ST open

RESEARCH ARTICLE

System for automatic detection and classification of cars in traffic

Niko Bralić, Josip Musić

University of Split, Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture, Split, Croatia

Objective: To develop a system for automatic detection and classification of cars in traffic in the form of a device for autonomic, real-time car detection, license plate recognition, and car color, model, and make identification from video.

Methods: Cars were detected using the You Only Look Once (YOLO) v4 detector. The YOLO output was then used for classification in the next step. Colors were classified using the k-Nearest Neighbors (kNN) algorithm, whereas car models and makes were identified with a single-shot detector (SSD). Finally, license plates were detected using the OpenCV library and Tesseract-based optical character recognition. For the sake of simplicity and speed, the subsystems were run on an embedded Raspberry Pi computer.

Results: A camera was mounted on the inside of the windshield to monitor cars in front of the camera. The system processed the camera's video feed and provided information on the color, license plate, make, and model of the observed car. Knowing the license plate number provides access to details about the car owner, roadworthiness, car or license place reports missing, as well as whether the license plate matches the car. Car details were saved to file and displayed on the screen. The system was tested on real-time images and videos. The accuracies of car detection and car model classification (using 8 classes) in images were 88.5% and 78.5%, respectively. The accuracies of color detection and full license plate recognition were 71.5% and 51.5%, respectively. The system operated at 1 frame per second (1 fps).

Conclusion: These results show that running standard machine learning algorithms on low-cost hardware may enable the automatic detection and classification of cars in traffic. However, there is significant room for improvement, primarily in license plate recognition. Accordingly, potential improvements in the future development of the system are proposed.

Correspondence to:
Josip Musić
Ruđera Boškovića 32, 21000 Split, Croatia
jmusic@fesb.hr

Cite as:
Bralić N, Musić J. System for automatic detection and classification of cars in traffic. ST-OPEN. 2022; 3: e2022.2102.11.

DOI:
https://doi.org/10.48188/so.3.10

Introduction

As a product of technological advancements and developments within the automotive industry, cars are now equipped with cutting-edge safety systems. Traffic accident reduction has been a driving force behind new safety systems as human error accounts for 94% of all auto accidents [1]. In 2016, within the European Union, traffic accidents accounted for 25,600 deaths and 1.4 million injuries [2]. Worldwide, between 1 and 1.24 million people die on the roads every year and 20-50 million people are injured [3]. This has triggered initiatives such as VisionZero [4], which aim to eliminate all traffic fatalities and severe injuries using various modalities (including active and passive car safety features).

Car safety systems are commonly referred to as Advanced Driver Assistance Systems (ADAS) and may feature different autonomy levels [5], from 0 (no automation) to 5 (full automation/autonomy). The parking assistant is one such advanced system projected for serial installation into all commercial vehicles by 2025, based on the National Highway Traffic Safety Administration (NHTS, USA) recommendations [6]. There are also plans to introduce systems such as traffic lane monitoring and pedestrian and traffic sign detection. ADAS has achieved significant reductions in car crash rates – as much as 78% for reversing collision [7]. Driving Monitoring and Assistance Systems (DMAS) are somewhat similar, but unlike ADAS, these focus on monitoring driver behavior [3]. Although this paper primarily focuses on technical features, user acceptance needs to be considered side-by-side when introducing any new technology [8, 9]. This is especially important when it comes to artificial intelligence (AI)-based systems [10].

The limitations of car's own computation power need to be taken into consideration when implementing any real-time driving assistance systems. More specifically, these may be expanded, but this will take up more space. This is especially true of systems that source data from images and use machine learning for data processing [11]. Loce et al. [12] presented a good overview of widely available, computer vision-based ADAS/DMAS technologies, such as the lane departure warning system, pedestrian detection system, driver monitoring systems, etc. The authors also note that the application of these systems in road safety and traffic monitoring is possible (which is the application of the system proposed in this paper). A more recent overview of computer vision-based ADAS technologies with slightly different categorization is presented in Horgan et al. [13].

Car safety features (regardless of type) are reliable only as far as they are regularly inspected and serviced. Consequently, unregistered cars (and unroadworthy cars) present a particular traffic safety concern. According to an observational study conducted in Queensland, Australia, around 3% of cars on the roads were unregistered [14]. A significant correlation between unregistered vehicles and a high risk of causing accidents has also been observed. This has been explained by such vehicles not being roadworthy and the risky behavior of drivers who usually did not hold a valid driver's license. Comparable numbers of unregistered vehicles have also been observed on the roads in other parts of the world; for example, 3.38% in California, USA [15]. In Croatia, traffic police identified 17,832 unregistered (not roadworthy) vehicles during regular patrols in 2020, which was a 13.8% increase compared to 2019 [16].

Traditionally, unregistered cars have been detected using specialized Automatic Number Plate Recognition (ANPR) cameras installed along roads. Dark [17] used this method to collect data from 256 ANPR cameras installed across the UK, which were made publicly available for further analysis. The obvious disadvantage of these systems is that they are static and therefore easier to avoid for unconscientious drivers. Additional challenges for license plate recognition systems include 1) varying lighting conditions, including glare, shadows, and blur, 2) high-speed vehicles, and 3) various obstacles [18].

However, recent developments in electronics and optical systems have made these cameras more mobile and they can now be mounted within police vehicles. For example, the Croatian police use the Nero-ANPR system [19], produced in Croatia [20]. The system, launched in 2016 as a pilot project, costs around ten thousand euros and consists of a camera that is mounted on the roof of a police vehicle. The system can scan up to 10 vehicles per second (in day and night conditions) with 90% accuracy, even at speeds over 200 km/h [21]. Despite its impressive nominal features, the disadvantages of the system, including its high cost and rooftop installation (a more complicated installation setup) may preclude/impede its wider application. The cost of these systems is considerable in other countries as well [22]: approximately 6 million AUD for about 450 units. Police in other countries uses similar systems, mostly to look for stolen cars [23]. Intriguing initiatives aiming to involve citizens in the detection and analysis of car license plates through interactive games have also been proposed [24].

The above discussion clearly demonstrates the rise in the usage of cameras and related computer vision algorithms in cars and car safety [25, 26]. The groundbreaking development of neural networks during the last decade has also contributed to their application in image processing [27], where one of their most important functions is the detection of specific objects in images [28]. Object detection in images is a key technology behind advanced driver assistance systems. For example, these systems allow cars to detect lanes and other road markings [29] and thus prevent illegal lane cutting while driving. Additionally, they detect and track pedestrians in traffic [30] to improve road safety. Object detection is also useful in video surveillance, autonomous driving [31], robotic vision [32], and similar applications.

Deep neural networks, recurrent neural networks, and convolutional neural networks are just some of the many types of neural networks with practical applications in various technological and scientific fields [27]. Convolutional neural networks (CNN) are the backbone of modern object detection methods [28]. The name comes from a linear mathematical operation between matrices called convolution. CNNs consist of convolutional layers, nonlinear layers, pooling layers, and fully connected layers. The convolutional layer is a CNN's main layer and its task is to extract features from images, such as edges and colors. It can also detect more complex features when combined with other layers, such as various shapes, digits, or parts of the face. Like the convolutional layer, the pooling layer reduces the spatial dimension of a convoluted image. This, in turn, cuts down on the computing power required to process data because it downsizes the matrix dimension of an image [33]. Put together, the convolutional and pooling layer make up the i-th layer of a convolutional neural network. Depending on the complexity of an image, the number of these

layers may be increased to reveal more details, but this will require more computing power. Fully connected layers form the last few layers in the network. The input to the fully connected layer is a flattened vector with features selected by the convolutional layer. The output is normally connected to the softmax function to normalize the output and produce a vector with the probabilities of every possible outcome of a classification problem.

This paper aimed to design a deep convolutional networks-based system that can identify license plates, colors, makes, and models of cars in traffic from available video feeds (in real or near-real time). The camera should be mounted on the inside of the windshield of a car in traffic to detect the cars in front of it. The system is intended to run on an embedded Raspberry Pi computer and car color, license plate, make, and model information should be displayed on the computer screen. Special attention was given to convolutional neural network-aided detection, which was the core element of the system. Seeing as the system comprises various subsystems, the paper also provides the rationale for selecting specific algorithms and describes the process of collecting required images and files as well as neural network learning. The resulting system achieved acceptable results for the target application and is more cost-effective and easier to set up/install than the systems that are currently in use.

Materials and methods

The flowchart of the proposed solution for the automatic detection and classification of cars in traffic is shown in **Figure 1**. As shown in the figure, the system consists of three subsystems. The input for all subsystems is CNN output in the form of part(s) of images (called “patches”) where a car in traffic was detected.

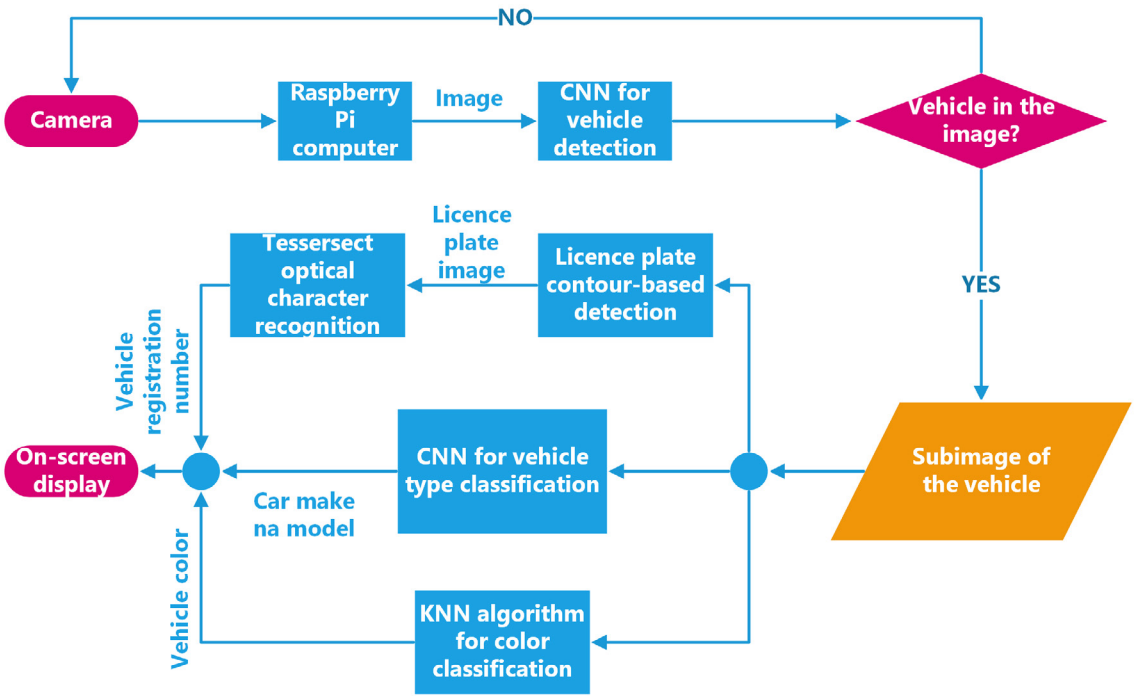

Figure 1. A flowchart of the proposed system.

The first subsystem detects the license plate position in the input car image and then forwards that patch (containing only the license plate) to a character recognition algorithm, which then scans the license plate. Car detection in videos/images is done by the You Only Look Once (YOLO) v4 neural network architecture. The second subsystem detects car makes and models (for which it was trained) based on additional CNN architectures, whereas the third subsystem detects car color using computer vision techniques. The rest of this section briefly describes the algorithms used in this study as well as the data collection for network learning and system testing. In this paper, the term "car make" refers to a car manufacturer (e.g., Audi, Alfa Romeo, Renault), while "car model" refers to a product line designation used by the manufacturer (e.g., Q1, 159, Scienic). Both parameters (make and model) were used to avoid confusion in future system extensions in cases where two different manufacturers use the same car name (e.g., Oldsmobile Fiesta and Ford Fiesta or Opel Diplomat and Dodge Diplomat). Of course, car models may be further subdivided based on the year of production and variant, but as this paper does not delve into such detailed classifications, this remains a possible future expansion of the system.

