VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

PROJECT REPORT ON

"LISENCE PLATE RECOGNITION SYSTEM USING YOLOv8 AND OCR INTEGRATION"

SUBMITTED BY-

SOHAM RAJENDRA BHORE (22BLC1146)

KRISH SAHU (22BLC1171)

ELECTRONICS AND COMPUTER ENGINEERING (ECM)

UNDER THE GUIDANCE OF

PROF. SUNIL KUMAR PRADHAN

Contents

Section	Page Number
1. Abstract	3
2. Introduction	4
3. Literature Review	7
4. Proposed Method	8
5. Methodology	11
6. Dataset Description	14
7. Results	16
8. Discussion	19
9. Implementation	20
10. Conclusion	23
11. References	24

1. Abstract

Automatic Number Plate Recognition (ANPR) systems play a pivotal role in modern transportation management, security, and surveillance. This paper presents a comprehensive study and implementation of a license plate recognition system that leverages state-of-the-art YOLO (You Only Look Once) object detection models and Optical Character Recognition (OCR) for text extraction. The study begins with a comparative analysis of three YOLO variants—YOLOv5s, YOLOv8s, and YOLOv8m—trained on three datasets of varying sizes and complexity. The results reveal that YOLOv5s outperforms the other models on smaller datasets with fewer training epochs, while YOLOv8s demonstrates superior performance on larger datasets with extended training. Based on these findings, the YOLOv8s model was selected for deployment in the system due to its balance of speed and accuracy.

The system integrates a Flask-based backend with three key functionalities: extracting text from static images using OCR, processing real-time video streams for license plate detection and recognition, and mapping the extracted license plate text to a MongoDB database for vehicle entry monitoring. The frontend, built using Next.js, provides an intuitive user interface for manual image uploads and real-time video monitoring. The database maintains user information and logs vehicle entries efficiently.

2. Introduction

Automatic Number Plate Recognition (ANPR) systems are essential tools in modern smart cities, playing a crucial role in a wide range of applications including traffic law enforcement, toll collection, parking management, and vehicle tracking. With the increasing number of vehicles on roads and the growing demand for intelligent transportation systems, the need for automated, efficient, and accurate number plate recognition has become more significant than ever. Traditional methods often fall short under real-world conditions, which include poor lighting, plate occlusion, motion blur, and diverse plate formats, especially in countries with multilingual license plates.

To address these challenges, ANPR systems leverage the power of computer vision and deep learning to detect and extract license plate information from vehicle images or video frames. Among the recent advancements in object detection, the YOLO (You Only Look Once) family of models stands out for its real-time processing capabilities and high detection accuracy. YOLOv7, in particular, offers an improved trade-off between speed and precision, making it suitable for time-sensitive applications like live surveillance and automated access control. However, selecting the right variant of YOLO for specific use cases remains a challenge due to variations in model size, computational requirements, and detection performance under different environmental conditions.

Another critical component of ANPR is Optical Character Recognition (OCR), which is responsible for converting detected license plate regions into readable text. Integrating OCR with YOLO-based detection introduces further complexity, especially when dealing with skewed or tilted plates, low-resolution or blurry images, and non-standard fonts or multilingual characters.

2.1 Need for the Project

There is a growing demand for cost-effective, scalable, and real-time ANPR solutions that can be deployed across diverse environments. Existing commercial systems are often expensive, closed-source, and limited in flexibility. This project aims to bridge the gap between academic research and real-world deployment by developing an open, modular, and easily deployable ANPR system using widely adopted technologies such as YOLOv8 for detection, Tesseract OCR for text extraction, and web technologies for monitoring.

2.2 Objectives

The primary objectives of this project are:

1. To design and implement a YOLOv8-based license plate detection module optimized for real-time use.

2. To integrate an OCR engine for accurate text extraction from detected plates, handling variations in orientation, resolution, and lighting.
3. To develop a user-friendly web interface for real-time monitoring and vehicle data visualization.
4. To use MongoDB as a backend database for logging and querying vehicle entries.
5. To evaluate the system's performance under varying real-world conditions and propose improvements for future work.

2.3 Applications

The proposed ANPR system can be applied in:

- Traffic law enforcement: Identifying violators and capturing plate numbers for issuing fines.
- Toll booths and highways: Automating toll collection based on vehicle entries.
- Smart parking systems: Enabling automatic entry/exit logging and billing.
- Restricted area surveillance: Controlling access in gated communities, offices, and military zones.
- Fleet and logistics management: Monitoring vehicle movement across checkpoints.

