Journal of Electrical and Computer Engineering Innovations
(JECEI)
Journal homepage: http://www.jecei.sru.ac.ir

Research paper

Artificial Intelligence-Based YOLOv3 using DarkNet-53 Deep Convolutional Neural Network Model Architecture for Automatic Vehicle Inventory System Design with a Dynamic Relational Database System and Google Cloud Storage

Vincent Andrew Akpan [1,*], ***David Ayo-oluwa Adegoke*** [2], ***Ebunoluwa Temiloluwa Adejayan*** [3], ***Kehinde Adesola Adepoju*** [4]

[1]*Department of Biomedical Engineering, The Federal University of Technology, Akure, Ondo State, Nigeria.*
[2]*Biomedical Engineering Department, Redeemer's Health Village, Redemption City of God, Mowe, Ogun State, Nigeria.*
[3]*Biomedical Engineering Unit, Federal Medical Centre, MMA Road, Owo, Ondo State, Nigeria.*
[4]*Department of Biomedical Engineering, Lagos State University Teaching Hospital (LUTH), Yaba, Lagos State, Nigeria.*

Article Info

Article History:
Received 22 December 2025
Reviewed 06 January 2026
Revised 24 January 2026
Accepted 29 January 2026

Keywords:
Automatic Vehicle Inventory System (AVIS)
Computer vision
Real-time object detection
Relational Database System (RDBS)
You Only Look Once version 3 (YOLO v3)

*Corresponding Author's Email Address: *vaakpan@futa.edu.ng*

Abstract

Background and Objectives: The manual method of writing down vehicle plate numbers (VPNs), vehicle types, date, and time-stamps at the point of entry into and/or exit from the premises of organizations as well as the exit and/or entry time are not only time-consuming and stressful but are also prone to errors, delays, inconsistency and possible loss of hand-written data due to possible environmental hazards which makes this archaic method unreliable especially for security reasons. This study presents an artificial intelligence-based YOLOv3 using DarkNet-53 deep convolutional neural network (CNN) model architecture for the development of an automatic vehicle inventory system (AVIS) with a PostgreSQL-based dynamic relational database system (RDBS) for captured data storage and retrieval in real-time using Google cloud storage/retrieval.
Methods: The AVIS with dynamic RDBS employs power-over-Ethernet (PoE) switch, PoE IP-based camera, Airtel router/Wi-Fi module and YOLOv3 using DarkNet-53 algorithm to capture and process VPNs from streaming video of moving vehicles. The processed results are stored in a properly designed dynamic RDBS over Google cloud storage system. The dynamic RDBS automatically creates and inserts all relevant vehicle information for security surveillance and tracking purposes. Several standard quantitative and qualitative metrics have been used to evaluate the performances of the YOLOv3 using DarkNet-53 architecture against YOLOv8 using CSPDarkNet-53 and YOLOv3 using SqueezeNet model architectures for comparison purposes.
Results: Quantitatively, the YOLOv3 using DarkNet-53 and YOLOv8 using CSPDarkNet-53 achieved virtually equal performance metrics except for the excessive long execution time of 4.5839 hours used by YOLOv8 with CSPDarkNet-53compared to the 2.9713 minutes used by the YOLOv3 with DarkNet-53. The YOLOv3 with SqueezeNet used only 1.9901 minutes with relatively lower performance metric values. Qualitatively, successful and accurate LPNs detection and recognition with dynamic RDBS update to the cloud within 3 seconds for 25 random vehicles entering and/or exiting the premises of a car dealer company for a period of three days between 10:00am and 2:00pm daily has been achieved with YOLOv3 using DarkNet-53 model architecture.
Conclusion: The proposed low-cost AVIS based on YOLOv3 using DarkNet-53 model architecture with a PostgreSQL dynamic RDBS and Google cloud storage have been successfully designed, implemented and validated with successful results for LPN detection and recognition. The proposed techniques offer promising potentials for timely and accurate data collection to optimize vehicle inventory management, control and operations for security surveillance design.

This work is distributed under the CC BY license (http://creativecommons.org/licenses/by/4.0/)

How to cite this paper:
V. A. Akpan, D. A. Adegoke, E. T. Adejayan, K. A. Adepoju, "Artificial intelligence-based YOLOv3 using darknet-53 deep convolutional neural network model architecture for automatic vehicle inventory system design with a dynamic relational database system and google cloud storage," J. Electr. Comput. Eng. Innovations, 14(2): 435-472, 2026.
DOI: 10.22061/jecei.2026.12552.889
URL: https://jecei.sru.ac.ir/article_12554.html

2345- 3044/© 2026 The Authors. Published by Shahid Rajaee Teacher Training University.

Introduction

This work arose from the observations of manual vehicle inventory methods among some higher institutions and organizations in Nigeria. The recognition of Nigerian vehicle number plates has been reported in [1]. The observed methods involved manual inventory of license plate numbers (LPNs), vehicle types and the time of entry into organizations as well as the exit time. Apart from being time-consuming and stressful, the current methods of manual vehicle inventory management are prone to errors, delays, and inconsistency, which make it unreliable especially for security reasons. Furthermore, Atieh et al. noted that automated data entry can improve inventory management processes and reduce costs associated with manual data entry [2]. To address these challenges, there is a need to develop an automatic vehicle inventory system (AVIS) with dynamic database that can automatically and accurately identify and recognize vehicles with LPNs, captures the LPNs, time-stamps (entry and exit times) and update the inventory database in real-time.

The need for efficient and automatic vehicle inventory management system has become more critical in recent times. The development of an automatic vehicle inventory system (AVIS) with a dynamic relational database system (RDBS) is a crucial area of research for vehicle security and in the transportation industry [3]–[5]. To the best of the authors knowledge and unfortunately, this is the first work of this nature and subsequently there are no literature on the current study. It is important to note that the current work differs from vehicle inventory management systems and automotive inventory management software reported in [6]–[9]. The aim of this research is to develop an artificial intelligent-based convolutional neural network for automatic vehicle inventory system that automatically captures and stores the license plate numbers of vehicles together with the time-stamps (entry and exit times) using IP cameras and a deep learning algorithm. The captured data will then be stored and updated automatically in a dynamic relational database system, which can be accessed in real-time by authorized personnel.

The use of deep learning algorithms is critical in computer vision based systems for recognition, feature extraction, detection, segmentation, localization with endless applications. The most popular and recent computer vision deep learning algorithms are the You Only Look Once (YOLO) family of algorithms [10]–[14]. Among the YOLO family of algorithms, YOLO v3 have been proven to be effective in recognizing and identifying objects in real-time with several advantages compared to other versions of the YOLO algorithm and found suitable for this work [15]–[19]. In a study by Rajput et al. [20], the YOLOv3 algorithm was used to develop an automatic vehicle identification and classification model for a toll management system. The study reported high accuracy in identifying and classifying vehicles based on their license plate numbers. Similarly, Li in [21] used the RES-YOLO model to develop a deep learning automatic vehicle recognition algorithm, which reported a faster processing time and better accuracy compared to other existing models. These findings highlight the importance of deep learning algorithms in feature extraction and pattern recognition and a potential tool for the development of the proposed automatic vehicle inventory systems.

The integration of different technologies such as an IP camera, a router, and RJ-45 cables is vital for the enhancement of the proposed automatic vehicle inventory system development. In 2021, Wu et al. [22] developed an automatic vehicle detection system that uses roadside LiDAR data to detect vehicles under rainy and snowy conditions. The study reported that the system was effective in detecting vehicles in adverse weather conditions. The use of PoE IP-based camera, routers, RJ-45 cables combined with the YOLOv3 algorithm will enhance automatic capture of license plate numbers and vehicle classification, which can be transmitted to a remote location for storage and analysis. Furthermore, the integration of Google Cloud Storage and PostgreSQL as the storage system will enable real-time access to captured data and provide a secure and reliable database infrastructure. An integrated positioning system is another critical component that could enhance the development of the proposed automatic vehicle inventory system. Cui et al. developed an integrated positioning system for unmanned automatic vehicles in coal mines which improved efficiency and safety of real-time vehicle positioning [23].

One of the key areas of research in the possible development of an automatic vehicle inventory system is the use of computer vision and machine learning techniques for vehicle detection and classification. Researchers in the field have proposed various methods for detecting and classifying vehicles, including the use of convolutional neural networks (CNNs) and ensemble classifiers [9], [26]–[27]. Zhang et al. [28] show that real-time monitoring in inventory systems significantly reduces lead times and improves performance.

The automatic vehicle inventory system also includes dynamic relational database management. This feature allows businesses to store and analyze inventory data effectively and automatically. The use of PostgreSQL as storage and/or retrieval system ensures that businesses can easily access and analyze data, making informed decisions about inventory management and control [29]. Dynamic relational database systems are essential in the

development of an automatic vehicle inventory system like the one proposed in this research. One of the benefits of using a dynamic relational database is the ability to handle large amounts of data from various sources accurately and efficiently. The proposed system, which captures vehicle license plate numbers using YOLOv3 with Darknet-53 and an Internet Protocol (IP)-based camera, generates a large amount of data that must be accurately and efficiently stored and managed. A dynamic relational database system such as the PostgreSQL makes it possible to store and retrieve data in a structured and comprehensive manner, ensuring that the system can handle the volume of data generated effectively [30], [31]. By analyzing the data in real-time, the system can quickly identify any anomalies or discrepancies, allowing for immediate action to be taken [24], [25], [32]. Additionally, the use of a cloud-based database system like PostgreSQL, coupled with Google cloud storage, provides the flexibility to access data from anywhere, as well as providing an added layer of security and redundancy [33]–[35].

Related Work

Due to the originality of this work, there is no existing literature on this specific topic. However, the various components of the proposed AVIS with dynamic RDBS are carefully considered.

Automatic Vehicle Location (AVL) data are fundamental for measuring service reliability, which is essential for improving transportation system efficiency. According to Ma et al. [36], AVL data can improve the accuracy of arrival time prediction, delay detection, and route optimization. AVL can also facilitate the collection of vehicle information such as speed, direction, and location, which are useful for developing an automatic vehicle inventory system. Nieto et al. [8] further emphasized the importance of automatic vehicle detection in parking management systems.

The license plate recognition (LPR) system can detect and identify vehicles based on their number plates. The use of structured elements has been found to improve the accuracy of automatic vehicle license plate recognition [37]. According to Prabhakar et al. [38] , automatic vehicle license plate recognition technology can identify stolen vehicles, enforce traffic regulations, and prevent traffic congestion. The authors suggested a method of detecting and recognizing number plates by optimizing the image captured using image processing techniques.

Aerial remote sensing has been used to extract vehicle trajectories from traffic videos. Azevedo et al. [39] proposed a methodology for automatic vehicle trajectory extraction using aerial remote sensing. Similarly, Wu and Xu [40] proposed an automatic procedure for vehicle tracking using a roadside LiDAR sensor. The authors used a LiDAR sensor to scan the road surface and detect vehicles based on their reflection patterns. The proposed system achieved high accuracy in detecting and tracking vehicles.

Sulaiman [41] proposed a development of an automatic vehicle plate detection system that uses image processing techniques to detect and recognize vehicle number plates. The authors demonstrated that their proposed system is capable of detecting and recognizing license plate numbers (LPNs) with high accuracy using machine learning algorithms.

Furthermore, existing studies have explored various automatic vehicle identification systems that use license-plate recognition technology. For instance, Duan et al. [42] developed an automatic vehicle license-plate recognition system that uses a deep learning algorithm to improve the accuracy of the identification process. Similarly, Kim and Song [43] proposed a vehicle recognition system that uses radar and vision sensors to identify vehicles and enable automatic emergency braking. These studies show that automatic vehicle identification systems are effective in enhancing efficiency and accuracy in vehicle management.

Wang and Chan [44] argue that a robust replenishment and production control policy can optimize inventory control in a single-stage production/inventory system with inventory inaccuracy. This implies that the proposed AVIS with dynamic RDBS must incorporate an effective inventory control mechanism to ensure that captured vehicle plate numbers are properly stored in the database system.

The automatic vehicle inventory system must have an efficient queuing-inventory system to handle the captured plate numbers effectively. Krishnamoorthy and folks [45] argue that analyzing a multi-server queuing-inventory system can help optimize the system's performance. The automatic vehicle inventory system must have a multi-server queuing-inventory system to store and retrieve captured plate numbers efficiently.

A. Automatic Vehicle Inventory Technologies

Automatic vehicle inventory systems are becoming increasingly popular as they provide a more efficient way of managing vehicle identification and tracking. One of the most commonly used types of automatic vehicle inventory systems is the Automatic Number Plate Recognition system [46]. Automatic number plate recognition (ANPR) systems use image processing techniques and optical character recognition (OCR) algorithms to capture and recognize license plate numbers from Closed Circuit Television (CCTV) footage or images captured by cameras [47]. The system extracts the vehicle's plate number and stores it in a database, enabling easy identification and tracking of vehicles.

Another type of automatic vehicle inventory system is the Automatic Vehicle Identification (AVI) system. This system uses radio-frequency identification (RFID) technology to identify and track vehicles as they pass through a checkpoint. RFID tags, which are attached to the vehicle, are read by the AVI system when the vehicle passes through the checkpoint, and the data is stored in a centralized database [30], [48]–[50]. The data collected by the AVI system can be used for various purposes, including traffic management, toll collection, and security.

A third type of automatic vehicle inventory system is the Automatic Vehicle Location (AVL) system. AVL systems use Global Positioning System (GPS) technology to track the location of vehicles in real-time. The system collects data such as the vehicle's location, speed, and direction of travel, and stores it in a centralized database [51]. The data collected by the AVL system can be used for various purposes, including route planning, fleet management, and emergency response.

B. Challenges of Automatic Vehicle Inventory Technologies

Firstly, there is a need for constant Internet connectivity to access the system, as it requires the use of an IP camera and router. This poses a challenge in areas with poor internet connectivity and may lead to a delay in updating the database. Furthermore, the use of an IP camera makes the system susceptible to intrusion by external parties, which could compromise the confidentiality of the stored data [52].

Secondly, the system's accuracy is dependent on the quality of the image captured by the camera. The use of YOLO v3, an object detection algorithm that identifies and localizes vehicles' LPNs, is highly dependent on the clarity of the image. Poor image quality due to weather conditions or other factors may lead to inaccurate readings of the plate number [39], [53], [54]. This could result in discrepancies in the inventory records, leading to incorrect information about the vehicles in the system.

Thirdly, the storage capacity of the database limits the number of vehicles that can be captured and recorded. While using Google Cloud Storage and PostgreSQL provides sufficient capacity for storing vehicle information, the cost of maintaining and upgrading the system may be high. [55]–[57]. Additionally, the system requires a reliable power supply to function properly, and any power outage could lead to data loss or system malfunction.

Lastly, the capturing of vehicle LPNs could be time-consuming, leading to a delay in the system's operation. This is especially true in areas with high traffic, where several vehicles may need to be captured within a short time frame. The delay in capturing and processing the information could lead to the vehicle's exit before the system records it, resulting in inaccurate inventory records.

C. Overview of the Proposed Research

The proposed AVIS with dynamic RDBS will adopt the well-proven YOLOv3 using DarkNet-53 model architecture which will be implemented using Python with its associated software on core-*i*7 laptop (HP Pavilion Plus Laptop 16-ab0010nr, Windows 11 Home, 16", Intel® Core™ *i*7-14700HX operating at 5.5 GHz, 16GB RAM, 512GB SSD, WQXGA, Natural silver). The hardware shall include power-over-Ethernet (PoE) switch (LS108GP 8-Port 2 Gigabit Desktop Switch with 8-Port PoE and 2 PoE Uplink ports), SONY Vision CCTV IP-based SV-5MP-POE-4B PAL 120m/2.8mm Camera, Airtel 4G Smartbox Router/Wi-Fi module, RJ-45 cables and power supply unit (which includes two Li-Po batteries, 12-V battery charger, 220-V to 12-V converter for the router, and a boost converter and power switching interface). Additional software will include Google Cloud Storage and PostgreSQL to support data storage and retrieval. The use of YOLOv3 algorithm will provide high accuracy vehicle plate number recognition and capturing while the Google Cloud Storage and PostgreSQL will provide a scalable and secure storage solution [24], [25], [58]. Overall, the proposed AVIS with dynamic RDBS based on YOLOv3 using Darknet-53 will be compared with YOLOv8 using CSPDarknet-53 and YOLOv3 using SqueezeNet, and performance comparisons will be made, with conclusions drawn based on several quantitative and qualitative benchmarking criteria and performance metrics.

Materials

The materials required for the development and implementation of the proposed automated AVIS with dynamic RDBS inventory system involve both hardware and software. The hardware include one core-*i*7 laptop (HP Pavilion Plus Laptop 16-ab0010nr, Windows 11 Home, 16", Intel® Core™ *i*7-14700HX operating at 5.5 GHz, 16GB RAM, 512GB SSD, WQXGA, Natural silver); a SONY Vision CCTV IP-based SV-5MP-POE-4B PAL 120m/2.8mm camera; Airtel 4G Smartbox Router/Wi-Fi with Fast Internet connectivity; LS108GP 8-Port 2 Gigabit Desktop Switch with 8-Port PoE and 2 PoE Uplink ports; RJ-45 cables; and a power supply unit (which includes two Li-Po batteries, 12-V battery charger, 220-V to 12-V converter for the router, and a boost converter and power switching interface). The software components include Python 3.13, Matplotlib 3.10.3, OpenCV 4.12, TensorFlow 2.16, Keras 3.10, PostgreSQL database (dynamic relational database system) and Google cloud storage as well as YOLOv3 using DarkNet-53, YOLOv8 using CSPDarkNet-53 and YOLOv3 using SqueezeNet model architectural algorithms.

Methodology

A. Block Diagram of the AVIS

The development of the proposed automatic vehicle inventory system (AVIS) with dynamic relational database system (RDBS) is a revolutionary approach in the efficient, accurate and accountable management strategy for vehicle inventory into an organization. The block diagram of the AVIS with dynamic RDBS is shown in Fig. 1. It illustrates the connections among the IP camera, Ethernet switch, router, and Wi-Fi module, and shows how these components interact within the AVIS network.

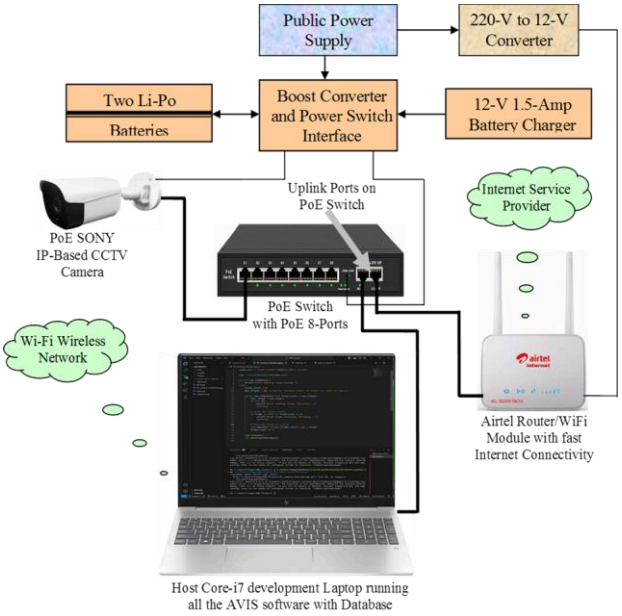


Fig. 1: Block diagram of the automatic vehicle inventory system (AVIS).

The IP camera is connected to the switch via an Ethernet cable, and the switch is in turn connected to the router via Ethernet. The Wi-Fi module, typically integrated into the router, provides wireless connectivity to other devices on the network. The main components are as follows:

1). IP Camera: This is the surveillance camera that captures video footage and transmits it over the network. It connects to the network using an Ethernet cable.

2). Ethernet Switch: The switch acts as a central connection point for multiple IP cameras (or other wired devices). It receives data from the cameras and forwards it to the router.

3). Router: The router manages the network traffic, connecting the local network (LAN) to the internet (WAN). It receives data from the switch and routes it to the appropriate destination, including the internet or other devices on the local network.

4). Wi-Fi Module: The Wi-Fi module, often integrated into the router, enables wireless communication. The Wi-Fi module allows the host laptops to connect to the network wirelessly.

In the simplest case, the connection is as follows: *(i) Camera to Switch:* An Ethernet cable connects the IP camera to one of the ports on the Ethernet switch; *(ii) Switch to Router:* Another Ethernet cable connects the switch to a port on the router; *(iii) Router to Internet:* The router is connected to the internet service provider's modem, usually through a dedicated WAN port; and *(iv) Wi-Fi Module:* The Wi-Fi module is part of the router or a separate access point, providing wireless connectivity to devices within its range.

However, a high-level performance architecture with the power-over-Ethernet (PoE) switch with 2 uplink ports connected to the router and the host laptop which also acts as a network video recorder (NVR) is used in this work. This approach is advantageous as it allows the host laptop to be connected directly to the PoE switch and fetch data directly from the camera without having data bog down the router.

B. The YOLOv3: Description and Algorithm

B. 1. Architectural Description

The You Only Look Once, Version 3 (YOLOv3) is a real-time one-shot object detection algorithm that uses a dynamic pre-trained DarkNet-53 deep convolutional neural network (CNN) model architecture as the backbone to identify, segment and locate objects and patterns within images and videos [13], [17], [59], [60]. The architectural description of the YOLOv3 algorithm is shown in Fig. 2. The YOLOv3 algorithm takes an image or streaming video as input and then uses a deep CNN called DarkNet-53 to detect objects in the image or streaming video [15]. DarkNet-53 is derived from the ResNet architecture and it is tailor-made for object detection tasks, boasting 53 convolutional layers and achieving top-notch performance across various object detection benchmarks [11]. YOLOv3 with Darknet-53 is an efficient algorithm due to its speed and ability to detect multiple objects simultaneously [13], [59]. After the release of YOLOv3 in 2018, several versions of YOLO have been released up to YOLOv12 but the several advantages why YOLOv3 is used in this work can be found in [12] and [13].