Detecting cars

Cars were detected with a YOLOv4-tiny detector [34]. YOLOv4-tiny is based on a YOLOv4 architecture but features a streamlined network structure with fewer parameters and filters. This makes it suitable for use on mobile and embedded devices [34]. Thus, it was an appropriate choice for the Raspberry Pi computer. In general, CNN-based detectors may be divided into two-shot (e.g. various versions of Region-based Convolutional Neural Networks – R-CNN [35], Region-based Fully Convolutional Networks – R-FCN [36]) and single-shot (e.g. YOLO and SSD) detectors. All two-shot detection models work by proposing regions. The detection process has two stages. The model first proposes a set of specific areas based on the selected search network or regional proposal network and the classifier then processes only candidates for the selected region (which makes it computationally expensive). A different approach to detection is skipping the region proposal stage and directly initiating image detection. These (single-shot) detectors are much faster and simpler but at the cost of somewhat poorer performance.

Being a single-shot detector, YOLO skips the region proposal step and takes a single "look" at the image to detect objects, which speeds up the process compared to two-shot architectures [34, 37, 38]. YOLO divides the input image into an SxS grid. If the center of an object falls into a single grid cell, that cell is then responsible for detecting the object. In other words, the object is assigned to a cell containing the center of the object.

The classification and localization network can only detect one object at a time, which means that the network can detect only one object per cell. This can cause several issues: 1) as the total number of grid cells is S^2, the maximum number of objects that the model can detect is also S^2. 2) if the grid cell contains more than one object, the model will not be able to distinguish between them, 3) if an object spans more than one cell, it may be detected more than once [34].

The first-generation YOLO was trained to detect 20 different classes, such as a person, cat, dog, car, etc. [34]. YOLO uses a single convolutional network to predict multiple bounding

boxes at the same time, as well as the probability of these boxes. The network was inspired by the GoogleNet image classification model. It has 24 convolutional layers followed by 2 fully connected layers [34].

The YOLOv2 architecture is mainly focused on improving the recall and localization of the YOLO network while maintaining classification accuracy. The following options enable better performance:

- Batch normalization – improves mean Average Precision (mAP) by more than 2%.
- High-resolution classification – network learning using 224×224 images, but the last 10 epochs use 448×448 images. This ensures better network performance for high-resolution image inputs and a 4% mAP increase.
- Anchor fields – predicts multiple objects per grid cell [34].

YOLOv2 identifies the best anchor field shapes to facilitate network learning and calculates bounding box coordinates directly, using fully connected layers. The creators of YOLOv2 recommend using Darknet-19, a new classification model with 19 convolutional layers and 5 pooling layers. It has 91.2% accuracy on the ImageNet database – an improvement over the previous version, which had 88% accuracy [39]. To make YOLOv2 robust for detection in different-sized images, the model was trained using various input image sizes. This allows the network to predict objects in different resolutions, providing an easy compromise between speed and accuracy.

During its heyday, YOLOv2 was the fastest detector. However, detectors developed in recent years achieve superior accuracies, but YOLOv2 is still one of the fastest. Consequently, YOLOv3 traded in some of its speed for improved accuracy [40]. A major change with this version was the new architecture consisting of 53 convolutional layers, significantly more than YOLOv2. The salient feature of the new version is detections at three different sites. Detection at different layers helps address the issue of detecting small objects, which was a major drawback with YOLOv2. The first detection is responsible for detecting large objects, whereas the second and third layers detect medium and smaller objects, respectively.

YOLOv4, the most recent version of the detector (at the time of designing the proposed system), strikes a balance between detection speed and accuracy. As detectors are trained off-line, this advantage is harnessed to develop better training methods and produce detectors with greater accuracy at the same cost. Training strategy-changing methods are known as the “bag of freebies” [41]. The purpose of data augmentation is to increase the variability of input images and make the object detector more robust to images obtained from various environments. Photometric and geometric distortions are the two most frequently used data augmentation methods. Photometric distortion adjusts image brightness, contrast, hue, saturation, and noise, whereas geometric distortion adds random image scaling, cropping, and rotation. In terms of architecture, the objective is to strike the optimal balance between the input network resolution, number of convolutional layers, number of parameters, and number of output layers. A reference model optimized for classification may not be optimized for detection. Therefore, Soviany and Ionescu [38] conducted extensive research on this topic using three potential backbones of the YOLOv4 architecture: CSPResNext50, CSPDarknet53, and EfficientNer-B3. The CSPDarknet53 neu-

ral network used in this paper has been experimentally proven to be the optimal backbone model for the detector. An additional advantage of this basic architecture was its use in the OpenCV library, which shortened the system implementation time. CSPDarknet53 was overlaid with a Spatial Pyramid Pooling Layer (SPP) [42] as this significantly increases the receptive field, isolates the most salient contextual features, and causes almost no network speed reduction. Another key decision when designing the model was using Path Aggregation Network (PANet) to aggregate parameters from different backbone layers for different detection levels [43]. YOLOv4, therefore, consists of a CSPDarknet53 backbone, an additional SPP module, PANet-based information flow, and YOLOv3 as the detector. YOLOv4 is 12% faster and 10% more accurate than its predecessor [41].

YOLO has various implementations: OpenCV-dnn, TensorFlow, Pytorch, and TensorRT. Since this system is designed to run on a Raspberry Pi computer, the most appropriate YOLO implementation must be selected. The decision is all the more important because the finalized system is supposed to run in parallel the plate and car model detection, plate classification, and character recognition on the detected plate. The implementation used in this paper was OpenCV-dnn as this was the fastest YOLOv4 implementation for CPU. This has been proven by several tests [44, 45] on different CPU configurations as well as for various deep architectures (e.g., VGG-16 and DenseNet121).

The algorithm used to detect cars in an image, draw bounding boxes, and extract car images for a more in-depth car classification is shown in **Appendix 1**.

After successful detection, the detected car has to be cut out from the image and then sent for processing and detection. As the cropped image is smaller than the input image, there are less data to process, so this approach speeds up the process and makes it more computationally efficient.

Detecting car make and model

Single Shot Detector (SSD) [46] is another convolutional neural networks-based detector that was used in this study. Removing bounding box suggestions and pixel or feature sampling phases results in a fundamental improvement in speed. Upgraded features include a small convolutional filter to predict objects and offsets in bounding boxes with the help of separate filters to detect different aspect ratios. These modifications provide for a high level of accuracy; selecting relatively low-resolution grants additional increases in detection speed. Although these may seem like minor contributions taken separately, SSD testing has yielded very favorable results [46].

SSD harnesses the principle of feedback convolutional networks. It generates fixed-size bounding boxes and gauges the probability of an object appearing in the boxes and then uses a non-maximum suppression for the final detection. Early network layers are based on standard image classification architecture. This is followed by an auxiliary detection structure with the following salient features:

- Multi-scale feature maps – appends feature layers of decreasing size to the end of the base grid to allow multi-ratio detection.
- Convolutional predictors – every appended feature layer can produce a fixed set.

- Default boxes – a set of default bounding boxes associated with each cell feature, for more features at the top of the grid. The default box value can be used to assign convolutional annotations to the background, for a fixed position of each box in relation to the corresponding cell. For each cell, feature maps predict offsets relative to default box shapes in the cell, as well as grades per class. These, in turn, indicate the presence of a class instance in each of these boxes [46].

During learning, default boxes that match detection should be established and the network trained accordingly. Ground truth boxes are matched with default boxes using the best Jaccard coefficient. Default boxes match ground truth boxes if their Jaccard coefficient is greater than 0.5. This simplifies the network learning process and allows the network to overlap several default boxes instead of choosing the box with the greatest overlap [47].

An important learning parameter is selecting default box scales and ratios. To handle different scale objects, some methods suggest processing an image in different sizes and then combining the results. However, using a feature map from several different layers can mimic the same effect while sharing parameters. This also improves semantic segmentation as the deepest layers record more fine detail.

The SSD was selected for the car model classification task precisely due to its speed. We used the implementation from TensorFlow, a Python library created by Google. It should be noted that SSD algorithms implicitly contain features for image classification, although their primary use is in object detection in images. In cases where this subsystem receives a reliable patch containing a car from the car detection step, a simple network, such as MobilNet v2, would be sufficient for classification. However, we believe our method helped avoid potentially problematic real traffic situations where the previous step may deliver a box containing more than one car (because cars overlap or there is another car in the target car's rectangular box) and therefore laid a foundation for updated versions of the system with options to track and classify more than one car per image (video), making the system more modular.

TensorFlow can train and run deep neural networks for digit classification, object detection in images, text creation, sequence models for machine translation, natural language processing, and simulations based on partial differential equations [48].

The procedure is the same as for training the YOLOv4-tiny network. Detectable car models should be defined before the image collection stage. Given the vast number of different car makes and models on roads, the network was trained using only a small subset of (most popular) vehicles: VW Golf 7, Renault Clio, Suzuki Vitara, Peugeot 208, Renault Scenic, VW Polo, Citroen C4, BMW 3. The "Others" class was added to classify cars that did not make the list and address the challenge posed by a large number of car models on roads.

Networks can be trained from scratch but there are also pre-trained models available. Transfer learning usually refers to taking a model trained for solving one problem and applying it to a different problem. In deep learning, the transfer learning technique refers to using a problem that is similar to the problem we are trying to solve to train a neural network model. One or more layers from the trained model are then used in the new model. The advantages of this type of learning include reduced training time per neural

network model, smaller training sets, and a potentially smaller generalization error. This study used a 320×320 SSD MobileNet v2 model as a relatively good compromise between performance and speed. SSD was chosen because it is compatible with Raspberry Pi, but there are other models suitable for model classification. SSD training takes place wholly on Google Collaborative and a GPU is used to speed up training. Some of the hyper-parameters used to train this network were: *batch_size* = 24, *initial_learning_rate* = 0.004, *decay* = 0.9, *momentum_optimizer_value* = 0.9, *epsilon* = 1, RMSProp optimizator.

TensorFlow Lite is a toolset that helps developers to run the TensorFlow model on mobile, embedded, and the Internet of Things (IoT) devices. TensorFlow Lite is designed to efficiently execute network models on mobile and other embedded devices with limited computing and memory resources. Part of its efficiency stems from the use of a special model storage format. TensorFlow models have to be converted to this format before they can be used by TensorFlow Lite. Model conversion reduces file size and provides optimizations without affecting model accuracy. The TensorFlow Lite converter provides options that allow for a further file size reduction and execution speed increase [49].

The "Model" function detailed in **Appendix 1** receives a car cut-out image, which is then sent to SSD to classify the car model and make. The functionality returns the name of the detected car model.

License plate detection and recognition

Automatic Number Plate Recognition (or ANPR) is a system designed to detect and recognize vehicle license plates without human intervention. It includes the following steps: 1) license plate detection in input images and 2) application of the optical character recognition to detected plates [18].

ANPR is a challenging branch of computer vision due to the wide variety of license plates in different countries around the world. In this study, car license plates were detected using the OpenCV library. The first step in detection was changing image resolutions to avoid issues due to differently sized images. Images were then converted from RGB to black and white format. As the pixel color value does not have an effect on the location of the plate in the image, this change sped up the subsequent image processing operations. All other content in the image except the plate is useless information that may cause interference and therefore has to be eliminated. This is achieved by applying a two-sided filter to blur unwanted details in the image. By changing the parameter values for this filter, the image was blurred with different blur intensities to eliminate interference. The parameter value needed to be carefully adjusted as a too-large value may negatively affect the useful part of the image, i.e. the license plate.

The plate's edges could be detected after eliminating interference. This was done by defining the minimum and maximum values of the intensity gradient to show only edges with intensity gradients within the two values. Only the top 30 detected contours were then sorted by size, in descending order. These 30 contours were assumed to contain the edge contours of a license plate. Since license plates are rectangular, the system detected 4 contours that circumscribe a rectangle. After identifying the plate's bounding contours, the remaining operations were performed only on that image patch and the rest of the

image was eliminated. Croatian license plates uniquely feature the national coat-of-arms of the Republic of Croatia, but this may cause recognition errors. To avoid this, a two-sided filter and a threshold filter were applied to plates to reduce the effect of the coat-of-arms.

