BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

OCCUPANCY ESTIMATION OF A PARKING LOT FROM IMAGES
URČENÍ OBSAZENOSTI PARKOVIŠTĚ Z OBRAZU

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR   **RAUL AGHAYEV**
AUTOR PRÁCE

SUPERVISOR   **prof. Ing. ADAM HEROUT, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2021

Ústav počítačové grafiky a multimédií (UPGM)   Akademický rok 2020/2021

Zadání bakalářské práce

23263

Student: **Aghayev Raul**
Program: Informační technologie
Název: **Určení obsazenosti parkoviště z obrazu**
           Occupancy Estimation of a Parking Lot from Images
Kategorie: Zpracování obrazu
Zadání:

1. Seznamte se s problematikou počítačového vidění v dopravě. Prostudujte a popište existující přístupy k počítání automobilů v obraze.
2. Vyhledejte a popište dostupné datové sady pro učení a vyhodnocování řešených algoritmů.
3. Pořizujte vlastní data z realistických scén pro vyhodnocování, případně učení algoritmů.
4. Experimentujte s vybranými algoritmy, zhodnoťte jejich praktickou použitelnost v různých scénářích.
5. Vyviňte vlastní řešení zadaného problému. Zhodnoťte jeho použitelnost na vhodných datech.
6. Vytvořte demonstrátor svého řešení; diskutujte jeho použitelnost v praxi.
7. Iterativně vylepšujte své řešení směrem "k dokonalosti".
8. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- X. Jiang et al., "Density-Aware Multi-Task Learning for Crowd Counting," in IEEE Transactions on Multimedia, doi: 10.1109/TMM.2020.2980945.
- Dobeš et al., Density-Based Vehicle Counting with UnsupervisedScale Selection, DICTA 2020
- Bharath Ramsundar, Reza Bosagh Zadeh: TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning, O'Reily Media, 2018
- Gary Bradski, Adrian Kaehler: Learning OpenCV; Computer Vision with the OpenCV Library, O'Reilly Media, 2008
- Richard Szeliski: Computer Vision: Algorithms and Applications, Springer, 2011

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2, značné rozpracování bodů 3 až 5.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/
Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstract

The aim of a diploma is to create an application that will work detect vehicles on a video from parking areas and determine the occupancy of parking area, by detecting the cars, saving data about them and count the number of busy slots. This kind of application can substitute sensors in a future whereas the cost of it is much cheaper and it detects the new coming cars in a real-time, in addition with some statistics such as the most popular parking slots or total amount of cars on a parking today.

Abstrakt

Cílem diplomové práce je vytvořit aplikaci, která bude pracovat s detekcí vozidel na videu z parkovišť a určením obsazeností těch samých parkovišť, ukládáním dat o nich a počítáním počtu obsažených slotů. Tento druh aplikace může v budoucnu nahradit senzory, zatímco jeho cena je mnohem levnější a aplikace detekuje nová přicházející auta v reálném čase, navíc s některými statistikami, jako jsou nejoblíbenější parkovací sloty nebo celkový počet automobilů na parkoviště dnes.

Keywords

Vehicle, detection, Mask R-CNN, parking, real-time, openCV, self-detecting

Klíčová slova

Vozidlo, detekce, Mask R-CNN, parkování, real-time, openCV, samodetekce

Reference

Extended Abstract

V moderním věku počet aut roste a roste každý den. Takový počet aut určitě vyžaduje místa kde tě auta je možné nechávat (například přes noc nebo na dobu dovoleny). Ty místa jsou garáže nebo parkovací místa vedle nějakých objektů.

Každý den se nastává situace když přes miliony lidé chtějí zajít někam do restaurace, obchodního domu nebo musejí jet do práce. Půlka z tich lidé využije auto k tomu abych dostaly do města kam chtějí dostat. Parkovišti v těch místech (a jich okolí) byly postavení kvůli tomu aby lidé mohli někde nechávat své auta.

Pokud před několika dekády parkoviště nebyly tak populární a velké (kvůli populace země a počtu aut), teď parkoviště můžou být obrovské 9-patrové budovy, které jsou shony umístit 1000 aut na jednom patře. Postavení takových budov potřebují hodně investice a vyžaduje rok (případně 2) času. A když už jsou postaveny, tak je zapotřebí ještě hodně peněz na to aby v těch parkovištích se dalo dobře orientovat a pochybovat. Markery které ukazují směr jízdy, sensory, výtahy...

Skoro ve všech parkovištích jsou instalovány senzory. Ty bývají elektromagnetické nebo ultrazvukové. Sensory slouží k tomu aby detekovat auto na parkovacím místě a signalizovat že je obsazeno, nebo volně pokud auto tam nestojí. Toto všechno pomůže jinému autu, které přijelo na parkoviště schopno najít pro sebe volné místo. Instalování senzory je drahé "potěšení", mnohem levněji bude navrhnout nějaké programovací řešení. Toto programovací řešení bude muset skenovat auta z kamery na parkoviště a z toho vytvářet kompletní mapu parkovišti, navíc s nějaké dodatečné informace.

Prvním krokem při vytvářeny aplikace které umožňuje detekovat auta bylo vyhledat a prostudovat literaturu a časopisy o počítačovém vidění. Najít existující nebo podobné řešení. Pochopit jak funguje detekování aut z obrazy, najít pro toto vhodný nástroje a programy které lze snadno pro této ucelí využít. Přečíst ještě víc naučni literatury.

Následujícím krokem bylo najít vhodné data sety pro testování a taky vytvoření nebo sbírani své vlastně testovací datové sady. Datové sady pak museli být otestovánu s různý detektory a nástroje.

Jakmile bylo nalezeno vyhovující nástroje, došlo k vytvoření návrhu aplikace. Aplikace musela fungovat autonomně. Nová auta musejí být nalezený samy (nalezený aplikací) . Existuje vhodné řešení kde parkovací sloty musejí být označení uživatele, ale této aplikace musí detekovat auta sama. Omyly a programové chyby které byli nalezený během detekce taky musejí být opraveni aplikací. Navíc aplikace by musela představovat statistiky. Takové statistiky jak celkový počet aut anebo nejpopulárnější místa byly by vhodné k implementaci. Aplikace by měla být spustitelná na vhodných počtů data setů a taky pracovat s video přímo z kamery v režimu živého přenosu. Taky aplikace by neměla havarovat a mít nedeterministické chování.

Hlavním cílem aplikace je být nástrojem které pomůže se dobře zorientovat na parkovišti, kvůli tomu jednotlivé parkovací sloty by museli být vyznačení čísly.

Všechny této požadavky a znalosti které byly získne z knih a časopisu dovolili vytvořit aplikace která dobře analyzuje parkovišti.

Po každé implementace nějaké nové funkce nebo zvláštní byly provádění testy, které ukázali na defekty nebo nedostatky. Konečně řešení bylo otestování na 4 hlavních data setech a výsledky byly vyhovující. Navíc testování na data setech objevovalo více chyb které by museli být opraveny a právě toto zlepšilo konečnou verzi aplikace. Testování bylo provádění na různých výškách kamer, v noci. Všechny výsledek byly analyzovány.

Konec koncem je to aplikace které dokáže detekovat auta z přesnosti 93.3%, analyzuje informace o nich a poskytuje statistiky.

Occupancy estimation of a parking lot from images

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of prof. Ing. AdamHerout, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

<div align="right">

. . . .

Raul Aghayev

May 8, 2021

</div>

Acknowledgements

I would like to thank my supervisor without whom I would not have had enough motivation, knowledge and strength to complete this thesis.

Contents

Chapter 1

Introduction

Due to the fact that vehicles become more and more popular every day, the building of parking is vital to the society. Wherever there is a need to park a car somewhere, parking should be close. Especially in a big cities or cities that attract tourists a lot. What is there standing in front of any parking building or any mall? – exactly, the scoreboard with information about free parking slots in parking area.

All of these constructions (that provides the parking's statistics) are created using the big computers and sensors that are placed on top of each parking slot and indicate whether the parking slot is free or not by collecting data from sensors and communicating with the main computer. Additionally, all of these technologies are expensive, even without fee for installation. The application that will be described in this diploma could possibly change the world of parking detection from head to feet.

The idea is in installing cameras in front of parking areas (blocks), these cameras will use the program which will obtain the „parking-map" from cameras in a few days after launching. Afterwards, all of these cameras (number of which will be much more smaller than amount of sensors) and the program will have data about the free parking slots, along with the geolocation of that slots. This information will be used to display information on a scoreboard outside. The inside part will be solved by placing a table with the actual map in the beginning of each row, they can be placed on the top of sideways.

Exactly these kinds of ideas made me choose this topic and encouraged me to implement a program that, with the use of neural networks, will be capable of finding, detecting and counting the amount of parking slot and then the number of free parking places. In addition I wanted that application to be self-correcting and auto-detecting. Self-correcting means that program will fix the bugs or errors that it has created itself. Auto-detecting means that there is no operator needed to exploit the application. Once you run it, it will do everything on its own. Chapters below will explain and describe the program more deeply, along with creation process, bugs and difficulties that were faced during the creation of an application.

The aim of my diploma was to get acquainted with the computer vision in transport, objects detecting and analyses. Understanding the basic and advanced principles of detection, experimenting with different tools and techniques and finding the one proper way of implementation of the project.

More precisely, the responsibilities of the project are to detect the vehicles on a video or images data sets. Further inspection, which allows the program to collect the complete vision of a parking with information of a parking occupancy rate.

The thesis's task mainly focuses on understanding of computer vision and vehicles detection. However contains a lot of other targets to achieve. These targets are:

1. It was vital to get acquainted with the issues of computer vision in transport.

2. Exploring and understanding existing vehicle counting approaches [9].

3. Finding available data sets for testing.

4. Acquiring my own data from realistic scenes for evaluation or learning of the algorithm.

5. Planning and developing my own solution to the problem. Testing and testing and of course iterative improvement of solution towards „perfection.

6. Implementation of features such as statistics and etc.

At the end, good-working free-parking slot auto-detecting application was created, it shows over 90.9% accuracy in detecting of free parking areas. Based on those results it is possible to bring that kind of application to live and change the world of parking.

Chapter 2

Relevant Computer Vision Approaches

2.1 Computer Vision and Image Processing by Neural Networks

Computer vision[1] is a theory and technology of creating machines that can detect, track and classify objects. As a scientific discipline, computer vision refers to the theory and technology of creating artificial systems that acquire information out of images (frames). These images could be parts of videos, data sets or for instance collected from the medical scanner. Next chapters will describe the detectors, their advantages and disadvantages in terms of a thesis.

2.2 Existing Solutions for Vehicles Detecting and Counting

The world of parking now mainly uses the sensors on top or bottom of each parking slots, these techniques could be found in almost any parking. Those sensors could be Electromagnetic or Ultrasonic, other methods such as counting the number of cars in entrances and exists could be also used, but that exact method don't provide enough data. Drivers will know that there is one empty lot, but nothing about its geolocation.

Ultrasonic sensors [6] – uses sound waves to detect objects. By sending sounds in a high-frequency that reflects off objects, a sensor can catch the reflected waves and compare it to referential value which will state that the lot is busy or not.

Electromagnetic sensors [12] – creates an electromagnetic field which detects anything that entrances that field. As a result, when a vehicle is entering a parking lot it will be detected.

In addition there are a lot of existing programming solutions both for detecting or data (statistics) collecting. All of them could be divided in several groups.