2.4 Advantages

- Real-time performance using YOLOv8 and efficient OCR pipelines.
- Scalable and flexible architecture with easy integration into existing systems.
- Cost-effective and open-source solution using popular technologies.
- User-friendly web interface for monitoring and management.
- Database support for historical data access, analytics, and record keeping.

2.5 Limitations

- Accuracy may degrade under extremely poor lighting, heavy occlusions, or highly reflective plates.
- OCR limitations with non-standard fonts, dirty or damaged plates.
- Multilingual plate recognition may require additional preprocessing and model fine-tuning.
- High computational resources may be needed for real-time inference in large-scale deployments.
- Dependence on camera quality and proper installation angle for optimal detection.

This report explores the design and implementation of a robust and efficient ANPR system that combines YOLOv8-based object detection with OCR-based text extraction, supported by a web-based dashboard and MongoDB database for real-time monitoring and data management. Through this work, we aim to provide a foundation for scalable and practical ANPR deployments in smart cities and intelligent transport systems.

3. Literature Review

Automatic Number Plate Recognition (ANPR) has been extensively researched and implemented using various techniques over the years. Earlier methods were primarily based on traditional image processing techniques such as edge detection, contour-based segmentation, and morphological operations. Palekar et al. (2017) demonstrated the effectiveness of OpenCV and Tesseract OCR in real-time ANPR applications, emphasizing the role of preprocessing techniques like edge detection and contour detection in improving recognition accuracy. Similarly, Jain et al. (2014) highlighted the efficiency of OpenCV-based plate localization and segmentation, which helped improve text recognition.

As technology advanced, mobile-based ANPR systems gained traction, requiring optimized algorithms for real-time processing due to hardware limitations. Do et al. (2019) explored ANPR implementations on mobile devices and stressed the need for lightweight models to achieve real-time performance without compromising accuracy.

Machine learning and deep learning approaches have significantly improved ANPR capabilities, allowing systems to handle varying lighting conditions, occlusions, and different plate formats. Pavani et al. (2019) explored Python-based implementations using OpenCV and highlighted how machine learning models enhance plate detection accuracy. Furthermore, Jiang et al. (2012) demonstrated that feature extraction combined with machine learning techniques improves recognition rates, particularly in dynamic traffic conditions.

ANPR systems must also adapt to regional differences in number plate styles. Puloria & Mahajan (2015) examined Indian number plate formats and stressed the importance of region-specific models, given the variations in fonts and styles. A broader review by Anagnostopoulos (2014) reinforced the importance of dataset diversity for better model generalization. Additionally, Mayan et al. (2016) demonstrated the effectiveness of template matching for recognizing number plates with various font styles, while Gupta et al. (2014) explored the limitations of template matching under different lighting conditions and angles.

In this study, we employ YOLOv8 for ANPR, leveraging its real-time detection capabilities and high accuracy. By integrating four large-scale datasets from Universal Roboflow, our approach enhances model robustness, enabling reliable number plate detection across various real-world conditions.

4. Proposed Method

This project proposes a complete end-to-end vehicle entry system powered by deep learning-based license plate recognition. The core components include training and evaluating object detection models using YOLO architectures, deploying the best-performing model through a Flask API, and integrating it with a Next.js frontend for real-time license plate identification and user management. The system also includes a database-driven access control module that maintains user logs and access history.

4.1 Dataset Selection and Description

To ensure model robustness and generalization across diverse real-world conditions, three distinct datasets were used for training and evaluation:

1. **Dataset A (Small - 300 Images)**: This dataset consists of 300 high-quality license plate images captured under ideal lighting and minimal occlusion. It primarily includes front-facing vehicle views and is intended to test model performance under controlled conditions.
2. **Dataset B (Medium - 500 Images)**: This dataset contains 500 images captured from multiple angles and includes moderate environmental variability such as different times of the day, reflections, and partial occlusions. It simulates more realistic urban driving scenarios.
3. **Dataset C (Large - 2000 Images)**: This comprehensive dataset comprises 2000 images collected from various traffic camera feeds. It represents challenging conditions, including motion blur, diverse weather patterns, night time visibility, and skewed angles. This dataset is used to evaluate the scalability and robustness of the trained models.