Optical Character Recognition (OCR) is a technology for extracting text from images, scanned documents, and photographs. It converts images that contain text into machine-readable textual data. Tesseract [50] is an OCR method that supports more than a hundred languages and can learn new languages. In this paper, we used Tesseract to recognize characters on studied license plates because it provides support for Croatian language. Tesseract can convert and segment images into lines of text or words. A license plate can then be viewed as a single line of text or as a single word. After testing Tesseract's two segmentation options, it was concluded that observing the plate as a single word provided slightly better results. Detectable characters were limited to Croatian capital letters and numbers as these are the only types of characters that appear on license plates [50].

To recognize characters on a license plate, Tesseract first had to detect a license plate. This was a challenging task because license plates take up a minor portion of images. The remainder of the image further complicates matters because it interferes with detecting plate contours. Due to the small size, characters on detected plates are hard to recognize. Images are also often blurry because the cars were in motion. This method may therefore detect wrong contours or altogether fail to detect any contours. The structure of the functionality for detecting and recognizing characters on a plate is shown in **Appendix 1**.

Finally, we note that no original or additional YOLO (or any other) neural networks were used in license plate detection or recognition to improve computational efficiency. More specifically, preliminary tests had shown that this tended to slow down the system, whereas the utilized computer vision methods did not have such limitations. In addition, the size of the training dataset (images) would need to be significantly bigger, which would in turn make the extension to detecting and recognizing foreign license plates more challenging (additional dataset).

Identifying car color

Colors were classified using the K-Nearest Neighbors algorithm trained by Red-Green-Blue (RGB) color histogram [51]. It can classify white, black, red, green, blue, orange, yellow, and violet colors. To increase the number of classifiable colors or improve accuracy, the user needs to work on training data or consider other color characteristics, such as color moment or color correlogram. With color classification, the first step includes feature extraction by RGB color histogram. RGB color histogram shows the color distribution in an image. For digital images, an RGB histogram is a representation of the number of pixels with the same color value, as shown in the example in **Figure 2**.

The k-Nearest Neighbors (kNN) classification method stores all RGB values from training images and classifies new cases based on the distance function. The kNN algorithm assumes that similar colors are neighbors in the color space. Car color was classified based on the Euclidean distance between two points in the color space. The kNN algorithm was implemented in an optimized OpenCV library focusing on real-time recognition [51].

The car color classification function is shown in **Appendix 1**. The function first drew an RGB color histogram for the image and then classified the color by comparing these values with the trained RGB histogram.

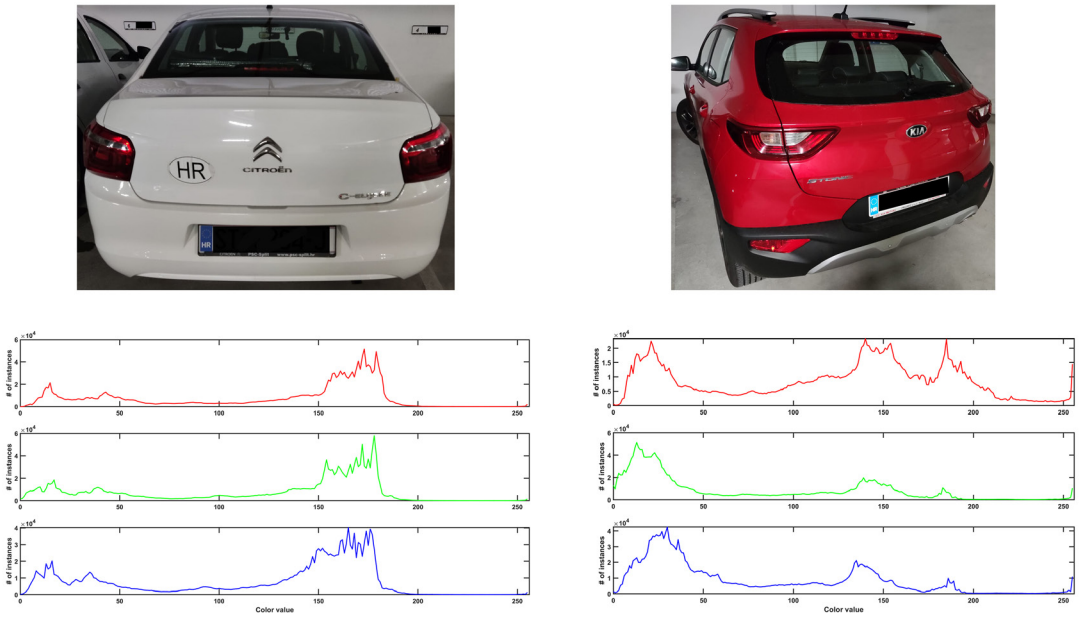

Figure 2. RGB color histogram for two different images. Different color profiles (red, green, and blue channels) are visible for the two cars of different colors (white and red).

The k-Nearest Neighbors (kNN) classification method stores all RGB values from training images and classifies new cases based on the distance function. The kNN algorithm assumes that similar colors are neighbors in the color space. Car color was classified based on the Euclidean distance between two points in the color space. The kNN algorithm was implemented in an optimized OpenCV library focusing on real-time recognition [51].

The car color classification function is shown in **Appendix 1**. The function first drew an RGB color histogram for the image and then classified the color by comparing these values with the trained RGB histogram.

Data collection

Creating a class detection system requires the following files: 1) an image set with coordinates and specific class annotations, 2) a configuration file matching the number of classes, 3) files with class names and absolute paths to specific directories, and 4) learning and testing text files with image names.

One of the seminal tasks when training a neural network is collecting and labeling data (images) since CNN requires large amounts of data for training. Annotated car images were collected online, from the Open Images Dataset. Open Images Dataset comprises 9 million annotated images with object bounding boxes, object segmentation masks, visual relationships, and localized narratives. Considering object detection alone, it contains 2 million images with 16 million bounding boxes for more than 600 classes [52]. Examples of images from the database are shown in **Figure 3**.

Figure 3. Examples of cars (Open Images Dataset).

Bounding boxes from the database were mostly hand-drawn by experts to ensure accuracy and consistency. The images were diverse, including different-sized images and images containing complex scenes and multiple objects. The downloaded database was expanded with images downloaded from various motor vehicle classifieds and a portion of images was collected independently, on roads. Since the diversity of images was essential, the image database was created using as many different sources as possible. The goal was to have the observed class in different environments, with a varying number of surrounding objects, in conditions of good and poor visibility. In the collected image database, the "Car" class is shown on the highway, downtown, in parking lots, during the day, and in the dark. The total number of collected images was 2,100.

In photographs and images sourced online, observed objects were annotated manually. Several programming tools enable loading images, annotating relevant classes for neural network training, and storing relevant data as notes in the appropriate format. LabelImg is a graphical image annotation tool. It is written in Python and uses Qt for its graphical interface. Annotations are saved in Pascal VOC and MS COCO format [53]. The user interface is shown in **Figure 4**.

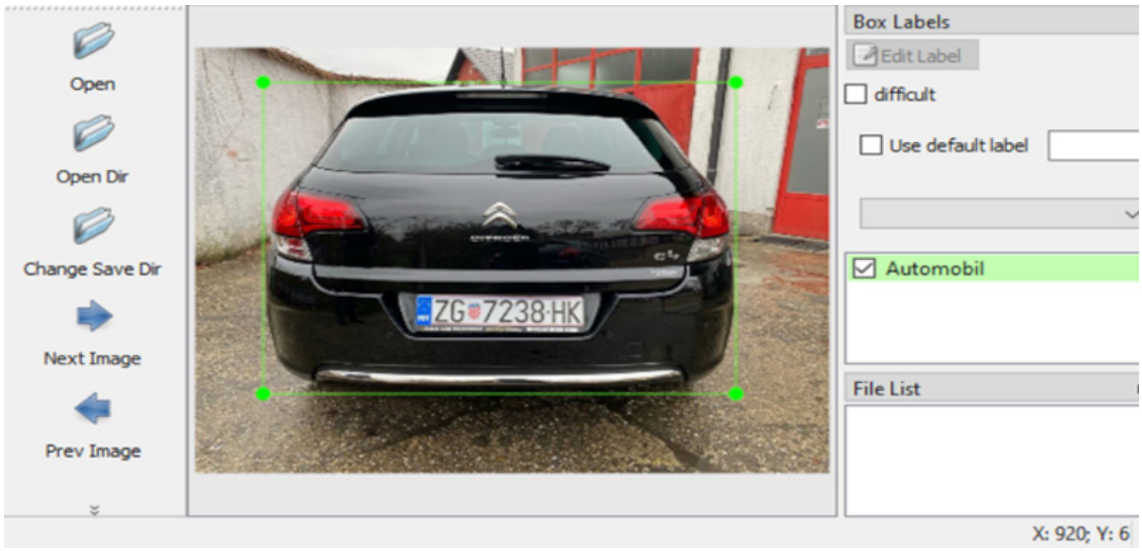

Figure 4. LabelImg tool interface during annotation of a car class (green box).

Collecting images to train the car make and model classification system was somewhat more complicated than for car detector training because it required images of specific car models and makes. Most of the images were sourced online or from car classifieds. This approach assumed that the camera was installed on the windshield of a moving car and that the detected cars were moving in front of that car. This translated into looking for rearview car images. The number of images collected per car model was 200 (2,000 in total, as we used 10 classes), separated into 190 training images and 10 images for network testing (for a total of 1,900 training images and 100 testing images). The images were distributed in the 95:5% ratio as finding images of specific car models required a considerable amount of time and a greater focus was placed on training. We note that a validation dataset was not employed during distribution, given that its primary purpose is to tune neural network hyperparameters to increase accuracy [54]. The focus of this phase of system development was on proving the concept, so the step was omitted. Assigning images to a validation set would unnecessarily reduce the (already limited) set of training images and thus potentially reduce the accuracy of the results. After distribution, the cars were annotated with LabelImg tool. This was followed by the creation of a feature map containing the names of all car models.

To classify car colors, the first step involved collecting images of cars of different colors and sorting them by color into directories. Next, a file with the RGB value of each image was created. Car colors were identified based on these values by finding the nearest known neighbor in the color space using the kNN algorithm. To ensure better color classification results, the kNN algorithm was only applied to patches that contained a car. Seeing as the car was detected by the YOLOv4-tiny convolutional neural network, the kNN algorithm was applied only to the neural network output (**Figure 1**). For even better results, patches containing a car were split in half horizontally to eliminate the rear windshield. This was done because testing had shown that reflections from rear windshields and tinted rear windshields may have a significant effect on the algorithm output.

Table 1 shows a summary of the number of images that were used to run and/or develop parts of the system. We note that images from one subsystem were not used in other subsystems (except for license plate detection and recognition systems). This means that the total image set for all subsystems contained 4,220 different images. The (image) dataset used did not contain any negative examples or images without a car.

Table 1. Biodiesel yield and comparison of its properties obtained with different catalyst concentration and particle sizes with ASTM D6571-02 standard

Subsystem	Images total	Source(s)	Comments
Detecting cars	2100	Open Images Dataset (1,800), online classifieds (200), our original webcam recordings of moving cars in traffic (100)	1,900 randomly selected training images and 200 randomly selected testing images
Color classification	120	Various online sources	20 images per color (white, black, green, blue, orange, and purple)
Car make and model classification	2000	Online ads	190 random training images per class (1,900 total) and 10 random testing images per image (100 total)
License plate detection and recognition	200	Open Images Dataset, online classifieds, our original webcam recordings of moving cars in traffic	Images used only for algorithm testing (as no learning parameters were defined); images from the test set for the car detection subsystem

Training neural networks

Neural networks can be trained on a local computer or by using Google Colaboratory in the cloud. If using a local computer, the training time largely depends on its processor. Additionally, only some NVIDIA graphics cards provide Graphics Processing Unit (GPU) support. Using Google Colaboratory, therefore, makes learning easier and faster. Colaboratory, or "Colab" for short, is a product from Google Research that allows writing and running Python code through a web browser. Colab is especially well suited to machine learning, data analysis, and education. The product does not require any programming environment setup and provides free access to computer resources, especially the Tesla K80 graphics card. The major advantages of Colab are thus its ease of use and free GPU access [55].

