Journal of Engineering Science and Technology
Vol. 19, No. 5 (2024) 1672 - 1692

ENHANCING CAMPUS SECURITY AND VEHICLE MANAGEMENT WITH REAL-TIME MOBILE LICENSE PLATE READER APP UTILIZING A LIGHTWEIGHT INTEGRATION MODEL

MUHAMMAD HAZIQ BIN KAMAROZAMAN [1],
A. R. SYAFEEZA[1,*], Y.C. WONG[1], NORIHAN ABDUL HAMID[1],
WIRA HIDAYAT MOHD SAAD[1], AIRUZ SAZURA ABDUL SAMAD[2]

[1]Faculty of Electronics and Computer Technology and Engineering (FTKEK), Machine Learning and Signal Processing (MLSP), Centre for Telecommunication Research & Innovation (CeTRI), Universiti Teknikal Malaysia Melaka, 76100, Durian Tunggal, Melaka, Malaysia
[2]Ges Venture Manufacturing Sdn. Bhd, Johor Bahru, Johor
*Corresponding Author: syafeeza@utem.edu.my

Abstract

The increasing number of vehicles owned by campus residents, combined with a limited number of staff parking lots, poses challenges for security personnel in distinguishing between staff, student vehicles, and visitors. Additionally, the presence of untracked external visitors and the potential manipulation of vehicle registrations by residents pose safety risks. Implementing a License Plate Detection and Recognition (LPDR) mobile app could ease the burden on security patrols. However, traditional LPDR systems face real-world limitations, including various backgrounds, illumination, weather, and distances. Therefore, opting for a deep learning-based LPDR approach is the way forward. Nevertheless, implementing deep learning on resource-constrained mobile devices demands significant storage and computational power. This may lead to user hesitation when it comes to downloading the app onto their mobile devices. This paper introduces an automated Android mobile app for real-time license plate reading, integrating a lightweight YOLOv8n for license plate detection and ML Kit Optical Character Recognition (OCR) for text recognition. The inference integration model is performed on the mobile device to streamline security tasks, aiming to assist security personnel in identifying any wrongdoing by campus residents and visitors. The lightweight model addresses mobile device resource limitations by implementing an on-device machine learning model and storing vehicle ownership information on a cloud server without compromising accuracy. The plate detection accuracy is approximately 97.5%, the character recognition accuracy is around 91.2% which is on par with other LPDR existing works. The study results in a precise end-to-end mobile app for license plate reader, benefiting patrol units with low error rates and real-time capabilities. The mobility feature enables swift security responses, ultimately enhancing overall campus safety.

Keywords: Android, Campus safety, End-to-end mobile app, Object detection, Real-time detection, Real-world scenarios.

1. Introduction

Vehicle parking management on university campuses faces challenges due to increasing vehicle numbers, impacting both residents and visitors, compounded by limited staff parking spaces. University staff encounters issues when students or visitors occupy their designated parking spots, forcing them to seek alternative locations. Parking lots, particularly those lacking CCTV surveillances, require additional security measures. At the Technical University of Malaysia Malacca, the current system relies on manual processes, allowing only authorized vehicles with entrance stickers. The university also serves as a thoroughfare for external residents traveling between the main entrance and the exit gate leading to other regions of Malacca.

Despite manual security at the entrances to student hostels, this situation raises security concerns for residents in other areas of the campus. Untracked external visitors, staff parking space occupation issues, and potential manipulation of vehicle registrations by residents pose safety risks. Manual vehicle monitoring and management are considered the standard security approach for most campuses in Malaysia. Therefore, a license plate reader can offer a solution to this issue.

The challenges associated with license plates include variations in typefaces, sizes, colours, and placement on vehicles. Recognition by the License Plate Detection and Recognition (LPDR) system faces additional hurdles from environmental factors like poor lighting, dirt, adverse weather conditions, and obstructed plates [1]. Detecting license plates is particularly challenging when they are dirty. Recognition rates also fluctuate based on surrounding conditions such as lighting and the plate's background. As LPDR is a complex pattern recognition task, AI-based solutions, particularly machine learning and deep learning, are preferred approaches.

Despite the availability of various machine learning solutions, deep learning has demonstrated superior performance when compared to traditional machine learning methods. This is attributed to the capability of deep learning techniques in handling challenging real-world scenarios and cases better than traditional techniques as a unified end-to-end solution [2]. However, implementing deep learning on limited-resource mobile devices presents an additional challenge.

While these intelligent mobile vision applications showcase impressive capabilities, they demand significant storage, computational power, and incur high energy and network bandwidth consumption. This may lead to user hesitation when it comes to downloading them onto their mobile devices [3]. Deploying deep learning on mobile devices offers unique advantages in data privacy, communication overhead, and system cost when compared to traditional deep learning solutions using cloud servers.

Cloud computing models are extensively utilized in intelligent monitoring systems for processing diverse types of video and image data. Nevertheless, they face bottlenecks in actual LPDR, primarily characterized by (1) challenges in achieving optimal performance for real-time systems in both license plate detection and recognition; (2) the transmission of video and image data potentially leading to a significant increase in energy consumption [4].

Wang et al. [3] suggested addressing this issue by reducing the resource requirements for running deep learning models and optimizing mobile device hardware for Mobile Deep Learning Applications (MDLAs). Popular solutions involve compressing deep learning models, which, although potentially impacting

accuracy, significantly reduces the demand for computation and storage resources. The reason is due to common methodologies, such as Convolutional Neural Networks (CNNs) in mobile vision tasks, entail high time and space complexity, leading to resource bottlenecks. Utilizing MDLA's for the purpose of license plate reader adds on complexity to the design.

The primary goal of this research is to develop a License Plate Detection and Recognition (LPDR) mobile app by integrating a lightweight YOLOv8n detector and ML Kit Optical Character Recognition (OCR). The app aims to assist security personnel in their patrol duties. Leveraging the capabilities of mobile devices, such as smartphones and laptops, ensures the feasibility of this objective. Smartphones serve as key computing and communication devices equipped with embedded sensors, powerful CPUs, and high-resolution cameras, meeting the necessary criteria for an efficient LPDR system.

In cloud-based systems, data transfer for image and video poses a bottleneck. On-device machine learning processes input data directly on the device, rather than transmitting it to a server for processing. Examples of the challenging condition of license plate as shown in Fig. 1.

Fig. 1. Examples of the challenging condition of license plate.