The first division is language of implementation. The most popular are Python and C language. Python language is used more than C due to the simplicity. The second division criteria is made by the detector. The most popular are Mask R-CNN and YOLO. The final division is the way that parking slots are detected(created).

---

[1]https://en.wikipedia.org/wiki/Computer_vision

More precisely about last group. There are a lot of solutions that requires the user to circle the parking lots with mouse[2], whereas there is only one solution that detects the cars on a first frame and then makes parking slots from them[3], however that solution doesn't handle the new coming cars.

My solution should be an auto-detective, that means that new parking lots should be detected by application automatically. I couldn't find any solutions that works in that way. So in my solution, any place on a video becomes a parking slot in case if a car stands there for a few minutes. Furthermore the statistics such as most popular places, the total number of cars and the run-time parking occupancy should also be implemented.

All of these 4 methods have one aim, detect the cars and provide the data. In comparison it is obvious that the sensor will be the most expensive out of this list. Ultrasonic sensors price is 2500 to 3000 USD per each lot[4]. Electromagnetic sensor will cost cheaper, from 300 to 1000 USD[5]. The reliability is high. Counting cars in the entrances and exists will cost much less, the only required thing is sensor that will count every car entering and exiting the parking. The price for this varies up to 1000USD including installation and setting per one gate, the best price for that is 500USD[6] however the one big minus as it was mentioned before is lack of geolocation.

The programming technique is obviously the cheapest methods out of all. This application could be written for 2 weeks of hard-work when the price for that will be no more than 500 USD. Despite application could sometimes lie in the dark, it is still a good choice. In addition there is no perfect detector and all of the detectors will sometimes fail, the percent of failure is small. All facts mentioned before illustrates why programs could replace sensors on a parking.

In summary the programming method should replace the sensors at parking due to the price and simplicity, however this could happen only when the detectors will work better than they are doing now. The programmers from all over the world are working on improvements of computer vision, that's why the detectors' capabilities will rice dramatically next decades and the sensors will be changed by applications.

2.3   Object Detection: Mask R-CNN

Mask R-CNN[7] is a neural network that is used to detect objects on a video or image, those objects could be almost everything, starting from car's plate, finishing with animals. On of the main features of Mask R-CNN is capability to illustrate the objects, even if there are many same object that are corresponding to each other. This detector is capable to differ all of them. Mask R-CNN works in a following steps [4]:

1. Prepare the image for the next step.

---

[2]https://github.com/olgarose/ParkingLot

[3]https://medium.com/@ageitgey/snagging-parking-spaces-with-mask-r-cnn-and-Python-955f2231c400

[4]https://www.alibaba.com/product-detail/Parking-Ultrasonic-Sensor-Intelligent-Guidance-System_62088969902.html?spm=a2700.7724857.normal_offer.d_title.48fd1d80kEwSKq&s=p

[5]https://connectedthings.store/gb/lorawan-sensors/bosch-parking-lot-sensors.html?gclid=CjOKCQjwsqmEBhDiARIsANV8H3YE1e-3261gWqYVTE1cGd_o5-CZWKXCC4yuY5llSOTJyT2ixtpU46oaAmKREALw_wcB

[6]https://www.optex.co.jp/e/products/vehicle-detection/fmcw/ovs-01gt.html

[7]https://github.com/matterport/Mask_RCNN

2. Region Proposal Network – predicts if there are any objects are presented in particular regions of an image.

3. Resizing all of the object obtained from previous step to a same size and passing all of the regions to another neural network.

4. That neural network takes proposed regions and will assign them to several specific areas of a feature map level.

5. Scans these areas, generates objects. These objects will contain masks, boxes, object's names (labels) and classes (see figure 2.1).

I decided to chose Mask R-CNN as a detector due to the good detection, simplicity and big variety of additional options, in addition it seemed to me that connecting my application with that detector in Python language will be easy.
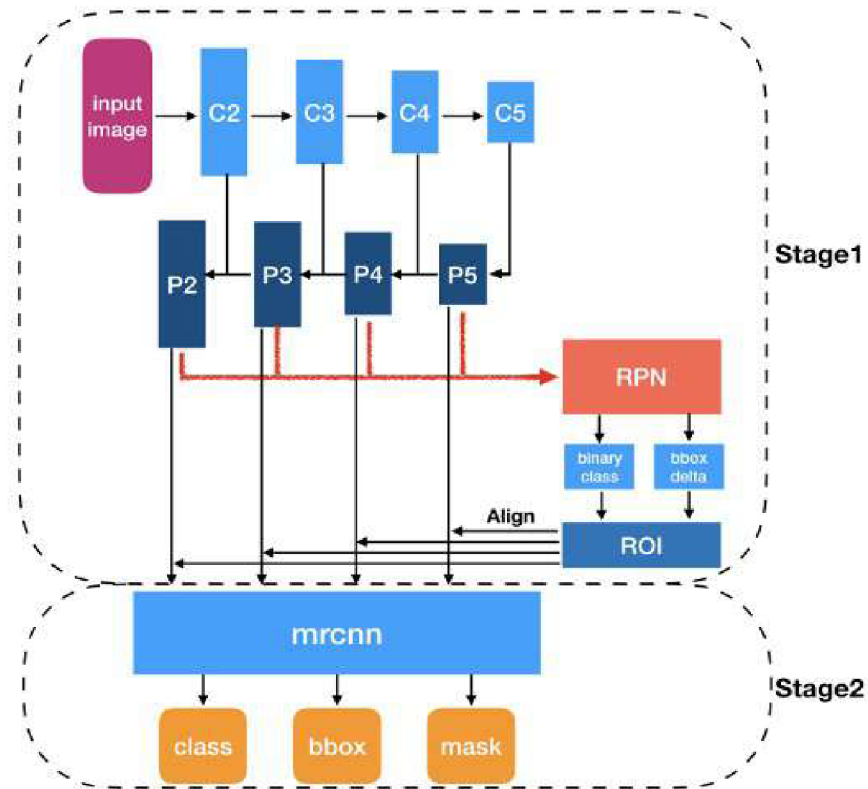

Figure 2.1: Mask R-CNN structure[8]

2.4   Mask R-CNN COCO data set

Mask R-CNN COCO data set is a data set for object detection [7], segmentation and captioning. Coco models should be used in the application due to the two main features, one of them is recognition in the context, the second one is big variety of object instances
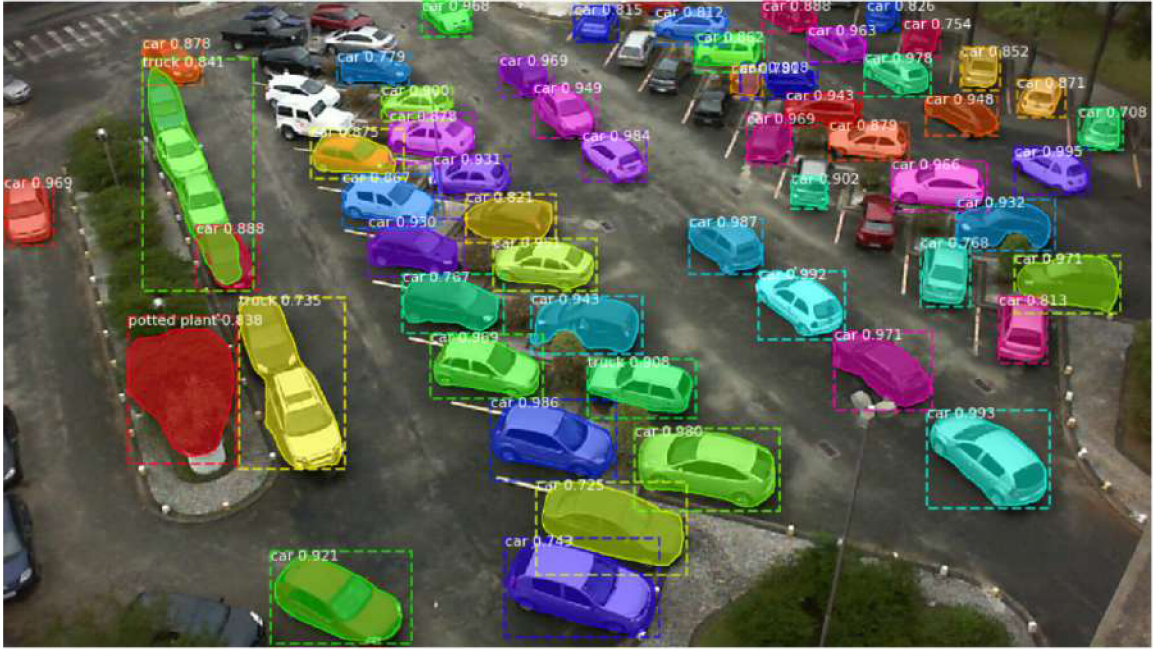
---

[8]https://alittlepain833.medium.com/simple-understanding-of-mask-rcnn-134b5b330e95

Figure 2.2: Mask R-CNN object detection demonstration[9]

and the labels. The application uses COCO model pre-trained data set that could be found on the Mask R-CNN official GitHub and are marked there as "easy to start„ weight, the training of new data set was unreasonable due tot the fact that COCO model already exists and perfectly suits for the task.

Image 2.2 demonstrates the capabilities of Mask-RCNN detection, using COCO pre-trained weights. As it could be seen all visible cars are detected.

2.5 Object Detection: YOLO

YOLO ("You Only Look Once") [10] is a object detector, that is capable of detecting almost every objects in the world, staring from animals finishing by car's plates. It gives the same output like any other detector, good at detecting same objects that are corresponding. YOLO uses different approach[11]. For instance Object prediction is processed in one run of the algorithm,in addition YOLO processes whole image (see figure 2.3). YOLO works in the following steps[8]:

1. Dividing frame into a cells using a (usually) 19x19 grid.

2. Each cell is responsible for predicting several bounding boxes.

3. Find probability of that the object exists in that boxes.

4. Delete boxes with small probability, bound boxes with big probability.

YOLO detector is fast and accurate, with big amount of options and high real-time accuracy. However, Mask-RCNN suited more for the purposes of the thesis.