Before initiating neural network training, the configuration file needs to be customized to accommodate classes and various learning parameters. **Table 2** shows parameter values set in the configuration file. The "Batch" parameter defines the number of images sent to the network in batch to enable network learning. In addition, **Table 2** shows other hyperparameters introduced to ensure the repeatability of the experiment. Note that most of these were default parameters from the pre-trained network and no algorithms were used to optimize them. Consequently, better hyperparameters that would ensure greater system performance and accuracy may be available, but their effect on execution speed is unclear.

Table 2. Colab configuration file parameters during YOLOv4-Tiny neural network training

Parameter	Value
Batch size	64
Height [pixels]	416
Width [pixels]	416
Maximum batch number	6000
Batch step size (80% and 90% max. batch number)	4800, 5400
Number of classes	1
Number of filters	18
Momentum	0.9
Decay	0.0005
Saturation	1.5
Learning rate	0.00261
Activation functions	Leaky ReLU
IoU loss	cIoU
NMS (Non-Maximal Suppression) algorithm	greedynms
cls_normalizer	1
iou_normalizer	0.07
beta_nms	0.6

Due to the simplicity of the YOLOv4-tiny architecture, the "Batch Size" parameter was set at 64. As the height and width of images had to be multiples of 32, both were set to 416 pixels. This size was a compromise between high-performing detection and training time. For better results, height and width parameters would need to be increased, but this would also increase training time. The maximum number of batches is calculated using formula (1):

$$Max_{batch} = class\ number * 2000 \quad (1)$$

This parameter cannot be set to less than 6,000, so all detectors with less than 4 classes set this parameter to 6,000. The value for a 5-class detector was 10,000. The next parameter was a step, which had two values set to 80% and 90% of the maximum batch value, respectively. This parameter was set to 4,800 and 5,400, respectively, to train a single class. The number of classes had to be set to the desired value, in this case 1 because the network was trained to detect cars only. The number of filters was set based on formula (2):

$$Filters = (class\ number + 5) * 3 \quad (2)$$

Two files had to be created to train the YOLOv4-tiny detector. The first file contained the names of all classes so that there was one class name per line. For car detection purposes, this file had a single line because the system was trained to detect only one class. The second file defined the number of classes and absolute paths to several files, most importantly a backup folder with all required files during neural network learning. The final two files required before the start of the training were training and test files with relative paths to all training and testing images, generated using a Python script.

During cloud-based learning, there is a possibility of Colab reporting an error or remaining idle during training. A partially learned model would not be lost even in this case because it is saved into the backup directory after every 100 iterations. Learning can then resume from the last saved weight file to avoid having to restart a new neural network learning process. YOLOv4 uses the Complete Intersection over Union (CIoU) loss function that introduces two new concepts. The first concept is the measure of distance between the actual center point and the projected center point of the bounding box. The second concept is the box ratio: it compares the actual box ratio and the projected box ratio. These two concepts measure the quality of the bounding box prediction. After training, a loss function diagram can be observed.

Implementing real-time hardware access

The system for automatic detection and classification of cars in traffic was implemented on an embedded Raspberry Pi 4 (4GB) computer. The Raspberry Pi is a low-cost, credit-card-sized computer. It is capable of doing everything a desktop computer can do, from browsing the internet and playing high-definition videos, to making spreadsheets, word-processing, playing games, and programming. Several different models have been developed over the years, but the model chosen for the purposes of this study was the Raspberry Pi 4 with 4GB of RAM. The basic features of the Raspberry Pi are shown in **Table 3**.

Table 3. Technical specifications of the Raspberry Pi 4 (4GB) used in the proposed system

Component	Description/Features
Processor	4-core Cortex-A72 (ARM v8) 64-bit 1.5GHz
Random Access Memory (RAM)	4GB LPDDR4-3200 SDRAM
Connectivity	2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
Ports	2 × microHDMI, 2 × USB2.0, 2 × USB3.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)
General Purpose Input/Output (GPIO) pins	40-pin header
Storage	MicroSD
Power supply	USB-C 5V, 3A
Operation system	Raspbian OS

Figure 5. The system installed in a car.

After merging individual subsystems into a single system, the programming portion of the work was over and the only thing left was to test the system on the Raspberry. After installing all required libraries, the automatic traffic detection and classification system was ready for in-car mounting and video testing.

The automatic traffic detection and classification system was mounted within a car as shown in **Figure 5**. The camera was mounted on the rearview mirror. System speed is a key factor in video testing and this system ran at 1 frame per second (1 fps). The system speed was acceptable for areas where car speeds were low, such as downtown and near traffic lights. The car detection alone took about 0.3 s. Plate detection and recognition took about the same time. Car model classification took 0.3 s, whereas color classification was quicker and took less than 0.1 s (**Figure 6**).

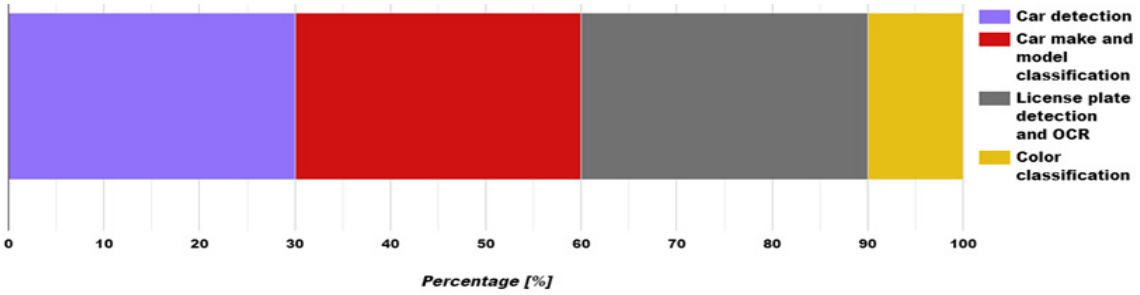

Figure 6. Graphical representation of the execution time for each phase of the proposed system as percentages of the total time (1 second).

Results

Figure 7 shows successful and partially successful examples of system-wide classifications. More specifically, **Figure 7** e) shows a successful detection of a car, car color, and model, but not license plate detection, whereas **Figure 7** f) shows incorrect car color and model classifications.

The selected YOLOv4 single-stage CNN architecture was trained for several hours within the Google Colab environment. The achieved training loss function output is shown in **Figure 8** below. As shown in the figure, losses were reduced with training and converged at a given value. The loss function declined rapidly and the value of losses stabilized after 2,000 iterations. The final loss value was 0.4, indicative of a well-trained detection system. This confirmed that the network training time had been sufficient. The code used to obtain this result is shown in **Appendix 1**.

Figure 9 shows a bounding box around the car and the probability of a car appearing in the box (upper left corner of the box in the figure). The probability threshold is a limit value used to decide whether the required object was, in fact, detected: if the value is less than the default value, the detected object is assumed not to be the required object, so the bounding box is not drawn. The threshold value for car detection in this paper was set to 0.8. This value was determined experimentally and was somewhat more rigorous than normal to minimize (eliminate) the number of false-positive detections and ensure the normal functioning of other subsystems that work with patches containing detected cars.

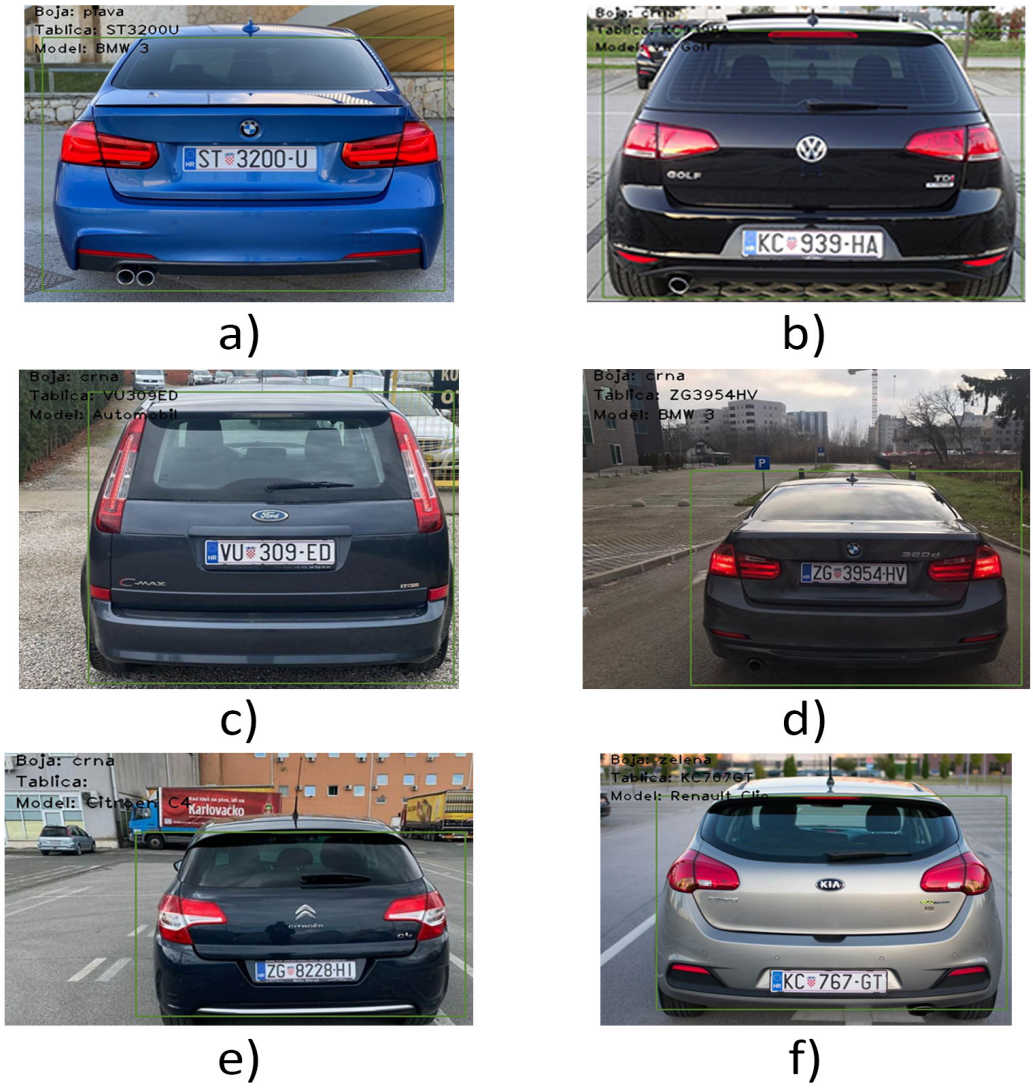

Figure 7. Examples of the final classification of the proposed system: a), b), c), d) – successful car classifications; e), f) – partially successful classifications; (e – no plate detected; f – wrong car model). Consider the detection results for each subsystem in the upper right corner of each image.

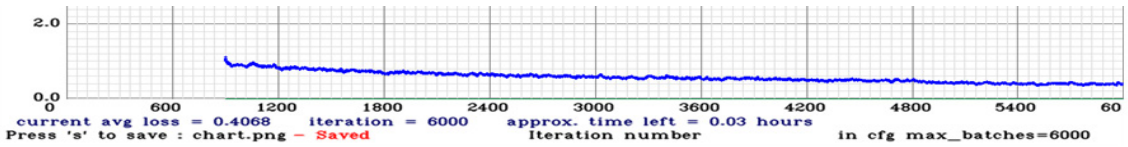

Figure 8. YOLOv4-tiny neural network learning losses showing its convergence.

