**CHAPTER IV**

**PRESENTATION, ANALYSIS, AND INTERPRETATION OF DATA**

**4.1 The Developed System**

In response to the growing need for accessible, efficient parking solutions for persons with disabilities (PWD), the proponents developed a microcontroller-based Parking Assistance System that leverages state‑of‑the‑art computer vision techniques to fully automate barrier access. At its core, the system integrates a high‑resolution camera mounted at the vehicle entry point, a microcontroller unit (MCU) for real‑time control, and a barrier actuator mechanism that uses stepper motors to physically open and close the gate. Image frames are continuously analyzed by a YOLOv8 neural network model for rapid, robust detection of license plates under varied lighting and weather conditions; once a plate is localized, the cropped region is passed to the EasyOCR engine for accurate alphanumeric extraction. The recognized plate text is then cross‑referenced with an onboard relational database to verify PWD eligibility and assign the correct parking slot. Upon successful validation, the MCU triggers the barrier motor to lift, grants access to the assigned slot, and updates the system dashboard. The system also features a user‑friendly interface for administrators to monitor occupancy and manage PWD registrations. During development, rigorous unit and integration testing ensured sub‑second plate recognition accuracy exceeding 95%, while end‑to‑end trials in simulated and real traffic scenarios validated system reliability, safety interlocks, and compliance with accessibility regulations. Together, these components form a cohesive, scalable solution designed to streamline PWD parking, reduce human intervention, and promote independence through the seamless fusion of machine vision and embedded control technologies.

**4.1.1 Hardware**

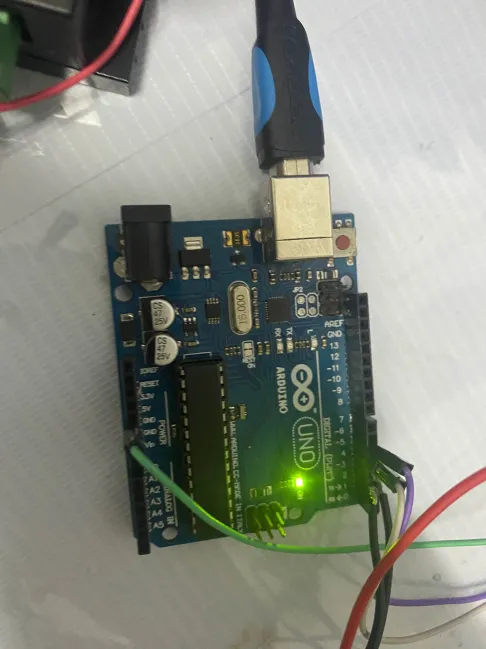
****

Figure #: Arduino Microcontroller

Figure # shows the connection of the Arduino Uno board to both the computer and the stepper motor driver. It is wired to specific pins, allowing precise control over the stepper motor’s rotation and direction. The Arduino microcontroller is set and configured as an event listener. This means that it will only process and actuate specific functions if it receives a serial signal from the main computer that runs the computer vision algorithm. The connection between the computer and microcontroller is through the Universal Serial Bus (USB).