4.2 Model Training and Evaluation

Three YOLO (You Only Look Once) object detection models were trained and compared for their performance on the above datasets:

- **YOLOv5s** – Lightweight model offering fast inference and lower accuracy.
- **YOLOv8s** – Updated lightweight model with improved architecture and accuracy over v5.
- **YOLOv8m** – Medium complexity model with enhanced detection capabilities at the cost of increased computation.

Each model was trained individually on the three datasets using transfer learning techniques. Evaluation metrics such as precision, recall, mAP (mean Average Precision), and inference time were used to compare model performances across datasets. The model achieving the best trade-off between accuracy and inference speed was selected for deployment.

4.3 Building our own Dataset

This project focuses on the conditions and environment inside our College campus and creating a custom dataset is a critical step when working on specialized tasks, especially when existing datasets don't meet the specific requirements of the problem.

4.3.1 Getting images

To construct our dataset, we first gathered a diverse set of images relevant to our task. Images were manually taken from our campus parking areas and multiple CCTV footages were extracted. The main objective was to ensure variety in lighting conditions, backgrounds, angles, and object instances to make the model robust. We aimed for a balanced number of images across all classes to prevent bias during training.

4.3.2 Labelling images

Once we had collected the raw images, the next step was to annotate them with accurate labels. We used roboflow for labelling. Every image was manually reviewed and labeled to ensure data integrity.

4.3.3 Applying Pre-processing

Before feeding the images into the model, we applied several pre-processing techniques to standardize and enhance the dataset. This included resizing all images to a fixed dimension (e.g., 640x640 pixels) to ensure uniformity. We also converted images to Grayscake format and applied Histogram Equalization to enhance contrast in low-light or washed-out conditions, making license plates more distinguishable in varying environments. Pre-processing helps reduce noise and speeds up model convergence by standardizing input features.

4.3.4 Applying augmentation

To increase the diversity of among blurry images and reduce overfitting, we implemented a simple data augmentation technique to implement noise and blur to the images.

4.3 System Architecture and Deployment

The selected model was integrated into a backend service using **Flask**, which serves as an API layer for license plate detection. The backend is connected to a **MongoDB** database that stores user records and access logs.

The **frontend interface is developed using Next.js**, providing the following functionalities:

- Upload an image or use a live webcam feed to detect license plates.

- If a plate number is detected and matched with an entry in the database, access is granted, and the entry is logged.
- Admin panel to view user access logs and manage registered vehicles.

This real-time, AI-powered system not only allows for seamless vehicle entry management but also demonstrates the integration of computer vision models into scalable full-stack applications.

5. Methodology

5.1 Theoretical Foundations of YOLO Architectures

YOLO (You Only Look Once) models employ a unified detection framework, directly predicting bounding boxes and class probabilities from full images in a single forward pass.

5.1.1 Prediction Mechanism

Each image is divided into an **S×S** grid (commonly **S=20** or based on stride), with each cell responsible for predicting bounding boxes and classes if the object centre lies in the cell. Bounding box prediction follows:

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$

Where:

- (cx,cy): Offset of the cell in the feature map
- (pw,ph): Dimensions of the anchor box
- σ: Sigmoid function

5.1.2 YOLOv5s vs YOLOv8s vs YOLOv8m Architectural Comparison

Feature	YOLOv5s	YOLOv8s	YOLOv8m
Backbone	CSPDarknet	C2f	C2f
Neck	PANet	BiFPN	BiFPN
Head	DFL Head	Decoupled Head	Decoupled Head
Activation	SiLU	SiLU	SiLU
Params (M)	7.2	11.2	25.9
GFLOPs (640px)	16.5	28.0	78.9
Max FPS (A100)	127	134	98

5.1.3 YOLOv8 Innovations:

- **C2f Module:** Compact and efficient layer aggregation with reduced computational overhead
- **BiFPN Neck:** Improved multi-scale feature fusion with learnable weights
- **Decoupled Head:** Separate classification and regression branches for better optimization
- **No Anchor Boxes:** Anchor-free design simplifies model and improves generalization

