License Plate Detection and Recognition for Automatic Parking Lots

Asmany Akter
Dept. of Computer Science
Hof University of Applied Sciences
Hof, Germany
asmany2016@gmail.com

Iana Costa Sten
Dept. of Computer Science
Hof University of Applied Sciences
Hof, Germany
icsten@gmail.com

Leonardo Anyelo Rodriguez Martinez
Dept. of Computer Science
Hof University of Applied Sciences
Hof, Germany
leonardorm7@hotmail.com

***Abstract*—Automated parking lots have gained popularity since they are faster than manual or semi-automated parking lots. At the same time, they are more convenient for users who do not have to keep a ticket or a card that can be lost. However, automated systems implement special cameras designed specifically for license plate detection. In this project, a system is proposed that is capable of working with any camera through the implementation of artificial intelligence models. The system utilizes YOLOv9c, YOLOv8n, YOLOv5n, and Faster R-CNN, which were fine-tuned to determine the most suitable model for the license plate detection task in this project. For character extraction, models such as pre-trained EasyOCR, Paddle OCR and a fine-tuned EasyOCR model were implemented with our own labeled dataset containing 834 images. The models for license plate detection were fine-tuned with a dataset of 1042 images, achieving an accuracy of 94.9% for image detection and 69.61% for character extraction. To implement these models in a complete system, were used web development tools: Flask for the backend, SQLite3 for the database, and React for the frontend. The code can be found in the following GitHub repository.**

***Index Terms*—automated parking systems, YOLO models, OCR, computer vision, artificial intelligence**

I. Introduction

Parking management systems have become more important nowadays, due to the increasing number of vehicles. Efficient parking management not only saves time, but also reduces traffic congestion. Over time, parking systems have progressed greatly, moving from traditional manual approaches and towards advanced automated solutions. Parking systems can be categorized into three main types: manual, semi-automated and automated [1]. Each type of parking system offers different advantages and disadvantages, which must be carefully considered when developing efficient solutions.

Manual parking systems are simple and low-cost, but they have several drawbacks. They significantly rely on human intervention, which subjects them to errors and delays. Manually processing vehicle entries and exits is time-consuming and impractical for handling large numbers of vehicles. Furthermore, manual solutions lack the scalability required for modern parking lots, where efficiency is the most important.

Semi-automated systems include technical innovations, such as ticket booths. Those systems help reduce the reliance on staff members and increase processing speeds. Its solutions provide a middle ground between manual and automated procedures. However, they also face some challenges. The machines require regular maintenance, and the difficulty caused by missing tickets might lead to user dissatisfaction. Furthermore, semi-automated systems frequently produce paper waste, which is not environmentally favorable.

The most advanced type of parking management are fully automated systems. They use modern technology, such as cameras and software for recognition, enabling them to detect and manage without human intervention. They are very effective in identifying unauthorized cars and, in the long run, are cost effective. In contrast to these benefits, they are not recommended for small-scale facilities, since they have high initial setup costs and are dependent on complex technology.

We are proposing a system which addresses some of the limitations of the existing solutions, enabling us to detect license plates using any standard camera. Additionally, the proposed systems use open source technologies to detect and recognize the vehicles with no expensive software. This approach significantly reduces the installation costs, and therefore are recommended for any type of facility, especially the ones looking for automating their services. Moreover, the system substitutes the need for physical tickets, hence reducing the use of paper. Affordable technology is leveraged to make the proposed system available to more facilities. It is an efficient, cost-effective and scalable solution that can handle a large number of vehicles with high accuracy.

The remainder of this paper is organized as follows: section II reviews previous studies on license plate detection and recognition. Section III describes the methodology used for this study, including data collection, model training, full system integration and workflow. Section IV shows the inference results, addresses limitations encountered, and discusses the overall systems performance. Section V concludes the paper and provides a future direction for the study.

II. Related Work

License plate detection plays an important role in systems such as traffic control and electronic payment in parking spaces. However, this process can be complicated due to image conditions such as lighting, weather conditions, resolution, and angles that can vary, requiring too many calculations to analyze the image, slowing the process. In the paper [2] a

License Plate Detection and Recognition (LPDR) system is implemented to detect license plates characters. First, they process the images, convert them to grayscale, adjust the resolution, and remove noise. Then, they apply a wavelet function to identify vertical edges, which helps detect the position of the license plate within the image. Using a convolutional neural network (CNN), they classify areas of the image to determine if they contain a license plate.

Once the license plate area is located, a second CNN is applied to recognize each character individually. For this, they use a pre-trained CNN model, such as Inception-v3, which identifies characters in three steps: segmentation, classification, and comparison with known letters and numbers to extract the final plate number. In the study, this system was tested on vehicles with Moroccan license plates, achieving an accuracy of 99.43% for plate localization and 98.9% for character recognition.