The structure of the YOLOv3 with DarkNet-53 model architecture includes: *(i) Single Neural Network:* YOLOv3 uses a single convolutional neural network to predict bounding boxes and class probabilities directly from the image; *(ii) DarkNet-53:* The DarkNet-53 is built on a modified version of the DarkNet-53 convolutional neural network, which includes residual blocks to help with gradient flow in deep networks; *(iii) Feature Pyramid Network (FPN):* YOLOv3 utilizes FPN to extract

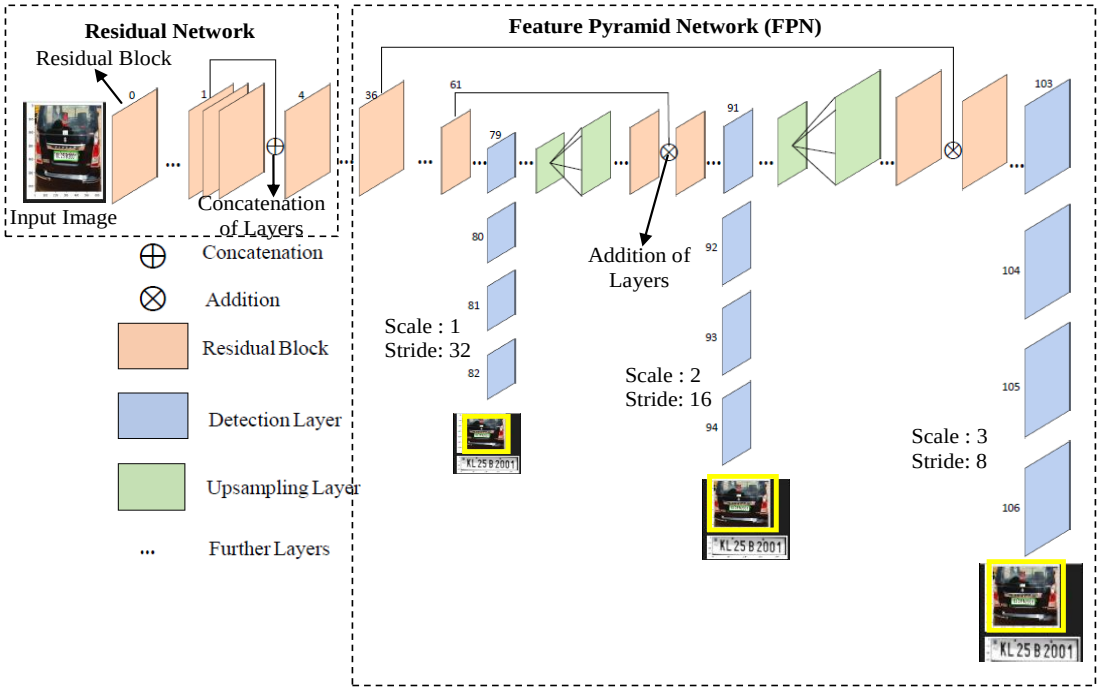


Fig. 2: Architecture of the YOLOv3 using DarkNet-53 with residual and feature pyramid networks.

features at different scales, allowing it to detect objects of various sizes; and *(iv) Anchor Boxes:* YOLOv3 uses anchor boxes (predefined bounding boxes with different aspect ratios and scales) to improve the detection of objects with varying shapes and sizes, according to [11].

YOLOv3 with DarkNet-53 uses a deep CNN to detect objects in images or streaming videos. YOLO is a single-shot object detection algorithm that is able to detect objects within an image or video in a single pass without the need for multiple stages. The YOLO family of algorithms are extremely fast because they do not deal with complex pipelines. At the minimum, YOLO algorithm processes images at 45 frames per second (FPS). Furthermore, YOLO algorithms reach more than twice the mean average precision (mAP) compared to other real-time object detection algorithms [14].

The YOLOv3 using DarkNet-53 model architecture is divided into three parts, namely: a head, a neck, and a backbone. The backbone is a series of convolutional layers that extract features from the input image. The neck combines features from different scales to improve object detection. The head is a set of fully connected layers that predict the location and class of each object in the image. YOLOv3 using DarkNet-53also uses anchor boxes, which are pre-defined bounding boxes of different sizes and aspect ratio. These anchor boxes are used to predict the location and sizes of objects in the image.

B. 2. The YOLOv3 Algorithm: Structure

Based on the YOLOv3 architecture shown in Fig. 2, the YOLOv3 algorithm accepts an image or streaming video as input and applies the CNN pre-trained DarkNet-53 model architectures to detect objects in the image according to the following procedures, namely:*1)* residual networks; *2);* Feature pyramid network (FPN); *3);* Bounding box regression; *4)* Intersection over union (IOU); and *5).* Non-maximum suppression as briefly described below.

1). Residual Network: The residual network accepts and divides the original input image (I) into *S* x *S* grid cells of equal shape by the YOLOv3 classifier, where *S* is a number. Each cell in the grid is responsible for localizing and predicting the class of the object that it covers together with the probability and the confidence value. The residual network also provide skip connections which ensures that inputs coming from one layer of the network are fed into another layer while avoiding vanishing or exploding gradients and prevents convergence degradation problems [14], [17], [61].

2). Feature Pyramid Network (FPN): FPN is not an object detector but rather a feature extractor algorithm. FPN extracts feature maps and later feeds into a feature detector algorithm to generate a pyramid of feature maps and the region-of-interests (ROIs). FPN composes of a bottom-up and a top-down pathway. The bottom-up pathway is the usual convolutional network for feature extraction. As we go up, the spatial resolution decreases. With more high-level structures detected, the semantic value for each layer

increases. The feature map is selected based on the width (w) and the height (h) of the ROI according to the following expression [61], [62]:

$$k = \left[k_0 + \log_2 \left(\frac{\sqrt{wh}}{224} \right) \right] \quad (1)$$

where 224 is the canonical ImageNet pre-training size; k_0 is the target level on which the ROI with $w \times h = 224^2$ should be mapped into; and *k* is the P_k layer in the FPN use to generate the feature patch. While the reconstructed layers are semantically strong, the object locations are not precise due to repeated down-sampling and up-sampling operations. Lateral connections are added between reconstructed layers and the corresponding feature maps to improve the accuracy of object localization. These connections also act as skip connections, facilitating easier training.

3). Bounding Box Regression: The next step in the detection process is the determination of the bounding boxes corresponding to rectangles (as many bounding boxes as there are objects in the given image), highlighting all the objects in the images or streaming video frames. The YOLOv3 determines the attributes of these bounding boxes using a single regression module in the following format:

$$Y = [pc, bx, by, bh, bw. c1, \ldots, cn] \quad (2)$$

where *Y* is the final vector representation of each bounding box; *pc* denotes the probability score of the grid containing an object; *bx* and *by* are the x and y coordinates of the center of the bounding box respectively with respect to the enveloping grid cell; *bh* and *bw* are the corresponding height and width of the bounding box respectively with respect to the enveloping grid cell; *c1,…, cn* correspond to *n* classes respectively where n can be chosen based on the classes to be detected in a given image.

4). Intersection Over Union (IOU): The goal of the IOU (a value between 0 and 1) is to discard such grid boxes to only keep those that are relevant. This is because most often than not, a single object in an image can have multiple grid box candidates for prediction, even though not all are relevant. The idea of the IOU is to compare the ratio of the overlapping area to the total combined region of two adjacent boxes given as:

$$IOU = \frac{Overlapping\ \ Region}{Combined\ \ Region} \quad (3)$$

here is the logic behind the IOU is summarized as follows:

(i) The user defines its IOU selection threshold, which can be, for instance, 0.5.

(ii) Then, YOLO computes the IOU of each grid cell, which is the Intersection area divided by the Union Area.

(iii) Finally, it ignores the prediction of the grid cells having an IOU ≤ threshold and considers those with an IOU > threshold.

5). Non-Max Suppression or (NMS): The NMS is used to keep only the boxes with the highest probability detection score. This is because setting a threshold for the IOU is not always enough because an object can have multiple boxes with IOU beyond the threshold, and leaving all those boxes might include noise.

B. 3. The YOLOv3 with DarkNet-53 Algorithm: Implementation Framework

As illustrated in Fig. 2, the basic idea behind YOLOv3 is to divide the input image into a grid of cells and, for each cell, predict the probability of the presence of an object and the bounding box coordinates of the object. The YOLOv3 strategy is based on the architecture of feature extraction model, DarkNet-53. For reducing the detection of objects, 53 layers are stacked together, therefore resulting in a fully convolutional architecture of 106 layers [15]. As shown in Fig. 2, the objects are identified at three layers of the architecture: 82nd, 94th, and 106th [16]. Each layer of the 53-layered architecture is followed by batch normalization layer and the implementation of leaky rectified linear unit (ReLU) activation function. The basic idea of the YOLOv3 architecture is to divide the image into cells of sizes where one grid cell per object is responsible for the object's prediction [16], [60]. The YOLOv3 using DarkNet-53 algorithm can be summarized as follows [11], [15], [16], [18]:

1). Input image is passed through a CNN to extract features from the image;

2). The features are then passed through a series of fully connected layers, which predict probabilities and bounding box coordinates;

3). The image is divided into a grid of cells, and each cell is responsible for predicting a set of bounding boxes and class probabilities;

4). The output of the network is a set of bounding boxes and class probabilities for each cell;

5). The bounding boxes are then filtered using a post-processing algorithm called non-max suppression to remove overlapping boxes and choose the box with the highest probability; and

6). The final output is a set of predicted bounding boxes and class labels for each object in the image.

C. Performance Metrics for Quantitative Benchmarking of Yolov3 with Darknet-53

1). Accuracy: Binary accuracy refers to a performance metric that is typically measured using metrics that

evaluate both the correct classification of objects and the precision of their localization as well as assessing bounding box predictions. Binary accuracy is the ratio of correct predictions to the total number of predictions.

$$\left.\begin{aligned}\text{Accuracy} &= \frac{Number\ of\ correct\ predictions}{Total\ Number\ of\ predictions}\\ &= \frac{TP+TN}{TP+TN+FP+FN}\end{aligned}\right\} \quad (4)$$

2). Precision: Precision is a crucial metric that measures the accuracy of positive predictions made by the model. It quantifies how many of the detected bounding boxes that the model identified as containing an object actually correspond to a real object in the image. Precision is the ratio of true positive predictions to the total predicted positives.

$$\text{Precision} = \frac{TP}{TP+FP} \quad (5)$$

3). Recall (Sensitivity or True Positive Rate): Recall is a single scalar metric that quantifies the proportion of actual positive instances (objects) that were correctly detected by the model. Recall measures the model's ability to identify all relevant objects within an image. Recall is the ratio of true positive predictions to the total actual positives.

$$\text{Recall} = \frac{TP}{TP+FN} \quad (6)$$

where TP (True Positives), TN (True Negatives), FP (False Positives), FN (False Negatives) for precision and recall. These metrics are computed using the predicted outputs ($\hat{y}$) and the actual labels (y) after model training.

4). Precision-Recall (PR) Curve: The precision-recall (PR) curve is a graphical representation that plots the precision values (y-axis) against the recall values (x-axis) at various confidence thresholds for the model's predictions. It shows the trade-off between the two metrics: generally, improving one often comes at the expense of the other.

5). F1-Score: The F1-score is the harmonic mean of precision and recall. The F1-score will be used to provide a balanced measure of model performance. This metric is useful to handle the imbalance in class distribution and is especially relevant for cases where certain classes may dominate the dataset. The F1-score is calculated as:

$$F1\ Score = 2\times\frac{Precision\times Recall}{Precision+Recall} \quad (7)$$

This metric is particularly important in situations where both precision and recall are important, as it balances the trade-off between them.

6). AUC-ROC: The area under receiver operating characteristic (AUC-ROC) curve is mathematically described as the integral of the True Positive Rate (TPR) with respect to the False Positive Rate (FPR) across all possible classification thresholds, and can be interpreted as the probability that a randomly chosen positive instance will be ranked higher than a randomly chosen negative instance. It quantifies the performance of a binary classifier across its entire range of operating points, with values closer to 1 indicating better performance. AUC-ROC can be expressed mathematically as:

$$\left.\begin{aligned}AUC-ROC &= \int_0^1 TPR\left(FPR\right)dFPR\\ &= \int_0^1 TPR\left(FPR^{-1}\left(x\right)\right)dx\end{aligned}\right\} \quad (8)$$

7). Confusion Matrix: A confusion matrix does not have a single formula but is a metric used to summarize the performance of a classification model by showing the actual versus predicted classes. For a two-class problem, it includes four components: True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). These components are then used in formulas to calculate performance metrics such as the Accuracy (as defined in (4)), Precision (Positive Predictive Value, PPV given in (5), Recall (Sensitivity, True Positive Rate defined in (6)), Specificity (True Negative Rate), and the F1 score (given by (7)).

8). Mean Average Precision (mAP): Mean average precision is a metric that averages the precision of a model over all classes. It is calculated by first finding the average precision (AP) for each class (often across a range of recall values) and then averaging the APs together to get mAP. The mean of average precision (mAP) values are calculated over recall values from 0 to 1. mAP formula is based on the following procedures: *1).* Calculate the intersection over union (IoU) ; *2).* Generate the prediction scores using the model; *3).* Convert the prediction scores to class labels; *4).* Calculate the confusion matrix—TP, FP, TN, FN; *5).* Calculate the precision and recall metrics; *6).* Calculate the area under the precision-recall curve; and *7).* Measure the average precision. The mAP is calculated by finding average precision (AP) for each class and then average over a number of classes.

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \quad (9)$$

where AP_i is the AP of class *i*, *i* is the number of

classes and *N* is the total number of classes.

9). *Execution Time:* Execution time (or processing speed) is paramount for the YOLO-based object detection algorithms because it directly enables real-time performance, which is a critical requirement for practical, time-sensitive applications like autonomous vehicles, surveillance, and robotics. Furthermore, execution evaluation is crucial due to its direct impact on real-time application performance and overall system efficiency.

D. Quantitative Benchmarking Criteria for YOLOv3 with DarkNet-53 Model Architecture

In other to evaluate the performance of the proposed deep CNN YOLOv3 with DarkNet-53 pre-trained model architecture, it is imperative to quantitatively compare its performances with two widely used model architectures for object detection and recognition, namely: YOLOv8 with CSPDarkNet-53 (cross stage partial DarkNet-53) [63]–[67] and YOLOv3 with SqueezeNet [67]–[71].

The complete description and treatment of these two pre-trained model architectures can be found in [63]–[71].

D. 1. YOLOv8 with CSPDarkNet-53

YOLOv8 is the latest iteration, offering improvements in performance and efficiency over previous versions [63]–[67]. YOLOv8 typically includes features for better backbone networks, improved loss functions, and optimizations for both training and inference. YOLOv8 uses CSPDarkNet-53 as the backbone for its implementation. This means that the original DarkNet-53 feature extraction part of YOLOv8 has been replaced by CSPDarkNet-53 [65], [67]. The CSPDarkNet-53 is a modified version of the original DarkNet-53 architecture designed for better gradient flow and reduced computational cost. YOLOv8 with CSPDarkNet-53 features a mix of convolutional layers and residual connections, which help in learning more robust features, particularly for object detection tasks [63]–[67]. The backbone of YOLOv8, based on the CSPDarknet-53 architecture, has been modified and enhanced with C2f blocks and a spatial pyramid pooling fusion (SPPF) module at the end. YOLOv8 with the CSPDarknet-53 architecture is designed to efficiently extract features from input images by using cross-stage partial (CSP) connections to improve information flow and gradient propagation. [63]–[71].

D. 2. YOLOv3 with SqueezeNet

SqueezeNet is a convolutional neural network designed to be small and efficient. It introduces “fire modules” which consist of a “squeeze” layer (1x1 convolutions) followed by an “expand” layer (1x1 and 3x3 convolutions) [67]. The squeeze layer reduces the number of input channels to the 3x3 filters in the expand layer, leading to fewer parameters [67]–[67]. SqueezeNet is primarily designed for image classification, with a focus on achieving high accuracy using a very small model size. Although it is mainly used for classification, SqueezeNet can also serve as a lightweight backbone for object detection tasks when integrated with architectures such as YOLOv3. This involves replacing the standard Darknet-53 backbone in YOLOv3 with SqueezeNet for feature extraction, followed by adding the YOLOv3 detection heads on top of the SqueezeNet features. This modification aims to produce a more lightweight and efficient object detector, potentially at the cost of some accuracy compared to a deeper backbone such as Darknet-53 [67]–[71].

Experiments

A. Hardware Setup

A. 1. Setting up the IP Camera with an RJ-45 Cable, an Ethernet Switch and Router

The hardware setup and configuration is as illustrated and discussed in line with the block diagram shown in Fig. 1. The demonstration starts in the laboratory to connect and configure the hardware. All cables used for hardware connections are the RJ-45 cable while all power cable is standard factory fitted cables. Care should be taken to connect the host development computer (laptop) and the Airtel router/Wi-FI module to the appropriate uplink ports on the PoE switch.

The complete hardware setup is shown in Fig. 3 and Fig. 4.

The front view of the powered IP camera connected to the router via an Ethernet switch with an RJ-45 cable is shown in Fig. 3 while the rear view of the same setup is shown in Fig. 4.

Fig. 3: Front view of the IP camera set-up with an RJ-45 cable and a router.

Fig. 4: Rare view of the IP camera set-up with an RJ-45 cable and a router.

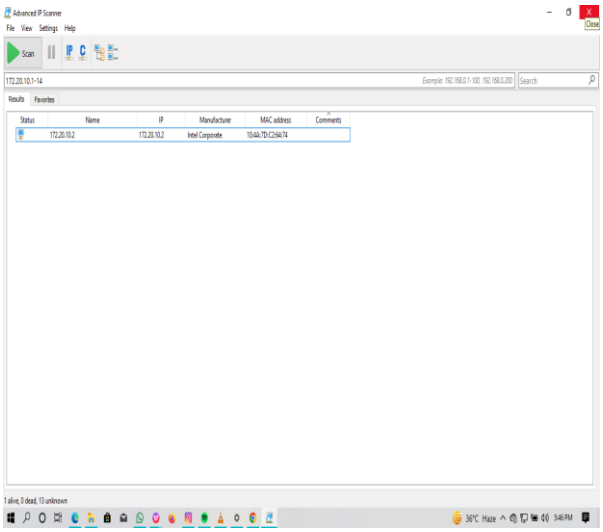


Fig. 5: The IP address of the camera using advanced IP scanner.

After the hardware setup, the camera is powered with electricity via the 12-Volt power adapter (converter) and the camera switches ON as can be seen in Fig. 3 with the red LED ON, plug the power adapter into the camera and a wall outlet. Note that the camera can still be powered via the PoE switch in the absence of public power supply. In this case, the camera is connected to the router using the RJ-45 cables where one end of the RJ-45 cable is plugged into the camera's Ethernet port and the other end of the RJ-45 is plugged into an available Ethernet port on the router as illustrated in Fig. 1.

A. 2. Obtaining the IP Address of the Camera

With proper setup and configuration, the IP-based camera functions as described in the previous subsection. The step is to obtain the IP address of the camera.

Obtaining the IP address of the camera can be achieved in several ways, depending on the model of the camera. However, for the SONY Vision CCTV IP-based SV-5MP-POE-4B, the IP address is obtained as follows:

1). Camera manual: Check the manual for instructions on finding the IP address. It might be printed on a label on the camera itself.

2). Manufacturer's website: Some manufacturers provide tools or software for discovering cameras on a network.

3). Router interface: Log in to your router's web interface and look for a list of connected devices. The camera's IP address should be listed there.

4). Use a Network Scanning Tool: In this study, Advanced IP Scanner is used as the scanning tool to obtain the camera's IP address. This is achieved by running a network scan and identifying the device that matches the camera's manufacturer or model name. The obtained camera's IP address is http://192.168.1.11/ as shown in Fig. 5.

A. 3. Accessing the Camera's Web Interface

Accessing the camera's web interface involves opening a web browser on the host computer and entering the camera's IP address into the address bar and hit the Enter key.

There comes up a prompt for a username and password.

These are provided in the camera's manual or on a sticker on the camera.

By following these steps, the camera's web interface is accessed and the IP camera is successfully set up for monitoring remote immediate and remote locations. It is recommended to always consult the camera's manual for specific instructions and troubleshooting tips. However, configuring of the camera settings is required. After logging into the camera's web interface, the camera's settings are configured as illustrated in Fig. 6 and Fig. 7 as follows:

1). Network settings: You can usually choose between the automatic dynamic host configuration protocol (DHCP) or the static IP address assignment. In this study, the DHCP is selected for online dynamism.

2). Resolution and image quality: Choose the video resolution and quality that best suits your needs and network bandwidth. The 125600 option is selected.

3). Recording settings: Configure motion detection, recording schedules, and storage options are selected.

4). Secure your network: Make sure your router's Wi-Fi network is password-protected to prevent unauthorized access to your camera.

5). Update the camera's firmware: Regularly check for and install firmware updates from the camera manufacturer to ensure optimal performance and security.

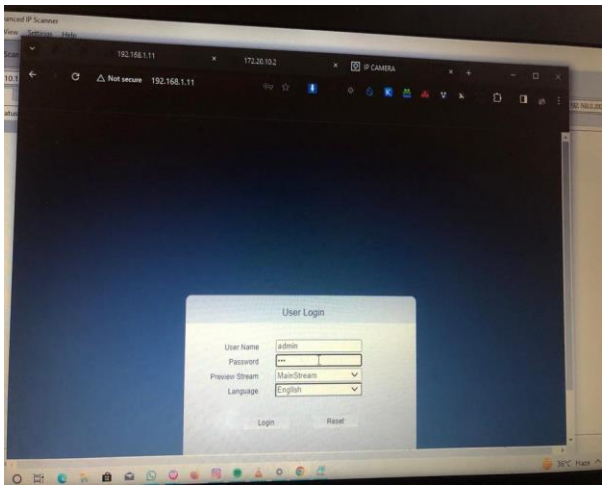


Fig. 6: Accessing the camera's web interface using web browser.

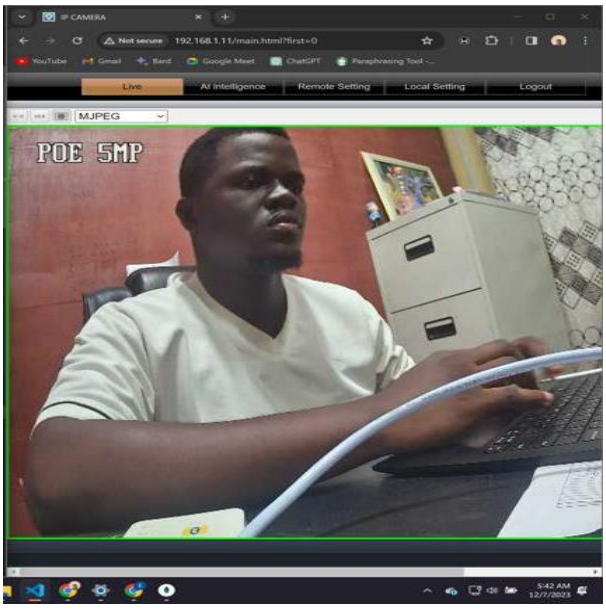


Fig. 7: Accessed the camera's web interface using web browser.

A. 4. Router Configuration

After accessing and configuring the camera's web interface, the next step is to configure the router as follows:

1). Configure IP Camera Settings (Optional):

Some IP cameras may have configuration requirements. Refer to the camera's user manual to find out if there are specific settings or software that need to be configured. This may include setting up an IP address, configuring network settings, or installing any necessary software.