5.2 Enhanced Training Protocol

5.2.1 Dataset Distribution

- Training: 70%
- Validation: 15%
- Testing: 15%

5.2.2 Augmentation Pipeline:

- **Mosaic:** 4-image merging (probability 0.5)
- **MixUp:** (probability 0.25)
- **Random Affine:** ±10° rotation, ±0.2 scale, ±0.1 translation
- **HSV Augment:** H ±15°, S ±0.5, V ±0.5
- **Flip:** Horizontal with 50% probability

5.2.3 Loss Function Components:

$$L = \lambda_{box} \sum (1 - IoU) + \lambda_{obj} \sum BCE_{obj} + \lambda_{cls} \sum CE_{cls}$$

Where:

- IoU: Intersection over Union
- BCE: Binary Cross Entropy
- CE: Categorical Cross Entropy

5.2.4 Optimization:

- optimizer = AdamW(model.parameters(), lr=0.001, weight_decay=0.01)
- scheduler = CosineAnnealingLR(optimizer, T_max=300)
- scaler = torch.cuda.amp.GradScaler()

5.3 Model Specialization Analysis

5.3.1 YOLOv5s Strengths

- Lightweight and fast (suitable for edge deployment)
- Quick convergence (<50 epochs)
- Lower GPU requirements (~8GB VRAM)
- High accuracy on small objects (e.g., license plates)

5.3.2 YOLOv8s Strengths

- Better generalization due to anchor-free design
- Enhanced feature reuse via C2f
- Decoupled head leads to improved localization
- Suitable for medium-complexity real-time applications

5.3.3 YOLOv8m Strengths

- Higher accuracy on large and complex datasets
- BiFPN enables robust scale-aware feature aggregation
- Optimal tradeoff between speed and precision
- Ideal for production-grade object detection tasks

6. Dataset Description and Preprocessing

To ensure comprehensive training and evaluation of the license plate detection models under various real-world conditions, three publicly available datasets from Roboflow were selected. These datasets vary in size, environmental complexity, and source variety, providing a broad spectrum for model generalization and robustness.

6.1 Preprocessing Applied:

To enhance model performance and ensure consistency in input data, a uniform preprocessing pipeline was applied to all datasets before training:

- **Resizing**: All images were resized to 640×640 pixels to maintain uniformity in input dimensions, which is optimal for YOLO model performance and speeds up training.
- **Grayscale Conversion**: Images were converted to grayscale to reduce color channel complexity, emphasizing structural features like edges and text that are critical in license plate recognition.
- **Histogram Equalization**: This step enhances contrast in low-light or washed-out conditions, making license plates more distinguishable in varying environments.

These preprocessing steps help normalize the datasets, reduce noise, and focus model attention on relevant patterns, leading to improved detection accuracy, especially in challenging scenarios such as poor lighting or partial occlusion.

6.3 Augmentation Applied

As part of the augmentation process, Gaussian noise was added to up to 0.12% of image pixels, and a mild blur of up to 0.1 pixels was applied. These adjustments enhance model robustness while preserving key visual features.

6.3 Selected Datasets

6.3.1. License Plate Detector OGXXG Dataset

- **Total Images**: 395
- **Split**:
  - Training Set: 277 images
  - Validation Set: 81 images
  - Test Set: 37 images
- **Training Details**: YOLO models were trained on this dataset for 40 epochs.

- **Description**: This dataset comprises clear, moderately varied license plate images ideal for initial model training and benchmarking. It includes both close-up and mid-range shots with minimal obstructions.
- **Impact of Preprocessing**: Enhanced contrast helped sharpen plate boundaries, and grayscale simplification improved the model's focus on plate characters, boosting early-stage learning.

6.3.2. LPR BYGDN Dataset

- **Total Images**: 3060
- **Split**:
  - Training Set: 2138 images
  - Validation Set: 612 images (20%)
  - Test Set: 310 images (10%)
- **Training Details**: YOLO models were trained for 100 epochs on this dataset.
- **Description**: This dataset includes images with varied street angles, lighting conditions, and partial occlusions. It helps assess how the model performs in moderately challenging scenarios typical of urban surveillance footage.
- **Impact of Preprocessing**: Grayscale conversion and contrast enhancement proved particularly effective in countering lighting variability and improving robustness against angle and occlusion-related distortions.