Generally, image-processing techniques face challenges with images taken in unconstrained scenarios such as uneven illumination and different weather conditions. One of the most common methods involves using a YOLO network for license plate detection and CRNET for character segmentation and recognition. However, a different approach is presented in [3] where the CNN VertexNet model is used for license plate detection and the Squeeze character recognition network (SCR-Net) for license plate recognition achieving an accuracy of 99% on the CCPD and AOLP datasets. This method named VSNet implements VertexNet that uses a low-resolution image to achieve high inference speed and reduce memory usage. This does not affect the recognition in SCR-Net since the license plate area is resampled from the finest input image and rectified to high resolution according to the vertices predicted by VertexNet. The proposed network architecture uses only CNNs to obtain detection and recognition results, making it suitable for real-time systems with limited resources.

Despite these advancements, traditional models are trained on closed datasets, making it difficult for them to recognize license plates from different regions. To address this limitation, the research presented in [4] proposes OneShotLP, a video-based framework for license plate detection and recognition. This model can detect the location of a license plate from the first frame and extract its characters in a zero-shot manner; this means that it does not require prior training on specified license plate types. OneShotLP consists of three main modules: Cotracker [5], which tracks the position of the license plate throughout the video and guides the segmentation process. EfficientSAM [6], is a segmentation module that accurately isolates the license plate based on the tracked positions and Monkey-Chat [7], a multi-modal large language model (MLLM), specialized in visual question-answering tasks. The model employs an adapted language prompt to convert the recognition task into an image description problem, effectively extracting the license plate characters from videos and enabling it to adapt to various license plate styles from different countries without additional training.

For an automated parking lot, it is important not only to achieve good performance in detecting and recognizing the license plate, but also to provide transparency and reliability on the model's output. This study paper [8] introduces the application of Explainable Artificial Intelligence (XAI) to improve the model's interpretability with a region of features. It uses a CNN-GRU fusion model for automatic license plate recognition, and employs SHAP, a XAI method, to interpret the model's decision by highlighting the regions of the image that influenced the output.

III. Methodology

This section describes the process implemented to develop the license plate recognition system, from data preparation and training models for detection and recognition, to the implementation of the complete system controlled through a user interface. Furthermore, before the models were implemented, a comparative analysis was conducted to select the most effective one, as detailed in this section.

A. License Plate Detection

1) Data Collection: For this study, 652 license plate images were collected from a variety of public sources, including the Roboflow Public Object Detection dataset [9], OpenALPR Benchmarks on GitHub [10], and two Kaggle datasets: the Large License Plate Dataset [11] and the Car Plate Detection Dataset [12]. In order to train a model with significant generalization skills in practical applications, these sources include license plates from a variety of locations and lighting circumstances. Figure 1 shows a sample of the license plate images collected to create the dataset.

Fig. 1: Sample license plate images.

Data augmentation techniques were used to produce synthetic variants of the original photos, representing difficult weather conditions including rain, fog, and shadow, since license plate detection models are commonly used in unpredictable outside contexts. About 20% of the original dataset was replicated for each condition, producing 130 more pictures for each condition. The purpose of the augmentation techniques was to recreate specific environmental effects. The hazy appearance of fog was created by applying a blur effect. The uneven lighting characteristic of shadows was replicated by selectively adjusting brightness and contrast. In order to create the rain effect on the images, a random noise was added. By replicating these environmental challenges, a range of cases were added to the dataset, enabling the model to detect license plates in difficult situations. In Figure 2 it is possible to see images before and after data augmentation.

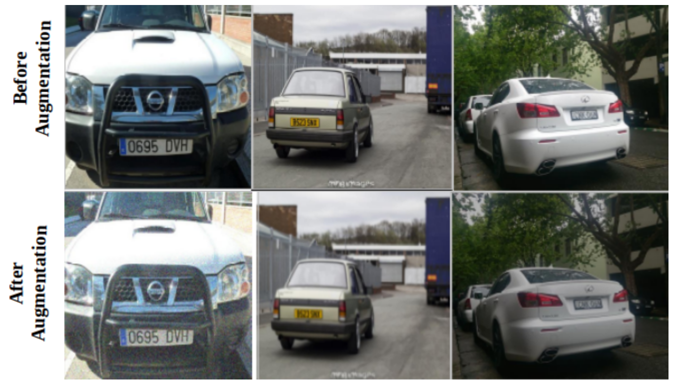


Fig. 2: Before and after image with rain, fog and shadow filter.

The final dataset was a total of 1042, which split into training, validation, and testing subsets with a 70/20/10 distribution respectively. A balanced dataset was generated by this splitting technique, which reduced overfitting and enabled reliable evaluation in a range of scenarios. This data preparation approach not only increased the dataset's size but also significantly strengthened its quality.

2) Training Environment: The models were trained on a GPU-accelerated environment running PyTorch 2.1.0. CUDA was used to accelerate training, resulting in less time for adjusting and testing on a large data set.

