BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF MECHANICAL ENGINEERING

FAKULTA STROJNÍHO INŽENÝRSTVÍ

INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

ÚSTAV AUTOMATIZACE A INFORMATIKY

COMPUTER VISION FOR AUTONOMOUS VEHICLES

COMPUTER VISION FOR AUTONOMOUS VEHICLES

MASTER'S THESIS

DIPLOMOVÁ PRÁCE

AUTHOR AUTOR PRÁCE	Bc. Michal Lečbych
SUPERVISOR VEDOUCÍ PRÁCE	Ing. Mhd Ali Shehadeh

BRNO 2022

Assignment Master's Thesis

Institut:	Institute of Automation and Computer Science
Student:	**Bc. Michal Lečbych**
Degree programm:	Applied Computer Science and Control
Branch:	no specialisation
Supervisor:	**Ing. Mhd Ali Shehadeh**
Academic year:	2021/22

As provided for by the Act No. 111/98 Coll. on higher education institutions and the BUT Study and Examination Regulations, the director of the Institute hereby assigns the following topic of Master's Thesis:

Computer Vision for Autonomous Vehicles

Brief Description:

The idea of having a self–driving car is not only appealing in fairy–tales and science fiction. In fact Autonomous vehicles is of great interest both within academia and industry fields, with applications for example in self–driving cars and trucks, and unmanned aerial vehicles.
Realizing autonomy is a hot research topic for automatic vehicles in recent years. For a long time, most of the efforts to this goal concentrate on understanding the scenes surrounding the autonomous vehicle. By completing low–level vision tasks, such as detection, tracking and segmentation of the surrounding traffic participants, e.g., pedestrian, cyclists and vehicles, the scenes can be interpreted.
The objective of this work is to familiarise the student with the theoretical and technological basis of computer vision, and in particular cover algorithms and tools for acquiring data by computer vision to be used as an input for motion plannig.
making sure to find a collision–free motion for a vehicle
from given start pose to given destination pose by studying event detection and intention prediction in autonomous driving.

Master's Thesis goals:

– descibing methods in computer vision which are for autonomous vehicles, including recognition, reconstruction, motion estimation, tracking, scene understanding, etc.
– getting familiar with neural networks application in comuter vision.
– building a computer vision model.

Recommended bibliography:

CHENG, Hong. Autonomous intelligent vehicles: theory, algorithms, and implementation. New York: Springer, c2011. Advances in computer vision and pattern recognition. ISBN 1447122801.

XUE, JR., FANG, JW. & ZHANG, P. A Survey of Scene Understanding by Event Reasoning in Autonomous Driving. Int. J. Autom. Comput. 15, 249–266 , 2018.

JANAI, Joel & GUNEY, Fatma & BEHL, Aseem & GEIGER, Andreas. (2020). Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art. 10.1561/9781680836899.

SZELISKI, Richard. Computer Vision: Algorithms and Applications. London: Springer, 2010. Texts in computer science. ISBN 978-1-84882-934-3.

SOLEM, Jan Erik. Programming computer vision with Python. Sebastopol: O'Reilly, 2012. ISBN 978-1-449-31654-9.

Deadline for submission Master's Thesis is given by the Schedule of the Academic year 2021/22

In Brno,

L. S.

doc. Ing. Radomil Matoušek, Ph.D.	doc. Ing. Jaroslav Katolický, Ph.D.
Director of the Institute	FME dean

ABSTRACT

Perceptive systems in autonomous cars are a heavily researched topic these days and an essential part of making fully autonomous vehicles possible. First, we make a short summary of the development of such a system, then we explain different approaches to make these systems possible, and we focus on object detection, as this will be the main part of our own created perceptive system. A new model for object detection is implemented, and some additional parts like distance estimation and lane detection are added.

ABSTRAKT

Percepční systémy v autonomních vozech jsou v dnešní době intenzivně zkoumaným tématem a nezbytnou součástí potřebnou k vytvoření plně autonomních vozidel. Nejprve, stručně shrneme vývoj takových systémů, vysvětlíme si různé přístupy potřebné k vytvoření percepčních systémů a zaměříme se na detekci objektů, protože to bude naše hlavní část pro námi vytvořená systém. Nový model pro detekci objektů je implementován, spolu s několika dalšími částmi jako odhad vzdálenosti a detekce jízdních pruhů.

KEYWORDS

self-driving cars, computer vision, deep learning, object detection, distance estimation, lane detection, CNN, YOLO

KLÍČOVÁ SLOVA

samořídící auta, strojové videní, hluboké učení, detekce objektů, odhad vzdálenosti, detekce pruhů, CNN, YOLO

INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

2022

BIBLIOGRAPHIC CITATION

MICHAL, Lečbych. *Computer vision for autonomous vehicles.* Brno: Brno University of Technology, Fakulta strojního inženýrství, Ústav automatizace a informatiky, 2022, 63 p. Master's Thesis. Supervised by Ing. Mhd Shehadeh Ali

AUTHOR’S DECLARATION

ACKNOWLEDGEMENT

I would like to thank my supervisor, Ing. Mhd Shehadeh Ali, not only for providing important literature and valuable advice in this expertise but also for the guidance during the writing part. More thanks belong to my family, for supporting me during my studies and helping me during my journey.

CONTENTS

1 INTRODUCTION

The automotive industry is one of the biggest industries in the world. As in many other industries, the process of trying to create automation systems, which would need minimal or no human intervention, has merged even to this field. Although there have been some attempts to create fully autonomous vehicles almost 100 years back, the first successful prototype was created in the 1980s. There were so many hard challenges yet to be solved that it seemed almost impossible to have even partially autonomous vehicles. But due to new innovative possibilities, we are approaching the phase, where it could be possible within a few years.

Driving a vehicle seems for most people like an easy task, but for automation systems, it is an unsolvable task yet. Existing approaches to self-driving can be roughly categorized into modular pipelines and monolithic end-to-end learning approaches. Both approaches require different approaches and face different problems.

The modular approach-based systems should be more reasonable, as they offer better safety. This means the problem of self-driving is broken down into several tasks, where each task is solved with a little bit different approach. Systems must have a great understanding of their surroundings and must be robust enough not to be dangerous for local places. This is done by leveraging machine learning, especially deep learning methods for computer vision tasks.

The popular tasks solved by machine learning are object detection, tracking, semantic (instance) segmentation, reconstruction, motion estimation, and scene understanding techniques. Perception systems mainly based on object detection do not require additional complex solutions and could potentially be used in simple devices, providing additional safety for many drivers.

Object detection has received significant attention in recent years. Original machine learning models do not come even close to the complexity and precision of models being used these days.

2 STATE OF THE ART

2.1 Brief History of Autonomous Driving

The start of partially autonomous vehicles began in the 1920s when the inventor Francis Houdina demonstrated a radio-controlled car, which was able to drive on streets without anyone behind the steering wheel. He equipped a Chandler Model with a transmitting antenna on the cargo bed and operated the Chandler Model from another car. Transmitted radio waves controlled electric motors in the car [38].

This car was able to start the engine, shift gears and use the horn without the driver. This new technology was presented in New York City in 1925, where the autonomous car was driving through heavy traffic [38].

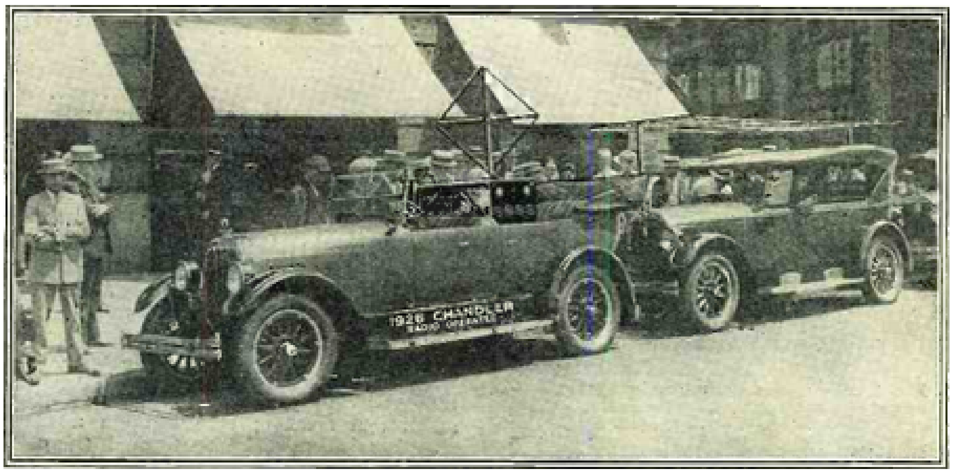


Fig. 1: The radio-operated automobile American Wonder [38].

Car manufacturers had a vision of creating self-driving cars before it was even possible. Several prototypes were created, such as the GM Firebird II in 1956, RCA Labs' wire controlled car in 1960 as well as a Citroen in 1970. However, these manufacturers never managed to make self-driving cars, which would not be wire controlled by people, and these prototypes were always limited to a specific use [7].

In the 1980s the Navlab was introduced, the first car which could be described in today's terms as a self-driving car and not radio-controlled. Research on computer-controlled vehicles began at Carnegie Mellon in 1984 and production of the first vehicle began in 1986. The research team introduced an imitation learning approach, where a neural network was optimized to keep the vehicle on the road. The car achieved a major milestone in the self-driving world, when the Navlab was able to drive from Washington, D.C. to San Diego, CA autonomously for 98% of the time. At this time neural networks approach started to overtake other models used and neural networks became the go-to option when it comes to self-driving [18].

In the early 1990s, Dean Pomerleau wrote a dissertation thesis describing, how neural networks could possibly control autonomous vehicles via image processing in real-time. The paperwork presents the learning system ALVINN (Autonomous Land Vehicle In a Neural Network), allowing to drive in single-lane paved and unpaved roads, multilane lined, and obstacle-ridden environments [31].

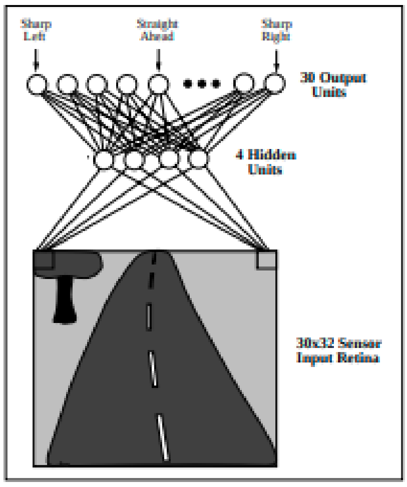


Fig. 2: Learning system ALVINN [31]

In Fig: 2 ALVINN learning system is shown. The input layer consists of 30x32 units onto which an image from the video camera is projected. It is then presented to the neural network as the input layer. The architecture of the model is very simple, as there was very little compute power back then, so the model has only 1 hidden layer consisting of 4 hidden units. The output of the layer is presented in 30 different units, which are then translated into the vehicle steering command [31]. Dean Pomerleau was not the only one using neural networks for self-driving cars. But his use of neural nets proved way more efficient than alternative attempts to manually divide images into “road” and “non-road” categories. In 1995 a more complete version of this car was developed, which managed to do 3 000 mile long ride.

The self-driving car development looked very promising at the time and several people thought, a fully self-driving car system could be developed within years. But many people were proved wrong in the early 2000s. Defense Advanced Research Projects Agency (DARPA) announced its first Grand Challenge in 2002 offering a $1 million prize to scientists from top research institutions if they could build an autonomous vehicle able to navigate a 240 km course through the Mojave Desert.

The challenge was held in 2004 and none of the 15 participants were able to complete the course [18].

In 2007, DARPA organized the next race. This competition required vehicles to drive a 96 km route through a town at George Air Force Base while obeying traffic laws, avoiding obstacles, and merging into traffic. The CMU research team was able to finish first, this team relied upon a multi-beam LiDAR. Multi-beam Lidar showed the best promising results when it came to obtaining depth measurements of the obstacles [18].

In the early 2000s were developed commercially used parking systems using computer vision techniques, able to do automatic parallel parking or angle parking.

Google secretly launched its self-driving car program in 2009 by hiring top scientists participating in the Darpa Challenges. Their program included a new driving platform and affordable multi-beam LiDAR scanners. Later in 2013 claiming, that they reached 300 000 miles of self-driving without a single accident. Many manufacturers were caught off guard by this result because Google was at this time way ahead of all competitors. By 2013 big manufacturers like GM, Ford, Mercedes, and BMW started working on self-driving technologies too. Nowadays Google's project is called Waymo and its system is based on multi-beam LIDAR, radar, and cameras [7] [6].

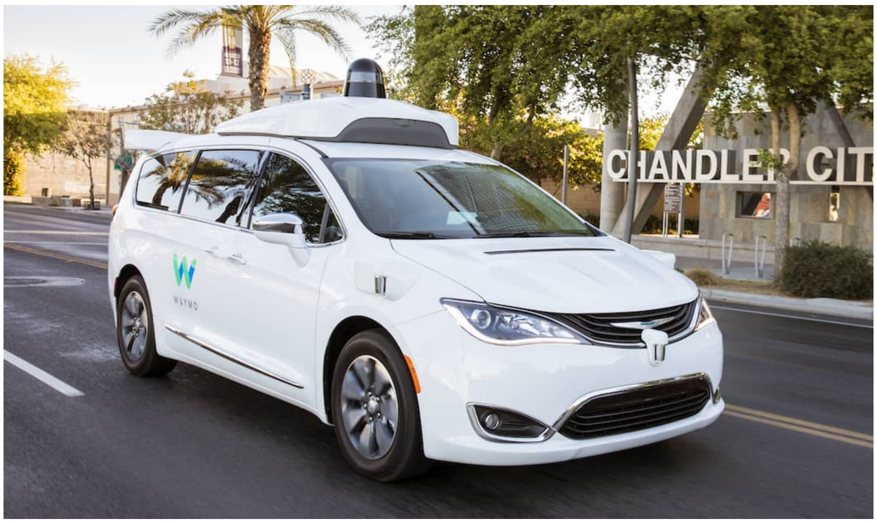


Fig. 3: Waymo driverless car [6].

In 2013 the S500 Intelligent Drive by Mercedes Benz was presented. Object detection and free-space analysis were performed using radar and stereo vision. Monocular vision was used for traffic light detection and object classification. A combination of these two techniques showed a more robust solution when it came to self-driving in more complex areas like inner-city environments or bad weather conditions [14].

Tesla announced the Tesla Autopilot, a driver-assistance system, in 2014. Initially, the goal was not to produce a production-ready system, which would offer a fully self-driving system, but rather the Autopilot would function as a complementary system to increase comfort and safety when conditions are clear. In November 2016 Autopilot added a function to create a point cloud to improve navigation during low visibility conditions. In February 2017 the Autopilot was able to navigate freeways, change lanes without driver input, transition from one freeway to another, and exit the freeway [39][40].

In 2016 NVIDIA joined the competition, as their advanced GPU production and strong processing power came in very handy when it came to the development of the self-driving cars. The NVIDIA research team presented paperwork, where a single CNN model was able to mAP the pixels from a single front-facing camera directly to steering commands [3].