Several studies have proved the feasibility to integrate a license plate recognition algorithm into the mobile platform [4, 5]. Therefore, major contributions of this study are to:

- To develop a lightweight deep learning model capable of accurately detecting license plates in various conditions without compromising accuracy.
- To evaluate the license plate detection model performances in terms of accuracy and execution time of various lightweight object detection models, mean Average Precision (mAP), inference time, and total processing time.
- To develop a mobile integration application of LPDR in the form of mobile app that is based on the best object detection model (YOLOv8n) and ML Kit OCR for the character recognition purpose and with the aid of cloud database.

The remainder of this article is organized as follows: Section 2 summarizes the background study and related works. Section 3 discusses the methodology,

providing a formal description of the overall system. Section 4 presents the results and discussion, and the final section concludes the article.

2. Background and Related Works

License plate detection and recognition have been prominent subjects of research in computer vision, with numerous approaches proposed by researchers in this field. Relevant work in the context of campus security is detailed in Dias et al. [6], where Faster-RCNN was employed for license plate localization, and Tesseract OCR was utilized for license plate recognition. The reported minimum loss achieved was 0.011. However, for real-time mobile applications, Faster-RCNN is considered unsuitable due to its large network size.

The implementation involved YOLO, PaddleOCR, and Tesseract OCR [4]. Unfortunately, the specific type of YOLO model used was not specified. It is noteworthy that both of these works were designed for web-based applications. Notably, the only work focusing on License Plate Detection and Recognition (LPDR) for a mobile-based application is presented [5]. The authors developed CampusSense to assist the security department in ensuring the safety and satisfaction of students, faculty, and staff members during parking at the university campus. However, the paper only presented ideas of developing a mobile app but did not delve into the discussion of any specific network model employed for LPDR in their implementation.

Table 1 displays a comparison of various deep learning approaches for License Plate Detection and Recognition (LPDR) in simulation form across different applications. The results obtained from these studies indicate that YOLO-based models and Optical Character Recognition (OCR) modules, such as Tesseract OCR and Easy OCR, are popular approaches.

Table 1. Comparison of the deep learning method in LPDR technique.

Ref.	Methods	Comments
[7]	Two-stage YOLOv2	LP detection for clear weather 99% but 74% for night scene.
[8]	YOLOv4 and Tesseract OCR	Limited amount of training dataset (1000 images). Achieved 94.6% accuracy on the detection rate but recognition rate was not reported.
[9]	YOLOv6	The F1-score is 0.95. YOLOv6 is more suitable for industrial applications where speed is more important than accuracy.
[10]	SSD-MobileNetv1 and Easy OCR	SSD-based detector works efficiently with an accuracy of 94.87 %. Easy OCR operates with an accuracy of 90%.
[11]	SSD-MobileNetv1 and k-Nearest Neighbour OCR	The accuracy is 0.91 for small motorcycle dataset (1524 images to train and validate the SSD model)
[12]	Tiny-YOLOv2	Performed on Taiwan's license plate dataset. The result obtained is 99.62% mAP and 9 ms inference time.
[13]	RPNet (Deep CNN)	Large China License plate dataset, CCPD (250k unique car images) with 95.5% precision and 61 fps.
[14]	Fast-YOLO	They introduced Brazilian license plates (FPR-ALP dataset). The system surpasses commercial solutions like Sighthound and OpenALPR by 93.53% accuracy and 47 fps.
[15]	Deep CNN	The result obtained is 0.993377 accuracy for 302 test images.
[16]	Deep CNN	LPDR was applied on country-specific plates, such as American or European, Chinese, Indian and Korean license plate. They achieved 99% detection and 93% recognition accuracy.

Table 2 compiles articles that implemented LPDR using edge servers/gateways [4] and edge devices [5]. LPDR was implemented on edge servers to address the high latency and energy consumption associated with cloud servers [4]. However, edge gateways face limitations in terms of memory and computation capabilities.

On the other hand, undertook LPDR using mobile edge computing (MEC) chips instead of large Graphics Processing Unit (GPU) servers [5]. This approach poses limitations in the chip's small computing capacity. Nevertheless, leveraging a license plate recognition module on mobile edge computing chips offers advantages in terms of low latency, power efficiency, mobility, reduced network dependency, privacy preservation, cost efficiency, scalability for edge devices, quick deployment, and enhanced security compared to relying on large GPU servers. Therefore, both approaches opt for a lightweight deep learning model to overcome their respective limitations.

The approaches employed in Table 2 differ from those in this article, where LPDR is implemented as a mobile app on mobile devices. Kim et al. [15] made a comparison between YOLOv5 and YOLOv8 showed that YOLOv8 exhibited slightly better accuracy and reduced training time. In line with the research objective of developing a lightweight model suitable for a mobile-based application, YOLOv8n was selected. Additionally, YOLOv8n has not been previously applied in any LPDR design.

Table 2. Comparison of the LPDR system using deep learning method for mobile edge computing application.

Ref.	Methods	Comments
[4]	YOLOv7	OpenVINO was selected as the edge gateway platform for model inference. The result obtained is 95.6% mAP and 187.6 fps.
[5]	Pruned VGG-16	Proposed an end-to-end Chinese LPDR system on mobile edge computing chip with 63 fps and 91.5% precision.

3. Methods

To successfully conclude this project, which involves the development of an Android mobile application for a license plate reader system utilizing deep learning techniques, it is crucial to address several essential steps. The accompanying flowchart in Fig. 2 illustrates the research process. Table 3 provides a list of hardware used, while Table 4 presents the mobile phones utilized in this research. From Table 4, it is evident that the mobile phone has limited RAM capacity.

Table 3. List of hardware used in the project.

No.	Hardware	Version
1	Laptop	Asus VivoBook with NVIDIA Integrated Graphic Card
2	Virtual Graphic Card on Google Colab (Mainly to train a model)	NVIDIA T4 Tensor Core GPU
3	Android Smartphone	• Samsung Galaxy J7 Pro • Samsung Galaxy Note 5

Table 4. List of mobile phone used in the project.

