

# Car Detection Project Documentation

## Team members :

202200997	عمرو خالد عرب
202203223	هيثم الصيد حسين ابوزيد
202202732	محمد رجب سعد رجب
202202787	مكاريس ممدوح شفيق فاخوري
202201568	يوسف صاوي محمد علي
202203703	يوسف أحمد يوسف محمد

## Introduction

This project presents a simple yet effective application for real-time car detection using the YOLOv8 (You Only Look Once version 8) object detection model. The system provides a graphical user interface (GUI) that allows users to upload video files and analyze them frame by frame to detect cars. YOLOv8 is a state-of-the-art, deep learning-based object detection model developed by Ultralytics. It offers high-speed and accurate performance, making it ideal for real-time applications such as traffic monitoring, autonomous driving, and surveillance. The goal of this project is to demonstrate how YOLOv8 can be integrated into a user-friendly desktop application using Python, OpenCV, and Tkinter. This allows users—especially those without technical backgrounds—to interact with powerful AI-based car detection technology through a simple graphical interface.

# Project Overview

This project is a simple desktop application for detecting cars in video files using the YOLOv8 model. It combines a graphical interface (Tkinter) with real-time video processing (OpenCV) and deep learning-based object detection. Users can upload a video, and the system will display detected cars frame by frame with bounding boxes.

## System Requirements

To run this application, you will need:

- A computer:

Works on Windows, macOS, or Linux.

Should have at least 4GB of RAM.

- Python:

Make sure you have Python 3.8 or higher installed on your computer.

- Internet connection:


To download the required tools and libraries (if not already installed).

- No special hardware needed:

If you have a graphics card (GPU), it will speed up the car detection process, but it's not necessary.

# Code-level Documentation


## 1. Importing Libraries



```
1 import tkinter as tk
2 from tkinter import filedialog
3 from PIL import Image, ImageTk
4 from ultralytics import YOLO
5 import cv2
```

- tkinter: A library used for creating the graphical user interface (GUI) that allows the user to select a video file.
- filedialog: A part of tkinter that lets the user browse and select a video file from their system.
- Pillow: An image-processing library (though not used in the code directly here).
- YOLO: The library used to load the YOLOv8 model for object detection.
- cv2: OpenCV, used for video processing and adding annotations like bounding boxes on detected objects.

## 2. Load YOLOv8 Model



```
1 model = YOLO('yolov8n.pt')
2 CAR_CLASS_ID = 0
```

- model: This loads the YOLOv8 pre-trained model (yolov8n.pt), which is used to detect objects in the video frames.
- CAR\_CLASS\_ID: A constant that represents the class ID for "car" in the COCO dataset, where 2 corresponds to cars.

### 3. Detecting Cars in a Frame

```
1 def detect_cars(frame):
2     results = model(frame)[0]
3     for box in results.bboxes:
4         cls = int(box.cls[0])
5         if cls == CAR_CLASS_ID:
6             x1, y1, x2, y2 = map(int, box.xyxy[0])
7             conf = float(box.conf[0])
8             label = f"Car {conf:.2f}"
9             cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
10            cv2.putText(frame, label, (x1, y1 - 10),
11                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
12    return frame
```

- **detect\_cars(frame):**

This function is responsible for detecting cars in each frame of the video. It does this by passing the frame to the YOLO model for object detection. The results contain information about all detected objects in the frame.

- **results = model(frame)[0]:**

This line runs the YOLOv8 model on the given frame. The model detects all objects (like cars, people, etc.) in the frame and returns the results, which are stored in results. The [0] at the end refers to getting the first detection result (as the model might return multiple results for different frames).

- **for box in results.bboxes:**

This loop iterates through all the detected objects in the frame. Each box represents a single detected object.

- **cls = int(box.cls[0]):**

box.cls[0] gives the class (type) of the detected object. For example, a "car" might have the class ID 2 (as per the COCO dataset). We convert this value to an integer to easily compare it.

- **if cls == CAR\_CLASS\_ID:**

This condition checks whether the detected object is a car. We defined CAR\_CLASS\_ID = 2 earlier, which corresponds to the class ID for cars in the COCO dataset.

- **x1, y1, x2, y2 = map(int, box.xyxy[0]):**

box.xyxy[0] provides the coordinates of the bounding box around the detected object in the format [x1, y1, x2, y2], where:

(x1, y1) is the top-left corner of the bounding box.

