# An Automatic Car License Plate Recognition System 

Ionuț-Andrei Pătrăuceanu, Adrian Iftene, Daniela Gîfu<br>Faculty of Computer Science, "Alexandru Ioan Cuza" University of Iași, Romania<br>\{andrei.patrauceanu, adiftene, daniela.gifu\}@info.uaic.ro


#### Abstract

Taking into account the increase in the number of cars and, implicitly, the traffic problems, an automatic license plate recognition (ALPR) system becomes a significant task in smart surveillance and transportation. The interest for the implementation and integration of ALPR technology into the daily security operations is visible, although there are still insufficiently resolved issues. They are based on either the help of image processing techniques or deep learning techniques. More precisely, they are using object detection algorithms such as You Only Look Once (YOLO). In order to obtain a system capable of detecting the license plates, we propose a method using YOLOv4. This system achieved impressive frames per second, the results show that we are talking about a method that is relatively insensitive to background variations.


## Author Keywords

Image classification, OCR, YOLOv4, Usability tests.

## ACM Classification Keywords

H.5.2. Information interfaces and presentation (e.g., HCI): User Interfaces. H.3.2. Information Storage and Retrieval: Information Storage.

## General Terms

Human Factors; Design.
DOI: 10.37789/rochi.2021.1.1.6

## INTRODUCTION

Annually, the number of cars increases rapidly (e.g., between 2015 and 2019, the Europe's car fleet increased by about 8 million units). Consequently, car monitoring systems should be widely implemented, being able to automate processes that normally depend on a worker [11]. Moreover, this would discourage acts that are against the law, such as: leaving the scene of an accident, kidnapping, car theft etc. [22]. Automatic License Plate Recognition (ALPR), invented in 1976 by the UK's Police Scientific Development Branch [14], represents a field that acquired importance to many traffic related applications such as road traffic monitoring [2]. In fact, ALPR plays a major role in monitoring of traffic rules and maintaining law enforcement on public roads [17] by their number-plates using recognition techniques [6;12]. We know too well that cars can be monitored based on their license plates. Delays, traffic jams, thefts, etc. could be prevented if the control entities (e.g., police) were equipped with intelligent systems for detecting and recognizing registration numbers [16]. The legitimate research question of this paper intends to answer: How does automatic license
plate recognition improve public safety? The main focus of this work is to build and implement a real time system, assisted by YOLOv4, able to automate surveillance processes in Romania. In fact, the computational limitations could be attained easily, as the application only requires a machine to run on and the media files to examine, given in the form of either images or video streams. No matter how well we choose the right algorithms, a successful ALPR system deployment may require additional hardware to maximize its accuracy. The rest of paper is organized as follows: Section 2 presents a short overview of license plate systems very useful in the control traffic and other ways (e.g., criminality detection), while Section 3 describes the new system based on YOLOv4. Section 4 briefly discusses the evaluation of this new system before drawing some conclusions in the last section.

## RELATED WORK

In Romania, ALPR systems are used only by the National Company for Road Infrastructure Management (CNAIR) in two cases: for verifying the payment of "Rovinieta" and the toll for the "Fetești-Cernavodă" bridge. CNAIR uses 106 fixed video cameras and 14 mobile video cameras, located on several roads to check who paid or not the Rovinieta or to identify drivers who did not acquire a toll. Basically, the CNAIR recognition system has the following functionalities: (1) photograph the registration number; (2) send real-time information to a computer center, where an operator confirms that the automatically registered number is correct; (3) the system checks if the driver paid one of the two taxes or both. In Europe, more precisely, in the United Kingdom, there have a network of 11,000 video cameras, which send about 50 million "readings" of registration numbers daily to the main center, where they are stored for approximately one year after their appearance, alongside metadata such as time, date and location regarding the place where the detection happened. The main purpose of this network is different than Romanian one. It is used to help criminality detection. An ALPR system can be implemented either on the basis of image processing techniques or on the basis of using deep learning techniques, more precisely using object detection algorithms such as: "Single Shot Detector" (SSD), "Fast Region-based Convolutional Neural Network" (Fast RCNN), "You Only Look Once" (YOLO) etc.