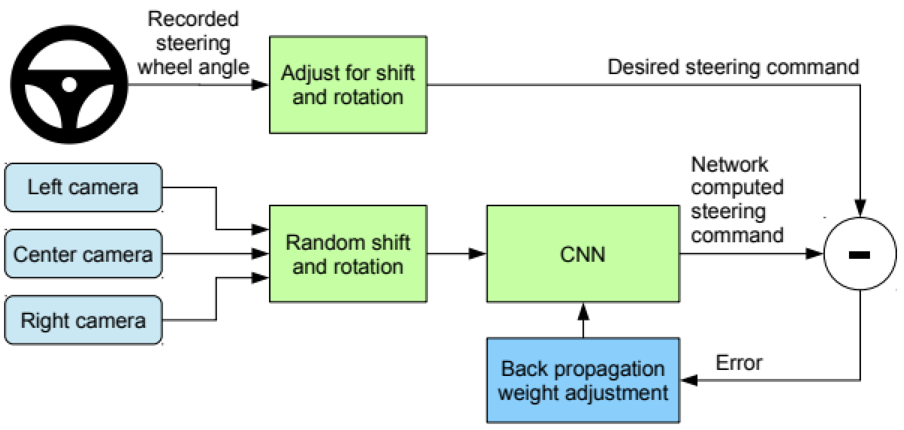

Recorded steering wheel angle
Adjust for shift and rotation
Desired steering command
Left camera
Center camera
Right camera
Random shift and rotation
CNN
Network computed steering command
−
Back propagation weight adjustment
Error

Fig. 4: End-to-end learning systems proposed by NVIDIA [3]

Results showed that CNNs are able to learn the entire task of lane and road following via imitation learning to predict vehicle control. There is no need for manual decomposition of the image, path planning, or semantic abstraction the model was able to learn directly from input images. This end-to-end approach proved to be surprisingly powerful [3].

NVIDIA with Volkswagen unveiled a new self-driving car chip, called Xavier, that incorporates artificial intelligence in 2018. The Volkswagen-NVIDIA collaboration is the first to connect A.I. to production-ready hardware. It opens up the possibility for self-driving cars to perform better [6].

2.2 Classification of Autonomous Cars

In 2014 classification system was introduced by SAE International (Society of Automotive Engineers). The system is based on six levels of autonomy.

1) Level 0 (No automation): Driver must take care of steering, throttle, braking, watch surroundings, and navigating through the world. There can be implemented warning systems and such. The majority of cars today are on this level of automation.
2) Level 1 (Driver assistance): Automated vehicles can handle the braking and basic turning for some circumstances, but the driver must still be ready to take over driving and the car cannot be left alone to drive itself.
3) Level 2 (Partial assistance): The last stage where the driver is responsible for monitoring the surroundings, traffic, weather, and road conditions.
4) Level 3 (Conditional assistance): Uses various driver assistance systems and artificial intelligence to make decisions based on changing driving situations around the vehicle. People inside the vehicle do not need to supervise the technology, which means they can engage in other activities. A human driver must be present.
5) Level 4 (High Automation): The automatic vehicle can handle most environments except some extreme ones.
6) Level 5 (Full Automation): The full performance of driving under all environmental conditions can be managed by an automatic driver. Human intervention is not needed at all [28].

2.3 Autonomous Drive learning

The more information we can get from the world the better the autonomous system can be built. The basic pipeline could look like this: perception devices scan the scenery and get the data from the surroundings that we use for specific algorithms to analyze the sensory data. The system then makes predictions to plan trajectory based on algorithms output and feeds the control. In the control module, the trajectory is then translated to the actuators [19].

These systems are usually complex systems made of numerous tasks like detection, segmentation, motion estimation, reconstruction, etc. There are two main approaches, to how autonomous driving systems are built. One is to consider autonomous driving as an end-to-end learning problem. That means one deep neural network is trying to learn tasks of perception, planning, and control directly from cameras to handle the steering. Another is to divide the whole learning process into subproblems, where they would take results from each component and then typically combine them in a planning module that feeds the control. [18].

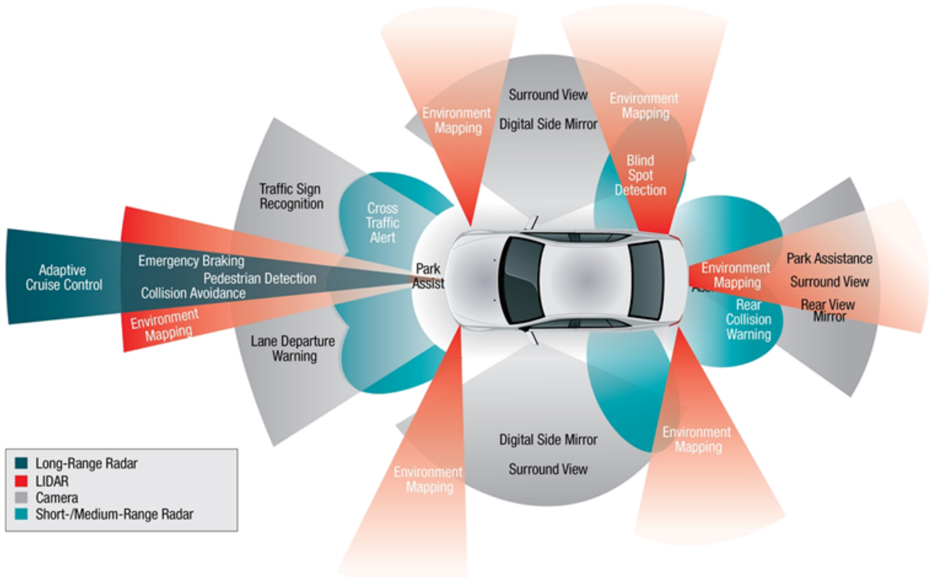


Fig. 5: Perception modules in autonomous car [19]

2.4 Scene understanding

The complex task of outdoor scene understanding involves several sub-tasks such as depth estimation, scene categorization, object detection and tracking, event categorization, and more. The goal is to understand the surroundings and get a compact representation of it. In contrast to modeling these problems in 2D, 3D reasoning allows geometric scene understanding and results in a more informative representation of the scene in the form of 3D object models, layout elements, and occlusion [18]. There are several ways, how we can obtain information from the surroundings. Some approaches are based on more complex hardware, others are not. First approach which could potentially offer very precise results is end-to-end learning.

End-to-End Learning for Autonomous Driving
Autonomous driving as a whole could be divided into 3 parts: perception, planning, and control. The perception module gathers information from the surroundings. The planning module forecasts the intention of other road users and computes a trajectory. The output of the planning module is passed to the control module, which finally calculates the final control output. Today, most autonomous cars use this paradigm, it enables the decomposition of a problem into simpler sub-problems [19].

However, in more complex scenarios this approach can become very limited. The perception provides the planning module with just limited info on detected cars, pedestrians, etc. While other information about this difficult scenario is lost [19].

End-to-end driving attempts to deal with such difficult scenarios. This approach maps raw input from all the sensors directly to the neural network, which takes control of other control systems like throttle, brake, and steering angle. Complex situations with detailed information can be encoded in high-dimensional feature space and preserved while being passed through the neural network. Moreover, the development of such a system is less difficult, because there is no need for handwritten rules, the whole network is learning end-to-end. This end-to-end learning could be divided into 3 different techniques:

Imitation learning- The goal of this technique is to clone the human driver by leveraging driving data in unsupervised learning [18]. In this approach, information from drivers' actions like throttle, brake, and steering is recorded at each step, so there is no need for any annotations. We are able to collect large amounts of training data at a low cost, by this type of data collection. Generally, there is one big disadvantage associated with this approach. Usually, to let the neural network perform well, we need datasets to be distributed equally. Better drivers will always face fewer failures. Therefore, self-driving systems will have a lack trained failure scenarios, in which systems could act accordingly [19].

Reinforcement learning- This approach is based on self-supervised learning. There is an agent, which tries to learn by itself via interaction with the environment. This is obviously a very dangerous and costly approach, so reinforcement learning is done via computer simulations most of the time. Learning is based on rewards instead of labels, where the goal is to maximize the reward accumulated over time. The agent could be rewarded for keeping in lanes where it is supposed to be or making turn maneuvers. Simultaneously the agent is penalized for bad behavior. This could happen when a car departs from the correct lane, crashes, or other dangerous scenarios. RL can prevent distribution mismatch (directly learns to drive by itself without the need for an expert) between the situations encountered during training and test. So when an agent encounters bad behavior in the real world, it is more likely to act on it. Reinforcement learning, in general, has the problem of long training time, because rewards are weaker and sparser learning signals than explicit labels in supervised learning. Usually, stimulating environments are not complex enough to match the real-world scenarios [18][19].

Direct perception - Finally, direct perception represents a hybrid approach of these two. The neural network tries to learn an intermediate interpretable representation. The network could directly predict the distance to the vehicle ahead and feed it to the controller. This additional information could improve the whole model, which is still based on a rule-based controller [19].

Now let us take a look at approaches, which try to deal with self-driving problems by processing the input data. Let us start with a more complex one.

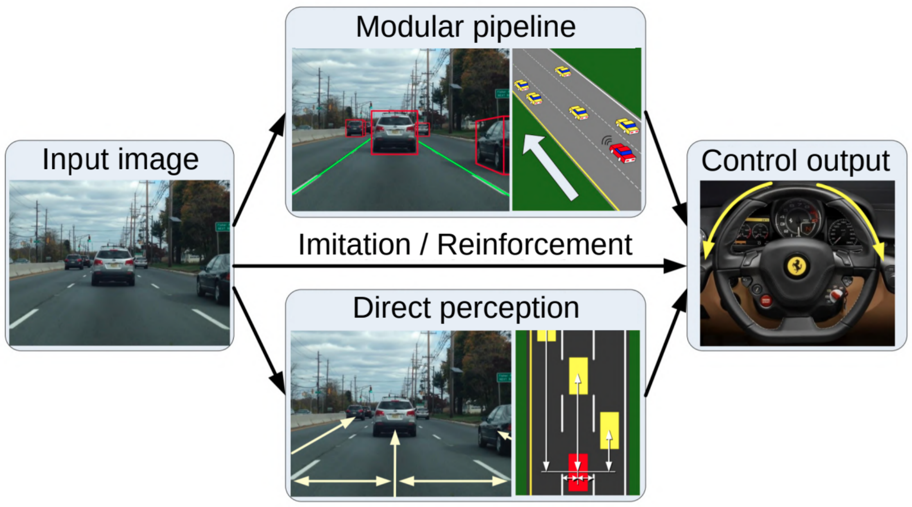


Fig. 6: Reinforcement learning [19]

Fusing sparse depth and dense RGB

Fusing sparse depth and dense RGB uses multi-modal inputs. In particular, the combination of camera and LiDAR which have a much higher resolution than radar and ultrasonic sensors. The LiDAR actively measures surroundings with laser beams, producing a sparse 3D point cloud. The camera is there to capture scenes and gain dense 2D image information. 3D data are projected into the 2D image space using extrinsic and intrinsic calibration.

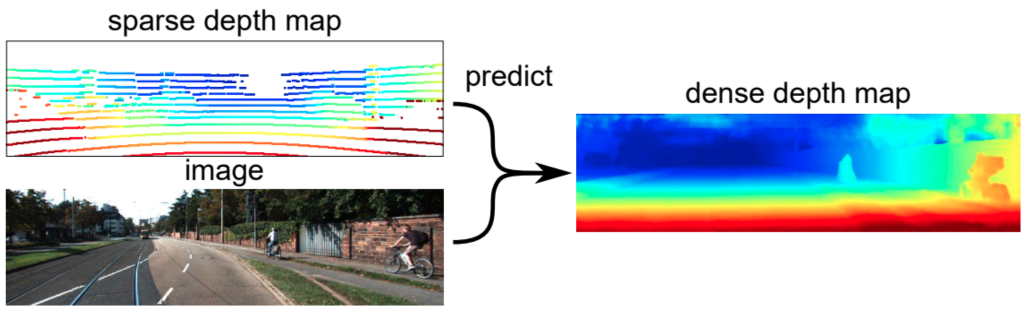


Fig. 7: Dense depth map fusion [12]

This is done in Fig: 7 by assigning the corresponding depth value to each projected 2D pixel. The neural network then completes missing points in sparse data from learned appearance priors [12].

For this kind of problem U-Net architecture is used [36]. The model consists of an encoder (downsampling part) and a decoder (upsampling part). Skip connections between encoder and decoder are realized by copying and concatenating the downsampled encoder features to the upsampled decoder features. Due to a large receptive field, the network allows the incorporation of context-aware depth predictions, which is particularly helpful in low-density regions [19].

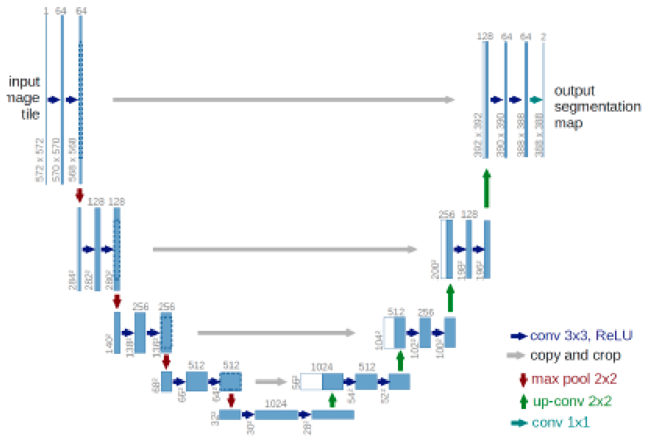


Fig. 8: Unet architecture [36]

Semantic segmentation

The goal of semantic image segmentation is to label each pixel of an image with a corresponding class of what is being represented. When we are trying to make predictions on a pixel level, this task is commonly referred to as dense prediction. The purpose is to do segmentation of images into regions that are typically found in street scenes (cars, pedestrians, or roads), which helps in the understanding of the surroundings to improve the self-driving car. The task could be difficult based on the complexity of the scene. Deep learning is found to be great for this task, especially their U-net type architectures [37].

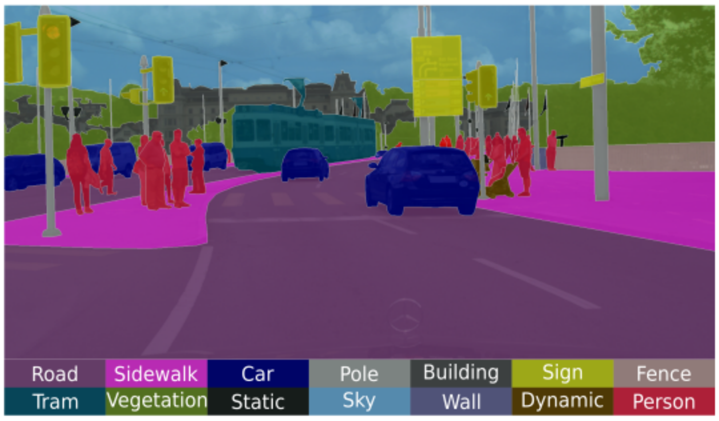


Fig. 9: Image segmentation [18]

Object detection

Object detection is an important computer vision task that deals with detecting instances of visual objects of a certain class. The research of object detection is usually done in two ways:

1) General object detection: explores different methods used for detecting different types of objects to simulate human vision.
2) Real-life applications: explores detection under application scenarios (pedestrian detection, face detection, text detection) [45].

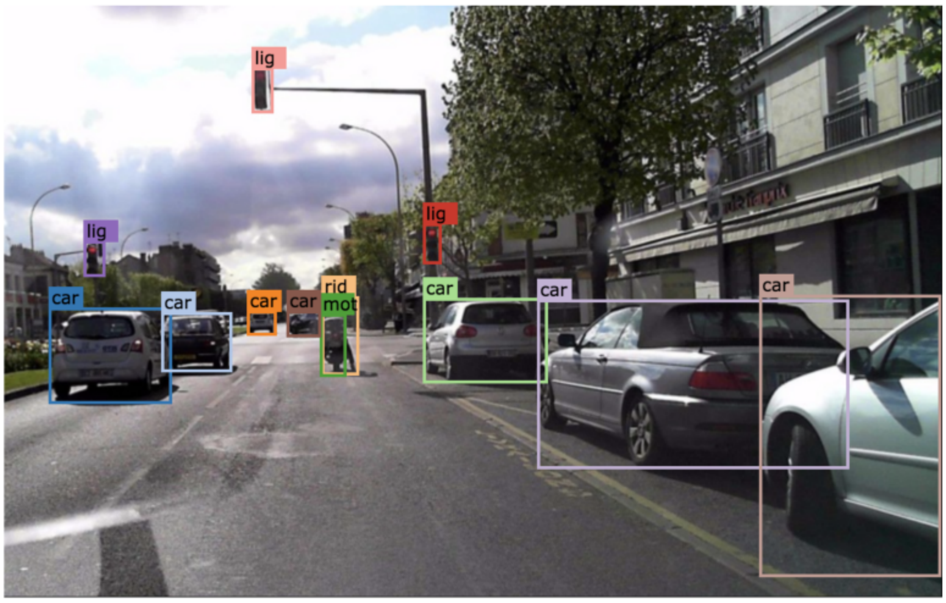


Fig. 10: Object detection [10]

Approaches based on end-to-end learning or LiDAR hardware are hard to replicate due to the additional complex setup needed. Additionally, segmentation tasks are more computational-heavy tasks when it comes to real-time systems and usually, pixel-level prediction is not needed for these tasks when object detection is precise enough. Due to this reasoning, I have chosen to build a neural network that could be implemented in an object detection-based system for a simple video camera, that can be found on everyone's cell phone. This is why this thesis will focus solely on object detection problems and models.