(x2, y2) is the bottom-right corner of the bounding box.

These values are converted to integers using map(int, ...) because the model outputs floating-point values.

- **conf = float(box.conf[0]):**

box.conf[0] gives the confidence score of the detection. It's a measure of how confident the model is that the detected object is actually a car. This value is converted to a float for easier handling.

- **label = f"Car {conf:.2f}":**

This line creates a label that includes the word "Car" and the confidence score rounded to two decimal places. For example, "Car 0.98" would indicate that the model is 98% confident the detected object is a car.

- `cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2):`

This line draws a green rectangle around the detected car using the coordinates of the bounding box (x1, y1) for the top-left corner and (x2, y2) for the bottom-right corner. The color (0, 255, 0) is green, and the 2 represents the thickness of the rectangle's border.

- `cv2.putText(frame, label, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2):`

This line adds the label text (e.g., "Car 0.98") to the frame above the bounding box. The label is placed 10 pixels above the top-left corner of the bounding box ((x1, y1 - 10)). The font used is `cv2.FONT_HERSHEY_SIMPLEX`, with a font size of 0.5, and the text color is green.

- `return frame:`

After detecting and labeling the cars, the function returns the modified frame with bounding boxes and labels.

## 4. Processing the Selected Video

```
1  # Process selected video
2  def process_video(video_path):
3      cap = cv2.VideoCapture(video_path)
4      if not cap.isOpened():
5          print("Error opening video file.")
6          return
7
8      while True:
9          ret, frame = cap.read()
10         if not ret:
11             break
12         frame = detect_cars(frame)
13         cv2.imshow("YOLOv8 Car Detection", frame)
14         if cv2.waitKey(1) & 0xFF == 27: # Press ESC to stop
15             break
16
17     cap.release()
18     cv2.destroyAllWindows()
```

- **process\_video(video\_path):**  
This function handles the processing of the video file. It opens the video, reads each frame, and applies the detect\_cars() function to each frame for car detection.
- **cap = cv2.VideoCapture(video\_path):**  
This line initializes a video capture object to read the video file from the provided video\_path. It allows the program to access the video and its frames.
- **if not cap.isOpened():**  
This checks whether the video file was successfully opened. If not, it prints an error message and stops the function.
- **ret, frame = cap.read():**  
This reads the next frame from the video. ret is a boolean that indicates whether the frame was successfully read, and frame contains the actual image data for the current frame.
- **if not ret:**  
If no frame was read (ret is False), it breaks the loop, meaning the video has ended.
- **frame = detect\_cars(frame):**  
The detect\_cars() function is called to detect and annotate cars in the current frame.
- **cv2.imshow("YOLOv8 Car Detection", frame):**  
This displays the processed frame in a window titled "YOLOv8 Car Detection". The frame includes any bounding boxes and labels drawn by the detect\_cars() function.
- **if cv2.waitKey(1) & 0xFF == 27:**  
This waits for a key press. If the ESC key (code 27) is pressed, it exits the loop and stops processing the video.
- **cap.release():**  
This releases the video capture object and frees any resources used by it.
- **cv2.destroyAllWindows():**  
This closes all OpenCV windows.

## 5. Selecting a Video File

```

1  # GUI: Browse and select a video file
2  def select_video():
3      file_path = filedialog.askopenfilename(filetypes=[("Video files", "*.mp4 *.avi *.mov *.mkv")])
4      if file_path:
5          process_video(file_path)

