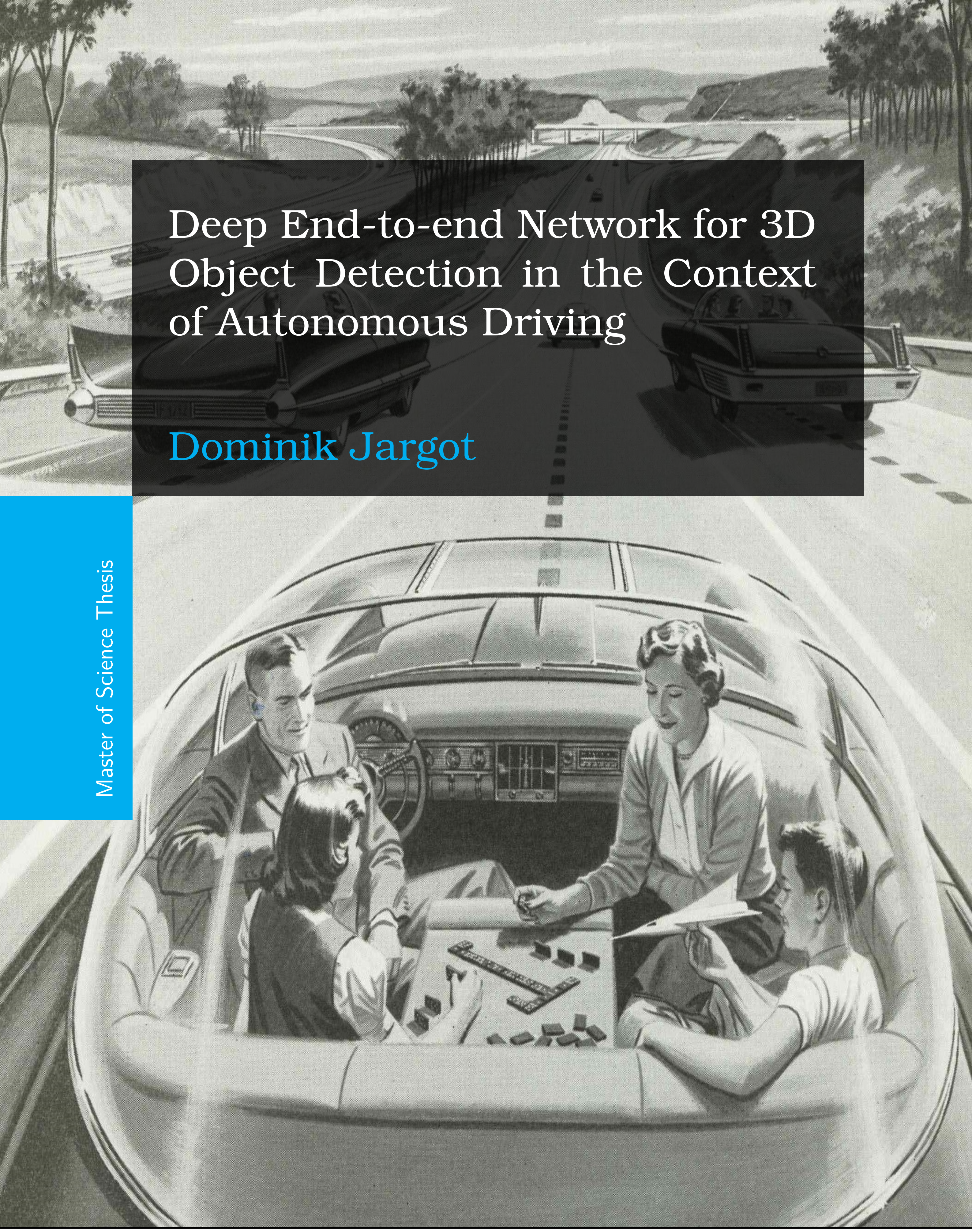


Deep End-to-end Network for 3D Object Detection in the Context of Autonomous Driving

Dominik Jargot

Master of Science Thesis

Delft University of Technology

Deep End-to-end Network for 3D Object Detection in the Context of Autonomous Driving

Master of Science Thesis

For the degree of Master of Science in Vehicle Engineering at Delft University of Technology

Dominik Jargot

February 7, 2019

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis was completed in the Research and Development division of Daimler AG. Their cooperation is hereby gratefully acknowledged.