Regarding the first category, one of the most used and simple image processing techniques for detection is to use the Sobel filter together with the Hough transform [1; 9; 15; 23]. The Sobel filter is used to detect the edges of a picture. These are
represented by areas with strong contrasts, which can be identified by the major difference in intensity between a pixel and the one after it. The image we are working with is represented only by the edges captured in the initial picture. Considering the fact, that the registration number is a rectangle, the search is limited to identifying and filtering the representative matching shape for it. This can be done using Hough Transform, which is a feature extraction technique, specialized in identifying different types of patterns. This approach can be found in specialized libraries for computer vision, such as OpenCV.

The second category, based on neural networks [10; 18], teaches a computer to study as a human, through examples, using multi-layered neural networks. For a deep learning neural network, each hidden layer performs the training of a unique set of features that are based solely on the output of the layers that have appeared previously. In order to build the system proposed, we need a model in which accuracy and computing time play equally important roles. After carefully analyzing the options currently available, a YOLOv4 model seems to be the one that fits best the task, as it successfully combines the constants needed $[3 ; 4 ; 21]$.
Our solution uses a YOLO model to recognize the license plates in pictures and for identifying them from a live feed. Then, highlighted areas, detected from the image, will be preprocessed to be delivered to an OCR (Optical Character Recognition). The state-of-the-art (SOTA) selected in order to be used for the fulfillment of the OCR is Tesseract provided by Google. All components will ultimately be embedded in a PyQt5 graphical user interface, which is easy to use and attractive.

## DATASETS AND TECHNOLOGIES USED

Datasets - Open Images Dataset V4 is a collection of almost 9 million pictures, annotated in various ways and arranged in 600 distinct classes. The annotations provided are generally made by professionals in this field, thus ensuring the accuracy and consistency of the data. The images are extremely diverse and often contain several objects in a single picture ( 8.4 objects per image, on average). Analyzing all the available classes, the one of interest for us is "Vehicle registration plate", which can help achieve the proposed goal. The class is accessible using the following path within the hierarchy: Entity, Vehicle, Land vehicle, Auto part. The data collected from the interest class are divided into two: training data and test data. In addition to the actual images, they contain an annotation file, which indicates the exact area in the photo where the license plate is, in the shape of a rectangle. The number of annotations in each image can range from a single annotation to 8 in a picture, in the case of test images and can reach up to 28 , in terms of training images. The number of images in each batch is as follows: 5,368 images for training, in which there are 7,853 registration numbers and 386 pictures for test, in which there are 513 annotated plates.

OIDv4 ToolKit - Given that Open Images Dataset V6 is a huge dataset, its manual manipulation could only be done with great difficulty and with a gargantuan waste of time. OIDv4 ToolKit comes in support of this problem and offers a simple and easy to use solution for fast download of the necessary data [20].

Roboflow - Roboflow is a complex web Computer Vision developer framework, useful for the preprocessing phase of many machine learning algorithms. In data pre-processing, there are several steps involved such as resizing, changing image orientation, contrasting, followed by various data augmentation techniques. As a web application, it is very well synchronized and allows extremely fast uploading of voluminous data and its manipulation.
Darknet - Neural Network Framework - Darknet is an open-source neural network framework written in C and CUDA (Compute Unified Device Architecture). CUDA is fast, easy to install, and provides both CPU and GPU computation. Darknet is installed with only two optional dependencies: OpenCV, if users want a larger diversity of supported image types and CUDA, for GPU computation.

Google Colab - In 2018, Google launched Google Colab which is a web platform that allows programmers to write and execute Python code. In general, the service is used to train machine learning models that are time and resource consuming. The files created by this platform are called Colab notebooks, which are nothing but Jupyter notebooks, hosted by Colab. Jupyter Notebook is an open-source web application that can be used to create and share documents that contain live code, plots, images and text.
"tensorflow-yolov4-tflite" repository - Given that a YOLOv4 specific weight file cannot currently be used directly with TensorFlow, because in the latest release of YOLO, a new activation function, named Mish, was introduced, there was a need for a transformation to happen. That being said, the repository at hand contains the implementation for a parser that transforms YOLOv4specific .weights files to a format that is accepted by Tensorflow. Thus, making use of the code provided, the .weights files have been converted to.$p b$ files.