2). Router Configuration (Optional):

In some cases, you might need to access your router's settings to configure port forwarding or other network settings for the IP camera. Refer to your router's user manual for instructions on how to access and configure these settings.

3). Check Connection:

Once everything is connected, check the status of the IP camera. It may have indicator lights that show if it's successfully connected to the network.

4). Streaming Video Verification:

Verify that the camera is streaming correctly by viewing the live feed through the camera's interface or any associated software.

5). Secure the Setup (Optional):

If security is a concern, consider changing default passwords, enabling encryption, and ensuring that the camera's firmware is up to date. Figure 8 shows the setup of the camera and resolution in color mode.

Fig. 8: Testing the Setup and resolution (color Mode).

B. Software Setup

B. 1. Python and Add-On Libraries Installation

The primary software used in this work is Python and its associated add-on libraries listed in sub-section c of this Section.

All the necessary Python and related software required for this work were downloaded from appropriate websites, properly installed, setup and configured accordingly and tested.

B. 2. OpenCV Installation

The OpenCV library for Python was downloaded, installed and properly configured. The OpenCV is essential for working with images and video streams. The complete Python code for the installation, configuration and interfacing of the IP camera to the computer is shown in Fig. 9.

The test result for the successful installation, interfacing and configuration of Python and its associated software for accessing the IP camera is shown in Fig. 10.

```python
import cv2
import numpy as np
import time
import os

# Load the YOLOv3 model
model_cfg_path = r"C:\Users\Tsega\500 Project\yolov3-from-opencv-object-detection\model\cfg\darknet-yolov3.cfg"
model_weights_path = r"C:\Users\Tsega\500 Project\yolov3-from-opencv-object-detection\model\weights\model.weights"
class_names_path = r"C:\Users\Tsega\500 Project\yolov3-from-opencv-object-detection\model\class.names"

# Load class names
with open(class_names_path, 'r') as f:
    class_names = [j[:-1] for j in f.readlines() if len(j) > 2]
    f.close()

# We load the YOLOv3 model
net = cv2.dnn.readNetFromDarknet(model_cfg_path, model_weights_path)

# Create a folder to save recognized plate images
save_folder = 'recognized_plate_images'
if not os.path.exists(save_folder):
    os.makedirs(save_folder)

# # Set the IP camera URL
# video_url = "rtsp://admin:KAD@192.168.1.11:554"

cap = cv2.VideoCapture(0)

ct = 0

# Define the cooldown period (in seconds)
cooldown_period = 10

# Initialize a variable to keep track of the last saved time
last_saved_time = 0

# Inside the loop where you process YOLOv3 results
while True:
    ct += 1

    ret, img = cap.read()
    if not ret:
        break

    imS = img.copy()

    H, W,   = img.shape
```

```python
    blob = cv2.dnn.blobFromImage(img, 1 / 255, (416, 416), (0, 0, 0), True)
    net.setInput(blob)
    output_layer_names = net.getUnconnectedOutLayersNames()
    detections = net.forward(output_layer_names)

    for detection in detections:
        for obj in detection:
            scores = obj[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5 and class_id == 0:  # Assuming class_id 0 is for plates

                # Get the current time
                current_time = time.time()

                # Check if the last saved time is more than the cooldown period
                if current_time - last_saved_time > cooldown_period:
                    # Save the entire frame as an image
                    save_path = os.path.join(save_folder, f'captured_frame_{current_time}.png')
                    cv2.imwrite(save_path, img)

                    # Update the last saved time
                    last_saved_time = current_time

                # Drawing the rectangle around the detection (you can customize this part)
                x, y, w, h = (obj[0:4] * np.array([W, H, W, H])).astype(int)
                cv2.rectangle(imS, (x, y), (x + w, y + h), (0, 255, 0), 2)

                # Add text to the image
                text = "Detected Plate"
                cv2.putText(imS, text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)

    imS = cv2.resize(imS, (720, 600))
    cv2.imshow('Video', imS)

    k = cv2.waitKey(1) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

[82] ✓ 33.6s

Fig. 9: The Python code snippet for interfacing the IP-based camera with the YOLOv3 model.

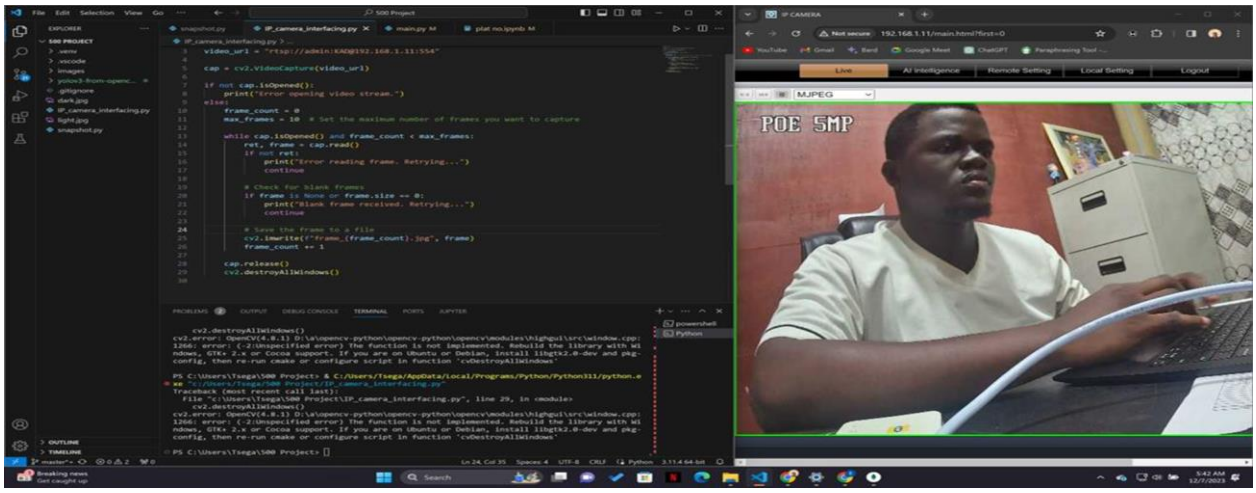


Fig. 10: Python Installation and interfacing with the IP camera.

B. 3. Python Code Implementation and Description

This Python script uses the OpenCV library to perform object detection on video frames using the YOLOv3 using DarkNet-53 model architecture. The specific object that is being looked for is a license plate number. The Python code implementation and description is highlighted in the following:

1). Imports necessary libraries: cv2 for computer vision tasks, numpy for numerical operations, time for time-related functions, and OS for interacting with the operating system.

2). Load the YOLOv3 model: The paths to the configuration file, weights file, and class names file for the YOLOv3 model are defined. The model is then loaded using cv2.dnn.readNetFromDarkNet.

3). Prepares for saving images: A directory is created to save images of detected license plates.

4). Sets up video capture: The script is set up to capture video from the default camera (camera index 0). If you uncomment the lines with *video_url and cap = cv2*. VideoCapture (*video_url*), it would capture video from the specified IP camera instead.

5). Initializes variables: ct is a counter, cooldown_period is a time period during which new detections are ignored to prevent saving multiple images of the same vehicle plate, and last_saved_time keeps track of when the last image was saved.

6). Processes video frames: In an infinite loop, the script reads frames from the video, preprocesses them into a format suitable for the YOLOv3 model, and feeds them into the model to get detections.

7). Handles detections: For each detection, the script checks if the detected object is a license plate (assuming that class ID0 corresponds to license plates) and if the confidence of the detection is above 0.5. If both conditions are met and the time since the last saved image is more than the cooldown period, the script saves the current frame as an image.

8). Draws bounding boxes and text: The script draws a rectangle around the detected license plate and adds a text label.

9). Cooldown Period Check and Frame Saving: The script checks if the time since the last saved frame is greater than a cooldown period (10 seconds in this work). If the cooldown check passes, the entire frame is saved as an image in the '*recognized_plate_images*' folder.

10). Displays the video: The processed video frame is displayed in a window. If the user presses the Esc key, the loop breaks and the video capture is stopped.

11). Cleans up: The video capture is released and all OpenCV windows are destroyed.

B. 4. Bounding Box, Detection and Capturing of VPNs

The project code is a Python script that connects to a PostgreSQL database, fetches unprocessed images and uses the YOLOv3 model to detect objects in the images and/or video streams. The Python script then uses EasyOCR to read text (in this case, LPNs) from the detected vehicles. The Python script also handles updating the database with the processed status, validity, and the detected license plate number. The Python script also updates the Google Cloud Storage. The description of the automated Python script implementation is as follows:

1). Connects to a PostgreSQL database using SQLAlchemy.

2). Loads the YOLOv3 model from the specified paths.

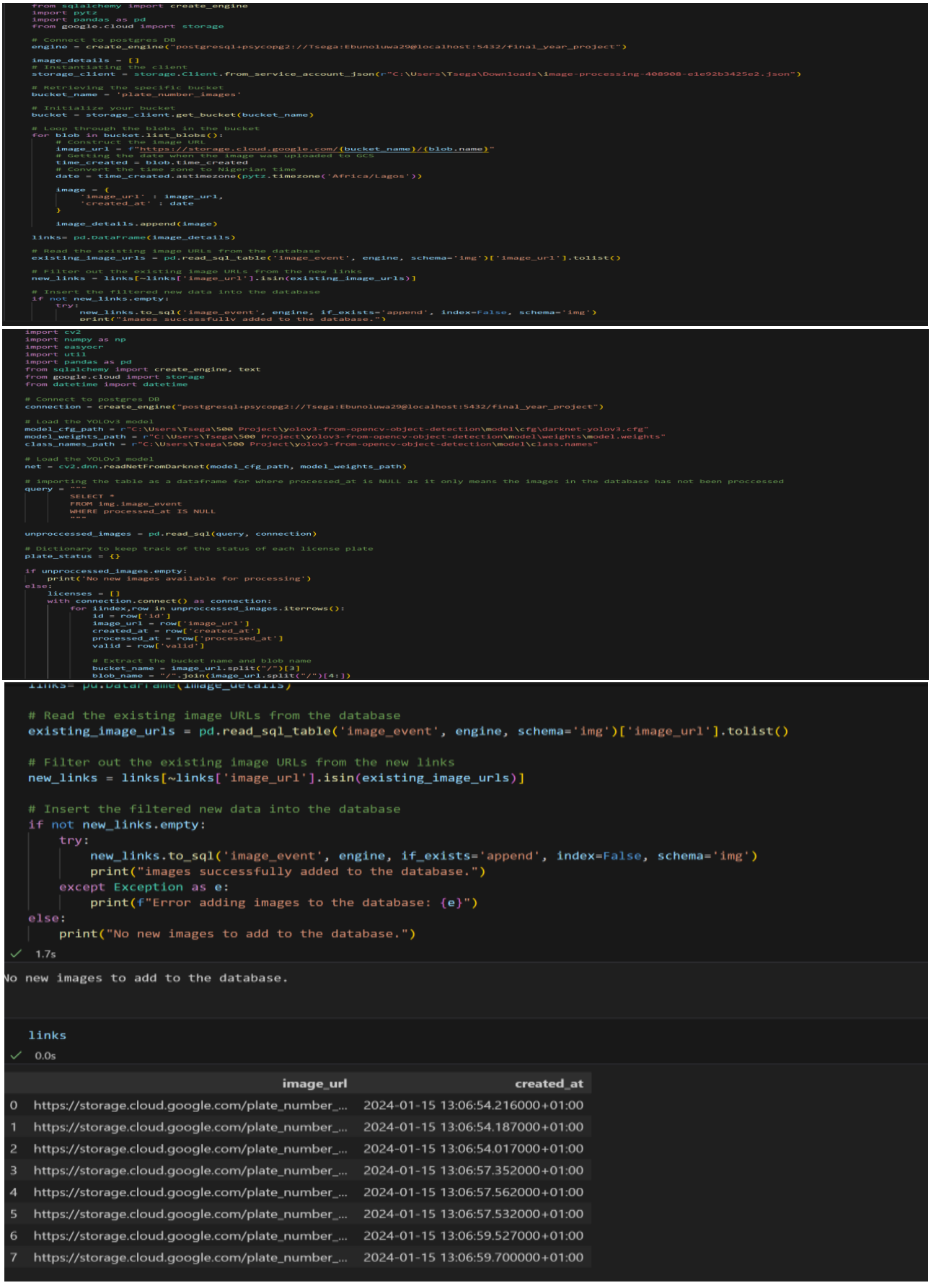
3). Fetches unprocessed images from the database.

4). If there are unprocessed images, it iterates over each image.

5). For each image, it downloads the image from Google Cloud Storage and decodes it into a NumPy array.

6). The image is then processed using the YOLOv3 model to detect objects.

7). If objects are detected, it applies Non-Maximum Suppression to filter out overlapping bounding boxes.



```
from sqlalchemy import create_engine
import pytz
import pandas as pd
from google.cloud import storage

# Connect to postgres DB
engine = create_engine("postgresql+psycopg2://Tsega:Ebunoluwa29@localhost:5432/final_year_project")

image_details = []
# Instantiating the client
storage_client = storage.Client.from_service_account_json(r"C:\Users\Tsega\Downloads\image-processing-408908-e1e92b3425e2.json")

# Retrieving the specific bucket
bucket_name = 'plate_number_images'

# Initialize your bucket
bucket = storage_client.get_bucket(bucket_name)

# Loop through the blobs in the bucket
for blob in bucket.list_blobs():
    # Construct the image URL
    image_url = f"https://storage.cloud.google.com/{bucket_name}/{blob.name}"
    # Getting the date when the image was uploaded to GCS
    time_created = blob.time_created
    # Convert the time zone to Nigerian time
    date = time_created.astimezone(pytz.timezone('Africa/Lagos'))

    image = {
        'image_url' : image_url,
        'created_at' : date
    }

    image_details.append(image)

links= pd.DataFrame(image_details)

# Read the existing image URLs from the database
existing_image_urls = pd.read_sql_table('image_event', engine, schema='img')['image_url'].tolist()

# Filter out the existing image URLs from the new links
new_links = links[~links['image_url'].isin(existing_image_urls)]

# Insert the filtered new data into the database
if not new_links.empty:
    try:
        new_links.to_sql('image_event', engine, if_exists='append', index=False, schema='img')
        print("images successfully added to the database.")
```

```
import cv2
import numpy as np
import easyocr
import util
import pandas as pd
from sqlalchemy import create_engine, text
from google.cloud import storage
from datetime import datetime

# Connect to postgres DB
connection = create_engine("postgresql+psycopg2://Tsega:Ebunoluwa29@localhost:5432/final_year_project")

# Load the YOLOv3 model
model_cfg_path = r"C:\Users\Tsega\500 Project\yolov3-from-opencv-object-detection\model\cfg\darknet-yolov3.cfg"
model_weights_path = r"C:\Users\Tsega\500 Project\yolov3-from-opencv-object-detection\model\weights\model.weights"
class_names_path = r"C:\Users\Tsega\500 Project\yolov3-from-opencv-object-detection\model\class.names"

# Load the YOLOv3 model
net = cv2.dnn.readNetFromDarknet(model_cfg_path, model_weights_path)

# importing the table as a dataframe for where processed_at is NULL as it only means the images in the database has not been proccessed
query = """
        SELECT *
        FROM img.image_event
        WHERE processed_at IS NULL
        """

unproccessed_images = pd.read_sql(query, connection)

# Dictionary to keep track of the status of each license plate
plate_status = {}

if unproccessed_images.empty:
    print('No new images available for processing')
else:
    licenses = []
    with connection.connect() as connection:
        for iindex,row in unproccessed_images.iterrows():
            id = row['id']
            image_url = row['image_url']
            created_at = row['created_at']
            processed_at = row['processed_at']
            valid = row['valid']

            # Extract the bucket name and blob name
            bucket_name = image_url.split("/")[3]
            blob_name = "/".join(image_url.split("/")[4:])
```

```
links= pd.DataFrame(image_details)

# Read the existing image URLs from the database
existing_image_urls = pd.read_sql_table('image_event', engine, schema='img')['image_url'].tolist()

# Filter out the existing image URLs from the new links
new_links = links[~links['image_url'].isin(existing_image_urls)]

# Insert the filtered new data into the database
if not new_links.empty:
    try:
        new_links.to_sql('image_event', engine, if_exists='append', index=False, schema='img')
        print("images successfully added to the database.")
    except Exception as e:
        print(f"Error adding images to the database: {e}")
else:
    print("No new images to add to the database.")
```

✓ 1.7s