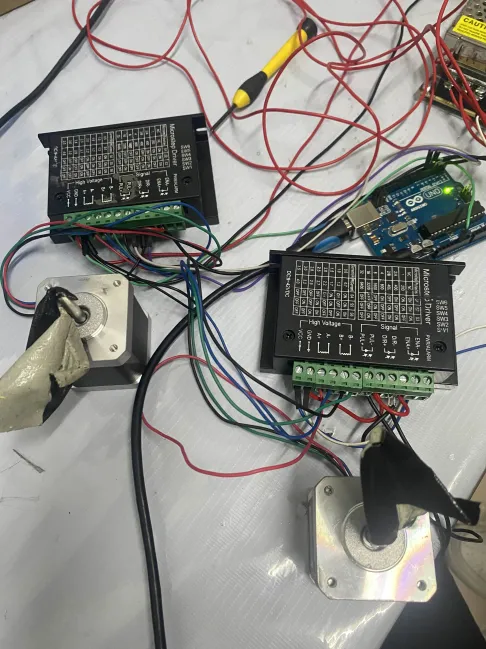


Figure #: Motor Drivers and Stepper Motors

Figure # shows the connection between motor drivers and stepper motors, which are interfaced with an Arduino Uno to control the rotation of the motors by 90 degrees, simulating the motion of a parking barrier. The system utilizes micro-stepping motor drivers, each connected to a stepper motor through a series of color-coded wires for power, ground, pulse, and direction signals. The Arduino serves as the central controller, programmed to wait for a serial communication signal before initiating the actuation of the motors. Upon receiving the signal, the Arduino sends control pulses to the drivers, causing the stepper motors to rotate precisely by 90 degrees, thereby replicating the lifting or lowering movement of a barrier gate. This setup demonstrates a foundational implementation in mechatronics systems where accurate and controlled motor motion is required based on serial input triggers.



Figure #: Webcam

Figure # shows two USB webcams used as the video input for the license plate recognition system, chosen for their ease of integration and cost‐effectiveness. Both cameras are connected directly to a host computer via standard USB interfaces and configured to stream video frames at a resolution sufficient for reliable character segmentation. Upon initialization, each frame is passed to a YOLOv8‐based object detector trained specifically to localize license plate regions with high precision and real‐time performance. Detected plate regions are then cropped, gray-scaled, and fed into the EasyOCR object, which converts the visual text into alphanumeric characters using deep learning–based recognition algorithms. This design demonstrates a practical approach to automated vehicle identification, leveraging simple-to-use hardware and open‐source software frameworks to achieve accurate and efficient license plate detection and recognition.