Delft University of Technology
Department of
Cognitive Robotics (CoR)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Deep End-to-end Network for 3D Object Detection in the Context of Autonomous Driving

by

Dominik Jargot

in partial fulfillment of the requirements for the degree of

Master of Science Vehicle Engineering

Dated: February 7, 2019

Supervisor(s):

Prof.dr. D.M. Gavrila

Dipl.-Inform. M.R. Roth

Reader(s):

Dr. J.F.P. Kooij

Dr.ing. J. Kober

Dr. M. Kok

Abstract

Nowadays, autonomous driving is a trending topic in the automotive field. One of the most crucial challenges of autonomous driving research is environment perception.

Currently, many techniques achieve satisfactory performance in 2D object detection using camera images. Nevertheless, such 2D object detection might be not sufficient for autonomous driving applications as the vehicle is operating in a 3D world where all the dimensions have to be considered. In this thesis a new method for 3D object detection, using deep learning approach is presented. The proposed architecture is able to detect cars using data from images and point clouds. The proposed network does not use any hand-crafted features and is trained in an end-to-end manner.

The network is trained and evaluated with the widely used KITTI dataset [1].

The proposed method achieves an average precision of 81.38%, 67.02%, and 65.30% on the easy, moderate, and hard subsets of the KITTI validation dataset, respectively. The average inference time per scene is 0.2 seconds.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisor prof. dr. Dariu M. Gavrila for the opportunity to work in one of the best known automotive companies in the world. Additionally, I would like to express my gratitude for his patience and forbearance. I would like to also thank my daily supervisor Markus R. Roth for the support and guidance during my stay at Daimler AG, and the trust after my stay.

Great thanks to my parents, Beata and Grzegorz, who supported me no matter what. They gave me more than they could afford and I promise to make them proud. I would also like to thank my beloved girlfriend, Adrianna Kaźmierczak, for reading and reviewing my work. She has always been there for me and believed in me even when I did not believe in myself.

The time I have spent at the University in Delft and at Daimler in Ulm was not only a great academic and professional experience, but also a chance to make friends for life. I would like to thank all of the people I have met during this over two years long journey for making this time unforgettable. It was a pleasure for me to study at TU Delft and I am glad to be where I am now.

Delft, University of Technology
February 7, 2019

Dominik Jargot

To my parents.

Chapter 1

Introduction

Nowadays, autonomous driving is a subject undergoing intense study with many researchers working towards a goal of a fully automated vehicle. One of the milestones towards this full automation is the vehicle's ability to perceive the environment. Without a human in the loop the vehicle has to take over all the tasks related to road observation, including detection of cars and vulnerable road users (VRU)[1].

Object detection and classification is currently a matter of broad interest. A great share of research in this field is related to autonomous driving. For a human, object detection is natural and is processed subconsciously, whereas a machine has to be told explicitly how to perform this task on noisy sensor data via dedicated software. Additionally, the implementation in a road-legal car has to be robust as the road situation can change dynamically or adverse weather conditions can worsen the performance of sensors. Moreover, the system has to work even in case of individual sensor malfunction. Consequently, what is easy for people can be challenging for machines, hence, the reliable solution to the problem is far from trivial.

Nowadays, many modern cars are equipped with Advanced Driver-Assistance Systems (ADAS) and sensors such as cameras and radars are mounted within the chassis. For instance, the Lane Keeping Assistance utilizes the camera to prevent an unwanted lane departure. Other popular examples of ADAS are the Adaptive Cruise Control (ACC) and the Blind Spot Detection, both of them utilize radar. ACC automates the longitudinal movement of the car, i.e. it keeps a safe distance to the car in the front. The Blind Spot Detection helps the driver by providing him information whether there are vehicles in the areas not visible in the side mirrors.