3) Model Training: For license plate detection, four different models were trained: YOLOv9c [13], YOLOv8n [14], YOLOv5n [15] and Faster R-CNN [16]. Here's a quick look at each model:

YOLOv9c is the most recent addition to the YOLO series of object detection models. This model consists of approximately 25 million parameters and 384 layers. The training results show the model's performance across 30 epochs. Validation accuracy increased rapidly throughout the first epochs, reaching 80% by epoch five. This model continued to grow steadily during the training. Meanwhile, the training loss decreased significantly, starting at roughly 3.0 and dropping to less than 0.5 by the last epoch. These findings demonstrate the model's good learning capabilities, giving it a solid choice for accurate detection tasks. Figure 3 shows the training loss and validation accuracy results of the YOLOv9c training.

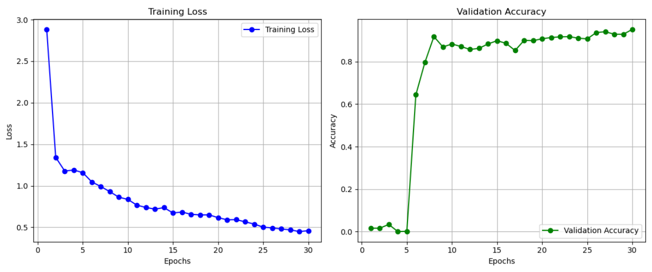


Fig. 3: Training loss and validation accuracy plot of yolov9c model.

Although YOLOv9c trained with 92.9% accuracy, a higher accuracy is required to detect license plates successfully. The YOLOv8n model is also largely used for object detection tasks, making it a suitable alternative for license plate detection in real-time applications. This model consists of 3 million parameters and 168 layers. This model also trained for 30 epochs and showed almost similar results compared to YOLOv9c, with validation accuracy of 94.9%, as shown in Figure 4.

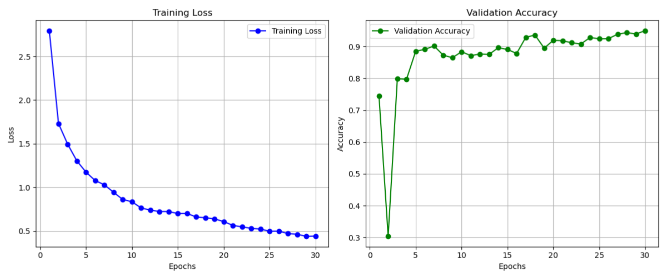


Fig. 4: Training loss and validation accuracy plot of yolov8n model.

YOLOv5n was trained to compare results after YOLOv8n demonstrated better performance than the larger YOLOv9c model. YOLOv5n is a reliable option for object detection tasks that require fast processing, because it consists of only 1.9 million parameters and an optimized architecture. As YOLOv8n performs better than YOLOv9c, YOLOv5n is also trained to check if it's the best fit. Although the training process was good, it does not outperform larger versions in terms of accuracy. Figure 5 shows the results of YOLOv5n training.

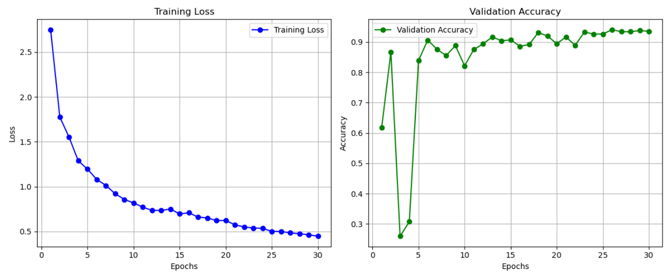


Fig. 5: Training loss and validation accuracy plot of YOLOv5n model.

For license plate detection, Faster R-CNN proved to be a solid choice, especially when accuracy is the primary concern. The model's two-stage architecture, which first generates region proposals and then refines them for final detection, allows it to precisely locate license plates in various conditions. During training, the loss plot looks promising, and the accuracy plot appears to be somewhat static, which could indicate the possibility of overfitting to the data. But overall achieved 94.1% accuracy, as observed in Figure 6. In addition, Faster R-CNN needs more computational power than models like the YOLO model, which makes the least perfect choice for this task.

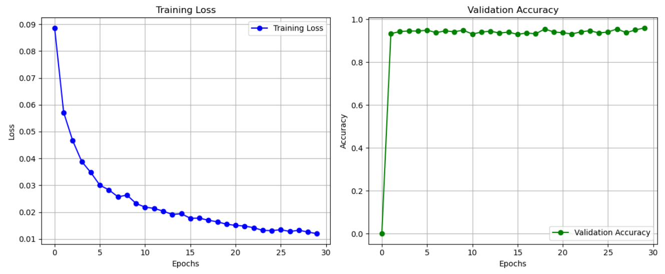


Fig. 6: Training loss and validation accuracy plot of Faster R-CNN model.