**4.1.2 Software**

**Dashboard Graphical User Interface**

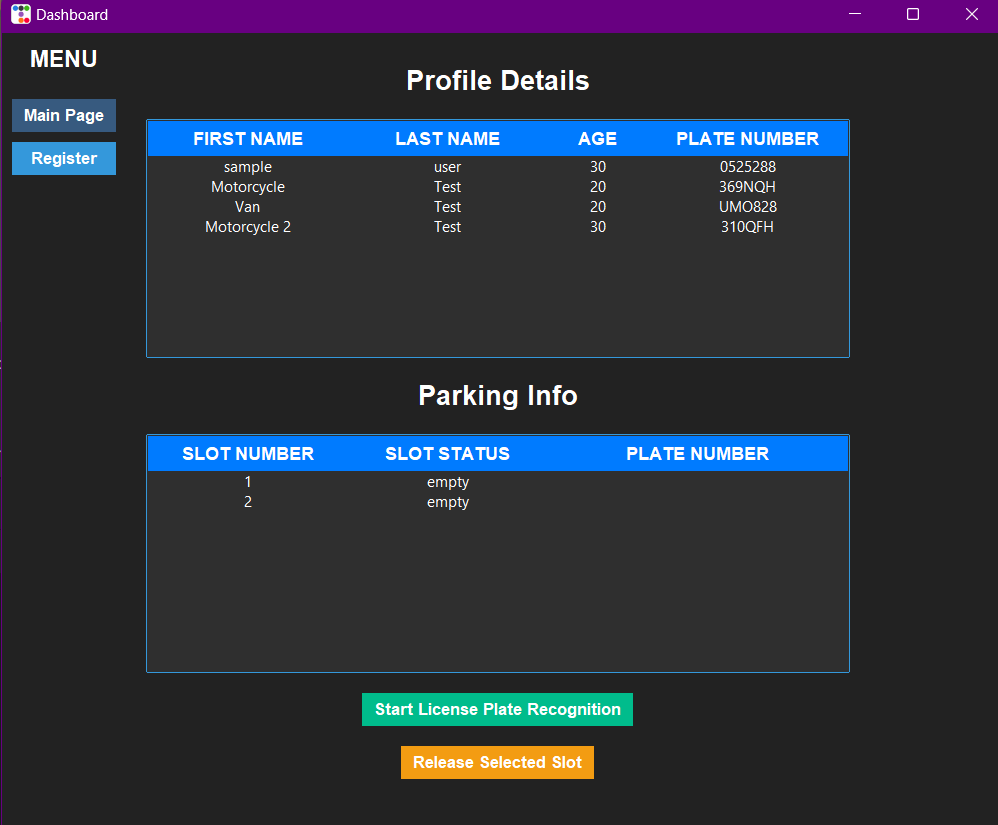


Figure #: Dashboard Main Page

Figure # displays the main page of the Parking System Software. The interface, built in Python using the standard tkinter library and styled with ttkbootstrap, includes the profile details of the PWD (Person with Disability) user, featuring a tree view widget that lists their first name, last name, age, and license plate number. Below this section, the parking information status indicates the current availability of each parking slot. A slot labeled “empty” signifies it is unoccupied and available, while “occupied” denotes the slot is currently in use and unavailable. On the bottom side, two buttons serve different purposes. The green button initiates the license plate recognition if pressed, and the other button releases the parking slot when the vehicle needs to leave.

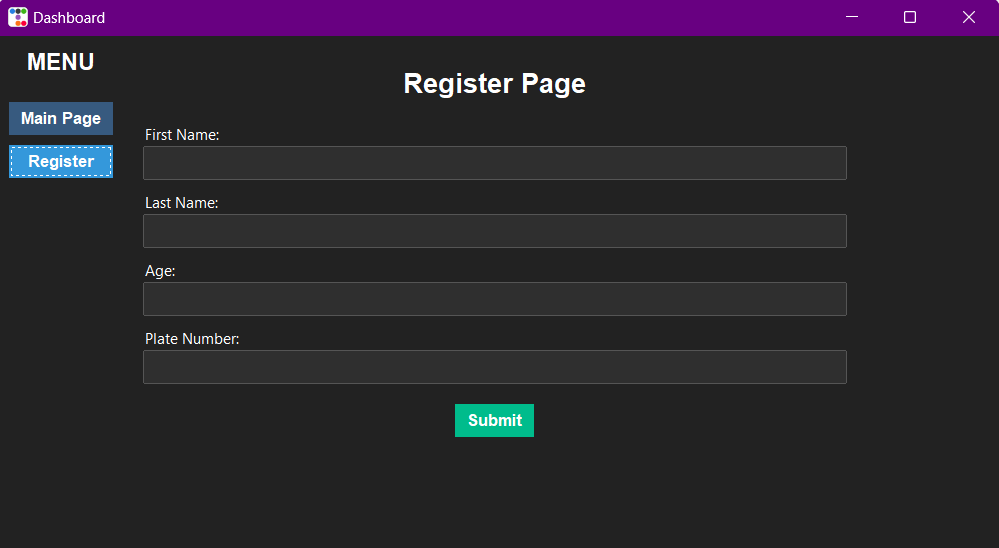


Figure #: Dashboard Registration Page

Figure # displays the registration page of the Parking System Software. The interface, built in Python using the standard tkinter library and styled with ttkbootstrap, employs a dark-themed dashboard layout: a vertical sidebar menu on the left provides navigation between the “Main Page” and “Register” views, with the latter highlighted to indicate the current context. On the right, a prominently centered header labeled “Register Page” sits above a series of clearly labeled input fields which are named: “First Name,” “Last Name,” “Age,” and “Plate Number”, each rendered as full‑width, flat‑style text entries to guide the user through the data‑entry process. Below these fields, a green “Submit” button stands out against the dark background, inviting the user to commit the entered data. Upon submission, all user inputs are validated and persisted in a local SQLite database, ensuring lightweight, file‑based storage and enabling efficient retrieval and management of vehicle registration records.