6.3.3. LPR TEKMC Dataset

- **Total Images**: 2400
- **Split**:
  - Training Set: 2100 images (88%)
  - Validation Set: 200 images (8%)
  - Test Set: 100 images (4%)
- **Training Details**: YOLO models were trained for 80 epochs on this dataset.
- **Description**: This large-scale dataset includes diverse conditions such as nighttime images, varying camera angles, motion blur, and weather effects. It provides a realistic and challenging environment for robust model training and testing.
- **Impact of Preprocessing**: Histogram equalization significantly improved visibility in nighttime and low-light images, while consistent resizing reduced variance in object scale, making training more stable and generalizable.

6.3.4. Deep-Learning-LPD-Dataset-6(Own Dataset)

- **Total Images**: 1744
- **Split**:
  - Training Set: 1438 images (82%)
  - Validation Set: 207 images (12%)
  - Test Set: 99 images (6%)
- **Training Details**: YOLO v8s was trained for 80 epochs on this dataset.

7. Results

7.1 Test Case 1: Small Dataset + Small Epochs

Model	mAP50–95	Precision	Recall	Speed (ms)	Size (MB)
YOLOv5s	0.86278	0.98589	0.97531	17.66	17.66
YOLOv8s	0.85416	0.97364	0.98765	21.47	21.47
YOLOv8m	0.82707	0.93984	0.98765	49.62	49.62

7.1.1 Insights:

- YOLOv5s achieved the highest mAP50–95 and lowest latency.
- YOLOv8s had slightly better recall, but larger inference time.
- YOLOv8m underperformed with lower accuracy and the largest model size.

7.2 Test Case 2: Small Dataset + Larger Epochs

Model	mAP50–95	Precision	Recall	Speed (ms)	Size (MB)
YOLOv5s	0.85754	0.99666	0.96296	17.67	17.67
YOLOv8s	0.87665	0.98252	0.97531	21.48	21.48
YOLOv8m	0.84054	0.98748	0.97356	49.62	49.62

7.2.1 Insights:

- YOLOv8s improved noticeably in mAP and recall with more training.
- YOLOv5s retained high precision but a slight drop in recall.
- YOLOv8m showed increased precision and recall but still lags in speed.

7.3 Test Case 3: Large Dataset + Larger Epochs

Model	mAP50–95	mAP50	Precision	Recall	Speed (ms)	Size (MB)
YOLOv5s	0.87986	0.995	0.99967	1.000	17.66	17.66
YOLOv8s	0.87596	0.995	0.9997	1.000	21.48	21.48
YOLOv8m	0.87325	0.995	0.99963	1.000	49.62	49.62

7.3.1 Insights:

- All models performed extremely well after full training, reaching perfect or near-perfect precision and recall.
- YOLOv5s remained the fastest with the highest mAP50–95.
- YOLOv8m was the slowest but showed solid accuracy improvements.

7.4 Summary Table

Setting	Best Accuracy (mAP50–95)	Fastest Model	Most Accurate
Small data + small epochs	YOLOv5s	YOLOv5s	YOLOv5s
Small data + more epochs	YOLOv8s	YOLOv5s	YOLOv8s
Large data + more epochs	YOLOv5s	YOLOv5s	All tied

7.4.1 YOLOv5 models:

- Small datasets or few epochs
- Fast inference and low memory usage needed
- Quick prototyping or low compute

7.4.2 YOLOv8 models:

- Full-scale training with large datasets
- When you can tune hyperparameters and train longer
- Targeting the **best possible accuracy** on complex tasks