4) Model Evaluation and Comparison: When comparing the four models—YOLOv9c, YOLOv8n, YOLOv5n, and Faster R-CNN, each has its strengths and weaknesses depending on the task. YOLOv8n is the fastest and most precise of the YOLO models, delivering solid results for accuracy and mAP, but it takes longer to process compared to YOLOv5n. YOLOv5n provides a strong balance between speed and accuracy, delivering faster inference times than both YOLOv9c and YOLOv8n. While YOLOv9c excels in handling more complex scenarios, it does not match YOLOv8n in terms of speed and accuracy. Faster R-CNN, although precise, has significantly slower processing times, particularly during pre-processing and inference. Because of this, it is less suited for real-time tasks where speed is crucial, but could be more appropriate for situations where accuracy is prioritized over processing speed. The metrics scores of the four models can be seen in Table I.

TABLE I: Model performance comparison across metrics.

Model	Precision	mAP@50	mAP@50-95	Preprocessing Time (ms)	Inference Time (ms)
YOLOv9c	0.929	0.950	0.736	0.1	6.1
YOLOv8n	0.949	0.949	0.742	0.1	1.3
YOLOv5n	0.935	0.948	0.738	0.1	1.1
Faster RCNN	0.941	0.926	0.615	57.87	32.49

The results show that adding more complexity, as with YOLOv9c, is not always beneficial. Smaller models with fewer parameters can sometimes outperform larger ones, especially for simpler tasks. YOLOv8n, with its efficient design, achieved the best balance of speed and accuracy. Its lightweight architecture makes YOLOv8n particularly well-suited for real-time applications, such as live traffic monitoring or edge devices in smart parking systems.

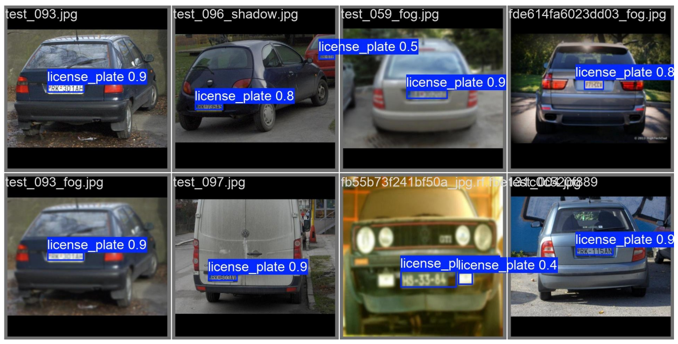


Fig. 7: Inference results with yolov8n model.

In conclusion, this study demonstrates that YOLOv8n is a clear winner for license plate detection tasks where speed, accuracy, and resource efficiency are critical. Its exceptional performance makes it a practical choice for a wide range of real-world applications.

B. License Plate Recognition

After identifying the coordinates within the image, an Optical Character Recognition (OCR) model is used to extract the characters from the image. OCR uses an optical mechanism to automatically recognize the characters [17].

For this task, two OCR models were: EasyOCR [18] and PaddleOCR [19]. These models were chosen based on their performance with character recognition tasks. The EasyOCR model is widely recognized and is often used for recognition tasks in license plate projects [20], [21], because of its robustness and ease of implementation. The PaddleOCR is a recent model that shows satisfactory results with the pre-trained model. Although it is a new model, there is a study that implements PaddleOCR for license plate recognition [22].

After thorough research on both models, a fine-tuning for the EasyOCR model was conducted in order to improve its performance. The PaddleOCR model was not fine-tuned, since their documentation recommends at least 5,000 images for the fine-tuning recognition part of the model.

1) Dataset Preparation: The dataset for fine-tuning the EasyOCR model was created using the dataset from the license plate detection task. The images were cropped according their detected coordinates and then manually labeled. The images that were not able to be identified were excluded from the dataset.

The final dataset contains 546 images for training, 191 images for validation and 97 images for testing. The annotation file for the dataset is a CSV file with two columns: the images filename and the ground truth license plate number. Figure 8 illustrates a sample of the license plate recognition dataset.

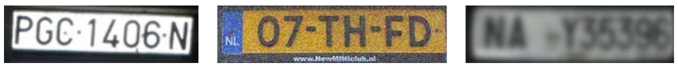


Fig. 8: Sample images from license plate recognition dataset.

For this task, the same training environment from the license plate detection task was used.

2) Training Process: The GitHub repository from EasyOCR was cloned, and the file *trainer.ipynb* was used for fine-tuning the pre-trained model. Some adjustments were made to log the metrics to tensorboard during the training process.

Moreover, some parameters in the configuration file were changed to improve the training results. Key modifications made:

- Increase of embedding size;
- Decrease number of iterations and validation interval, accordingly;
- Change of optimizer to Adam;
- Decrease of learning rate;

- Insertion of small contrast adjust for the images pre-process.

Various parameters adjustments were explored to identify a configuration that created the model with the best performance on the test dataset.

Figure 9 shows the accuracy and normalized edit distance (norm ED) metrics for the training. The fine-tuning of the EasyOCR model achieved approximately 74% accuracy, which is a satisfactory result considering the dataset size. The normalized edit distance first measures the Levenshtein distance between the predicted and the ground truth, and then normalizes the result [23].