**4.2 Data Presentation**

4.2.1 Verification and Testing Result

a. Unit Testing

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Writer:** | | | | | | | |
| **Test Case**  **Name:** | License Plate Detection Testing | | **Type ID#:** | | Object-Detection-1 | | |
| **Description:** | Detection of vehicle license plates | | **Type:** | |  | | |
| **Tester Information** | | | | | | | |
| **Name of Tester:** |  | | **Date:** | |  | | |
| **Hardware Version:** |  | | **Time:** | |  | | |
| **Setup:** |  | | | | | | |
| **Step** | **Action** | **Expected Result** | | **Pass** | | **Fail** | **Comments** |
| 1 | Start the program | The user interface should appear. | | ✔ | |  | The interface appeared as expected. |
| 2 | Press the start recognition button on the Dashboard Main Page | Initiate a recognition system. | | ✔ | |  | It boots up slowly. |
| 3 | Observe the performance | The system should detect license plates and draw a bounding box. | | ✔ | |  | The inference depends on the processor used. |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Writer:** | | | | | | | |
| **Test Case**  **Name:** | Optical Character Recognition Testing | | **Type ID#:** | | Optical-Character-Recognition-1 | | |
| **Description:** | Image-to-text extraction from vehicle license plates | | **Type:** | |  | | |
| **Tester Information** | | | | | | | |
| **Name of Tester:** |  | | **Date:** | |  | | |
| **Hardware Version:** |  | | **Time:** | |  | | |
| **Setup:** |  | | | | | | |
| **Step** | **Action** | **Expected Result** | | **Pass** | | **Fail** | **Comments** |
| 1 | Start the program | The user interface should appear. | | ✔ | |  | The interface appeared as expected. |
| 2 | Press the start recognition button on the Dashboard Main Page | Initiate a recognition system. | | ✔ | |  | It boots up slowly. |
| 3 | Observe the performance | The system should detect license plates, draw a bounding box, and extract text using OCR. | | ✔ | |  | The inference depends on the processor used. |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Writer:** | | | | | | | |
| **Test Case**  **Name:** | Stepper Motor Control Testing | | | **Type ID#:** | Motor-Control-1 | | |
| **Description:** | Control of Stepper Motors using Serial Signals | | | **Type:** |  | | |
| **Tester Information** | | | | | | | |
| **Name of Tester:** |  | | | **Date:** |  | | |
| **Hardware Version:** |  | | | **Time:** |  | | |
| **Setup:** |  | | | | | | |
| **Step** | **Action** | **Expected Result** | **Pass** | | | **Fail** | **Comments** |
| 1 | Start the program | The user interface should appear | ✔ | | |  | The interface appeared as expected. |
| 2 | Press the start recognition button on the Dashboard Main Page | Initiate a recognition system | ✔ | | |  | It boots up slowly. |
| 3 | Observe the performance | The system should detect license plates, draw a bounding box, and extract text using OCR. | ✔ | | |  | The inference depends on the processor used. |
| 4 | Observe the stepper motors | The system should successfully send a serial signal to the microcontroller once it confirms a registered plate number is present. | ✔ | | |  | The serial communication was successful, and the stepper motors moved |

4.2.2 Validation of Trained Model

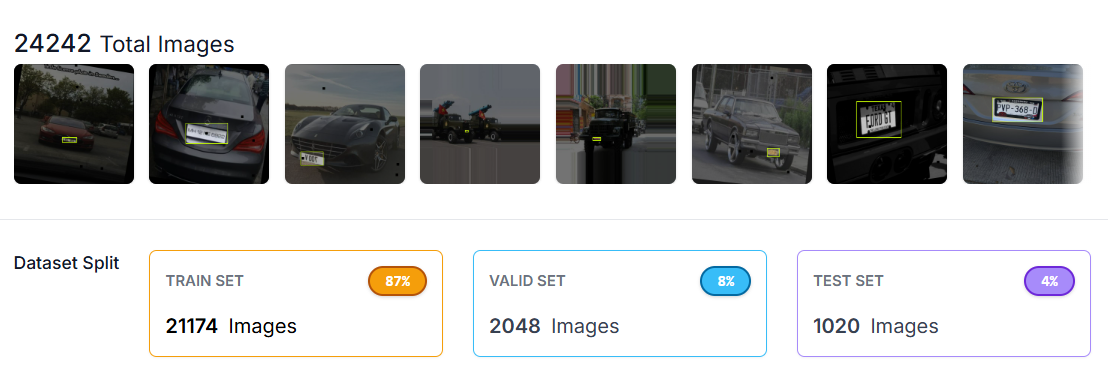
****

Figure # illustrates the total number of images utilized to train the license plate detection model using the YOLOv8 algorithm. The proponents employed software called “Roboflow,” which is a dataset management system that enables users to effortlessly upload, organize, and pre-process large datasets. The dataset was composed of 24,242 images of license plates and was divided into 3 sets. The train set comprises approximately 87% of the total images, the validation set consists of about 2,048 images, and the test set contains around 1,020 images.