2.5 Object detection

Object detection is one of the most crucial requirements to realize autonomous driving. As there can be many other traffic participants like cars, pedestrians, animals, and other objects, it is necessary especially in urban areas to have awareness of these objects. The process of detecting pedestrians is particularly difficult because of their complex, highly varying motion, a large variety of appearances due to different clothing and articulated poses, and the interactions between pedestrians with each other and the world. Other problems can occur due to the nature of changing weather conditions. Older algorithms had problems with speed and generalization. But due to robust deep learning algorithms, new object detection systems were developed [45].

Difficulties and Challenges in Object Detection

Even though different detection tasks may have different challenges and may vary from each other, there are usually very common problems for most of them.

1) Object localization - determining object position is a major challenge in object detection. Researchers often use a multi-task loss function to cover both misclassifications and errors in localization.
2) Viewpoint variation - Since most models are trained and tested in ideal scenarios, it is a difficult task for detectors to recognize objects from different viewpoints.
3) Multiple aspect ratios and spatial sizes - The objects vary in terms of aspect ratio and sizes. Algorithms should be robust enough to catch these changes.
4) Deformation - Objects can be found in a different position from the one they were represented in the dataset.
5) Occlusion - objects occur partially in images.
6) Lighting - Illumination of the object plays a huge role, in how an object will be represented on a pixel level.
7) Cluttered or textured background - If the background of an image is cluttered or textured, there's a risk of the objects of interest blending into the background.
8) Intra-class variation - Objects within the same class could have completely different shapes and sizes.
9) Real-time detection speed - Algorithms should be close to real-time processing as it is possible.
10) Limited data - Detection datasets remain substantially smaller in scale and vocabulary than image classification datasets [26].

Models used in Object Detection

The pipeline for classical video camera object detection is usually very similar. It consists of steps like preprocessing, region of interest extraction (ROI), object classification, and verification or refinement. Older techniques usually used a sliding window approach, which was very computationally demanding. Later, as object detection became more important in several tasks, newer proposed techniques for reducing the search space were developed [45].

We will list some of the most known models that are used in object detection:

Viola-Jones detector
The Viola-Jones object detection framework is an object detection framework proposed in 2001 by Paul Viola and Michael Jones. The model combines the concepts of Haar-like Features, Integral Images, the AdaBoost Algorithm to create a system for object detection that is fast and accurate. The detector was hundreds of times faster than any other algorithms at that time when it came to comparable detection accuracy [21].

Haar features: are extracted from input images. A Haar-like feature consists of dark regions and light regions. It produces a single value by taking the sum of the intensities of the light regions and subtracting that from the sum of the intensities of dark regions. Instead of using a set of manually selected Haar basis filters, the authors used the Adaboost algorithm [21].

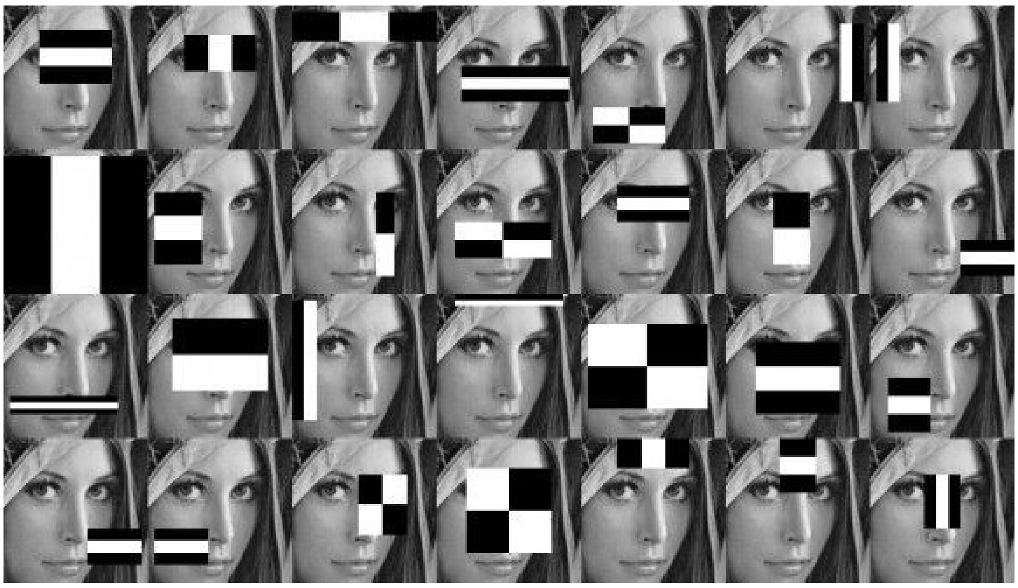
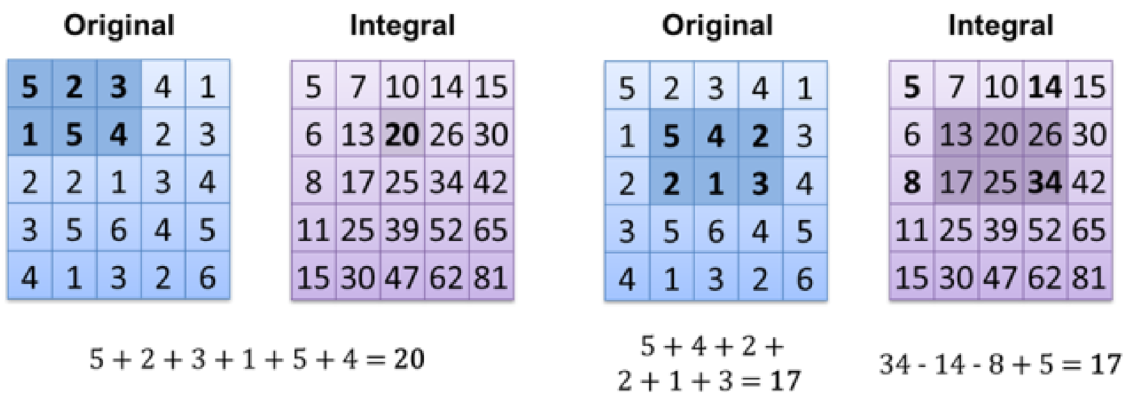


Fig. 11: Haar-like features [25]

An Integral Image: is an intermediate representation of an image where the value for location x, y on the integral image equals the sum of the pixels above and to the left of the x, y location. So instead of calculating by looping through all pixels 1 by 1, it can be calculated in constant time, by multiplying the rectangular regions above and left. Since Viola-Jone's algorithm involves calculating the sum of dark/light rectangular regions, while extracting Haar-like features, this intermediate representation allows for fast calculation of rectangular regions.



5	2	3	4	1
1	5	4	2	3
2	2	1	3	4
3	5	6	4	5
4	1	3	2	6

5	7	10	14	15
6	13	20	26	30
8	17	25	34	42
11	25	39	52	65
15	30	47	62	81

Fig. 12: The integral image calculation [43]

The AdaBoost algorithm: is a machine learning algorithm made up of linear combinations of weak classifiers for selecting the best subset of features among all available features. The output of the algorithm is a strong classifier. The technique counts occurrences of gradient orientation in the localized portion of an image. It performs better than any other edge descriptor as it computes the magnitude as well as the angle of the gradient to compute features [21].

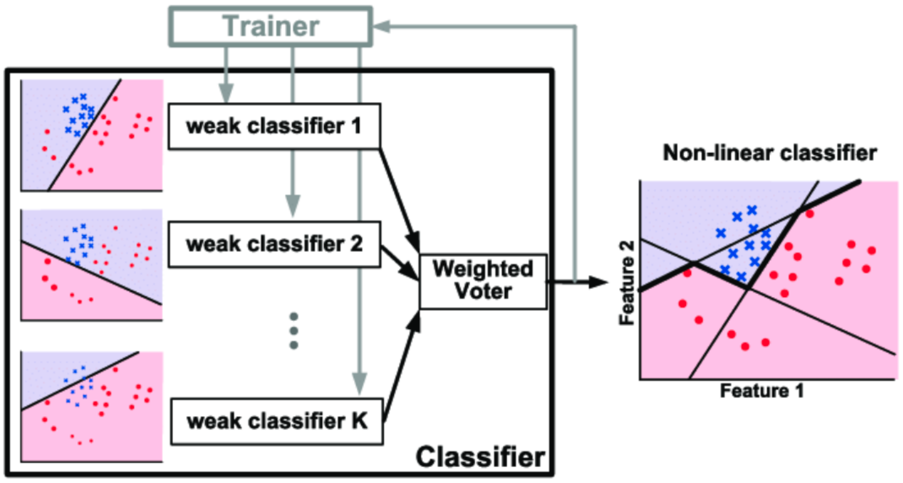


Fig. 13: The AdaBoost algorithm [21]

In 2012, the world witnessed the rebirth of convolutional neural networks [20]. R. Girshick took the lead in state-of-the-art models by proposing the RCNN in 2014. Since then deep learning models took the lead and object detection models started to evolve at an unprecedented speed. The new deep learning models can be divided into two groups: one-stage detector and two-stage detector. These state-of-the-art models for object detection have evolved over time and are now considered a strong foundation for much more powerful networks existing today [45].

R-CNN

The first model, R-CNN, which stands for region-based convolutional neural network, consists of three modules. The key concept is region proposals, which are created by a selective search algorithm. These region proposals are then used to localize objects within an image.

Selective search:

1) Generate many candidate regions.
2) The greedy algorithm will recursively combine similar regions into larger ones.
3) Generated regions will be used to produce final candidate region proposals.

The selective search considers four types of similarities when combining the initial small segmentation into larger ones.

1) Color Similarity - The histogram of each channel of RGB image is generated. Similarity computed by:

$$S_{color}(r_i, r_j) = \sum_{k=1}^{n} min(c_i^k, c_j^k)$$

(1)

where c_i, c_j is k^{th} value of the histogram bin of region r_i and r_j respectively.

2) Texture Similarity: calculated using generated 8 Gaussian derivatives of the image and extracting the histogram with 10 bins for each color channel. Then we get 10 x 8 x 3 = 240 dimensional vectors for each region.

$$S_{texture}(r_i, r_j) = \sum_{k=1}^{n} min(t_i^k, t_j^k) \tag{2}$$

where t_i, t_j is k^{th} value of the texture histogram bin of region r_i and r_j respectively.

3) Size Similarity : The idea is to make smaller regions merge more easily. Otherwise larger regions would keep merging with larger regions.

$$S_{size}(r_i, r_j) = 1 - (size(r_i) + size(r_j))/size(img) \tag{3}$$

where $size(r_i)$, $size(r_j)$ and size(img) are sizes of regions r_i, r_j and the image the respectively in pixels.

4) Fill Similarity : Measures how well two regions fit with each other. If they fit, they will be merged.

$$S_{fill}(r_i, r_j) = 1 - (size(BB_{ij}) + size(r_i) - size(r_j))/size(img) \tag{4}$$

where $size(BB_{ij})$ is the size of the bounding box around i and j [29] [41].

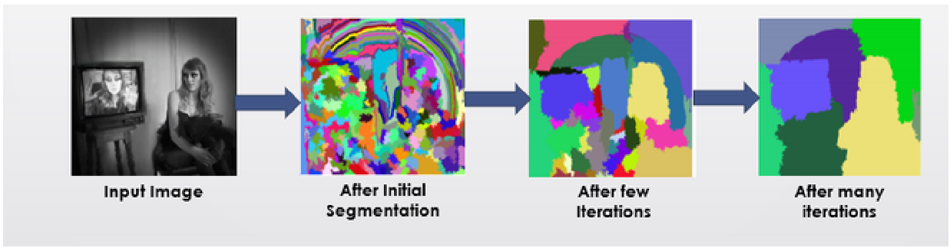


Fig. 14: Selective search [29]

The first module of R-CNN generates category-independent region proposals by using a selective search algorithm. These 2000 candidate region proposals are rescaled into a square and fed into the next module. The second module is a convolutional neural network, which acts as a feature extractor, and produces 4096-dimensional features from the region as output. And the third module, called the SVM classifier [42], is used to predict the presence of objects in that candidate region. In addition to predicting the presence of an object, SVM also predicts four values which are offset values to obtain better precision of the bounding box. For example, if there is the presence of just a part of the object in the image, offset values help in adjusting the bounding box [11].

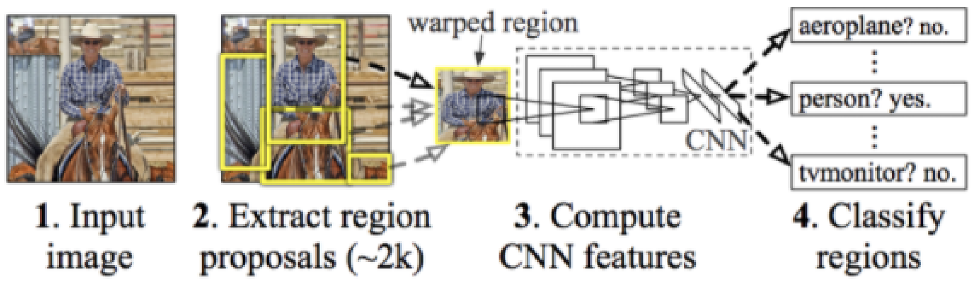


Fig. 15: The R-CNN model [11]

R-CNN architecture was revolutionary at that time, but the redundant feature computation of the overlapping proposals took too long to be usable in self-driving systems [45].

Fast R-CNN

As for the next generation, the same author solved some of the drawbacks. This new updated model was named Fast R-CNN. The model comes with a new idea, where the image is fed to the CNN just once, to generate convolutional feature maps, which are fed to a Region of Interest (RoI) pooling layer. The proposed new RoI layer extracts equal-length feature vectors from all proposals in the same image. So these feature vectors can be fed into a fully connected layer. From the RoI feature vector, we use a softmax layer to predict the class of the proposed region and also the offset values for the bounding box. By this procedure, we can avoid the classification of 2000 region proposals, instead, convolution is done once per image, and a feature map is generated [9].

Fast R-CNN got rid of the SVM classifier and instead used Softmax [27]. Fast RCNN enables us to simultaneously train a detector and a bounding box regressor under the same network configurations [45].