These systems can be perceived as an intermediate step towards the full automation, but for now, they only offer assistance for a driver to enhance the safety. For a car to be fully autonomous, it has to be able to take over all the tasks of the human driver. One of the most important ones is the object detection, used to spot and localize other road users and obstacles.

[1]In the European Union Intelligent Transport Systems (ITS) directives vulnerable road users (VRU) are defined as: "non-motorised road users, such as pedestrians and cyclists as well as motor-cyclists and persons with disabilities or reduced mobility and orientation" [2].

In the literature, most approaches to object detection are data-driven, thus, there is a need for large and annotated datasets to teach the system how to perceive the environment. There are a lot of datasets with only 2D data, i.e. images from cameras and there are many 2D detectors that are already mature and perform well, for instance, SSD [3], YOLO [4], Fast R-CNN [5], or Faster R-CNN [6].

However, the car is in fact operating in a 3D world and we would like the autonomous vehicle to be able to also perceive the world that way. Hence the need for the datasets providing 3D data, together with 3D annotations. For now, not much labelled 3D data is available, with one of the most popular datasets being the KITTI dataset [7] providing images and lidar point clouds and 3D annotations.

Currently, the research for 3D object detection in the context of autonomous driving is focused mainly on the camera + lidar setting. The machine learning approach is adopted to build models that are able to recognize the features of the object to be detected and predict the object's location and class (e.g. a car or a pedestrian). At the moment, the best performing methods are based on deep learning algorithms.

There have already been some commercial vehicle implementations, such as a Waymo car, see Figure 1-1. However, many manufacturers and researchers still struggle with delivering a vehicle ready for a series production. Apart from legal issues, which are not discussed here, there are many technical limitations that has to be considered. One of the factors to take into account is to provide the redundancy in the system, in case of the sensor's malfunction or poor operation conditions, e.g. a camera has limited performance when it is dark and a lidar in a foggy weather. Another difficulty is the placement of the sensors which ideally would be hidden within the chassis, unlike in the Waymo car (Figure 1-1).

Figure 1-1: A commercial autonomous vehicle developed by Waymo. Image [8].

Regarding how the data from different modalities can be combined, there are several approaches to sensor fusion. In the early fusion, the low level data such as raw data or feature maps is combined and then processed together. In the late fusion, also called decision fusion, data from different sources is processed separately and only the outputs are fused. Lately, also a method called deep fusion has been applied. It consists in fusing the data several times on the intermediate steps of the system.

In this work, we introduce a method for 3D object detection on camera and lidar data. The method is a deep learning based approach with deep fusion of lidar and camera features. We provide an end-to-end framework which automatically learns features from raw camera and lidar data.

1-1 Research goals

For this thesis, the following research goals have been set:

> **Goal**: Develop a new end-to-end deep learning based method for 3D object detection in the context of autonomous driving using camera and lidar data, with a focus on car detection.
>
> > **Subgoal**: Ensure that the method is able to learn from raw camera and lidar data without the need for any hand-crafted features.
>
> **Goal**: With a designed network, in the available benchmark achieve performance comparable with state-of-the-art methods
>
> > **Subgoal**: Achieve high detection rates in the 3D camera and lidar data
> >
> > **Subgoal**: Achieve runtime of the method around 10 Hz, so that it can be considered close to real-time detection.

1-2 Outline of the thesis

This work is structured as follows. A short introduction is given in Chapter 1. Chapter 2 introduces the theoretical background most important for this thesis. Specifically, data representation methods, basics of neural networks, and fusion methods are discussed. Later on, previous related work in the field of object detection with the focus on the autonomous driving scenario is presented in Chapter 3. Chapter 4 is devoted to the description of the proposed method. The general concept and architecture, as well as the implementation details are provided there. Subsequently, the results of the experiments and their evaluation and comparison are discussed in Chapter 5. Finally, Chapter 6 contains the conclusions, as well as the future work comments and the recommendations.

Chapter 2

Theoretical background

In this chapter, theory and background information about the subjects crucial for this thesis project are presented. At first, the possible ways of representing data from a camera and a lidar are explained. Later on, different fusion methods for different sensor modalities are described. Subsequently, a brief and general introduction to neural networks is given with the main focus on convolutional neural networks.

