A YOLO model for Car License Plates Detection

Ngoc Nguyen[1][0000−1111−2222−3333]

Trinity University, One Trinity Place, San Antonio, TX 78212, USA

Abstract. The abstract should briefly summarize the contents of the paper in 150–250 words.

Keywords: Object detection · optical character recognition · license plate recognition · convolutional neural network.

1 Introduction

License Plate Recognition (LPR) plays a crucial role in modern Intelligent Transportation Systems (ITS), with applications ranging from traffic monitoring and highway toll management to parking enforcement, stolen vehicle identification, and digital security surveillance. Recent advancements in LPR research have focused on developing high-performing, real-time models; however, most studies limit their scope to specific countries. Given the increasing importance of managing cross-border traffic and the diverse license plate formats worldwide, creating LPR models trained on geographically diverse datasets becomes essential for ensuring robust performance in different regions.

In this paper, we propose the use of YOLO for license plate detection. To enhance the model's versatility, we compile a dataset featuring license plate images from various countries, including China, Taiwan, and Brazil, among others.

2 Literature Review

License Plate Recognition (LPR) has become essential in modern Intelligent Transportation Systems (ITS) due to its diverse applications, such as traffic monitoring, toll collection, and vehicle security. Researchers have developed various approaches to enhance LPR, typically divided into traditional algorithms and deep learning-based methods.

Traditional algorithms in LPR rely on techniques like AdaBoost, Support Vector Machine (SVM) classifiers, and cascade models that use features such as Haar and Histogram of Oriented Gradients (HOG) [4]. These methods have been effective, but with the rise of deep learning, Convolutional Neural Networks (CNNs) have gained popularity for their ability to learn complex spatial patterns. In particular, multi-task CNN models have been introduced to address real-world challenges in LPR, including variations in lighting, weather conditions, and occlusions. One such model, MTLPR, integrates multiple tasks—like plate classification, boundary box regression, and color recognition—into a single

framework, achieving recognition accuracy up to 98% by using a large dataset of Chinese license plates [4].

Another innovative model, RPnet, unifies detection and recognition tasks into a single processing pass. This integration reduces processing time and boosts accuracy by allowing the detection and recognition stages to share convolutional features. When evaluated on the comprehensive Chinese City Parking Dataset (CCPD), RPnet achieved 98.5% accuracy with a processing speed of over 60 frames per second, making it highly efficient for real-time applications [5].

To further optimize LPR for real-time use, especially in mobile and low-power devices, PP-OCRv3 was introduced. This ultra-lightweight Optical Character Recognition (OCR) system supports a range of text recognition tasks, including LPR, through the SVTR-LCNet, a hybrid CNN-Transformer model that combines CNN's speed with Transformer's contextual processing capabilities. PP-OCRv3 also incorporates components like LK-PAN and RSE-FPN to refine text detection without sacrificing speed, making it highly suitable for deployment in resource-constrained environments. Tests reveal that PP-OCRv3 improves detection accuracy by 5% over previous versions, delivering a robust and efficient OCR solution for LPR [2].

These studies highlight the evolution from traditional multi-stage LPR approaches to advanced end-to-end deep learning frameworks that better balance accuracy, speed, and robustness. The development of datasets like CCPD, alongside architectures like MTLPR, RPnet, and PP-OCRv3, addresses the critical challenges in LPR for diverse and unpredictable environments. Collectively, these innovations push LPR technology toward more adaptable, high-performance solutions suited to various deployment settings.

3 Problem Definition

This research aims to improve license plate recognition (LPR) in two key areas: geographical diversity of license plate images and processing speed. Much of the existing LPR research focuses on license plate images from specific regions, limiting the models' ability to generalize across a variety of plate styles and patterns. Consequently, the proposed models in this study are designed to handle a more geographically diverse set of license plate images, addressing this gap.

Furthermore, many LPR-related research projects suffer from inaccessible or poorly documented source code, making it difficult to replicate results or build upon existing models. One notable exception is the study "A Light CNN for End-to-End Car License Plates Detection and Recognition" [4], which provides accessible source code. This study implements a lightweight, robust two-stage model using MTCNN and LPRNet in PyTorch. While this approach achieves an image processing speed of approximately 80 ms per image on a Nvidia Quadro P400, this speed — while reasonable given the hardware limitations — is inadequate for high-speed applications like live video processing. Therefore, there is room for improvement in processing speed to meet the demands of real-time LPR systems.