OpenCV - OpenCV (Open Source Computer Vision Library) is an open source library, written in C and $\mathrm{C}++$. The library has been designed to provide computational efficiency for real-time computer vision applications.
TensorFlow - Tensorflow is an open-source library that contains algorithms for numerical calculations and implementations of complex machine learning algorithms and neural networks. The library is easy to use but also very fast, the source code being written in $\mathrm{C}++$, while the interfaces that call the source code being written in Python. Tensorflow helps in training complex neural networks which can classify different objects such as: handwritten numbers, images, etc.

Tesseract - Tesseract is an OCR (Optical Character Recognition) engine for automatic character recognition of different types of documents: printed, handwritten and typed [7, 19]. Before this technology was available, the only way to digitize a text was to rewrite it manually, in its entirety. This process, in addition to being massively timeconsuming, also led to inconsistencies and typos.
PyQt5 - PyQt is a GUI (Graphical User Interface) widget toolkit and represents a Python interface for Qt, one of the most popular cross-platform GUI libraries. In other words, PyQt serves as a link between the Python programming language and the Qt library. The most stable and the one constantly receiving new improvements is PyQt5, so it was chosen for the purpose of this work.

## THE MODELS

## Architecture

Object detection is one of the classic problems in computer vision that should not be confused with classification, as it is much more complex. The latter can recognize objects but does not indicate the area in which they are located. In addition, classification cannot work on frames that contain multiple objects.
YOLO ("You only look once") is part of the category of systems that allow the detection of several objects within the same frame. It is popular because, in addition to achieving high accuracy, it can also run-in real time. The algorithm is named so because it requires a single forward propagation through the neural network in order to produce results. After non-max suppression, the algorithm exposes the objects present in the frame, surrounded by delimitation boxes, ensuring that they are not detected several times. Models for object detection can be divided into two categories (1) TwoStage Detectors, (2) One-Stage Detectors. YOLO, as expected, is part of the category of one-stage detection models (also called one-shot detection models), which do not need the last preliminary step, present in the case of twostage detectors. This last phase, within these detectors, is a costly one from a computational point of view, slowing down the detection process. This happens because it involves the detection of important regions and then, based on them, the detection of the objects sought. Thus, after receiving the input, there are 3 phases for fulfilling the purpose of the model (1) Backbone, (2) Neck, (3) Dense Prediction (Head).
Backbone is a neural network made up mostly of convolutional layers. Its purpose is to extract the essential features from the input data, having a particularly important role in the final performance of the model. YOLO can be used together with several types of backbone, the ones that are widely used being YOLOv4 tiny and Darknet53:

- Darknet53 has 53 convolutional layers, being much more precise but slower compared to the architecture used by YOLOv4 tiny.
- YOLOv4 tiny has only 29 convolutional layers, so it is less accurate, but faster and suitable for devices that do not have a very high computational capacity.

Both versions adopt a structure called Cross Stage Partial (CSP) which is inspired by the DenseNet architecture.

Neck - The next step to achieve the proposed goal is to mix and combine the features extracted in the backbone in the "Neck" block to prepare them for the detection step. The authors of YOLOv4 propose in this block a series of tools already used in the world of deep learning algorithms, but modified to better fit the needs of the model and to increase its final accuracy.
After CSPDarknet53, an SPP (Spatial Pyramid Pooling) block is added in order to increase the receptive field and separate out the most important features extracted by the backbone. Original SPP arose from the need to have an approach that can have multi-scale input images for training because, in the past, only fixed-size images could be used due to the presence of the fully connected layers. The problem with these types of layers started from the fact that for them we have to specify which the input layers and the output layers are. Before these layers, there is always a series of convolutional or max pool layers that need to be flattened in order to be used as input. This last step represents the bottleneck in being able to receive inputs of variable size, since the input of a fully connected layer must remain of fixed length. The first deep learning models used simple networks that manipulate the input data through a succession of layers, each layer having as input data the output of the previous one. Due to the evolution of models of this type, it has been observed that as we progress towards the end of the network, some information that may help fine-tune the prediction may be lost. In response to this problem, the authors of the YOLOv4 model used PaNet, whose architecture allows better information propagation. The components of the neck typically flow up and down among layers and connect only the few layers at the end of the convolutional network. One aspect to note is that in the original PaNet implementation, low-level information is brought to the top layers by adding together the current layer and information from the previous layer to form a new vector. Instead, the authors of YOLOv4, propose a modified variant that uses a new vector created by concatenating the input and the vector from the previous layer.

## Head - the last part of the model deals with the following:

- Locating bounding boxes;
- Classifying what is inside each box.

