

BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM

Villamosmérnöki és Informatikai Kar

Computer Vision Systems Project Summary

Kürti Ádám HZUCDC

Pucsok Sándor PP22N9

2023

**Content**

[**1.** **Project Description** 3](#_Toc136267272)

[**2.** **Car detection algorithm** 4](#_Toc136267273)

[**2.1** **YOLO Algorithm** 4](#_Toc136267274)

[**2.2** **Car detection** 5](#_Toc136267275)

[**3.** **License plate detection and read** 6](#_Toc136267276)

[**4.** **Estimate Vehicle Speed** 8](#_Toc136267277)

[**5.** **Test** 9](#_Toc136267278)

[**6.** **Summary** 10](#_Toc136267279)

[**Literature** 11](#_Toc136267280)

1. **Project Description**

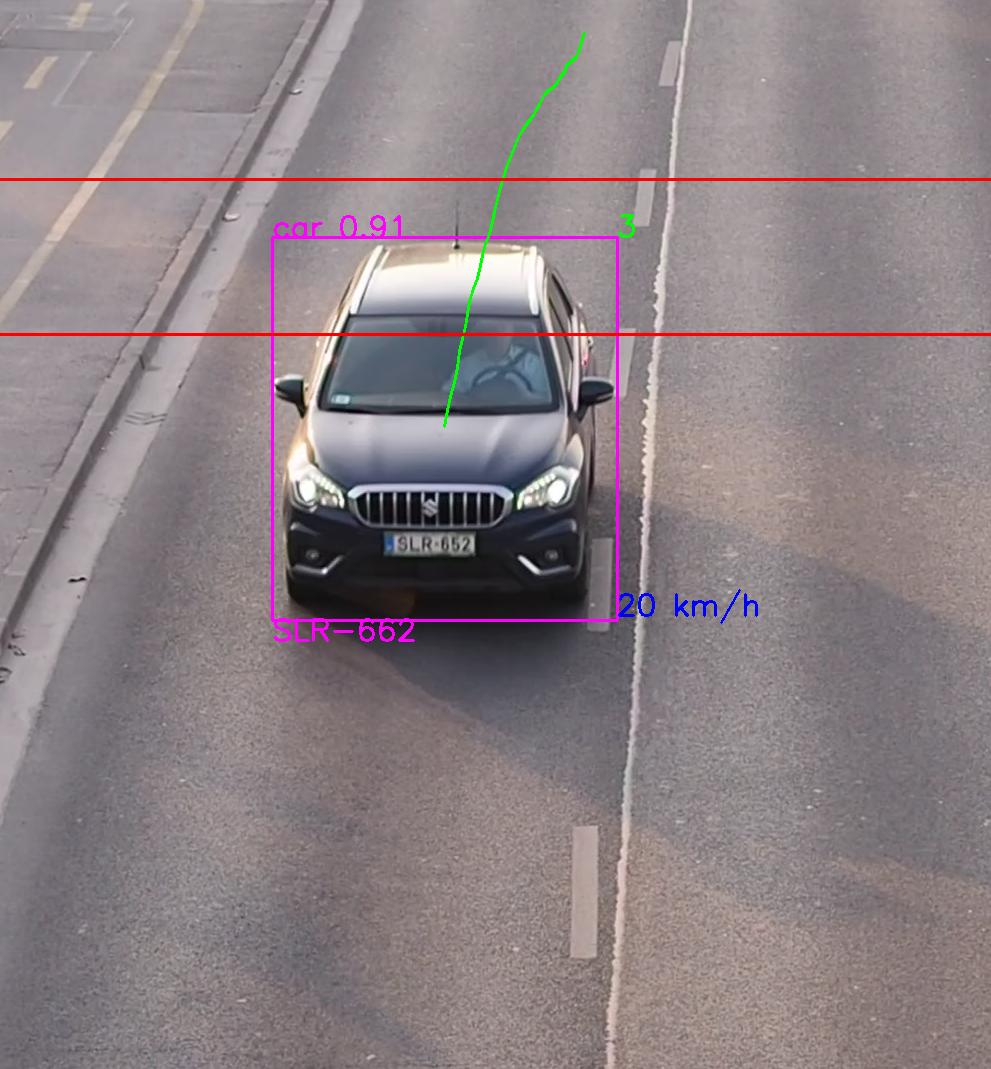
**The aim of this project is to develop an algorithm that can efficiently recognize and detect cars, their license plates and approximate their velocity in video recordings. These will be combined to form a speed camera.**

**The foundation of car detection lies in a vast dataset used during the training phase of machine learning algorithms. The algorithm is first trained using images containing a variety of vehicles and their corresponding labels. Subsequently, it becomes capable of recognizing learned patterns and features in new images, even those not present in the training data.**

**Our main idea was to use the following techniques to achieve the speed camera functionality:**

1. **Vehicle detection and cropping**
2. **License plate detection and cropping**
3. **License plate reading with optical character recognition**
4. **Vehicle velocity measurement using optical flow**

**In the following will we go through these techniques and present our solutions.**



**1**. figure: The speed camera algorithm in action

1. **Vehicle detection algorithm**

In this chapter we introduce the algorithm what we use for vehicle detection.

