npOutput);   // Thực hiện transpose để hoán đổi trục   var transposedOutput = np.transpose(squeezedOutput);   // Gọi hàm Postprocess với dữ liệu đầu ra từ mô hình  Postprocess(imagePath, transposedOutput, imgWidth, imgHeight, 0.5f, 0.45f, outputImagePath);  return this.objectKeyValue;  }   private static DenseTensor<float> ConvertToTensor(NDArray ndArray)  {  var shape = ndArray.shape;  var data = ndArray.Data<float>(); // Lấy dữ liệu float từ NDArray   // Chuyển NDArray thành DenseTensor<float> với kích thước tương ứng  return new DenseTensor<float>(data.ToArray(), shape);  }    private static NDArray PreprocessImage(string imagePath, int inputWidth, int inputHeight)  {  Mat img = new Mat();  img = CvInvoke.Imread(imagePath);  // Lấy kích thước ban đầu của ảnh  imgWidth = img.Width;  imgHeight = img.Height;    // Chuyển đổi không gian màu từ BGR sang RGB (tương đương cv2.cvtColor)  Mat imgRgb = new Mat();  CvInvoke.CvtColor(img, imgRgb, Emgu.CV.CvEnum.ColorConversion.Bgr2Rgb);   // Thay đổi kích thước ảnh theo input shape (tương tự cv2.resize)  Mat resizedImage = new Mat();  CvInvoke.Resize(imgRgb, resizedImage, new Size(inputWidth, inputHeight));   // Tạo NDArray để lưu dữ liệu ảnh  var imageData = np.zeros(new int[] { inputHeight, inputWidth, 3 }, np.float32);  var imgdata = resizedImage.ToImage<Rgb, byte>();  // Duyệt qua các pixel và chuyển đổi BGR sang RGB  for (int j = 0; j < inputHeight; j++)  {  for (int i = 0; i < inputWidth; i++)  {  // lấy giá trị pixel tại vị trí (i, j)  var pixel = imgdata[i, j];  imageData[i, j, 0] = pixel.Red / 255.0f;  imageData[i, j, 1] = pixel.Green / 255.0f;  imageData[i, j, 2] = pixel.Blue / 255.0f;   }  }   // Transpose ảnh để đưa kênh màu lên đầu, giống với cách xử lý của OpenCV  imageData = np.transpose(imageData, new int[] { 2, 0, 1 }); // Channel first   // Thêm chiều mới ở vị trí đầu tiên để phù hợp với input (expand\_dims)  imageData = np.expand\_dims(imageData, axis: 0).astype(np.float32);   // In hình dạng của dữ liệu đầu vào (giống với Python)  Console.WriteLine("Input shape: " + string.Join(", ", imageData.shape));   return imageData;  }   private void Postprocess(string imagePath, NDArray outputData, int imgWidth, int imgHeight, float confidenceThres, float iouThres, string outputImagePath)  {  // Tính toán hệ số tỷ lệ cho tọa độ hộp giới hạn  int inputWidth = \_modelInput[\_modelInput.Count - 2]; // Chiều rộng của ảnh đầu vào cho mô hình  int inputHeight = \_modelInput[\_modelInput.Count - 1]; // Chiều cao của ảnh đầu vào cho mô hình   float xFactor = (float)imgWidth / inputWidth;  float yFactor = (float)imgHeight / inputHeight;   var boxes = new List<Rectangle>();    var scores = new List<float>();  var classIds = new List<int>();   // Xác định số lượng hàng và cột của dữ liệu đầu ra  var shape = outputData.shape;  int rows = shape[0];  int cols = shape[1];  int count = 0;  for (int i = 0; i < rows; i++)  {  // Extract the current row  var row = outputData[i, ":"];   // Confidence score  float confidence = row[4].GetValue<float>();    // Find class with the highest score  float maxScore = 0;  int classId = -1;   for (int j = 4; j < cols; j++)  {  float score = row[j].GetValue<float>();  if (score > maxScore)  {  maxScore = score;  classId = j - 4;  }  }   if (maxScore >= confidenceThres)  {  count++;  // Compute bounding box coordinates  float x = row[0].GetValue<float>();  float y = row[1].GetValue<float>();  float w = row[2].GetValue<float>();  float h = row[3].GetValue<float>();   int left = (int)((x - w / 2) \* xFactor);  int top = (int)((y - h / 2) \* yFactor);  int width = (int)(w \* xFactor);  int height = (int)(h \* yFactor);   boxes.Add(new Rectangle(left, top, width, height));  scores.Add(maxScore);   classIds.Add(classId);  }  }  Console.WriteLine("So object: " + count);   // Áp dụng Non-Maximum Suppression (NMS)  var indices = DnnInvoke.NMSBoxes(boxes.ToArray(), scores.ToArray(), confidenceThres, iouThres);   // Chuyển đổi chỉ số thành danh sách các hộp giới hạn   // Khởi tạo danh sách để lưu các hộp, điểm số, và ID lớp cuối cùng  var finalBoxes = new List<Rectangle>();  var finalScores = new List<float>();  var finalClassIds = new List<int>();   // Duyệt qua các chỉ số sau NMS và thu thập thông tin  foreach (var index in indices)  {  if (index >= 0 && index < boxes.Count) // Đảm bảo chỉ số hợp lệ  {  finalBoxes.Add(boxes[index]);  finalScores.Add(scores[index]);  finalClassIds.Add(classIds[index]);  }  }   // Vẽ các hộp giới hạn lên ảnh  DrawBoundingBoxes(imagePath, finalBoxes, finalScores, finalClassIds, outputImagePath);  }   private void DrawBoundingBoxes(string imagePath, List<Rectangle> boxes, List<float> scores, List<int> classIds, string outputImagePath)  {  // Tạo đối tượng Bitmap từ ảnh đầu vào  using var image = new Bitmap(imagePath);  using var graphics = Graphics.FromImage(image);   // Định nghĩa font chữ và cọ để vẽ text  var font = new Font("Arial", 18);   // Tạo một đối tượng Random để tạo màu ngẫu nhiên  var random = new Random();   // Duyệt qua các hộp giới hạn và điểm số tương ứng  for (int i = 0; i < boxes.Count; i++)  {  var box = boxes[i];  var score = scores[i];  var classId = classIds[i];   // Tạo màu ngẫu nhiên cho mỗi hộp giới hạn  Color randomColor = Color.FromArgb(random.Next(256), random.Next(256), random.Next(256));  var pen = new Pen(randomColor, 6);  var brush = new SolidBrush(randomColor);   // Vẽ hộp giới hạn lên ảnh  graphics.DrawRectangle(pen, box);   // Lấy tên class dựa trên classId  string className = classNames.ContainsKey(classId) ? classNames[classId] : "Unknown";   if (objectKeyValue.ContainsKey(className))  {  objectKeyValue[className] += 1;  }  else  {  objectKeyValue.Add(className, 1);  }   // Vẽ điểm số tin cậy và tên class lên ảnh  string text = $"{className}: {score:0.00}";   // Tính toán kích thước của phần text  SizeF textSize = graphics.MeasureString(text, font);   // Tạo nền cho text bằng cách vẽ một hình chữ nhật nhỏ  var backgroundBrush = new SolidBrush(Color.FromArgb(150, 0, 0, 0)); // Màu đen với độ trong suốt  graphics.FillRectangle(backgroundBrush, box.X, box.Y - 20, textSize.Width, textSize.Height);   // Vẽ chữ lên ảnh, với brush đã tạo trước  graphics.DrawString(text, font, Brushes.White, box.X, box.Y - 20); // Dùng màu trắng để chữ nổi bật  }   // Lưu ảnh đã được vẽ hộp giới hạn  image.Save(outputImagePath, ImageFormat.Jpeg);  }  private Dictionary<int, string> LoadClassNamesFromYaml(string yamlFilePath)  {  var classNames = new Dictionary<int, string>();   // Đọc nội dung