```

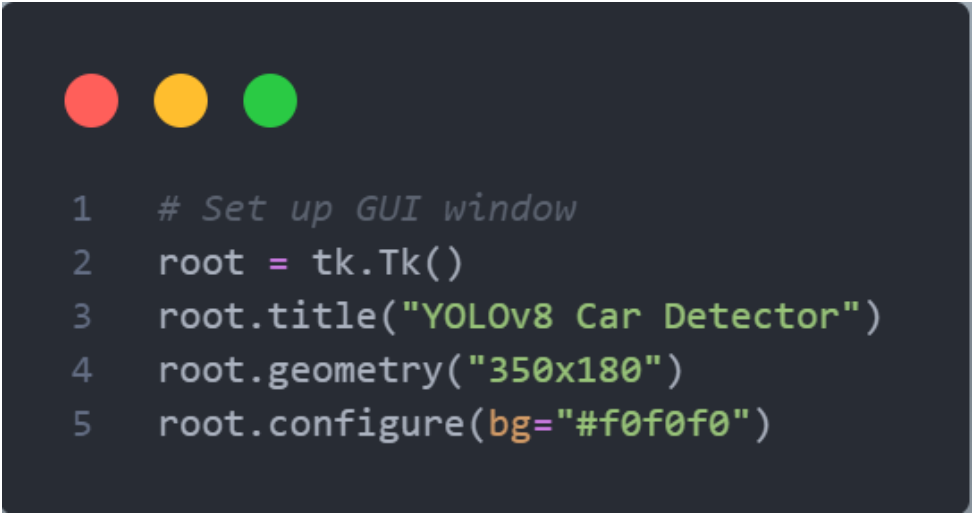
- **select\_video():**  
This function is triggered when the user clicks the "Browse Video" button in the GUI.
- **filedialog.askopenfilename(...):**

Opens a file selection dialog allowing the user to choose a video file. It filters the files to common video formats like .mp4, .avi, .mov, and .mkv.

- **if file\_path:**

If the user selects a file (i.e., the path is not empty), the selected file is passed to the process\_video() function to begin the detection process.

## 6. Setting Up the GUI Window



```
1  # Set up GUI window
2  root = tk.Tk()
3  root.title("YOLOv8 Car Detector")
4  root.geometry("350x180")
5  root.configure(bg="#f0f0f0")
```

- **tk.Tk():**

Initializes the main application window using Tkinter.

- **root.title(...):**

Sets the window's title that appears in the top bar of the window.

- **root.geometry(...):**

Specifies the size of the window as 350 pixels wide and 180 pixels tall.

- **root.configure(...):**

Sets the background color of the window to a light gray (#f0f0f0).

## 7. Building and Running the GUI

```

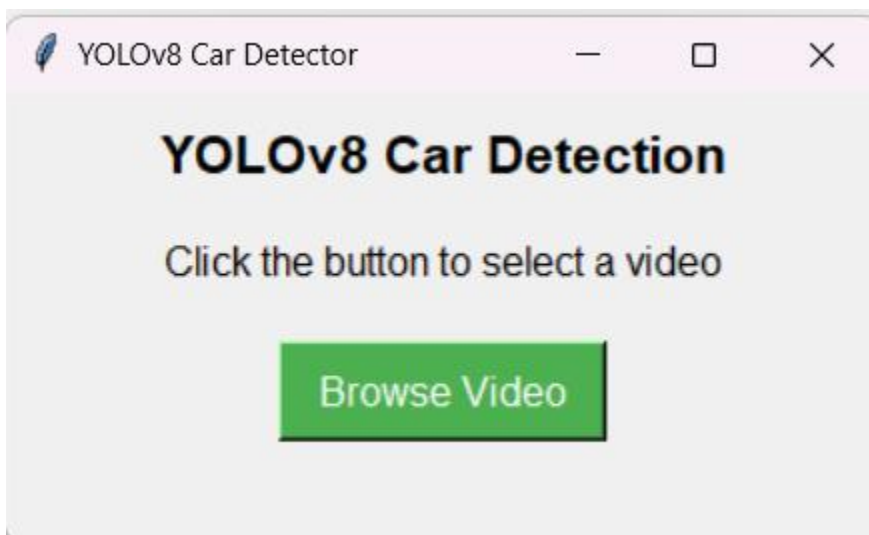
1  # GUI Elements
2  title = tk.Label(root, text="YOLOv8 Car Detection", font=("Helvetica", 16, "bold"), bg="#f0f0f0")
3  title.pack(pady=10)
4
5  instruction = tk.Label(root, text="Click the button to select a video", font=("Helvetica", 12), bg="#f0f0f0")
6  instruction.pack(pady=5)
7
8  browse_button = tk.Button(root, text="Browse Video", command=select_video, font=("Helvetica", 12),
9                           bg="#4CAF50", fg="white", padx=10, pady=5)
10 browse_button.pack(pady=15)
11
12 # Start GUI
13 root.mainloop()

```

- **tk.Label(...):**  
Creates text labels in the GUI. One acts as the title and the other as an instruction for the user.
- **tk.Button(...):**  
Adds a green “Browse Video” button. When clicked, it opens a file dialog to select a video for car detection.
- **.pack(...):**  
Arranges the GUI elements vertically with spacing (pady) for visual clarity.
- **root.mainloop():**  
Starts the GUI's main loop. It keeps the window open and interactive until the user closes it.