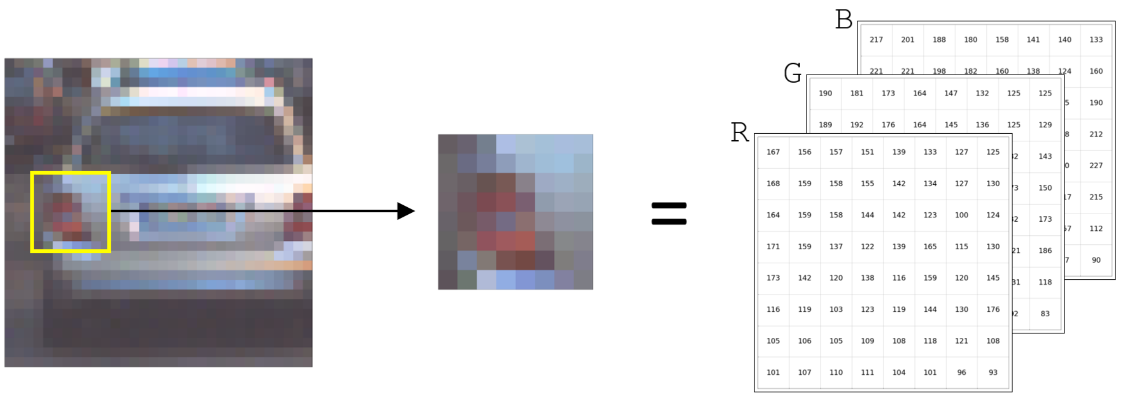
2-1 Data representation

A camera is a well known, relatively cheap, and widely used sensor. It captures the information of the environment in a mean of images or videos (sequences of images). Images are relatively easy to visualize and provide discriminative information. Nevertheless, due to the lack of a depth information it is a challenging task to detect objects of varying shapes, sizes, and orientations by using only one camera.

On the other hand, a lidar provides a 3-dimensional point cloud of the environment, which yields information about the distance in the scene, i.e. depth. This data is very well applicable for obstacle detection. However, point clouds obtained by lidar sensors lack color and texture information and only sparsely capture the environment.

2-1-1 Data from a camera

An image from a modern, digital camera can be represented by a two-dimensional, dense matrix. The smallest element of each image is a pixel which can be encoded with different structures, such as a single intensity value (e.g. gray-scale image), or a vector (e.g. RGB, or HSV values). In the latter case, the representation of the whole image becomes three dimensional, which is encoded with a 3D tensor. See Figure 2-1 for the machine interpretation of an image.



167	156	157	151	139	133	127	125
168	159	158	155	142	134	127	130
164	159	158	144	142	123	100	124
171	159	137	122	139	165	115	130
173	142	120	138	116	159	120	145
116	119	103	123	119	144	130	176
105	106	105	109	108	118	121	108
101	107	110	111	104	101	96	93

Figure 2-1: What people perceive as an image, for a machine is a tensor of numbers. Left image [7].

2-1-2 Data from a lidar

From the raw lidar data the following information can be extracted and prepared for the further processing: reflectance, distance, and relative XYZ position expressed in the local coordinate system. This information is contained in a list of points, which is not structured, i.e. permutation of the list does not change the resulting point cloud. Unlike data from a camera, data from a lidar is sparse, hence, it is not possible to represent the point cloud in a form of a dense tensor. Apart from being sparse, the data has highly variable density of points, with density decreasing with distance from the sensor. The reasons behind it are, e.g. non-uniform sampling of the space, effective range, occlusion, as well as relative position and orientation of the sensor. A visualization of an example point cloud used in this thesis is shown in Figure 2-2.

