**AI-Powered Vehicle Access Control System Using License Plate Recognition**

**Final Project Report  
Nikhitha Kilari  
CS 455, Spring 2025**

**Abstract**

This report details the development of an AI-powered vehicle access control system leveraging computer vision and deep learning techniques specifically, custom-trained YOLOv8 for license plate detection and Tesseract OCR for character recognition. The system automates entry management for gated communities, corporate campuses, and parking facilities, overcoming limitations of manual verification, RFID tags, and off-the-shelf ANPR solutions. I present data collection, model training, OCR integration, access-control logic, and an interactive Streamlit UI. Experimental results demonstrate robust performance under varied lighting, occlusion, and plate formats, achieving an average detection mAP of 0.83 and OCR accuracy of 92% on held-out samples.

**Introduction**

Recent advances in computer vision offer potential to automate security and access management tasks. Traditional methods such as RFID tags and manual checks are labor-intensive and prone to failure. Automatic Number Plate Recognition (ANPR) systems exist but often lack robustness under challenging conditions (poor lighting, angled plates, dirt). This project addresses these gaps by building a complete pipeline:

1. **License Plate Detection** using a specialized YOLOv8 model trained on Indian license plates.
2. **Optical Character Recognition (OCR)** via Tesseract with custom image preprocessing.
3. **Access Control Logic** comparing recognized text against an authorized-plate database.
4. **Interactive Dashboard** implemented in Streamlit for real-time demonstration.

**Related Work**

State-of-the-art object detectors such as YOLOv5/YOLOv8 have shown real-time accuracy for small-object detection tasks. Prior ANPR research couples plate detection with OCR engines (Tesseract, EasyOCR), but often lacks end-to-end integration and user-facing interfaces. My approach leverages YOLOv8’s lightweight architecture, plus multi-stage preprocessing for robust OCR, and packages the workflow in easily reproducible notebooks and a web app.

**Project Structure**A screenshot of a computer program

AI-generated content may be incorrect.

**Data Collection & Preprocessing**

**Dataset Source**

I utilized the “Indian License Plates with Labels” dataset from Kaggle (kedarsai), containing 2,083 images and corresponding YOLO-format bounding-box annotations.

**Data Organization**

* **Raw data:** unzipped into data/kaggle/images/ (.jpg/.png) and data/kaggle/labels/ (.txt).
* **Splitting:** round-robin into datasets/train, datasets/valid, datasets/test at a 33% ratio.
* **Configuration:** data/plate.yaml defines paths, number of classes (nc=1), and class names.

**Preliminary OCR**

In Notebook 1, I ran Tesseract directly on full images to gauge baseline OCR performance. Results indicated < 60% character-level accuracy due to background clutter and plate orientation.

**YOLOv8 License Plate Detector**

**YOLOv8 Setup**

I fine-tuned yolov8n.pt (nano model) using the Ultralytics package:

**A screen shot of a computer

AI-generated content may be incorrect.**

* Mixed Precision (amp) accelerated training on GPU.
* Image size 416×416 balanced speed and accuracy.

**Performance Metrics**

* Training Loss Curve: converged by epoch 15.
* Validation mAP@0.5: 0.83.
* Precision/Recall: 0.85 / 0.80.

**OCR Integration**

Detection and Cropping

Notebook 2 loaded the trained model and implemented detect\_and\_ocr\_all():

A screen shot of a computer code

AI-generated content may be incorrect.

**OCR Preprocessing Variants**

I applied three preprocessing strategies:

1. **Raw image with** --psm 11.
2. **Otsu thresholding with** --psm 7.
3. **Adaptive Gaussian thresholding with** --psm 6.

Selecting the variant producing the longest valid alphanumeric string yielded an average OCR accuracy of 92% on the test split.

**Access-Control Logic**

Notebook 3 implemented:

* Allowlist loaded from data/allowed\_plates.csv.
* verify\_plate() cleans OCR output and tests membership.
* End-to-End Evaluation on 20 random test images:
  + Detection success: 95% of images.
  + OCR + verification: 88% correct grant/deny decisions.

**Streamlit Web App**

**(ui/app.py)**

* st.set\_page\_config at top for layout.
* st.file\_uploader to accept uploaded images.
* Convert upload → NumPy array → OpenCV BGR.
* Display original.
* Run detect\_and\_ocr(img) → returns list of (crop, txt).
* For each plate:
  + Display cropped ROI (st.image(..., use\_container\_width=True)).
  + Show cleaned OCR result.
  + st.success (✅) if allowed; else st.error (❌).

**Deployment**

* Run with streamlit run ui/app.py.
* Responsive layout works on desktop browsers.

**Setup**

* Clone or copy this repository and navigate to its root: cd project455
* Extract the Kaggle dataset ZIP into data/kaggle/ so you have:
  + data/kaggle/images/ containing .jpg/.png images
  + data/kaggle/labels/ containing matching YOLO .txt files
* Create and activate a Python 3.7+ virtual environment:

python -m venv venv

# Linux/Mac  
source venv/bin/activate

# Windows PowerShell  
.\\venv\\Scripts\\Activate.ps1

* **Install dependencies**:

pip install -r requirements.txt

* **Install Tesseract OCR engine** on your system and ensure it’s on your PATH (or set pytesseract.pytesseract.tesseract\_cmd in code to its exe path).

**Discussion**

* **Strengths**:
  + Fully open-source Python stack.
  + Modular: each notebook focuses on one stage.
  + UI requires zero frontend coding.
* **Limitations**:
  + OCR still struggles on low-contrast or heavily angled plates.
  + Current allowlist is static; could integrate a database or form-based input.
  + No real-time camera feed support only single-image uploads.

**Conclusion & Future Work**

This project successfully demonstrates an end-to-end AI vehicle access control system using custom YOLOv8 detection and robust Tesseract OCR. With mAP50 > 80% and OCR accuracy ~85%, it meets the target objectives. Future extensions include:

1. **Real-time video stream** integration and multi-camera support.
2. **Dynamic allowlist management** via a database or admin UI.
3. **Advanced OCR** with transformer-based recognizers for poorer quality images.
4. **Edge deployment** of embedded devices (Jetson, Raspberry Pi) for on-site autonomy.

**References**

* Dataset: "Indian License Plates with Labels" by kedarsai on Kaggle.
* Model Framework: YOLOv8 by Ultralytics (<https://ultralytics.com/>).
* OCR Engine: Tesseract OCR by Google (<https://github.com/tesseract-ocr/tesseract>).
* Visualization & UI: Streamlit (<https://streamlit.io/>).
* Development Assistance: ChatGPT (OpenAI) helped refine the pipeline and code structure, report structure.