```
No new images to add to the database.
```

```
links
```

✓ 0.0s

	image_url	created_at
0	https://storage.cloud.google.com/plate_number_...	2024-01-15 13:06:54.216000+01:00
1	https://storage.cloud.google.com/plate_number_...	2024-01-15 13:06:54.187000+01:00
2	https://storage.cloud.google.com/plate_number_...	2024-01-15 13:06:54.017000+01:00
3	https://storage.cloud.google.com/plate_number_...	2024-01-15 13:06:57.352000+01:00
4	https://storage.cloud.google.com/plate_number_...	2024-01-15 13:06:57.562000+01:00
5	https://storage.cloud.google.com/plate_number_...	2024-01-15 13:06:57.532000+01:00
6	https://storage.cloud.google.com/plate_number_...	2024-01-15 13:06:59.527000+01:00
7	https://storage.cloud.google.com/plate_number_...	2024-01-15 13:06:59.700000+01:00

Fig. 11: The Python code snippet for license plate number segmentation and recognition.

8). For each detected vehicle, it extracts the region of interest (the VPN) and uses EasyOCR to read the text from it.
9). It then checks the database for the latest record of the detected LPN and checks if the cooldown period has passed.
10). If the cooldown period has passed, it updates the status of the LPN (either "IN" or "OUT") and updates the database with the processed status, validity, and the detected LPN.
11). If no objects are detected in the image, it updates the database to reflect that the image was picked but not successfully processed.

It should be noted that the following Python packages were implemented, namely: *cv2, numpy, easyocr, pandas, sqlalchemy, and google.cloud.storage*. The implementation required a pre-trained *YOLOv3* model and a Google Cloud Storage service account *JSON file*.

B. 5. License Plate Recognition and the Description of the Python Code

License Plate Recognition (LPR), also known as Automatic License Plate Recognition (ALPR) is a technology that involves the automatic detection, extraction, and recognition of license plate information from images or video streams. Using Python and various libraries and frameworks, an LPR system is implemented. A brief explanation of the key steps in the implementation of the LPR system using Python and its associated libraries is presented, along with the Python code shown in Fig. 11. The following are the step-by-step breakdown of what the Python script of Fig. 11 does.

1). Connects to a PostgreSQL database using SQLAlchemy.
2). Loads the YOLOv3 model from the specified paths.
3). Fetches unprocessed images from the database.
4). If there are unprocessed images, it iterates over each image.
5). For each image, it downloads the image from Google Cloud Storage and decodes it into a NumPy array.
6). The image is then processed using the YOLOv3 model to detect objects.
7). If objects are detected, it applies Non-Maximum Suppression to filter out overlapping bounding boxes.
8). For each detected object, it extracts the region of interest (the license plate) and uses EasyOCR to read the text from it.
9). It then checks the database for the latest record of the detected license plate and checks if the cooldown period has passed.
10). If the cooldown period has passed, it updates the status of the license plate **(**either **"IN"** or **"OUT")** and updates the database with the processed status, validity, and the detected license plate number.
11). If no objects are detected in the image, it updates the database to reflect that the image was picked but not successfully processed.

B. 6. Database Documentation

The following procedures are employed for the creation of the database documentation:

1). Choose a relational database system (e.g., MySQL, SQLite, PostgreSQL). The PostgreSQL is used in this work.
2). Install the appropriate database driver for Python. Themysql-connector-python for MySQL is used in this work.
3). Create a database and tables to store vehicle information.

This documentation provides insights into the purpose and structure of the database, schema, and table. It helps users, developers, or administrators understand the context and organization of the data within the database with the following descriptions:

i). Database Name: image_processing_db
ii). Schema: img
iii). Purpose: The 'img' schema may be used to organize database objects, such as tables, views, and functions, related to image processing or storage. It provides a logical grouping within the database, helping to organize and manage objects with a common theme.
iv). Table: img.image_event
v). Purpose: The img.image_event table is designed to store information related to events associated with processed images. This includes details such as image URLs, time-stamps of creation and processing, license plate information, and the status of the associated event.

Fig. 12: Database documentation table.

vi). Table Structure: The description for the creation of the database documentation is shown in Fig. 12 and is achieved as follows:

CREATE TABLE IF NOT EXISTSimg.image_event
(
*id*SERIAL PRIMARY KEY,
image_url VARCHAR(255) NOT NULL,
created_at TIME-STAMP WITHOUT TIME ZONE,
processed_at TIME-STAMP WITHOUT TIME ZONE,
valid BOOLEAN,
plate_number VARCHAR(20),
status VARCHAR(5),
CONSTRAINT*image_event_image_url_key UNIQUE (image_url)*
);

Columns

id (SERIAL PRIMARY KEY): A unique identifier for each image event, automatically incremented.

image_url (VARCHAR(255) NOT NULL): The URL or path of the associated image.

created_at (TIMESTAMP WITHOUT TIME ZONE): Time-stamp indicating when the image event was created or when the image was taken by the camera

processed_at (TIMESTAMP WITHOUT TIME ZONE): Time-stamp indicating when the image was processed for extraction of necessary information.

valid (BOOLEAN): Boolean flag indicating the validity of the processed image if true or false.

plate_number (VARCHAR(30)): The detected license plate number from the processed image.

status (VARCHAR(5)): The status of the associated event, typically indicating whether the vehicle is "IN" or "OUT" of the premises.

Constraints

image_event_image_url_key (UNIQUE): Ensures that each image URL is unique, preventing duplicate entries.

B. 7. Database Development, Set-Up, Interaction, and Documentation

The Python code snippets were combined into a cohesive Python script that captures the camera feed, performs license vehicle plate recognition, and updates the database. The direct implementation of the proposed vehicle plate identification system involves: *1).* Simulating the combined cohesive Python code; *2).* Choose a relational database system (e.g., MySQL, SQLite, PostgreSQL); *3).* Install the appropriate database driver for Python (e.g., mysql-connector-python for MySQL); *4).* Create a database and tables to store vehicle information; and *5).* Finally, execute the Python script and observe the system in action. Ensure that the camera captures the feed, and the license plate information is correctly stored in the database.

These step-by-step procedures provide a foundation for creating an artificial intelligent-based automatic vehicle inventory system. Adjustments may be necessary based on specific hardware, software, and database choices. Additionally, consider implementing security measures for data protection and system access. Lastly, connect to the database and insert/update records with the recognized license plate information. The developed database setup, database interaction, and database documentation for the capturing and storing vehicle data is show in Fig. 13.

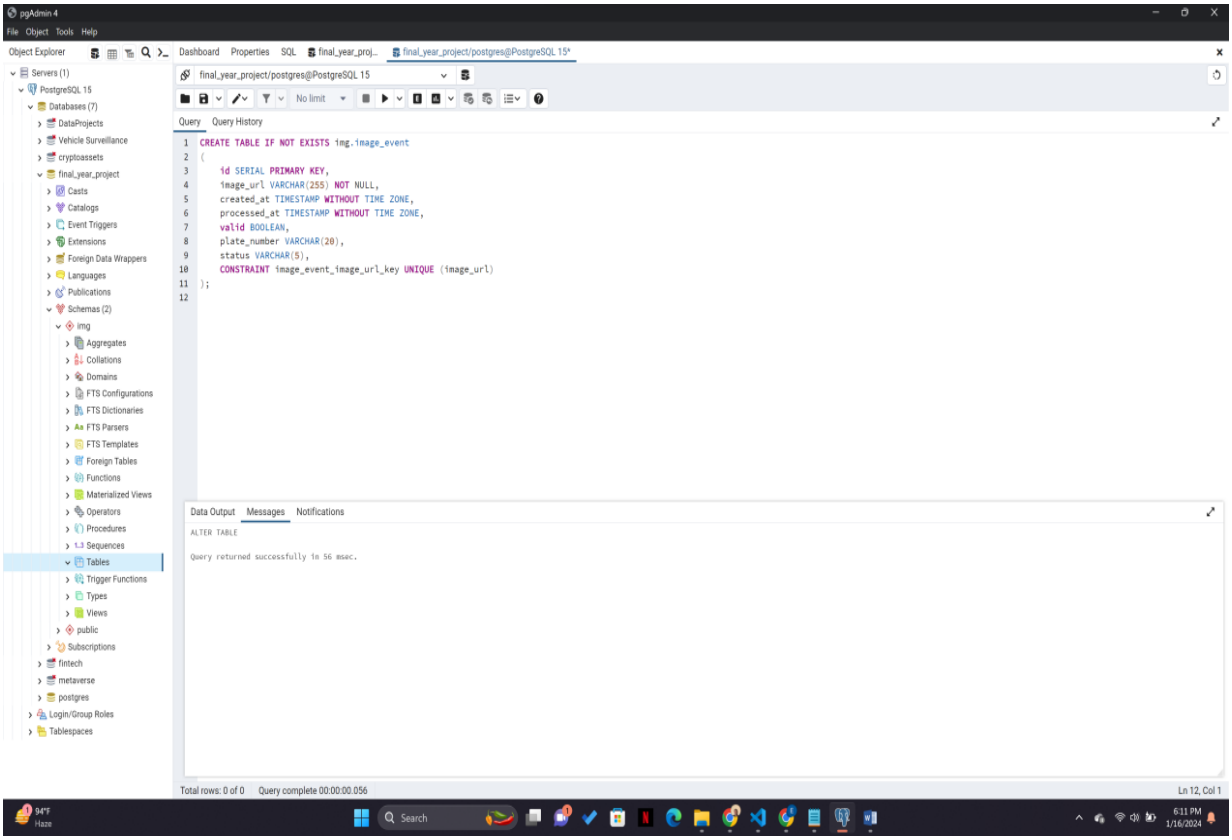


Fig. 13: Database interaction, Database setup and Database documentation.

C. *Real-Time Implementation of the YOLOv3 with DarkNet-53*

C. 1. *Implementation*

The real-time implementation of the deep CNN based on the DarkNet-53 model architecture is illustrated diagrammatically in Fig. 2. The sizes of the input license plate numbers (LPNs) were different and therefore the LPNs were pre-processed and automatically resized. In the LPN recognition problem based on the YOLOv3 with DarkNet-53, the input images were automatically resized to 320 × 320 pixels. All images were automatically normalized using Z-score normalization based on the same mean, variance and standard deviation. The implementation of the deep CNN based on YOLOv3 with DarkNet-53 model architecture is summarized as follows:

1. Input image is passed through a series of convolutional layers that extract features using convolutional filters with appropriate stride and padding techniques.
2. The output of each layer is passed through a rectified linear unit (ReLU) activation function to introduce non-linearity.
3. Pooling layers are used to reduce dimensionality of output and make the model more efficient.
4. The convolutional and pooling layers computations are repeated 5 times at each epoch due to the complexity of the LPN detection and recognition problem.
5. The output of the final convolutional layer is flattened and fed into the fully connected layer (FC Layer) that performs classification on the extracted features from the VPN.
6. The output of the FC layer is passed through a softmax activation function with the associated cost functions to produce the detected and recognized.
7. The implementation of the deep CNN based on the YOLOv3 with DarkNet-53 model architecture for a LPN detection and recognition converged after 80 epochs (iterations) which demonstrate the robustness of the algorithm as shown in Fig. 14.

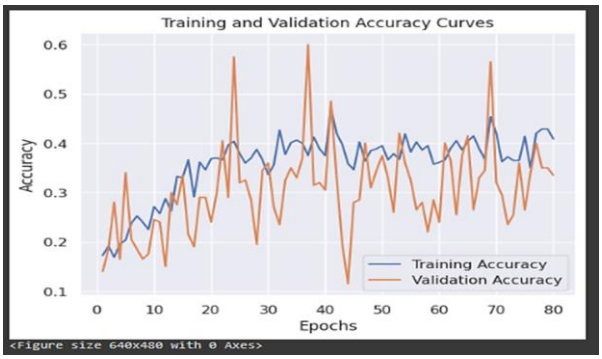


Fig. 14: Training and validation accuracy results over 80 iterations obtained by deep CNN based on YOLOv3 with DarkNet-53 model structure.

C. 2. *Additional Implementation Considerations*

The methodology for developing and implementing the deep CNN based on YOLOv3 with DarkNet-53 for the accurate detection and recognition of LPNs involves additional consideration listed as follows:

1. *Data Preprocessing*: The dataset was cleaned and preprocessed to ensure consistency and quality. This included resizing images, normalizing pixel intensities, and handling missing data, although no missing data were present.
2. *Model Architecture Selection*: In choosing an appropriate deep CNN pre-trained model architecture suitable for real-time LPN detection and recognition tasks, the DarkNet-53 is used because it is the backbone of the YOLOv3 as it balances model complexity, efficiency, performance, and appreciable less computational complexities with low-cost affordable hardware.
3. *Model Training*: The training starts by capturing the LPN via streaming video of the moving vehicle and loading the DarkNet-53 pre-trained model architecture. The model training proceeds by adding additional layers specifically a global average pooling layer, which reduces the spatial dimensions of the output and prepares the YOLOv3 using DarkNet-53 for detection and recognition and a softmax layer with a sigmoid activation function. The model is then compiled using the Adam optimizer and a custom loss function. The custom loss function, *get_weighted_loss*, incorporates class weights to address class imbalance in the captured image dataset. These weights are learned from an image dataset and serve as a starting point for the fine-tuning process. During training, the deep CNN based on YOLOv3 with DarkNet-53 learns to recognize patterns and features indicative of the presence or absence LPNs.
4. *Hyper-Parameter Tuning*: The training process for YOLOv3 with DarkNet-53 involves tuning several parameters, including the learning rate, regularization, batch size, training epochs, number of layers, and filters in the DarkNet-53 architecture of Fig. 2 as well as other parameters as summarized in Table 1. These hyper-parameters significantly impact the performance of the trained model and fine-tune the model's hyper-parameters to improve its performance. The detailed hyper-parameters used for tuning the deep CNN based on YOLOv3 with DarkNet-53 are listed in Table 1 [72]–[75].
5. *Model Evaluation*: The performance of the trained model was assessed during training using hyperparameter tuning. Various performance metrics, such as accuracy, precision, and the area under the receiver operating characteristic curve (AUC-ROC),

were evaluated to measure the model's effectiveness in accurately detecting and recognizing LPNs. The model assigns a probability score to each detected and recognized LPN (e.g., probability of being normal and probability of being abnormal).

6. *Iterative Refinement*: The model was iteratively refined based on evaluation results and feedback. This involved revisiting the model architecture and adjusting hyperparameters to address specific challenges and limitations identified during the LPN detection and recognition process.

Table 1: Hyper-parameter training and tuning options

S/N	Parameter and Layer Type		Specification
1	Network Architecture		YOLOv3
2	imageInputLayer		[32 32 1]
3	Convolution2dLayer		[5, 20]
4	fullyConnectedLayer		14
5	Learning rate		[1e-4 1e-3]
6	Batch size		64
7	Input size	Width	320 320 x 320
		Height	320
8	Anchors		[10, 13, 16, 30, 33, 23, 30, 62, 45, 59, 119, 116, 90, 156, 198, 373, 326]
9	Number of anchors		3
10	Number of classes		80
11	Confidence threshold		0.25
12	NMS Threshold		0.45
13	IOU threshold		0.5
14	Number of filters		64
15	Number of layers		53
16	Activation function		Leaky ReLU
17	Optimizer		Adam
18	Maximum epochs		80
19	maxPooling2dLayer		2
20	Stride		[32, 16, 8]
21	Scale		[1, 2, 3]
22	Dropout Rates		0.5
23	L2Regularization		[1e-8 1e-2]
24	Momentum		[0.5 0.99]
25	Sequence padding Direction		Left
26	Padding		2
27	Shuffle		Every epoch
28	Metrics		Accuracy
29	Gradient Threshold	Algorithm	1
30	Loss function		Cross entropy
31	Data augmentation		[True, False]

Results and Discussion

A. Results

A. 1. Quantitative Results

The YOLOv3 with DarkNet-53 model is designed for the detection and recognition of license plate numbers of vehicles using deep learning techniques. The architecture used is DarkNet-53, a deep CNN pre-trained model that has demonstrated strong performance in various computer vision tasks, including object detection and recognition. Each layer receives input not only from the previous layer but also from all preceding layers, which encourages feature reuse and improves information flow through the network. This model is trained on a large dataset of labeled classes, where each image is associated with the presence or absence of specific objects. The loss function used during training guides the model to minimize the difference between predicted and actual label classes. The training process involves adjusting the model's parameters to minimize the difference between predicted and actual labels. For each captured license plate number (LPN) image, the model generates predictions for the presence or absence of the LPN, assigning a probability to each class. These probabilities represent the model's confidence in its predictions. The final layer of the DarkNet-53 model consists of neurons corresponding to the detected LPN classes. The extracted features are then fed into a trained classification model. Based on these features and the internal logic of the model, the LPN is identified for its intended use. The results are initially evaluated with associated probabilities. Each probability represents the model's estimate of the likelihood of LPN detection and recognition. A threshold is applied to convert probabilities into predictions (presence or absence of a LPN). For example, if the threshold is 0.5, probabilities above 0.5 are considered positive predictions. The training and validation result for the accuracy curve is shown in Fig. 14. Due to space economy, only the results for first five LPN detection and recognition have been presented in this paper together with their respective performance comparison metrics figures and tables.