After successful detection, an image patch containing a detected car had to be cut out and sent for further processing and detection. As the cropped image was smaller than the input image, there were less data to process, so this approach sped up the entire system.

Figure 10 shows examples of correct and incorrect car color classifications. Color detecting largely depended on the car detection in the previous step. Precise (narrow) bounding boxes around a car necessarily include less surrounding environment and help ensure successful color classification. **Figure 10**, however, shows that even in narrow boxes, there was always some background due to the mismatch between the box and car shapes. The

Figure 9. A car detection output using YOLOv4-tiny neural network.

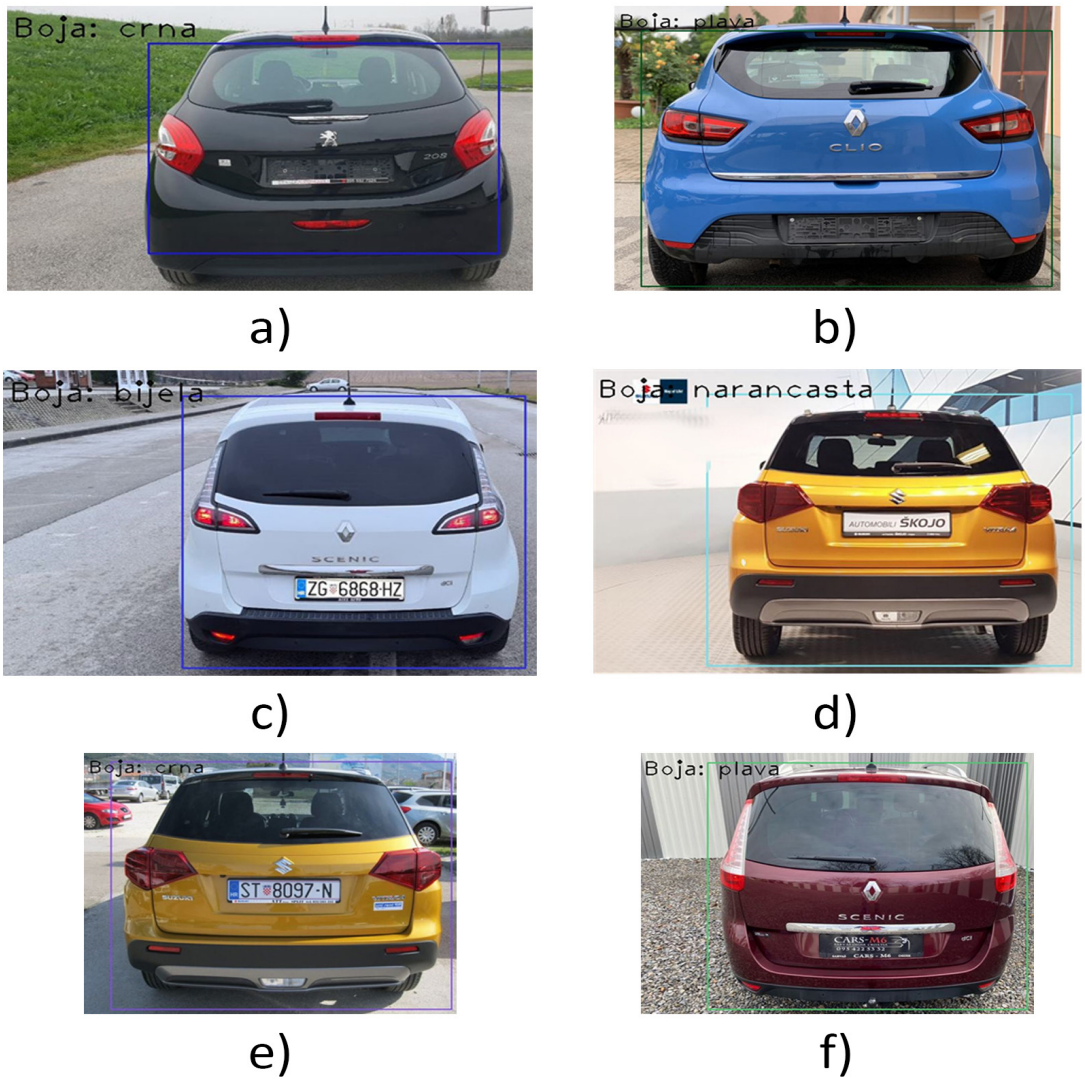

Figure 10. Examples of car color detection: a), b), c), d) show examples of accurate color classifications; e), f) show examples of poor color classifications. Consider the color detected by the algorithm in the upper left corner of each image.

figure also shows how the color of the rear windshield can affect the RGB color histogram of the whole car.

An example of the license plate detection and recognition subsystem operation is shown in **Figure 11**.

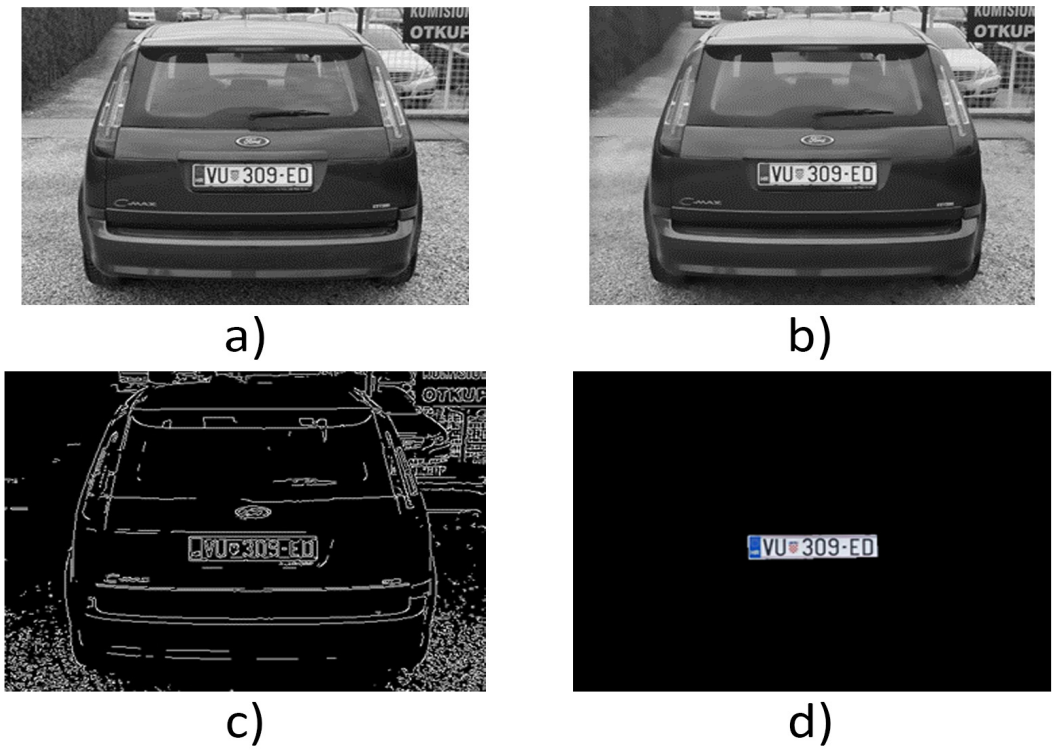

Figure 11. Examples of license plate detection and recognition: a) conversion to black and white image, b) application of a two-sided filter, c) detection of edges, d) table.

The SSD network was trained for 4 hours, or 8 thousand iterations. The loss function for classification was a weighted sigmoid function. The loss function showed a loss reduction and convergence with increasing iterations. The final loss value was 0.1 (**Figure 12**). After training, the model was ready to use, but the model first had to be converted to a format that can be understood by the Raspberry Pi.

The system for automatic detection and classification of cars in traffic was tested using three different datasets. **Table 4** shows the test results for 1,000 images used for network training. The output percentage shows relatively good network behavior (detection efficiency over 90%), which was the expected result since the output was based on the same images that were used for learning (development) in individual subsystems.

Table 4. Testing results for different parts of the proposed system using a set of training images (also used for learning)

Total number of images in the dataset	Number (percentage) of successful car detections	Number (percentage) of successful color detections	Number (percentage) of successful car model detections	Number (percentage) of successful plate detections
1000	921 (92.1%)	704 (70.4%)	814 (81.4%)	539 (53.9%)

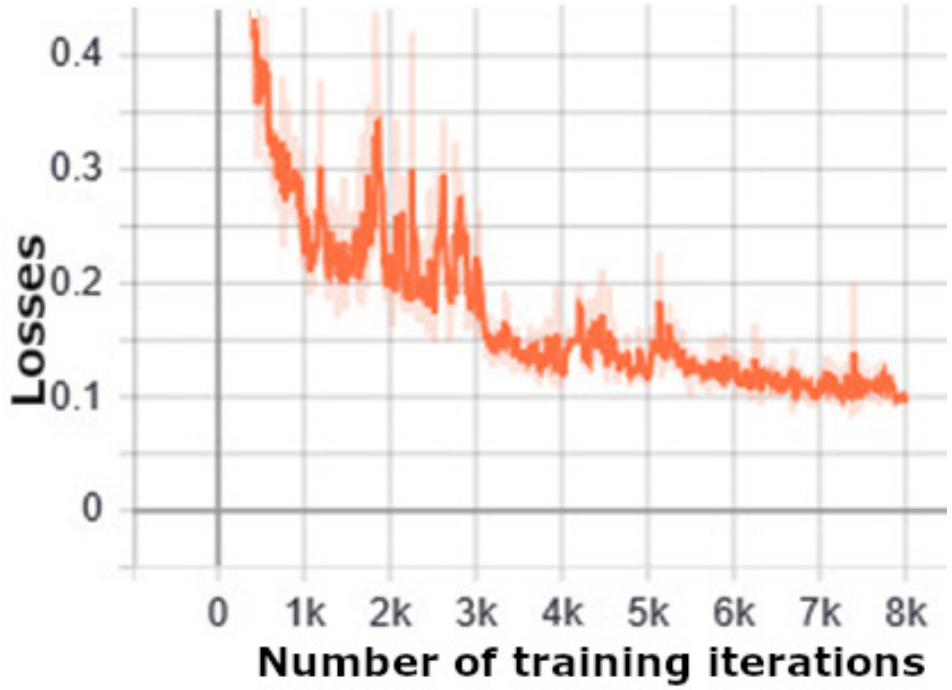

Figure 12. Single Shot Detector (SSD) neural network losses during learning, showing its convergence.

System performance was then tested using images that the subsystems did not see during training (test set), so their significance in terms of the achieved results was somewhat higher. The results obtained for 200 such images are shown in **Table 5**. Readers should note that the tables use the accuracy parameter as a representation of system performance (the same method was used in similar papers, e.g. [56]), but there are other parameters (such as IoU) that may give different insights into the system's functioning but were not used to report results in this paper.

Table 5. Test results for different parts of the proposed system outside the training image set

Total number of images in the dataset	Number (percentage) of successful car detections	Number (percentage) of successful color detections	Number (percentage) of successful car model detections	Number (percentage) of successful plate detections
200	177 (88.5%)	143 (71.5%)	157 (78.5%)	103 (51.5%)

The results shown in **Table 5** match similar systems available in the literature, except for the percentage of successful license plate detections and recognitions, which was slightly lower [57, 58]. Consequently, the following section gives a detailed analysis of the latter. To ensure a correct interpretation of this percentage, the following list identifies several general parameters that are not related to the used algorithm but affected the presented result: the quality of images, license plates for which the system was developed, and result reporting/interpretation methods. Additionally, a computation simplicity criterion for real-time execution (which often involves sacrificing some accuracy) of the system (itself being just part of the complex overarching architecture) needs to be taken into account.

Given that the proposed system analyzed Croatian license plates, it makes sense to compare it directly with systems from literature that work with the same type of license plates.