Due to the architecture present in the neck, the head's input will contain spatially rich information from the bottom-up stream and the semantic rich information from the top-down stream. The output of the algorithm is composed of 4 values that are meant to describe the predicted bounding box ( $x, y$, $h, w)$ and the probability of $k$ classes and an additional one,
for the background. An anchor box is predefined and has an aspect ratio set, which is defined beforehand by running, on the dataset, a k-means clustering.

## YOLOv4 tiny particularities

In terms of activation function, YOLOv4 uses Mish, while YOLOv4 tiny uses Leaky ReLU, which is a function based on maximum, being less computationally expensive than the one used by the first model. In the part of feature fusion, Yolov4 tiny differs by removing the SPP and using only the first step within the PaNet, called FAN (Feature Pyramid Network).

## Training process and results

The process of creating models is started by preparing the data. Thus, using the OIDv4 ToolKit, the training and testing datasets are extracted. After extracting all the necessary images from the Open Images Dataset V4, we have attached for each one a .txt file with the same name as the picture, in which all the present registration plates are annotated in a classic format. Basically, for each license plate, in order to identify it, in the annotation files, the top left corner and the bottom right corner of the bounding box are mentioned. However, the annotation format required by YOLO is different, based on the center of the rectangle and its width and height. Therefore, to make the transition from the first format to the desired one, the Roboflow was used. After this step, we can start using the Darknet framework together with the work environment from Google Colab. The steps to be taken to achieve the proposed goal can be summarized as follows:

1. The first step is to set the use of a GPU hardware accelerator in Google Colab in order to receive the results in the shortest time that the work environment can provide.
2. The second step is cloning and building the Darknet framework within Google Colab.
3. After this, it is necessary that both the training and the test images, together with the files adapted for the format required by YOLOv4 to be uploaded in Google Drive, in two .zip archives called obj and test.
4. The next step is to create the files needed to start the training:

- Customized configuration files with .cfg extension, one for each model (YOLOv4 and YOLOv4 tiny).
- obj.data and obj.name files; the first file contains information such as the number of classes on which the detection is made (in our case one, the class of the license plates), the location of the obj.names file, train.txt and test.txt files, as well as the location where the generated models will be saved.
- train.txt and test.txt files; files can be generated using a Python script and contain the paths of all previously uploaded images from the location where they were unzipped.

5. One last step, before starting the training process, is to download pre-trained weights for both YOLOv4 and the YOLOv4 tiny model. These are trained on the basis of the well-known data set called COCO, and can be used thanks to the process called transfer learning. Thereby, instead of randomly setting weights, we will use these ones. This saves a lot of time and helps minimize the number of computations required while training the custom model.
6. Finally, the training is launched, the results and the waiting time for each of them are presented in Table 1.

| Model | YOLOv4 | YOLOv4 tiny |
| :--- | :---: | :---: |
| Training duration | 9 hours and 4 <br> minutes | 2 hours and 12 <br> minutes |
| Batch size | 64 | 64 |
| Input size | $416 \times 416$ | $416 x 416$ |
| Activation function | Mish | Leaky ReLu |
| IoU | $50 \%$ | $50 \%$ |
| Iterations | 6000 | 2500 |
| mAP (Mean average <br> precision) | $91.58 \%$ | $78.20 \%$ |

Table 7. Models results.
Table 1 shows some aspects that are mentioned in the configuration files, such as the size of the input and the size of each batch. The batch size defines the number of samples that will be propagated together through the network. The training time is, as expected, different in the case of the two models, YoloV4 tiny being at least 4 times faster. In the case of the number of iterations, not all those mentioned in the table are useful for obtaining the final model used. This is due to the Darknet framework which uses the mAp evaluation metric, once every few hundred/thousand iterations to evaluate the model even during training, stopping the process when it starts to lose accuracy due to various causes. However, the framework saves the model at each evaluation, so that in the end we can be sure that we benefit from the best possible model. The metric mentioned previously is widely used, being found as an evaluation method for various models. Despite the name, the metric is not calculated by taking the average of precision values. The mAP compares the ground-truth bounding box to the detected box and returns a score. In these calculations, the IoU (intersection over union) index and the results obtained based on it are vital: false positive, false negative, true negative and true positive. In short, the IoU index takes into account the bounding box received from the model in response and the one we have in our data set. If the model answers a box that overlaps well enough with the original, it is classified as a valid answer. In our case, the percentage of overlap that must be met is at least $50 \%$. The accuracy of the two models can be improved by enlarging the ImageV4 data
set, with real images, which can be taken by anyone with a camera. These are more valuable in terms of learning, comparable to any augmented images that can be added.