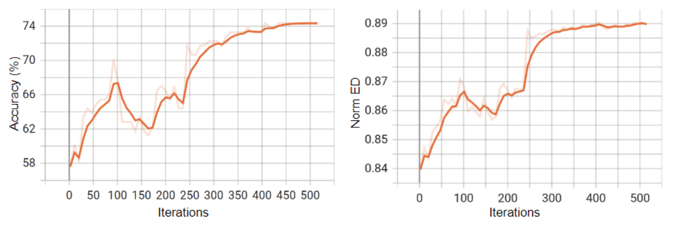


Fig. 9: Accuracy and Normalized Edit Distance training results.

The edit distance counts how many characters need to have an operation for transforming the predicted result to the ground truth. The best possible result for the edit distance is 0, which means no operation is necessary since they are the same word.

A normalized edit distance score of 1 indicates a perfect result, and as seen in the Figure 9, the fine-tuned model achieved approximately 0.89, thus showing a good result.

It was observed through Figure 10 signs of overfitting during the training. Thus, some main approaches were taken to avoid this:

- A small weight decay was inserted in the Adam optimizer to add a small penalty in the weight, for introducing a regularization technique;
- The learning rate for the optimizer was decreased;
- The batch size was increased for increasing generalization;
- Early stop training was also tested in the training process.

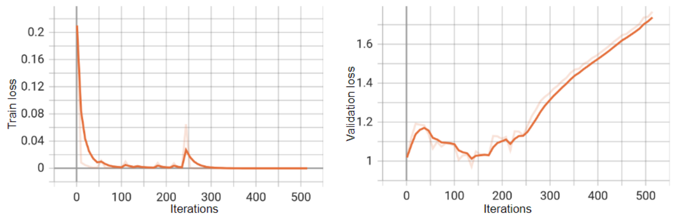


Fig. 10: Train and validation loss training results.

Although the strategies to overcome overfitting were helpful, the main challenge encountered was the dataset size. For increasing the model's performance, a bigger dataset is required.

3) Model Evaluation and Comparison: The pre-trained EasyOCR, the fine-tuned EasyOCR and the pre-trained PaddleOCR were evaluated on the test license plate recognition dataset. Table II compares the results of those three models.

TABLE II: Comparison of model's performance on the test dataset.

Model	**True/False Predictions**	**Accuracy (%)**	**Norm Ed**	**Mean Character Error Rate (%)**	**Mean Word Error Rate (%)**	**Mean Confidence Score (%)**
EasyOCR	8/94	7.84	0.50	57.35	92.16	45.04
EasyOCR Fine-Tuned	**71/31**	**69.61**	**0.88**	**19.83**	**30.39**	**85.39**
PaddleOCR	65/37	63.72	0.84	23.52	36.28	85.12

Three additional metrics were used for this comparison:

- Mean Character Error Rate: calculates the mean false predicted characters of the license plate number compared to the ground truth.
- Mean Word Error Rate: calculates the same, but considering the whole word.
- Mean Confidence Score: the model provides a confidence score on the predicted result, regarding how sure the model is on the result. When no word is recognized, no confidence score is provided.

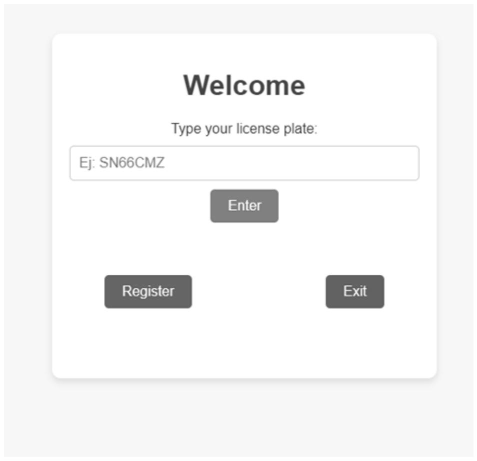
As observed, the EasyOCR fine-tuned model shows the best results. Therefore, it was implemented in the *main* branch of the project.

C. Full-Stack System Development

To integrate the models into a parking system capable of capturing images, detecting and recognizing license plates, and generating actions based on the stage of the process, a web application was developed. The backend was implemented using the Flask framework for its simplicity and flexibility, while React, one of the most popular frameworks in web development, was selected for the frontend.

Before proceeding with the implementation, the system's logic was defined, considering the typical workflow of a parking system. The process was divided into three main stages: vehicle entrance, payment, and vehicle exit. Although some parking systems utilize floor sensors to differentiate between motorcycles and cars (since each has a different fee), these functionalities were not included in this project. Instead, the entrance, exit, and payment processes were managed entirely through the user interface.