4 The proposed method

4.1 Datasets

To develop a more versatile license plate detection model capable of handling plates from various regions, we propose training the model on license plate images sourced from a diverse set of regions. Potential datasets include the CCPD (Chinese City Parking Dataset) [5], UFPR-ALPR [1], and Tunisian License Plates datasets.

CCPD (Chinese City Parking Dataset) Overview: The Chinese City Parking Dataset (CCPD) is one of the largest publicly available ALPR datasets, featuring over **250,000 unique car images**. Each image is annotated with license plate location annotations, making it an invaluable resource for training ALPR models. The dataset includes a wide range of license plate types, environments, and perspectives, drawn from **real-world urban settings** in China. The extensive variety of images in CCPD, including different car models, angles, and environmental conditions, makes it an excellent dataset for developing and testing robust, scalable ALPR systems. The CCPD dataset was introduced by **Q. Zhang et al.** in the paper *"CCPD: A Large-Scale Dataset for License Plate Recognition in the Wild"* [5].

UFPR-ALPR Overview: The UFPR-ALPR dataset is a comprehensive collection of **4,500 fully annotated images**, containing over **30,000 license plate characters**. It was curated from **150 vehicles** in **real-world scenarios**, providing a rich variety of license plates captured in different conditions, such as varying lighting, angles, and backgrounds. This dataset is particularly valuable for training and evaluating Automatic License Plate Recognition (ALPR) systems due to its diverse and realistic nature, making it suitable for addressing the challenges faced in real-world license plate detection and recognition tasks. The dataset was introduced by **L. M. G. B. Oliveira et al.** in their paper titled *"UFPR-ALPR: A Dataset for License Plate Recognition in Real-World Scenarios"* [1].

4.2 Model Type

Next, we propose using the YOLO (You Only Look Once) model to streamline the image processing pipeline and improve detection speed. YOLO is a state-of-the-art, real-time object detection system known for its fast and accurate performance [3]. In this research, we develop YOLOv8-based models trained on these diverse datasets, evaluate their accuracy and processing speed, and compare the results with those of existing models from related research.

5 Model Evaluation

The model will be evaluated based on a set of metrics related to both **accuracy** and **processing speed**. Specifically, the **accuracy metrics** include Precision, mAP50 (mean Average Precision at an IoU threshold of 0.50), and mAP50-95 (mean Average Precision averaged over IoU thresholds from 0.50 to 0.95). The speed metrics encompass Preprocess, Inference, Loss, and Postprocess. A detailed explanation of these metrics is provided in the below sections.

5.1 Accuracy Metrics

- **Precision**: Precision measures how accurate the positive predictions of a model are. In this case, it reflects the proportion of correct predictions out of all instances where the model detected a license plate in an image. A higher precision means fewer *false positives*—incorrect predictions where the model mistakenly identifies a license plate when there is not one. For example, if the model detects 100 license plates and 90 of them are correct, the precision would be 0.9 or 90%. This indicates the model is correct 90% of the time when it detects a license plate.
- **mAP50** (mean Average Precision at IoU threshold 0.50): mAP50 is a key metric used to assess how accurately the model detects objects and places bounding boxes around them. It calculates the average of the precision values at an Intersection over Union (IoU) threshold of 0.50.
  - **IoU** (Intersection over Union): IoU measures the overlap between the predicted bounding box and the actual ground truth bounding box. An IoU of 0.50 means that the predicted bounding box must overlap at least 50% with the ground truth to be considered a correct detection.
  - **Average Precision (AP)**: Average Precision measures the precision across different recall levels, evaluating how accurate the detected objects are.
  - **mAP50**: mAP50 represents the average precision at an IoU threshold of 0.50, indicating how well the predicted bounding boxes overlap with the true bounding boxes. For example, in the image below, the model returns an IoU of 86.63% for the detected license plate. Since this value exceeds the 50% threshold, the prediction is considered correct.
- **mAP50-95** (mean Average Precision averaged over IoU thresholds from 0.50 to 0.95): mAP50-95 evaluates the average precision across a range of IoU thresholds from 0.50 to 0.95, increasing by 0.05 at each step. This metric assesses how well the model performs under more stringent overlap conditions, providing a more detailed evaluation of detection accuracy.