## Recognition of the license plate

In order to detect the characters on the license plate we use Tesseract. Its creators claim that in order to have optimal results, the characters must be black, on a white background [5]. As it can be seen, the YOLO model gives us as a result an area in the photo, in the form of a rectangle, in which the license plate is framed (number 1 in Figure 1; from this moment the indications in this section will be made exclusively on this figure). Thus, a series of preprocessing steps are required to get rid of the noise in the photo and to deliver the letters to the OCR in the best condition. For these preprocessings we use the OpenCV library which helps fulfill the purpose.


Figure 1. Steps leading to detection.
In the first phase (number 2), the image will be converted to grayscale, in order to be delivered to the function that binarizes the photo, based on Otsu thresholding. Otsu's thresholding divides the pixels into foreground and background, based on calculations with the purpose of obtaining a binary image, in the best shape (number 3). The next step is to add a padding to the entire image to prepare it for identifying the contours. This step has the role of avoiding the situation in which a letter cannot be identified because it is too close to the margin of the photo (number 4). Evidently, after identifying the contours (number 5), many of the ones found are redundant, our next goal being to sort them and keep only the ones that fit the letters (number 6). The last two preprocessing phases are done based on the binarized image, but for a better visualization, the original image was kept in the figure. The identification of contours that are not relevant for detection is done by removing those that have less than $5 \%$ and more than $30 \%$ of the total width of the image. In addition, another criterion is that the contour be between $25 \%$ and $90 \%$ of the total height of the image. However, in order to minimize the risk of taking into account unnecessary data, a filter is applied, eliminating those contours in which the difference in height between them is greater than $15 \%$. These filters were fixed based on the careful analysis of a Romanian registration plate and the tests were made on multiple images. The next step is to take the
remaining contours and cut them, leaving only the letters and numbers (number 7). Finally, a padding is added to each character, because an improvement in accuracy has been observed thanks to this aspect. Regarding the configuration imposed on the Tesseract, it looks like this:

```
-c tessedit_char_whitelist = 0123456789ABCDE
FGHIJKLMNOP\overline{QRSTUVWXYZ}
--psm 10 --oem 3
```

The above instruction has the role of allowing only a series of characters to be detected (tessedit_char_whitelist), indicating that the detection is done only on one letter (psm) and that any internal OCR engine can be used (oem).

## THE APPLICATION

The model created for the detection and recognition of the license plate can be integrated in various applications that can simplify or automate vital processes for the safety and smooth running of daily life. In the following sections, we propose to present the implementation of a MVP (Minimum Viable Product) application capable of managing a parking lot and the access to it. The system aims to allow access in the parking lot to vehicles that are pre-registered in the system and to charge them accordingly, depending on the length of stay in it. The parking owner can customize the system settings such as: the fee for one hour in the parking lot and the number of places available in it. He can also register people who have access to the parking lot, recharge their accounts, edit people's identification data and generate various reports about vehicles that have transited the system. The data required for the operation of the application is stored in a relational database, PostgreSQL. The system will be embedded in a graphical PyQt5 application that is intuitive and does not allow actions that can lead to inconsistencies in the data saved in the database.

## Application Architecture

Owner page - The page has the role of adding a new owner in the system. Before entering the data into the database, a series of sanity checks are performed in order to prevent their inconsistency. These are made with the help of regular expressions, and if there are any problems with one of the completed fields, the user receives a warning message (see Figure 2 left).

Edit Owners' Information Page - The role of this page is to offer the option to edit information about owners added using the previous page or to permanently delete an owner from the system. At the top of the page, there exists a modified QTableWidget, in which all of the owners of the system are displayed. This table only allows the selection of rows and when the user selects one, a trigger automatically fills in the editable fields present below the table with the information about that person. Obviously, in this case too, the data given by the user is validated with the help of regular expressions and, if some fields do not respect the format imposed, the user is informed about it.