For object detection there are several widely used neural network architectures each with different speed and accuracy. The R-CNN (Region-based Convolutional Neural Network) family and the YOLO (You Only Look Once) family are two popular approaches for object detection in computer vision.

The R-CNN family consists of several variants, including R-CNN, Fast R-CNN, and Faster R-CNN. These models use a two-stage approach. First, they generate region proposals using selective search or similar algorithms. Then, each proposed region is classified and refined using a CNN. The R-CNN family is slower compared to YOLO models. It involves multiple passes through the CNN for each region proposal, which can be computationally expensive. The R-CNN family tends to provide higher accuracy in object detection tasks, especially for smaller objects. The two-stage approach allows for more precise localization and classification. The R-CNN family performs well in scenarios with a wide range of object sizes and densities. The two-stage approach helps in handling diverse objects effectively.

The YOLO family on the other hand, follows a one-stage architecture. YOLO divides the input image into a grid and predicts bounding boxes and class probabilities directly using a single pass through the network. YOLO models are known for their real-time object detection capabilities. They process the entire image in a single forward pass, resulting in faster inference times. YOLO models sacrifice some accuracy for speed. They prioritize detecting larger objects and can struggle with small objects or objects close together. However, newer versions of YOLO, such as YOLOv4, have improved accuracy by incorporating various techniques. YOLO models excel in detecting larger objects and objects with moderate density. They can struggle with small objects and densely packed scenes.

In summary, if accuracy is the primary concern and real-time performance is not critical, the R-CNN family is a good choice. On the other hand, if real-time performance is crucial, and some loss in accuracy is acceptable, the YOLO family is a more suitable option, that’s why we chose the YOLO family.

## **YOLO Algorithm**

You Only Look Once (YOLO) is one of the most popular model architectures and object detection algorithms. It uses one of the best neural network architectures to produce high accuracy and overall processing speed, which is the main reason for its popularity.

YOLO algorithm aims to predict a class of an object and the bounding box that defines the object location on the input image. It recognizes each bounding box using four numbers:

* Center of the bounding box ((bx, by)
* Width of the box (bw)
* Height of the box (bh)

In addition to that, YOLO predicts the corresponding number c for the predicted class as well as the probability of the prediction (Pc).

Let’s say that we have an image with 2 vehicles, one car and one truck. The first step that YOLO does is dividing the image into a grid.

With the existence of a grid, it’s possible to detect one object per grid cell instead of one object per image. For each grid cell, we can encode a vector that will describe the cell. For instance, the first cell from the top-left doesn’t have any object, and we describe it as:

where (Pc) is the probability of the object class, Bx and By are coordinates of the center of the bounding box, relative to the cell, Bx and By are width and height of the bounding box relative to the whole image, C1 and C2 are 0 or 1 depending on which class represents the bounding box (C1 for car and C2 for truck). Vector (C1, 1) consists of symbols ? because if the first component (Pc) is equal to zero, then the rest of the components can have random numbers are they are not taken into consideration.

Next, if we take the cell that contains the center of the blue bounding box with the cat, we’ll have a vector.