Unfortunately, the number of available papers on this subject in the literature is limited (e.g. [59-61]), as are the possibilities for comparison. For example, Romić et al. [59] reported the accuracy of "about 80%", Henry et al. [60] reported the accuracy of 97%, and Novosel [61] reported the accuracy of 81.05%. However, for the sake of a correct interpretation of these results and their comparison with the results of our study, it should be noted that Romić et al. [59] and Henry et al. [60] used higher-quality images of stationary cars, with a constant distance between car and camera. Due to the uniqueness of Croatian tables, Henry et al. [60] interpreted the "O" and "0" characters as a single character (i.e., wrong character classifications for the pair were not reported). In Novosel [61] a similar approach was used which treated not only "O" and "0" but also the "1" and "I" characters as equivalent values, as these were impossible to distinguish even by humans at lower resolutions. The reported accuracy of 81.05% was obtained by calculating the total number of characters on all license plates in the images and comparing them with the number of incorrectly detected characters. In contrast, this paper reports the results for accurate detections of the full license plate (without introducing any character equivalence, such as between "O" and "0" or "I" and "1", respectively). Therefore, even in cases where the system accurately detected 7 out of 8 characters on a plate, this was labeled as an incorrect classification (in other words, 0% as opposed to the 87.5% accuracy when using the result interpretation method as in Novosel [61]). The proposed system did not use the more complex approaches seen in Henry et al. [60] and was tested on static and dynamic lower-resolution images shot from various angles. Due to all the above, we believe that the achieved result of 51.3% was relatively good, but also that there is room for improvement, primarily by adding a classification neural network (which would, consequently, require more robust hardware).

Finally, the system was tested on a limited set of images that did not contain cars to test system behavior in such situations that may arise in real-world applications. The results of this testing are shown in **Table 6**.

Table 6. Testing results for different parts of the proposed system using a set of training images (also used for learning)

Total number of images in the dataset	Number (percentage) of successful car detections	Number (percentage) of successful color detections	Number (percentage) of successful car model detections	Number (percentage) of successful plate detections
10	0 (100%)*	0 (100%)*	0 (100%)*	0 (100%)*

* – in this experiment, non-detection of a car/plate/color (since there are no cars in the images) was a successful outcome.

According to the results shown in **Table 6**, the system behaved appropriately in these scenarios and there were no false-positive car detections. Taking into account that the results shown in the table were obtained without any negative examples in the training set, their inclusion should not be essential in this regard. However, this may affect car detection rate (by reducing false-negative examples by lowering the threshold of the YOLO algorithm).

Based on the results presented in **Tables 3**, **4**, and **5**, we may conclude the following: A YOLO detector was used to detect cars. In tests, the system correctly detected cars in 92.1% of cases (on the training set) and 88.5% on the test set. Diminished accuracy on the testing set compared to the test set was expected. Since the decline in accuracy was not significant

(and the accuracy was comparable), it may be concluded that there was no overfitting of the neural network. Accuracy comparison between parts of the proposed car detection system and similar systems from the literature is given in **Table 7**. The table also provides additional data on the images and platforms used in the studies to provide more context for the obtained results and their comparison with the proposed system.

Table 7. A comparison between car detection accuracy results and similar systems from the literature

Paper	Image database	Database size	Platform	Metrics	Metrics value
Novosel [61]	custom; two sites on Croatian roads; 1440x1080; static camera	421 images	not specified (off-line)	accuracy	94.77%
Ni et al. [62]	custom and available on search engines	5,000 (3,500 training and 1,500 testing)	i9-9920X@3.5GHz, 16GB, GeForce RTX2080Ti	average accuracy	92%
Sindhu [63]	CCTV camera images	1,000 (800 training and 200 testing)	Google Colaboratory	accuracy	53%
Wang et al. [56]	KITTI and BDD100k	114,999 (100,000 BDD and 14,999 KITTI)	Personal computer (unspecified) with NVIDIA RTX 2080Ti	AP70 (KITTI) / Accuracy (BDD)	90.5-92.38% / 69.2-77.6%
Alsanabani et al. [64]	from the literature (surveillance camera footage); 1920x1080	8,640 (7,200 training and 1,440 testing)	Personal computer (unspecified) with NVIDIA GTX 1080Ti	mAP@0.5	38.1%
Tran et al. [65]	AI City 2021 Track-1 image database	2860	Xenon Silver 4216@2,1GHz, 16GB, Tesla T4	mAP@0.5	87.7%
proposed system	Open Image with Dataset, internet, custom recording (various resolutions)	2,100 (1,900 training and 200 learning)	learning with Google Colaboratory; running on Raspberry Pi 4B	accuracy	88.5%

For color identification, accuracy was 70.4% because color classification depended on car detection. In other words, if a bounding box was outsized relative to the car, it tended to include more surroundings, which in turn may affect the color of the image (using RGB color histogram). These results are comparable to results for computer vision and machine learning-based systems available in the literature (excluding neural networks). For example, the Support Vector Machine (SVM) approaches used in Lin et al. [66] yielded an average accuracy of 68% or 85.75% (for four colors), depending on the approach. The results were obtained using a dataset containing 832 images, but these were pre-processed to contain only specific cars. In Brown [67] a more realistic images were used (a total of 168 images split into two datasets) that included some background (due to the shooting angle, this was always the road). Six colors were classified using several approaches. The obtained accuracy results ranged from 71% to 89% for set 1 and 68% to 73% for set 2, respectively. Finally, Wang et al. [68] used stationary surveillance cameras to collect traffic images with more than 42,000 isolated individual vehicles. Images were processed using a desktop computer (i7@2.67GHz) and classified according to the 7 most commonly used colors. The average accuracy was 88.18%. It is interesting to note that the authors also compared their results with six methods from the literature (including a neural network-based system) with the average accuracy ranging from 64.03% to 84.77%.

Car models and make classification using SSD neural networks yielded the accuracy of 81.4% and 78.5% on the training and test set, respectively. As was the case for car detection, a small decrease in accuracy was also observed here, suggesting (in combination with the losses in **Figure 12**) that the neural network learned correctly and that there was no overfitting. Visual examples of car make and model detection are shown in **Figure 13**. To provide some context for the results for this subsystem, we will provide a short comparison with similar systems from the literature (focusing exclusively on detecting car makes and models, not the whole system, as was the case here). In Lee et al. [69], the convolutional neural network used to detect car makes and models had an accuracy of 96.3%. To develop the system, a database with 291,602 images (80% training and 20% testing) and 766 car models (collected during one calendar year) was created using a stationary camera. The system ran on a platform with an i7@2.6GHz processor and GeForce GTX 1080 graphics card. Interestingly, the results shown in this paper were compared with 7 other systems from state-of-the-art literature where the accuracy ranged from 85% to 98.63%. Ni and Huttunen [70] also provide a brief literature review but make a distinction between make recognition systems (4) and model recognition systems (19). Consequently, the accuracy of car make recognition ranged from 81.3% to 99.6% (for image databases ranging from 1,482 to 30,955 images), whereas car model recognition accuracy ranged from 64.3% to 98.5% (for image databases containing between 300 and 90,940). Novosel [61] classified car makes on Croatian roads by extracting and identifying vehicle logos (based on car front view images). The database contained 142 vehicles (with an average logo size of 20×20 pixels) and achieved an accuracy of 77.46%. Interestingly, the accuracy of commercially available neural network-based solutions [71] ranged from 75% to 95%, depending on the database used. The authors included 400 available car makes and 7,000 models, but did not specify the images and hardware (video cameras) used for detection.

Table 8 shows the confusion matrix using 200 test images for car model classification. The biggest challenge for this subsystem was posed by the "Others" class as it comprised a large number of different car models. Not surprisingly, the results for this class (68.8%) had the most significant negative impact on the overall accuracy of the subsystem. To detect license plates, the authors used a method from the OpenCV library (*findContours()*) to find plates in a binary image using contours. The binary image was generated using the Canny edge detector. A shortcoming of this approach was that it sometimes detected the rear windshield instead of the plate. When recognizing characters on a plate, the characters may be too small to recognize. The most common errors involved similar characters, such as the capital letters "I" and number "1" or conflating the letter "Z" with numbers "2" or "7". Croatian license plates can also contain diacritics, such as the letters "č", "ć", "đ", "š", and "ž". Detecting diacritical characters on plates was problematic because the glyphs are generally poorly done, which makes them hard to recognize even with the naked eye. Tesseract was not able to recognize the diacritics because they were connected with the letter. In this paper, the total full plate recognition accuracy was 51.5% for tested cases.

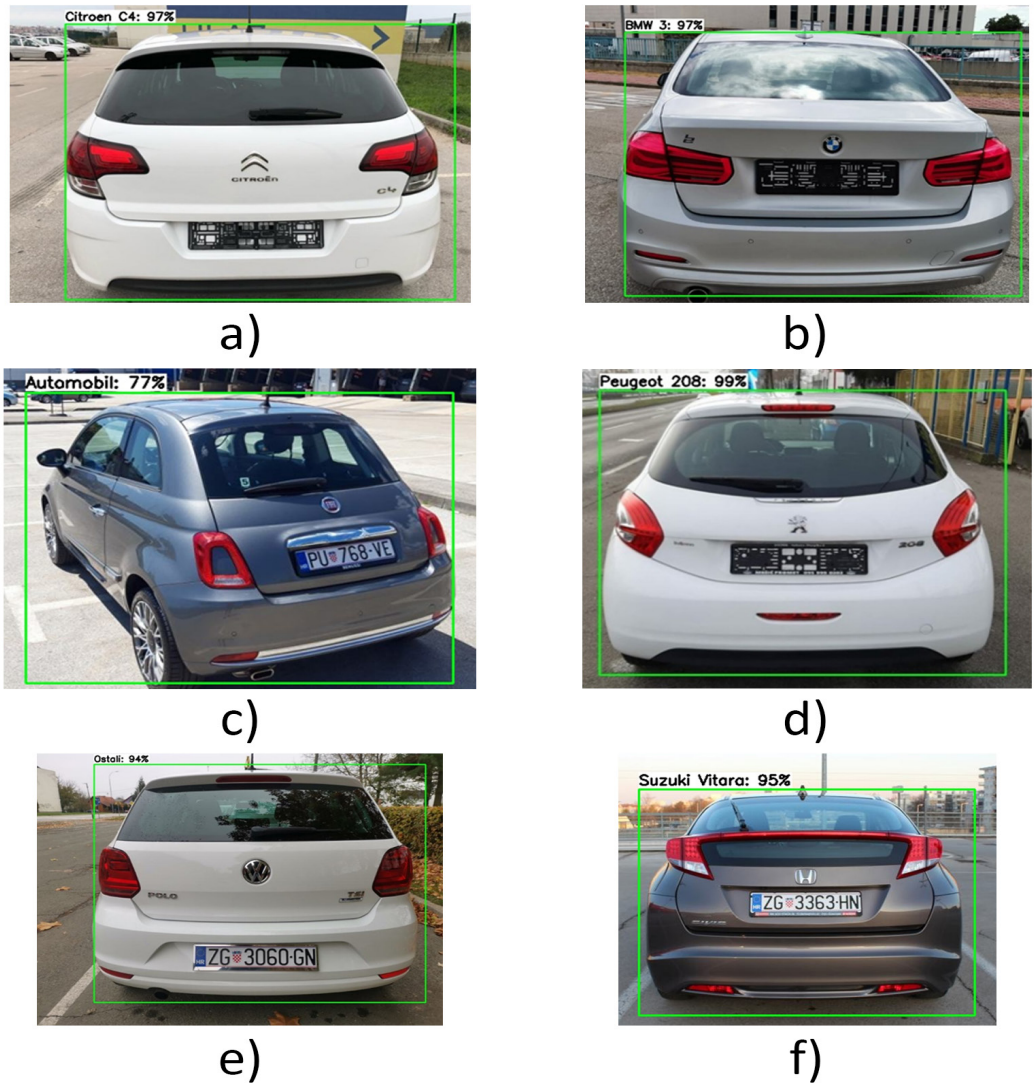

Figure 13. Examples of car model and brand detection: a), b), c), d) – correct classifications of car models using SSD, e), f) – incorrect model classifications. The text in the upper left corner of the detection box shows the detected car make and model as well as certainty.