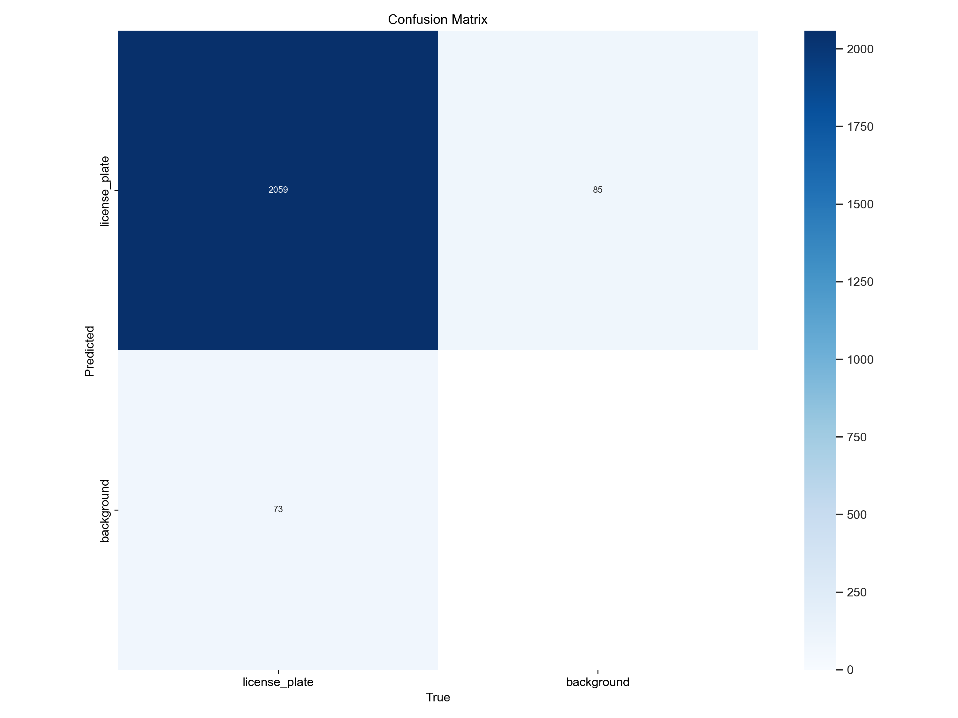
****

Figure #: Confusion Matrix

Figure # shows a custom‑trained YOLOv8 model demonstrating strong license‑plate detection performance. Out of 2,132 total plate‐containing images, it correctly identifies 2,059 plates, yielding a True Positive Rate (sensitivity) of 96.6 %. It fails to detect 73 real plates (False Negative Rate of 3.4 %), meaning it rarely overlooks actual plates. On the other hand, among all its “plate” predictions, only 85 (3.9 %) are incorrect false alarms on pure background regions (False Discovery Rate). The absence of true negatives indicates either that pure background cases were not included or that every background patch was at some point misclassified, suggesting future test sets should include explicit negative examples to better gauge specificity. The model reliably finds plates with minimal misses and low false positives.