S X S grid on input

Bounding boxes + confidence

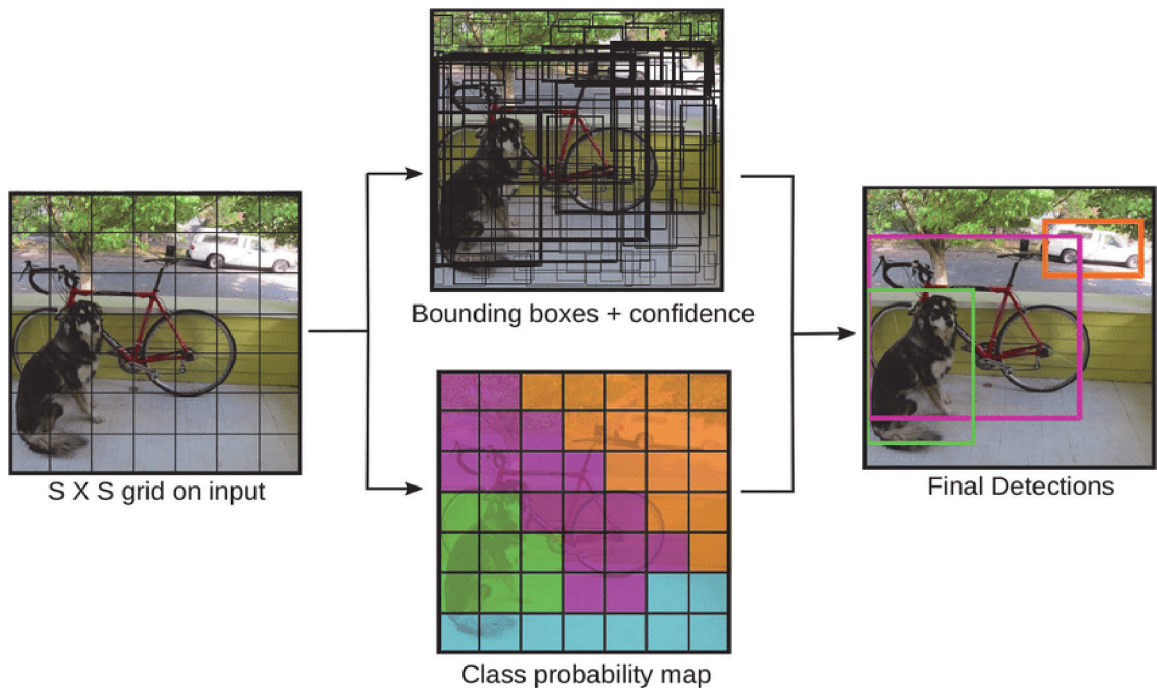Class probability map

Final Detections

Figure 2.3: YOLO object detection demonstration[12]

2.6   Object Detection: Darknet

Darknet[13] is an open source neural network written on C language[11] and CUDAtechnology, the darknet detector is one of the most accurate and fast detector that was tested by me. It suits really well for real-time predictions. However, due to the fact that darknet is using C language, I decided to not to use it due to compatibility problems.

---

[12]https://www.researchgate.net/figure/YOLO-model-detection-as-a-regression-problem-17-Thus-the-input-image-is-divided-into-a_fig5_337146307

[13]https://github.com/pjreddie/darknet

Chapter 3

Relevant Tools for Image Processing and Computer Vision

3.1 OpenCV Library

OpenCV (Open Source Computer Vision Library) [3] is an open-source library for computer vision, image processing and general purpose numerical algorithms. Implemented in C / C++, developed for Python, Java and other languages. In addition, that library is widely used in the world, documentation is heavy and the internet is full of tutorials. Moreover to computer vision, OpenCV library is handling the frames processing in the application, and is also handling drawing of rectangles and ellipses.
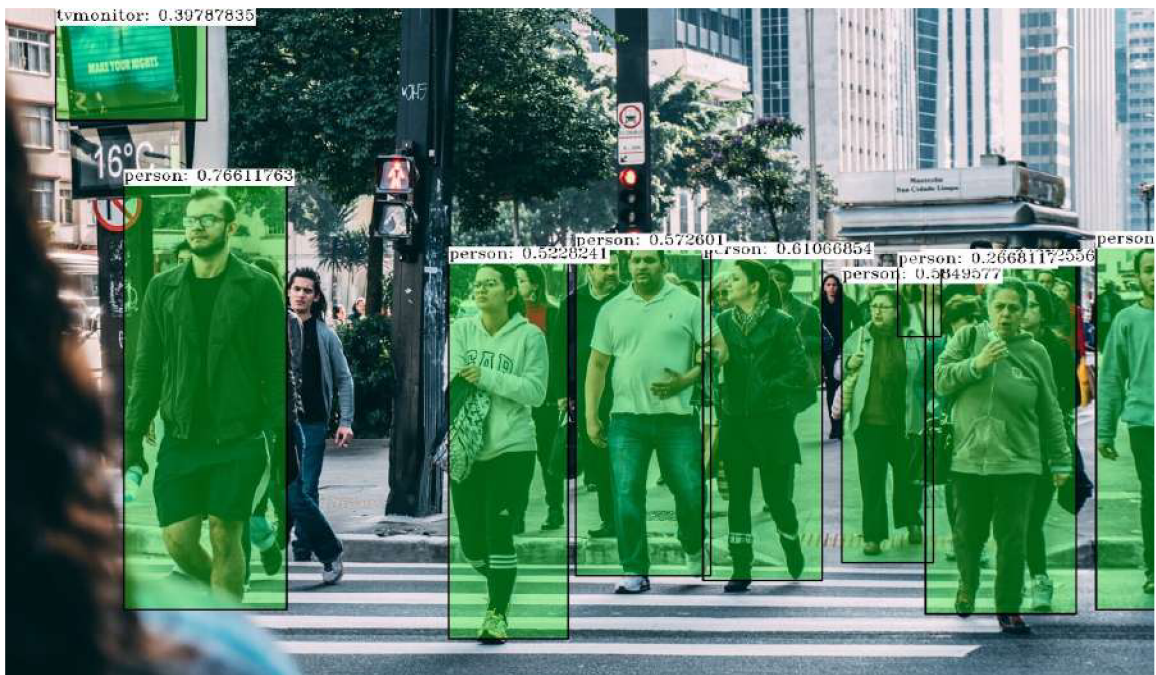

Figure 3.1: Open CV capabilities demonstration[1]

In addition OpenCV has a features with mouse handling. On the left bottom corner of the figure 3.2 some information could be found. While processing a video, user can hover a mouse over a screen and OpenCV will display the coordinates of a mouse and color of a pixel that was hovered over. The color will be expressed as an RGB values.
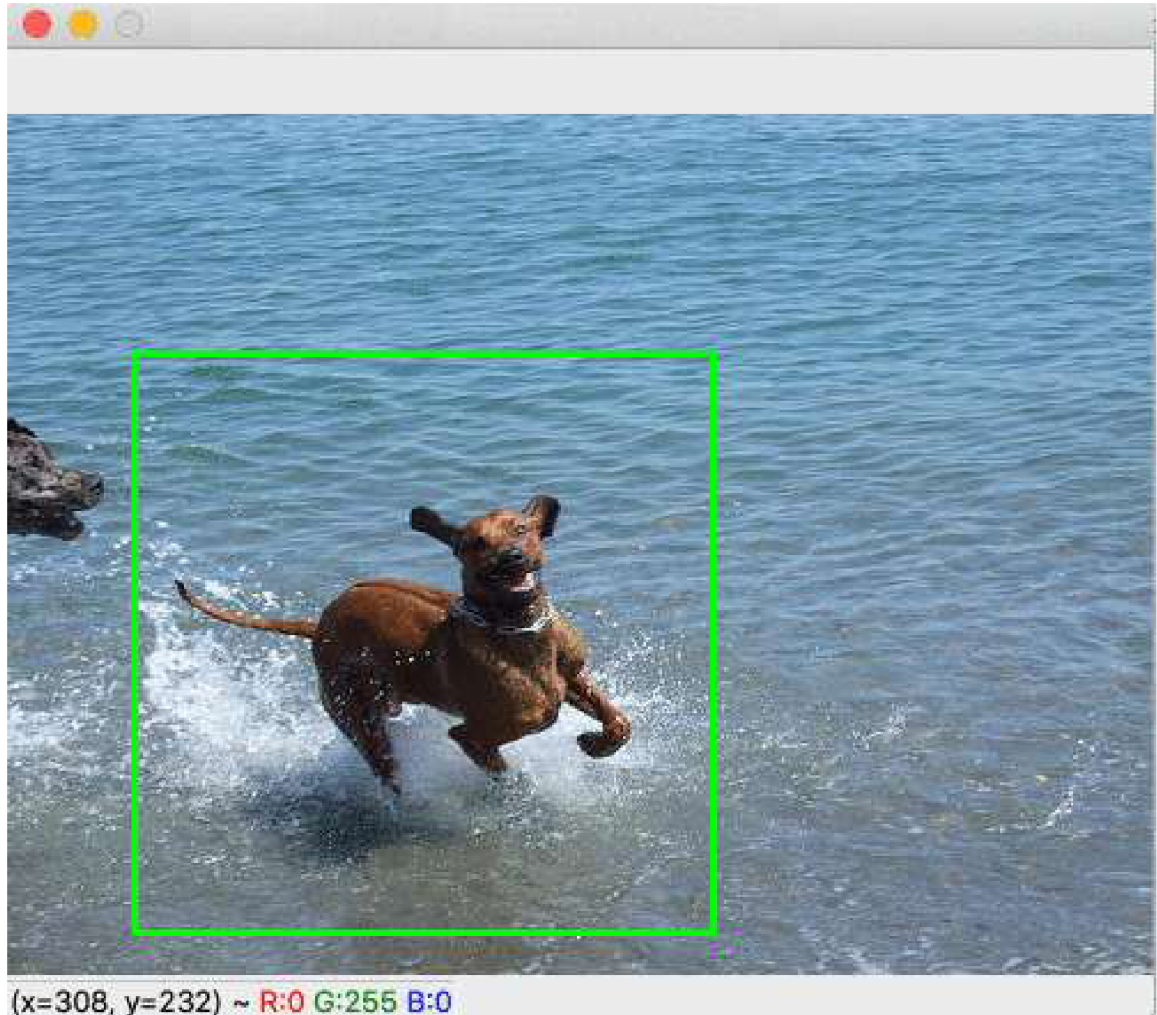

Figure 3.2: Open CV capabilities demonstration[2]

3.2 Development in the Python Language

In the world of big varieties of programming languages, Python[3] is widely used and is reasonably in the tops. Python[4] is interpreted language, which is a big plus comparing to other languages, in addition Python is really simple, object oriented language to learn, with enormous amount of libraries, almost all libraries in the world were developed for Python also. Additionally Python is widely used for computer vision purposes [10].The amount

---

[2]https://scipython.com/blog/cropping-a-square-region-from-an-image/
[3]https://existek.com/blog/ai-programming-and-ai-programming-languages/
[4]https://www.sam-solutions.com/blog/image-recognition-programming-language/

of code that should be written in Python to perform some action is much more less in comparison to other languages.

3.3   TensorFlow

TensorFlow[5] is an open-source machine learning software library developed by Google for solving problems of building and training a neural network in order to automatically find and classify patterns. The main API for working with the library is implemented for Python, there are also implementations for C#, C++, Java and etc. More information about TensorFlow could be found in the book. [13]

---

[5]`https://www.tensorflow.org/`

Chapter 4

Proposals for Application Creation

4.1   Selection of The Eligible Tools

The experiments with tools were wide [2].

The first choice was the **Darknet** detector. "Darknet is an open source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation.„[1] That detector showed perfect accuracy, however that detector wasn't suitable for me due to the language of implementation which was C language. In addition it had long "run command„ (command user enters to run the program). Due to the fact that I decided to write the program on Python language, connecting two languages or changing the language of implementation could be unreasonable.However, in spite of long loading and processing time darknet showed perfect accuracy in detecting all kind of vehicles.

Second choice was **YOLO** detector tool were tested. It showed a perfect accuracy. The detection of the bounds were satisfying, all of the objects appeared in a different colors which of course helps with differing objects on a screen. However, while testing I came up to a problem with visualizing masks of detected objects. Masks of the object are useful for debugging purposes and overall computer vision understanding. Reading deeper I couldn't found out information about YOLO's masks, however I found out that masks virtualization is easily accessible in Mask-RCNN, I decided to test that detector.

MASK R-CNN, as any other detector showed a perfect detection. In addition, it is incredibly simple to manipulate with that tool in Python language. Using all of its functional and features. Mask R-CNN is easy customizable tool which allows to set the minimal percentages of detection, which hardware it should use and all of the data is easily accessible right from Python code. Accessing the masks or any other information from code was extremely comfortable. One more big plus is that 90% of the detecting parked cars program were written using Mask-RCNN. That's why, after careful considerations I decided to use Mask R-CNN.