The frontend includes the user interface, which provides components for user interaction. As illustrated in Figure 11, the interface contains labels, a text field for entering the license plate number and three buttons: Enter, register and exit. The Enter button directs the user to the next screen after entering the license plate number. The Register button simulates the vehicle's entrance into the parking lot. When pressed, it captures a photo and registers the vehicle. The Exit button simulates the vehicle leaving the parking lot. It captures another photo of the license plate to verify if the vehicle is authorized to exit. The backend manages the system's logic, ensuring proper interaction between the user interface, the database, and the detection and recognition models.



Welcome

Type your license plate:

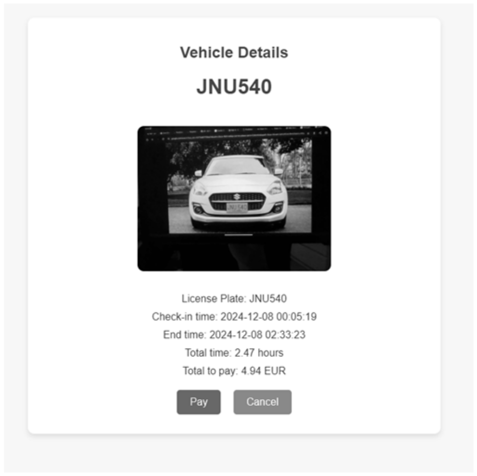
Ej: SN66CMZ

Fig. 11: Entry screen.

D. System Workflow

When the Register button is pressed, the frontend sends a request to the backend to initiate the registration process. It activates the camera to capture a photo of the vehicle. The captured image is processed using the YOLO model to extract the license plate coordinates. These coordinates are used to crop the license plate region from the image. The cropped image is then passed to the OCR model to extract the characters of the license plate. The extracted data (license plate characters) is stored in the database, along with the vehicle's photo and the entry timestamp, to calculate parking fees later. Subsequently, the system updates the vehicle's status to 'inside' and displays a welcome message to the user.

At the payment stage, the user inputs their license plate number and is directed to a payment page Figure 12. This page displays the vehicle's data, including the photo, entry timestamp, and total fee, calculated based on the duration of the vehicle inside the parking lot. Once the user confirms payment by pressing the Pay button, the system updates the vehicle's status to 'paid'.



Vehicle Details

JNU540

License Plate: JNU540

Check-in time: 2024-12-08 00:05:19

End time: 2024-12-08 02:33:23

Total time: 2.47 hours

Total to pay: 4.94 EUR

Fig. 12: Payment Screen.

During the exit process, the user presses the Exit button, which simulates the system detecting the vehicle at the parking exit. The camera captures a new photo, and the backend verifies whether the parking fee has been paid. If the status is 'paid', the system allows the user to exit and updates the status to 'outside'. Otherwise, the system redirects the user to the payment interface to complete the transaction.

The database stores key information about each vehicle, including the license plate number, the vehicle image, the entry timestamp, the payment status (inside, paid, or outside), and the exit timestamp. This structure allows the system to track the state of each vehicle and calculate parking fees efficiently.

The backend interacts with the frontend through API endpoints. These endpoints handle tasks such as receiving and sending vehicle information, retrieving and updating data, and calculating fees. The modular design of the backend facilitates the integration with the detection and recognition models, enabling efficient communication between the system components.

IV. Results

A. Inference Results

Figure 13 presents examples of inference results from our proposed system. The detection part was performed using YOLOv8n and the recognition using our fine tuned EasyOCR model.

The first three images demonstrate the system's capability to detect and recognize the license plate numbers under different adversarial conditions. The last three images illustrate some limitations of the system.

Image a) shows an ideal scenario, where the picture is clear and the characters are easy to recognize. Image b) represents a rain scenario, and image c) a fog situation in which the picture is slightly shifted. Despite these conditions, the system was able to detect and recognize the plate numbers.

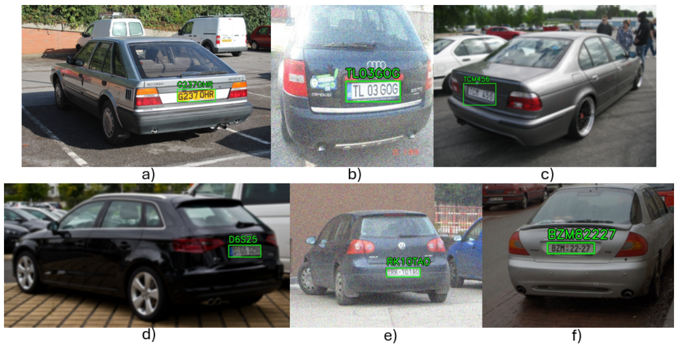


Fig. 13: Inference results examples of our system.

In image d), the system was able to identify some of the characters in the license plate, but not all of them. In addition to the image being blurry, the detected bounding boxes sometimes crop a small part of the license plate, making it difficult for the recognition model to accurately extract all the characters, especially in shifted images.

Image e) represents a rain condition where the character '1' was mistaken by 'T'. Some similar issues were observed with the characters '4' and 'L" or 'N' and 'H'. The models were trained in a diverse dataset, thus learning different representations of numbers and letters. Although this diversity improves the model's robustness, it can also lead to some misinterpretations.