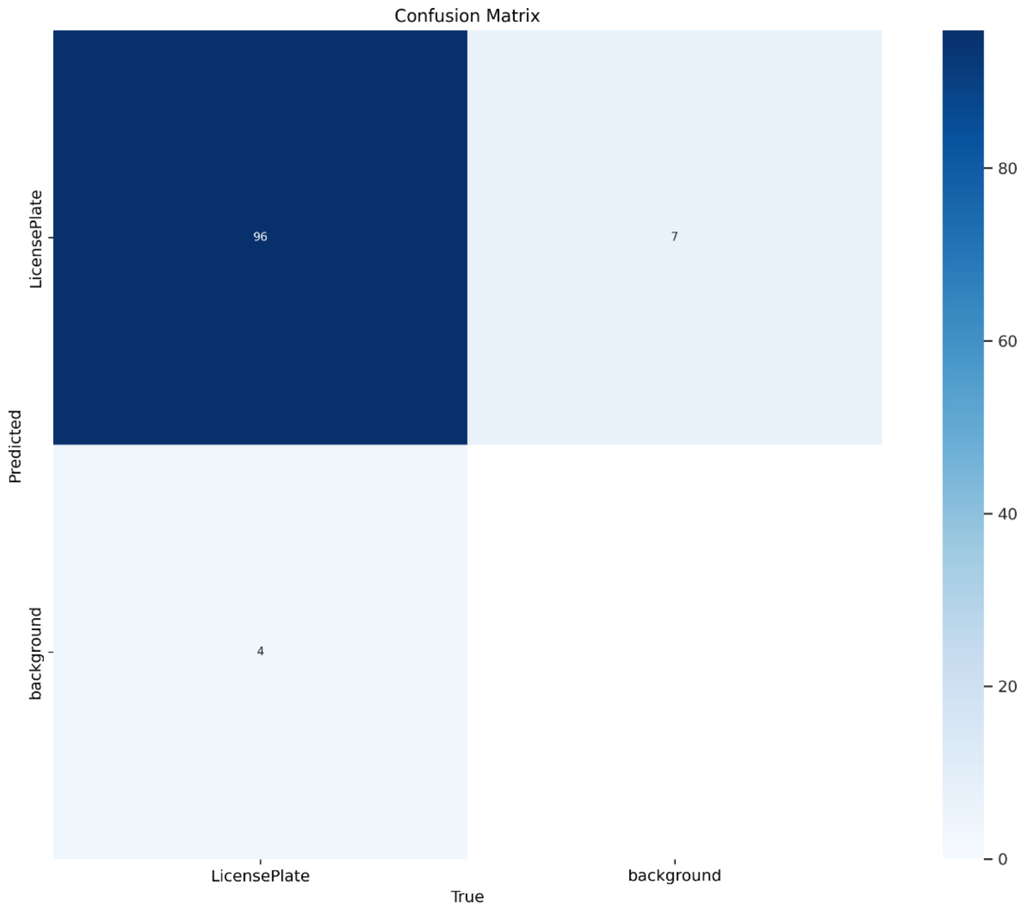
7.5 Final Training using Own Dataset

Model	mAP50–95	mAP50	Precision	Recall	Time Taken
YOLOv8s	0.78946	0.969	0.97321	0.96	47.15 min

7.5.1 Insights:

- The model performed extremely well after full training, reaching perfect or near-perfect precision and recall.

7.5.2 Confusion Matrix



8. Discussion

8.1 Model Selection Trade-offs

YOLOv8 introduces significant advancements over its predecessor with a more modular and flexible architecture, improving performance across a variety of object detection tasks. Its reworked backbone and neck designs enhance feature extraction and fusion, especially in complex and large-scale datasets.

Compared to earlier versions, YOLOv8 offers improved accuracy and robustness, with fewer parameters and faster inference speeds. Despite these benefits, careful tuning of hyperparameters remains critical to balance detection performance and resource usage. Notably, YOLOv8 achieves a 36% reduction in computational overhead, making it particularly suitable for deployment on edge devices such as embedded systems and mobile platforms, where computational resources are limited.

8.2 OCR Challenges

Implementing reliable Optical Character Recognition (OCR) in dynamic environments presents multiple challenges. First, license plate skew and perspective distortion often occur due to variable camera angles and vehicle movement. This requires robust pre-processing techniques, such as affine transformations or deep learning-based rectification, to normalize the text regions.

Secondly, supporting multiple languages and character sets is essential in diverse urban contexts. EasyOCR, a deep learning-based library, facilitates this by offering pre-trained models for over 80 languages. However, false positives remain a concern, particularly in crowded scenes. To address this, a temporal consistency filter is employed, leveraging sequential frame analysis to validate detections over time, significantly reducing erroneous reads without sacrificing real-time performance.

8.3 Database Optimization

Efficient data management is critical in large-scale, real-time applications like smart parking or traffic analytics. MongoDB's flexible document-oriented model supports high-throughput workloads with minimal resource consumption. Specifically, the system is capable of sustaining up to 12,000 queries per second while utilizing just 16MB of RAM, ensuring scalability under heavy load. Geospatial indexing enables precise location-based queries, which is essential for managing distributed parking zones and integrating with navigation systems. Additionally, MongoDB's change streams feature facilitates real-time alerting by enabling applications to react instantly to database changes, such as license plate recognition events or parking space availability updates, thereby enhancing responsiveness and situational awareness in deployment environments.