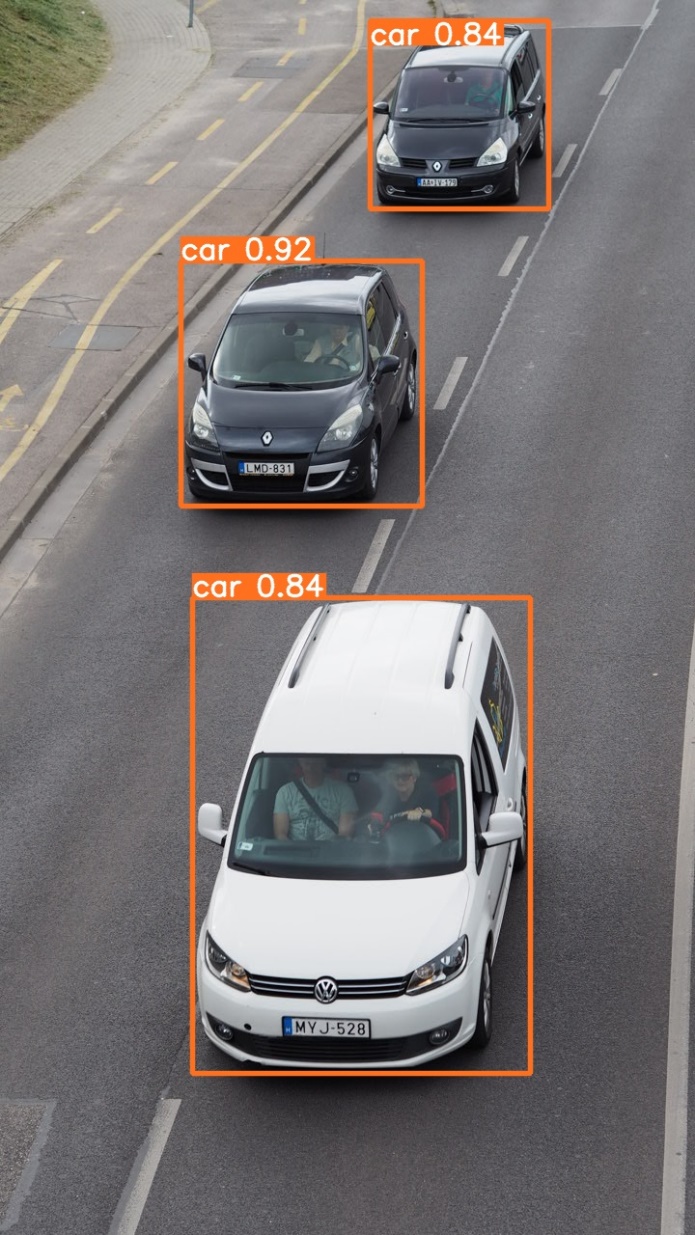
Following this procedure, if we define one vector for each grid cell, the whole image is represented with nine vectors with size 7 or 3x3x7 tensor. This means that in our data set, each image sample is labeled with one 3x3x7 tensor. Using that data set, we are able to create a training and test set and train the convolutional network, which is exactly how YOLO works [1].

* 1. **Vehicle detection**

As mentioned in the previous chapter, the detection was performed using the YOLO algorithm, which was able to achieve adequate accuracy to use it in our case.

First, we tried to train our own YOLO model for detecting various vehicle types. For this we use the [MIO-TCD](https://tcd.miovision.com/challenge/dataset.html) localization dataset, containing 137,743 images at different times of the day and different periods of the year by thousands of traffic cameras deployed all over Canada and the United States. It has 11 different labels for different vehicle types. The training code can be seen at [here](https://github.com/Kurtiadam/speedcam_cvs/blob/main/YOLO_vehicle_detection/code/YOLO_train.ipynb). During training over 50 epochs, we achieved a 0.762 mAP50 and 0.564 mAP50-95 score. However, after trying the trained network with our own made and real images from vehicles, the model performed disappointingly. This reason for this probably was, that the trained images were low resolution and contained even lower resolution vehicles with very little detail, but our images very clear, high-resolution images with a lot of details.

Knowing this, we went for an easier approach. We used the pre-trained YOLOv8 model that can detect different types of vehicles out of the box, namely cars, trucks, busses or motorcycles. From the pre-trained the weights, we tested the nano, small and medium versions and we concluded that the small version achieves adequate and quick results for our vehicle detection algorithm.



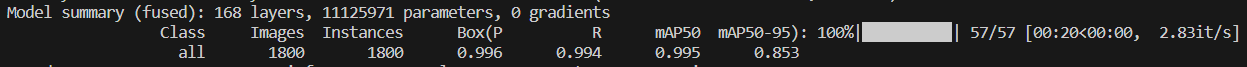
2. figure Vehicle detection with the YOLOv8s model

Most of the vehicle detections were above 80% confidence and the inference time was around 30 ms, not slowing our video pipeline down too much. As the detection worked well after several tests, we could move on to the next task, which was to read the number plates.

1. **License plate detection and reading with OCR**

In this chapter we introduce the algorithm what we use for license plate detection and reading.