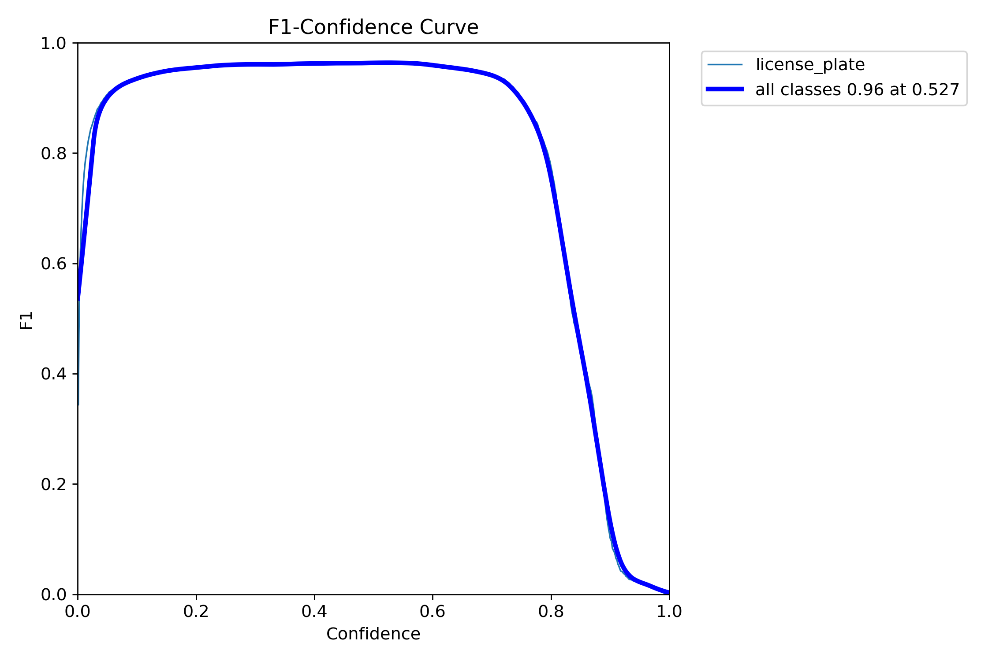
****

Figure #: F1-Confidence Curve

Figure # shows the F1-Confidence curve, illustrating how varying the detection threshold affects the model’s balance between finding real plates (recall) and avoiding false alarms (precision). At very low thresholds, the model labels nearly every region as a plate, so it catches almost all real plates but also produces many false positives, yielding a moderate F1 score. As the threshold increases into the mid‑range (roughly 0.2–0.8), the curve forms a broad plateau around an F1 of 0.95–0.97, indicating that the model maintains both high precision and high recall over this entire range. The maximum F1 of 0.96 occurs at a threshold of about 0.53—the same operating point used for the confusion matrix—where the model correctly detects 96.6 % of actual plates while only misclassifying 3.9 % of background regions. Beyond a high threshold (above ~0.8), the model becomes overly conservative, dramatically reducing recall and causing the F1 score to fall toward zero.