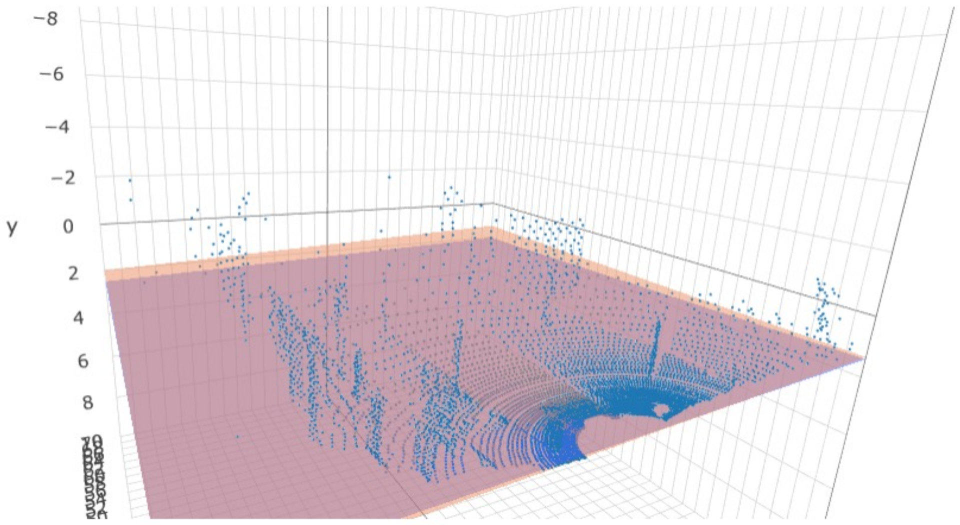


Figure 2-2: Example of a point cloud obtained by a lidar sensor.

There are many approaches available to process a point cloud provided by a lidar. Here two of them that are widely used in the literature related to object detection are presented. The first one is the voxelization, i.e. discretization of the euclidean 3D space into voxels (three-dimensional version of a pixel). This creates a grid of voxels which is most often equally spaced. This approach can be found in [9]. Another approach is to perform a projection of the point cloud into a 2D map, e.g. in [10, 11]. Popular projections are a front view and a top view, also referred to as a bird-eye view (BEV). While projecting the point cloud into either top view or front view it is possible to generate different maps, such as intensity map, height map, distance map, maps in longitudinal or lateral direction. The front view uses, for instance, a cylindrical projection [11].

Examples of lidar projections can be found in Figure 2-3.

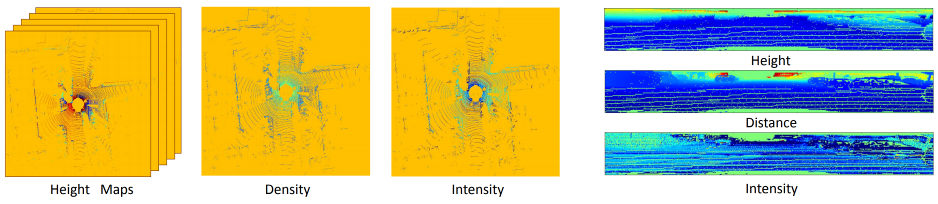


Figure 2-3: Example of BEV projections (first three from left) and a cylindrical, front view projection (three images on the right) [10].

2-2 Neural networks

Artificial neural networks (ANNs) or simply neural networks (NNs) are computing systems that use a network of functions to process complex data. NNs are loosely inspired by animal brains and the way biological neurons operate together. By looking at the Figure 2-4 one can see that the artificial neuron is indeed a very simple model of a biological neuron. Multiple artificial neurons are stacked in several layers, creating a network. A single artificial neuron is a mathematical function that transforms an input $\mathbf{x}$ into an output y,

$$y = f(\mathbf{w}^T\mathbf{x} + b) = f(\sum_{i=0}^{N-1} w_i x_i + b) \tag{2-1}$$

where N indicates the number of neurons in the layer. Weights w_i and bias b are trainable parameters that are updated during training process. Most commonly, the function f is a non-linear function and is called an activation function.

The architecture of NN can be divided into three main parts, namely, an input layer, hidden layers, and an output layer. An example in Figure 2-5a shows a fully-connected network, i.e. all of the nodes of a preceding layer are connected with all nodes of a subsequent layer. For the NN to be able to learn, one has to define a function that will indicate how close is a given prediction to the true value. Such a function is called a loss function (see Section 2-2-1).