In image f), the system identified the symbols between the characters 'M' and '2' as a letter '8'. As explained above, the dataset contains license plates from different countries. Since the balance between the countries was not taken into account, the recognition model did not learn that those symbols are not relevant characters.

Overall, the system showed excellent performance for rain and shadow images, but revealed some problems in identifying the license plate under foggy situations. Despite attempts to enhance the images with sharpening, the results did not improve.

B. Integrated System

After the integration of the system's components, the system workflow was tested as described in Section III-D. This process was conducted locally, with both the backend and frontend executed on a laptop. The laptop's built-in camera, with a resolution of 640x480 pixels, was used to capture images. Additionally, a tablet was used to display an image of a car, as shown in Figure 14, which illustrates a car entering the parking lot. Despite lighting conditions, the system was able to successfully detect and identify the license plate characters.

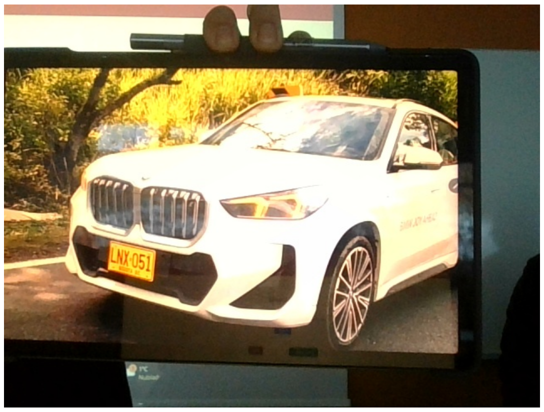


Fig. 14: Example of an image captured with the laptop's built-in camera.

The system generates two folders: the first folder, Figure 15, contains the picture taken by the camera, the image with the detected license plate, and a cropped license plate image. The second folder, Figure 16, contains the pictures of the vehicles registered in the system, with each photo named according to the corresponding license plate number.

Overall, the testing demonstrated good performance in efficiently capturing, detecting, and processing license plates, ensuring smooth operation within the defined workflow. However, in a few cases, due to lighting conditions, the images

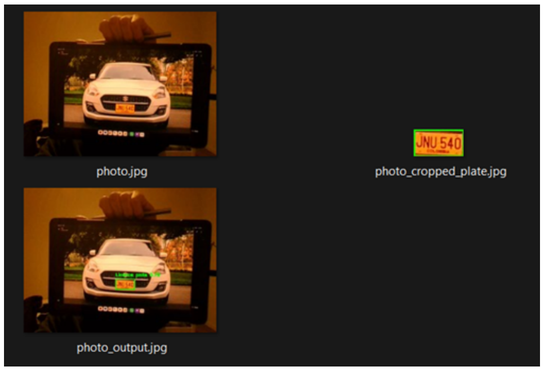


Fig. 15: Folder containing captured image and license plate detection.

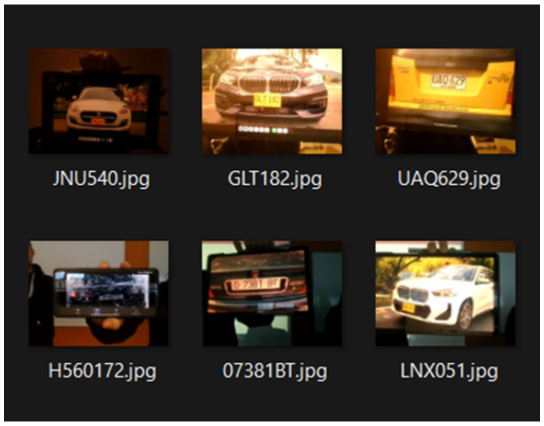


Fig. 16: Folder containing registered vehicle images with their respective license plate numbers.

captured were not visible, and as a result, the system was unable to detect the correct license plate number. However, in a real scenario, the cameras used in parking lots are placed under controlled conditions to avoid lighting issues.

V. Conclusion

This study created a license plate detection and recognition system, encompassing data collection, model training, and full system integration. The models were trained under adversarial conditions, as observed in the results, the data augmentation process did in fact improve the model's robustness and ability to identify the characters in different weather, lighting and angle scenarios.

Besides the effectiveness of the system, some challenges were faced, mainly the different license plate formats, varying weather and lighting conditions, and dataset creation. Future work could be done to address these problems and enhance the models performance. In the dataset creation, selecting a target country could improve the results since the output pattern would be known. Moreover, increasing the dataset could lead to better accuracy and avoid overfitting during training [8].

In such scenarios as illustrated in Figure 13 d), if the user enters its license plate and it is not detected, the system could integrate a feature to search for similar license plates, considering also the time and date of the entry. This would allow the user to verify if one of the suggestions is actually their vehicle. Furthermore, an explainable artificial intelligence method could be used in order to better understand and interpret the results, especially for the users.