Smartphone	OS	Chipset	CPU	GPU	RAM
Samsung Galaxy J7 Pro	Android 9.0 (Pie)	Exynos 7870 Octa (14 nm)	Octa-core 1.6 GHz Cortex-A53	Mali-T830 MP1	3GB
Samsung Galaxy Note 5	Android 7.0 (Nougat)	Exynos 7420 Octa (14 nm)	Octa-core (4x2.1 GHz Cortex-A57 & 4x1.5 GHz Cortex-A53)	Mali-T760MP8	4GB

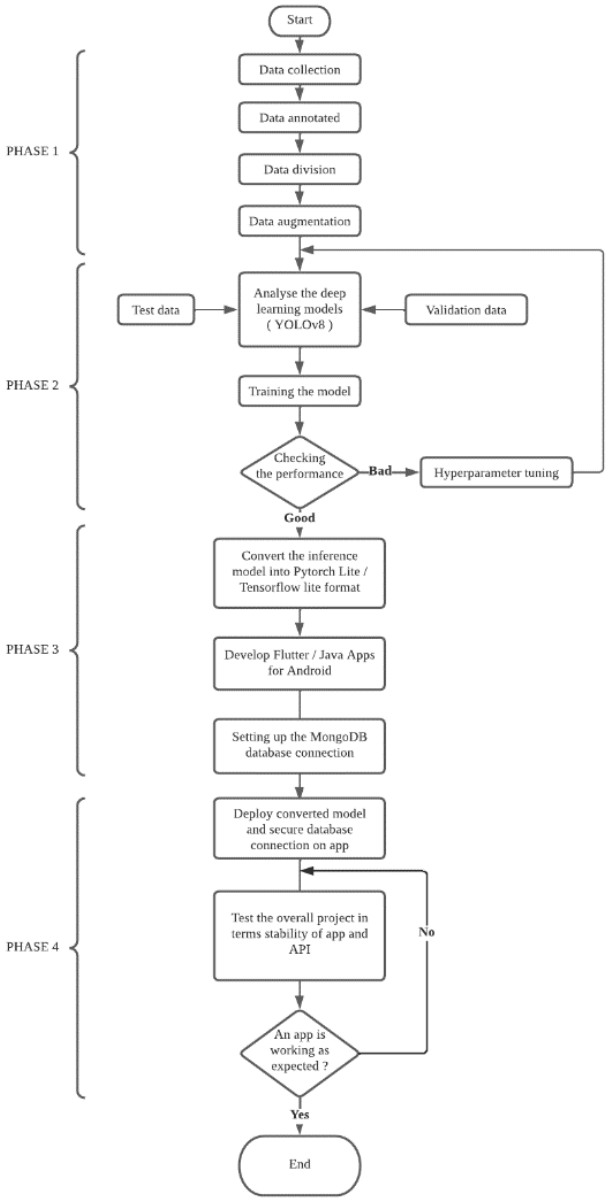

Fig. 2. Flowchart of the project.

3.1. Data preparation-PHASE 1

The data collection process involved capturing real-world urban traffic images in Malaysia to obtain approximately 3,000 Malaysian license plates from cars and motorcycles around the campus. The images were taken at distances ranging from 1.0 to 1.5 meters, with a resolution of 640×640 pixels, in various locations around the campus's parking lot. The collection incorporated diverse distances, sizes of cars and motorcycles, lighting conditions (including night scenes), tilt angles, and weather conditions (raining and clear weather). The sample distribution among cars, trucks, and motorcycles was maintained at a ratio of 90:5:5, respectively.

An additional dataset comprising around 2,200 publicly available images was utilized for training, featuring various images of cars and trucks. The primary objective of the system is the recognition of characters on Malaysian license plates, but it can be extended to other countries' license plates if samples are available.

For the initial phase of the work, the tool Roboflow was employed for tasks such as data augmentation, annotation, labelling, and division. Roboflow was selected for its capability to empower developers in building computer vision applications, covering data preparation, model training, deployment, and active learning. Users can upload custom datasets, draw annotations, adjust image orientations, resize images, modify image contrast, and perform data augmentation using Roboflow. The tool's provided features facilitate developers in preparing their data samples. In standard practice, data annotation involves manually labelling objects of interest in the collected samples, a crucial step for training machine learning models in computer vision tasks.

Once annotated, the data was stored alongside the original images. The samples were then divided into train, validation, and test sets using an 80:10:10 ratio, equivalent to 4200:519:521 samples, respectively.

3.2. Model training and evaluation-PHASE 2

This phase is crucial for the development of machine learning models as it involves training and evaluating a model using labelled data to assess its performance and generalization capabilities. Hyperparameter tuning becomes necessary to optimize the model's performance, speed, and accuracy.

Batch size and epoch are crucial hyperparameters in the model training process. The batch size determines the number of samples processed before updating model parameters. Optimizing batch size involves trying different sizes and assessing performance on the validation set. The best batch size yields the highest accuracy. Epochs define how many times the learning algorithm works through the training dataset. Overfitting can occur with too many epochs, while too few can result in an underfit model. Early stopping halts training when validation performance plateaus.

The learning rate is crucial in training deep learning models, including object detection, as it controls learning speed. It determines how quickly the model learns from data. A higher rate enables faster learning but may overshoot or get stuck. Conversely, a lower rate provides more precise updates, albeit with longer training.

Optimizers vary in type and characteristics. Options like SGD, Adam, RMSProp have unique rules and adaptability to models and datasets. Some include momentum, adaptive learning rates, or weight decay to improve convergence.

Neural networks use gradient-based optimization, adjusting parameters based on loss function gradients. The choice of optimizer significantly impacts convergence efficiency. Factors like learning rate, optimizers, loss functions, and dataset composition affect training. Validation on unseen data assesses performance. Once satisfactory, the model can be adapted for mobile use. Table 5 summarizes the hyperparameters used in this research.

In this phase, four lightweights YOLO models such as YOLOv8n, YOLOv7-tiny, YOLOv6n, and YOLOv5n were trained and evaluated using the same hyperparameters and training settings.

Table 5. List of tested hyperparameter values.

Hyperparameters	Selections
lr0	initial learning rate (0.1, 0.01, 0.001)
lrf	final learning rate (lr0 * lrf)
batch	number of images per batch (-1 for AutoBatch)
imgsz	size of input images as integer or w,h
optimizer	optimizer to use, choices=[SGD, Adam, Adamax, AdamW, NAdam, RAdam, RMSProp, auto]
momentum	SGD momentum/Adam beta1
box	box loss gain

3.3. Model conversion and app development-PHASE 3

This phase involves selecting the optimal lightweight YOLO model in Google Collaboratory and subsequently deploying it on a mobile phone. The model is converted into a PyTorch Lite model, tailored for Android App Development.