Even though NNs are a relatively old concept [13], it is the recent increase in computational power that gave this field a significant push. Especially, neural networks with a large number of hidden layers, called deep neural networks (DNNs), had gained popularity in the past decade (see Figure 2-5b).

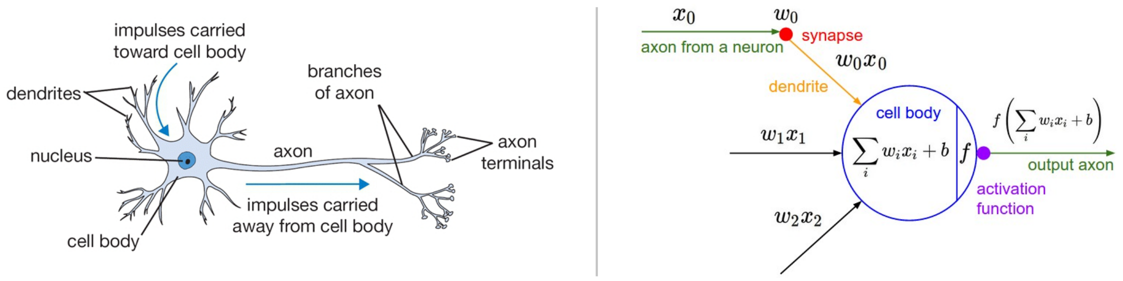


Figure 2-4: Each biological neuron (left) via dendrites receives the input signal from other neurons. The information is processed in the cell body and the output is sent further through the axon. A mathematical model (right) is inspired by the behaviour of the biological neuron, where "dendrites" lead the weighted input to the activation function. The output can be an input to another artificial neuron [12].

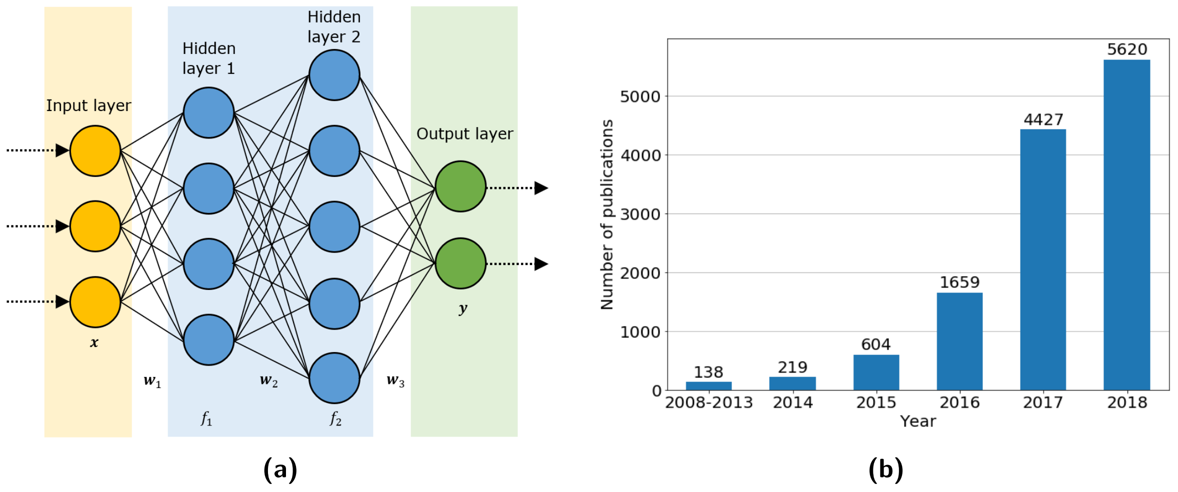


Figure 2-5: An example of a neural network with input **x**, two hidden layers f_1 and f_2, and output **y** (a). Number of publications with keywords "deep learning" and "neural network"[1](b).

DNNs are a part of deep learning approach for machine learning. In this thesis a term *deep learning* refers specifically to deep neural networks. Other branches, such as deep belief networks or recurrent neural networks are not considered here.