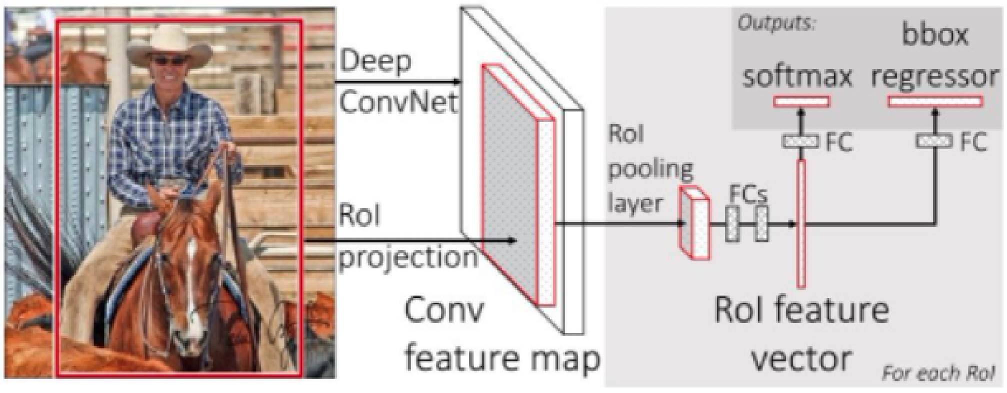


Fig. 16: The Fast R-CNN model [9]

Faster R-CNN
It is the first near-real-time deep learning detector. That is due to the introduction of the Region Proposal Network, which generates proposals with various scales and aspect ratios. The concept of anchor boxes is firstly introduced. An anchor box is a reference box of a specific scale and aspect ratio [9].

The input image is resized into 600 x 1000 pixels. The VGG-16 was used as the backbone, achieving the state-of-the-art object detection accuracy [34]. The network has to learn whether an object is present in the input image at its corresponding location and estimate its size. For each location on the output of the feature map, sets of 9 anchors are placed. These anchors indicate possible objects of various sizes and aspect ratios at this location.

512 feature maps are obtained from the backbone part. The next layer is the 3 x 3 convolution layer, which divides the output into 2 branches. The first one is a 1 x 1 convolution layer creating 36 feature maps for bounding box regression. The outputs are 4 regression coefficients for each anchor, which are used to improve the coordinates of the anchors that contain objects. The second branch outputs 18 feature maps for classification. This output is used to give probabilities of whether or not each point in the backbone feature map contains an object in all 9 anchor boxes [9] [34].

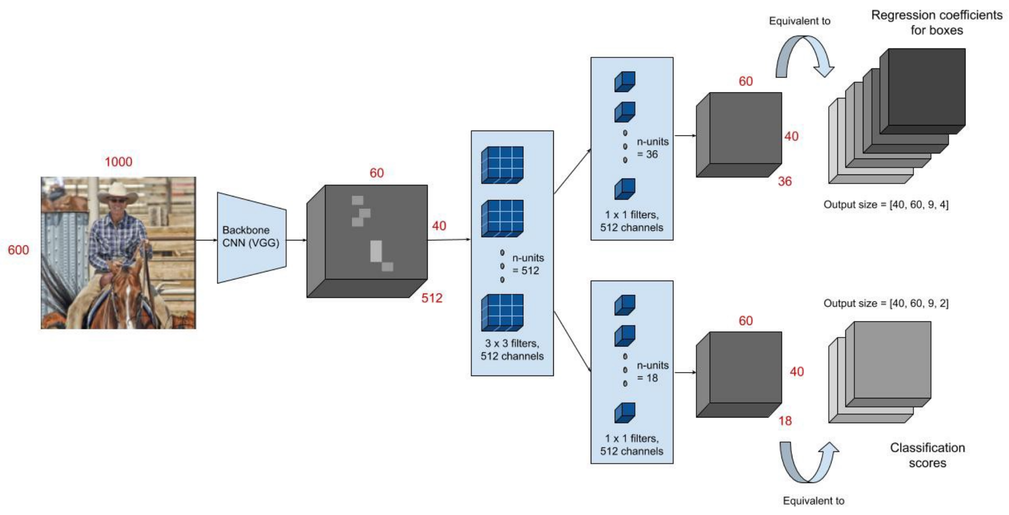

1000
600
Backbone CNN (VGG)
60
40
512
n-units = 512
3 x 3 filters, 512 channels
n-units = 36
1 x 1 filters, 512 channels
n-units = 18
1 x 1 filters, 512 channels
60
40
36
60
40
18
Equivalent to
Regression coefficients for boxes
Output size = [40, 60, 9, 4]
Output size = [40, 60, 9, 2]
Classification scores
Equivalent to

Fig. 17: Faster R-CNN model [2]

The proposals are generated using a network, which can be trained end-to-end to be customized for the detection task. It produces better region proposals compared to generic methods like Selective Search

SSD

SSD consists of two components: the backbone part and the SSD head. For the backbone, the pre-trained neural network is usually used, which works as a feature extractor. The last convolution layer divides the image into 38 x 38 grid, where each grid cell is responsible for detecting objects in that region of the image. For each cell, it makes 4 object predictions. Each prediction consists of a boundary box and 21 scores for each class (20 classes + 1 non-object class), where the class with the highest score is picked. As CNN reduces the feature layer dimension, the resolution of the feature maps also decreases, so the model is able to make predictions faster [22].

SSD was proposed in 2015. The main advantage of SSD is the introduction of the multi-resolution detection technique, due to which, the detection accuracy of one-stage detectors was improved.

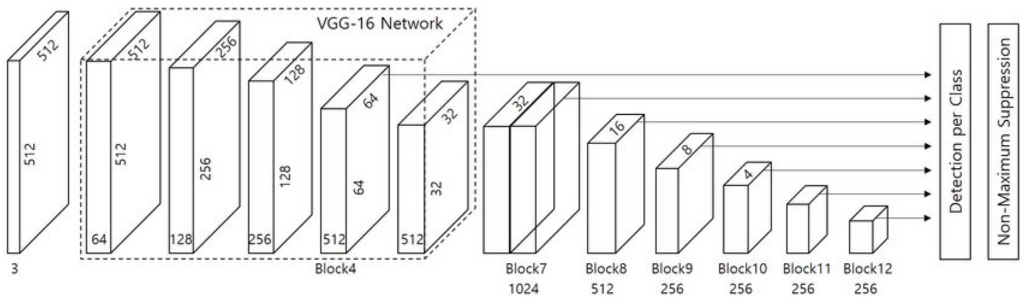


Fig. 18: SSD model model [22]

YOLO

YOLO was proposed in 2015 by the Facebook AI Research team. The network only looks at the image once to detect multiple objects. Thus, it is called You Only Look Once. By being able to make predictions when looking at an image only once, the detection can be done in real-time. At that time it was a state-of-the-art deep learning object detection approach. YOLO combines a single neural network to perform both classification and prediction of bounding boxes for detected objects. It is highly optimized for detection performance and can run much faster than running two separate neural networks to detect and classify objects separately [32].

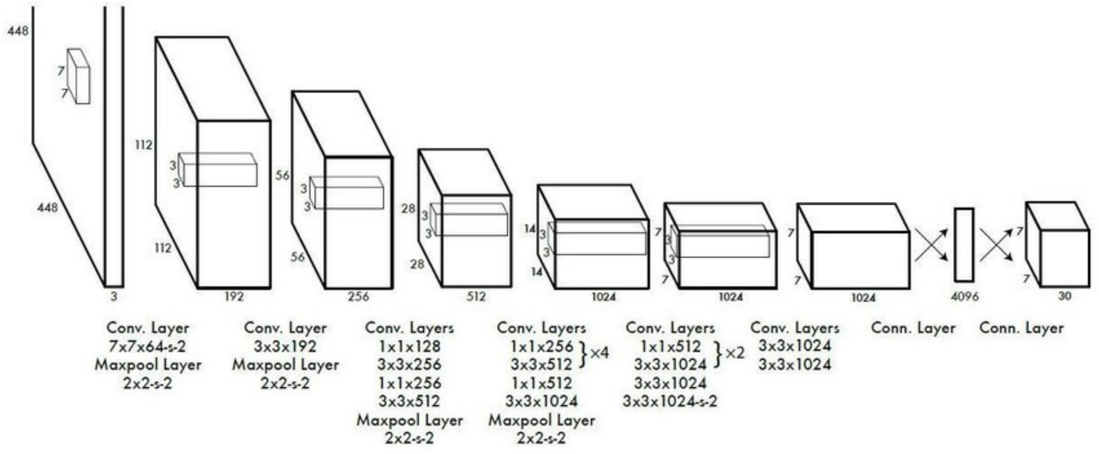

Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 ×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 ×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

Fig. 19: The YOLO model [32]

The input image is divided into a 7x7 grid, where each grid cell is responsible for detecting the object if it contains the center of that object. Each cell predicts 2 bounding boxes and estimates scores for those boxes. The confidence score represents how sure the model is, that the box contains an object. Bounding boxes contain 5 values. The first 2 values represent the coordinates of the center of the box relative to the bounds of the grid cell, third and fourth values represent width and height. The last value represents confidence between the predicted box and ground truth box [32].

3 MODEL IMPLEMENTATION

3.1 Model architecture

The main inspiration for our model was taken from the YOLO-v3 architecture [33]. YOLO-v3 comes from the very famous YOLO series, and this particular model offered the best precision/speed trade-off in 2018 for object detection. The model created in this thesis, further just called MyModel, could be divided into 3 main parts. These 3 main parts will be called the backbone, the neck, and the head.

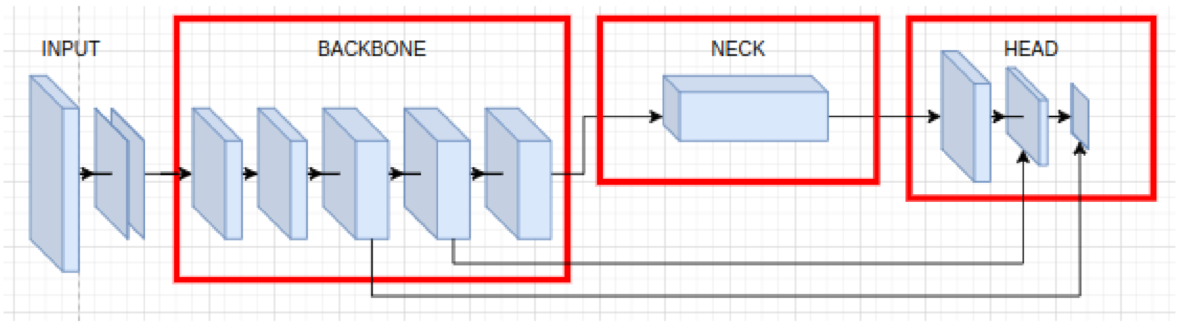


Fig. 20: MyModel architecture

Before explaining the whole architecture, let us talk about one main problem, which can occur, when creating a new architecture.

Bias and variance trade-off problem: The goal of any supervised machine learning model is to best estimate a function f for the output y given the input data x. These supervised models are often not able to perfectly fit function f. The difference between f(x) and y is called prediction error, which could be further divided into bias error and variance error. Bias is the simplifying assumption made by a model to make the target function easier to learn. The variance is the amount that the estimate of the target function will change if different training data was used. Good prediction performance by the model should be achieved by keeping low bias and low variance for the model. Basically, we are trying to create a model large enough to be able to learn most of the dataset, but not large enough to overfit the dataset.

The input

Input mages in our model are in the shape of Bx3x416x416. The images are represented as a four-dimensional tensor, where B is the batch size, 3 is the number of channels (RGB in this case), and the last two dimensions are the width and height of the image.

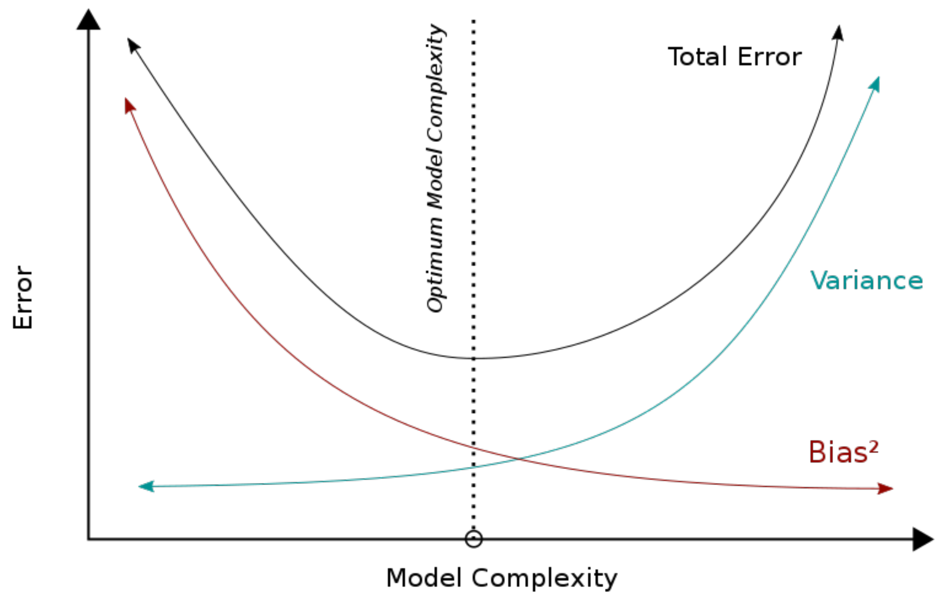


Fig. 21: Bias-variance trade-off [16]

The backbone

This is the main part of the MyModel, where most of the upgrades to the official version happened. The backbone part here refers to the feature extraction part and starts with two convolutional blocks (2D convolution + 2D Batch Normalization + Mish) followed by ConvNeXt Block, which is introduced in 'A ConvNet for the 2020s' paperwork [23].

The backbone has 4 layers in total, where each layer is made of a ConvNeXt block followed by one convolution layer. The number of input feature layers is doubled every subsequent layer, so we can obtain more additional information from the convolution layers.

The ConvNeXt block has an inverted bottleneck structure. The hidden part of the block is four times wider than the input dimension. The ConvNeXt block also introduces a 7x7 kernel, which was often not used in older architectures, as it made the model more computationally expensive. But newer SOTA architectures are known to have larger receptive fields, so models with larger kernel sizes could possibly benefit from that. The 7x7 kernel size is followed by Batch Normalization. Point-wise 1x1 convolution is here done by linear layer, followed by Mish function. The hidden size of the ConvNeXt block is converted back to its previous size by a linear layer. ConvNeXt block also uses grouped convolutions, multiple kernels per layer, resulting in multiple channel outputs per layer. This leads to wider networks that help a network learn a varied set of low-level and high-level features [10]. For my type of task, The ConvNeXt block was found to work better with batch normalization and Mish function.

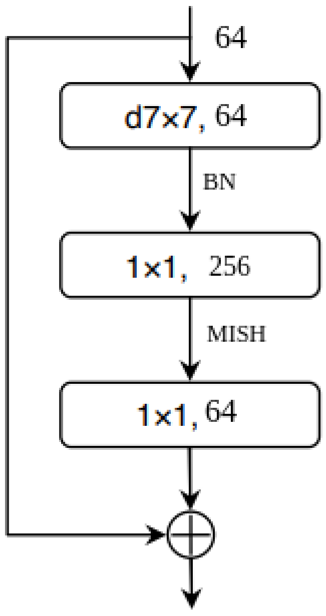


Fig. 22: Modified ConvNeXt block

Micro Design explained

Activation function: the purpose of activating a function is to add nonlinearity to a neural network, allowing the network to learn complex patterns in the data. The Rectified Linear Unit (ReLU) function was used in CNN models for many years and became the most used option for most architectures. This is not surprising because ReLU offers many advantages:

1) ReLU takes less time to learn and is computationally less expensive than other common activation functions like tanh, sigmoid.
2) ReLU involves simpler mathematical operations than most of the other activation functions.
3) It can solve the problem with vanishing gradient, where gradients shrink drastically in backpropagation.

But there is one problem with this activation function, called the dying ReLU problem. The dying ReLU problem refers to the scenario when many neurons have negative values, so the activation function will have the value 0, which potentially worsens the network. When most of these neurons return zero output, the gradients do not flow during backpropagation, and the weights are not updated. So some parts of the network become dead and it is likely to remain unrecoverable.

This is where the Mish activation function comes in, solving the problem, by assigning non-linear output of the activation function for negative input values. in negative values. This activation function is now widely used in some modern architectures.