5.2 Speed Metrics

- **Preprocess**: Preprocess refers to the time (in seconds) taken for data preprocessing before the model begins inference.

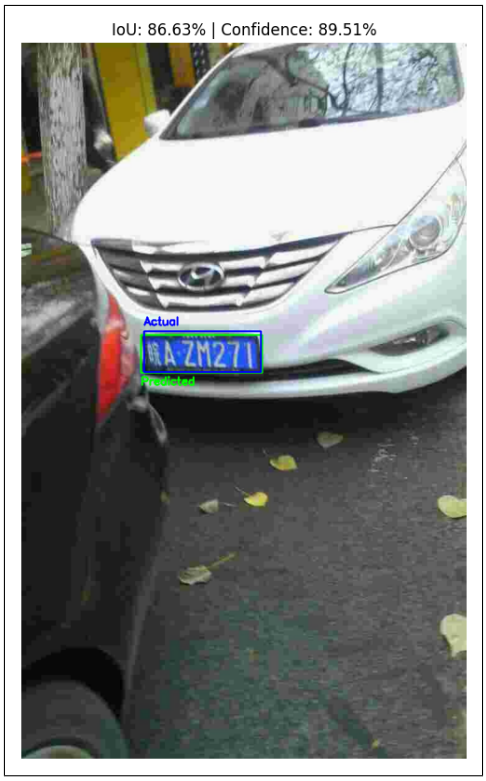

Fig. 1. Example of IoU between the predicted and actual bounding box.

- **Inference**: Inference is the time the model takes to make predictions on the data during inference.
- **Loss**: Loss is a measure of how well the model's predictions align with the actual labels. A lower loss indicates better performance.
- **Postprocess**: Postprocess refers to the time (in seconds) spent on postprocessing tasks, such as refining, transforming, or preparing the results after the model has made predictions.

6 Simulation results

The YOLO model has been trained and evaluated in four main ways, using two different datasets-CCPD and CCPD dataset:

- **Model 1**: Trained using 200 images, with 60 images for validation and 30 images for testing from the CCPD dataset.
- **Model 2**: Trained using 1000 images, with 300 images for validation and 200 images for testing from the CCPD dataset.

- **Model 3**: A further refinement of Model 2, achieved by customizing the model's hyperparameters. Originated from Model 3, Model 3.1 is tested and fine-tuned using the CCPD dataset dataset.
- **Model 4**: Trained using 1800 images, with 900 images for validation and 1800 images for testing from the UFPR-ALPR dataset. Originated from Model 4, Model 4.1 is tested and fine-tuned using the CCPD dataset.

Considering both accuracy and processing speed provides a comprehensive understanding of the models' performance. The accuracy and speed metrics for the models are presented in the two tables below.

Table 1. Summary of model accuracy metrics.

Model	**Train/Test**	**Precision(B)**	**mAP50(B)**	**mAP50-95(B)**
Model 1	Test	1.0	0.9559	0.636
	Test	1.0	0.995	0.7005
Model 2	Train	0.9992	0.995	0.7235
	Test	0.999	0.995	0.7324
Model 3	Train	0.9996	0.995	0.7499
	Test	1.0	0.995	0.729
Model 3.1	Train	0.9775	0.97341	0.6326
	Test	0.37728	0.12689	0.0578
Model 4	Train	0.9756	0.99365	0.6899
	Test	0.96268	0.98187	0.70983
Model 4.1	Train	1.0	0.995	0.70179
	Test	0.9992	0.995	0.6756

Table 2. Summary of model speed metrics.

