Smart Vision: Automatic Number Plate Recognition using YOLOv11 and OpenOCR

Samith Kamarthi
Team Member
SmartVision Project
New York, USA
skamarthi@mercy.edu

***Abstract*—Manual extraction of vehicle license plate numbers from video footage is a labor-intensive process that scales poorly for real-world applications such as traffic monitoring, parking management, and post-incident investigation. Raw video streams frequently present visual challenges, including motion blur, variable lighting, and partial occlusions, which degrade the performance of simple frame-by-frame inspection. This paper presents "Smart Vision," a privacy-centric local web application designed to automatically detect and recognize license plates. The system implements a sequential machine learning pipeline utilizing YOLOv11 for object detection and OpenOCR (ONNX) for text recognition. Evaluated on the Roboflow License Plate Recognition dataset (10,125 images), the detection model achieved a Precision of 0.9893 and a Mean Average Precision (mAP@50) of 0.9813. The system ensures reproducibility and usability by generating annotated video overlays and structured JSON logs containing per-frame detection metrics.**

***Index Terms*—ALPR, YOLOv11, OpenOCR, Object Detection, Computer Vision, Edge AI.**

I. Introduction

A. Problem Definition

The ability to accurately identify vehicles is a cornerstone of modern intelligent transportation systems (ITS). Automatic License Plate Recognition (ALPR) has become essential for applications ranging from automated toll collection and smart parking systems to law enforcement and access control. However, manual verification of license plates from video footage is slow, error-prone, and difficult to scale [1]. In high-volume use cases, such as monitoring a busy campus entrance or auditing industrial traffic, the sheer volume of video data makes human review impractical.

Furthermore, real-world video data is rarely ideal. Uncontrolled environments introduce significant visual noise. Common challenges include motion blur caused by speeding vehicles, extreme changes in lighting (e.g., glare from headlights, deep shadows, or low-light night recording), varied camera angles, and partial occlusions by other vehicles or obstacles [2]. These factors significantly reduce the reliability of basic Optical Character Recognition (OCR) tools when applied directly to a video frame without a dedicated localization stage [44].

B. Proposed Solution

This project, titled "Smart Vision," addresses these challenges by developing a robust, automated pipeline for detecting and reading license plates from uploaded videos. Unlike cloud-based solutions that raise data privacy concerns, Smart Vision is designed as a local web application. This approach minimizes deployment complexity and ensures that sensitive video data remains on-premise [3], [8].

The system objectives are threefold:

1) **Robust Localization:** Accurately locate license plate regions within complex video frames, regardless of the vehicle's position [4].
2) **Accurate Text Extraction:** Extract alphanumeric text using a specialized OCR pipeline capable of handling standard license plate fonts [5].
3) **Actionable Output:** Produce verifiable outputs, including visual overlays for human verification and structured JSON data for downstream analytics and database integration [6], [7].

II. Related Work

ALPR systems have evolved significantly over the past two decades. Early approaches relied heavily on traditional image processing techniques. These methods typically involved edge detection (e.g., Sobel, Canny), morphological operations, and connected component analysis to locate rectangular shapes within an image. While computationally inexpensive, these "hand-crafted" feature extractors were highly sensitive to lighting changes and complex backgrounds, often failing when the plate was skewed or dirty [43].

With the advent of Deep Learning, Convolutional Neural Networks (CNNs) have replaced traditional methods as the state-of-the-art. Object detection models such as SSD (Single Shot MultiBox Detector) and the YOLO (You Only Look Once) family have demonstrated superior performance in localizing plates in real-time [41]. Recent studies indicate that YOLO-based detectors offer an optimal trade-off between inference speed and detection accuracy, making them suitable for video processing tasks where high frame rates are required [43].

This project builds upon these advancements by utilizing the latest iteration, YOLOv11, combined with a dedicated OCR engine (OpenOCR) running on the ONNX runtime. This combination allows for hardware-accelerated inference without the overhead of heavy deep learning frameworks in the deployment environment.

III. System Architecture

The system relies on a two-stage sequential pipeline: Object Detection followed by Text Recognition. This modular approach allows for independent optimization of the localization and recognition tasks.

A. Stage 1: License Plate Detection (YOLOv11)

The first stage employs YOLOv11, implemented via the Ultralytics library [15], [42].

1) Model Rationale: YOLOv11 was selected for its balance of speed and accuracy. License plates are relatively small objects within a full 1080p or 4K frame, occupying a small percentage of the total pixels. YOLO architectures are optimized to learn robust visual features for such localization tasks while maintaining high inference speeds suitable for video processing [22], [23]. The model treats object detection as a regression problem, predicting bounding boxes and class probabilities directly from full images in a single evaluation.