PyTorch Lite is selected for its lightweight design, tailored for deployment on resource-constrained devices like mobile phones, edge devices, and embedded systems. It emphasizes minimal memory usage and efficient execution, ideal for devices with limited processing power, memory, and storage. Through optimization techniques like model quantization, pruning, and compression, PyTorch Lite achieves efficiency. It enables running deep learning models directly on the device, eliminating the need for continuous internet access or reliance on cloud-based inference. This supports real-time and privacy-preserving applications that retain data on the device. Leveraging PyTorch's ecosystem and optimization techniques, PyTorch Lite enables real-time AI while respecting the constraints of devices with limited resources [17].

Android Studio is used to develop an application for stable and swift processing from inputting captured images to obtaining predicted model results. Extensive experiments are planned to validate system performance in terms of stability and consistency. The Android app development process involves leveraging UI widgets and libraries to craft visually appealing and efficient mobile apps. Features such as plate detection, text recognition, and database access are integrated into the app. MongoDB serves as the database for storing and managing vehicle-related data, connected via a connection string link.

The app development process utilized Flutter SDK, iteratively designing the user interface (UI) and integrating various features for improved user experience. Special attention was given to crafting a user-friendly UI with dynamic animations

using Lottie for captivating visual effects. Figure 3 depicts the LPDR system's welcome page UI.

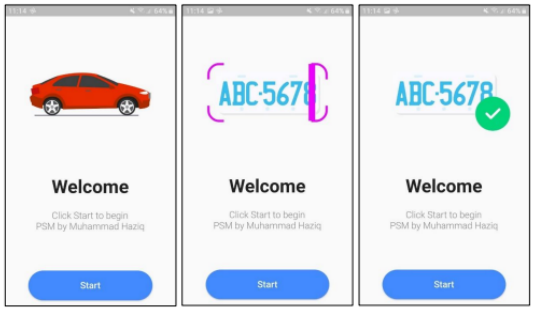

Fig. 3. Welcome page Ui designed by using flutter.

3.3.1. The app main menu

Upon opening the app, users encounter an informative interface. A brief introduction outlines the app's purpose and key features, serving as a guide for users. The first button triggers the model using gallery images, the second activates the live camera feed, and the third provides access to the app's database functionality. Figure 4 displays the main menu UI designed with Flutter.

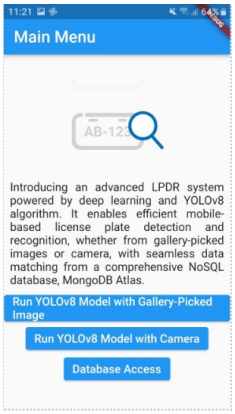

Fig. 4. Main Manu UI designed using flutter.

3.3.2. Model execution by gallery-picked image and camera

The first button, labelled "Run Plate Detection," focuses on detecting rectangular plate shapes within gallery images. Upon selection, the app's deep learning model identifies and outlines regions containing license plates using advanced computer vision algorithms. The second button enhances functionality by incorporating text recognition capability. In addition to detecting plate bounding boxes, this button extracts alphanumeric characters from the identified license plate regions, enabling users to read and interpret the text. The third button introduces further functionality. Upon text recognition, it queries the app's MongoDB database, retrieving relevant

information associated with the license plates. Figure 5 displays a screenshot of the camera feed feature.

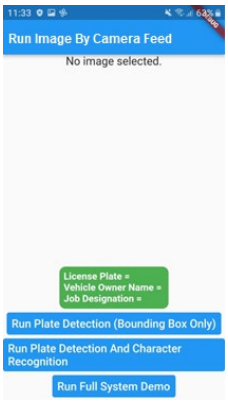

Fig. 5. Screen capture of the executed model by camera feed.

3.3.3. Database access

The database menu UI seamlessly integrates with MongoDB, providing efficient data management. It includes sections for specific actions: display, update, delete, query, and insert. The display section presents structured database contents for easy navigation. Users can browse records and gain insights. The update section enables modification of existing records. The delete section allows removal of specific records or entire datasets. The query section facilitates advanced searches based on specific criteria. The insert section enables adding new records directly into the database. Figure 6 illustrates the database menu.

Fig. 6. Database menu.

3.4. Integration and testing-PHASE 4

This section outlines the steps to integrate a license plate recognition system. It starts with importing required libraries and integrating the model into the Android application. The app handles plate detection, image capture, and transmission to the OCR service (ML Kit OCR) for text recognition.

Google's OCR service operates independently to extract text from license plates. ML Kit OCR was chosen for its native support in Android development and seamless integration into Android Studio. In Flutter, ML Kit OCR enables robust text recognition from images using OCR techniques as part of ML Kit. Easy OCR, previously used, did not perform well on the research phone, likely due to limited device specifications.

The next step is to establish a connection between the client and the MongoDB database. The OCR service forwards the extracted text to the database. The database uses the text to retrieve relevant car information and sends it back to the originating smartphone application.

The stability and performance of the object detection and OCR system are evaluated on the test set through metrics such as accuracy, recall, precision, mean Average Precision (mAP), inference time, and total processing time. The performance of the database matching process is also assessed to ensure the retrieval of correct records within a specific timeframe. Average Precision (AP) and mean Average Precision (mAP) are commonly used metrics for evaluating object detection models, which face challenges such as occlusion, scale variations, and object class imbalance, necessitating improvements in accuracy. Addressing speed challenges requires efficient algorithms, hardware utilization, and optimization techniques to achieve real-time performance.

The overall workflow of the developed application system is illustrated in Fig. 7. It begins with loading the image into the detection model, followed by the character detection process and verification, and finally, retrieval of ownership and vehicle details. Once the model and app development are complete, it can integrate the desired model into the application by importing important package manager libraries provided in Flutter Pub.dev. Then, it's necessary to import ML Kit OCR into the Android App Development to recognize the text of the detected bounding box from the previous phase.