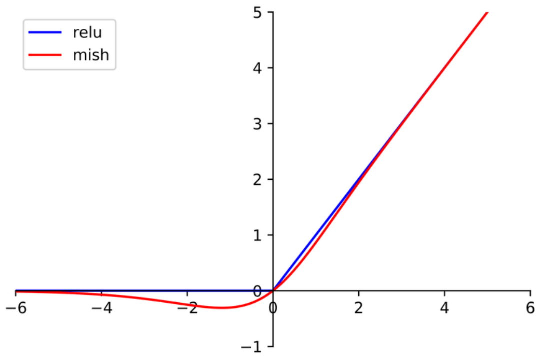


Fig. 23: The ReLU vs. Mish activation function [15]

Normalization: is used to normalize the output of neurons. This speeds up the convergence of the training process. There are two main types of normalization widely used these days. The first is batch normalization, which is often used in CNNs and was also used in the original YOLOv3, and the second is called layer normalization, used mostly for NLP tasks.

Batch normalization was chosen as it was found to work better for this type of problem.

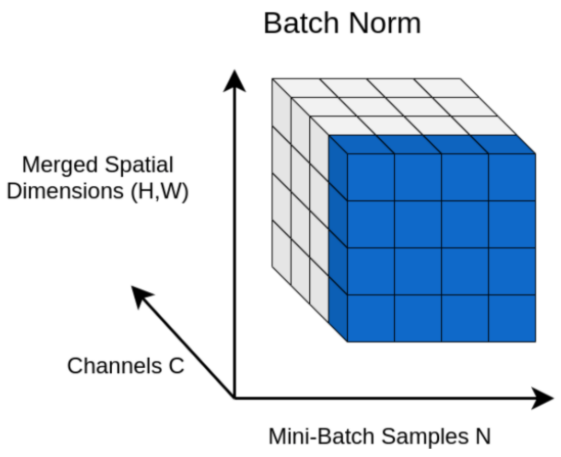


Fig. 24: Batch normalization [1]

The neck

The purpose of the neck block is to add additional layers between the backbone and the head. Therefore, the head's input will contain spatial rich information. The neck part is a pooling layer, called spatial pyramid pooling. The spatial pyramid pooling maintains spatial information in local spatial bins. In each spatial bin, responses of each filter are pooled. The Spatial Pyramid Pooling can generate a fixed-length representation regardless of image size [13].

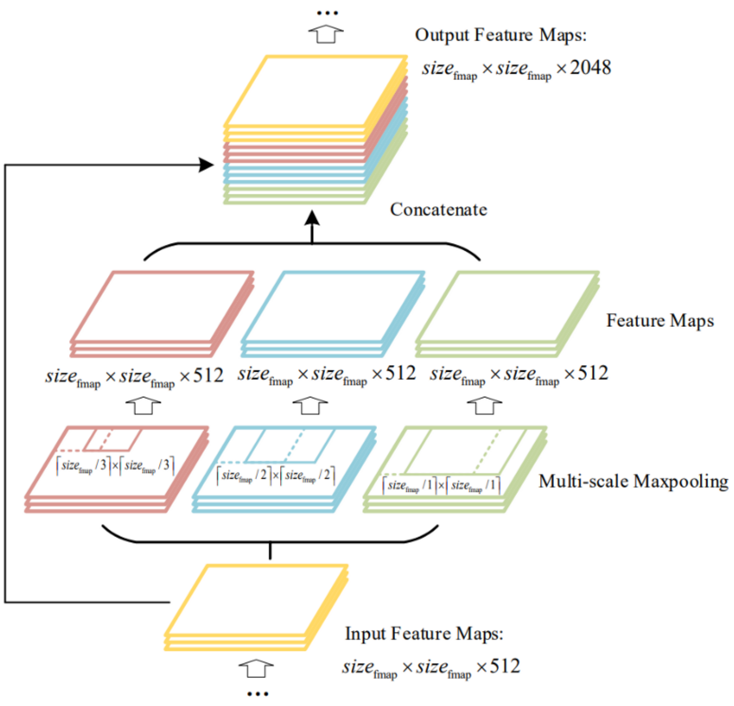


Fig. 25: The SPP block in MyModel [13]

A maximum pool is applied to a sliding kernel of size 5×5, 9×9, 13×13. The spatial dimension is preserved. The feature maps of different kernel sizes are then concatenated as output.

The head

Finally comes to the last layer to fully understand the whole structure This layer is called the head, it is responsible for making predictions. This part is unchanged from the official version of YOLO-v3.

The goal of the head's part is to output the bounding box coordinates (x,y,w,h) and the confidence score for each class. This is done on 3 different scales, so the best fitting bounding box could be chosen.

Model detection

We have explained all the important parts of a model architecture, now we can combine them and explain how object detection is done.

The input image is in shape Bx3x416x416. This goes through a few convolution layers and to the backbone layer. As was mentioned above, the backbone acts as a feature extractor and outputs feature images of 3 different shapes. The final shape is dependent on the dataset. Six numbers represent each bounding box (pc, bx, by, bh, bw, c). Because MyModel is trained on the PASCAL VOC dataset, which contains 20 classes, the output is a list of bounding boxes along with the 20 recognized classes. Totally we have (I x (5 + c)) entries in the feature map shape. Where I is a number of scales, 5 represent class probability score, 2 coordinates, width, height, and c 20 classes. This makes it 3 x (5 + 20) = 75 entries on the feature map.

The backbone's outputs are in the shape [B, 75, 13, 13], [B, 75, 26, 26], [B, 75, 52, 52]. Where B is the number of batches on which it will be trained. 75 feature layers, which will be converted to the required shape, and the last two numbers represent the number of cells, in which the image will be divided.

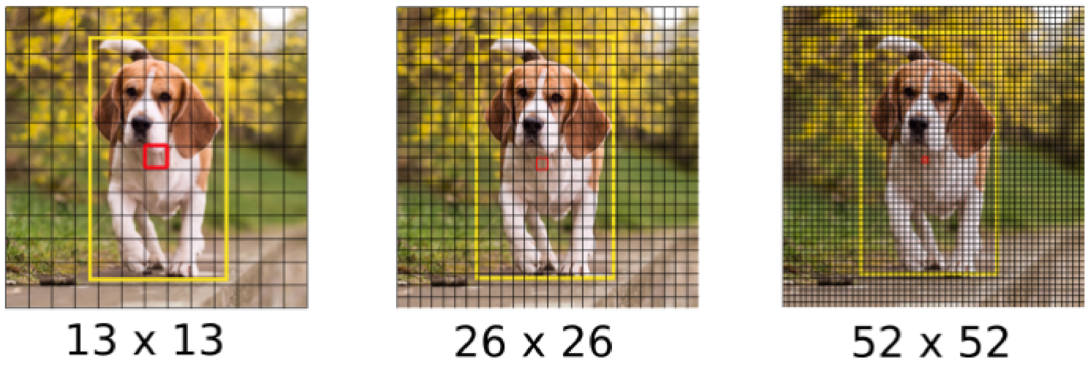


Fig. 26: Three different scale predictions [33]

These features are combined through the neck part to get the best possible result and sent to the head part. The output of head layer is in shape [B, 52, 52, 3, 25], [B, 26, 26, 3, 25], [B, 13, 13, 3, 25]. This is because the shape of 75 features was divided for each scale, so every cell in the image can predict objects of 3 different sizes. These features produced by the convolutional layers are passed onto a classifier, which makes the detection prediction, so each cell can predict a fixed number of bounding boxes.

Each cell predicts 3 bounding boxes using 3 given anchors for each scale, making the total number of anchors used 9.

The concept of anchor boxes: Trying to predict the width and height of the bounding box right from the neural net could be done, but that usually leads to unstable gradients during training [24]. Here the concept of anchor boxes, which was introduced in faster R-CNN, is used. So instead of predicting width and height, most modern object detectors predict offsets to pre-defined default bounding boxes called anchors. Then a transformation of these anchor boxes is provided to make the best fit to the ground truth box (the label).

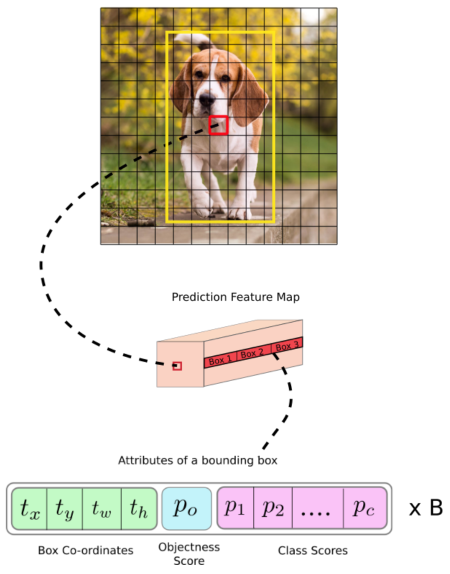


Fig. 27: The Model prediction shape [33]

Anchor boxes are k-Means centroids [44] with IoU as the similarity metric . They are obtained by using clustering algorithm onto the dataset, because we are using very popular dataset for object detection, there is no need to do the same exact job. The anchor boxes were taken from the official source.

The following formula describes how to obtain predictions of the bounding box from the coordiantes:

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w * e^{t_w} \\ b_h &= p_h * e^{t_h} \end{aligned}$$

Tx, ty, tw, th is what the network outputs. We are running our center coordinates prediction through a sigmoid function, so the values are rescaled into the interval $[0, 1]$ (prediction must happend within the cell, otherwise neighbour cell would make prediction). The values $\mathrm{c_x}$ and $\mathrm{c_y}$ are the top-left coordinates of the grid. $\mathrm{P_w}$ and $\mathrm{p_h}$ are anchors dimensions of the box. $\mathrm{B_x}$, $\mathrm{b_y}$, $\mathrm{b_w}$, $\mathrm{b_h}$ are the x, y center coordinates, width, and height.

The object score and class confidences: The object score represents the probability that an object is contained inside a bounding box. The object score is then passed through a sigmoid function and the output is interpreted as probability. The class confidence score should define a score for every class in the dataset, so the best class for the detected object can be chosen.

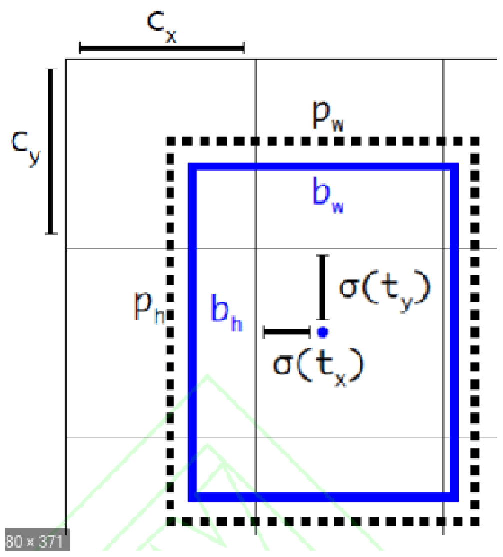


Fig. 28: The anchor boxes [33]

Loss function

$$
\begin{aligned}
&YOLOv3 - LOSS = \\
&\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
&+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 \\
&+ \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
&+ \sum_{i=0}^{S^2} 1_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2
\end{aligned} \tag{5}
$$

The YOLOv3 loss function was used, it consists of 5 expressions:

1) The x and y coordinates are parametrized to be offsets of a particular grid cell location. The sum of square error is estimated only when there is an object inside the cell.
2) The width and height of the bounding box are normalized by the width and the height of the image so that they fall between the values 0 and 1. The sum of square error is again estimated only when there is an object. Square roots are used because deviations in large boxes matter less than in small boxes.
3) Confidence predictions for boxes that contain objects
4) Confidence predictions for boxes that do not contain objects. Most grid cells do not contain any object, which pushes confidence scores towards zero, over-

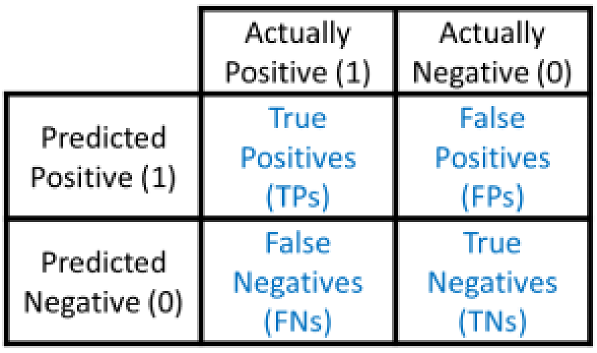
powering confidence of grids, which contain objects. To make training more stable, confidence predictions for boxes that do not contain objects are decreased by $\lambda_{noobj} = 0,5$.

5) Class Probabilities when there is an object

Evaluation

We want to use a metric during training, which would tell the model how well the boundary box fits the label (ground truth box). There is one popular technique used to measure how precise the bounding box is fitting, and that metric is called mean Average Precision (mAP). The mean of average precision values is calculated over recall values (see 3.1) from 0 to 1. It is based on several basic principles, which will be firstly introduced.

Confusion Matrix:



	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Fig. 29: The confusion matrix [8]

In statistical classification, a confusion matrix is a specific table that allows visualization of the performance of an algorithm.

1) True Positives (TP): The model predicted a label and matches correctly according to the ground truth.
2) True Negatives (TN): The model does not predict the label and is actually not part of the ground truth.
3) False Positives (FP): The model predicted a label, but it is not part of the ground truth (Type I error).
4) False Negatives (FN): The model does not predict a label, while it is a part of the ground truth. (Type II error).

Precision and recall:

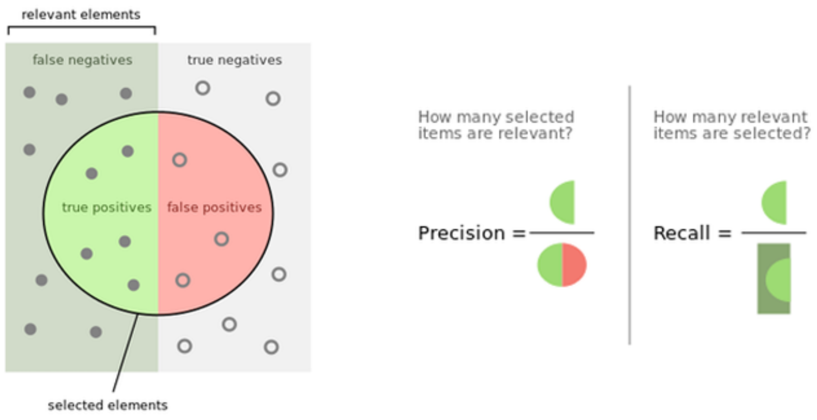


Fig. 30: Precision recall evaluation [35]

Precision measures how well you can find true positives (TP) out of all positive predictions (TP + FP). Recall measures how well you can find true positives (TP) out of all predictions (TP + FN).

The value may vary depending on the model's IoU (Intersection over Union) confidence threshold.

Average Precision: AP is calculated as the weighted mean of precision at each threshold, the weight is the increase in recall from the prior threshold.

Mean Average Precision: mAP is the average AP of each class. In the PASCAL VOC 2007 challenge, AP for one class of objects was calculated for an IoU threshold of 0.5. So the mAP was averaged over all object classes.

$$mAP = 1/N(\sum_{i=1}^{N} AP_i) \tag{6}$$

The total output of the model for our 416 x 416 image is ((52 x 52) + (26 x 26) + 13 x 13)) x 3 = 10647 bounding boxes. This is quite a large number to do for every frame and still have real-time processing power. This is where non-max supression (NMS) pre-processing comes in.

NMS processing: As said above, every bounding box is a vector of 25 values for our dataset (20 classes). Bounding boxes with low objectness score can be left out because the box is not very confident about detecting a class. This helps us filter out most of the boxes. After this filtering, there are a few boxes left, which meet the objectness score threshold condition. Now comes the second filter, called NMS, which is based on IoU to select the best bounding box.

IoU: Intersection over union is a metric that tells how precise the bounding box is by measuring the area of overlap of the boundary box and the ground truth box. We can say an IoU > 0.5 is decent, > 0.7 pretty good and > 0.9 almost perfect.