2) Inference Implementation: The detection model is loaded from a '.pt' weights file. The system supports loading local weight files or automatically downloading references from the Hugging Face Hub, improving reproducibility [18], [19]. For each frame, the inference call is executed as follows:

```
results = yolo.predict(frame, conf=0.25, device=
    device)
```

The output includes the bounding box coordinates (x, y, w, h), a confidence score ('score'), and a class ID ('class_id') [16], [17], [20].

B. Stage 2: Text Recognition (OpenOCR)

The second stage utilizes OpenOCR with an ONNX (Open Neural Network Exchange) runtime backend [24].

1) Crop and Recognize: Once a bounding box is identified by YOLO, the region of interest (ROI) is cropped from the original frame. This cropping step is crucial as it removes background noise, allowing the OCR engine to focus solely on the alphanumeric characters. The engine is initialized using:

```
OpenOCR(backend="onnx", device=..., drop_score=0.0)
```

The OpenOCR pipeline typically abstracts two sub-tasks: text detection (refining the text area within the crop to handle rotation) and text recognition (decoding the character sequences) [25].

2) Post-Processing: The raw text returned by the OCR engine often contains artifacts or misinterpretations of special characters. To address this, a specialized normalization function, '_safe_plate_text()', is applied. This function:

- Converts all text to uppercase to ensure consistency.
- Strips all non-alphanumeric characters (e.g., dashes, spaces, symbols) [27], [28].

This ensures the final output adheres to standard license plate syntax, facilitating database lookups.

IV. Software Implementation

The theoretical pipeline is wrapped in a practical software application designed for local deployment.

A. Job Processing & State Management

Processing high-resolution video is computationally intensive and time-consuming. To prevent web timeouts and ensure a responsive user experience, the application implements an asynchronous background job system. When a user uploads a video, a job is created with a unique ID. The system tracks the state of the job through a lifecycle: 'queued' → 'running' → 'done' (or 'error') [39]. Users can poll the status of their job via API endpoints.

B. Data Management

Outputs are organized structurally on the file system to facilitate logging and retrieval. Each job is assigned a dedicated directory 'jobs/¡job_id¿/' [40]. This folder contains:

- The original input video file.
- The processed 'annotated.mp4' with visual overlays.
- The 'result.json' log file containing machine-readable data.

C. Structured Output (JSON)

For integration with external analytics dashboards or security databases, the system generates a 'result.json' file. This artifact provides a granular breakdown of the detection events:

- **Metadata:** Global video properties such as FPS, resolution (width/height), and the model path used for inference [34].
- **Timeline:** An array of entries for every detected frame, including the 'frame_index' and 'time_sec' to map detections to specific moments in the video [35].
- **Detections:** Detailed objects containing the bounding box 'box', detection 'score' (confidence), and the cleaned 'text' string [36].

D. Hardware Acceleration & Fallback

The system is designed to be hardware-agnostic to support various deployment environments. Both YOLO (via PyTorch) and OpenOCR (via ONNX Runtime) are configured to dynamically check for GPU availability.

- If 'CUDAExecutionProvider' is detected and the relevant GPU libraries are installed, the system defaults to device '0' for acceleration. This significantly reduces inference time per frame [21], [26], [45].
- If GPU resources are unavailable, the system gracefully falls back to CPU execution rather than failing [37].

Additionally, the system includes validation logic to detect incorrect 'openocr' module installations—a common dependency issue—and returns specific error messages to guide the user in troubleshooting [38].

V. Dataset

The project utilized the "License Plate Recognition" dataset (Version 11) from Roboflow Universe [9].

A. Characteristics

The dataset consists of 10,125 images of vehicles captured in diverse real-world conditions [10]. It focuses on a single object class: 'License_Plate' [11], [14]. The images vary significantly in terms of:

- **Lighting:** Day, night, direct sunlight, and shadows.
- **Angle:** Frontal, rear, and oblique views.
- **Distance:** Varying scales of license plates relative to the frame size.

Crucially, no artificial augmentations were applied to this version of the dataset. This ensures that the model was trained on the natural variability of the source images rather than synthetic distortions, promoting robustness in real-world scenarios [13].

B. Data Split

To ensure rigorous evaluation and prevent overfitting, the dataset was partitioned into standard machine learning splits [12]:

- **Training Set:** $\approx$ 70% (7,057 images) used for model weight updates.
- **Validation Set:** $\approx$ 20% (2,048 images) used for hyper-parameter tuning during training.
- **Test Set:** $\approx$ 10% (1,020 images) reserved for final performance evaluation.