## Visual Documentation

### 1. Graphical User Interface (GUI)





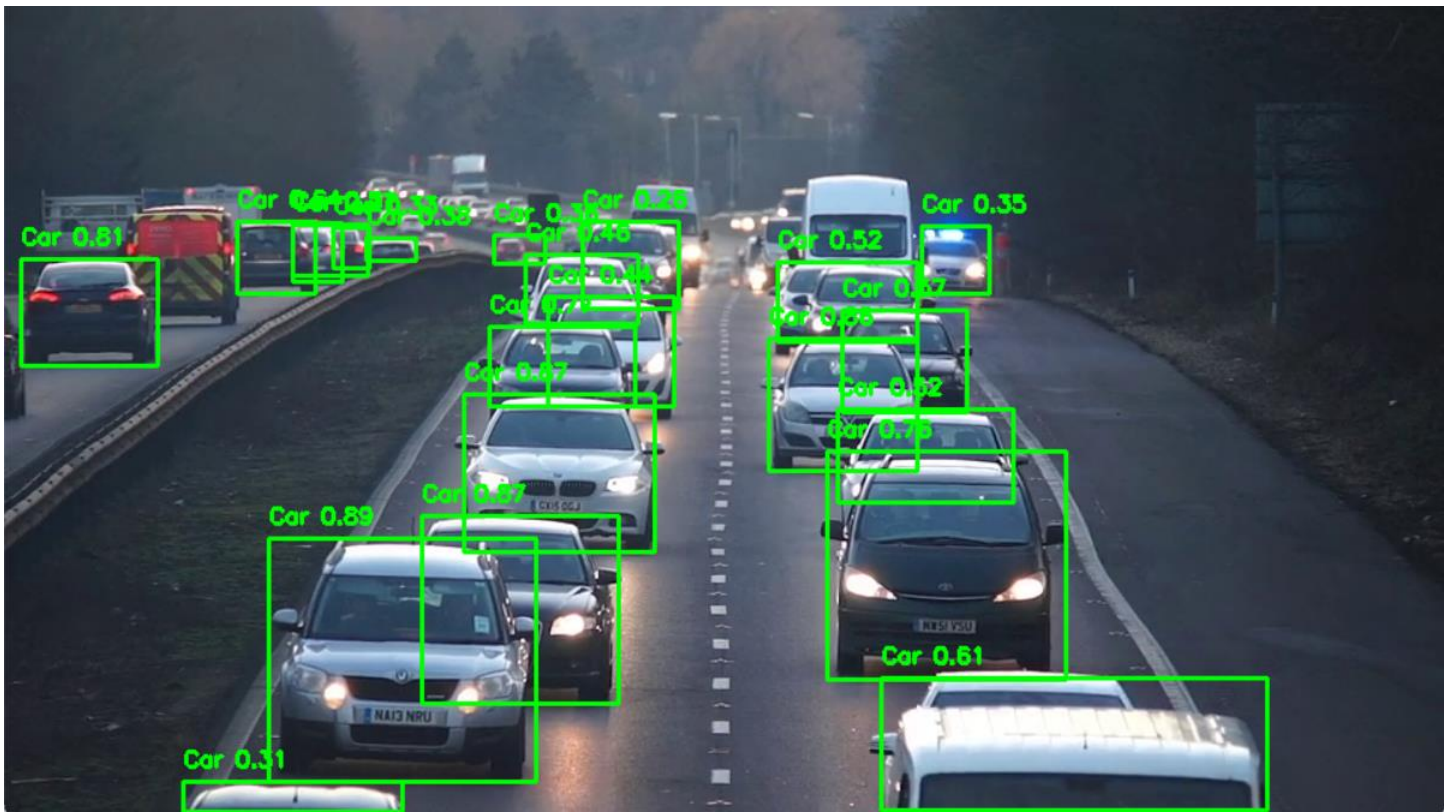
The image above shows the user interface of the YOLOv8 Car Detector application. It provides a simple and user-friendly layout where users can click the "Browse Video" button to select a video file for car detection.

## 2. Detection Results

This section demonstrates the effectiveness of the YOLOv8 model in detecting vehicles in a video frame. The comparison shows how the original input is processed and enhanced with detection results.