2-2-1 Loss function

During the training, the outputs of the network are compared with the provided ground truth. The network's task is to minimize this difference via minimizing the loss function. Different loss functions are suitable for different applications, for instance, for regression, a mean squared error (MSE, l_2 loss)

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2\,, \tag{2-2}$$

[1]Data retrieved from Scopus database on 15th of October 2018

where y_i is a true value and $\hat{y}_i$ is a prediction, or a mean absolute error (MAE, l_1 loss)

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{2-3}$$

and for classification, the cross-entropy

$$H(p, \hat{p}) = -\sum_i p_i \log \hat{p}_i, \tag{2-4}$$

where p is a set of true labels and $\hat{p}$ is a set of predictions.

2-2-2 Bias and variance

In the context of machine learning there are two important prediction errors called bias and variance.

In practice, if the bias is high, the model has poor performance on both training and the test data. This is called underfitting and can occur due to the fact that the model is too simple. In case of a high variance, the model performs significantly better on the training data than on test data. This means that the model has learned to fit too closely to the training data and cannot generalize well, which is called overfitting. Overfitting can be caused by an overcomplicated model or too few training data.

There are multiple approaches how to tackle both under- and overfitting. In case of underfitting, a solution could be choosing a more complex model. When it comes to overfitting, one uses such techniques as, e.g. l_1 or l_2 weight decay, or dropout. The idea of weight decay is to add a regularization term to the loss function, using an l_1 or l_2 norm:

$$J(\mathbf{w}) = L + \lambda \sum_i |w_i| \tag{2-5}$$

$$J(\mathbf{w}) = L + \lambda \sum_i w_i^2, \tag{2-6}$$

where L is the loss function, $J(w)$ is a regularized loss function, w_i are weights, and λ is a regularization parameter ($\lambda > 0$). On the other hand, dropout consists in dropping randomly chosen neuron activations and their connections during the training. This prevents complex co-adaptation between the neurons and consequently, reduces overfitting.

All methods used to reduce the generalization error are collectively called regularization techniques [14].

2-2-3 Activation function

According to Eq. (2-1), each node in the hidden layers applies a usually nonlinear function, called activation function, to the affine transformation of the input. Nowadays, the most commonly used node type is a rectified linear unit (ReLU), which uses the following activation function:

$$f(z) = \max\{0, z\}. \tag{2-7}$$

Prior to ReLUs, the most popular activation functions were the logistic sigmoid function and the hyperbolic tangent:

$$f(z) = \frac{1}{1+e^{-z}} \tag{2-8}$$

$$f(z) = \tanh(z). \tag{2-9}$$

Both of them saturate for most of their domain, what can cause problems with gradient-based learning. This is solved in ReLU ((2-7)), which is unbounded. On the other hand, the Eq. (2-7) is not differentiable at $z = 0$. However, in practice, gradient descent is found to perform well enough. A problem occurs when gradient-based methods are used for learning in cases where activation equals zero. For that purpose, several generalizations of ReLU, e.g. leaky ReLU, are proposed [14].

2-2-4 Convolutional neural networks

Convolutional neural networks (CNNs) are neural networks in which classic matrix multiplication is replaced by a convolution. They are especially suitable for data with a grid-like topology, such as images [14]. If the data is just as useful after permutations of data entries, one should not use CNN.

In general, the convolution is an operation on two functions, here continuous functions of time $x(t)$ and $w(t)$:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da \tag{2-10}$$

Most of the time one has to deal with discrete data, where t is an integer time index. Hence, the discrete convolution is defined as:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2-11}$$

In CNNs, the x is called the input, w the kernel (or filter), and s the output (or feature map).