Table 8. A confusion matrix of car model classifications. The numbers in the table are a representation of detections of each car model in the test set disaggregated by class

Real class / Predicted class	Clio	Scenic	Golf	Polo	208	BMW 3	Vitara	C4	Other
Clio	20	-	-	-	-	-	-	-	2
Scenic	-	19	-	-	1	-	1	1	1
Golf	-	1	15	1	-	-	-	-	2
Polo	-	1	-	15	1	-	-	-	-
208	-	-	-	-	21	-	-	1	-
BMW 3	-	-	-	-	1	19	-	-	-
Vitara	-	3	-	-	-	-	20	-	-
C4	-	-	-	-	1	-	1	17	-
Other	-	2	-	-	1	1	-	1	11

Discussion

In this paper, we developed an automatic convolutional neural network-based computer vision system for the detection and classification of cars in traffic [33]. The system is autonomous and automatically detects cars on the road as well as their makes and models, classifies colors of observed objects, and recognizes license plates.

Preconditions for the proper functioning of the system include having a rear-view image of a car and a manageable distance between the camera and the observed car (otherwise, the license plate may be too small for reliable detection and character recognition). The system was implemented on a Raspberry Pi 4 laptop and mounted on a car's windshield (on the inside) to observe cars in front of the camera. The system provides quick, straightforward solutions to enable real-time implementation on a Raspberry Pi.

Car detection using the YOLO architecture had a total accuracy of 88.5%. This accuracy is comparable to the Nero-APNR system, which has an accuracy of 90% [20, 21]. However, the Nero-APNR system can operate at higher shooting speeds and higher car speeds with up to 10 simultaneous detections. It is unknown whether it can classify car models and makes as well as colors, which we believe is a strength of our system, as it facilitates the detection of irregular (e.g., stolen) license plates mounted on cars which can potentially deceive standard license plate recognition systems (such as Nero-APNR). The proposed system can achieve all of the above with a much simpler setup and at a fraction of the cost [20, 21]. The proposed system classifies color and model and detects license plates only in a patch containing a detected car, as opposed to the whole image. Consequently, these steps depend in part on the car detection phase. Car detection accuracy has the greatest repercussions for color classification. More precisely, the potential for incorrect car color classification increases when the bounding box contains a lot of surrounding environments. The SSD detector used for car model and make classification achieved a 78.5% accuracy. The final task was detecting and recognizing characters on license plates. A plate was detected using various filters to find contours. This was a complex task due to blur in images of moving cars as well as Tesseract's struggle with recognizing characters at greater distances. Rainy, foggy, and night conditions on the road significantly affect the performance of the system because they degrade the quality of images recorded by the camera [11-13]. Additionally, cars or their parts (e.g. license plates) can be dirty even in favorable weather conditions, which may pose a challenge for some of the proposed subsystems. The system has not been tested in these and similar situations, which is a limitation of the present study.

The system was set up on a Raspberry Pi 4 which can process one frame per second (1 fps) from the available real-time video feed. This system speed was sufficient as there were no quick vehicle changes in front of the camera. Higher system speeds may be achieved by replacing the Raspberry Pi computer with a Jetson Nano, as its GPU would significantly speed up the overall system [72]. Although conducted in similar, but not identical settings (hardware + algorithms), a benchmark comparison between the speed of the Tiny YOLO v3 algorithm running on a Raspberry Pi 3 and a Jetson Nano computer confirmed this assumption [73]. More specifically, on a Raspberry Pi computer, this algorithm ran at 0.5 fps, while the Jetson Nano ran at 25 fps [73], which is a 50-fold acceleration. Car detection

using the YOLO architecture yielded good results in the proposed system, but other neural network-based architectures may also be used to achieve higher accuracy, albeit this would slow down the system. The replacement would also improve color classification results. Lastly, another potential area for improvement is replacing the OpenCV-based license plate detection method with a detector (e.g. a neural network-based system [74]) to ensure greater accuracy and better handle more realistic operating conditions, such as a partially dirty license plate. Naturally, the existing plate detection and recognition methods may also be improved and extended by adding image processing algorithms, such as morphological operations to extract plate contours.

Based on the above, it may be concluded that the proposed system works very well in realistic, although idealized conditions, but it has not been tested in more challenging scenarios that may occur in routine use in traffic (e.g., rain, direct light, etc.). Therefore, such scenarios need to be included in the testing image set to further develop and improve the system in accordance with the obtained results and observed shortcomings. Currently, the system's major shortcomings have to do with plate detection and recognition, especially at higher speeds, which additionally highlights the need to increase the low processing speed of 1 frame per second. However, the promising accuracy and low cost of this automatic system for the detection and classification of cars in traffic laid the foundations for its continuing development, including the development of a neural network for license plate recognition. With some minor modifications, the system could also be used in car parks for automatic ticket payment, mounted on traffic light control systems, as well as in other similar applications.

Provenance: Submitted. This manuscript is based on the master's thesis by Niko Bralić, deposited in the Dabar repository (https://urn.nsk.hr/urn:nbn:hr:179:151216).

Peer review: Externally peer reviewed.

Received: 12 December 2021 / **Accepted:** 29 August 2022 / **Published online:** 31 October 2022.

Funding: This research received no specific grant from any funding agency in public, commercial or not-for-profit sectors.

Authorship declaration: NB participated in the definition of a work topic and conducted all the experimental parts of the work and thus contributed to the collection, analysis, implementation of the estimate, and interpretation of the data. He wrote the first version of the manuscript and contributed to the revisions. JM devised the original work topic and contributed to the experimental design, research concepts, data collection, analysis and interpretation. He participated in manuscript revisions.

Competing interests: The authors completed the ICMJE Unified Competing Interest form (available upon request from the corresponding author), and declare no conflicts of interest.

Additional material: This article contains electronic supplementary material which is available for download at **st-open.unist.hr**.

ORCID

Josip Musić https://orcid.org/0000-0002-9185-5762

References

1. Singh, S. Critical reasons for crashes investigated in the national motor vehicle crash causation survey: U.S. Department of Transportation - National Highway Traffic Safety Administration; 2015. 3p. Report No.: DOT HS 812 506.
2. European Commission - European Road Safety Observatory. Annual Accident Report 2018. Brussels, Belgium: Directorate-General for Transport; 2018. 86 p.
3. Khan MQ, Lee S. A Comprehensive Survey of Driving Monitoring and Assistance Systems. Sensors. 2019;19(11):2574.
4. Swedish Transport Administration (Trafikverket). Road Safety: Vision Zero on the Move. Borlange, Sweden: Swedish Transport Administration (Trafikverket); 2015. 20 p.
5. U.S. Department of Transportation - National Highway Traffic Safety Administration. Automated Driving Systems 2.0: A Vision for Safety. Washington DC, USA: U.S. Department of Transportation - National Highway Traffic Safety Administration; 2017. 36 p. DOT HS 812 442.
6. Krasner G, Katz E. Automatic parking identification and vehicle guidance with road awareness. In Keren N,Reichman A,editor. ICSEE 2016: Proceedings of the 2016 IEEE International Conference on the Science of Electrical Engineering; 2016 Nov 16-18; Eilat, Israel. IEEE; 2016. p. 1-5.
7. Insurance Institute for Highway Safety - Highway Loss Data Institute. Real-world benefits of crash avoidance technologies [Internet]. Arlington, USA: Insurance Institute for Highway Safety - Highway Loss Data Institute; December 2020 [cited 2021 Oct 29]. 1 p. Available from: https://www.iihs.org/media/259e5bbd-f859-42a7-bd54-3888f7a2d3ef/e9boUQ/Topics/ADVANCED%20DRIVER%20ASSISTANCE/IIHS-real-world-CA-benefits.pdf.
8. Thomas E, McCrudden C, Wharton Z, Behera A. Perception of autonomous vehicles by the modern society: a survey. IET Intell Transp Sy. 2020;14(10):1228-39.
9. U.S. Department of Transportation - National Highway Traffic Safety Administration. Use of Advanced In-Vehicle Technology By Young and Older Early Adopters; Selected Results from Five Technology Surveys. Washington DC, USA: U.S. Department of Transportation - National Highway Traffic Safety Administration; 2008. 76 p. DOT HS 811 004.
10. Nastjuk I, Herrenkind B, Marrone M, Brendel A, Kolbe L. What drives the acceptance of autonomous driving? An investigation of acceptance factors from an end-user's perspective. Technol Forecast Soc Change. 2020;161: 120319.
11. Borrego-Carazo J, Castells-Rufus D, Biempica E, Carrabina J. Resource-Constrained Machine Learning for ADAS: A Systematic Review. IEEE Access. 2020;8: 40573-98.
12. Loce R, Bernal E, Wu W, Bala R. Computer vision in roadway transportation systems: a survey. J Electron Imaging. 2013;22:041121.
13. Horgan J, Hugues C, McDonald J, Yogamani S. Vision-based Driver Assistance Systems: Survey, Taxonomy and Advances. In: O'Conner L, editor. Smart Mobility for Safety and Sustainability. ITSC 2015: Proceedings of the 18th IEEE International Conference on Intelligent Transportation Systems; 2015 Sep 15-18; Canary Islands, Spain: IEEE; 2015. p. 2032-39.
14. Armstrong K, Livingstone K, Wilson A, Watson B, Watson A, Barraclough P. Prevalence of unregistered driving and associated driver behaviours in Queensland, In: McNaught, H, editor. ACRS 2012: Proceedings of the Australasian Road Safety Research, Policing and Education Conference; 2012 Oct 4-6; Wellington, New Zealand: TRAFINZ; 2012, p. 1-10.
15. Younglove T, Malcolm C, Durbin TD, Smith MR, Ayala A, Kidd S. Unregistration Rates for On-Road Vehicles in California. J Transp Stat. 2004;7(23):#1.
16. Ministry of the Interior of the Republic of Croatia. Bilten o sigurnosti cestovnog prometa u 2020. [in Croatian]. Zagreb, Croatia: Ministry of the Interior of the Republic of Croatia; 2021. [cited 2021 Oct 29] 142 p. Available from: https://mup.gov.hr/UserDocsImages/dokumenti/bilteni/Bilten_o_sigurnosti_cestovnog_prometa_2020.pdf.
17. Dark M. Roadside survey of vehicle observations [Internet]. London, UK: Department for Transport; 2017 Nov [cited 2021 Oct 29]. 7 p. Available from: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/659918/vehicle-excise-duty-evasion-statistics-2017.pdf.

18. PyImageSearch. OpenCV: Automatic License/Number Plate Recognition (ANPR) with Python [homepage on the Internet]. 2020. [cited 2021 Oct 29]. Available from: https://www.pyimagesearch.com/2020/09/21/opencv-automatic-license-number-plate-recognition-anpr-with-python/.

19. Ministry of the Interior of the Republic of Croatia [homepage on the Internet]. Automatski sustav za prepoznavanje i očitavanje registarskih pločica pod nazivom „Nero-ANPR“ sustav [in Croatian]; 2016 [cited 2021 Nov 2]; Available from: https://mup.gov.hr/policijske-uprave/automatski-sustav-za-prepoznavanje-i-ocitovanje-registarskih-plocica-pod-nazivom-nero-anpr-sustav/239050.

20. Bratković F. Primjena pametnih kamera u inteligentnim transportnim sustavima [online thesis – in Croatian]. Zagreb: Faculty of Transport and Traffic Sciences, University of Zagreb; 2020 [cited 2021 Nov 2]. Available from: https://repozitorij.fpz.unizg.hr/islandora/object/fpz%3A2083/datastream/PDF/view.

21. Poslovni dnevnik [homepage on the Internet Internet] Novom tehnologijom MUP identificira neregistrirana vozila i sankcionira prekršitelje [in Croatian] 2021 May 19 [cited 2021 Nov 2]; Available from: https://www.poslovni.hr/hrvatska/novom-tehnologijom-mup-identificira-neregistrirana-vozila-i-sankcionira-prekrsitelje-4287807.