Model	**Train/Test**	**Preprocess**	**Inference**	**Loss**	**Postprocess**
Model 1	Train	0.9546	464.01922	5.3439e-05	4.6332
	Test	2.1148	365.7802	0.00037e-05	4.0029
Model 2	Train	0.6569	349.1737	3.2584e-05	1.5148
	Test	0.6	396.2358	7.5102e-05	0.9032
Model 3	Train	0.5663	189.4467	1.2716e-05	0.2178
	Test	1.1345	384.6922	0.000437	0.44921
Model 3.1	Train	0.8607	353.0874	0.0008	0.7465
	Test	0.6117	191.8582	0.00021	0.39129
Model 4	Train	0.3608	150.1464	2.225e-05	0.2449
	Test	0.382	155.8975	4.7286e-05	0.23342
Model 4.1	Train	0.6968	196.422	0.0001	0.66772
	Test	0.89457	218.851	9.1791e-05	0.57458

In general, the models produce accurate detection results, as evidenced by the high Precision and mAP50 metrics. Although the mAP50-95 scores, ranging between 0.626 and 0.7499, are generally considered good, there is room for improvement, particularly under stricter conditions.

Fig. 2. Example of outputs from Model 2.

The relatively high mAP50 scores combined with lower mAP50-95 scores suggest that while the models perform well at detecting license plates with a loose overlap (around 50%), they struggle when stricter overlap conditions are required (closer to 95%). This discrepancy likely indicates that the models are proficient at detecting plates in simpler scenarios but may not have encountered enough challenging examples—such as small or overlapping objects—that demand more precise bounding boxes. Another possible explanation is that the models may perform well with easier detection tasks (e.g., large objects or images with clear backgrounds) but face difficulties with smaller or more complex details.

Additionally, the fitness scores for most models fall between 0.72 and 0.76. Fitness, often used in optimization algorithms to measure how well a model's performance aligns with its goals, indicates that while the models are performing reasonably well, they are not yet fully optimized. Therefore, further adjustments or fine-tuning could enhance their performance, especially in more challenging detection scenarios.

Interestingly, increasing the training set from 200 to 1000 images to develop the detection models on CCDP improves the accuracy in strict thresholds. This is proven by the testing mAP50-95 scores of 0.7005 and 0.7324 for Model 1 and Model 2, respectively. However, the effort to fine-tune Model 2 results in Model 3 showing slightly reduce performance in terms of accuracy. Although the precision score increases and the speed metrics improve significantly, the mAP50-95 values slightly decrease for the test dataset (from 0.7324 to 0.729). This suggests that hyperparameter tuning may have caused the model to become somewhat overfitted. Reducing the extent of the hyperparameter changes could help address this issue.

Table 3. Summary of model tuning process using CCDP dataset.

Hyperparameter	**Model 2**	**Model 3**
Optimizer	AdamW	SGD
Learning Rate	0.002	0.01
Momentum	0.9	0.937
Weight Decay	0.0005	0.001
Training Epochs	5 epochs	10 epochs

The key changes in the model lie in the optimizer, learning rate (lr0), momentum, weight decay, and training epochs. Specifically, we switched the optimizer from the default AdamW to SGD (Stochastic Gradient Descent). The learning rate is increased from 0.002 to 0.01, with the aim of accelerating training. The weight decay (L2 regularization) is introduced to help prevent overfitting by penalizing large weights. The slight increase in momentum (from 0.9 to 0.937) in the tuned model is intended to help the optimizer make more consistent progress toward the minimum. Additionally, the weight decay for the weights is slightly increased (from 0.0005 to 0.001) to improve generalization and reduce the risk of overfitting to the training data. The number of training epochs is also doubled, from 5 to 10, to give the model more time to learn. While these changes result in faster training, they have also led to instability in more detailed and stringent evaluations, suggesting the need for further fine-tuning.

Moreover, since the detection models were developed separately for the CCPD and UFPR-ALPR datasets, we also cross-test the models on the other dataset to evaluate their generalization performance.

Table 4. Accuracy Metrics of Cross-Tuned Models

Model	**Development Method**	**Training Metrics**	**Testing Metrics**
Model 3.1	Originally trained with CCPD, tuned with UFPR-ALPR.	Precision (B): 0.9775 mAP50 (B): 0.97341 mAP50-95 (B): 0.6326	Precision (B): 0.37728 mAP50 (B): 0.12689 mAP50-95 (B): 0.0578
Model 4.1	Originally trained with UFPR-ALPR, tuned with CCPD.	Precision (B): 1.0 mAP50 (B): 0.995 mAP50-95 (B): 0.70179	Precision (B): 0.9992 mAP50 (B): 0.995 mAP50-95 (B): 0.6756

While both models perform accurately on their respective training sets, the detection results from Model 3.1 are less satisfactory on the test sets. Here are the key observations from the cross-tuning process:

– **Model 3.1:** This model is initially trained on the CCPD dataset and later fine-tuned with the UFPR-ALPR dataset. Despite the tuning, Model 3.1 demonstrates poor performance on the UFPR-ALPR test set, as indicated by its low precision, mAP50, and mAP50-95 scores. This result can be attributed to the model being initially optimized for the CCPD dataset, which

may differ significantly from the UFPR-ALPR dataset in terms of object types, image quality, or annotation styles.