Figure 2. Screenshot from "Add Owner Page" (left) and "Modify Budget and Fee Page" (right).
Modify Budget and Fee Page - The role of this page is to modify additional data, vital for its proper functioning. It is divided into 3 sections as follows: "Modify the budget for an owner", "Modify the number of parking spaces" and "Modify the parking fee" (see Figure 2 right). Each of the three sections has a button or two associated with the following role:

1. "Get budget" button has the role of automatically filling, based on the registration number entered by the user, the budget associated with the respective car, within the "Current budget" field. "Modify budget" button modifies the car budget, if the user enters a valid value.
2. "Modify parking spaces" button triggers the update in the database of the number of parking spaces, if the data entered by the user is valid.
3. "Modify fee" button triggers the update in the database of the fee for one hour of parking, if the data entered by the user is valid.

Input Flow Page - The role of the page is to analyze the flow of entry into the parking lot and to start the necessary actions to save the data about the cars (see Figure 3 left). The page offers the user the option to select a video file, previously saved on the personal computer, with the help of the "Choose file" button. Next to it is the "Analyze file" button that triggers an algorithm that sends a frame for detection, once every 5 frames to the YOLOv4 tiny model. After this step, the Tesseract OCR identifies the characters on the license plate, and the result, if the sanity checks for a valid license plate are met, is checked in the database. At the same time, at the top of the page, the analyzed file will be broadcasted in a QVideoWidget. After the checks that take place in the database, 5 distinct messages can appear under the area where the analyzed video is running:

1. "Number not registered. ACCESS DENIED."
2. "Number recognized successfully. ACCESS ALLOWED."
3. "Balance negative. ACCESS DENIED."
4. "Car already inside the parking lot. Account still charged."
5. "Parking lot is full. ACCESS DENIED."


Figure 3. Screenshot from "Input Flow Page" (left) and "Output Flow Page" (right).
Output Flow Page - The role of the page is to analyze the cars leaving the parking lot, to charge the drivers accordingly and to notify them by email in case their budget becomes negative after their last visit (see Figure 3 right). The implemented payment system allows any owner registered who has a strictly positive balance to enter the parking lot. At the end of the visit, it is charged accordingly, allowing the balance to become negative. Moreover, if necessary, the system will notify the car owner that a top-up of the account is necessary, using the email address registered in the system, also letting them know that their access is being restricted on subsequent visits. The email notification service for drivers is implemented using the "SMTPLIB" library in Python which uses the SMTP protocol (Simple Mail Transfer Protocol), that handles sending emails and routing emails between mail servers. After each car leaving the parking lot, 4 messages can be displayed within the interface:

1. "Number not registered. Goodbye!"
2. "The fee is \{price\}. The new balance is \{new_credit\}. Please top up. Have a nice day!"
3. "The fee is \{price\}. The new balance is \{new_credit\}. Have a nice day!"
4. "An error occurred. Please contact the nearest operator."

In the case of messages 2 and 3, the "price" field will be replaced accordingly for each car with the amount it had to pay for the period it was parked in the parking lot, while the "new_credit" field will be replaced with the remaining amount after completing the payment for the last stay.
Live Stream Page - The page is not available in the MVP version, but it exists in order to be used for further development of the application. The page aims to be able to take live video streams for both the input flow and the output flow, acting similarly to page "Input Flow Page" and "Output Flow Page". The biggest challenge for implementing this page will be getting the live stream needed in order to test the features.

Reports Generator Page - The page has the role of generating reports on the flow of cars. It is divided into two sections: "Generate report from the past" and "Generate report with currently parked cars". The first section allows the user to set the date and time frame (start and end of the period) for generating the report. The "Choose directory" button gives the user the option to select the directory where he wants the report to be saved, while the "Generate report"
button starts the generation and finally saves a document called "report_past_parked_cars.pdf" at the location indicated by the user. The second section allows the user to generate a report with the cars that are currently in the parking lot. The buttons have the same functionality as the first section, but the document generated in this section will be called "report_currently_parking_cars.pdf". Both reports are built using FPDF which is a library for PDF document generation under Python. Compared to other PDF libraries, FPDF is simple, small and versatile, with advanced capabilities and is also easy to learn.

| ID CAR | ENTRANCE | EXIT | FEE | PHOTO |
| :--- | :--- | :--- | :--- | :--- |
| 19 | $2021-06-01$ <br> $01: 50: 37$ | 2021-06-01 <br> $01: 51: 15$ | 15 |  |
|  |  |  |  | NTOO JO |
|  |  |  |  |  |

Figure 4. Excerpt from the report on cars parked in the past.