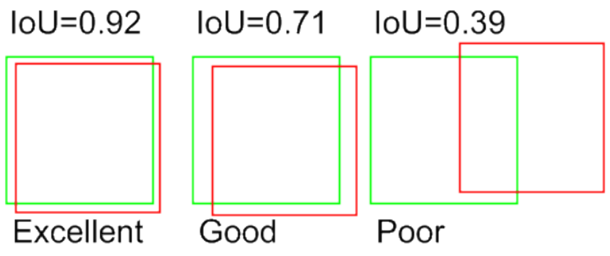


Fig. 31: Intersection over Union [5]

Now, to implement non-max suppression, the key steps are:

1) Select the box that has the highest score;
2) Compute its overlap with all other boxes, and remove boxes that overlap it more than IoU threshold x
3) Go back to step 1 and iterate until there are no more boxes with a lower score than the selected box.

Finally, the whole process of obtaining the best bounding box by model is done. Now it can be displayed on an image.

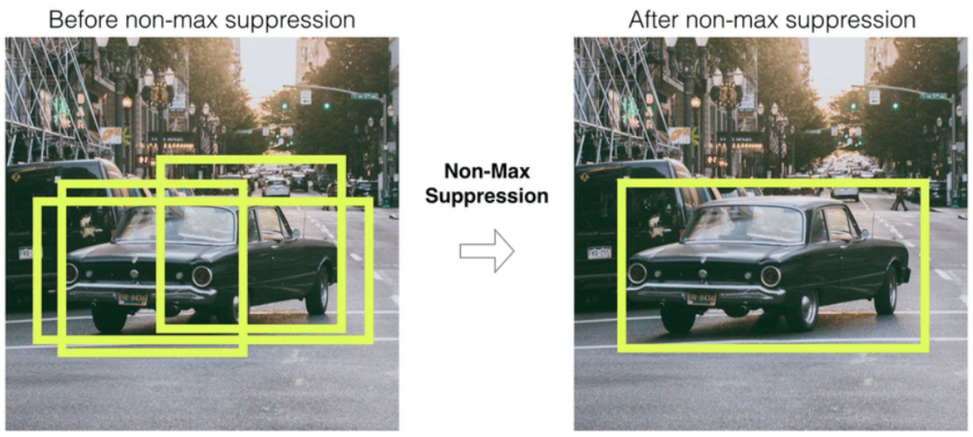


Fig. 32: Non Maximal Suppression NMS [17]

Results

The MyModel neural network is a larger neural network consisting of 70 million learnable parameters. This neural network was trained from scratch on PASCAL VOC dataset. This dataset was chosen, as it often occurs in object detection algorithms, so our model could be easily compared with other models. Deep learning models are trained on GPUs, as it offers much faster learning time. But the problem with CNN models is that it usually requires a large amount of GPU RAM, as we are working with quite large images. The model was trained in Google Colab on a P100 GPU, which offers 16GB RAM. The training time for the PASCAL VOC dataset was 36 hours on 16,000 images for the training dataset. MAP was measured on a test dataset consisting of 4000 images. The initial learning rate was 3^{-4} and after 10 epochs it was reduced to 3^{-5}.

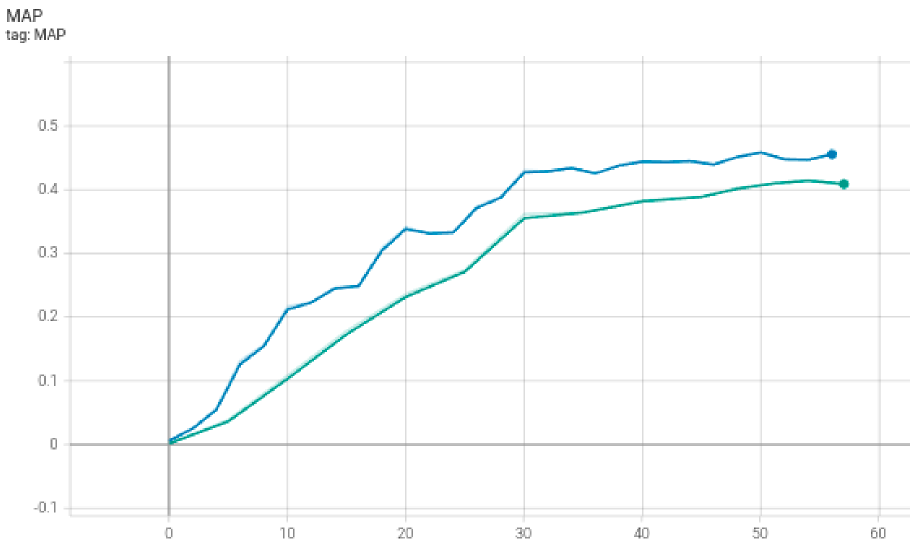


Fig. 33: MAP

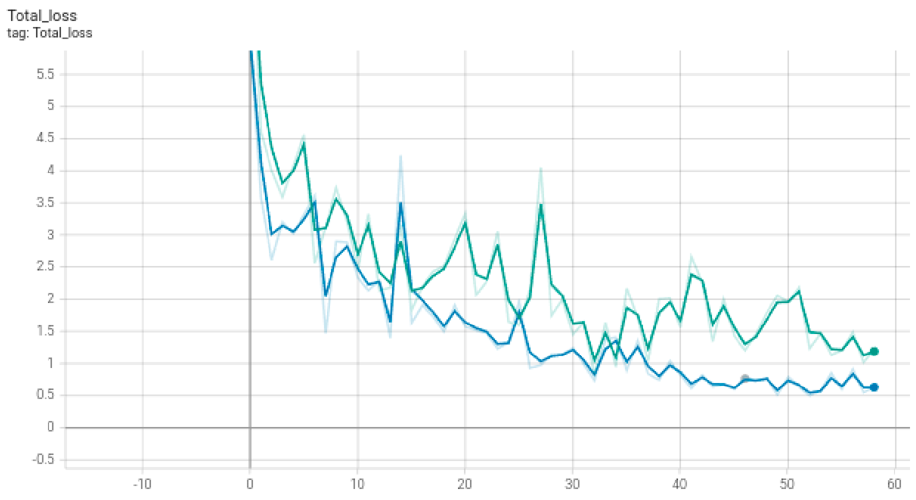


Fig. 34: Total loss

Blue lines represent MyModel net and green official implementation of YOLOv3. Both models were further trained, but because none of them showed better improvements, only this part is included. For MyModel highest reached mAP was 0.445 and for official implementation almost 0.42.

4 DISTANCE ESTIMATION

For the object detection system, it is useful to add distance estimation for some objects as it can provide additional information. For systems which are not fully autonomous (all of the current ones), this could help the driver to have more awareness of the surroundings. And if the object were too close, the driving systems could potentially warn the driver, which would provide more safety for the entire crew.

The KITTI dataset provides annotated dataset, which was produced by a single camera mounted on the front of a car. This dataset consists of training images, test images, and annotations in the form of a CSV file: filename, xmin, ymin, xmax, ymax, zloc, and some additional informations, which are not needed. Here, xmin, ymin, xmax, and ymax are bounding box coordinates, zloc is the distance.

Because the model for estimating the bounding boxes was already built. This dataset could be used for another model, to make distance estimations on the last model, which provides bounding box estimation.

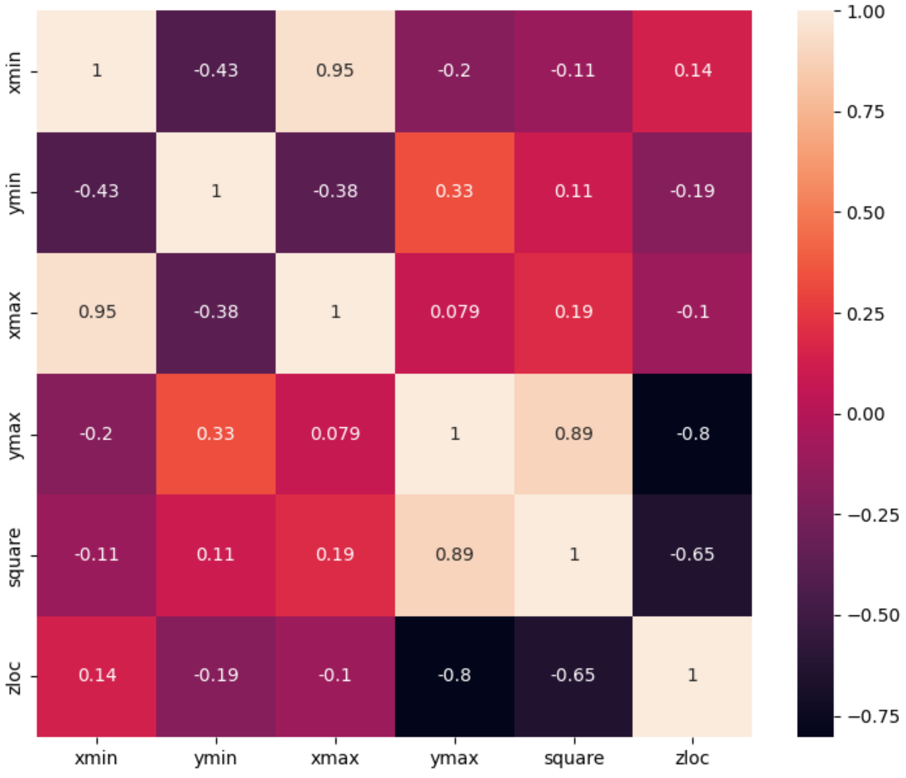


Fig. 35: Feed forward network

There was suspected some relationship between distance and square area of the bounding box. So, a new column called square was created for enhancing more precise results. This is shown in the correlation matrix Fig: 35, where the correlation between square area and distance is -0.65.

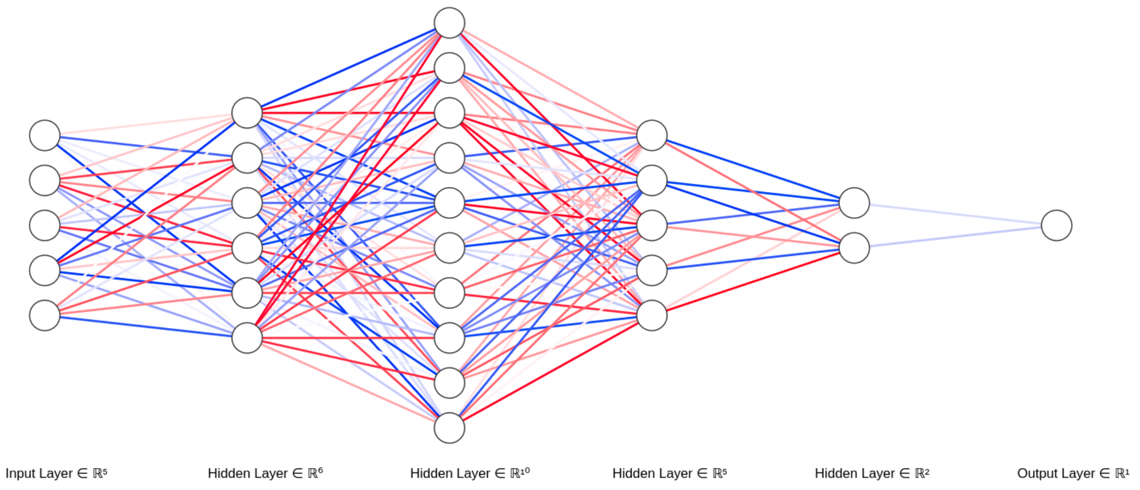


Fig. 36: Feed forward network

A model with inputs xmin, ymin, xmax, ymax, and square was trained on 25 822 labels (10% validation data), the test set contained 2 922 labels. The best model was found to have 4 hidden layers in the form of 6-10-5-2 hidden neurons and 1 output that predicted the distance. For this regression task, a mean squared error function had been chosen.

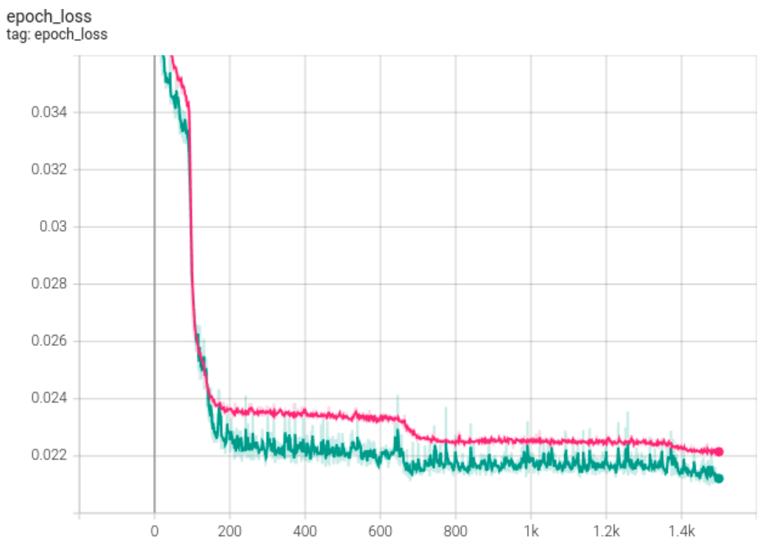


Fig. 37: Mean squared error

The red line in Fig: 37 maps the error function for training data and the green line for the validation data. The model was tested on test data afterward, where MSE = 0,21.

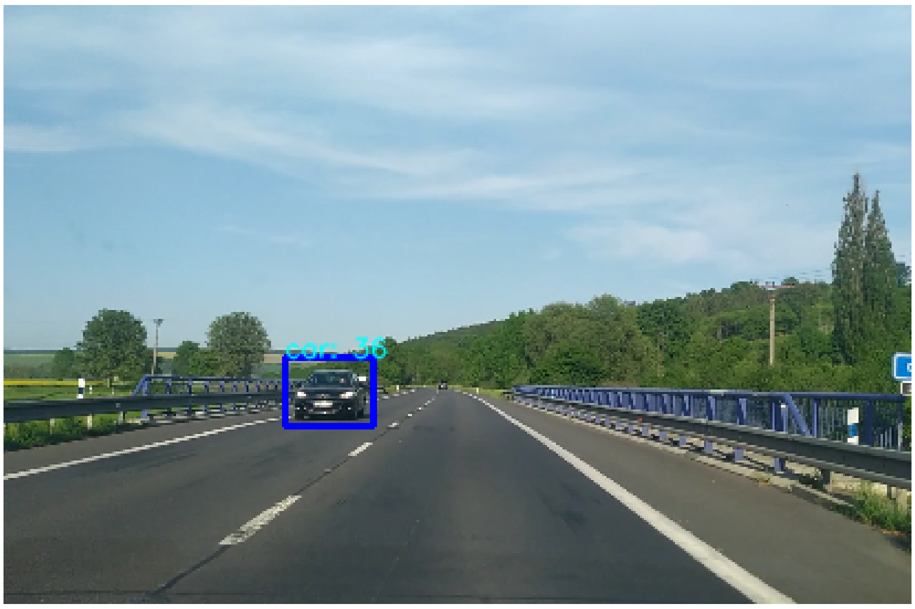


Fig. 38: Distance estimation of cars

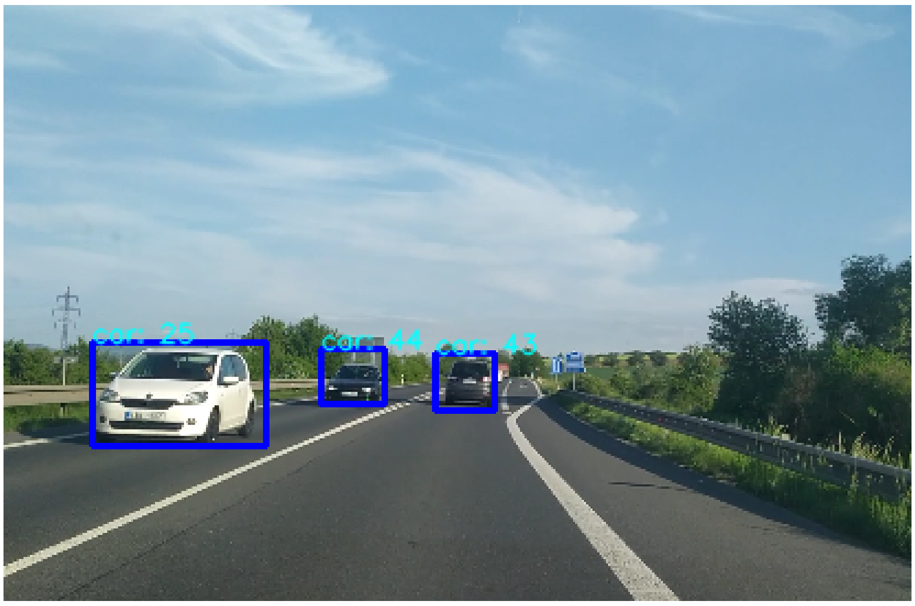


Fig. 39: Distance estimation of cars

5 LANE DETECTION

There are many ways in which a lane detection system can be built. Many of those are based on segmentation tasks with deep neural networks like Unet, which provides one of the best results. As most of our computer resources and GPU RAM were used for object detection and distance estimation, because it is the main part of our self-driving system, the lane detection system was built based on the OpenCV library. Opencv enables fast image post-processing with very little GPU usage. The lane detection system consists of several steps which are applied to the image taken from the display.