The real-time implementation results for the three deep CNN based on YOLOv3 with DarkNet-53, YOLOv8 with CSPDarkNet-53 and YOLOv3 with SqueezeNet model architectures for the detection and recognition of the LPNs for all the 25 vehicles used in this work are given in Figs. 15–18 which shows the detected and recognized LPNs; Tables 2–5 show the database tables; Figs. 19–23 show the precision-recall (PR) and area under receiver operating characteristic (AUC-ROC) curves; and Figs. 24–28 show the confusion matrices percentage accuracies. The three deep CNN pre-trained model architectures have been qualitatively and quantitatively evaluated in terms of several performance metrics for comparison and the results are shown in Tables 6–8 where the tables show the confusion matrix minimum and maximum values of the percentage successes of detected versus target LPN detection and recognition recorded by the confusion matrices; the performance metrics comparison by the three deep CNN pre-train models; and the average values of all the performance metrics respectively.

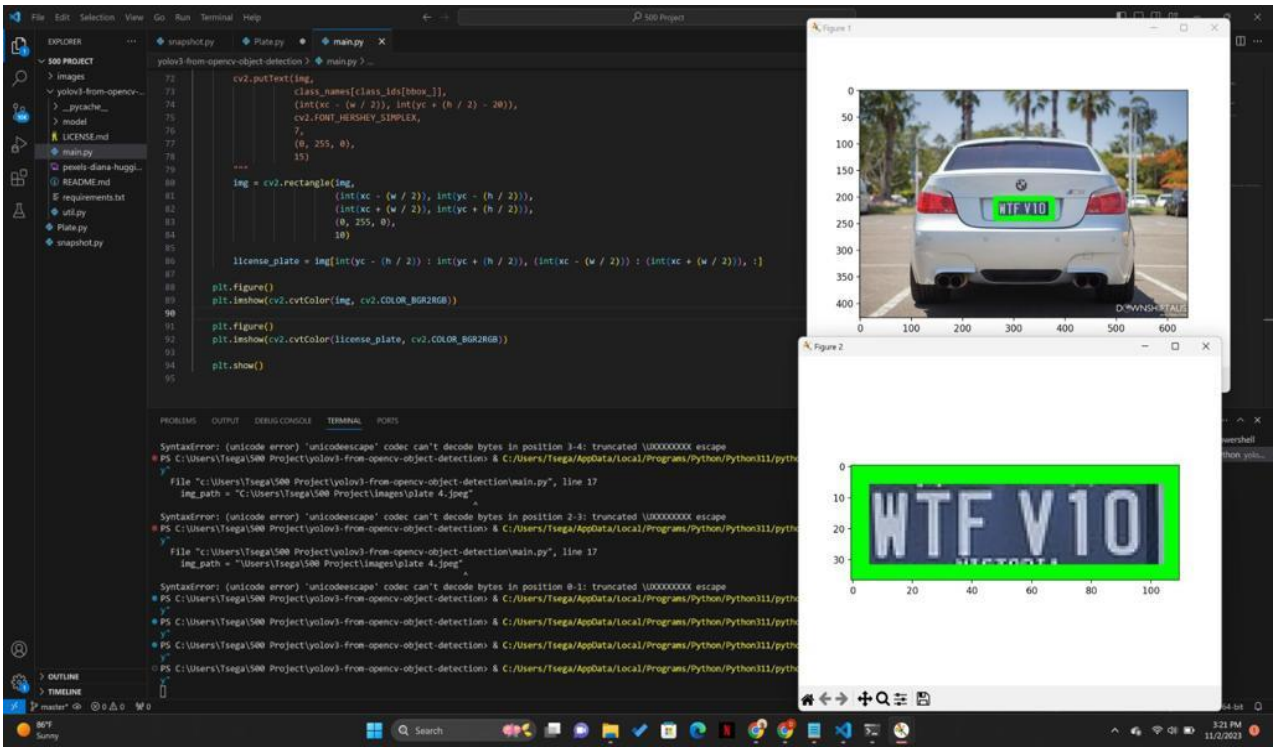


Fig. 15: Image processing of vehicle with plate number WTF V10: plate number detection, segmentation and capturing with time stamps.

Fig. 16: Image processing of vehicle with plate number HR268U0375: plate number detection, segmentation and capturing with time stamps.

Fig. 17: Image processing of vehicle with plate number KL 25B 2001: plate number detection, segmentation and capturing with time stamps.

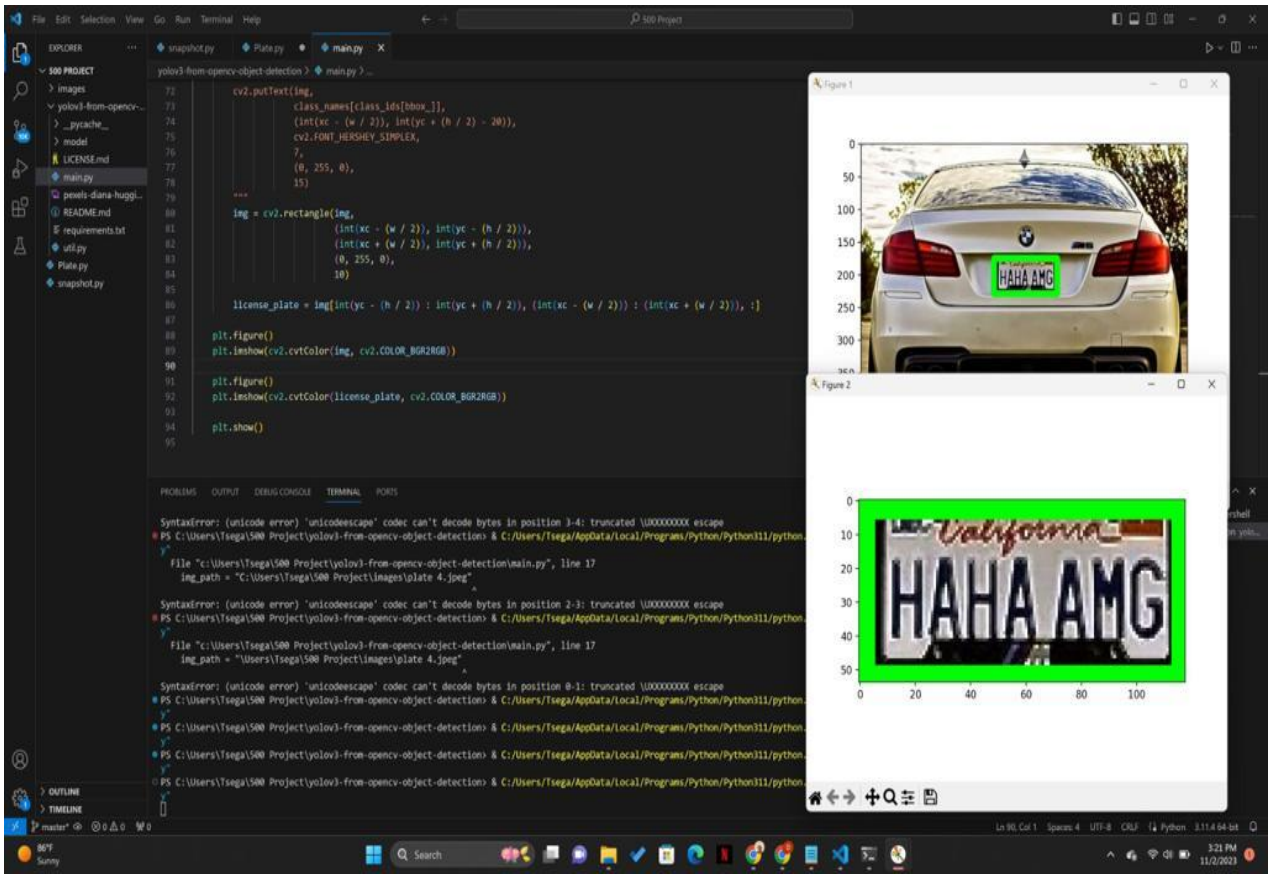


Fig. 18: Image processing of vehicle with plate number HAHA AMG: plate number detection, segmentation and capturing with time stamps.

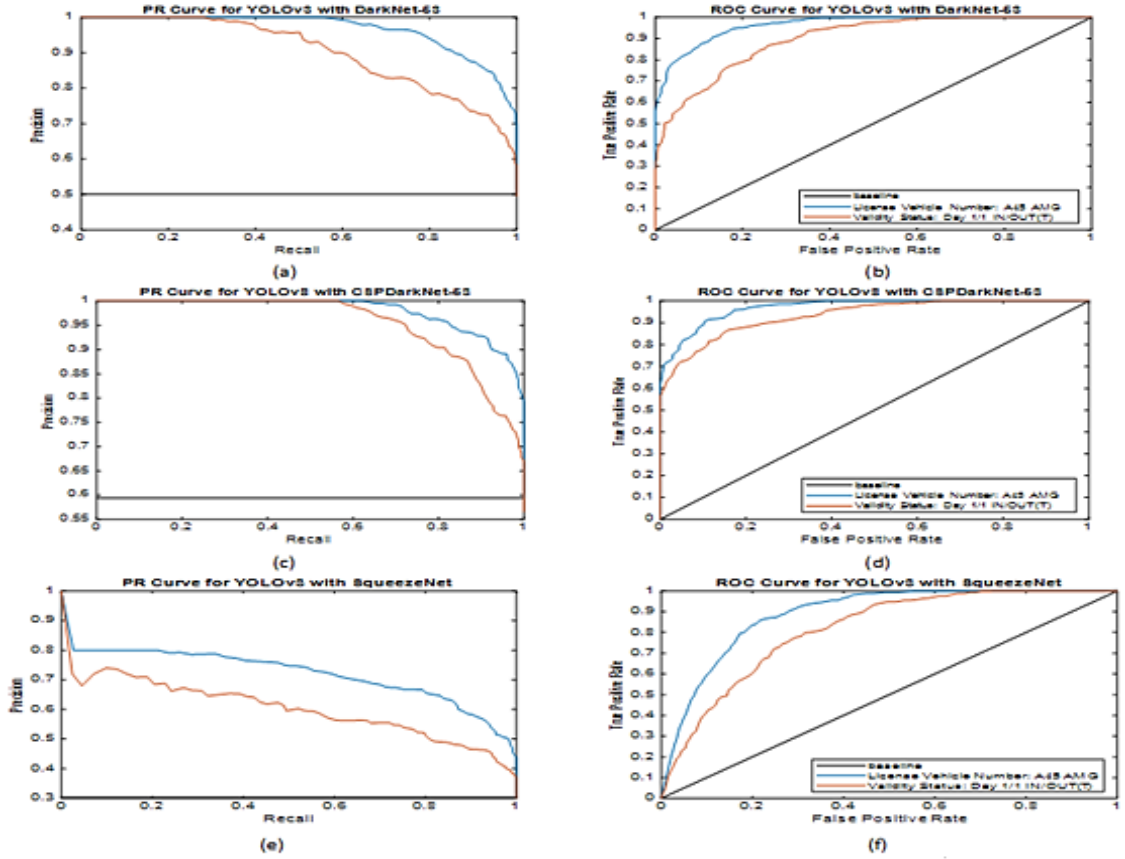


Fig. 19: PR and ROC curves for LPN A45 AMG by: YOLOv3 using DarkNet-53 (a) and (b); YOLOv8 using CSPDarknet-53 (c) and (d); and YOLOv3 using SqueezeNet (e) and (f) respectively.

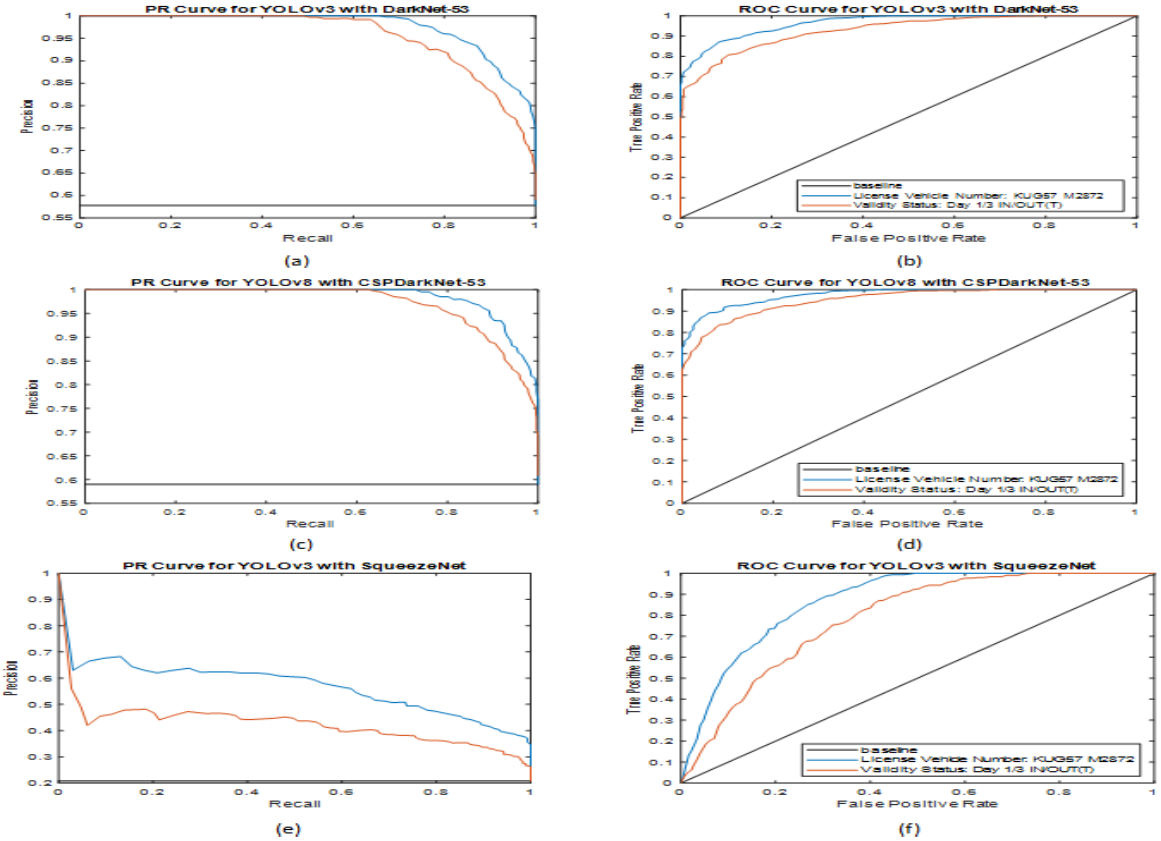


Fig. 20: PR and ROC curves for LPN KUG57 M2872 by: YOLOv3 using DarkNet-53 (a) and (b); YOLOv8 using CSPDarknet-53 (c) and (d); and YOLOv3 using SqueezeNet (e) and (f) respectively.

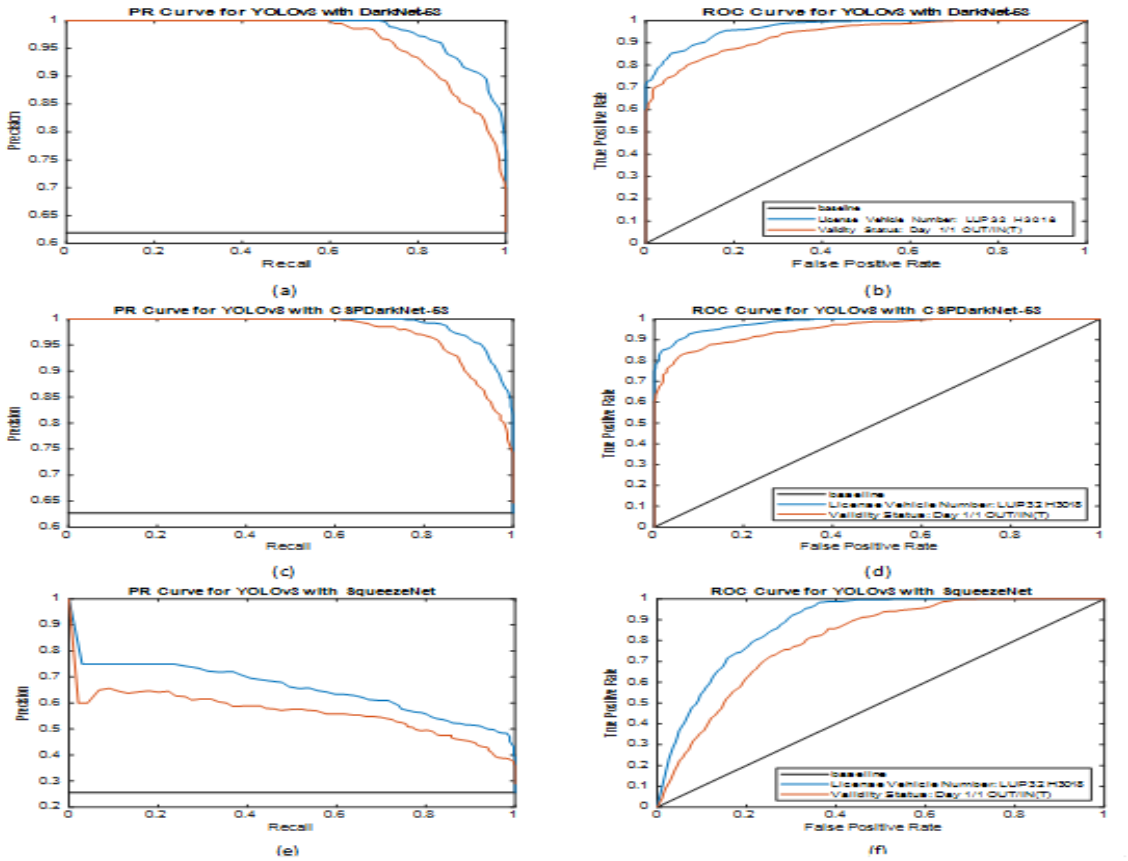


Fig. 21: PR and ROC curves for LPN LUP32 H3018 by: YOLOv3 using DarkNet-53 (a) and (b); YOLOv8 using CSPDarknet-53 (c) and (d); and YOLOv3 using SqueezeNet (e) and (f) respectively.

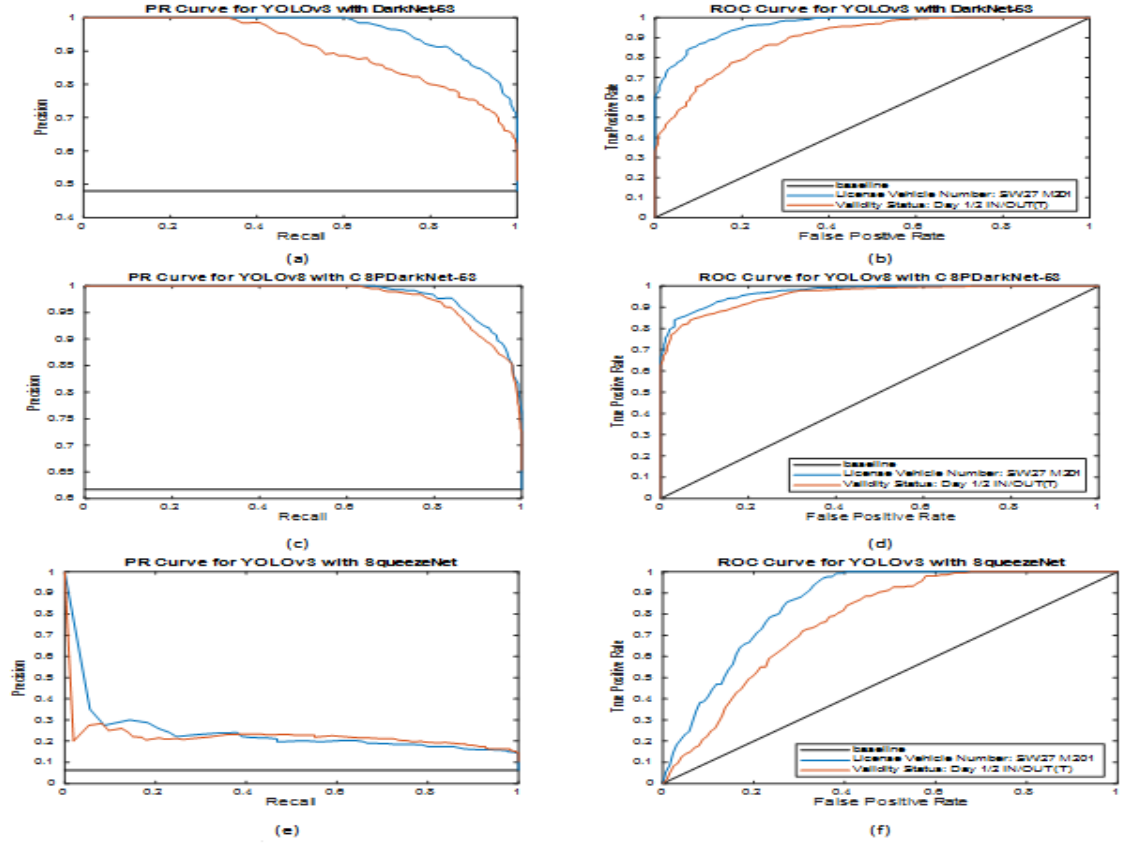


Fig. 22: PR and ROC curves for LPN SW27 M201 by: YOLOv3 with DarkNet-53 (a) and (b); YOLOv8 with CSPDarknet-53 (c) and (d); and YOLOv3 with SqueezeNet (e) and (f) respectively.

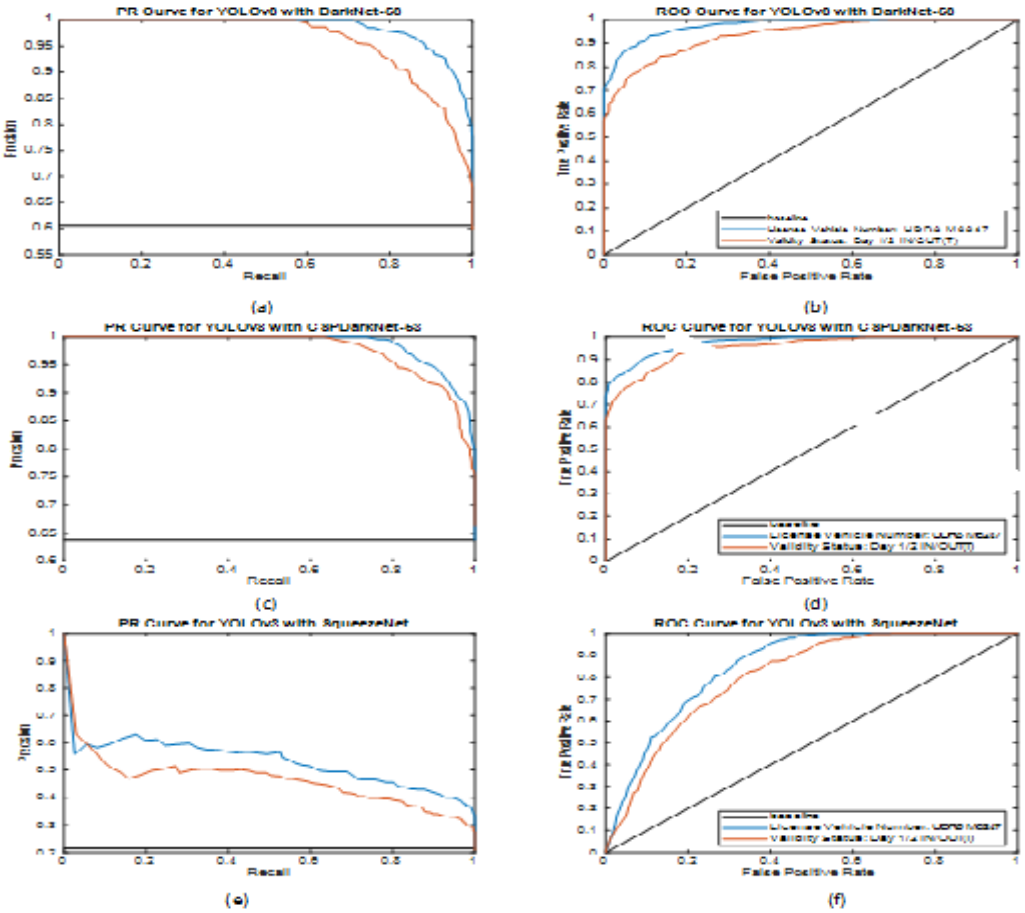


Fig. 23: PR and ROC curves for LPN UDR8 M6247 by: YOLOv3 with DarkNet-53 (a) and (b); YOLOv8 with CSPDarknet-53 (c) and (d); and YOLOv3 with SqueezeNet (e) and (f) respectively.

A. 2. Discussion of the Quantitative Results

The results obtained from the implementation and deployment of the AVIS together with the dynamic RDBS based on YOLOv3 with DarkNet-53 algorithm are shown in Fig. 15 to Fig. 18 as well as in Table 2 to Table 5. The databases shown in Table 2 to Table 4 are arranged in columns as follows: *1).* The first column is configured to record captured LPN serially at the point of entry and exit for up to 100 vehicles before resetting to serial number 1; *2).*

The second column records the Serial Integer Xi (where *i* is an integer that varies from 1 to 50 in accordance to the number of vehicle entry and exits based on captured plate numbers); *3).*The third column (*Image_url*) is the uniform resource locator (URL) webpage link that is automatically created by the PostgreSQL database for storing vehicle plate numbers with relevant information on Google cloud; *4)*. The date and time-stamp when the plate number is captured and the database is created is shown in the fourth column; *5)*. The date and time-stamps when the plate number is processed and stored in the database are shown in the fifth column; *6).*

The validity of the vehicle whether entry (False or True) as a Boolean variable (sixth column); 7*).* The seventh column (Plate Number) captures the actual vehicle plate number with unique ID Serial Integer (Xi); and *8).* The Status of the vehicle whether entering (IN) or exiting (OUT) as captured in the eight column. Note that once a vehicle originally in the company premises exits (OUT), it Boolean status is False and only changes to True when the vehicle returns (IN) into the company. Similarly, once a new vehicle enters (IN) the company premises, its Boolean validity status will be False and can only change to True when the vehicle exits (OUT). Furthermore, note that a vehicle is assigned a unique serial integer once captured whether at the point of entry (IN) or exit (OUT); and the same unique serial integer is used to verify when the vehicle exit (OUT) or enter (IN) the company's premises respectively. By comparing the time-stamp *created_at* and *processed_at*, it can be seen that the average time lag is 3 seconds; which implies that the database processing time is averagely 3 seconds between when the database is created and information stored for a particular captured vehicle license plate number (LPN) is updated.

Table 2: Database of vehicle inventory showing vehicle plate numbers, entry status (IN/OUT), processing time, time in/time out for Day 1 (18th March, 2024)

S/N	ID Serial Integer	Image_url	Created_at timestamp without time zone	Processed_at timestamp without time zone	Valid boolean	Plate_number character varying (100)	Status character string ($)
1	X1	http://storage.cloud.google.com/plate_images/A45AMG_952/21.1.jpg	2024-03-18 10:12:42.952	2024-03-18 10:12:46.691655	False	A45 AMG	IN
2	X2	http://storage.cloud.google.com/plate_images/KUG57M2872_245/11.3.jpg	2024-03-18 10:42:50.245	2024-03-18 10:42:53.731457	False	KUG57 M2872	IN
3	X3	http://storage.cloud.google.com/plate_images/LUP32H3018_691/17.4.jpg	2024-03-18 10:53:17.691	2024-03-18 10:53:20.540126	False	LUP32 H3018	OUT
4	X4	http://storage.cloud.google.com/plate_images/KAMEI_435/21.7.jpg	2024-03-18 11:14:50.435	2024-03-18 11:14:54.011789	False	KAMEI	IN
5	X5	http://storage.cloud.google.com/plate_images/SW27M201_435/19.2.jpg	2024-03-18 11:22:55.908	2024-03-18 11:22:58.155067	False	SW27 M201	IN
6	X1	http://storage.cloud.google.com/plate_images_A45AMG_567/32.6.jpg	2024-03-18 11:43:22.567	2024-03-18 11:43:26.539536	True	A45 AMG	OUT
7	X4	http://storage.cloud.google.com/plate_images_KAMEI_847/91.5.jpg	2024-03-18 11:54:13.847	2024-03-18 11:54:17.214581	True	KAMEI	OUT
8	X6	http://storage.cloud.google.com/plate_images/UDR8M6247_137/61.4.jpg	2024-03-18 12:06:43.137	2024-03-18 12:06:47.265734	False	UDR8 M6247	IN
9	X7	http://storage.cloud.google.com/plate_images/KL63E954_455/35.2.jpg	2024-03-18 12:26:34.455	2024-03-18 12:26:37.452863	False	KLS 3E 954	IN
10	X8	http://storage.cloud.google.com/plate_images/KL541H367_352/17.3.jpg	2024-03-18 12:38:44.352	2024-03-18 12:38:47.834242	False	KL 54H 369	IN
11	X9	http://storage.cloud.google.com/plate_images/WTG50KE042_406/82.4.jpg	2024-03-18 12:50:43.406	2024-03-18 12:50:43.641238	False	WTG50 KE042	IN
12	X10	http://storage.cloud.google.com/plate_images/KL55R2473_436/37.3.jpg	2024-03-18 01:10:34.436	2024-03-18 01:10:37.521894	False	KL 55R 2473	IN
13	X3	http://storage.cloud.google.com/plate_images/LUP32H3018_124/37.6.jpg	2024-03-18 01:33:43.124	2024-03-18 01:33:46.103958	True	LUP32 H3018	IN
14	X11	http://storage.cloud.google.com/plate_images/WB74X7603_728/42.5.jpg	2024-03-18 01:46:15.728	2024-03-18 01:46:18.594206	False	WB 74X 7603	IN
15	X12	http://storage.cloud.google.com/plate_images/VNE40L1533_155/71.8.jpg	2024-03-18 01:55:56.155	2024-03-18 01:55:59..621548	False	VNE40 L1533	OUT

B. Discussion of Results

B. 1. Discussion of the Quantitative Results

The real-time implementation results are analyzed qualitatively to assess the performances of three pre-trained model architectures using multiple performance metrics discussed previously which include (i) accuracy, (ii) precision, (iii) recall, (iv) precision recall (PR), (v) F1-score, (vi) area under receiver operating characteristic curve (AUC-ROC), (vii) confusion matrix, (iii) mean average precision (mAP), and (ix) execution time. All the results discussed in this section are presented in their respective figures and tables.

1). Accuracy: Accuracy is an important metric in deep learning that shows the proportion of correctly predicted samples over the total number of samples according to (4). As it can be seen in the second row of Table 8, the YOLOv8 with CSPDarkNet-53 has the best accuracy of 99.8572% followed by YOLOv3 using DarkNet-53 and YOLOv3 with SqueezeNet with an average accuracy of 99.7640% and 91.4080% respectively.

2). Precision: Precision is another important metric which shows the proportion of correct detection and recognition of the LPN. Precision quantifies the accuracy of positive predictions.

Table 3: Database of vehicle inventory showing vehicle plate numbers, entry status (IN/OUT), processing time, time in/time out for Day 2 (19th March, 2024)

S/N	ID Serial Integer	Image_url	Created_at timestamp without time zone	Processed_at timestamp without time zone	Valid boolean	Plate_number character varying (100)	Status character string ($)
16	X13	http://storage.cloud.google.com/plate_images/ZRP38K2041_851/82.5.jpg	2024-03-19 10:08:25.851	2024-03-19 10:08:28.534855	False	ZRP38 K2041	IN
17	X14	http://storage.cloud.google.com/plate_images/RJ27TC0530_501/31.4.jpg	2024-03-19 10:28:37.501	2024-03-19 10:28:40.292431	False	RJ27 TC0530	IN
18	X15	http://storage.cloud.google.com/plate_images/HR260U0375_035/71.1.jpg	2024-03-19 10:48:58.035	2024-03-19 10:49:02.117239	False	HR26BU0375	IN
19	X5	http://storage.cloud.google.com/plate_images/SW27M201_634/22.6.jpg	2024-03-19 10:52:35.634	2024-03-19 10:52:38.223064	True	SW27 M201	OUT
20	X16	http://storage.cloud.google.com/plate_images/YJP21Z542_2131/76.2.jpg	2024-03-19 11:06:27.213	2024-03-19 11:06:30.761524	False	YJP 21Z 542	OUT
21	X17	http://storage.cloud.google.com/plate_images/WPA27X6437_914/52.4.jpg	2024-03-19 11:18:35.914	2024-03-19 11:18:38.425567	False	WPA27 X6437	OUT
22	X6	http://storage.cloud.google.com/plate_images/UDR8M6247_807/10.8.jpg	2024-03-19 11:32:14.807	2024-03-19 11:32:18.390182	True	UDR8 M6247	OUT
23	X18	http://storage.cloud.google.com/plate_images/KL25B2001_011/76.4.jpg	2024-03-19 11:54:28.011	2024-03-19 11:54:31.426713	False	KL 25 B 2001	IN
24	X19	http://storage.cloud.google.com/plate_images/KA51M2143_532/31.4.jpg	2024-03-19 12:08:39.532	2024-03-19 12:08:15.431810	False	KA 51M 2143	IN
25	X20	http://storage.cloud.google.com/plate_images/WTFV103_664/82.7.jpg	2024-03-19 12:23:56.319	2024-03-19 12:23:59.060542	False	WTF V10	OUT
26	X12	http://storage.cloud.google.com/plate_images/VNE40L1533_107/20.8.jpg	2024-03-19 12:30:19.107	2024-03-19 12:30:22.002587	True	VNE40 L1533	IN
27	X9	http://storage.cloud.google.com/plate_images/WTG50KE042_209/91.7.jpg	2024-03-19 12:43:27.209	2024-03-19 12:43:30.870913	True	WTG50 KE042	OUT
28	X21	http://storage.cloud.google.com/plate_images/HAHAAMG_651/38.4.jpg	2024-03-19 12:58:07.419	2024-03-19 12:58:10.139457	False	HAHA AMG	IN
29	X7	http://storage.cloud.google.com/plate_images/KLS3E954_705/30.3.jpg	2024-03-19 01:14:40.705	2024-03-19 01:14:45.307933	True	KLS 3E 954	OUT
30	X16	http://storage.cloud.google.com/plate_images/YJP21Z542_514/32.0.jpg	2024-03-19 01:28:47.514	2024-03-19 01:28:50.223305	True	YJP 21Z 542	IN
31	X8	http://storage.cloud.google.com/plate_images/KL54H369_632/54.6.jpg	2024-03-19 01:42:50.632	2024-03-19 01:42:53.055446	True	KL 54H 369	OUT
32	X22	http://storage.cloud.google.com/plate_images/ABJ97G3318_414/32.7.jpg	2024-03-19 01:50:25.414	2024-03-19 01:50:28.743216	False	ABJ97 G3318	OUT
33	X20	http://storage.cloud.google.com/plate_images/WTFV103_159/30.4.jpg	2024-03-19 01:57:09.327	2024-03-19 01:57:12.654127	True	WTF V10	IN

In terms of precision, the YOLOv8 with CSPDarkNet-53 slightly outperforms the YOLOv3 with CSPDarkNet-53 while the YOLOv3 with DarkNet-53 outperforms the YOLOv3 using SqueezeNet as evident in the third row of Table 8 with precisions of 99.8656%, 99.7908% and 91.6472% respectively.

3). Recall: Recall, also known as sensitivity or the true positive rate, is a crucial evaluation metric in object detection models such as those considered in this work. Recall quantifies the model's ability to correctly identify all relevant instances of a particular object class within a given image dataset.

Table 4: Database of vehicle inventory showing vehicle plate numbers, entry status (IN/OUT), processing time, time in/time out for Day 3 (20th March, 2024)

S/N	ID Serial Integer	Image_url	Created_at timestamp without time zone	Processed_at timestamp without time zone	Valid boolean	Plate_number character varying (100)	Status character string ($)
34	X10	http://storage.cloud.google.com/plate_images/KL55R2473_316/11.6.jpg	2024-03-20 10:13:38.316	2024-03-20 01:13:41.161115	True	KL 55R 2473	OUT
35	X23	http://storage.cloud.google.com/plate_images/AGL67P3516_317/58.7.jpg	2024-03-20 10:33:45.317	2024-03-20 10:33:48.785421	False	AGL67 P3516	IN
36	X11	http://storage.cloud.google.com/plate_images/WB74X7603_838/30.2.jpg	2024-03-20 11:13:51.838	2024-03-20 11:13:54.230089	True	WB 74X 7603	OUT
37	X14	http://storage.cloud.google.com/plate_images/RJ27TC0530_164/81.4.jpg	2024-03-20 11:43:49.164	2024-03-20 11:43:52.418535	True	RJ27 TC0530	OUT
38	X2	http://storage.cloud.google.com/plate_images/KUG57M2872_057/53.6.jpg	2024-03-20 11:57:27.057	2024-03-20 11:57:30.653204	True	KUG57 M2872	OUT
39	X21	http://storage.cloud.google.com/plate_images/HAHAAMG_309/32.7.jpg	2024-03-19 12:08:13.207	2024-03-19 12:08:16.276248	True	HAHA AMG	OUT
40	X24	http://storage.cloud.google.com/plate_images/BUP841C740_318/46.1.jpg	2024-03-20 12:17:27.318	2024-03-20 12:17:30.016472	False	BUP841 C740	IN
41	X25	http://storage.cloud.google.com/plate_images/AAZ64EK24_924/34.2.jpg	2024-03-20 12:39:30.924	2024-03-20 12:39:33.103246	False	AAZ64 EK24	IN
42	X15	http://storage.cloud.google.com/plate_images/HR260U0375_203/42.7.jpg	2024-03-20 12:43:52.203	2024-03-20 12:43:55.740717	True	HR26BU0375	OUT
43	X18	http://storage.cloud.google.com/plate_images/KL25B2001_172/39.4.jpg	2024-03-20 01:10:53.172	2024-03-20 01:10:56.794933	True	KL 25 B 2001	OUT
44	X24	http://storage.cloud.google.com/plate_images/BUP841C740_363/29.7.jpg	2024-03-20 01:23:32.363	2024-03-20 01:23:35.279215	True	BUP841 C740	OUT
45	X19	http://storage.cloud.google.com/plate_images/KA51M2143_152/49.7.jpg	2024-03-20 01:37:53.152	2024-03-20 01:37:56.479095	True	KA 51M 2143	OUT
46	X23	http://storage.cloud.google.com/plate_images/AGL67P3516_416/28.6.jpg	2024-03-20 01:49:24.416	2024-03-20 01:49:27.169782	True	AGL67 P3516	OUT

Table 5: Summary of vehicle entry and exit status with days and validity status

S/N	ID Serial Integer	Plate Number	Day1 Status (18/03/2024)		Day 2 Status (19/03/2024)		Day 3 Status (20/03/2024)		Validity Status
1	X1	A45 AMG	IN (F)	OUT (T)					True
2	X2	KUG57 M2872	IN (F)				OUT (T)		True
3	X3	LUP32 H3018	OUT (F)	IN (T)					True
4	X4	KAMEI	IN (F)	OUT (T)					True
5	X5	SW27 M201	IN (F)			OUT (T)			True
6	X6	UDR8 M6247	IN (F)			OUT (T)			True
7	X7	KLS 3E 954	IN (F)			OUT (T)			True
8	X8	KL 54H 369	IN (F)			OUT (T)			True