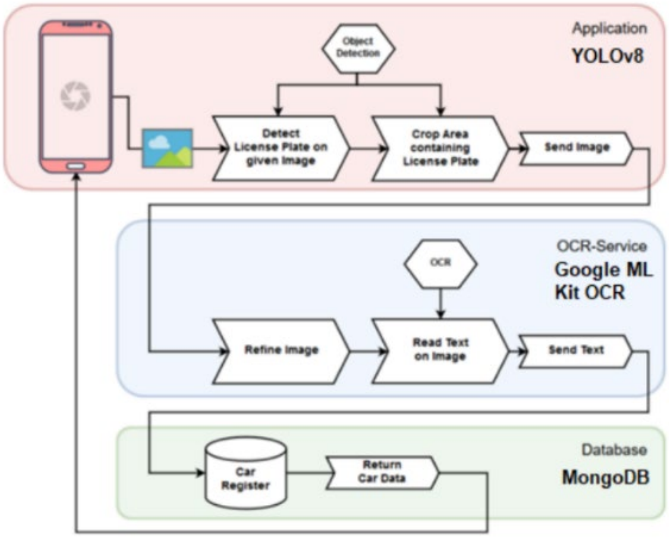

Fig. 7. Overall illustration of developed application system.

4. Results and Discussion

4.1. Comparative analysis of various YOLO models

The YOLOv8n, YOLOv7-tiny, YOLOv6n, and YOLOv5n models underwent training and subsequent comparison. The evaluation utilized consistent training and test sets across all models, assessing performance through recall, precision, and mAP metrics. These YOLO models are lightweight with minimal parameters, ensuring equitable comparison, particularly for mobile-based deep learning implementations. Training employed identical hyperparameters and settings, including a learning rate of 0.01, batch size of 16, epoch of 60, optimizer, and training schedule. In case of unsatisfactory performance, models underwent hyperparameter tuning. Notably, these small models were chosen for mobile compatibility and efficiency. Comparative test results are tabulated below.

According to Table 6, YOLOv8n and YOLOv5n stood out as the top performers. Employing an equal number of epochs ensures uniform training time across all models, eliminating bias from varying training durations. This methodology enables a focused comparison of the models' performance under consistent training conditions.

The YOLOv7-tiny model underperformed, showing poorer performance due to longer training times and lower mAP compared to other algorithms. Despite having a better mAP, the YOLOv6 model exhibits lower recall than others. Next, the two best models will undergo fine-tuning by adjusting hyperparameters to evaluate potential improvements over previous results. Key parameters for sensitivity include learning rate (lr), optimizer (SGD, Adam), and batch size (16, 32, 64).

Table 6. The performance results of the various YOLO models.

Algorithm	Recall %	Precision %	mAP_0.5 %	mAP_0.5:0.95 %	Training time (hours)
YOLOv8n	0.96404	0.94358	0.967	0.643	2.441
YOLOv7-tiny	0.7784	0.7769	0.804	0.351	3.288
YOLOv6n	0.688	0.958	0.95835	0.59312	2.063
YOLOv5n	0.95159	0.95298	0.971	0.635	1.8

4.2. Learning rate (lr) and optimizer tuning

The training process applied patience=10 and epoch=100 to prevent overfitting. If there's no improvement in results after 10 epochs, the training process stops. Learning rate parameters were adjusted alongside optimizers. Tables 7 and 8 show the performance of YOLOv8n and YOLOv5n with different optimizers and learning rates.

The results indicate SGD experiments achieved higher mAP values compared to Adam in all cases. Generally, higher mAP values indicate better performance. SGD was more effective for this object detection task. Lower learning rate values generally led to higher mAP values, indicating a smaller learning rate helped the model converge to a better solution.

Table 7. The performance of YOLOv8n with various optimizer and learning rate.

Algorithm	Optimizer	Learning Rate	Epoch	mAP_0.5	mAP_0.5:0.95
YOLOv8n	SGD	0.1	47	0.955	0.585
		0.01	52	0.967	0.646
		0.001	24	0.971	0.63
	Adam	0.1	27	0.738	0.366
		0.01	29	0.94	0.544
		0.001	25	0.969	0.632

Table 8. The performance of YOLOv5n with various optimizer and learning rate.

Algorithm	Optimizer	Learning Rate	Epoch	mAP_0.5	mAP_0.5:0.95
YOLOv5n	SGD	0.1	45	0.962	0.578
		0.01	38	0.965	0.619
		0.001	**81**	**0.969**	**0.621**
	Adam	0.1	61	0.721	0.295
		0.01	99	0.957	0.594
		0.001	61	0.962	0.623

4.3. Batch size tuning

From the previous result, the best hyperparameter tuning was selected to be tested on the various batch size values. Tables 9 and 10 show the performance of YOLOv8n and YOLOv5n, respectively.

YOLOv8n's accuracy improved from 0.971 to 0.975 after tuning the batch size. YOLOv5n also showed acceptable mAP. The analysis compared processing times for unseen data, as shown in Table 11. Ideally, the model should have reduced processing time and operate at a higher speed. Pre-processing involves converting an image from NumPy to PyTorch and normalizing pixel values from 0-255 to 0.0-1.0. Inference time refers to the duration spent within the model, while post-processing involves Non-Maximum Suppression (NMS) in object detection models like YOLO. Based on the results, YOLOv8n and YOLOv5n outperform others. YOLOv8n is selected as the primary model for Android mobile implementation due to its lower inference and total processing time.

Table 9. The performance of YOLOv8n with various batch size values.

Algorithm	Batch size	Epoch	mAP_0.5	mAP_0.5:0.95
YOLOv8n, Optimizer=SGD, Lr = 0.001	16	24	0.971	0.63
	32	31	0.975	0.624
	64	23	0.967	0.623

Table 10. The performance of YOLOv5n with various batch size values.

Algorithm	Batch size	Epoch	mAP_0.5	mAP_0.5:0.95
YOLOv5n Optimizer=SGD, Lr=0.001	16	81	0.969	0.621
	32	65	0.974	0.619
	64	58	0.966	0.622

Table 11. Evaluation of test dataset (unseen data).

Algorithm	Pre-process time (ms)	Inference time (ms)	Post-process (ms)	Total Time Processing (ms)
YOLOv8n	1.9	4.4	2.1	8.4
YOLOv5n	0.5	6.9	1.5	8.9

4.4. Model validation

The confusion matrix evaluates a model's classification performance by comparing predictions to actual labels, providing insights into label distributions. Represented in a table format, each cell signifies counts or proportions of instances in specific categories. True Positive, True Negative, False Positive, and False Negative classifications are defined within this matrix. Evaluation metrics like accuracy, precision, recall, specificity, and F1-score are derived from this analysis. These metrics aid in identifying performance issues such as imbalanced classes or high false positive/negative rates.