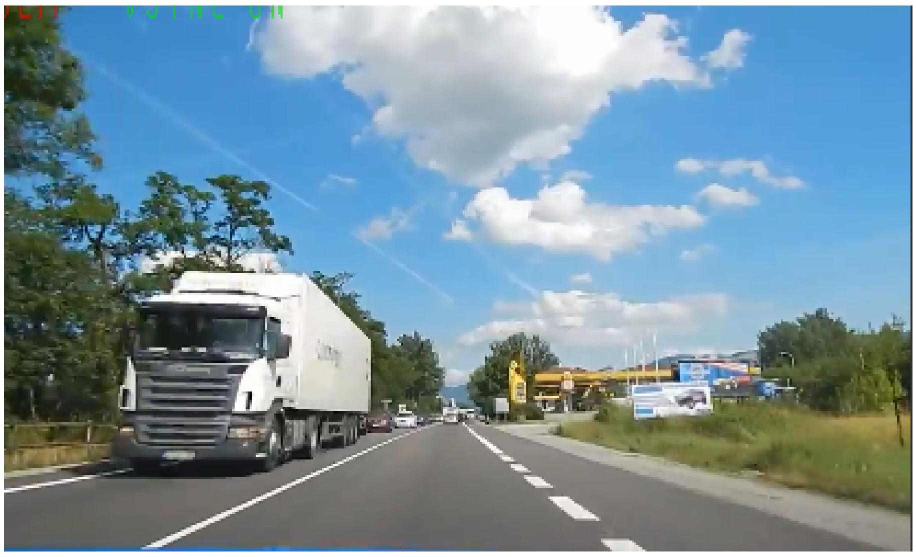


Fig. 40: The input image

The original image was converted to grayscale. This reduces the dimension of the image from 3D to 1D, which should speed up the process. Together with grayscale conversion, Gaussian blur was applied. Gaussian blurring is highly effective in removing Gaussian noise from an image.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{7}$$

The values of this distribution are used to build a convolution matrix that is applied to the original image.

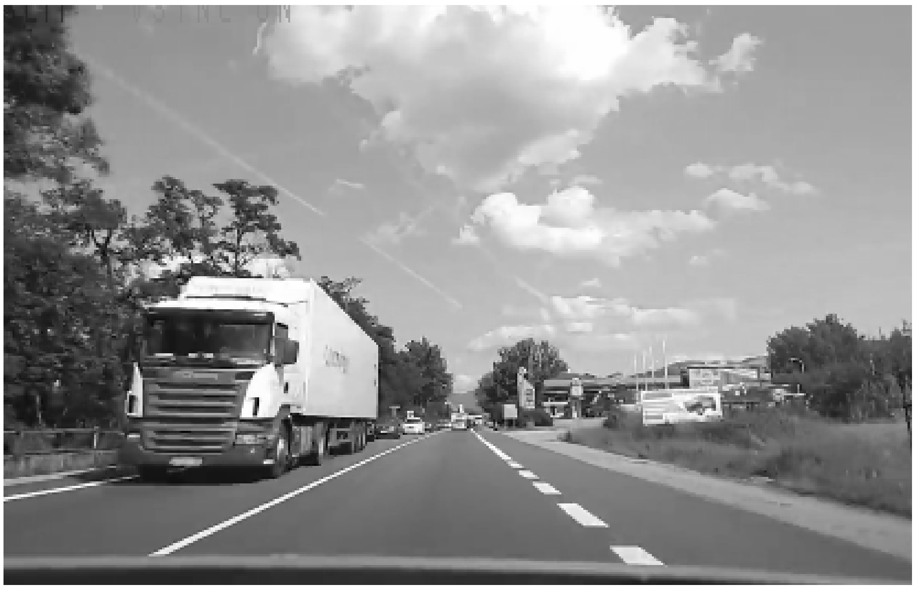


Fig. 41: Grayscale and Gaussian blur

Canny edge detection is applied. It consists of several phases:

1) Noise reduction - A 5x5 Gaussian filter is applied because edge detection is susceptible to noise
2) Intensity gradient - A Sobel kernel is applied to get the first derivative in the horizontal direction $\mathrm{G_x}$ and in the vertical direction $\mathrm{G_y}$. Then we can get an edge gradient and direction.

$$EdgeGradient(G) = \sqrt{G_x^2 + G_y^2}$$
$$Angle(\sigma) = \tan^{-1} \frac{G_y}{G_x}$$

   The gradient direction is always perpendicular to the edges.
3) Non-maximum suppression - A full scan of the image is done to remove any unwanted pixels. Every pixel is checked to see if it is a local maximum in its neighborhood in the direction of the gradient. The result is a binary image with "thin edges".
4) Hysteresis Thresholding - This stage decides which set of pixels are edges and which are not. Edges with an intensity gradient more than maxVal are marked as edges and those below minVal are marked as non-edges. Those who lie between these two thresholds are classified as edges or non-edges based on their connectivity. If they are connected to pixels marked as edges, they are classified as edges; otherwise, they are classified as non-edges. The result is seen in image Fig: 42

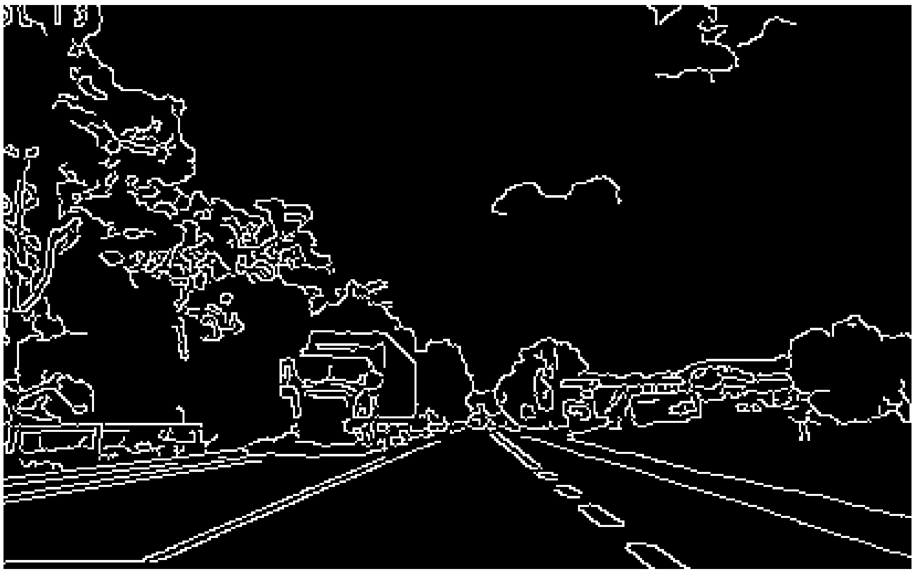


Fig. 42: Canny edge detector

The region of interest is selected, such that, most of the unwanted edges are left out.

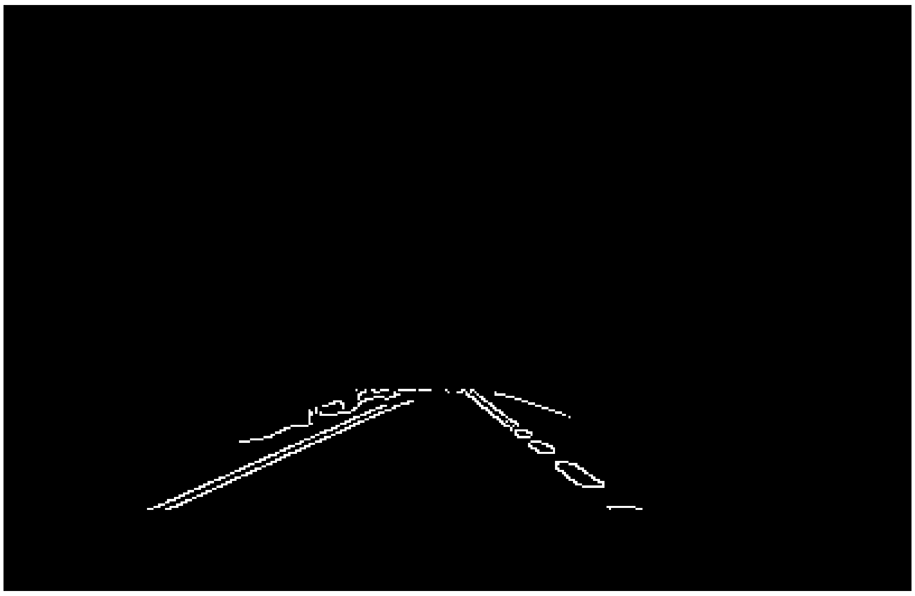


Fig. 43: Image masked

Later, when edges are obtained, they are used in HoughLinesP algorithm. The Hough transform takes a binary edge mAP as input and locates edges's place on straight lines [4]. The HoughLinesP algorithm is a more efficient implementation of the previous algorithm using a probabilistic approach. The output of HoughLinesP is the endpoints of the detected lines (x_0, y_0, x_1, y_1).

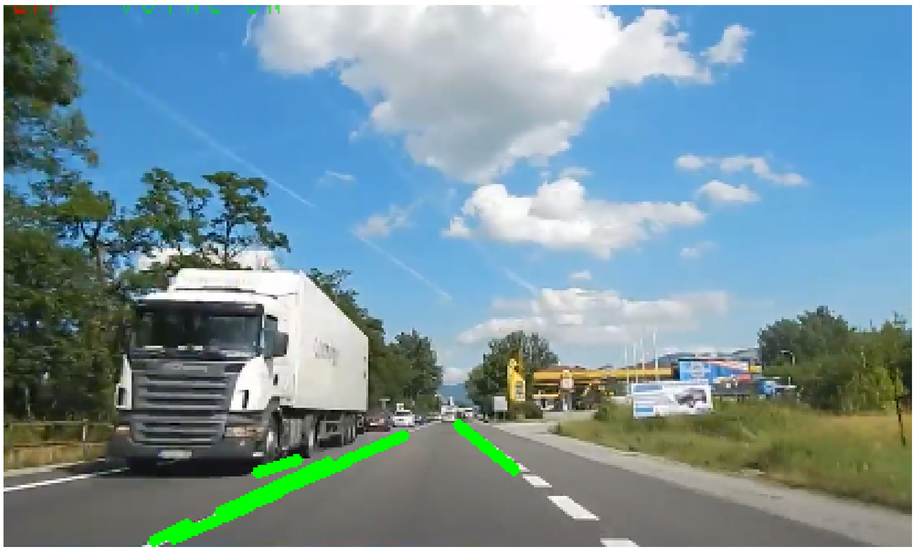


Fig. 44: Straight lanes found by the HoughLinesP algorithm

Detected straight lines are then grouped together by having negative slope or positive slope. Now the averaged position and slope of each group are found to represent a road lane.

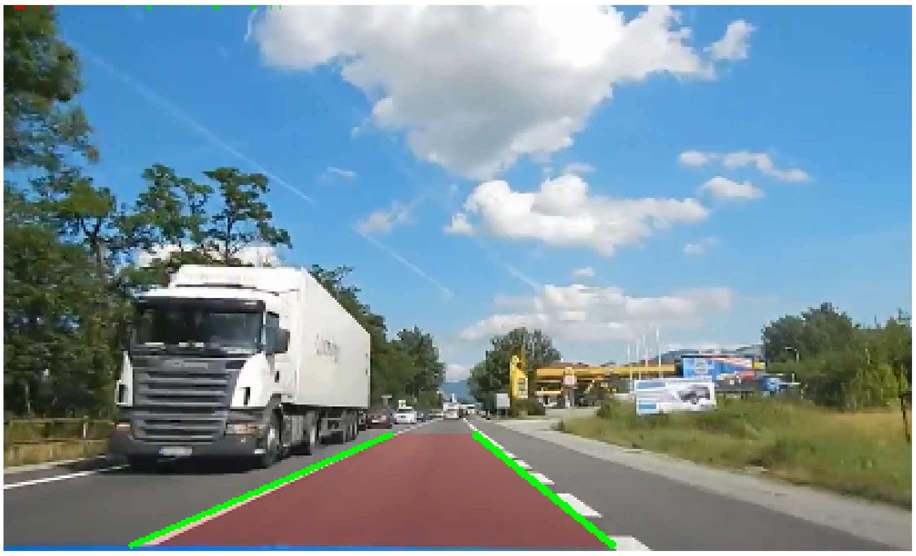


Fig. 45: The final lines are displayed.

This was the last part of the lane detection system to make the task possible. The final detection is shown in Fig.45, where green lines represent the boundary of the self-driving car path.

6 PERCEPTION SYSTEM

A mobile phone's camera was mounted on the rear-view mirror parallel to the road to capture the scenery in front of the car. The camera must be high enough not to see any parts of the car, as it could distort the predictions (we are trying to be close to the training dataset as much as possible). The image is sent to a computing device (in my case a laptop), where the captured image from a mobile phone is displayed on the monitor. Then OpenCV library reads a display window of size 416 x 416 in the left upper corner and every frame is processed by the perception system built in this thesis.

MyModel was trained on the PASCAL VOC dataset because it is a widespread dataset for object detection, which works as a good benchmark for other models. So they can be easily compared to each other. This is why it was a good choice for the evaluation part, but not all classes in the dataset are needed for the perceptive system of self-driving cars. Most of the classes for object detection were disabled and only object detection for bicycles, buses, cars, and motorbikes were left. The distance estimation dataset only included classes for cars, so it would not make sense to do it for other bounding boxes and only this one class is estimated.

The lane detection system is computationally very light and offers high fps, which was the main reason why this approach was taken, as most of the GPU resources were taken by a custom object detection model. But as this approach is based on straight lines, making very sharp turns make this detection imprecise.