Obviously that all of the detectors were good but have some differences, the table of comparison is placed below (see table 4.1), all of the data was generated by me using the 4 main data sets (videos).

---

[1]https://pjreddie.com/darknet/

Table 4.1: Basic comparison of detectors

	YOLO	MASK R-CNN	Darknet
Detected vehicles (data set №1)	100%	100%	100%
Detected vehicles (data set №2)	100%	100%	100%
Detected vehicles (data set №3)	96.9%	93.9%	93.9%
Detected vehicles (data set №4)	88.5%	82%	88.5%
Initializing time	10 seconds	23 seconds	29 seconds

4.2 Languages of Implementation Selection

From the big variety of programming languages C++, C, Java and Python languages are the most recommend for purposes of computer vision.[2]

C / C++[3] are widely used to create artificial intelligence applications. The built in libraries in that languages such as OpenGL or OpenCV already have features and tools which are helping in fast pictures or frames processing. All of this in combination could help to write a fast working application or program. Regarding disadvantages of C++ or C languages could be low-levelness and speed of execution in comparison to other languages which will be described below. I decided not to use the C or C++ due to the amount of code that should be written in comparison with more high-level languages and of course the compatibility with Mask R-CNN which is also implemented better in high-level languages.

Java[3] is also widely used language in which simple applications could be created along with complex artificial intelligence applications. The advantages of Java language is native detection libraries, compatibility with OpenCV and OpenGL, simplicity in comparisons with C or C++ and portability. From the disadvantages, the speed of execution which is slower, in comparison to all other languages that were mentioned and of course the rawness among all of the other languages in terms of artificial intelligence. Personally I decided not to choose Java due to the "incompleteness„ in that field.

Python[3] is one of the best languages for artificial intelligence in the world by the opinion of many programmers, it has enormous amount of libraries, almost every library in the world is also implemented for Python language. Python is fast language and due to the high-levelness, it is much more easier to learn Python than C / C++ or Java. In addition, Python is fast and interpreted language.I decided to choose Python language because it suited me the best in terms of artificial intelligence and compatibility with Mask R-CNN, the overall connection between that tool and this language is good, which definitely let me to flexibly set every desired option.

The overall comparison of the main aspects of the programming languages are demonstrated in the table 4.2.

Table 4.2: Basic comparison of programming languages in terms of computer vision[4]

	Java	C++	C	Python
Run time in milliseconds	<1.89	<1.56	<1.00	<71.90
OpenCV support	Yes	Yes	Yes	Yes
OpenGL support	Yes (without API)	Yes (without API)	Yes (APIs)	Yes

---

[2]https://reubenrochesingh.medium.com/comparison-of-10-programming-languages-f43b0ac337a4
[3]https://www.sam-solutions.com/blog/image-recognition-programming-language/

4.3 My Proposal

Application was implemented in Python language with usage of Mask R-CNN and OpenCV, the proposal of application execution process is next:

1. When application is launched, the input data set processing will start. Processing will be done frame by frame.

2. Initial frame is processed in a different manner. The program detects all cars on initial frame and instantly marks them as parking lots, already busy parking lots. All of the parking lots are marked as a yellow ones in that moment.

3. Further processing is taking place now, each frame is processed, statistics collection is taking place starting from that moment. Information, such as amount of cars on the parking and current busyness of a parking will be shown on top left corner of a video.

4. Application now waits for new cars to detect, detects empty places and provides the user with all sufficient information. The application can run forever in case of a live streaming, however it will need time, real time to collect data and start working on a full potential.

The application should be self-correcting. If car detection has failed in one frame, program should re-detect that car in a next frame. Detection should be set on detecting of cars or trucks only, not any other objects.

The statistic of total busyness of a parking will be collected by finding the number of busy parking lots. The popularity of a parking lot will be determined by getting the ratio of the time exact parking lot is busy to the total application run-time. The application run-time will be increasing each frame, the parking lot's busyness time only when the parking lot is busy. The popular place is a place which is busy for more that 90% of time.

The detection of a cars should be repeated in every frame, as it was mentioned before I could not find any solution of that kind. That's why I had to create it myself. My proposal for that was auto-detecting the cars on each frame.

One if the most interesting solutions that I have created for the problem of auto-detecting is going to be described now. As it was mentioned before all cars will be analyzed and marked as a parking lots on a first frame, on every further frame the detector should be called. The detector will scan each next frame. While scanning, it will remember all of the new cars it found (that means the boxes around the new coming cars). Then it should compare new arrived boxes to already existing boxes. These will be done by comparing the array that represent the parking lots to a new arrived car. On listing 4.1 **existing_parking_lots** represents the parking lots that already exist. The following lines in same listing are responsible for finding the distance to the closest parking lot. The maximum distance between different parking lot's same coordinates(top left, top right,bottom left or bottom right, more information of figure 4.1) could be no more than relative value for that data set (the relative value will be described 2 paragraphs below). By summing up the value, the program will understand if the new parking lot has already been detected before. In case if it is not overlapping with any other lot, the new parking lot will be added to candidates for existing parking lots.

---

[4]`https://jaxenter.com/energy-efficient-programming-languages-137264.html`

```
# Looping through the existing parking lots
for existing_parking_lot in existing_parking_lots:
    # Subtract numpy array's element to find difference between parking lots
    result = abs(coordinates_subtract(new_parking_lot, existing_parking_lot))
    # sum up the differences
    [total += total + x for x in result]
    # return true if there is any difference, otherwise return none
    return True if total <= relative_size else None
```

<div align="center">Listing 4.1: Comparing the overlapping pseudo-code</div>

The candidate will become full-fledged lot in case if the new box is not corresponding with any of the existing boxes and stays one the same places for 5 or more frames. This spot in that case will be marked as a parking lot and added to a list of a parking lots. The technique of 5 or more frames was used by me due to the accuracy, if parking lots will be created for less amount of frames, program will have a parking lots on a cars that are in move. The implementation of that solution is expressed in pseudo-code 4.2.

```
for new_car in all_cars:
if(
new_car not in already_detected_cars
&&
new_car.same_place_value >= 5
):
parking_boxes.append(new_car)
```

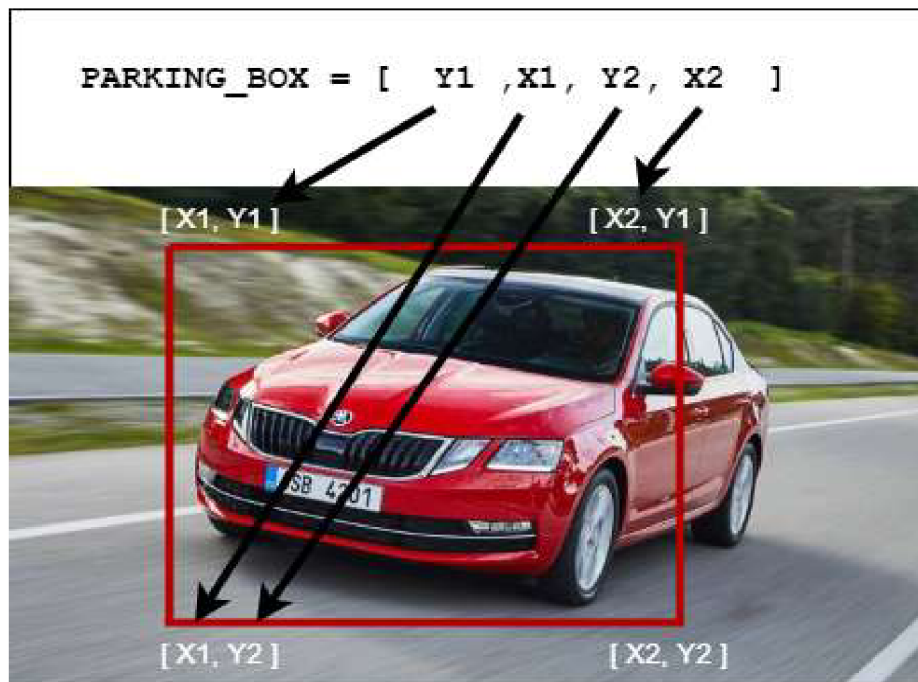<div align="center">Listing 4.2: New parking box adding pseudo-code</div>

<div align="center">Figure 4.1: Programmatic representation of parking lots[5]</div>

The relativity of a data set that was mentioned before will be described now. To find the maximum allowed distance between boxes, the total size (height and widths) of all of the found parking boxes should be summed up and divided to their total amount. This effect will give the program an effect of a size recognition. For instance, 200-300 pixels boxes could possibly mean that cars are located really close to camera, and the corresponding between them could possibly be in a wider range. Same idea for extra small 7 pixels boxes, where distance could be in range of 1-2 pixels. That's why this exact technique will perfectly suit all data sets and help program to become more effective.

The next achievement is drawing the boundaries around the parking lots. Drawing with different colors will depend mainly on intersections and popularity of a parking lot. Intersections or overlaps will be counted by finding the ratio between the area of a parking lot and a new coming car. In case if a new coming car is overlapping the existing parking lot for more than 45% that place is now counted as busy. Additionally if a place is popular, the program will find that out. Bounds of the lot will be yellow, in case if place is not popular, the box is red. If the place is empty, it will be green colored, if place is also popular the bounding will be blue.

One of the features of a program that to my of thinking, should be implemented is numbering the parking lots. As it could be seen from the picture below, the lots are numbered. The problem that were faced while creating this, is that the numbering of lots is chaining every frame. I decided to solve that in a way of creating a dictionary, key of that dictionary will be the coordinates of a parking box, whereas the value would be the assigned value. Initially the value will be 1 and each new detected parking lot will increase that value on one. This allows to number all of the vehicles on a first frame, in case of a new coming vehicle the number will be incremented, assigned and saved in that dictionary. From the benefits there is no need to delete elements form the dictionary due to the fact that the parking lot cannot disappear. That's why this solution should be the best. Marking of the lot is beneficial for orientation on a parking and can be also used for a lot of further features.

As you could notice there are some statistics information placed on the left top corner of some figures (see figure 4.2). Due to some conclusions, I decided to place all of the statistics to the left top corner. Colors of statistics' font should be changeable by button click, for that case the keyboard's 'f' button was chosen. Application will get the user's input and change the colors in order to make the statistic data display well in all backgrounds. Colors of the statistics should be: green, red and blue. The combination of these colors will show a good output according to my experiments. Experiments were conducted by finding the range of colors that will fit the background. The output was that at least one of that three colors (that were chosen) perfectly suited the data set.

On figure 4.2 the statistic states that the parking is filled by 100% however there are a lot of free spaces, these are caused due to the fact that parking is filled by 100% only according to the parking lots program already knows. In case if there were not vehicles on a parking lot yet, it won't be counted as a parking lot.

The next milestone is optimization. The first step of optimization should be optimizing the redundant code, which should be processed straight after implementation finishing. Additionally in a data set of images or videos, there is no need to scan every frame. That's why I made a decision to skip the frames and take frames on a distance 1.5 seconds from each other, this value could be bigger on less, the point is that after that kind of manipulations

---

[5]`https://www.skoda-storyboard.com/cs/0209_005-2/`

Figure 4.2: One of the variants of the final look of an application

program will definitely work faster. Additionally usage the good programming practices and clean code will also optimize the program.