22. NSW Police Force Public Affairs Branch – Office of the Commissioner. Annual report 2016-17. Sydney, Australia: NSW Police Force Public Affairs Branch – Office of the Commissioner; 2017. 124 p. ISSN: 1832-3472.

23. Lum C, Hibdon J, Cave B, Koper C, Merola L. License plate reader (LPR) police patrols in crime hot spots: an experimental evaluation in two adjacent jurisdictions. J Exp Criminol. 2011;7:321-345.

24. Milaj J, Ritsema van Eck G. Capturing licence plates: police-citizen interaction apps from an EU data protection perspective. Int Rev Law. Comput Technol. 2020;34(1):1-21.

25. Kovacic K, Ivanjko E, Gold H. Computer Vision Systems in Road Vehicles: A Review. In: Loncaric S, Segvic S, editors. CCVW 2013: Proceedings of the Croatian Computer Vision Workshop, Year 1; 2013 Sep 19; Zagreb, Croatia: University of Zagreb; 2013. p. 25-30.

26. Butt MA, Khattak AM, Shafique S, Hayat B, Abid S, Kim KI, et al. Convolutional Neural Networks Based Vehicle Classification in Adverse Illuminous Conditions for Intelligent Transportation Systems. Complexity. 2021;2021:#6644861.

27. Jiao L, Zhao J. A Survey on the New Generation of Deep Learning in Image processing. IEEE Access. 2019;7:172231-63.

28. Jiao L, Zhang F, Liu F, Yang S, Li L, Feng Z, et al. A Survey of Deep Learning-Based Object Detection. IEEE Access. 2019;7:128837-68.

29. Tian Y, Gelernter J, Wang X, Chen W, Gao J, Zhang Y, et al. Lane marking detection via deep convolutional neural network. Neurocomputing. 2018;280:46-55.

30. Brunetti A, Buongiorno D, Trotta G, Bevilacqua V. Computer vision and deep learning techniques for pedestrian detection and tracking: A survey. Neurocomputing. 2018;300:17-33.

31. Yurtsever E, Lambert J, Carballo A, Takeda K. A Survey of Autonomous Driving: Common Practices and Emerging Technologies. IEEE Access, 2020;8:58443-69.

32. Gupta A, Anpalagan A, Guan L, Khwaja A. Deep learning for object detection and scene perception in self-driving cars: Survey, challenges and open issues. Array. 2021;10:100057.

33. Goodfellow I, Bengio J. Deep Learning. 1st ed. MIT Press: Cambridge; 2016.

34. Redmon J, Divvala S, Girshick R, Farhadi A. You only look once: Unified, real-time object detection. arXiv 1506.02640 [Preprint]. 2016 [cited 2021 Nov 2]. Available from: https://arxiv.org/abs/1506.02640.

35. Ren S, He K, Girhick R, Sun J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In: Cortes C, Lawrence N, Lee D, Sugiyama M, Garnett R, editors. NIPS 2015: Proceedings of the 28th Conference on Neural Information Processing Systems: Advances in Neural Information Processing Systems; 2015 Dec 7-12; Montreal, Canada; Curran Associates, Inc.

36. Dai J, Li Y, He K, Sun J. R-FCN: Object detection via region-based fully convolutional networks. Adv Neural Inf Process Syst. 2016: 379-87.

37. Kamal A. YOLO, YOLOv2 and YOLOv3: All You want to know [homepage on the Internet]. 2019 [cited 2021 Nov 2]. Available from: https://amrokamal-47691.medium.com/yolo-yolov2-and-yolov3-all-you-want-to-know-7e3e92dc4899.

38. Soviany P, Ionescu R T. Optimizing the Trade-off between Single-Stage and Two-Stage Deep Object Detectors using Image Difficulty Prediction. SYNASC 2018: Proceedings of the 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing; 2018 Sept 20-23; Timisoara, Romania; IEEE Computer Society; 2018, p. 209-14.

39. Redmon J, Farhadi A. YOLO9000: Better, Faster, Stronger. In: O'Conner L, editor. CVPR 2017: Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition; 2017 Jul 21-26; Honolulu, USA: IEEE; 2017. p. 6517-6525.

40. Redmon J, Farhadi A. YOLOv3: An Incremental Improvement. arXiv 1804.02767 [Preprint]. 2018 [cited 2021 Nov 2]. Available from: https://arxiv.org/abs/1804.02767.

41. Bochkovskiy A, Wang CY, Liao HYM. YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv 2004.10934 [Preprint]. 2020 [cited 2021 Nov 2]. Available from: https://arxiv.org/abs/2004.10934.

42. Wang C-Y, Liao H-Y M, Yeh I-H, Wu Y-H, Chen P-Y, Hsieh J-W. CSPNet: A new backbone that can Enhance Learning Capability of CNN. arXiv 1911.11929 [Preprint]. 2019 [cited 2022 Mar 23]. Available from: https://arxiv.org/pdf/1911.11929.pdf.

43. Liu S, Qi L, Qin H, Shi J, Jia J. Path Aggregation Network for Instance Segmentation. arXiv 1803.01534 [Preprint]. 2018 [cited 2022 Mar 23]. Available from: https://arxiv.org/pdf/1803.01534.pdf.

44. Mallock S. CPU performance Comparison of OpenCV and other Deep Learning Frameworks [homepage on the Internet]. LearnOpenCV; 2018 [cited 2022 Mar 23]. Available from: https://learnopencv.com/cpu-performance-comparison-of-opencv-and-other-deep-learning-frameworks/.

45. LearnOpenCV. 2020. Deep Learning with OpenCV DNN Module: A Definitive Guide [homepage on the Internet]. [cited 2021 Nov 2]. Available from: https://learnopencv.com/deep-learning-with-opencvs-dnn-module-a-definitive-guide/.

46. Forson E. Understanding SSD MultiBox [homepage on the Internet]. Towards Data Science; 2017 [cited 2021 Nov 2]. Available from: https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab.

47. Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, et al. SSD: Single Shot MultiBox Detector. In: Leibe B, Matas J, Sebe N, Welling M, editors. Computer Vision – ECCV 2016: Proceedings of the 14th European Conference; 2016 Oct 11-14; Amsterdam, The Netherlands: Springer; 2016. p. 21-37.

48. TensorFlow [homepage on the Internet]. [place unknown: TensorFlow]; 2021. [cited 2021 Nov 2]. Available from: https://www.tensorflow.org/.

49. TensorFlow Lite [homepage on the Internet]. [place unknown: TensorFlow]; 2021. [cited 2021 Nov 2]. Available from: https://www.tensorflow.org/lite.

50. PyImageSearch. [homepage on the Internet]. OpenCV OCR and text recognition with Tesseract 2020. [cited 2021 Nov 2]. Available from: https://www.pyimagesearch.com/2018/09/17/opencv-ocr-and-text-recognition-with-tesseract/.

51. Color recognition. GitHub [homepage on the Internet]. 2021. [cited 2021 Nov 2]. Available from: https://github.com/ahmetozlu/color_recognition.

52. Open images dataset. [database on the Internet] 2020. Open Images Dataset V6. [cited 2021 Nov 2]. Available from: https://storage.googleapis.com/openimages/web/factsfigures.html.

53. LabelImg. GitHub [homepage on the Internet]. 2021. [cited 2021 Nov 2]. Available from: https://github.com/tzutalin/labelImg.

54. Yamashita R, Nishio M, Gian Do RK, Togashi K. Convolutional neural networks: an overview and application in radiology. Insight Imaging. 2018:9:611-29.

55. Colaboratory [homepage on the Internet]. [place unknown: Google]; 2021 [cited 2021 Nov 2]. Available from: https://research.google.com/colaboratory/faq.html.

56. Wang R, Wang Z, Xu Z, Wang C, Li Q, Zhang Y, Li H. A Real-Time Object Detector for Autonomous Vehicles Based on YOLOv4. Comput Intell Neurosci. 2021:2021:9218137.

57. Shashirangana J, Padmasiri H, Meedeniya D, Perera C. Automated License Plate Recognition: A Survey on Methods and Techniques. IEEE Access. 2021:9:11203-25.

58. Naveed Mufti L, Ali Shah S A. Automatic Number Plate Recognition: A Detailed Survey of Relevant Algorithms. Sensors. 2021:21(9):3028.

59. Romić K, Galić I, Baumgartner A. Character recognition based on region pixel concentration for license plate identification. Tehnički vjesnik. 2012:19(2):321-25.

60. Henry C, Yoon Ahn S, Lee S. Multinational License Plate Recognition Using Generalized Character Sequence Detection. IEEE Access. 2020:8:35185-99.

61. Novosel J. Sustav računalnog vida za automatsko prepoznavanje vozila u svrhu nadzora prometa [online thesis – in Croatian]. Zagreb. Faculty of Electrical Engineering and Computing, University of Zagreb; 2011 [cited 2022 Mar 28]. Available from: https://apps.unizg.hr/rektorova-nagrada/javno/stari-radovi/1158/preuzmi.

62. Ni Z, Tingna L, Li K, Bai Y, Zhu Z. Real-time vehicle detection and computer intelligent recognition through improved YOLOv4. Journal of Physics: Conference Series. 2021:2083:042006.

63. Sindhu V. Vehicle Identification from Traffic Video Surveillance Using YOLOv4. ICICCS 2021: Proceedings of the 5th International Conference on Intelligent Computing and Control Systems; 2021 May 6-8; Madurai, India. IEEE; 20121. p. 1768-75.

64. Alsanabani A, Saeed S, Al-Mkhlafi M, Albishari M. A Low Cost and Real Time Vehicle Detection Using Enhanced YOLOv4-Tiny. ICAICA 2021: Proceedings of the IEEE International Conference on Artificial Intelligence and Computer Applications; 2021 Jun 28-30; Dalian, China. IEEE; 2021. p. 372-77.

65. Tran VH, Dang LHH, Nguyen CN, Le NHL, Bui KP, Dam KT, et. al. Real-time and Robust System for Counting Movement-Specific Vehicle at Crowded Intersections. CVPRW 2021: Proceedings of the 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops; 2021 Jun 19-25; Nashville, USA. IEEE; 2021. p. 4223-30.

66. Lin C, Yeh C, Yeh C. Real-time vehicle color identification for surveillance videos. CONIELECOMP 2014: Proceedings of the 2014 International Conference on Electronics, Communications and Computers; 2014 Feb 26-28; Cholula, Mexico. IEEE; 2014. p. 59-64.

67. Brown L. Example-Based Color Vehicle Retrieval for Surveillance. AVSS2010: Proceedings of the 7th IEEE International Conference on Advanced Video and Signal Based Surveillance; 2010 Aug 29 - Sept 1; Boston, USA. IEEE; 2010. p. 91-96.

68. Wang Y, Han C, Hsieh C, Fan K. Vehicle color classification using manifold learning methods from urban surveillance videos. EURASIP J Image Video Process. 2014;2014(1).

69. Lee HJ, Ullah I, Wan W, Gao Y, Fang Z. Real-Time Vehicle Make and Model Recognition with the Residual SqueezeNet Architecture. Sensors. 2021: 19(5):982.

70. Ni X, Huttunen H. Vehicle Attribute Recognition by Appearance: Computer Vision Methods for Vehicle Type, Make and Model Classification. J Signal Process Syst. 2021:19: 357-68.

71. Spectrico. Car Make & Model recognition [homepage on the Internet]. Spectrico; 2021 [cited 2022 Mar 31]. Available from: http://spectrico.com/car-make-model-recognition.html.

72. Lang L, Xu K, Zhang Q, Wang D. Fast and Accurate Object Detection in Remote Sensing Images Based on Lightweight Deep Neural Network. Sensors. 2021;21(16):5460.

73. Jetson Nano: Deep Learning Inference Benchmarks. nVidia Developer [homepage on the Internet]. 2021. [cited 2022 Mar 31]. Available from: https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks.

74. Selmi Z, Ben Halima M, Pal U, Alimi MA. DELP-DAR system for license plate detection and recognition. Pattern Recognit. Lett. 2020;129:213-23.