- Accuracy: The overall accuracy of the model calculated as (TP + TN) / (TP + TN + FP + FN).
- Precision: The proportion of true positive predictions out of all positive predictions, calculated as TP / (TP + FP).
- Mean Average Precision (mAP): the mean value of the AP across different object classes, offering an overall evaluation of the model's performance. It considers both detection accuracy and the number of instances per class. Higher mAP scores signify superior object detection performance overall.
- Recall (Sensitivity or True Positive Rate): The proportion of true positive predictions out of all actual positive instances, calculated as TP / (TP + FN).
- Specificity: The proportion of true negative predictions out of all actual negative instances, calculated as TN / (TN + FP).
- F1-score: The harmonic means of precision and recall, providing a balanced measure between the two metrics. It is calculated as 2 * (Precision * Recall) / (Precision + Recall).

Figure 8(a) displays the result of YOLOv8n with an accuracy of 0.97 and a false negative rate of 0.03. The confusion matrix reveals a notable number of false positive detections, indicating instances where the system identifies non-existent plates. However, it shows relatively fewer false negatives, implying that while it may over-detect plates, it maintains a satisfactory accuracy in not missing genuine plates.

Figure 8(b) indicates an F1-score of 0.96 with a confidence threshold of 0.51, reflecting a balanced precision and recall for object detection. A precision of 1.00 in Fig. 8(c) signifies accurate predictions for confidence scores above 0.989, with very few false positives. Figure 8(d) shows precision of 0.971 suggests correct positive predictions 97.1% of the time, with minimal false positives. Similarly, a recall value of 0.971 indicates accurate detection of 97.1% of instances. An mAP@0.5 value of 0.971 denotes high accuracy and recall across classes with an Intersection over Union (IoU) threshold of 0.5.

Figure 9 presents examples of plate detection results on unseen data, showcasing improved performance under various challenging conditions like low lighting. However, some false positives, like car logos resembling plates, are detected, necessitating analysis of the threshold value to minimize such occurrences.

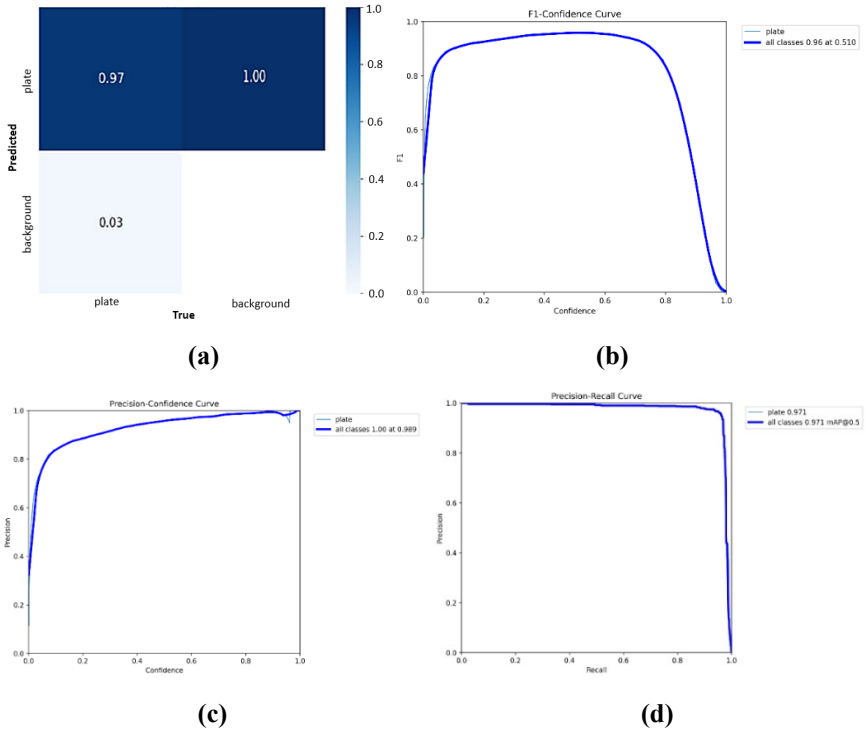

Fig. 8. (a) Confusion matrix, (b) F1-confidence curve graph, (c) Precision-confidence curve graph, (d) Precision-confidence curve graph.

Fig. 9. Example of detection plate result on unseen data.

4.5. Plate detection analysis

YOLOv8n demonstrates higher mAP values (0.5:0.95) compared to other models, as indicated in Table 12. Subsequently, the converted model is tested on the mobile application using unseen data (test images).

High accuracies, ranging from 85% to 95%, are achieved for each image, indicating the robustness of the detection system due to the large training dataset. Samples of LPDR running on a mobile device are depicted in Fig. 10, while Fig. 11 showcases samples under unconstrained conditions.

Table 12. System performance based on the evaluation metrics.

Algorithm	Optimizer	Learning Rate	Batch size	Epoch	mAP 0.5	mAP 0.5:0.95
YOLOv8n	SGD	0.001	32	24	0.975	0.624
YOLOv5n	SGD	0.001	32	81	0.974	0.619

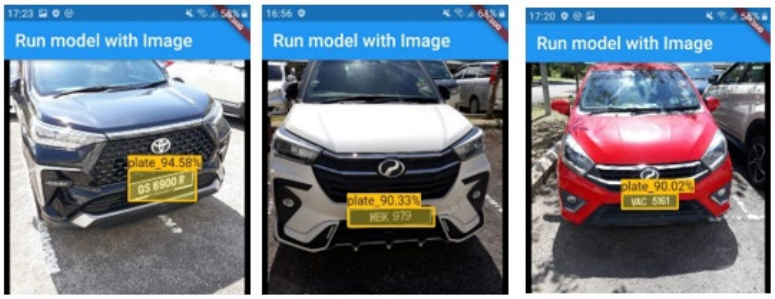

Fig. 10. Result of model testing on mobile device with accuracy.

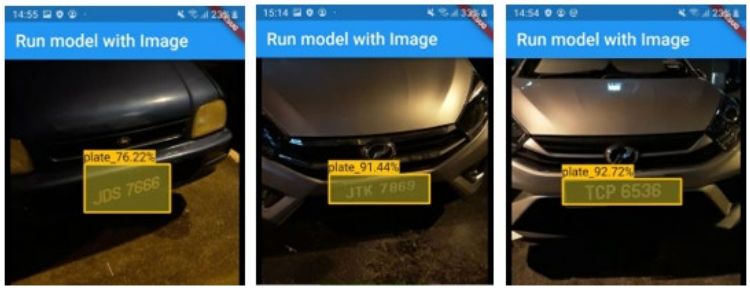

Fig. 11. Result of model testing under unconstrained condition.