9	X9	WTG50 KE042	IN (F)			OUT (T)			True
10	X10	KL 55R 2473	IN (F)				OUT (T)		True
11	X11	WB 74X 7603	IN (F)				OUT (T)		True
12	X12	VNE40 L1533	OUT (F)			IN (T)			True
13	X13	ZRP38 K0241			IN (F)				**False**
14	X14	RJ27 TC0530			IN (F)		OUT (T)		True
15	X15	HR26BU0375			IN (F)		OUT (T)		True
16	X16	YJP21Z 542			OUT (F)	IN (T)			True
17	X17	WPA27 X6437			OUT (F)				**False**
18	X18	KL 25 B 2001			IN (F)		OUT (T)		True
19	X19	KA 51M 2143			IN (F)		OUT (T)		True
20	X20	WTF V10			OUT (F)	IN (T)			True
21	X21	HAHA AMG			IN (F)		OUT (T)		True
22	X22	ABJ97 G3318			OUT (F)				**False**
23	X23	AGL67 P3516					IN (F)	OUT (T)	True
24	X24	BUP841 C740					IN (F)	OUT (T)	True
25	X25	AAZ64 EK24					IN (F)		**False**

Table 6: Confusion matrix showing minimum and maximum percentage of correct detections for target LPNs using YOLOv3 with DarkNet-53, YOLOv8 using CSPDarkNet-53, and YOLOv3 using SqueezeNet

S/N	ID Serial Integer	License Plate Number	Validity Status	% Success by YOLOv3 with DarkNet-53		% Success by YOLOv8 with CSPDarkNet-53		% Success by YOLOv3 with SqueezeNet	
				min	max	min	max	min	max
1	X1	A45 AMG	Status: Day 1/1 IN/OUT (T)	68.4	91.2	71.4	95.2	60.0	80.0
2	X2	KUG57 M2872	Status: Day 1/3 IN/OUT (T)	69.0	92.0	71.9	95.8	58.8	78.4
3	X3	LUP32 H3018	Status: Day 1/1 OUT/IN (T)	70.8	94.4	72.3	96.4	59.5	79.2
4	X4	KAMEI	Status: Day 1 IN (F)	72.3	92.1	71.2	95.4	56.7	77.4
5	X5	SW27 M201	Status: Day 1/2 IN/OUT (T)	67.8	90.4	71.4	95.2	55.8	74.4
6	X6	UDR8 M6247	Status: Day 1/2 IN/OUT (T)	70.2	93.6	73.0	97.4	57.6	76.8
7	X7	KLS 3E 954	Status: Day 1/2 IN/OUT (T)	69.6	94.3	71.2	96.3	58.7	79.5
8	X8	KL 54H 369	Status: Day 1/2 IN/OUT (T)	68.5	93.8	72.1	96.5	58.8	78.4
9	X9	WTG50 KE042	Status: Day 1/2 IN/OUT (T)	70.2	94.6	71.3	97.5	57.6	77.3
10	X10	KL 55R 2473	Status: Day 1/3 IN/OUT (T)	70.1	92.4	72.2	95.1	57.5	77.3
11	X11	WB 74X 7603	Status: Day 1/3 IN/OUT (T)	70.4	93.3	72.2	95.4	57.3	78.2
12	X12	VNE40 L1533	Status: Day 1/2 OUT/IN (T)	69.6	94.2	72.1	96.3	56.4	78.5
13	X13	ZRP38 K0241	Status: Day 2 IN (F)	68.5	94.8	71.6	96.2	58.2	79.6
14	X14	RJ27 TC0530	Status: Day 2/3 IN/OUT (T)	67.7	91.4	71.7	96.8	59.1	74.4
15	X15	HR26BU0375	Status: Day 2/3 IN/OUT (T)	71.3	92.3	73.4	95.7	56.5	75.2
16	X16	YJP21Z 542	Status: Day 2/2 OUT/IN (T)	70.2	94.3	72.6	95.6	60.4	75.1
17	X17	WPA27 X6437	Status: Day 2 OUT (F)	68.2	93.2	71.8	95.5	57.8	76.3
18	X18	KL 25 B 2001	Status: Day 2/3 IN/OUT (T)	67.3	94.2	73.4	96.4	58.7	74.5
19	X19	KA 51M 2143	Status: Day 2/3 IN/OUT (T)	68.1	91.9	72.5	95.6	58.6	74.6
20	X20	WTF V10	Status: Day 2/2 OUT/IN (T)	68.6	92.6	72.3	96.7	56.5	75.4
21	X21	HAHA AMG	Status: Day 2/3 IN/OUT (T)	71.5	92.7	72.2	97.6	60.4	76.5
22	X22	ABJ97 G3318	Status: Day 2 OUT (F)	69.7	93.8	71.2	95.2	57.3	75.2
23	X23	AGL67 P3516	Status: Day 3/3 IN/OUT (T)	69.8	94.9	73.1	95.4	56.2	77.1
24	X24	BUP841 C740	Status: Day 3/3 IN/OUT (T)	69.5	94.8	72.5	96.3	58.1	78.2
25	X25	AAZ64 EK24	Status: Day 3 IN (F)	72.4	93.7	71.4	95.3	60.5	79.6

Based on the recall performance evaluation, it can be seen in the fourth row of Table 8 that the YOLOv8 with CSPDarkNet-53 and YOLOv3 with DarkNet-53 perform well with average recall values of 99.8828% and 99.8164% respectively while YOLOv3 with SqueezeNet has 91.7112%.

4). *Precision-recall (PR):* PR curves are a common way to visualize the trade-off between precision and recall at various confidence thresholds. The precision-recall (PR) score measures how well the model correctly captures positive cases. It is the ratio of the true positives to the sum of the true positives and false negatives. As shown in the fifth row of Table 8, the deep CNN based on the YOLOv8 with CSPDarkNet-53 model architecture achieved an outstanding PR value of 0.9044 when compared to the 0.8852 and 0.6044 obtained using the YOLOv3 with DarkNet-53 and YOLOv3 with SqueezeNet-53 models respectively. This extremely high PR rates emphasize the effectiveness of the YOLOv8 with CSPDarkNet-53 and YOLOv3 with DarkNet-53 models in identifying positive cases correctly, hence reducing false negatives. These results follow from the PR curves of Figs. 19–23 for all the vehicles and the computation of the respective average values from the 7th column of Table 7 are given in the fifth row of Table 8.

5). *The F1-Score:* The F1-score represents the harmonic mean of precision and precision recall. In terms of the F1-score, the deep CNN based on the YOLOv8 with CSPDarkNet-53 model architecture shows superior performance which is closely followed by YOLOv3 with DarkNet-53 and the less performance YOLOv3 with SqueezeNet model architecture with an F1-score of 99.9012%, 99.8544% and 91.7680% respectively from the sixth row of Table 8. Thus, the first two models signify balanced performances in F1-scores.

Table 7: Performance comparison metrics for the deep CNN based on YOLOv3 with DarkNet-53, YOLOv8 with CSPDarkNet-53, and YOLOv3 with SqueezeNet

S/N	License Plate Number	YOLO Model Architectures	Performance Metrics								
			Accuracy (%)	Precision (%)	Recall (%)	Precision Recall (PR)	F1-Score (%)	AUC-ROC (%)	Confusion Matrix Accuracy (%)	Mean Average Precision (mAP) (%)	Execution Time (s)
1	A45 AMG	Yv3D	99.81	99.74	99.91	0.85	99.84	99.90	99.91	98.91	154.46
		Yv8C	99.92	99.86	99.94	0.87	99.90	99.92	99.94	99.51	1658.53
		Yv3S	90.24	91.21	91.67	0.54	91.46	91.76	91.76	87.60	97.55
2	KUG57 M2872	Yv3D	99.75	99.80	99.80	0.82	99.88	99.84	99.90	99.21	167.23
		Yv8C	99.86	99.87	99.88	0.87	99.92	99.91	99.93	99.60	1768.33
		Yv3S	91.72	91.69	90.98	0.68	91.96	91.93	92.89	86.70	101.34
3	LUP32 H3018	Yv3D	99.83	99.88	99.63	0.88	99.77	99.93	99.90	99.42	184.61
		Yv8C	99.92	99.92	99.78	0.89	99.86	99.95	99.94	99.71	1678.52
		Yv3S	89.86	90.61	91.74	0.53	91.98	91.89	91.97	86.91	110.45
4	KAMEI	Yv3D	99.55	99.90	99.85	0.89	99.75	99.90	99.92	99.38	185.41
		Yv8C	99.78	99.94	99.92	0.91	99.86	99.93	99.94	99.62	1757.12
		Yv3S	90.34	91.72	92.79	0.69	91.89	91.87	90.98	85.54	116.44
5	SW27 M201	Yv3D	99.62	99.61	99.75	0.88	99.90	99.92	99.93	98.82	183.46
		Yv8C	99.87	99.73	99.83	0.89	99.94	99.94	99.96	99.51	1778.32
		Yv3S	89.95	92.88	93.92	0.56	91.86	91.82	89.85	84.90	109.57
6	UDR8 M6247	Yv3D	99.81	99.89	99.89	0.91	99.87	99.91	99.90	99.30	174.08
		Yv8C	99.83	99.92	99.92	0.93	99.92	99.93	99.95	99.80	1768.30
		Yv3S	90.87	90.39	91.87	0.72	91.99	91.97	90.88	85.80	111.45
7	KLS 3E 954	Yv3D	99.64	99.77	99.77	0.87	99.90	99.92	99.92	98.89	165.38
		Yv8C	99.78	99.84	99.86	0.90	99.93	99.95	99.94	99.78	1698.54
		Yv3S	91.35	89.88	92.79	0.55	91.91	91.90	91.91	85.74	108.29
8	KL 54H 369	Yv3D	99.65	99.87	99.80	0.89	99.87	99.93	99.92	98.82	171.09
		Yv8C	99.79	99.90	99.88	0.91	99.92	99.96	99.94	99.69	1632.28
		Yv3S	93.49	91.93	91.97	0.63	91.83	91.79	91.67	85.87	116.52
9	WTG50 KE042	Yv3D	99.71	99.69	99.77	0.90	99.88	99.92	99.90	99.48	158.34
		Yv8C	99.82	99.70	99.88	0.93	99.90	99.95	99.93	99.58	1617.50
		Yv3S	92.68	91.97	90.69	0.61	91.94	91.90	91.69	84.98	94.35
10	KL 55R 2473	Yv3D	99.77	99.80	99.76	0.89	99.86	99.90	99.91	99.77	161.33
		Yv8C	99.86	99.88	99.87	0.92	99.90	99.93	99.94	99.92	1568.42
		Yv3S	91.69	90.97	89.98	0.57	91.79	91.86	90.87	86.76	107.32
11	WB 74X 7603	Yv3D	99.83	99.68	99.90	0.90	99.89	99.92	99.92	99.47	159.21
		Yv8C	99.86	99.78	99.93	0.92	99.91	99.94	99.94	99.85	1534.50
		Yv3S	90.85	91.98	90.98	0.64	91.83	91.88	91.85	86.81	103.25
12	VNE40 L1533	Yv3D	99.66	99.76	99.89	0.88	99.77	99.91	99.91	99.74	184.14
		Yv8C	99.74	99.89	99.93	0.91	99.88	99.93	99.94	99.87	1652.09
		Yv3S	91.94	92.97	91.86	0.68	91.94	91.94	91.79	87.54	127.55
13	ZRP38 K0241	Yv3D	99.67	99.76	99.76	0.91	99.87	99.93	99.92	98.67	193.18
		Yv8C	99.87	99.85	99.85	0.92	99.92	99.96	99.95	99.85	1612.16
		Yv3S	91.81	91.96	91.82	0.66	91.99	91.89	92.99	86.39	137.39
14	RJ27 TC0530	Yv3D	99.84	99.74	99.90	0.89	99.89	99.93	99.94	98.59	201.43
		Yv8C	99.88	99.85	99.93	0.90	99.93	99.96	99.96	99.81	1719.22
		Yv3S	90.93	89.89	91.99	0.53	91.85	91.97	91.97	85.87	153.42
15	HR26BU0375	Yv3D	99.69	99.66	99.84	0.87	99.79	99.92	99.90	99.69	204.46
		Yv8C	99.73	99.78	99.89	0.89	99.86	99.94	99.93	99.78	1699.38
		Yv3S	91.99	91.89	89.90	0.61	90.86	91.91	91.88	87.62	167.54
16	YJP21Z 542	Yv3D	99.85	99.88	99.77	0.89	99.69	99.90	99.94	98.99	167.50
		Yv8C	99.90	99.94	99.87	0.91	99.78	99.93	99.96	99.69	1690.19
		Yv3S	92.97	90.89	90.69	0.58	91.87	91.87	91.90	85.86	104.40
17	WPA27 X6437	Yv3D	99.89	99.80	99.59	0.88	99.84	99.93	99.91	99.58	198.33
		Yv8C	99.96	99.87	99.64	0.90	99.89	99.95	99.93	99.67	1640.36
		Yv3S	91.67	91.88	91.89	0.61	91.95	91.97	90.79	84.98	136.46
18	KL 25 B 2001	Yv3D	99.78	99.83	99.88	0.90	99.88	99.92	99.88	98.98	176.35
		Yv8C	99.87	99.92	99.92	0.91	99.93	99.94	99.91	99.68	1597.42
		Yv3S	89.76	92.84	93.96	0.57	91.94	91.69	91.93	86.56	117.04
19	KA 51M 2143	Yv3D	99.88	99.89	99.79	0.89	99.83	99.90	99.90	99.45	201.31
		Yv8C	99.91	99.92	99.85	0.90	99.87	99.93	99.93	99.74	1600.55
		Yv3S	90.96	91.98	92.89	0.63	90.94	91.99	91.89	87.48	147.44
20	WTF V10	Yv3D	99.69	99.74	99.90	0.88	99.97	99.91	99.92	99.53	153.09
		Yv8C	99.78	99.87	99.94	0.90	99.89	99.93	99.95	99.76	1549.54
		Yv3S	91.99	91.97	90.98	0.59	92.93	91.97	90.68	86.95	92.55
21	HAHA AMG	Yv3D	99.79	99.80	99.90	0.87	99.88	99.92	99.92	99.61	166.27
		Yv8C	99.86	99.86	99.95	0.89	99.92	99.95	99.94	99.76	1642.16
		Yv3S	92.91	90.91	91.80	0.52	92.97	91.87	91.91	86.91	105.22
22	ABJ97 G3318	Yv3D	99.88	99.87	99.80	0.89	99.90	99.92	99.90	98.87	184.51
		Yv8C	99.92	99.91	99.88	0.91	99.93	99.95	99.93	99.72	1597.40
		Yv3S	91.58	92.98	89.97	0.53	90.98	91.77	89.89	86.87	126.34
23	AGL67 P3516	Yv3D	99.87	99.73	99.89	0.90	99.89	99.92	99.93	98.88	181.27
		Yv8C	99.92	99.86	99.93	0.91	99.92	99.95	99.96	99.67	1576.33
		Yv3S	92.88	92.88	92.86	0.58	91.86	91.82	91.90	86.88	125.55
24	BUP841 C740	Yv3D	99.86	99.88	99.77	0.91	99.88	99.91	99.94	99.65	177.43
		Yv8C	99.89	99.90	99.86	0.92	99.93	99.93	99.96	99.74	1596.32
		Yv3S	90.89	90.94	91.91	0.63	91.69	91.67	90.89	87.51	118.34
25	AAZ64 EK24	Yv3D	99.78	99.80	99.90	0.89	99.87	99.90	99.92	99.54	203.14
		Yv8C	99.91	99.88	99.94	0.90	99.92	99.93	99.94	99.66	1620.52
		Yv3S	89.88	91.97	90.88	0.67	89.99	91.89	91.96	85.43	149.44

Key: Yv3D = YOLOv3 with DarkNet-53, Yv8C = YOLOv8 with CSPDarkNet-53, Yv3S = YOLOv3 with SqueezeNet

Table 8: Average values of the performance metrics for YOLOv3 with DarkNet-53, YOLOv8 with CSPDarkNet-53, and YOLOv3 with SqueezeNet model architectures

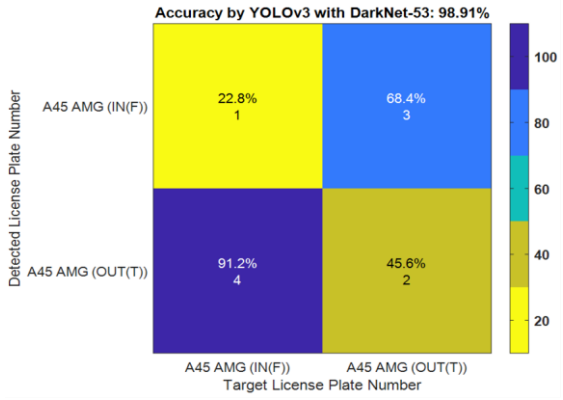
S/N	Performance Metrics		YOLOv3 with DarkNet-53	YOLOv8 with CSPDarkNet-53	YOLOv3 with SqueezeNet
1	Accuracy (%)		99.7640	99.8572	91.4080
2	Precision (%)		99.7908	99.8656	91.6472
3	Recall (%)		99.8164	99.8828	91.7112
4	Precision-Recall (PR)		0.8852	0.9044	0.6044
5	F1-Score (%)		99.8544	99.9012	91.7680
6	AUC-ROC (%)		99.9124	99.9396	91.8716
7	Confusion Matrix Accuracy (%)		99.2496	99.7188	86.4184
8	Confusion matrix percentage successes in detected versus target LPN from Table C1 (%)	min	69.5880	72.0800	58.1200
		max	93.2360	96.0320	77.0840
9	Mean Average Precision (mAP) (%)		99.9144	99.9416	91.5476
10	Execution Time (s)		178.2804 (2.9713 mins)	1650.20 (4.5839 hrs)	119.4084 (1.9901 mins)

6). *AUC-ROC:* The Area Under the Receiver Operating Characteristic Curve (AUC-ROC) evaluates the performance of a model across all possible detection and recognition threshold, balancing sensitivity and specificity. The AUC-ROC results follow from the AUC-ROC curves shown in Figs. 19–23 and their respective numerical percentages are shown in the seventh row of Table 8 obtained from the ninth column of Table 7, where the deep CNN based on YOLOv8 with CSPDarkNet-53 and YOLOv3 with DarkNet-53 model architectures achieved the very high AUC-ROC scores across all LPN detection and recognition scenarios, boasting a near-perfect score of 99.9396% and 99.9124% respectively when compared to YOLOv3 with SqueezeNet with an average AUC-ROC score of 91.8716%. These results are further justified by the PR and ROC curves shown in Figs. 19–23 where YOLOv8 with CSPDarkNet-53 shows very high threshold values followed by YOLOv3 with DarkNet-53 model architectures.

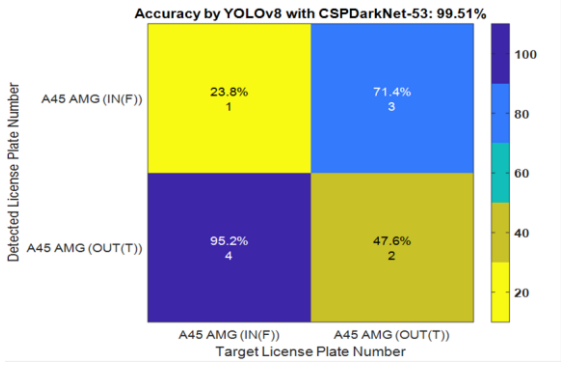
7). *Confusion Matrix:* The confusion matrix is a performance evaluation metric that visualizes the model's predictions against the ground truth, showing counts of true positives, false positives, and false negatives for each class. It helps assess how well the model is performing, identifies specific class-based errors, and provides a visual basis for calculating metrics like precision and recall. The confusion matrix typically considers thresholds for confidence and Intersection over Union (IoU) to classify predictions. The confusion matrices for the deep CNN based on the YOLOv8 with CSPDarkNet-53, YOLOv3 with DarkNet-53 and YOLOv3 with SqueezeNet model architectures are shown respectively in Figs. 24–28 for all the five LPN shown in this paper due to space economy.

It is obvious in all the five figures that the very percentage accuracies obtained by the YOLOv8 with CSPDarkNet-53 model exhibits robust performance followed by the YOLOv3 with DarkNet-53 model when compared to the lower performance recoded by YOLOv3 with SqueezeNet model. The percentage accuracies obtained from the confusion matrix for all the 25 vehicles considered in this work are shown in the tenth column of Table 7 for the three models. From Table 6 and Table 8, the average confusion matrix percentage accuracies obtained by the YOLOv8 with CSPDarkNet-53, YOLOv3 with DarkNet-53 and YOLOv3 with SqueezeNet models are 99.7188%, 99.2496% and 86.4148% respectively as given in the eighth row of Table 8. The results show that YOLOv8 with CSPDarkNet-53 recorded the highest average accuracy followed closely by YOLOv3 with DarkNet-53 with a lower average accuracy recorded by YOLOv3 with SqueezeNet model.

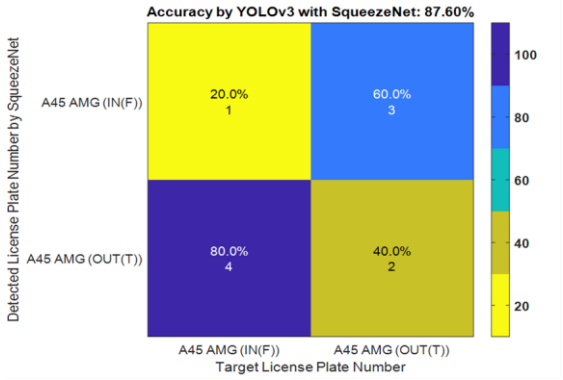
Furthermore, the minimum and maximum percentage successes in the detected versus target LPN of all 25 vehicles based on the confusion matrix are shown in Table 7 while their respective average values are shown in the ninth row of Table 8 for all the three models. From Table 8, the average respective minimum and maximum percentage successes of 72.0800% and 96.0320% obtained by the YOLOv8 with CSPDarkNet-53 model shows robust performance which is closely followed by an appreciable result of the YOLOv3 with DarkNet-53 model with average minimum and maximum percentage successes of 69.5880% and 93.2360% respectively when compared to the lower performance of 58.1200% and 77.0840% recoded by YOLOv3 with SqueezeNet model.



(a)

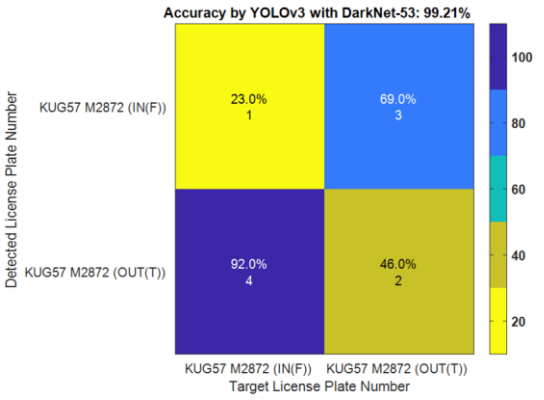


(b)

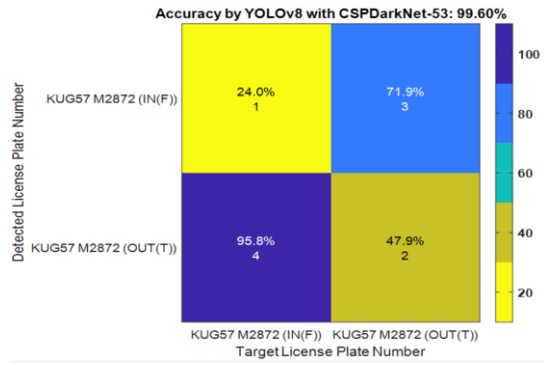


(c)

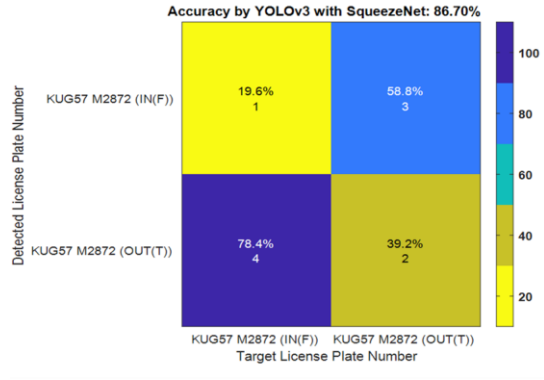
Fig. 24: Confusion matrix accuracy for LPN A45 AMG by: (a) YOLOv3 using DarkNet-53; (b) YOLOv8 using CSPDarknet-53; and (c) YOLOv3 using SqueezeNet.



(a)



(b)



(c)

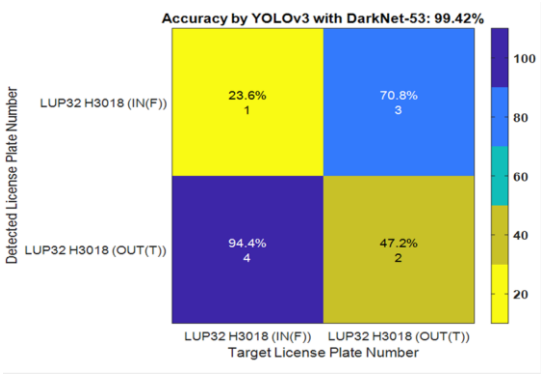
Fig. 25: Confusion matrix accuracy for LPN KUG57 M2872 by: (a) YOLOv3 using DarkNet-53; (b) YOLOv8 using CSPDarknet-53; and (c) YOLOv3 using SqueezeNet.

Summarily, the average percentage accuracies obtained from the confusion matrix shown in Table 6 and that of Figs. 24–28 for the three models shows that the YOLOv8 with CSPDarkNet-53 and YOLOv3 with DarkNet-53 models perform well in accurately detecting both TP, TN, FP and FN for the predicted classified and class target outputs when compared to YOLOv3 with SqueezeNet as also evident from Figs. 24–28.

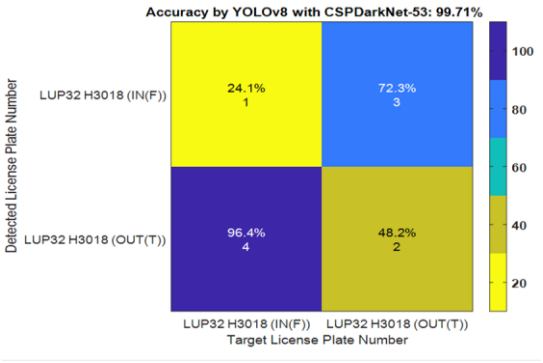
8). Mean Average Precision (mAP): Mean Average Precision (mAP) is a performance metric that summarizes accuracy by averaging the Average Precision (AP) for all object classes.