– **Model 4.1:** In contrast, this model is initially trained on the UFPR-ALPR dataset and subsequently fine-tuned with the CCPD dataset. Model 4.1 performs exceptionally well on the CCPD test set, achieving high scores in precision, mAP50, and mAP50-95. Furthermore, it exhibits strong performance across both the training and test sets, indicating superior generalization capabilities. This suggests that the UFPR-ALPR dataset provided robust foundational features, enabling the model to adapt effectively to the CCPD dataset during fine-tuning.

The performance discrepancy between Model 4.1 and Model 3.1 underscores the importance of the initial training dataset and its influence on cross-dataset tuning.

Fig. 3. Example of images in CCPD datasets.

Fig. 4. Example of images in UFPR-ALPR datasets.

The superior performance of Model 4.1 in the cross-dataset tuning process compared to Model 3.1 suggests that the UFPR-ALPR dataset has a stronger impact when used for initial training. This advantage may stem from the better alignment of features or more generalizable characteristics of the UFPR-ALPR dataset, which enabled the model to adapt effectively to the CCPD dataset during fine-tuning. In contrast, the underperformance of Model 3.1 can likely be attributed to the CCPD dataset's limited capacity to provide transferable features for cross-dataset tuning. Consequently, the model struggles to generalize when tested on the UFPR-ALPR dataset after fine-tuning. These results highlight the importance of selecting an appropriate initial training dataset for cross-dataset tuning. Specifically, beginning with a dataset that offers more generalizable features, such as UFPR-ALPR, followed by fine-tuning on a more diverse or challenging dataset, appears to yield better outcomes, as demonstrated by the success of Model 4.1.

Finally, considering the image processing speed, the reported processing times for our models range from 0.36 seconds to 2.11 seconds per image during testing, which is significantly longer than 80 ms (0.08 seconds) reported in the existing research. However, the hardware used for model inference plays a significant role in processing speed. The Nvidia Quadro P400, which is used in the mentioned research, is a dedicated GPU, which is optimized for parallel processing and deep learning tasks. In contrast, our model are developed with the Apple M2 chip, which is a system-on-a-chip (SoC) with an integrated GPU. While the Apple M2 chip is powerful and efficient for many tasks, it may not be optimized as well for deep learning operations compared to a dedicated GPU like the Nvidia Quadro P400. This hardware difference is likely contributing to the slower inference times for your models.

7 Conclusion

References

1. Rayson Laroca, Evair Severo, Luiz A. Zanlorensi, Luiz S. Oliveira, Gabriel Resende Gonçalves, William Robson Schwartz, and David Menotti. A robust real-time automatic license plate recognition based on the yolo detector. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10, 2018.
2. Chenxia Li, Weiwei Liu, Ruoyu Guo, Xiaoting Yin, Kaitao Jiang, Yongkun Du, Yuning Du, Lingfeng Zhu, Baohua Lai, Xiaoguang Hu, Dianhai Yu, and Yanjun Ma. Pp-ocrv3: More attempts for the improvement of ultra lightweight ocr system, 2022.
3. Rejin Varghese and Sambath M. Yolov8: A novel object detection algorithm with enhanced performance and robustness. In *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*, pages 1–6, 2024.
4. Wanwei Wang, Jun Yang, Min Chen, and Peng Wang. A light cnn for end-to-end car license plates detection and recognition. *IEEE Access*, 7:173875–173883, 2019.
5. Zhenbo Xu, Wei Yang, Ajin Meng, Nanxue Lu, Huan Huang, Changchun Ying, and Liusheng Huang. Towards end-to-end license plate detection and recognition: A large dataset and baseline. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 261–277, Cham, 2018. Springer International Publishing.