4.6. Character recognition analysis

Two techniques, YOLOv8n Second Stage Method and ML Kit OCR, were examined and compared, with the first stage being the YOLOv8n license plate detector. The YOLOv8n second stage underwent customized hyperparameter tuning. Upon analysis, it was observed that the trained YOLOv8n model had more misclassifications than successful character recognitions. Occasionally, it missed one or multiple characters during recognition, likely due to the variety of fonts used and the insufficient training data available for effective learning and generalization. Additionally, the model exhibited high sensitivity to the angle of the captured image.

Testing across various angles is crucial to ensure system robustness. If input images lack centering or are rotated, the model may struggle to classify characters accurately, resulting in a higher false positive rate. Increasing the threshold may lead to fewer characters being recognized. For instance, if the actual plate number is VVA5836, the model may only recognize VA36. The highest misclassification occurred at low threshold values. False positives decreased with a higher initial threshold. Figure 12 illustrates the impact of threshold values on the model.

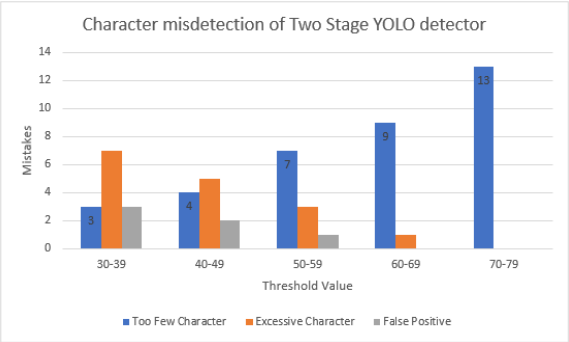

Fig. 12. Threshold values applied on YOLOv8n character recognition.

Initially, we tested YOLOv8n two-stage detection and recognition, and subsequently, the integration of YOLOv8n with ML Kit OCR. However, the integration of YOLOv8n and ML Kit OCR outperformed YOLOv8n Two-Stage LPDR, correctly identifying 7 out of 10 license plates by recognizing the full characters. The failure to detect the remaining 3 images was primarily due to extreme angles of the captured license plates. All classes achieved an excellent mAP, with at least 85% accuracy. Table 13 displays the results of various character recognition test.

Table 13. Results of character recognition.

Image	Actual Text	YOLOv8n Predicted Text	(YOLOv8n) Text Detected correctly?	ML Kit OCR Predicted Text	(ML Kit OCR) Text Detected correctly?
1	WSD8231	WSD8231	Yes	WSD8231	Yes
2	MCW6015	WW6015	No	MCW6015	Yes
3	ST5148X	3T5148X	No	ST5148X	Yes
4	JTP9776	TF776	No	JTP9776	Yes
5	WA1627Y	WA1627Y	Yes	-	No
6	VAC5161	VA561	No	VAC5161	Yes
7	BPN3068	BRN066	No	BPN3068	Yes
8	TCP6536	CP3	No	TCP6536	Yes
9	UTM6155	UTM6155	Yes	UTM655	No
10	VCA2983	VA293	No	-	No

The method was evaluated under various conditions, including low image quality, uneven lighting, reflections, shadows, tilting, and rotation, with ML Kit accurately recognizing most scenarios. ML Kit OCR demonstrates exceptional performance in quick and accurate license plate detection, particularly on mobile phones. Its suitability for mobile app development stems from its smaller OCR

architecture compared to the Two Stage YOLO detector, especially considering that Malaysian plate numbers only use alphanumeric characters.

Deploying two YOLOv8n models in a single app resulted in frequent app unresponsiveness and instability. Character recognition results are shown in Fig. 13, while Fig. 14 demonstrates the integration with the database system for vehicle ownership lookup. The recognition performance for YOLOv8n and ML Kit OCR integration is 91.5% accuracy tested towards 521 test samples.

The comparison with existing works is detailed in Table 14, revealing that the mobile app developed in this research is comparable to other LPDR approaches. A robust LPDR system enables the security personnel to check vehicle details, including the owner (student, staff, or visitor), potential wrongdoings, or unpaid summons using their personal smartphone in various lighting conditions, vehicle distances, license plate angles, and weather conditions.

This verification process involves comparing the vehicle with a predefined list of authorized vehicles which is stored in the cloud database. Once the vehicle's owner is identified, security personnel can take further actions, such as issuing a warning letter or summons to the campus residents. This mobile application ensures the confidentiality of vehicle owner information, accessible solely to security department personnel. While no surveys have been conducted to gather user feedback or perspectives from campus stakeholders, this is planned for the future.

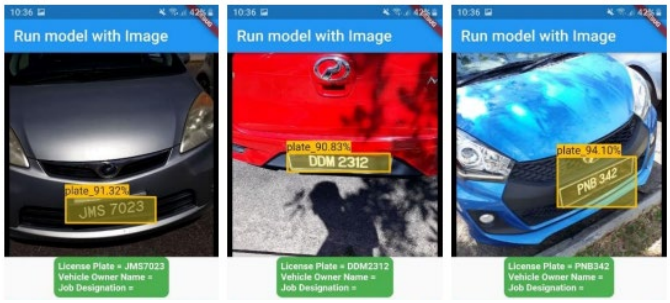

Fig. 13. YOLOv8n plate detection and ML Kit OCR for character recognition.

Fig. 14. Integration with database system for vehicle ownership lookup.

Table 14. Results comparison with other YOLO approaches.

Ref.	Methods	Results
[7]	Two-stage YOLOv2	LP detection for clear weather 99% but 74% for night scene.
[8]	YOLOv4 and Tesseract OCR	Achieved 94.6% accuracy on the detection rate but recognition rate was not reported.
[9]	YOLOv6	The F1-score is 0.95.
[12]	Tiny-YOLOv2	99.62% mAP and 9 ms inference time.
[14]	Fast-YOLO	93.53% accuracy and 47 fps.
This paper	YOLOv8n and ML Kit OCR	97.5% detection accuracy, 91.2% recognition accuracy for 521 test images.

5. Conclusion

The main objective of this research is to design a lightweight mobile-based license plate reader (LPDR) for Malaysian license plates to assist security personnel in enhancing smart campus security and vehicle management. Four lightweight models were analysed for the license plate detection, with YOLOv8n identified as the best detection model. For character recognition, two models were evaluated, and ML Kit OCR was selected.