Images taken during the drive:

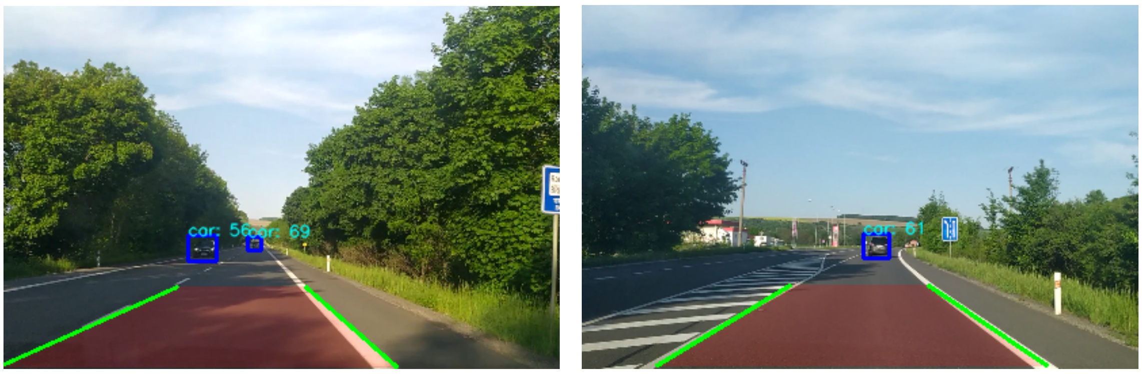


Fig. 46: Images during good conditions

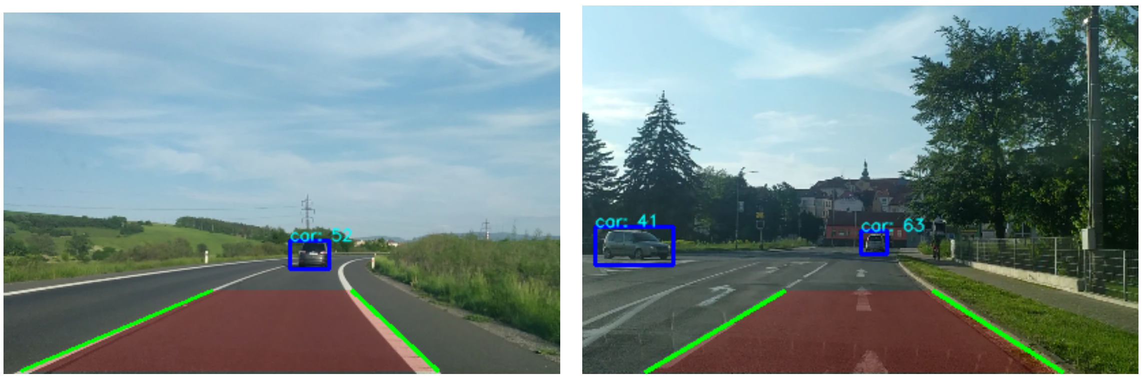


Fig. 47: Images during good conditions

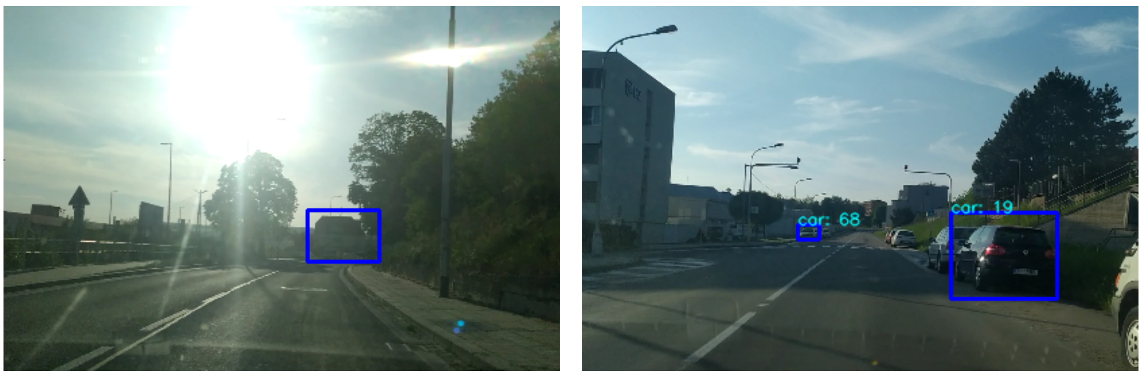


Fig. 48: Images during worse or more difficult conditions

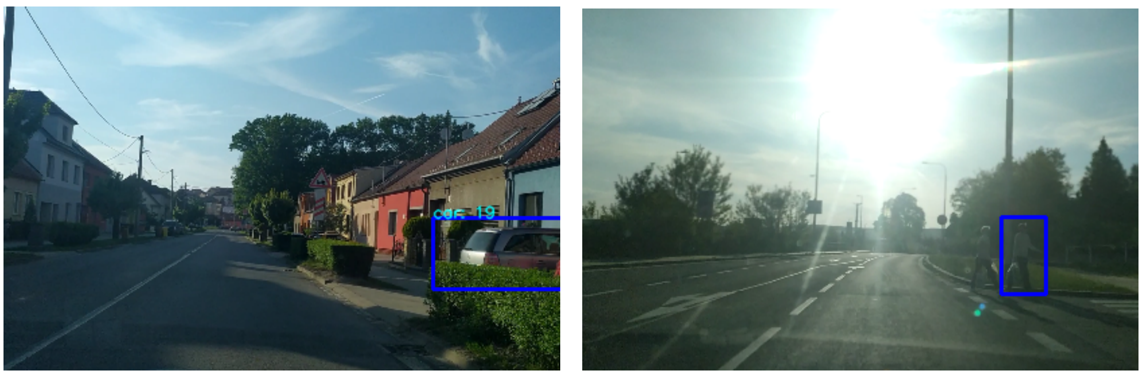


Fig. 49: Images during worse or more difficult conditions

7 CONCLUSION

An introduction to autonomous systems was described in the first part of this work. This summary includes the history and development of self-driving cars, the basic classification of how advanced these self-driving cars can be, some techniques used for learning such a system, and the last part is focused on object detection.

Object detection was the primary focus of this work, where the most well-known algorithms were described. During the object detection era a variety of models were built using many different techniques. But, as it turns out, the best models right now are based on deep learning, which is a special field of machine learning using neural networks.

In the implementation part, a basic autonomous system was built, which could be used in a self-driving car. This system is based on the detection of objects that can be found on the road. A totally new deep learning model was built, which was based on YOLO-v3. However, using more modern approaches, the model was properly evaluated and used in such a system for object detection.

The problem came up during the training part, as both of the models stopped converging around the 60th epoch. This could be due to several reasons. The first is, that the implementation could be wrong, but this seems unlikely, as, before the final training part, both of the models were over-fitted on 500 images, where both had over 90 mAP. Another reason could be the batch size, where 8 could be too small as most of these networks are usually learned on 64 or more, when learning from scratch. But this problem would require more than 16 GB of RAM, which was not possible for me.

To have a more complete version of that autonomous system, a distance estimation was built. Where prediction is done by a feed-forward neural network pre-trained on KITTI dataset containing distances for every bounding box.

The last part of the system was the detection of the road lane. There had to be several compromises, as there was not enough computing power for a more robust solution. This is why most of this part is done in OpenCV library.

The whole model performs quite well during certain conditions as it is shown in the last part. Ideally, the system could be used on nearly straight roads with well-defined lanes. However, the model had a much higher number of false positive boxes (FPs) when the images included bad weather conditions, poor illumination, or a lot of shadows. Therefore, I can say that more research is needed in that direction.

8 BIBLIOGRAPHY

[1] Adaloglou, N.: *In-layer normalization techniques for training very deep neural networks*. Technická zpráva, theaisummer, 2020.

[2] Ananth, S.: *Faster R-CNN for object detection*. 2019.
URL `https://towardsdatascience.com/faster-r-cnn-for-object-detection-a-techn`

[3] Bojarski, M.; Del Testa, D.; Dworakowski, D.; aj.: *End to end learning for self-driving cars*. *arXiv preprint arXiv:1604.07316*, 2016.

[4] Coste, A.: *Image Processing : Hough Transform*. 2012, doi: 10.13140/RG.2.2.30430.48969.
URL `https://www.researchgate.net/publication/330502710_Image_Processing_Hough_Transform`

[5] Cowton, J.; Kyriazakis, I.; Bacardit, J.: *Automated Individual Pig Localisation, Tracking And Behaviour Metric Extraction Using Deep Learning*. *IEEE Access*, ročník PP, 08 2019, doi:10.1109/ACCESS.2019.2933060.

[6] D'ALLEGRO, J.: *How Google's Self-Driving Car Will Change Everything*. Technická zpráva, Investopedia, 2022.

[7] Digitaltrends: *The history of self-driving cars*. Technická zpráva, Digital Trends Media Group, 2020.

[8] Draelos, R.: *Measuring Performance: The Confusion Matrix*. Technická zpráva, glassboxmedicine, 2020.

[9] Gad, A. F.: *Faster R-CNN Explained for Object Detection Tasks*. 2020.
URL `https://blog.paperspace.com/faster-r-cnn-explained-object-detection/`

[10] Gibson, P.; Cano, J.; Turner, J.; aj.: *Optimizing Grouped Convolutions on Edge Devices*. 2020, doi:10.48550/ARXIV.2006.09791.
URL `https://arxiv.org/abs/2006.09791`

[11] Girshick, R.; Donahue, J.; Darrell, T.; aj.: *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2013, doi:10.48550/ARXIV.1311.2524.
URL `https://arxiv.org/abs/1311.2524`

[12] Guan, L.; Chen, Y.; Wang, G.; aj.: *Real-Time Vehicle Detection Framework Based on the Fusion of LiDAR and Camera*. *Electronics*, ročník 9, 03 2020: str. 451, doi:10.3390/electronics9030451.

[13] He, K.; Zhang, X.; Ren, S.; aj.: *Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition*. In *Computer Vision – ECCV 2014*, Springer International Publishing, 2014, s. 346–361, doi:10.1007/978-3-319-10578-9_23.
URL `https://doi.org/10.1007%2F978-3-319-10578-9_23`

[14] Herrtwich, R.: *MERCEDES - AUTONOMOUS DRIVING - THE S500 INTELLIGENT DRIVE*. Technická zpráva, DAIMLER, 2014.

[15] Hu, X.; Yang, W.; Wen, H.; aj.: *A Lightweight 1-D Convolution Augmented Transformer with Metric Learning for Hyperspectral Image Classification*. *Sensors*, ročník 21, 03 2021: str. 1751, doi:10.3390/s21051751.

[16] ITBodhi: *Bias and Variance Trade off*. 2020.
URL `https://medium.com/@itbodhi/bias-and-variance-trade-off-542b57ac7ff4`

[17] Jain, H.; Nandy, S.: *Incremental Training for Image Classification of Unseen Objects*. 08 2019, doi:10.13140/RG.2.2.10266.47046.

[18] Janai, J.; Güney, F.; Behl, A.; aj.: *Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art*. Max-Planck-Institute for Intelligent Systems Tübingen and University of Tübingen, Germany.

[19] Jaritz, M.: *2D-3D scene understanding for autonomous driving*. Technická zpráva, 2020.

[20] Krizhevsky, A.; Sutskever, I.; Hinton, G. E.: *ImageNet Classification with Deep Convolutional Neural Networks*. In *Advances in Neural Information Processing Systems*, ročník 25, editace F. Pereira; C. Burges; L. Bottou; K. Weinberger, Curran Associates, Inc., 2012.
URL `https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`

[21] Lee, S.: *Understanding Face Detection with the Viola-Jones Object Detection Framework*. 2020.
URL `https://towardsdatascience.com/understanding-face-detection-with-the-viola-jo`

[22] Liu, W.; Anguelov, D.; Erhan, D.; aj.: *SSD: Single Shot MultiBox Detector*. In *Computer Vision – ECCV 2016*, Springer International Publishing, 2016, s. 21–37, doi:10.1007/978-3-319-46448-0_2.
URL `https://doi.org/10.1007%2F978-3-319-46448-0_2`

[23] Liu, Z.; Mao, H.; Wu, C.-Y.; aj.: *A ConvNet for the 2020s*. 2022, doi:10.48550/ARXIV.2201.03545.
URL `https://arxiv.org/abs/2201.03545`

[24] Mittal, A.: .

[25] Mittal, A.: *Haar Cascades, Explained, year = 2020, url = https://medium.com/analytics-vidhya/haar-cascades-explained-38210e57970d*.

[26] Mukherjee, S.: *Top ten challenges in object detection every data scientist should know*. Technická zpráva, analyticsindiamag, 2022.

[27] Nwankpa, C.; Ijomah, W.; Gachagan, A.; aj.: *Activation Functions: Comparison of trends in Practice and Research for Deep Learning*. 2018, doi: 10.48550/ARXIV.1811.03378.
URL `https://arxiv.org/abs/1811.03378`

[28] Ondruš, J.; Kolla, E.; Vertaľ, P.; aj.: *How Do Autonomous Cars Work? Transportation Research Procedia*, ročník 44, 2020: s. 226–233, ISSN 2352-1465, doi:https://doi.org/10.1016/j.trpro.2020.02.049, lOGI 2019 - Horizons of Autonomous Mobility in Europe.
URL `https://www.sciencedirect.com/science/article/pii/S2352146520300995`

[29] pawangfg: *Selective Search for Object Detection | R-CNN*. 2021.
URL `https://www.geeksforgeeks.org/selective-search-for-object-detection-r-cn`

[30] Peterisfar: *YOLOv3*. 2020.
URL `https://github.com/Peterisfar/YOLOV3/blob/master/config/yolov3_config_voc.py`

[31] Pomerleau, D. A.: *Neural Network Vision for Robot Driving*. Dizertační práce, Carnegie Mellon University.

[32] Redmon, J.; Divvala, S.; Girshick, R.; aj.: *You Only Look Once: Unified, Real-Time Object Detection*. 2015, doi:10.48550/ARXIV.1506.02640.
URL `https://arxiv.org/abs/1506.02640`

[33] Redmon, J.; Farhadi, A.: *YOLOv3: An Incremental Improvement*. 2018, doi: 10.48550/ARXIV.1804.02767.
URL `https://arxiv.org/abs/1804.02767`

[34] Ren, S.; He, K.; Girshick, R.; aj.: *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2015, doi:10.48550/ARXIV.1506.01497.
URL `https://arxiv.org/abs/1506.01497`

[35] Riggio, C.: *What's the deal with Accuracy, Precision, Recall and F1?* 2019.
URL `https://towardsdatascience.com/whats-the-deal-with-accuracy-precision-re`

[36] Ronneberger, O.: *U-Net: Convolutional Networks for Biomedical Image Segmentation.* Technická zpráva, Uni Freiburg, 2016.

[37] Siam, M.; Elkerdawy, S.; Jagersand, M.; aj.: *Deep Semantic Segmentation for Automated Driving: Taxonomy, Roadmap and Challenges.* 2017, doi:10.48550/ARXIV.1707.02432.
URL `https://arxiv.org/abs/1707.02432`

[38] Stamenković, D.; Blagojevic, I.; Popovic, V. M.: *A Brief Review of Strategies Used to Control an Autonomous Vehicle. ResearchGate*, 2017.

[39] Team, T. T.: *Dual Motor Model S and Autopilot.* Technická zpráva, Tesla, 2014.

[40] Tesla Autopilot: *Tesla Autopilot — Wikipedia, The Free Encyclopedia.* 2010.
URL `https://en.wikipedia.org/wiki/Tesla_Autopilot`

[41] Uijlings, J. R.; Van De Sande, K. E.; Gevers, T.; aj.: *Selective search for object recognition. International journal of computer vision*, ročník 104, č. 2, 2013: s. 154–171.

[42] Wang, L.: *Support vector machines: theory and applications*, ročník 177. Springer Science & Business Media, 2005.

[43] Zhang, C.; Zhang, Z.: *A Survey of Recent Advances in Face Detection.* 01 2010, 1-17 s.

[44] Zhong, Y.; Wang, J.; Peng, J.; aj.: *Anchor Box Optimization for Object Detection.* 2018, doi:10.48550/ARXIV.1812.00469.
URL `https://arxiv.org/abs/1812.00469`

[45] Zou, Z.; Shi, Z.; Guo, Y.; aj.: *Object Detection in 20 Years: A Survey.* 2019, doi:10.48550/ARXIV.1905.05055.
URL `https://arxiv.org/abs/1905.05055`

9 APPENDIX A

Source code on: https://github.com/LecbychMichal/Master-s-Thesis