The mAP evaluates both the classification correctness and the localization accuracy of the model's bounding boxes, providing a single score to compare performance. A higher mAP score (from 0 to 1) indicates better performance, meaning the model is more accurate at identifying and locating objects. Please note that the mAPs are presented in this paper as percentages for quicker interpretation.

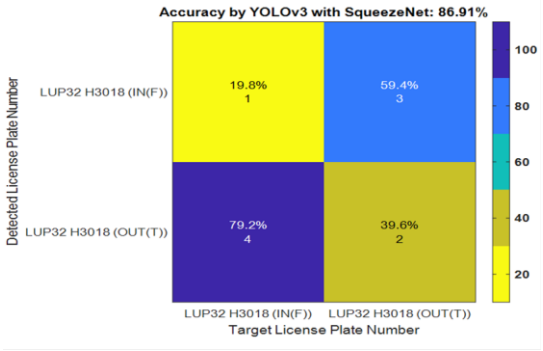
The mean average precision (mAP) for all the 25 LPN detection and recognition using the three models are shown in the eleventh column of Table 7 while the average values of the mAP are given in the tenth row of Table 8.



(a)

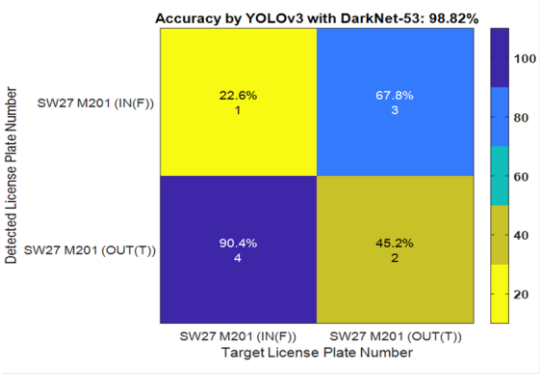


(b)

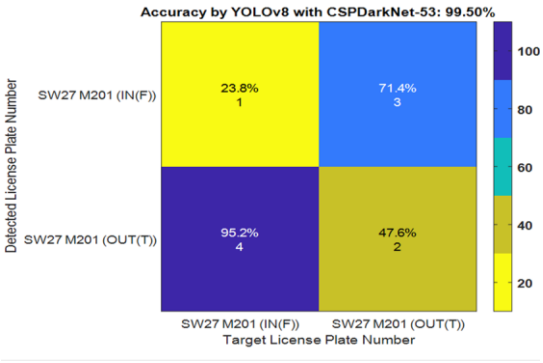


(c)

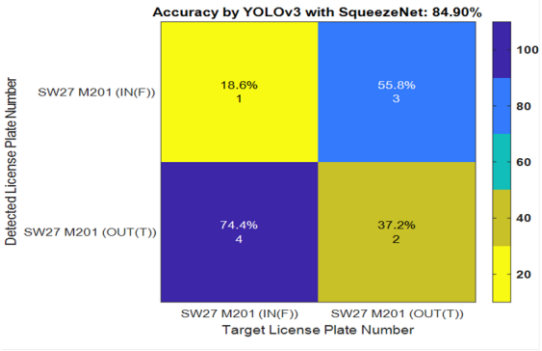
Fig. 26: Confusion matrix accuracy for LPN LUP32 H3018 by: (a) YOLOv3 with DarkNet-53; (b) YOLOv8 with CSPDarknet-53; and (c) YOLOv3 with SqueezeNet.



(a)



(b)



(c)

Fig. 27: Confusion matrix accuracy for LPN SW27 M201 by: (a) YOLOv3 with DarkNet-53; (b) YOLOv8 with CSPDarknet-53; and (c) YOLOv3 with SqueezeNet.

It can be seen that the mAP obtains by YOLOv8 with CSPDarkNet-53 and YOLOv3 with DarkNet-53 models perform closely well in prediction correctness and localization accuracy with mAP values of 99.9416% and 99.9144% respective when compared to a lower 91.5476 recorded by YOLOv3 using SqueezeNet model.

9). Execution Time: The execution time (or inference time) in real-time detection and recognition models from streaming videos is a critical factor that directly determines whether an application can achieve real-time performance, which is the primary strength and goal of the YOLO family of models such as the three model architectures considered in this work. The execution time influences the real-time application and success of the intended goals of the detection and recognition strategy. The three models were implemented on a Windows 11 Home, 16", Intel® Core™ i7-14700HX operating at 5.5 GHz, 16GB RAM, 512GB SSD, WQXGA, Natural silver) and the recorded execution times for the three models are shown in the twelfth column of Table 7 while the average values of the execution times are given in the eleventh row of Table 8.

It is evident that the YOLOv8 with CSPDarkNet-53 and YOLOv3 with SqueezeNet have the highest and lowest execution times of 1650.20 seconds (approximately 4.5839 or 4½ hours) and 119.4084 seconds (1.9901 or 2 minutes) respectively while the YOLOv3 with DarkNet-53 model requires approximately 178.2804 seconds (2.9713 or 3 minutes) for each implementation.

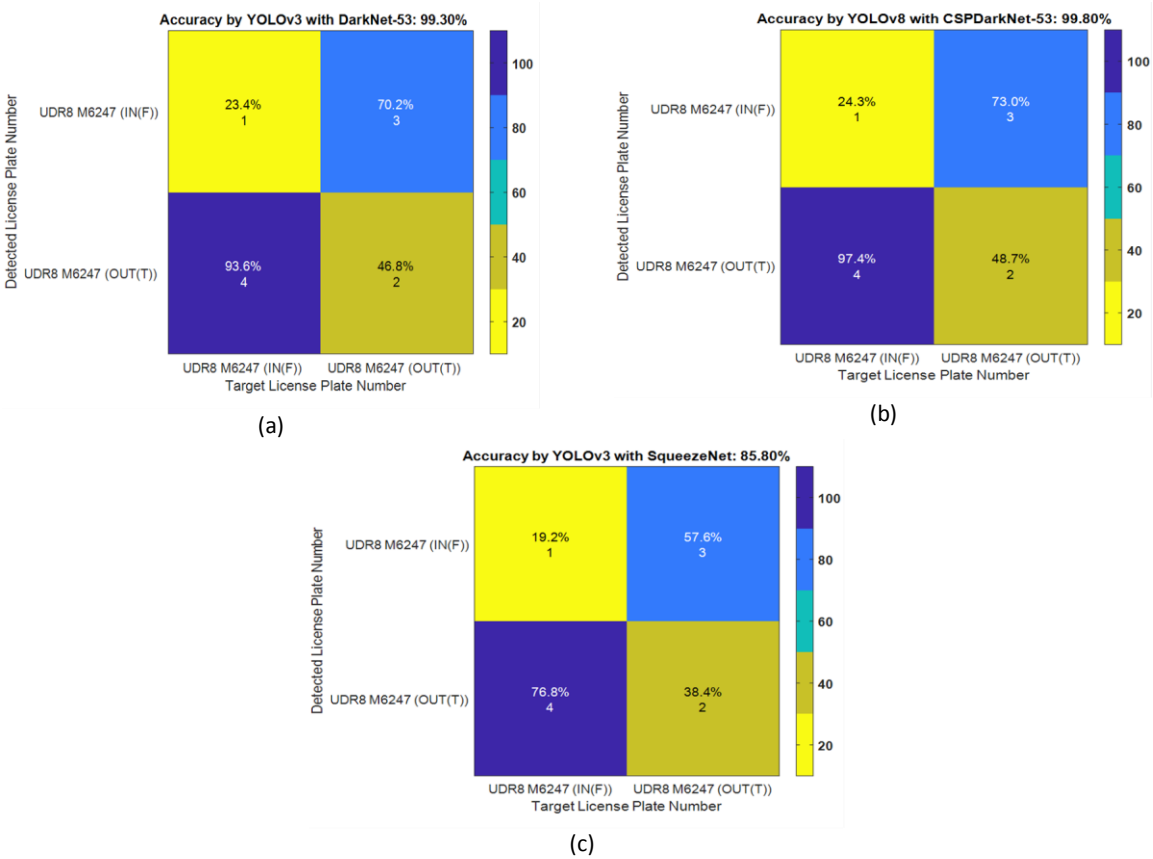


Fig. 28: Confusion matrix accuracy for LPN LUP32 H3018 by: (a) YOLOv3 with DarkNet-53; (b) YOLOv8 with CSPDarknet-53; and (c) YOLOv3 with SqueezeNet.

It is evident that the YOLOv3 with DarkNet-53 model is suitable for the current LPN detection and recognition based on the available computing hardware resources with the justified performance metrics achieved by the model at low hardware cost when compared to the YOLOv8 with CSPDarkNet-53 and YOLOv3 models.

B. 2. Discussion of the Qualitative Results

The feasibility and performance efficiency of the AVIS based on YOLOv3 with DarkNet-53 algorithm with dynamic RDBS has been implemented, tested and validated using an automobile car dealer company (Ola Motors Company located at No. 180 Oyemekun Road, Akure, Ondo State, Nigeria) premises as a case study with random 25 vehicles entering and/or leaving the premises for a period of three days between 10:00 am and 2:00 pm on each day.

The results from the AVIS implementation are shown in Fig. 15 to Fig. 18 respectively for the four case studies presented in this work due to space limitation and page number limit. The corresponding dynamic RDBS showing the vehicle plate number status either as IN and/or OUT together with their respective dates and time-stamps are shown in Table 2 to Table 4. Furthermore, Table 5 validates and shows the final validity status whether a particular vehicle entered without exiting or exited a particular vehicle entered without exiting or exited without returning to the premises as at the close of work by 2:00 pm daily.

The validity status of vehicle entry/exit and exit/entry status with days are summarized in Table 5. According to Table 2 to Table 5, note that the validity status for a particular vehicle is certified True only if that particular vehicle completes either entry/exit or exit/entry routines.

As it can be observed in Fig. 15 to Fig. 18, the AVIS based on the YOLOv3 with DarkNet-53 algorithm successfully captured the respective vehicle LPNs with their respective serial numbers, ID serial integers, time-stamps (created and processed), validity, plate number and the status as shown in Table 2 to Table 4 while the last row of Table 5 shows the final validity status of False which indicates whether a particular vehicle either entered without exiting or exiting without returning to the premises.

It can be seen in Fig. 15 that the YOLOv3 with DarkNet-53 correctly detects the plate number WTF V10 at the point of exit (OUT) on 19th March, 2024 (Day 2) at time 12:23:56 pm as captured by the database on serial number 25 with serial integer X20 on Table 3. The same vehicle with plate number WTF V10 returned back to the company premises on same day at time 01:57:09 pm

(33rd row). Note that the Validity (sixth column) changes from False (25th row) to True (33rd row) based on the unique ID serial integer.

As summarized in Table 5, for the vehicle with serial ID X20 and license plate number WTF V10, the system recorded an exit event (False) and a return event (True) on the same day (Day 2) which is indicated by validity status.

In another case, Fig. 16 shows a vehicle with plate number HR26BU0375 and automatically assigned ID serial integer of X15 that entered (IN) the company on 19th March, 2024 (Day 2) at time 10:48:58 am as captured and processed by the database at time 10:49:02 am on serial number 18 on Table 3. However, the vehicle (HR26BU0375) returned (IN) on Day 3 (20th March. 2025) at 12:43:52 pm as shown at S/N 42 on Table 4. As summarized in Table 5 with ID serial integer X15, the vehicle with plate number HR26BU0375 entered (False) the company on Day 2 and exit (True) on Day 3 which is indicated by the final validity status of True.

The same discussion in the last paragraph holds for Fig. 17 with ID serial integer of X18 and Fig. 18 with ID serial integer of X21. For the vehicle with plate number of KL 20 B 2001 having ID serial integer of X18 in Table 3, Table 4 and Table 5. The entry (IN) time-stamp on Day 2 with False status and exit (OUT) time-stamp on Day 3 with True status are 11:54:28 am and 01:10:53 pm respectively. It is noted the vehicle entered on Day 2 (Table 3) and exited on Day 3 (Table 4) which is reflected in Table 5 with a validity status of True.

As summarized in Table 5, vehicles with plate numbers: *1).* ZRP38 K0241 (ID serial integer X13) entered (False) the company on Day 2 at time 10:08:25 am (Table 3) without leaving the company's premises as at end of this demonstration with a validity status of False; *2).* WPA27 X6437 (ID serial integer X17) and ABJ97 G3318 (ID serial integer X22) exited the company's premises on Day 2 at 11:18:35 am and 01:50:25 pm respectively without returning and thus leaving the validity status as False; and *3).* AAZ64 EK24 (ID serial integer X25) entered (IN) the company's premises on Day 3 (see Table 4) at time 12:39:30 pm without exiting and renders the validity status as False as can be seen in Table 4 and Table 5.

Conclusion and Future Directions

A. Conclusion

A cost-effective automatic vehicle inventory system (AVIS) using the YOLOv3 with DarkNet-53 algorithm with a PostgreSQL dynamic relational database system and Google cloud storage have been designed, implemented and deployed as demonstrated for LPN detection and recognition with successful results.

The performance and efficiency of the AVIS have been demonstrated using the Ola Motors Company (located at No. 180 Oyemekun Road, Akure, Ondo State, Nigeria) premises as a case study on random 25 vehicles for a period of three days between 10:00 am to 2:00 pm on each day.

Quantitatively, results shows that the proposed AVIS based on YOLOv3 using DarkNet-53 competes favourably with YOLOv8 using CSPDarkNet-53 with near equal results within 3 minutes when compared to the about 4½ hours required by the later to achieve the same result.

Qualitatively, the techniques presented and the results obtained from the implementation and deployment of the AVIS demonstrate the efficiency and suitability of the proposed AVIS with a dynamic relational database system (RDBS) for deployment in establishments, companies, institutions, organizations, and government agencies for vehicle inventory management, monitoring, security surveillance, and the development of automatic vehicle inventory systems.

The study also highlights the importance of timely and accurate data collection from streaming video sources for optimizing vehicle inventory management, control, and operational processes. Furthermore, the use of dynamic relational database systems, such as PostgreSQL, in conjunction with Google Cloud Storage, enables flexible and efficient data storage and retrieval, thereby ensuring accurate, reliable, and up-to-date vehicle inventory information.

B. Future Directions

The deployment of two IP-based cameras within the AVIS can improve vehicle license plate number (LPN) capture for independent entry and exit routes. Although the AVIS was implemented on a Core i7 CPU operating at 5.5 GHz, it is worth noting that deploying the proposed AVIS with PostgreSQL and Google Cloud Storage on high-performance computing platforms, such as GPUs, multi-core CPUs, Jetson Nano devices, or FPGAs, could significantly reduce execution times. Such improvements would facilitate operation under hard real-time constraints and support shorter sampling intervals for LPN capture, tracking, detection, and recognition. Furthermore, the CNN–GRU model proposed by Suvarnam and Ch [76], together with the integration of the OCR-based text recognition method suggested by Sunayana and Asim [35], could be incorporated into the AVIS to enable character recognition without explicit character segmentation.

The use of IP-based night vision cameras with up to 1.5 km coverage and routers with fast Internet connectivity for the capture of vehicle license plate numbers during night times, in dark environments and

other varying weather conditions would improve the performance of the AVIS. Incorporating the YOLOv3-based AVIS with the suggestions of Dorbe [77] for the use of fully connected network (FCN) and long short temporary memory (LSTM) based computer vision system for recognition of vehicle type, license plate number, and registration country would create a more effective platform for real-time monitoring and tracking of vehicles.

It is expected that the incorporation of the proposed recommendations in future work will enhance the AVIS by improving the accuracy and efficiency of vehicle LPN capture, real-time vehicle monitoring and tracking, and the secure storage of captured data, while also reducing database synchronization latency and improving overall system performance..

Author Contributions

V. A. Akpan conceptualized and designed the project. All authors carried out the search and review of literature. D. A. Adegoke, E. T. Adejayan and K. A. Adepoju provided funding for the project. V. A. Akpan supervised the project while implementation was done by D. A. Adegoke, E. T. Adejayan and K. A. Adepoju. Furthermore, D. A. Adegoke, E. T. Adejayan and K. A. Adepoju collected all the necessary data for the project. V. A. Akpan wrote the original manuscript. All authors revised and discussed the results and approved the final reviewed manuscript.

Funding

This research is not supported by any external funding.

Acknowledgment

The authors gratefully acknowledge Alhaji Jimoh Momoh, Chief Executive Officer (CEO) of Ola Motors Company (located at No. 180 Oyemekun Road, Akure, Ondo State, Nigeria), for allowing his Company and premises to be used for the demonstration of this project.

The authors also wish to acknowledge the Editor-in-Chief of JECEI, Members of the Editorial Board, and the Reviewers for their helpful comments, excellent and thorough review of this paper which has enhanced the quality of this paper.

Conflict of Interest

The authors declare no potential conflict of interest regarding the publication of this work. In addition, the ethical issues including plagiarism, informed consent, misconduct, data fabrication and, or falsification, double publication and, or submission, and redundancy have been completely witnessed by the authors.

Abbreviations

AVIS	Automatic Vehicle Inventory System
RDBS	Relational Database System
IP	Internet Protocol
YOLOv3	You Only Look Once version 3
YOLOv8	You Only Look Once version 8
PoE	Power-over-Ethernet
Wi-Fi	Wireless Fidelity
FPN	Feature Pyramid Network
IOU	Intersection Over Union
NMS	Non-Maximum Suppression
CCTV	Closed Circuit Television
SQL	Structured Query Language
OpenCV	Open Computer Vision
URL	Uniform Resource Locator
VPN	Vehicle Plate Number
AVL	Automatic Vehicle Location
ANPR	Automatic Number Plate Recognition
LPR	License Plate Recognition
PoE	Power-over-Ethernet
LPN	License Plate Number
mAP	Mean Average Precision
CNN	Convolutional Neural Network
AUC-ROC	Area Under Receiver Operating Characteristic Curve
PR	Precision-Recall
ROI	Region of Interest
ReLU	Rectified Linear Unit
OCR	Optical Character Recognition

DHCP	Dynamic Host Configuration Protocol
Wi-Fi	Wireless Fidelity
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
LiDAR	Light Detection and Ranging
NVR	Network Video Recorder
LSTM	Long Short Temporary Memory
FCN	Fully Connected Network
AP	Average Precision
LED	Light Emitting Diode
GPU	Graphical Processing Unit
GRU	Gate Recurrent Unit
CPU	Central Processing Unit

References

[1] N. Owamoyo, A. Fadele, A. Abudu, "Number plate recognition for nigerian vehicles," Academic Research International, 4(3): 48-55, 2013.

[2] A. M. Atieh, H. Kaylani, Y. Al-Abdallat, A. Qaderi, L. Ghou, "Performance improvement of inventory management system processes by an automated warehouse management system," Procedia CIRP, 41: 568-572, 2016.

[3] B. Barabino, M. D. Francesco, "Diagnosis of irregularity sources by automatic vehicle location data," IEEE Intell. Trans. Syst. Mag., 13: 152-165, 2019.

[4] B. Barabino, M. D. Francesco, S. Mozzoni, "Regularity analysis on bus networks and route directions by automatic vehicle location raw data," IET Intell. Transp. Syst., 7(4): 471-480, 2013.

[5] B. Barabino, M. D. Francesco, S. Mozzoni, "Regularity diagnosis by automatic vehicle location raw data," Public Transp., 4: 187-208, 2013.

[6] M. Buzinkay, "Vehicle inventory management: A data-driven business," Last update 25th January, 2025. Retrieved 13th July, 2025.

[7] J. D. Power, "Automotive inventory management software," Last update 25th January, 2025. Retrieved 13th July, 2025..

[8] M. Karanam, A. K. Reddy, P. Mathyam, A. Mohammed, S. Naik, "Automatic vacant parking places management system using multicamera vehicle detection," in Proc. E3S Web of Conferences, 391(01045): 1-7, 2023.

[9] N. Shvai, A. Hasnat, A. Meicler, "Accurate classification for automatic vehicle-type recognition based on ensemble classifiers," IEEE Trans. Intell. Transp. Syst., 21(3): 1288-1297, 2020.