file YAML  var yamlStream = new YamlStream();  using (var reader = new StreamReader(yamlFilePath))  {  yamlStream.Load(reader);  }   // Lấy root node  var root = (YamlMappingNode)yamlStream.Documents[0].RootNode;   // Lấy phần 'names' từ file YAML  var namesNode = (YamlSequenceNode)root.Children[new YamlScalarNode("names")];   // Lặp qua các phần tử trong 'names' và thêm vào dictionary  for (int i = 0; i < namesNode.Children.Count; i++)  {  classNames.Add(i, namesNode.Children[i].ToString());  }  return classNames;  }   } } |
| --- |

## 5.6. VehicleDetection.src.CSharp.Views > Form1.cs



| using Emgu.CV; using Microsoft.ML.OnnxRuntime; using System; using System.Collections.Generic; using System.Data; using System.Drawing; using System.IO; using System.Text.Json; using System.Threading.Tasks; using System.Windows.Forms; using System.Diagnostics; using VehicleDetection.src.CSharp.Models; using VehicleDetection.src.CSharp.Services;  namespace VehicleDetection\_8.\_0\_ {  public partial class Form1 : Form  {  #region Thuộc tính và Khởi tạo  private readonly ImageExtractor \_imageExtractor;  private readonly string \_rootDir;  private readonly string \_extractImageFolder;  private string \_videoPath;  private int \_frameSkipQuantity = 1;   public Form1()  {  InitializeComponent();  \_rootDir = Path.GetFullPath(Path.Combine("..", "..", ".."));  \_extractImageFolder = Path.Combine(\_rootDir, "resources", "Image", "ExtractFromVideo");  \_imageExtractor = new ImageExtractor(\_extractImageFolder);  }  #endregion   #region Chọn File Video  private void btnSelectFile\_Click(object sender, EventArgs e)  {  using (var openFileDialog = new OpenFileDialog  {  Title = "Chọn File Video",  Filter = "Video Files|\*.mp4;\*.avi;\*.mov;\*.mkv|All Files|\*.\*"  })  {  if (openFileDialog.ShowDialog() == DialogResult.OK)  {  \_videoPath = openFileDialog.FileName;  wmpVideo.URL = \_videoPath;  wmpVideo.Ctlcontrols.play();  }  }  }  #endregion   #region Trích xuất Hình ảnh  private void btnExtractImages\_Click(object sender, EventArgs e)  {  if (\_videoPath == null)  {  MessageBox.Show("Vui lòng chọn Video trước khi trích xuất!");  return;  }   try  {  Helper.DeleteImagesInDirectory(\_extractImageFolder);  \_imageExtractor.ExtractImages(\_videoPath, \_frameSkipQuantity);  MessageBox.Show("Hình ảnh đã được trích xuất thành công!", "Thông báo", MessageBoxButtons.OK, MessageBoxIcon.Information);  }  catch (Exception ex)  {  MessageBox.Show($"Có lỗi xảy ra: {ex.Message}", "Lỗi", MessageBoxButtons.OK, MessageBoxIcon.Error);  }  }   private void nmrframeSkip\_ValueChanged(object sender, EventArgs e)  {  \_frameSkipQuantity = (int)nmrframeSkip.Value;  }  #endregion   #region Phát hiện Bằng Python  private async void runPython\_Click(object sender, EventArgs e)  {  string scriptPath = Path.Combine(\_rootDir, "src", "Python", "detect.py");  string modelPath = Path.Combine(\_rootDir, "model", "yolov8n.pt");  string outputPath = Path.Combine(\_rootDir, "resources", "Image", "OutputDetection", "vehicle\_image\_detected.jpg");   var pythonExecutor = new PythonExecutor("python", scriptPath);  string[] imageFiles = Directory.GetFiles(\_extractImageFolder, "\*.jpg");  int count = 0;  lbProcessed.Text = $"Đang xử lý: {count}/{imageFiles.Length}";   