VI. Experiments and Results

The YOLOv11 detection model was evaluated on the Test Set (1,020 images). The training metrics over 25 epochs are visualized in Fig. 1.

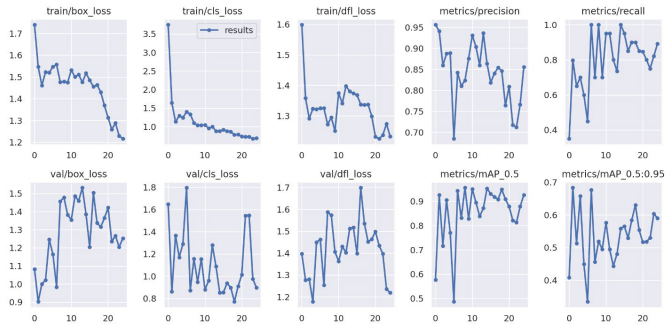

Fig. 1. Training and validation metrics over 25 epochs. Top row: Training losses (Box, Class, DFL). Bottom row: Validation losses and metrics (Precision, Recall, mAP). Note the steady decrease in validation box loss and the stabilization of mAP@0.5.

A. Evaluation Metrics

The performance was measured using standard object detection metrics: Precision, Recall, and Mean Average Precision (mAP) at different Intersection over Union (IoU) thresholds. The final results on the test set are summarized in Table I.

TABLE I
YOLOv11 Performance Metrics (Test Set)

Metric	Value
Precision	0.9893
Recall	0.9508
mAP @ 0.50 IoU	0.9813
mAP @ 0.50-0.95 IoU	0.7260

B. Analysis of Classification

The model's classification performance is further detailed in the Confusion Matrix (Fig. 2). The matrix indicates a high degree of separation between the target class and the background. The model achieved a normalized score of 0.80 for the true positive identification of the "licence" class, with minimal confusion with the background class (0.20), suggesting a low rate of false negatives.

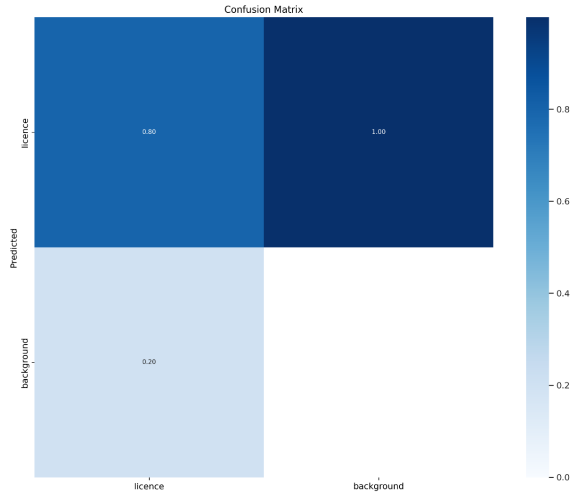

Fig. 2. Normalized Confusion Matrix for the License Plate class. The strong diagonal dominance indicates robust classification performance.

VII. Discussion

A. Precision vs. Recall

The model achieved a Precision of 0.9893, indicating an exceptionally low false-positive rate [29]. In the context of ALPR, high precision is desirable as it ensures that the system does not trigger downstream processes (like OCR) on non-plate objects (e.g., bumper stickers or road signs). This efficiency is critical for real-time video processing. The Recall of 0.9508 confirms that the model detected approximately 95% of all plates present in the test set [30]. While 5% of plates were missed, visual inspection suggests these are likely cases of extreme occlusion or motion blur where the plate is visually indistinguishable.

B. Localization Accuracy

A key insight from the results is the disparity between mAP@50 (0.9813) and mAP@50-95 (0.7260) [33].

- **mAP@50 (98.1%):** This near-perfect score confirms the model almost never misses a plate entirely. It successfully places a bounding box with at least 50% overlap on the target.
- **mAP@50-95 (72.6%):** This metric averages performance across stricter overlap thresholds (up to 95%). The drop to 72.6% suggests that while detection is robust, there is minor variance in the "pixel-perfect" tightness of the bounding boxes.

However, in a two-stage pipeline, this level of accuracy is substantial. The subsequent OCR stage typically requires a small margin around the text to function correctly, so extremely tight bounding boxes are not strictly necessary, provided the text itself is not cropped out [32].