One of the most useful features of an application is a dynamical bounding changing, firstly I decided to bound the car in a boxes, due to the fact that it is obvious and simple (see figure 2.1). However, at the end of the proposal, it was noticed that rectangle boundings are not always the best choice, see figure 4.4 the cars are lying perpendicular to cars. This was solved by changing the squares boundings to ellipse boundings. The boundings boxes are looking good in that case, and their main functions are visual effects (make user to differ the cars more easily). The figure below illustrates how good the ellipses are used exactly on that moment. However the ellipses are not always useful, these can be seen in the right side of figure 4.3.
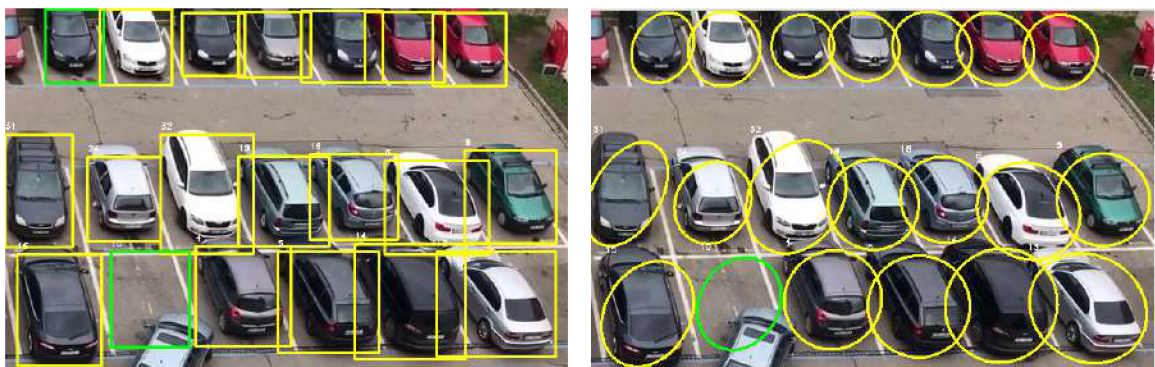

Figure 4.3: Two types of boundaries switching (squares and ellipses).

17

The most correct solution for that could be the possibility of chaining the boundaries view while program is running, that should be realised by button clicking. In case if the appropriate button is clicked, the squares will change to ellipses and vice versa. That button is keyboard's 't' button. This feature in addition with the statistic's color changing will be good visual addition. The figure 4.3 is perfectly demonstrating the effect of button clicking.

The application in a long perspective could be extended to a desktop application with complete information about the parking lots, in addition with beautiful GUI, optimization and a lot additional visual features that will make the application user-friendly, otherwise the program could focus on collecting statistics and become parking analyzer.

One of the possible final views of the application should look as it shown in figure 4.2.
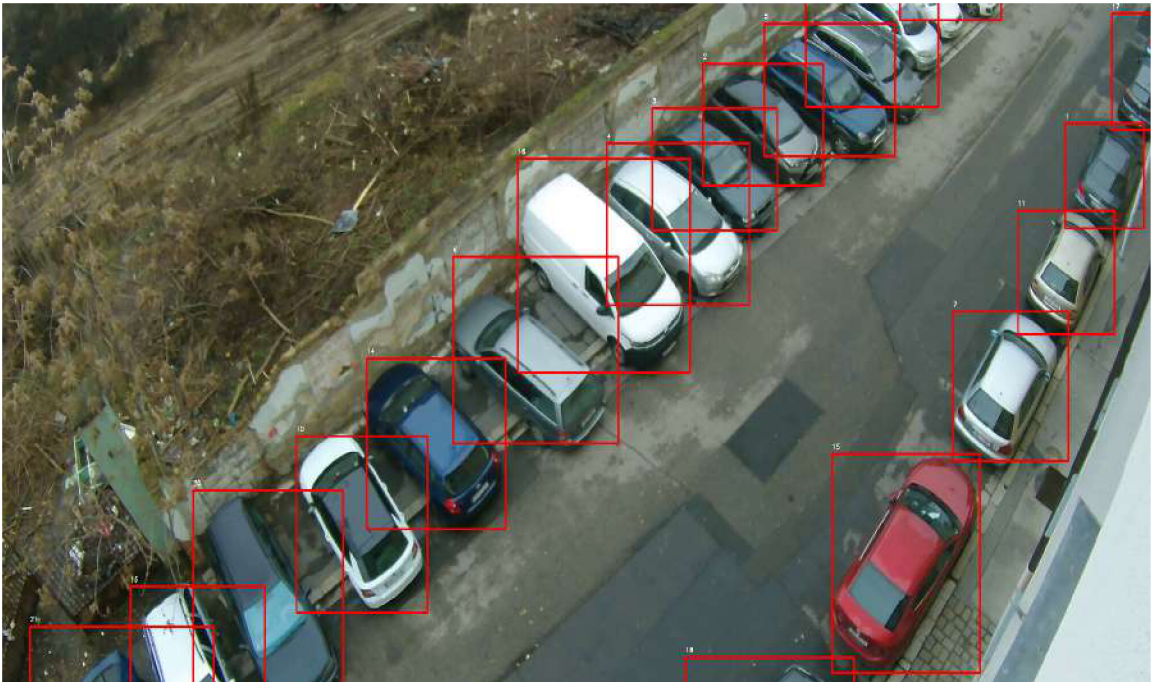

Figure 4.4: Squared boundaries around the parking lots

Chapter 5

Implementation

5.1 Implementation Details

This section describes the implementation details. All of the proposals were described in the section 4, this section will mainly focus on the implementation details. The input video is parsed and divided into the frames using OpenCV library.

The implementation is also using the COCO (see section 2.4) pre-trained models that were loaded into the Mask R-CNN detector. The detector itself is set on the desired options. The minimal confidence detection is set to 0.1 along with model name, amount of COCO classes and etc.

Mask R-CNN returns a big array which contains data such as the top left and bottom right coordinates of a detected car, identification of each car, also called as a label (car or truck), then the detection confidence rate and the object masks, for better understanding object masks and boxes with everything that was mentioned above are illustrated in figure 5.1. That means that when detection is process the Mask R-CNN detects only vehicles and cars.

Closer to the detection one of the most interesting solutions that were implemented in the project are boxes intersection. The function that finds whether the boxes are somehow crossing with each other or not (even partially). These were implemented in the next way.

As it was mentioned before the Mask R-CNN returns top left and right bottom coordinates of a detected car. Using those coordinates, it is possible to obtain all the coordinates of a car. The finding of intersections is executed in every frame. After getting and collecting the parking boxes (all parking boxes) they are sent to the function that gets every parking box and compares coordinates of it to coordinates of all other parking boxes. Comparison is completed by integers comparing. If the intersections length is less than the average distance between boxes on whole data set, the first box going to be deleted. This technique is similar to the one that is used to detect new coming cars. However, the boxes could have duplicity, top semi-duplicity, bottom semi-duplicity, right semi-duplicity and left semi-duplicity. Due to those reasons the implementation is a bit different (see chapter 6.2).

The next interesting feature of implementation is the numbering of parking lots. These were implemented by creating the dictionary, keys of that dictionary are represented as tuples of coordinates, these tuples are containing coordinates of a parking lot. The parking lot numbering function is called every frame, if there is no dictionary yet, it will be created with initial value of the first parking lot (initial value will be 1 due to the fact that the first slot should be marked as 1). When the program wants to get car's parking lot number, it

Figure 5.1: The Mask R-CNN objects detection

calls the function. Function checks if that parking lot was already numbered. Due to the fact that detection is changing every frame, the parking lot's border could slightly differ. That's why numbering is searched in range of 15 pixels from every corner(15 pixels is an optimal length). In case if the program already knows the parking lot that is placed not so far away from the parking lot that wants to be marked, it will return it. Otherwise it will crate a new key with the value that is not used in a dictionary. It could be new variable, that will be new generated value (value will be equal to total amount of parking lots), or the value of mistaken parking lot that was deleted from dictionary, but will be used again. These technique optimizes the program, it allows to not to crate excessive variables.

One more feature is handling the user's input. The OpenCV's waitKey() function is helping the program with that[1]. In particular case of an application, 3 buttons are handled. These are 'q', 't', 'f'. The 'q' button serves as a quit button. It destroys the windows, stops the application processing. The 'f' button changes the font of a statistic's. These are made using simple switch that changes the color from blue to green to red and then again to blue. The statistics is printed every frame, using the predefined color, button pressing changes that predefined color. The 't' button changes the representation, these are done by chaining the flag that is responsible for drawing ellipses or squares.

The last feature is a most popular parking lots detection algorithm. Implementation uses the dictionary. Coordinates are passed to a function; the coordinates are changed to a string and concatenated. In case if some coordinate is not in the dictionary yet, it will be saved with initial value of 1, otherwise if that coordinate is already in, the amount of value will be increased on 1 (on 1 each frame), them program will return the flag depending on how much of the total time that parking lot is busy. The total time amount increases every frame. I decided to count place as popular in case if it is busy for 90 % of the time. The

---

[1]https://docs.opencv.org/master/d7/dfc/group__highgui.html

box's color will be changed to red if place is busy for less than 90% of total application run-time, otherwise it will be yellow.

Application is supporting 2 input arguments. These arguments are responsible for providing a manual and choosing the data set. The help argument is called by –**help or -h** and will show the wide manual in command prompt. The –**dataset data_set or -d data_set** will specify the data set for the application. The data sets are more precisely described in section 6.3. These argument could be combined.

The handling of the application starts in the main file (parking_analyzer.py). All of the functions are called from that file. Additionally file for Mask R-CNN configuration was created. The variables were also created in a different file, this allows to use the variables comfortably in all files. All of the main debugging functions are handled by the functions. The same as statistics placing and parking lots' coloring.

5.2   Implementation of the Vehicle Detection Algorithm

More precisely about vehicle detection. First of all the video is parsed by OpenCV. Each frame is converted to BGR (Blue Green Red) 5.2 from RGB (Rad Green Blue). Afterwards that frame is sent to Mask-RCNN detection model. The Mask R-CNN assumes that application wants ti recognize objects in many frames, however the program sent only 1. That's why the program extract only first element of a returned values (detections on the first frame). The time distance between frames are set shortly after that moment. The final step is sending all detected objects to a function which will return only vehicles including their boxes, masks, labels and confidence rate. This is exactly how OpenCV and Mask R-CNN detect objects.
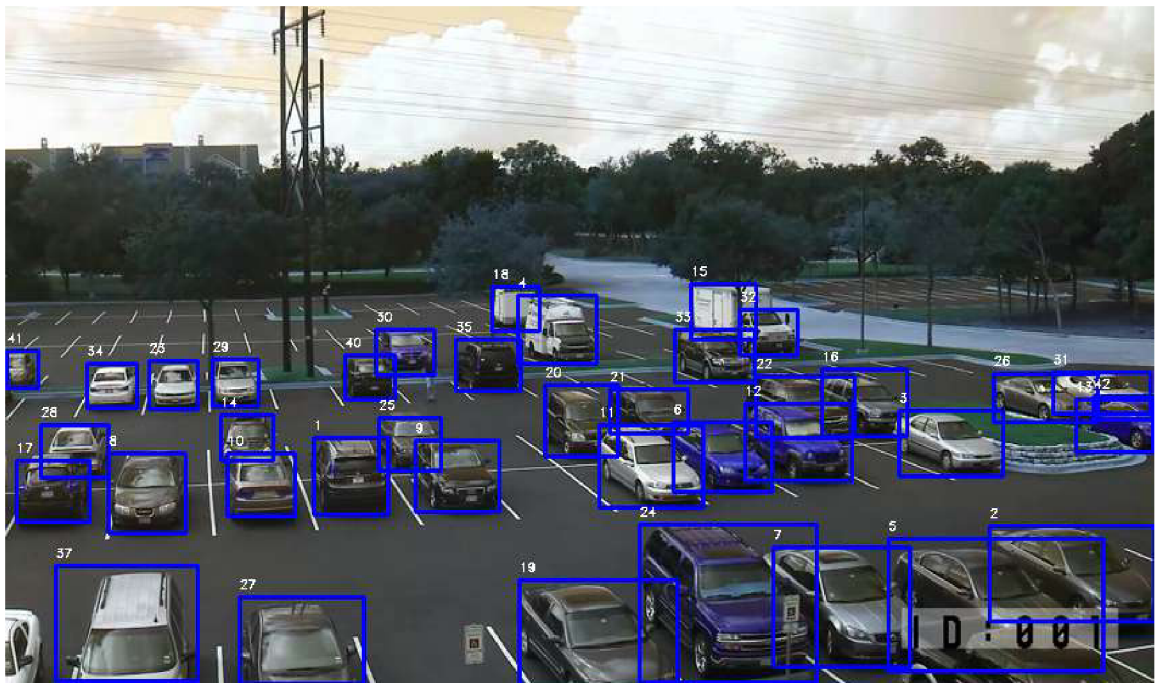

Figure 5.2: The frame converted from RGB to BRG color model

5.3 Implementation of the Parking Data Saving Algorithm

The algorithm of saving the information in described in the next steps. The detecting is processed every frame. In case of detection of a new car, the boxes are not created straight after. First of all the checks for duplicity and overlapping are processed, using the same algorithm that were described in section 4. Afterwards, values are inserted into the dictionary.