await Task.Run(() =>  {  foreach (var imageFile in imageFiles)  {  string args = $"{imageFile} {modelPath} {outputPath}";  var stopwatch = Stopwatch.StartNew();  string result = pythonExecutor.Execute(args);  stopwatch.Stop();  double elapsedTimeMilliseconds = stopwatch.Elapsed.TotalSeconds;   if (string.IsNullOrWhiteSpace(result))  {  MessageBox.Show("Không thể phân tích kết quả từ Python.", "Lỗi", MessageBoxButtons.OK, MessageBoxIcon.Error);  return;  }   try  {  var detectionResult = JsonSerializer.Deserialize<DetectionResult>(result);  if (detectionResult != null)  {  Invoke(new Action(() =>  {  DisplayDetectionResult(elapsedTimeMilliseconds.ToString(), detectionResult.TotalVehicles, detectionResult.VehicleCounts);  DisplayProcessedImage(outputPath);  }));  }  }  catch (JsonException)  {  MessageBox.Show("Dữ liệu JSON không hợp lệ từ script Python.", "Lỗi", MessageBoxButtons.OK, MessageBoxIcon.Error);  return;  }   count++;  Invoke(new Action(() =>  {  lbProcessed.Text = $"Đã xử lý: {count}/{imageFiles.Length}";  }));  }  });   MessageBox.Show("Hoàn thành xử lý tất cả các ảnh!", "Thông báo", MessageBoxButtons.OK, MessageBoxIcon.Information);  }  #endregion   #region Phát hiện Bằng ONNX  private async void btnONNXDetection\_Click(object sender, EventArgs e)  {  string modelPath = Path.Combine(\_rootDir, "model", "yolov8n.onnx");  string imageFolder = Path.Combine(\_rootDir, "resources", "Image", "ExtractFromVideo");  string outputFolder = Path.Combine(\_rootDir, "resources", "Image", "OutputDetection");  string yamlFilePath = Path.Combine(\_rootDir, "model", "coco.yaml");   var predictor = new YoloV8(modelPath, yamlFilePath);  string[] imageFiles = Directory.GetFiles(imageFolder, "\*.jpg");   await ProcessImagesAsync(imageFiles, predictor, outputFolder);  }   private async Task ProcessImagesAsync(string[] imageFiles, YoloV8 predictor, string outputFolder)  {  await Task.Run(() =>  {  foreach (var imageFile in imageFiles)  {  string outputImagePath = Path.Combine(outputFolder, Path.GetFileName(imageFile));  var stopwatch = Stopwatch.StartNew();  var results = predictor.Detect(imageFile, outputImagePath);  stopwatch.Stop();  double elapsedTimeMilliseconds = stopwatch.Elapsed.TotalSeconds;   UpdateDataGridView(results);  DisplayProcessedImage(outputImagePath);  Invoke(new Action(() =>  {  DisplayDetectionResult(elapsedTimeMilliseconds.ToString(), results.Values.Sum(), results);  }));  }  });  }  #endregion   #region Cập nhật Giao diện Người dùng (UI)  private void DisplayDetectionResult(string TotalTime, int TotalVehicles, Dictionary<string, int> results)  {  if (results == null) return;   lbTotalTime.Text = $"Tổng thời gian thực hiện: {TotalTime} giây";  lbTotalVehicles.Text = $"Tổng số phương tiện: {TotalVehicles}";   dataGridView1.Columns.Clear();  dataGridView1.Columns.Add("VehicleType", "Loại Phương Tiện");  dataGridView1.Columns.Add("Count", "Số Lượng");   dataGridView1.Rows.Clear();  foreach (var result in results)  {  dataGridView1.Rows.Add(result.Key, result.Value);  }  }   private void UpdateDataGridView(Dictionary<string, int> results)  {  Invoke((Action)(() =>  {  dataGridView1.Columns.Clear();  dataGridView1.Columns.Add("VehicleType", "Loại Phương Tiện");  dataGridView1.Columns.Add("Count", "Số Lượng");   dataGridView1.Rows.Clear();  foreach (var result in results)  {  dataGridView1.Rows.Add(result.Key, result.Value);  }  }));  }   private void DisplayProcessedImage(string outputImagePath)  {  if (File.Exists(outputImagePath))  {  Invoke((Action)(() =>  {  using (var img = Image.FromFile(outputImagePath))  {  pictureBox1.Image = new Bitmap(img);  pictureBox1.SizeMode = PictureBoxSizeMode.Zoom;  }  }));  }  }  #endregion  } } |