As for the license plate detection, we again used the YOLOv8 algorithm. Since the YOLO can’t recognize license plates out of the box, we had to look for a dataset to train on. We have found a good quality dataset called [RodoSol-ALPR Dataset](https://github.com/raysonlaroca/rodosol-alpr-dataset). It contains 20,000 images captured by static cameras located at pay tolls owned by the Rodovia do Sol (RodoSol) concessionaire, which operates 67.5 kilometers of a highway (ES-060) in the Brazilian state of Espírito Santo. There are images of different types of vehicles (e.g., cars, motorcycles, buses, and trucks), captured during the day and night, from distinct lanes, on clear and rainy days, and the distance from the vehicle to the camera varies slightly. All images have a resolution of 1,280 × 720 pixels. We had to ask for permission to use the dataset for our academical use case. The format again wasn’t ready to be used for YOLO training, so we created another script to convert it to a appropriate format. The training was done locally on our PC, with an RTX 3070 GPU. With only 5 epochs a quite good results could be achieved with only around 10 ms inference time:



1. Figure: The results of the license plate detection training



3. figure: License plate detection

* 1. **Reading of license plates**

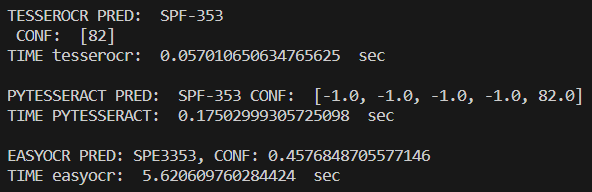
To read the license plates firstly we used the *pytesseract* OCR engine. This solution was good in terms of results but was very slow, it only could do inference in a few hundred miliseconds. The next one we tried was EasyOCR, but this solution was worse than the previous one in every respect, so we discarded it quickly. After the above-mentioned solutions didn’t offer adequate results, we found the API version of pytesseract: *tesserocr*. This did not slow down the program that much and worked with reasonable accuracy.

Tesserocr is a Python wrapper around the Tesseract C++ API. Whereas Pytesseract is a wrapper for the tesseract-ocr CLI. Therefore, with Tesserocr we can load the model at the beginning of our program and run the model separately (for example in loops to process videos). With pytesseract, each time we call image\_to\_string function, it loads the model and processes the image, therefore being slower for video processing [2].

The comparison for the different OCR techniques can be seen at 1. Figure for the input seen at 2. Figure.



2. Figure: An example license plate



3. Figure: The comparison of different OCR techniques.

We can see that the EasyOCR library didn’t achieve as good as a result in both accuracy and speed as the other two. TesserOCR was just as accurate as Pytesseract but achieved a three-fold speed increase over its rival, so we went with this technique. Sadly, the two best performing techniques don’t always report a confidence score for the prediction of each character as we can see on the 3. Figure with the -1.0 values. So, we couldn’t reliable use this value for updating the prediction when a higher confidence score was found, since it could be that the correct license plate text was found but with -1.0 confidence values.

**Estimating vehicle velocity**

For the vehicle velocity estimation, we tried different techniques. At first we measured a region of the image (between to dashed lane pieces) calculated a pixel/meter ratio as we knew how distant the two lane pieces in reality are. This solution wasn’t working properly however, because we assumed a

1. **Test**

When the algorithm was fully ready, we shot several test videos in the city with an Olympus OMD E-M10 camera. The most suitable location was Petőfi Bridge, Buda side, as we could film cars driving along the quay from the bridge from above.

However, the camera had a property that only objects closer to it were sharp enough, so a mask was added to the video so that detection, reading and speed measurement only occurred in this area, speeding up the algorithm and optimizing it.



5. ábra kép a maszkolásról

We made several tests and processed them, and based on them we can say that the algorithm meets our current learning goal and 90% of what we had in mind at the beginning of the project.

1. **Summary**

This project, in which we developed an algorithm for car detection, license plate recognition, and speed estimation, has been an incredible experience for us. Over the past period, we have put in intensive work to understand the principles of computer vision, the latest methods in object detection and character recognition, as well as techniques for speed estimation.

Working on this project not only enhanced our theoretical knowledge but also provided us with practical experience. We encountered challenges and difficulties along the way, which we successfully overcame. Preparing the data, debugging errors, and fine-tuning the algorithm were all tasks in which our team collaborated effectively.

In conclusion, this project has been a rewarding journey that has allowed us to delve into the fascinating field of computer vision and machine learning. We have gained valuable knowledge and skills that will undoubtedly be beneficial in our future endeavors.

# **Literature**

[1] What is YOLO: <https://www.baeldung.com/cs/yolo-algorithm>

[2] Tesserocr: <https://python.plainenglish.io/tesserocr-vs-pytesseract-d6720207bb54>

[3] Picture of OCR Process Flow: <https://nanonets.com/blog/ocr-with-tesseract/>