The key of the dictionary is a sting representation of a coordinates (left top and right bottom coordinates of a boxes). The value (of a dictionary) is an integer (default 0), it increases every time in case of one exact car is detected on the same coordinates. In case if car is moving car, the coordinates will be different, this helps the application not to make a parking lot from any detected car.

The dictionary is looped through every frame and in case if any key doesn't have any updates, the value of that key will be decreased. As soon as they reach -1 they will be deleted from the dictionary. The coordinates in the dictionary needs to be detected on the same place for 5 frames, otherwise they won't be appended to a parking boxes list. In one word this algorithm work life tachometer, in case if the rate of detection is the same for 5 frames it is a parking box, otherwise it is not. One more evidence for that the program should run for a few days until the full efficiency. The key with decreased values will be saved in the dictionary for the future purposes, they can be useful in the future.

Chapter 6

Testing & Evaluation of Achieved Results

This section will describe the testing and the methods of evaluation.

6.1 Testing the Vehicles Detection and Counting

The testing of the code was done in a simple way. In one word each new feature, bug fixing or just any code formatting caused the testing. That method of testing was called by me "right-away„ testing. The meaning of that is just to run the application every time something is added of fixed. This is the most sufficient way of testing for that kind of application.

Testing of detection of vehicles were made on 4 main data sets (are included in diploma files).The data set number 4 was collected by me. Data set number 3 was provided to me by supervisor. First data set were taken from internet[1], as same as the second one[2]. In addition the detection was tested on the random pictures taken from parking, and other data sets. Outcome of testings caused the change of the code. For instance the first testing caused the change of Mask R-CNN configuration, confidence rate and showed that the solving of the duplicity is necessary. More precise information about testing will be found below.

Testing of counting the vehicles and marking them as a parking slot were more deep. These were made mainly by printing out while implementing and comparing to values that was counted by me on the screen and values that were provided by OpenCV (see figure 3.2. As it was mentioned in the capital with implementation (see chapter 5), the dictionaries and bugs solving functions were printed and called respectively to achieve a higher result.

Additionally application was tested at night (data set number 3). The application detection of cars started to partially fail, according to observations the application rate of failure increases with the darkness, in case if there is at least little souse of light application will detect normally. More information could be found in section 6.3.

The detecting in the light changing (sun from different angles) was also tested, in that case no changes occurred.

---

[1]`https://github.com/eladj/detectParking/blob/master/datasets/parkinglot_1_480p.mp4`
[2]`https://github.com/olgarose/ParkingLot`

6.2   Bugs and their Solutions

These section will describe the appeared bugs and the way they were solved. The most popular appeared bugs are:

1. **Duplicity of parking boxes** – Comparing the corners coordinates, delete corresponding.

2. **Top semi-duplicity of parking boxes** – Comparing two X coordinates (left and right) and top Y coordinates, delete corresponding.

3. **Bottom semi-duplicity of parking boxes** – Comparing two X coordinates (left and right) and bottom Y coordinate, delete corresponding.

4. **Right semi-duplicity of parking boxes** – Comparing two right Y coordinates (top and bottom) and right X coordinate, delete corresponding.

5. **Left semi-duplicity of parking boxes** – Comparing two left Y coordinates (top and bottom) and left X coordinate, delete corresponding.

6. **Unreasonable enormous detection of nothing** – Finding the middle height of a parking boxes, delete all boxes that are distinctively bigger than mean.

7. **Wrong detection marked as empty parking slot on the first frame** – Delete all parking slots that appeared empty on the first frame.

One of the most common problems during the creation of the program was the vehicles detection problems. First issue was faced straight after the Mask R-CNN execution. Figure 6.1 illustrate huge problems with detecting.
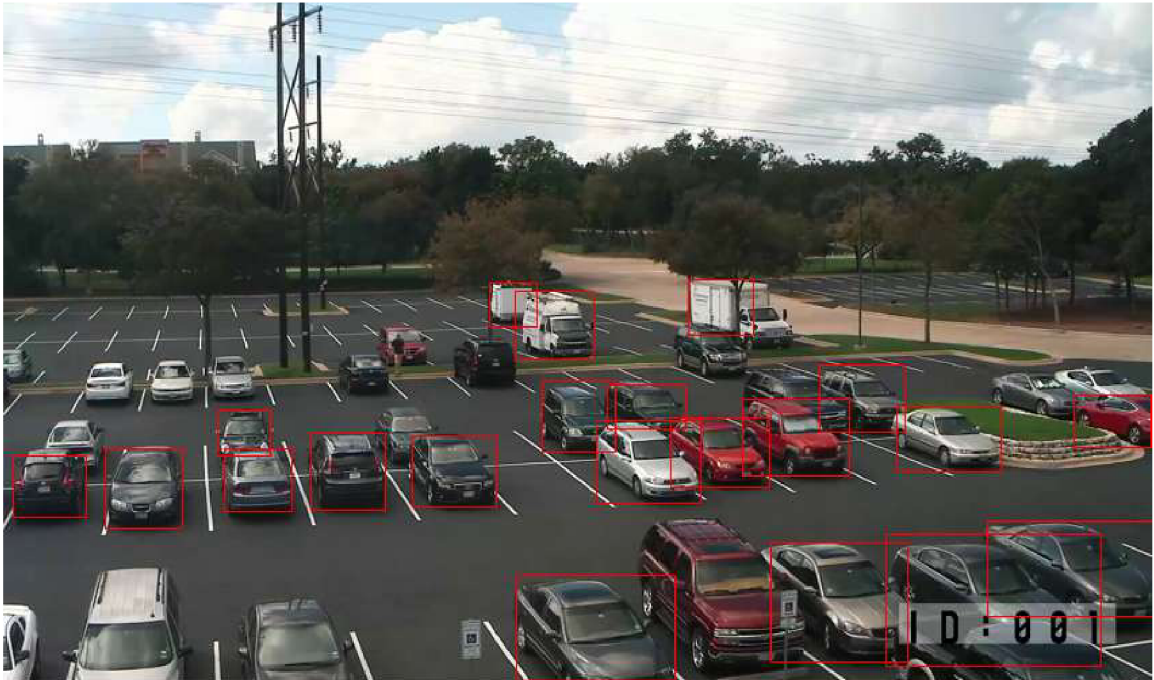

Figure 6.1: Mask R-CNN initial detection problems

Image demonstrates that in that period of time Mask R-CNN could detect only about 40% of a vehicles it should detect. These issues were solved by setting the correct options to the Mask R-CNN. First of all lower the percentage of confidence of showing. When confidence of detection is 60% Mask R-CNN won't detect a vehicle that he is not 60% (or more) sure is a vehicle. That's why by lowering the confidence to 10% and tell Mask R-CNN to detect all kind of vehicles the situation changed dramatically as it shown in a figure below. The other option for that would be retraining or additional training of the Mask R-CNN detector which I didn't find relevant due to the simplicity of the way the problem was solved.

Figure 6.2: MASK R-CNN detection after correcting the setting

As it is seen, after the detection become wider, some bigger bugs started to appear. On a figure 6.2 it seems that there is small red rectangle positioned close to number 5. In addition vehicle number 1 is detected two times, it could be understood from the width of a bounding. The first two bugs were called by me as duplicity and semi-duplicity respectively, the proposals for solving that problems were wide, starting from changing the detector or as it was mentioned before retraining the model. But almost all of the detectors will have the same issues, that's why I decided to solve this problem manually.

Finding all the boundings on a frame, get the coordinates of the boundings, and then compare each bounding with each other bounding in a frame. In case if some boundings are corresponding, or are located unreasonably close to each other(one in another), for instance from 1 to 5 pixels, one of the boundings is now counted as a mistaken and will be deleted from the parking lots. Furthermore while in the begging of an implementation these values were strict (up to 5 pixels) now it is changed to mean distance between all parking lots corners. These should allow the program to show same effectiveness on all kinds, height, width and resolution of a video.

The semi-duplicity (number 5) was solved in a bit different way, as it is visible from the figure 6.2 the duplicates here are most likely to be solved by processing the top two

coordinates. That means the left top X axes and right top X axes. The height for both of these coordinates will be the same. In general if the height and the X coordinates are corresponding to any other box, it will be counted as wrong, and will be deleted. The same idea of solving but with different coordinates will be used in case of semi-duplicity that is located at the bottom, right or left of the box. The only difference is using the bottom Y axis. For now it is already 3 functions that are used to solve the bugs, all of these function will be further combined with the car auto-detect function that will be described below.

Continue on programming, some further bugs appeared. As it could be seen from figure 6.2 two big green rectangles appeared on the right and left sides of the frame (number 4 and 3). It is obvious that there is nothing to detect in that positions, doing debugging I found out that detector identifies that elements as trucks. The thing that I have noticed while debugging that these spots are marked green straight from the first frame, I understood that it couldn't be possible due to the fact of that detector finds cars on a first frame, and makes a parking lot from every place that car is standing on, so it is an error of a detector. These were solved by checking all of the lots busyness on a first frame. If any lot that should be busy is marked as empty, program understands that this could not be possible and deletes that slot from parking lots, that kind of solution helped at all tested data sets.

One the most interesting bug could be found is placed in the center of figure 6.2, the parking lot (number 5) marked as an empty lot, whereas that exact place is not even a parking lot. Thanks to debugging I have found out that this exact bug is a detector's imperfectness. There are two ways this bug can be generated. First of it is an initial green parking lot marking, as it was mentioned above these bug's sub-type is solved by deleting the lot that are marked green straight on the first frame. The second way of generating is not always coming from initial detection, this could be also generated on a run time. This is solved by using the methods for solving duplicity and semi-duplicity, in combination both of these function are solving the out-coming issue.

The last issue is located next to number 6. As it could be seen, the big truck is detected two times. These are caused by the tree that is standing right in the middle of that car. This is the imperfectness of a detector, this kind of mistakes will be taken in account while creating the to overall evaluation statistics.