*This is a raw frame from a video showing multiple vehicles on a road. No detection or labeling has been applied yet.*

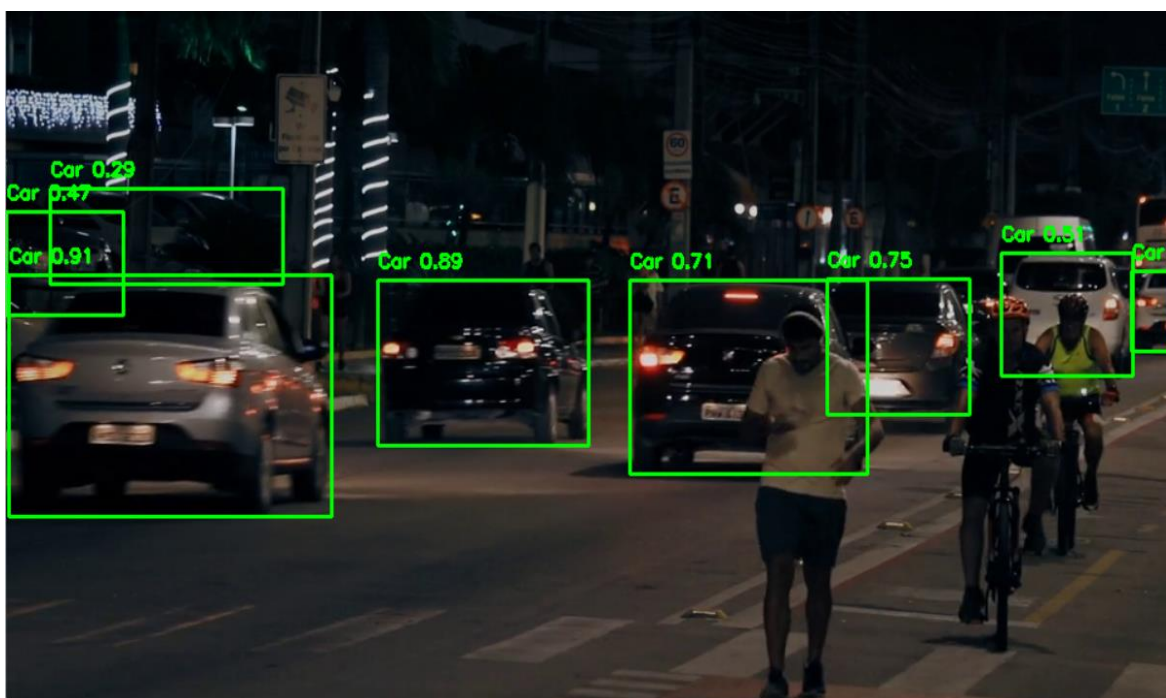


*This frame shows the output after applying the YOLOv8 model. Each detected vehicle is highlighted with a green bounding box and labeled with a confidence score, demonstrating the model's ability to recognize and locate cars in real time.*

Another example:



*Before detecting*



*After detecting*

# Applications and Benefits of Car Detection

Car detection using computer vision models like YOLOv8 has numerous real-world applications across various domains. Some key benefits and practical uses include:

## 1. Traffic Monitoring and Management:

Helps in real-time monitoring of vehicle flow, detecting congestion, and optimizing traffic signals to improve road efficiency.

## 2. Surveillance and Security:

Enhances security systems by detecting suspicious or unauthorized vehicles in restricted areas or during specific time intervals.

## 3. Smart Parking Systems:

Assists in identifying available parking spots and managing parking lots efficiently through automatic vehicle detection.

## 4. Autonomous Driving and ADAS (Advanced Driver Assistance Systems):

Essential for detecting surrounding vehicles, helping self-driving cars make safe navigation decisions.

## 5. Law Enforcement and Toll Collection:

Supports automatic number plate recognition (ANPR), speed enforcement, and automated toll collection without human intervention.

## 6. Data Collection and Analytics:

Enables gathering of vehicle counts, types, and flow patterns for urban planning and infrastructure development.

## References

This project did not rely on external research papers or documents. However, the following libraries and tools were used:

- Ultralytics YOLOv8 – for object detection
- OpenCV (cv2) – for video processing and drawing annotations
- Tkinter – for building the graphical user interface
- Pillow (PIL) – for image handling in the GUI

All tools are publicly available and open source.