| --- |

# 5.7. Detect.py

| import os import sys import json import time import cv2 from collections import Counter from ultralytics import YOLO  def detect\_vehicles(image\_path, model\_path, output\_path):  # Tải mô hình YOLO  model = YOLO(model\_path)    # Đọc ảnh  image = cv2.imread(image\_path)  if image is None:  print("Không thể mở ảnh")  return None   # Thiết lập kích thước mới cho ảnh  new\_width = 640  new\_height = 480  resized\_image = cv2.resize(image, (new\_width, new\_height))    # Bắt đầu đo thời gian  start\_time = time.time()    # Dự đoán bằng YOLO  results = model(resized\_image, verbose=False)  result = results[0]    # Đếm từng loại phương tiện  vehicle\_counts = Counter()  for box in result.boxes:  class\_id = int(box.cls)  class\_name = result.names[class\_id]  vehicle\_counts[class\_name] += 1   # Vẽ bounding box và số lượng phương tiện  annotated\_image = result.plot()  cv2.putText(annotated\_image, f'Total Vehicles: {sum(vehicle\_counts.values())}', (10, 40),  cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 0, 255), 2)   # Lưu ảnh đầu ra  cv2.imwrite(output\_path, annotated\_image)    # Tính tổng thời gian xử lý  end\_time = time.time()  total\_time = end\_time - start\_time   # Chuẩn bị dữ liệu JSON  data = {  "total\_time": total\_time,  "total\_vehicles": sum(vehicle\_counts.values()),  "vehicle\_counts": dict(vehicle\_counts)  }   # Trả về JSON dưới dạng chuỗi để in ra cho C#  return json.dumps(data, ensure\_ascii=False)  if \_\_name\_\_ == "\_\_main\_\_":  # Nhận các đối số từ dòng lệnh  if len(sys.argv) < 2:  print("Vui lòng cung cấp đường dẫn tới ảnh.")  sys.exit(1)   image\_path = sys.argv[1]  model\_path = sys.argv[2]  output\_path = sys.argv[3]   # Gọi hàm detect\_vehicles và trả về kết quả JSON  result\_json = detect\_vehicles(image\_path, model\_path, output\_path)  print(result\_json) # In JSON để C# có thể nhận được output |
| --- |

# 6. Hướng dẫn sử dụng

**Khởi động ứng dụng**

1. **Mở ứng dụng WinForms**: Chạy Form1 để mở giao diện chính của ứng dụng.
2. **Chọn file video**:
   * Nhấn nút "Chọn File Video".
   * Chọn file video từ máy tính (hỗ trợ các định dạng như .mp4, .avi, .mov, .mkv).
3. **Trích xuất frame từ video**:
   * Thiết lập giá trị frameSkip để xác định khoảng cách giữa các frame được trích xuất.
   * Nhấn nút **Trích xuất Hình ảnh** để lưu frame vào thư mục ExtractFromVideo.
4. **Nhận dạng phương tiện**:
   * **Sử dụng PyTorch**:
     + Nhấn nút Phát hiện Bằng Python để thực hiện nhận dạng phương tiện qua PyTorch.
   * **Sử dụng ONNX**:
     + Nhấn nút Phát hiện Bằng ONNX để sử dụng mô hình ONNX.
5. **Xem kết quả**: Kết quả nhận dạng phương tiện sẽ hiển thị trên giao diện với các thông tin sau:
   * Tổng số phương tiện
   * Loại phương tiện và số lượng tương ứng
   * Hình ảnh đã nhận dạng.