Regarding the corresponding boxes, program chooses which boxes to delete using special technique, the program is sorting the boxes from top to bottom that means that all the boundaries will be sorted from the smallest Y to biggest (from sky to floor at this data set). At the moment the program will compare boundaries it will take the top boundaries and delete the bottom ones. More precisely in case of duplicity number 5 (Figure 6.2) the program will first of all compare the small box with every other. When it will found that smaller box is corresponding with big one it will delete itself, not the bigger box.

These decisions are made after a huge research. All of the data sets proved that the top boundary is always (100 %) mistaken or wrong. Even on the figure 6.2 all the mistaken (duplicity) boundaries are located on top. In case of boxes next to number 2. The program will delete the middle box due to the fact that middle and bottom boxes are located extremely close to each other, according the overall ratio mentioned above. Top and middle boxes are located far enough from each other, that's why top box won't be deleted.

The figures below illustrate how application should detect new cars. There are four cars on the right side of the figure that have just arrived.

On the left side of the figure 6.3 the new arriving truck could be noticed. The truck will take his place in a few frames. Additionally, there is only one box around two vehicles on the right side of the left figure. This is caused by the red tree that stands next to the red

auto and confuses the detector. The tree is overlapping almost whole surface of the car. This bug is an imperfectness of a detector, it will be added to overall evaluation (see 6.3).

Figure 6.3: Detection of new arriving cars starting

In the figure 6.4 the vehicles are already marked as a parking places. These cars arrived and stopped at their places for 5 or more frames. Besides that application is perfectly detecting and processing other vehicles on the data sets. As it could be derived from image, the ideas described in the previous steps are now brought to life.
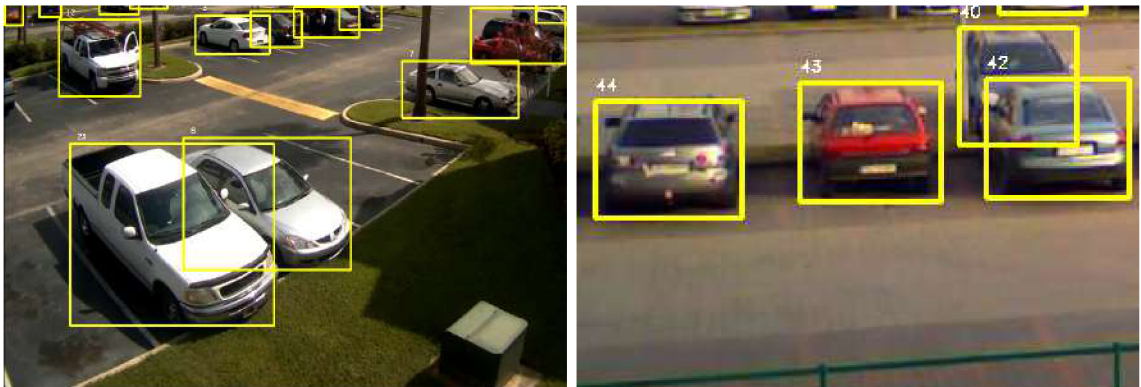

Figure 6.4: Mask R-CNN's new cars detection

6.3 Overall Evaluation & Success Rate

Overall evaluation will proceed on 4 data sets

1. Data set №1 – CCTV on moll parking

2. Data set №2 – CCTV on street (4 meters height)

3. Data set №3 – CCTV on street (14 meters height)

4. Data set №4 – CCTV on busy street (4 meters height)

Table 6.1: Overall Evaluation of an Application

	Dataset 1	Dataset 2	Dataset 3	Dataset 4
Total amount of cars (initial)	37	22	31	26
Actually detected cars (initial)	37	17	23	23
Accuracy of detecting	97.4%	88.2%	82.2%	88.4%
Percentage of actually detected cars	100%	77.2%	74.1%	88.4%
One frame processing time	3-4sec	4-5sec	3-4sec	3-4sec
Font color change (seconds)	5	10	8	5
Video's resolution	850x478	960x720	1920x1080	1280x720
Parking lot boundings change	10-15sec	12-15sec	13-15sec	13-15sec
Incorrect detections (initial)	6 cars	5 cars	8 cars	10 cars
Incorrect detections (bugs fixed)	1 cars	10 cars	8 cars	2 cars
Detection of a new arriving cars	100%	100%	100%	100%
Busyness detection	100%	100%	100%	100%
Night detection	NA	NA	50-70%	NA
Overall detection success rate	89.0%			
Overall accuracy rate	84.9%			

Initial amount of cars is the number of cars on the first frame, as it could be seen not all of the cars are detected initially, and some won't be detected at all.

On data frame № 1 (see figure 6.5) only one car which is located on bottom left is undetected. This is caused by the fact that only one third of a car is visible. Processing that parking lot is not logically correct due to the fact that parking lot area will be already divided by three (it also could be divided by 2,4 or more). As it was described in implementation section, the parking lot is counted as busy if overlapping with the car is more than 45 percent. However in that cases if auto is standing at 25% on initial parking slot, it will be marked as busy, exactly that's why the detection of that cars won't be reasonable.
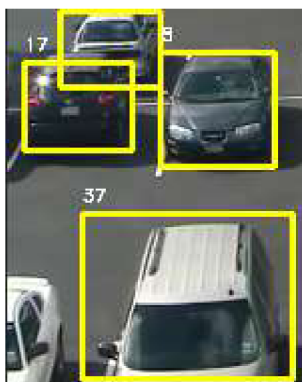

Figure 6.5: Undetected cars on first data set

On data frame №2 (see figure 6.6) the cars in the background are not detected. Obviously it is hard to detect the cars even by the human's eyes. Collecting information about that parking lot is not logically correct. That's why the detection of the cars here is 77.2% (see table 6.1).
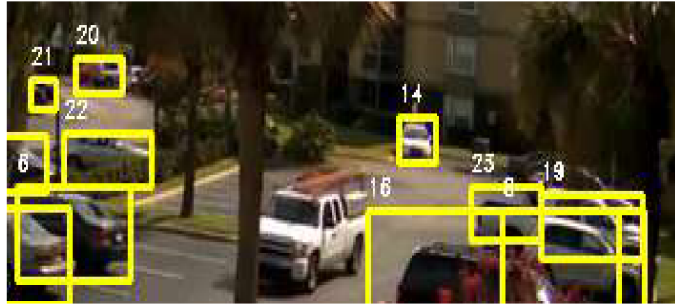
Figure 6.6: Problems with cars detection on second data set

On the data frame №3 (see figure 6.7) 4 cars are not detected. One car on the left side and three cars on the right side. Two cars on the right side are standing next to the lamp post, that is laying over them on the video. That's the reason why the detector couldn't detect that two vehicles. The vehicle on the top right is no detected due to the fact that car is visible only by 25 %. The vehicle on the left is not recognizes due to the angle of camera, detector recognize that two vehicles as one big lot, that's why there are only 3 parking boxes.
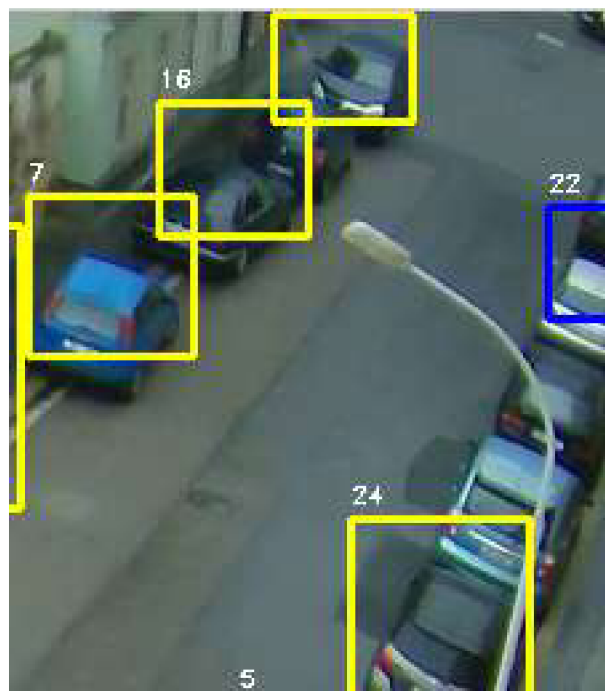

Figure 6.7: Undetected cars on third data set

On the last data №3 (see figure 6.8) 3 cars are undetected. The error of 2 cars in the middle is caused due to the error of detecting. The angle of the camera makes detector count these 2 cars as one, in the future iterations these cars will be counted as one too. The problem is that the size of the concatenated cars is bigger than mean, that's why it will be deleted. These issues were added to output evaluation. The car on the right bottom is not detected due to the fact that it is overlapped by a road sign, which confuses the detector.
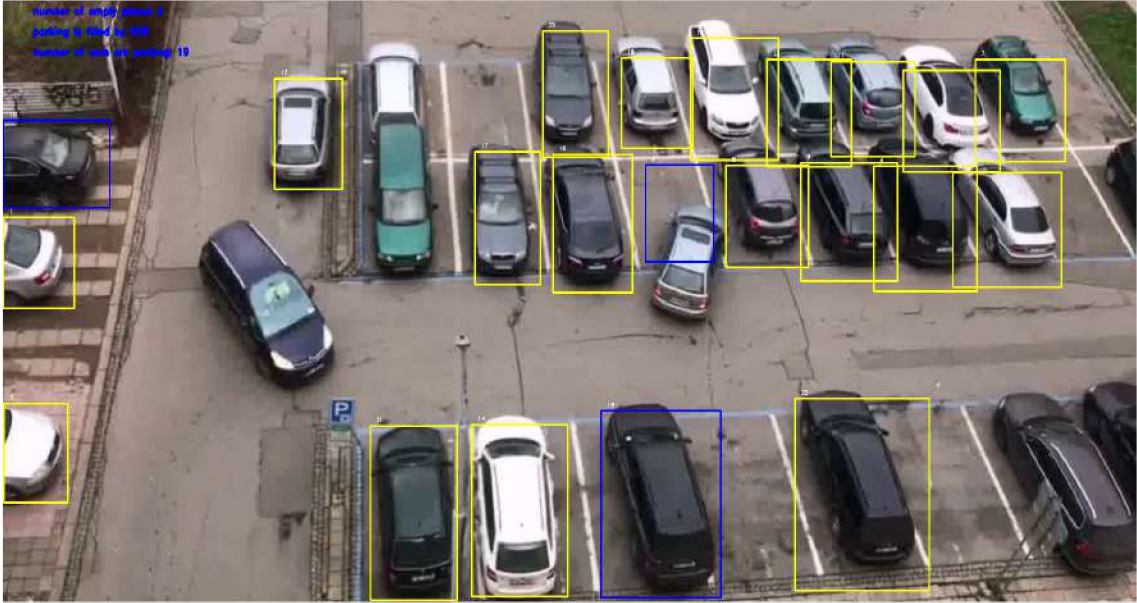
Figure 6.8: Undetected cars on fourth data set

In addition the same detection failures appeared exactly in this third set, as it could be seen from figure 6.9 the 2 cars are detected as one car. This is caused by the angle of the camera. The detector count these two cars as one. Moreover the car in the corner is marked wrong (as a little circle). That's make totally 3 cars with wrong detection. In addition the colors of that pairs of cars looks alike(4 cars in corner). In that particular cases detection fails. That's why that data set has the lowest detection rate. This will be added to overall evaluation (see 6.3).