The evaluation of the YOLOv8n algorithm's accuracy as a license plate detector involves crucial sections such as hyperparameter tuning and speed evaluation. Hyperparameter tuning optimized the model, focusing on parameters like batch size, learning rate, and optimizer. The plate detection accuracy reached 97.5% using a batch size of 32, a learning rate of 0.001, and the SGD optimizer. YOLOv8n also achieved a total processing time of 8.4 milliseconds.

ML Kit OCR achieved a character recognition accuracy of 91.2%, outperforming YOLOv8n second stage. YOLOv8n second stage required extensive training datasets covering various characters and styles, leading to inaccurate character detection due to insufficient data. This approach may result in larger model sizes, requiring more computational resources for efficient inference. In contrast, ML Kit OCR is designed to be lightweight and optimized for mobile devices, resulting in smaller model sizes and faster inference on resource-constrained platforms.

To enhance the LPDR system's performance, a database cloud system was integrated to match recognized license plate numbers with vehicle ownership information. This integration significantly improves the system's overall performance, making it highly suitable for end-to-end license plate reading. The LPDR mobile app is applicable to any campus with access to the resident's vehicle database. Further improvements in the app's operation may necessitate using a better end device with higher hardware specifications. Additionally, gathering user feedback and perspectives from campus stakeholders through a future survey will provide valuable insights.

Acknowledgement

The authors would like to acknowledge the support from the Faculty's Research Fund (Faculty of Electronics and Computer Technology and Engineering), Machine Learning and Signal Processing (MLSP), a research group under the Centre for Telecommunication Research & Innovation (CeTRI), Universiti Teknikal Malaysia Melaka (UTeM).

References

1. Mustafa, T.; and Karabatak, M. (2023). Challenges in automatic license plate recognition system review. *Proceedings of the* 2023 11th *International Symposium on Digital Forensics and Security*, Chattanooga, TN, USA, 1-6.
2. Abdullah, M.; Al-Nawah, S.M.; Osman, H.; and Jaffar, J. (2021). License plate recognition techniques: comparative study. *Malaysian Journal of Computer Science*, 1(*Advances in Applied Science and Technology Research Special Issue*), 94-105.
3. Wang, Y.; Wang, J.; Zhang, W.; Zhan, Y.; Guo, S.; Zheng, Q.; and Wang, X. (2022). A survey on deploying mobile deep learning applications: A systemic and technical perspective. *Digital Communications and Networks*, 8(1), 1-17.
4. Leng, J. et al. (2023). A light vehicle license-plate-recognition system based on hybrid edge-cloud computing. *Sensors*, 23(21), 8913.
5. Ma, Z.; Wu, Z.; and Cao, Y. (2023). End-to-end light license plate detection and recognition method based on deep learning. *Electronics*, 12(1), 203.
6. Dias, C.; Jagetiya, A.; and Chaurasia, S. (2019). Anonymous vehicle detection for secure campuses: A framework for license plate recognition using deep learning. *Proceedings of the* 2019 2nd *International Conference on Intelligent Communication and Computational Techniques* (*ICCT*), Jaipur, India, 79-82.
7. Yonetsu, S.; Iwamoto, Y.; and Chen, Y.W. (2019). Two-stage YOLOv2 for accurate license-plate detection in complex scenes. *Proceedings of the* 2019 *IEEE International Conference on Consumer Electronics* (*ICCE*), Las Vegas, NV, USA, 1-4.
8. Chang, C.-L.; Chen, P.-J.; and Chen, C.-Y. (2022). Semi-supervised learning for YOLOv4 object detection in license plate recognition system. *Journal of Imaging Science and Technology*, 66(4), 040404-1-040404-9.
9. Li, M.; and Zhang, L. (2023). Deep learning-based license plate recognition in IoT smart parking systems using YOLOv6 algorithm. *International Journal of Advanced Computer Science and Applications*, 14(12), 222-231.
10. Awalgaonkar, N.; Bartakke, P.; and Chaugule, R. (2021). Automatic license plate recognition system using SSD. *Proceedings of the* 2021 *International Symposium of Asian Control Association on Intelligent Robotics and Industrial Automation* (*IRIA*), Goa, India, 394-399.
11. Darji, M.; Dave, J.; Asif, N.; Godawat, C.; Chudasama, V.; and Upla, K. (2020). Licence plate identification and recognition for non-helmeted motorcyclists using light-weight convolution neural network. *Proceedings of the* 2020 *International Conference for Emerging Technology* (*INCET*), Belgaum, India, 1-6.
12. Lin, C.-H.; and Wu, C.-H. (2019). A lightweight, high-performance multi-angle license plate recognition model. *Proceedings of the* 2019 *International Conference on Advanced Mechatronic Systems* (*ICAMechS*), Kusatsu, Japan, 235-240.
13. Xu, Z.; Yang, W.; Meng, A.; Lu, N.; Huang, H.; Ying, C.; and Huang, L. (2018). *Towards end-to-end license plate detection and recognition: A large dataset and baseline*. In Ferrari, V.; Hebert, M.; Sminchisescu, C.; and Weiss, Y. (Eds.), *ComputerVision-ECCV* 2018. Springer International Publishing, 11217, 261-277.

14. Laroca, R.; Severo, E.; Zanlorensi, L.A.; Oliveira, L.S.; Goncalves, G.R.; Schwartz, W.R., and Menotti, D. (2018). A robust real-time automatic license plate recognition based on the YOLO detector. *Proceedings of the* 2018 *International Joint Conference on Neural Networks* (*IJCNN*), Rio de Janeiro, Brazil, 1-10.
15. Kim, H.-H.; Park, J.-K.; Oh, J.-H.; and Kang, D.-J. (2017). Multi-task convolutional neural network system for license plate recognition. *International Journal of Control*, *Automation and Systems*, 15(6), 2942-2949.
16. Montazzolli, S.; and Jung, C. (2017). Real-time brazilian license plate detection and recognition using deep convolutional neural networks. *Proceedings of the 2017 30th SIBGRAPI Conference on Graphics, Patterns and Images* (*SIBGRAPI*), Niteroi, Brazil, 55-62.
17. Foundation, T.L. (2023). PyTorch Mobile. Retrieved December 5, 2023, from https://pytorch.org/mobile/home/.