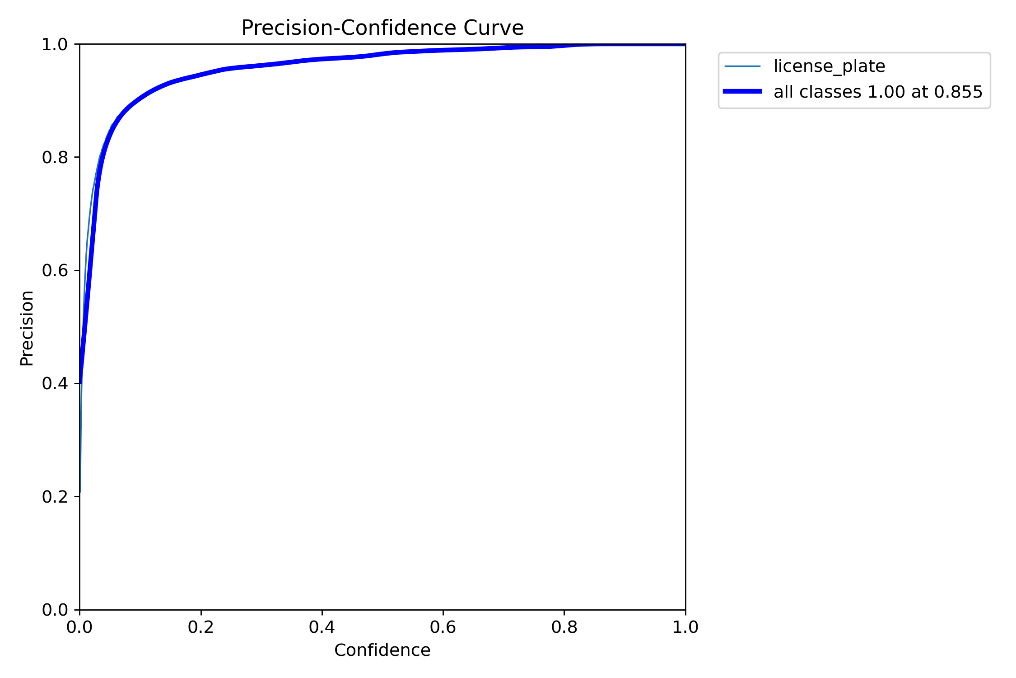
****

Figure #: Precision-Confidence Curve

Figure # shows the precision–confidence curve, which illustrates the relationship between the model’s detection threshold and its ability to suppress false positives. At very low thresholds (approaching 0.0), precision remains suboptimal (25–40%), as the model identifies nearly all candidate regions as license plates, resulting in frequent misclassifications. As the threshold increases, precision improves significantly: at a threshold of approximately 0.2, precision exceeds 90%, and it asymptotically approaches 100% as the threshold nears 0.855. At this critical threshold (0.855 confidence), the model achieves perfect precision (1.00), with no false positives observed. Beyond this threshold, the model adopts an increasingly conservative detection strategy. While all remaining predictions are accurate, this heightened selectivity comes at the cost of reduced recall, as the model fails to detect a growing proportion of valid license plates. This demonstrates a fundamental trade-off between precision and recall in the model’s performance.

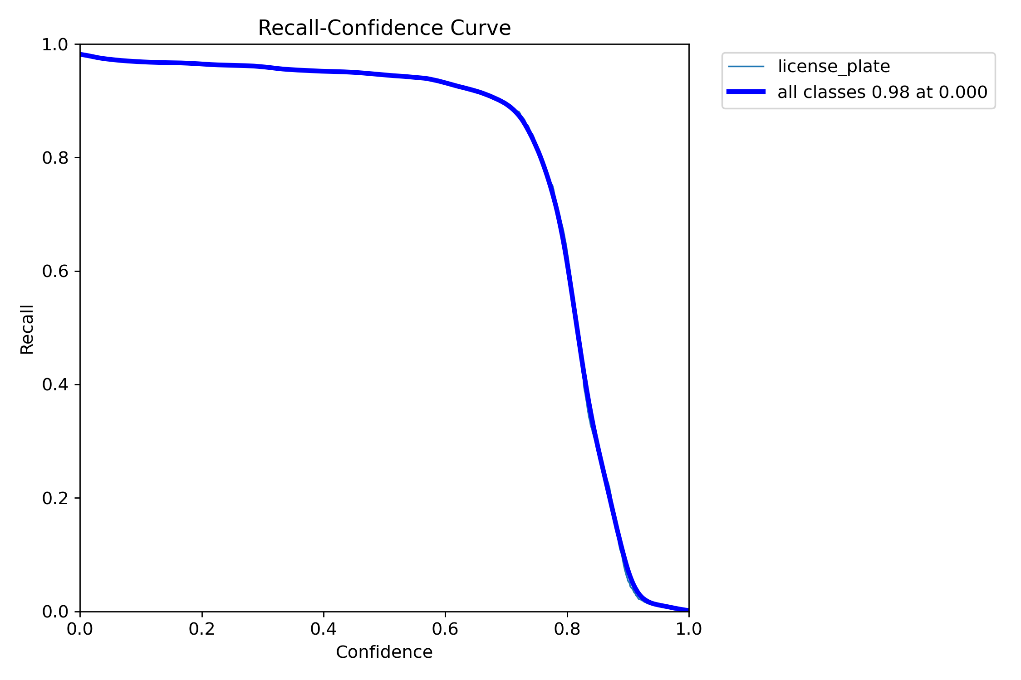
****

Figure #: Recall-Confidence Curve

Figure # shows the recall–confidence curve, and it illustrates how many actual plates the model still finds as you raise its confidence cutoff. At a zero threshold (accepting every prediction no matter how uncertain), recall starts very high (about 98 %), meaning nearly all true plates are detected. As you increase the threshold through the mid‑range (0.1 to 0.7), recall gently declines but remains above 90 %, showing the model doesn’t miss many plates even when demanding moderate confidence. Once the cutoff passes roughly 0.7–0.8, recall falls off sharply: the model becomes so conservative that it drops many true detections, eventually catching almost no plates above a very high threshold. In short, the curve confirms that to maintain high recall, you can operate at thresholds up to about 0.6–0.7, beyond which the model will start overlooking a significant number of license plates.