Figure 6.9: Wrong detection of cars on third data set

The output of analyzing all four data sets showed that some cars should not be detected, taking than in account might change the overall detection success rate of all four data sets.

In that case the first data set will have 97.4% detection rate. Second data set's rate could be increased to 90.9% due to the two cars that still could be detected. The third data set's rate could be increased by 4 cars, that means that the rate now will be 87%. The last data set will not change it's value (see table 6.2).

Table 6.2: Overall Evaluation of an Application without cars that should not be detected

	Dataset 1	Dataset 2	Dataset 3	Dataset 4
Percentage of actually detected cars	97.4%	90.9%	87%	88.4%
Overall detection success rate	90.9%			

Overall success rate will be 90.9% in that case.

One more interesting thing two notice is a speed of font color changing and parking lot representation. The font color change is faster on the data set with smaller resolution. The first data set has smaller resolution than the third one, and both of them have smaller resolution than the second one. That's why time is significantly different. In case of representation change, all of the timing are in the gap from 10 to 15 seconds. The outcome from this is that the representation change depends only on the computer power.

Regarding the other features, the detection of actual parking busyness and all other statistics in general works on 100% according to the tests (counting manually and comparing information on the screen).

The final point is evaluating the night detection. The detection at night could be possible with at least little souse of light. In case of full darkness, not even a person, can detect a thing. Figure 6.10 perfectly illustrates that if any light source persist in that area the detection success rate will remain the same. The source of light are headlights of a car.
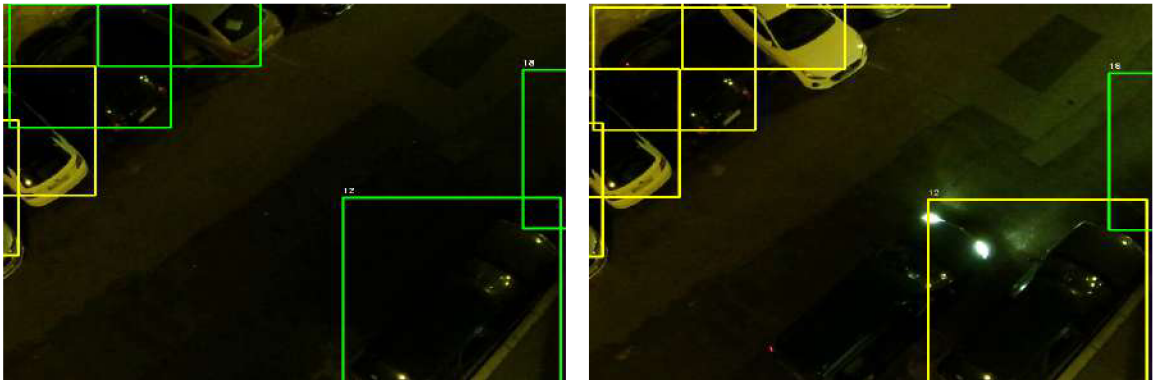

Figure 6.10: Night detection without light source and with light source.

Chapter 7

Conclusion

The aim of this diploma were first of all to get acquainted with computer vision, afterwards to learn how to detect objects on the videos, gather information about them, understand the principles, analyze already existing solutions, find helpful tools and decide how the application should work and behave. The detection then should have been performed on the parking lots and only vehicles should be detected. The information about the vehicles should be saved and analyzed, all of these informations and work that was done at the end gave the application that estimates the occupancy of a parking lots. The application works in the night, corrects itself, detects new vehicles automatically, does not crush, have statistics, font that is changing its colors and finally the capability to change representation of parking lots. Application is adaptable to any data set.The studies regarding computer vision took long time, a lot of time spent, a lot of articles and books were read and analyzed, a lot of implementation were tried, a lot of time were spent on coding. All of these at the end gave a good application with overall success rate 90.9%. The additional literature that were read during the implementation process could be found in bibliography [1] [5]. All of that book will give the read the complete understating of computer vision and are advised to read.

7.1   Possible Extensions

There are two main ways to extend the application. **First** idea is to focus on statistics and collect the bigger number of statistics. The possible ideas are:

1. The amount of cars for last 24 hours.

2. The most popular brand (color) of a vehicles in the parking.

3. The mean time of parking slot occupancy.

4. Amount of cars per month.

5. Busyness of parking by hours. (Most popular hour)

6. Total amount of hours that all cars spend on parking for 1 day.

All of these and many other statistics can be collected for the purposes of redesigning and optimizing the space on the parking all over the world. Implementation of all of these extensions will be using the same technologies for detecting the car's brands or colors. Also cameras in a good quality and some storage to save the information.

Second option is creating of a web application. With graphical interface, even faster optimization and other features that for example is saving or downloading data. This application will be bounded to some parking and can be accessed from any part of the world using Internet connection. The possible mock-up is illustrated on figure7.1.
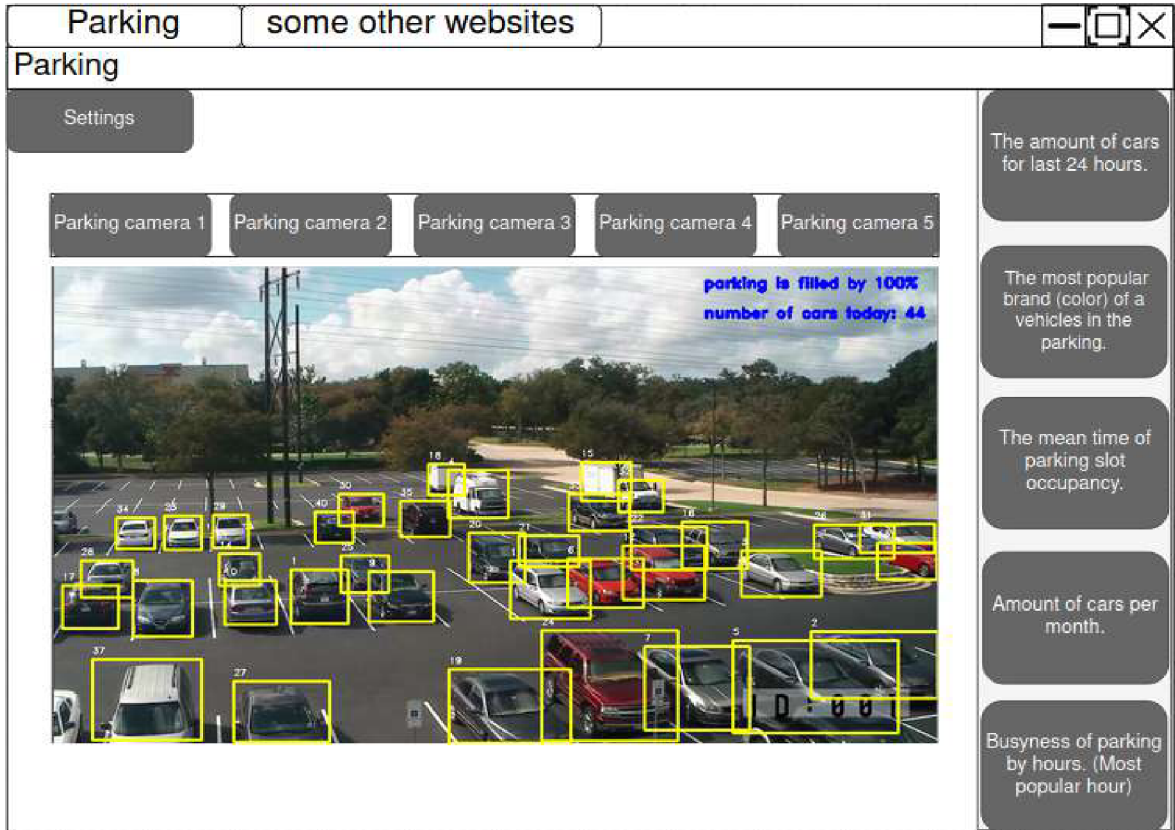

Figure 7.1: Possible design of a web application (Possible extensions).

The "Settings„ button will (for instance) allow to change the theme or colors of boxes. Switching between several cameras and obtaining information about them by pressing the buttons. That kind of application will be useful in the parking all over the world. The buttons on right sides will provide the detailed statistics about every parking. In addition application will be really fast and could be run on host.

Bibliography

[1] DOBEŠ, P., ŠPAŇHEL, J., BARTL, V., JURÁNEK, R. and HEROUT, A. Density-Based Vehicle Counting with Unsupervised Scale Selection. In: SPECIFIED not, ed. *Digital Image Computing: Techniques and Applications 2020*. Institute of Electrical and Electronics Engineers, 2020, p. 1–8. ISBN not specified. Available at: https://www.fit.vut.cz/research/publication/12360.

[2] DORRER, M. G. and TOLMACHEVA, A. E. Comparison of the YOLOv3 and Mask R-CNN architectures' efficiency in the smart refrigerator's computer vision. *Journal of Physics: Conference Series*. not specifiedth ed. IOP Publishing. nov 2020, vol. 1679, not specified, p. 042022. DOI: 10.1088/1742-6596/1679/4/042022. Available at: https://doi.org/10.1088/1742-6596/1679/4/042022.

[3] GARY BRADSKI, A. K. *Learning OpenCV*. Not specifiedth ed. O'Reilly Media, Inc, 2008. ISBN 9780596516130.

[4] HE, K., GKIOXARI, G., DOLLÁR, P. and GIRSHICK, R. B. Mask R-CNN. *CoRR*. not specifiedth ed. 2017, abs/1703.06870, not specified. Available at: http://arxiv.org/abs/1703.06870.

[5] JIANGI, X. Density-Aware Multi-Task Learning for Crowd Counting. In: SPECIFIED not, ed. *Not specified*. IEEE, 2020. DOI: 10.1109/TMM.2020.2980945. ISBN not specified.

[6] JOST, D. *What is an Ultrasonic Sensor?* [online]. 2019-10-7 [cit. 2021-14-04]. Available at: https://www.fierceelectronics.com/sensors/what-ultrasonic-sensor#:~:text=An%20ultrasonic%20sensor%20is%20an,sound%20that%20humans%20can%20hear).

[7] LIN, T., MAIRE, M., BELONGIE, S. J., BOURDEV, L. D., GIRSHICK, R. B. et al. Microsoft COCO: Common Objects in Context. *CoRR*. not specifiedth ed. 2014, abs/1405.0312, not specified. Available at: http://arxiv.org/abs/1405.0312.

[8] REDMON, J., DIVVALA, S., GIRSHICK, R. and FARHADI, A. *You Only Look Once: Unified, Real-Time Object Detection*. 2016.

[9] SZELISKI, R. Computer Vision, Algorithms and Applications. In: SPECIFIED not, ed. *Not specified*. Springer-Verlag London, 2011. ISBN 9781848829350.

[10] THORNE, B. Introduction to Computer Vision in Python. *The Python Papers Monograph*. not specifiedth ed. january 2009, vol. 1, not specified.

[11] WANG, C.-Y., BOCHKOVSKIY, A. and LIAO, H.-Y. M. *Scaled-YOLOv4: Scaling Cross Stage Partial Network*. 2021.

[12] WIKIPEDIA. *Parking sensor* [online]. Not specified [cit. 2021-14-04]. Available at: `https://en.wikipedia.org/wiki/Parking_sensor`.

[13] ZADEH, B. R. R. B. *TensorFlow for deep learning : from linear regression to reinforcement learning.* Not specifiedth ed. O'Reilly Media, Inc, 2018. ISBN 9781491980422.