Another approach that could be explored is replacing the recognition model with a multimodal language model. This method, as proposed in [4], leverages a multimodal large language model to process license plate images using text prompts. This approach eliminates the need for training data, as the multimodal model can directly extract the characters from the license plate image. Implementing this technique could potentially simplify the system and improve its adaptability to diverse datasets.

References

[1] M. Sarangi, S. K. Das, and K. S. Babu, "Smart parking system: survey on sensors, technologies and applications," in *2019 1st International Conference on Advances in Information Technology (ICAIT)*. IEEE, 2019, pp. 250–255.

[2] I. Slimani, A. Zaarane, W. Al Okaishi, I. Atouf, and A. Hamdoun, "An automated license plate detection and recognition system based on wavelet decomposition and cnn," *Array*, vol. 8, p. 100040, 2020.

[3] Y. Wang, Z.-P. Bian, Y. Zhou, and L.-P. Chau, "Rethinking and designing a high-performing automatic license plate recognition approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 8868–8880, 2021.

[4] H. Ding, Q. Wang, J. Gao, and Q. Li, "A training-free framework for video license plate tracking and recognition with only one-shot," *arXiv preprint arXiv:2408.05729*, 2024.

[5] N. Karaev, I. Rocco, B. Graham, N. Neverova, A. Vedaldi, and C. Rupprecht, "Cotracker: It is better to track together," *arXiv preprint arXiv:2307.07635*, 2023.

[6] Y. Xiong, B. Varadarajan, L. Wu, X. Xiang, F. Xiao, C. Zhu, X. Dai, D. Wang, F. Sun, F. Iandola *et al.*, "Efficientsam: Leveraged masked image pretraining for efficient segment anything," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 16 111–16 121.

[7] Z. Li, B. Yang, Q. Liu, Z. Ma, S. Zhang, J. Yang, Y. Sun, Y. Liu, and X. Bai, "Monkey: Image resolution and text label are important things for large multi-modal models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 26 763–26 773.

[8] P. Das, S. Mitra, S. Chakraborty, M. H. K. Mehedi, M. Y. M. Adib, and A. A. Rasel, "Cnn-gru based fusion architecture for bengali license plate recognition with explainable ai," in *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. IEEE, 2023, pp. 1–6.

[9] "License plates dataset," https://public.roboflow.com/object-detection/license-plates-us-eu, accessed in 2025-01-20.

[10] "Openalpr benchmark dataset," https://codeload.github.com/openalpr/benchmarks/zip/refs/heads/master, accessed in 2025-01-20.

[11] "Large license plate detection dataset," https://www.kaggle.com/datasets/fareselmenshawii/large-license-plate-dataset/data, accessed in 2025-01-20.

[12] "Car license plate detection," https://www.kaggle.com/datasets/andrewmvd/car-plate-detection, accessed in 2025-01-20.

[13] C.-Y. Wang, I.-H. Yeh, and H.-Y. Mark Liao, "Yolov9: Learning what you want to learn using programmable gradient information," in *European conference on computer vision*. Springer, 2025, pp. 1–21.

[14] "Ultralytics yolov8," https://docs.ultralytics.com/models/yolov8/, accessed in 2025-01-20.

[15] "Ultralytics yolov5," https://docs.ultralytics.com/models/yolov5/, accessed in 2025-01-20.

[16] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 6, pp. 1137–1149, 2016.

[17] R. Mithe, S. Indalkar, and N. Divekar, "Optical character recognition," *International journal of recent technology and engineering (IJRTE)*, vol. 2, no. 1, pp. 72–75, 2013.

[18] "Easyocr github repository," https://github.com/JaidedAI/EasyOCR, accessed in 2025-01-15.

[19] "Paddleocr documentation," https://paddlepaddle.github.io/PaddleOCR/latest/en/index.html, accessed in 2025-01-15.

[20] R. Laroca, L. A. Zanlorensi, G. R. Gonçalves, E. Todt, W. R. Schwartz, and D. Menotti, "An efficient and layout-independent automatic license plate recognition system based on the yolo detector," *IET Intelligent Transport Systems*, vol. 15, no. 4, pp. 483–503, 2021.

[21] D. Vedhaviyassh, R. Sudhan, G. Saranya, M. Safa, and D. Arun, "Comparative analysis of easyocr and tesseractocr for automatic license plate recognition using deep learning algorithm," in *2022 6th International Conference on Electronics, Communication and Aerospace Technology*. IEEE, 2022, pp. 966–971.

[22] O. Sarkar, S. Sinha, A. K. Jena, A. K. Parida, N. Parida, and R. K. Parida, "Automatic number plate character recognition using paddle-ocr," in *2024 International Conference on Innovations and Challenges in Emerging Technologies (ICICET)*. IEEE, 2024, pp. 1–7.

[23] "Tasks - icdar 2019 robust reading challenge on reading chinese text on signboard," https://rrc.cvc.uab.es/?ch=12&com=tasks, accessed in 2025-01-15.