[10] M. Hussain, "YOLO-v1 to YOLO-v8, the rise of YOLO and its complementary nature toward digital manufacturing and industrial defect detection," Machines, 11(677): 1-25, 2023.

[11] A. Mirkhan, "YOLO algorithm: Real-time object detection from A to Z," Retrieved 13th July, 2025.

[12] J. Nelson, "What is YOLO? The Ultimate Guide [2025]," Last updated 9th January, 2025. Retrieved 13th July, 2025.

[13] R. Kundu, "YOLO: Algorithm for object detection explained," Last updated 17th January, 2023. Retrieved 13th July, 2025.

[14] Z. Keita, "YOLO object detection explained: Understand YOLO object detection, its benefits, how it has evolved over the years, and some real-life applications," Last updated 28th September, 2024. Retrieved 13th July, 2025.

[15] J. Redmon, A. Farhadi, "YOLOv3: An incremental improvement," in Proc. the IEEE Conference on Computer Vision and Pattern Recognition: 7794-7803, 2018.

[16] M. Nagpal, "The ultimate guide to YOLOv3 architecture," Last updated 28th October, 2024. Retrieved 13th July, 2025.

[17] K. J. Oguine, O. C. Oguine, H. I. Bisallah, "YOLO v3: Visual and real-time object detection model for smart surveillance systems (3s)," in Proc. the 2022 5th Information Technology for Education and Development (ITED): 1-8, 2022.

[18] N. Klingler, "YOLOv3: Real-time object detection algorithm (Guide)," Last updated 2nd January, 2022. Retrieved 13th July, 2025.

[19] L. Zhao, S. Li, "Object detection algorithm based on improved YOLOv3," Electronics, 9(537): 1-11, 2020.

[20] S. K. Rajput, J. C. Patni, S. S. Alshamrani, V. Chaudhari, V. Chaudhari, A. Dumka, R. Singh, M. Rashid, A. Gehlot, A. S. AlGhamdi, "Automatic vehicle identification and classification model using the YOLOv3 algorithm for a toll management system," Sustainability, 14(15): 9163. 2022.

[21] Y. Li, J. Wang, J. Huang, Y. Li, "Research on deep learning automatic vehicle recognition algorithm based on RES-YOLO model," Sensors, 22(10): 3783, 2022.

[22] J. J. Wu, H. Xu, J. Zheng, J. Zhao, "Automatic vehicle detection with roadside LiDAR data under rainy and snowy conditions," IEEE Intell. Transp. Syst. Mag., 13(1): 197-209, 2021.

[23] Y. Cui, S. Liu, J. Yao, C. Gu, "Integrated positioning system of unmanned automatic vehicle in coal mines," IEEE Trans. Instrum. Meas., 70: 1-13, 2021.

[24] M. M. Ahmed, M. Abdel-Aty, "The viability of using automatic vehicle identification data for real-time crash prediction," IEEE Trans. Intell. Transp. Syst., 13(2): 459-468, 2012.

[25] M. M. Ahmed, M. Abdel-Aty, R. Yu, "Bayesian updating approach for real-time safety evaluation with automatic vehicle identification data," Transp. Res. Record, 2280: 6067, 2012.

[26] X. Luo et al., "Fast automatic vehicle detection in UAV images using convolutional neural networks," Remote Sens., 12(12):1-15, 2020.

[27] P. Li, W. Zhao, "Image fire detection algorithms based on convolutional neural networks," Case Studies in Thermal Engineering, 19(100625): 1-11, 2020.

[28] H. Zhang, X. Chao, C. Shi, "Closing the gap: A learning algorithm for lost-sales inventory systems with lead times," Manag. Sci., 66(5): 1962-1980, 2020.

[29] G. van Houtum, B. Kranenburg, "Spare parts inventory control under system availability constraints," International Series in Operations Research & Management Science, Springer Nature, London, Chapter 8, pp. 1 – 215, 2015.

[30] D. Abadi, "Consistency tradeoffs in modern distributed database system design: CAP is only part of the story," Computer, 45(2): 37-42, 2012.

[31] Z. Xu, W. Yang, A. Meng, N. Lu, H. Huang, C. Ying, L. Huang, "Towards end-to-end license plate detection and recognition: a large dataset and baseline," in Proc. Part XIII of the 15th European Conference on Computer Vision – ECCV 2018: 261-277, 2018.

[32] P. A. Ioannou, Z. Xu, "Throttle and brake control systems for automatic vehicle following," California Partners for Advanced Transportation Technology UC Berkeley Research Reports: 1-36, 1994.

[33] P. R. M. Júnior, J. M. R. Neves, A. I. Tavares, D. Menotti, "Towards an automatic vehicle access control system: License plate location," in the Proc. the 2011 IEEE International Conference on Systems, Man and Cybernetics: 2916-2921, 2011.

[34] J. Tang, L. Wan, J. Schooling, P. Zhao, J. Chen, S. Wei, "Automatic Number Plate Recognition (ANPR) in smart cities: A systematic review on technological advancements and application cases," Cities, 129(103833): 1-15, 2022.

[35] A. S. Kumar, A. Dharshan, C. Sekhar, S. Manikanta, "Detection of non-helmet riders and extraction of license plate number," in Proc. International Conference on Innovations in Engineering and Technology (ICIET): 1- 7, 2023.

[36] Z. Ma, L. Ferreira, M. Mesbah, "Measuring service reliability using automatic vehicle location data," Math. Probl. Eng., 2014(1): 1-12, 2014.

[37] R. Islam, K. F. Sharif, S. Biswas, "Automatic vehicle number plate recognition using structured elements," in Proc. The 2015 IEEE Conference on Systems, Process and Control: 44-48, 2015.

[38] P. Prabhakar, P. Anupama, S. Resmi, "Automatic vehicle number plate detection and recognition," in Proc. the 2014 IEEE International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT 2014): 185- 190, 2014.

[39] C. L. Azevedo, J. L. Cardoso, M. Ben-Akiva, J. P. Costeira, M. Maeques, "Automatic vehicle trajectory extraction by aerial remote sensing," Procedia: Social Behav. Sci., 111: 849-858, 2014.

[40] J. Wu, "An automatic procedure for vehicle tracking with a roadside LiDAR sensor," in Proc. the 97th Annual Meeting of the Transportation Research Board: 32-38, 2018.

[41] N. Sulaiman, S. N. H. M. Jalani, M. Mustafa, K. Hawari, "Development of automatic vehicle plate detection system," in Proc. the 2013 IEEE 3rd International Conference on System Engineering and Technology: 130-135, 2013.

[42] T. D. Duan, D. Tran, P. V. Tran, N. V. Hoang, "Building an automatic vehicle license-plate recognition system" in Proc. the International Conference in Computer Science (RIVF'05): 59 – 63, 2005.

[43] H. Kim, B. Song, "Vehicle recognition based on radar and vision sensor fusion for automatic emergency braking," i the Proc. the 2013 13th International Conference on Control, Automation and Systems: 1342 – 1346, 2013.

[44] Z. Wang, F. Chan, "A robust replenishment and production control policy for a single-stage production/inventory system with inventory inaccuracy," IEEE Trans. Syst., Man Cybern.: Syst., 45(2): 326-337, 2015.

[45] A. Krishnamoorthy, R. Manikandan, B. Lakshmy, "A revisit to queueing-inventory system with positive service time," Ann. Oper. Res., 233: 221- 236, 2015.

[46] P. Sharma, S. Gupta, P. Singh, K. Shejul, D. Reddy, "Automatic number plate recognition," in Proc. the 2022 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI): 1-8, 2022.

[47] P. Patil, C. Kanagasabapathi, S. Yellampalli, "Automatic number plate recognition system for vehicle identification," in Proc. the 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT): 431-434, 2017.

[48] M. P. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, S. Zdonik, "The Object-Oriented Database System Manifesto," in Building an Object-Oriented Database System: The Story of O_2: 1 - 20, 1992.

[49] Y. Feng, J. Sun, P. Chen, "Vehicle trajectory reconstruction using automatic vehicle identification and traffic count data," J. Adv. Transp., 49: 174-194, 2015.

[50] E. Oomoto, K. Tanaka, "OVID: Design and implementation of a video-object database system," IEEE Trans. Knowl. Data Eng., 5: 629-643, 1993.

[51] N. V. Oort, D. Sparing, T. Brands, R. M. P. Goverde, "Optimizing public transport planning and operations using automatic vehicle location data: the dutch example," in Proc. the 3rd International Conference on Models and Technologies for Intelligent Transportation Systems: 291-300, 2013.

[52] X. Zhao, F. Fan, X. Liu, J. Xie, "Storage-space capacitated inventory system with (r, Q) policies," Oper. Res., 55: 854- 865, 2007.

[53] S. Tuermer, J. Leitloff, P. Rernartz, U. Stilla, "Automatic vehicle detection in aerial image sequences of urban areas using 3d hog features," The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 38: 50-54, 2010.

[54] J. Wu, H. Xu, Y. Zheng, Y. Zhang, B. Lv, Z. Tian, "Automatic vehicle classification using roadside LiDAR data," Transp. Res. Rec., 2673: 153-164, 2019.

[55] P. Manuel, B. Sivakumar, G. Arivarignan, "A perishable inventory system with service facilities, MAP arrivals and PH – Service times," J. Syst. Sci. Syst. Eng., 16: 62-73, 2007.

[56] P. Manuel, B. Sivakumar, G. Arivarignan, "Perishable inventory system with postponed demands and negative customers," J. Appl. Math. Decis. Sci., 2007(94850): 1-12, 2007.

[57] P. Manuel, B. Sivakumar, G. Arivarignan, "A perishable inventory system with service facilities and retrial customers," Comput. Ind. Eng., 54: 484-501, 2008.

[58] T. Moranduzzo, F. Melgani, "Automatic car counting method for unmanned aerial vehicle images," IEEE Trans. Geosci. Remote Sens., 52(3): 1635- 1647, 2014.

[59] P. Petru, "What is YOLOv3? An Introductory Guide," Last updated 26th March, 2024. Retrieved 13th July, 2025.

[60] Y. Dai, W. Liu, H. Li, L. Liu, "Efficient foreign object detection between PSDs and metro doors via deep neural networks," IEEE Access, 8: 46723-46734, 2020.

[61] T. Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, S. Belongie, "Feature pyramid networks for object detection," in Proc. the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR): - 2125, 2017.

[62] J. Hui, "Understanding Feature Pyramid Networks for object detection (FPN)" Last updated 28th March, 2018. Retrieved 13th July, 2025.

[63] N. H. Quang, H. Lee, N. Kim, G. Kim, "Real-time flash flood detection employing the YOLOv8 model," Earth Sci. Inf., 17: 4809-4829, 2024.

[64] H. A. Parhusip, S. Trihandaru, D. Indrajaya, J. Labadin, "Implementation of YOLOv8-seg on store products to speed up the scanning process at point of sales," IAES Int. J. Artif. Intell., 13(3): 3291-3305, 2024.

[65] M. R. Upadhyay, S. Ushasukhanya, V. R. R. Teja, T. N. Malleswari, K. V. N. S. Mahendra, "A deep learning approach for pothole detection using Yolov8 model," in Proc. the International Conference on Advancement in Renewable Energy and Intelligent Systems (AREIS): 1-6, 2024.

[66] B. Khalili, A. W. Smyth, "SOD-YOLOv8—Enhancing YOLOv8 for small object detection in aerial imagery and traffic scenes," Sensors, 24(6209): 1-24, 2024.

[67] J. Terven, D. M. Córdova-Esparza, J. A. Romero-González, "A comprehensive review of YOLO architectures in computer vision: from YOLOv1 to YOLOv8 and YOLO-NAS," Mach. Learn. Knowl. Extr., 2: 1680-1716, 2023.

[68] S. I. Safie, N. S. A. Kamal, E. M. M. Yusof, M. Z. W. M. Tohid, N. H. Jaafar, "Comparison of squeezenet and darknet-53 based YOLO-V3 performance for beehive intelligent monitoring system," in Proc. The 2023 IEEE 13th Symposium on Computer Applications and Industrial Electronics (ISCAIE): 62-65, 2023.

[69] S. Teng, Z. Liu, X. Li, "Improved YOLOv3-based bridge surface defect detection by combining high- and low-resolution feature images," Buildings, 12(1225): 1-18, 2022.

[70] O. I. Obaid, M. A. Mohammed, A. O. Salman, S. A. Mostafa, A. A. Elngar, "Comparing the performance of pre-trained deep learning models in object detection and recognition," Int. J. Inf. Technol. Manag., 14(4): 40-56, 2022.

[71] N. F. Razali, I. S. Isa, S. N. Sulaiman, N. K. A. Karim, M. K. Osman, Z. H. C. Soh, "Enhancement technique based on the breast density level for mammogram for computer-aided diagnosis," Bioengineering, 10(153): 1-24, 2023.

[72] H. Ge, Y. Dai, Z. Zhu, R. Liu, "A deep learning model applied to optical image target detection and recognition for the identification of underwater biostructures," Machines, 10(809): 1-16, 2022.

[73] I. S. Isa, M. S. A. Rosli, U. K. Yusof, M. I. F. Maruzuki, S. N. Sulaiman, "Optimizing the hyperparameter tuning of YOLOv5 for underwater detection," IEEE Access, 10: 52818-52831, 2022.

[74] G. Tepteris, K. mamasis, I. Minis, "Logistics hub surveillance: optimizing YOLOv3 training for ai-powered drone systems," Logistics, 9(45), 2025.

[75] A. Ammar, A. Koubaa, M. Ahmed, A. Saad, "Aerial images processing for car detection using convolutional neural networks: comparison between faster R-CNN and YoloV3," arXiv:1910.07234v3l, 2019.

[76] B. Suvarnam, V. S. Ch, "Combination of CNN-GRU model to recognize characters of a license plate number without segmentation," in Proc. the 2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS: 317-322, 2019.

[77] N. Dorbe, A. jaundalders, R. Kadikis,"FCN and LSTM based computer vision system for recognition of vehicle type, license plate number, and registration country," Autom. Control Comput. Sci., 52(2): 146-154, 2018.

Biographies

Vincent Andrew Akpan obtained his B.Sc. (Hons) degree in Physics from Delta State University, Abraka, Delta State, Nigeria in 1997; a Master of technology (M. Tech) degree in Electronic Measurement & Instrumentation from The Federal University of Technology, Akure, Ondo State, Nigeria in 2003; and a Ph.D. degree in Electrical & Computer Engineering from Aristotle University of Thessaloniki, Thessaloniki, Greece in 2011. He is currently a Professor with the Department of Biomedical Engineering, School of Electrical Systems Engineering, The Federal University of Technology, Akure, Nigeria. His research interest is in artificial intelligence, electronic instrumentation, intelligent adaptive control, automation and embedded systems engineering.

- Email: vaakpan@futa.edu.ng
- ORCID: 0000-0002-5155-2863
- Web of Science Researcher ID: OGN-3642-2025
- Scopus Author ID: 35177320400
- Homepage: https://bme.futa.edu.ng/home/profile/3303

David Ayo-oluwa Adegoke obtained his Bachelor of Technology (B. Tech) degree in Biomedical Technology from the Federal University of Technology, Akure, Nigeria with Second class (Honours) Upper Division in 2024. He is currently a Biomedical Engineer at Redeemers Health Village (RHV), Ogun State, Nigeria. His research interests include Digital Health, Biomedical Imaging Systems, Health Data Analysis, and the application of Artificial Intelligence in healthcare.

- Email: ayodapoadegoke18@gmail.com
- ORCID: 0009-0004-8060-8848
- Web of Science Researcher ID: NA
- Scopus Author ID: NA
- Homepage: NA

Ebunoluwa Temiloluwa Adejayan obtained his Bachelor of Technology (B. Tech) degree in Biomedical Technology from the Federal University of Technology, Akure, Nigeria in 2021. He is currently works as a Biomedical Engineer at the Federal Medical Centre in Ondo State, Nigeria. His research interests are in health informatics, data infrastructure, bioinformatics, and the application of machine learning and block-chain technologies to real-world problems.

- Email: tyebunoluwa@gmail.com
- ORCID: 0009-0003-5551-1940
- Web of Science Researcher ID: NA
- Scopus Author ID: NA
- Homepage: https://tsegalion.github.io/EAdejayan/

Kehinde Adesola Adepoju obtained her Bachelor of Technology (B. Tech) degree in Biomedical Technology from the Federal University of Technology, Akure, Ondo State, Nigeria, with a second class (Honours) Upper Division in 2024. She gained valuable experience as a Biomedical Engineer at Lagos State Teaching Hospital (LUTH). Her research interests include artificial implants, clinical engineering, rehabilitation and tissue engineering.

- Email: adesolaadepoju28@gmail.com
- ORCID: 0009-0003-0469-1232
- Web of Science Researcher ID: NA
- Scopus Author ID: NA
- Homepage: NA