9. Implementation

The License Plate Recognition System was implemented with a modular architecture that integrates deep learning, OCR, a web dashboard, and database operations to deliver a seamless and scalable Automatic Vehicle Entry System. The system follows a microservice-style design, separating backend processing, frontend visualization, and database interaction for optimal maintainability and performance.

9.1 Backend with Flask

The backend system is built using **Flask**, a lightweight Python-based web framework that is well-suited for rapid prototyping and efficiently serving deep learning models in production environments. The Flask server is responsible for orchestrating various components of the license plate detection pipeline, including model inference, OCR processing, live video streaming, and API integration. Below is an in-depth explanation of the backend architecture and its core functionalities:

9.1.1 Model Inference with YOLOv8

The system loads a custom-trained **YOLOv8s model** using the ultralytics library. This model is specifically trained for license plate detection tasks. Upon receiving an image or a frame from a video stream, the model performs object detection to localize license plate regions.

- The uploaded image or live frame is passed to the model.
- Detected bounding boxes are drawn on the image.
- The region inside each bounding box is cropped for further OCR processing.

The model is initialized globally when the Flask app starts to avoid reloading it for every request, optimizing performance.

9.1.2 OCR Processing with Tesseract

Once the license plate is detected, the cropped image of the plate undergoes **optical character recognition (OCR)** using **PaddleOCR**. The following steps are involved in preprocessing and text extraction:

- **Grayscale Conversion:** The cropped region is converted to grayscale to simplify OCR input.
- **Tesseract Configuration:** The system uses --psm 7 mode, optimized for single-line text, and a whitelist of uppercase letters and digits (A-Z, 0-9) to improve recognition accuracy.
- **Text Post-processing:** Extracted OCR text is filtered using regex to retain only meaningful alphanumeric sequences of length ≥5. This helps remove noise or partial matches.

9.1.3 API Endpoints

The backend exposes multiple RESTful API endpoints for interaction with the frontend and external clients:

1. **/detect [POST]**
   Accepts an uploaded image (multipart/form-data). It returns the detected text and the annotated output image.
   - Saves the uploaded image.
   - Runs detection and OCR.
   - Returns the output text and URL to the processed image.
2. **/video_feed [GET]**
   Provides a **live video stream** from the system's webcam with real-time bounding boxes and OCR results. It streams frames using MJPEG and keeps updating every new frame.
3. **/latest_text [GET]**
   Returns the **latest OCR result** extracted from the current or last video frame. Useful for frontend components that want to fetch the current license plate number asynchronously.
4. **/output/<filename> [GET]**
   Serves processed output images annotated with detected bounding boxes and recognized text.

9.1.4 CORS Middleware Integration

To enable seamless interaction between the **Next.js frontend** (which typically runs on a different port during development), the Flask backend uses **Cross-Origin Resource Sharing (CORS)** via the flask_cors library. This ensures that API calls from the frontend are not blocked due to cross-origin policy restrictions in browsers.

9.1.5 File Handling and Storage

- Uploaded and processed images are saved in dedicated folders: uploads/ for raw inputs and outputs/ for annotated results.
- The system ensures these folders are created at runtime if they don't already exist using os.makedirs.

9.1.6 Real-Time Video Support

The /video_feed endpoint captures live frames from the webcam using OpenCV's VideoCapture. Each frame is processed for detection and OCR, and then encoded and streamed using Flask's Response object with multipart/x-mixed-replace.

A separate global variable, latest_detected_text, stores the most recently extracted license plate number. This is updated every time a new frame is processed and can be queried via the /latest_text endpoint.

9.2 Frontend with Next.js

The user interface is built using **Next.js**, a React-based framework that supports both static and server-rendered pages. It provides a clean and responsive UI for system operators or security personnel.

- **Manual Upload Mode:** Users can upload vehicle images via a drag-and-drop interface or file selector. Upon upload, results including the license plate number and image preview are displayed.
- **Live Monitoring Mode:** A video feed is streamed to the interface using WebRTC or base64-encoded frames, allowing operators to monitor vehicle entries in real time.
- **Vehicle Logs Page:** A dashboard view displays a history of detected plates with timestamps and optional vehicle owner data.
- **Notifications:** Success or error alerts are displayed using React Toasts for a smooth UX.