## EXPERIMENTAL RESULTS

We performed usability tests similar with [8;13] and collected opinions about the application from two different perspectives: (1) user of the application (the owner of the parking lot whose duties are: registers people in the system, edits data about existing people, changes the number of parking spaces and the price for an hour in the parking lot, generates reports based on the flow of people who passed the parking lot) and (2) beneficiary of the application (the driver of the car, which is registered in advance by the owner of the parking lot and whose registration number is scanned at the entrance and exit of the parking lot), to see what can be improved or changed in the future in the application.
Methodology: The conducted usability test consisted of an introduction, tasks for the application, a short interview, and a post-test questionnaire. We instructed the participants to think out loud and express their thoughts during the test. After the task series that we communicated verbally to the participants, we gathered their assessment of the overall experience using the QUIS (The Questionnaire for User Interaction Satisfaction) scale. The tasks that users performed covered the options from the application and each session took around 5 minutes.
Participants: We collaborated for evaluation with 11 persons, with an average age of 21.9 years, 9 out of 11 holding a driving license. Their selection was random, the group being formed of 7 women and 4 men. All of them have previous experience with desktop applications. They received the application and their interactions with it were during one day.
Results: From our observation during the test sessions, users have successfully used and understood all the functionalities that the application offers. Participants were asked to rate different sentences about the application with a grade from 1 to 9 , where 1 means "Strongly disagree" and 9 means "Strongly agree". They also had to answer 4 open-ended
questions aimed at identifying: (1) the most useful component, (2) the component that can generate the most problems, (3) shortcomings of the current version and (4) the challenges that may arise in using such a system. All questions to be assessed with a grade were chosen such that to form a clear delimitation of strengths and weaknesses, based on the score obtained. In the case of open-ended questions, all this has been designed so that it can provide future directions for the application and highlight problems that may arise when using such a system.
Thus, from the perspective of an application user, the following questions obtained the average of the answers as follows: "Overall, I am satisfied with how easy it is to use this system." - 8.09, "I felt comfortable using this system." 7.36, "I believe I could become productive quickly using this system." - 8, "Whenever I made a mistake using the system, I could recover easily and quickly." - 8.09, "The organization of information on the system screens was clear." - 8.18, "The interface of this system was pleasant." -7.09, "This system has all the functions and capabilities I expect it to have." 7.54, "Overall, I am satisfied with this system." - 8.18.

Summarizing the open-ended questions, users consider that the most useful component of the application is the automatic payment system that streamlines processes that normally happen manually. Users expressed concern that the license plate detection and recognition system could not operate in optimal parameters if the registration number is damaged, dirty or if the weather is unfavorable outside. In future versions, respondents want the graphical interface to be improved, the "Live Page" page to be implemented and the application to be available on multiple operating systems.
Instead, from the perspective of a beneficiary of the application, the following questions obtained the average of the answers as follows: "I see the charging system better than the classic ticketing system." - 8.45 , "I see the system used at the same time with the classic ticketing system, not a replacement for it." - 4.45, "I think this system will be perceived negatively by people." - 3.54 , "I believe that in the medium and long term such a system would bring more problems than the classic ticketing system." - 3. In opinion of those who answered the questionnaire, the biggest challenges that the system can have are the errors that can occur in the automatic recognition of the license plate and people's reluctance to the technological advance required by the implementation of this system.

## CONCLUSIONS

This work describes two of the YOLO models together with the process of segmentation and detection of a license plate and their integration in an application that deals with the management of a parking lot. Basically, this approach offers multiple functionalities for the user. Regarding this application, the Live page must be implemented in the following versions, whether we are talking about testing on a real parking lot with an access barrier or in a controlled test
environment, such as one based on microcontrollers. In addition, for the efficiency of the detection and recognition of the registration plate, an additional system that first detects the movement within the supervised area can be implemented. Based on motion detection, it triggers operations that are more expensive in terms of resources used, with no need for the main model to process frames continuously.

## ACKNOWLEDGMENTS

This work was supported by project REVERT (taRgeted thErapy for adVanced colorEctal canceR paTients), Grant Agreement number: 848098, H2020-SC1-BHC-2018-2020/ H2020-SC1-2019-Two-Stage-RTD.