C. Loss Convergence

Referring to Fig. 1, the 'val/box_loss' and 'val/dfl_loss' (Distribution Focal Loss) show a consistent downward trend, indicating that the model effectively learned to regress the bounding box coordinates. The 'metrics/mAP_0.5' curve rises sharply within the first 5 epochs and stabilizes, demonstrating that the model converges quickly, which is a characteristic strength of the YOLOv11 architecture.

VIII. Conclusion

This paper presented "Smart Vision," a comprehensive local ALPR solution integrating YOLOv11 and OpenOCR. The system successfully addresses the challenges of manual license plate verification by providing a robust, automated pipeline capable of handling real-world video variability. With a detection precision of 98.9% and a flexible software architecture that supports GPU acceleration and structured JSON reporting, the tool is well-suited for deployment in parking management and security auditing contexts.

The system's modular design allows for future enhancements. Future work will focus on:

- Reducing the bounding box variance to improve mAP@50-95 scores, potentially through more aggressive augmentation during training.
- Optimizing the OCR engine for extremely skewed viewing angles (e.g., top-down views from high-mounted CCTV).
- Exploring real-time streaming (RTSP) capabilities to allow live analysis of IP camera feeds beyond file uploads.

References

[1] Smart Vision Doc, "Problem Definition," Source 5.
[2] Smart Vision Doc, "Raw Video Challenges," Source 6.
[3] Smart Vision Doc, "Project Scope," Source 7.
[4] Smart Vision Doc, "System Requirement 1," Source 9.
[5] Smart Vision Doc, "System Requirement 2," Source 10.
[6] Smart Vision Doc, "System Requirement 3," Source 11.
[7] Smart Vision Doc, "Output Artifacts," Source 12.
[8] Smart Vision Doc, "Deployment," Source 13.
[9] Roboflow Universe, "LP Dataset," Source 15.
[10] Smart Vision Doc, "Dataset Size," Source 19.
[11] Smart Vision Doc, "Dataset Annotations," Source 20.
[12] Smart Vision Doc, "Dataset Splits," Source 21.
[13] Smart Vision Doc, "Augmentation," Source 34.
[14] Smart Vision Doc, "Single Class," Source 36.
[15] Smart Vision Doc, "YOLOv11," Source 43.
[16] Smart Vision Doc, "YOLO Output," Source 47.
[17] Smart Vision Doc, "Confidence Score," Source 48.
[18] Smart Vision Doc, "Model Loading," Source 52.
[19] Smart Vision Doc, "Hugging Face," Source 53.
[20] Smart Vision Doc, "Inference Call," Source 55.
[21] Smart Vision Doc, "GPU Selection," Source 58.
[22] Smart Vision Doc, "Why YOLO," Source 59.
[23] Smart Vision Doc, "YOLO Balance," Source 60.
[24] Smart Vision Doc, "OpenOCR," Source 63.
[25] Smart Vision Doc, "OCR Pipeline," Source 76.
[26] Smart Vision Doc, "ONNX GPU," Source 82.
[27] Smart Vision Doc, "Text Cleaning," Source 86.
[28] Smart Vision Doc, "Safe Plate Text," Source 87.
[29] Smart Vision Doc, "Precision," Source 101.
[30] Smart Vision Doc, "Recall," Source 105.
[31] Smart Vision Doc, "mAP 50-95," Source 112.
[32] Smart Vision Doc, "Bounding Box Tightness," Source 113.
[33] Smart Vision Doc, "Performance Summary," Source 115.
[34] Smart Vision Doc, "JSON Metadata," Source 122.
[35] Smart Vision Doc, "JSON Timeline," Source 123.
[36] Smart Vision Doc, "JSON Detection Fields," Source 124.
[37] Smart Vision Doc, "CPU Fallback," Source 133.
[38] Smart Vision Doc, "Dependency Validation," Source 135.
[39] Smart Vision Doc, "Job Queue," Source 138.
[40] Smart Vision Doc, "Job Directory," Source 139.
[41] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 779–788.
[42] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLO," Jan. 2023. [Online]. Available: https://github.com/ultralytics/ultralytics
[43] R. Laroca, E. Severo, L. A. Zanlorensi, L. S. Oliveira, G. R. Gonçalves, W. R. Schwartz, and D. Menotti, "A Robust Real-Time Automatic License Plate Recognition Based on the YOLO Detector," in *2018 International Joint Conference on Neural Networks (IJCNN)*, 2018, pp. 1–10.
[44] S. M. Silva and C. R. Jung, "License Plate Detection and Recognition in Unconstrained Scenarios," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 580–596.
[45] Microsoft, "ONNX Runtime," [Online]. Available: https://onnxruntime.ai/