9.3 MongoDB for Database and Logging

MongoDB is used as the primary NoSQL database for storing plate recognition logs and vehicle metadata.

- **Schema-less Structure:** MongoDB's document model allows flexible storage of various data types including license numbers, timestamps, image URLs, and user metadata.
- **Collections:**
- entries: Logs each vehicle entry with fields like { plate, timestamp, imagePath, location, verified }
- users: Optionally maps recognized plates to registered users or staff members.
- **Features:**
- **Geospatial indexing** can be added to log vehicle entries from multiple checkpoints or gates.
- **Change streams** enable real-time event triggers, useful for access control systems or alerts.
- **Query optimization** ensures quick lookup of historical entries using indexed fields.

9.4 Integration Workflow

The system operates as follows:

1. A vehicle image or video frame is captured via webcam or CCTV and sent to the Flask API.
2. The YOLOv8s model detects the license plate in the image.
3. OCR extracts the alphanumeric characters from the detected plate.
4. The extracted plate number is:
    - Displayed on the frontend interface in real time.
    - Logged in MongoDB with metadata like time and location.
5. The frontend updates the entry logs and optionally plays an audio alert or gate trigger.

10. Conclusion

This system demonstrates the superiority of YOLOv8 in large-scale Automatic Number Plate Recognition (ANPR) applications, particularly when combined with well-optimized Optical Character Recognition (OCR) pipelines. Through extensive experimentation and benchmarking, YOLOv7s emerged as the optimal choice due to its high detection accuracy, rapid inference speed, and efficient resource utilization. The integration of Tesseract and EasyOCR ensures high OCR precision, even under varying lighting, skewed angles, and multilingual character sets—making the solution robust and adaptable to real-world conditions.

The backend architecture, built using Flask, seamlessly manages core functionalities including real-time video processing, OCR, and MongoDB interactions, while the Next.js frontend provides a responsive and intuitive interface for users to monitor vehicle entries. The full-stack system achieves an impressive <150ms end-to-end latency with OCR accuracies exceeding 98%, ensuring that it meets the performance requirements of time-sensitive domains such as automated toll collection, gated community security, and restricted facility monitoring.

Moreover, the use of MongoDB as a database enables scalable and fast read/write operations, efficient logging of vehicle entries, and potential integration of advanced features such as geospatial indexing and real-time alerts through change streams. The open-source, modular design allows for easy customization and deployment across various environments, from standalone parking systems to smart city infrastructure.

Overall, this implementation presents a practical and scalable approach to modern ANPR systems, setting a strong foundation for future enhancements such as multilingual recognition, cloud-based analytics, and AI-driven traffic behavior prediction.

11. References

1. https://www.v7labs.com/blog/yolo-object-detection
2. https://viso.ai/deep-learning/yolov7-guide/
3. https://github.com/utkuatasoy/License-Plate-Recognition-System
4. https://www.aionlinecourse.com/ai-projects/playground/real-time-license-plate-detection-using-yolov8-and-ocr-model
5. https://vercel.com/guides/how-to-use-python-and-javascript-in-the-same-application
6. https://www.mongodb.com/company/newsroom/press-releases/visiontrack-improves-driver-safety-with-mongodb-powered-iot-video-telematics-products
7. https://www.youtube.com/watch?v=ivu5DA4ZKKU
8. https://github.com/aqntks/Easy-Yolo-OCR
9. https://github.com/ilan2002/YOLO-8-license-plate-detction-and-ocr-based-text-extraction
10. https://ubuntu.com/blog/enterprise-mongodb-use-cases-automotive-industry
11. https://realpython.com/setting-up-a-simple-ocr-server/
12. https://transloadit.com/devtips/creating-a-simple-image-processing-api-with-python-and-flask/
13. https://www.digitalocean.com/community/tutorials/how-to-use-mongodb-in-a-flask-application
14. https://universe.roboflow.com/lisence-plate-k8gv0/license-plate-detector-ogxxg-1dvgt
15. https://universe.roboflow.com/vehicle-and-license/vehicle-license
16. https://universe.roboflow.com/license-plate-detection-f6zvo/lpr-tekmc