## REFERENCES

1. Agrawal, P. Vehicle Number Plate Detection Using Sobel Edge Detection Techniques by MATLAB, (2017).
2. Ahmed, M.J., Sarfraz, M., Zidouri, A., and Al-Khatib, W.G. License Plate Recognition System. Proceedings of the 2003 10th IEEE International Conference on Electronics, Circuits and Systems, 2 (2003).
3. Ajay, S., and Mahmood, A. Review of Deep Learning Algorithms and Architectures. IEEE Access PP 99: 1-1, (2019).
4. Anand, J., and Meva, D. A Comparative Study of Various Object Detection Algorithms and Performance Analysis. International Journal of Computer Sciences and Engineering 8, (2020), 158-163.
5. Badr, A., Abdelwahab, M.M., Thabet, A.M., and Abdelsadek, A.M. Automatic Number Plate Recognition System, (2011).
6. Bailey, D.G., Irecki, D., Lim, B.K., and Yand, L. Test Bed for Number Plate Recognition Applications. Proceedings of the $1^{s t}$ IEEE International Workshop on Electronic Design, Test and Applications (DELTA '02), IEEE Computer Society. (2002).
7. Boiangiu, C., Ioanitescu, R., and Dragomir, R. VotingBased OCR System. Journal of Information Systems \& Operations Management, (2016).
8. Chițaniuc, M., and Iftene, A. GeoAR - An Augmented Reality Application to Learn Geography. In Romanian Journal of Human-Computer Interaction 11, 2 (2018), 93-108.
9. Choong, Y.J., Lee, K.K., and Tan, C.C. License plate number detection and recognition using simplified linear-model. DOI: 10.31838/jcr.07.03.09. (2020)
10. Coca, G.L., Romanescu, S.C., Botez, S.M., and Iftene, A. Crack detection system in AWS Cloud using

Convolutional neural networks. In 24rd International Conference on Knowledge-Based and Intelligent Information \& Engineering Systems, Procedia Computer Science, 176, (2020), 400-409.
11.Morales-Alvarez, W., Sipele, O., Léberon, R., Tadjine, H.H., and Olaverri-Monreal, C. Automated Driving: A Literature Review of the Take over Request in Conditional Automation. Electronics 2020, 9, 2087
12. Mufti, N., and Shah, S.A.A. Automatic Number Plate Recognition: A Detailed Survey of Relevant Algorithms. Sensors 21, 3028, (2021).
13. Pinzariu, M.N., and Iftene, A. Sphero - Multiplayer Augmented Game (SMAUG). In International Conference on Human-Computer Interaction, Iasi, Romania, (2016), 46-49.
14. Roberts, D.J., and Casanova, M. Automated License Plate Recognition (ALPR) Use by Law Enforcement: Policy and Operational Guide, (2012).
15. Rudeš, H., and Grd, P. License Plate Detection for Preserving Privacy using Haar Classifiers. (2015).
16. Sankari, M., Bremananth, R., and Meena, C. A Robust Diverged Localization and Recognition of License Registration Characters. Int. J. Electr. Comput. Eng. 6, (2013), 1225-1232.
17. Sarfraz, M. Computer-Aided Intelligent Recognition Techniques and Applications, (2005).
18. Shrestha, A., Mahmood, A. Review of Deep Learning Algorithms and Architectures, 1-1, (2019).
19. Smith, R. An Overview of the Tesseract OCR Engine. In $9^{\text {th }}$ International Conference on Document Analysis and Recognition, (2007).
20.Vittorio, A. Toolkit to download and visualize single or multiple classes from the huge Open Images v4 dataset, (2018) https://github.com/EscVM/OIDv4_ToolKit.
21. Walsh, J., O'Mahony, N., Campbell, S., Carvalho, A. Krpalkova, L., Velasco H.G., Harapanahalli, S., and Riordan, D. Deep Learning vs. Traditional Computer Vision, (2019).
22. Woods, L. Automated Number Plate Recognition: Data Retention and the Protection of Privacy in Public Places. Journal of Information Rights Policy and Practice 2 (1), (2017).
23.Zhao, Y., Gu, J., Liu, C., Han, S., Gao, Y., and Hu, Q.M. License Plate Location Based on Haar-Like Cascade Classifiers and Edges, (2010).