It is possible to perform convolutions over more axes, for multidimensional tasks. For instance, when dealing with 2D images, also a 2D kernel can be used. This would be denoted as $S(i,j) = (I*K)(i,j)$, where i and j are indices over two axes, I is an image, and K is a kernel. As convolution operation is commutative, an equivalent expression $S(i,j) = (K * I)(i,j) = (I * K)(i,j)$ is more commonly used due to its implementation advantages [14]:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{2-12}$$

Typically, a convolutional neural network layer consists of three components. At first, at the convolution step, the kernel slides over all cells and the convolutions are performed on the input. Then, at the detector layer, an activation function, such as ReLU, is applied to activations obtained at the convolution stage. Lastly, *a pooling layer* is used to reduce the spatial dimension of the output by, for instance, taking a maximum over the values within the kernel. Consequently, the number of parameters in the network is reduced, which can

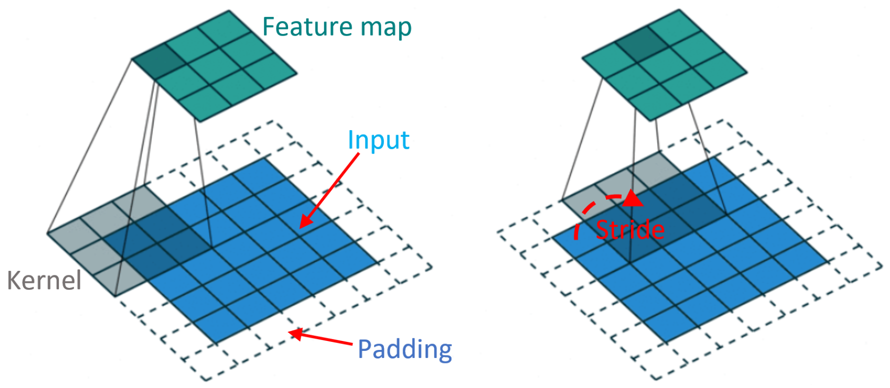


Figure 2-6: Convolution layer visualization together with the components, such as the kernel, input, feature map, padding, or stride. Figure from [15].

also help to control overfitting. In many architectures, the pooling layer is applied only once per multiple convolution and detector layers.

A visualization of how convolution in CNNs work is presented in an example in Figure 2-6. An inner product between the region from the input (blue) and the kernel of size 3×3 (gray) is computed and stored in the feature map in the corresponding cell (green). Commonly, padding is added around the input, i.a. in order to control the spatial size of the feature map. Usually, the same height and width of the output and the input after each convolutional layer is desired. Another parameter of a convolutional layer that can be defined is the stride. It indicates by how many pixels (cells) the center of the kernel moves at a time to compute the next value of a feature map. The most commonly used values of the stride are 1 and 2. Higher values cause high shrinking of the spatial dimensions [14, 15].

In this thesis also the terms *valid* and *same* are used when describing the padding. A brief graphical explanation can be found in Figure 2-7. *Valid* padding drops the last entries of data if the filter does not fully fit after the last stride jump in a particular direction. On the contrary, *same* padding tries to pad evenly both sides of the input. However, if there is an odd number of entries to be added, it adds the extra value at the end of data (in terms of indexing).

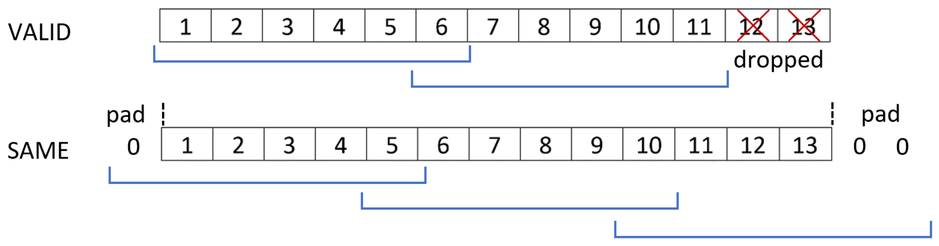


Figure 2-7: An example showing the difference between *valid* and *same* padding options. Parameters used in the example: input width 13, kernel width 6, stride 5.

1×1 **convolutions**

In the field of convolutional neural networks, 1×1 convolution introduced in [16] is an important concept that serves several purposes. 1×1 kernel size indicates that the filter does not care about the neighbouring data in the same feature map. Such a convolution can be used for dimensionality reduction across multiple feature maps similarly to how pooling is used for reducing the spatial dimensions of the feature maps. It is particularly useful for reducing the memory demand and increasing the efficiency of parts of the network. As the convolution is almost always followed by the non-linear activation function, 1×1 convolution can also be used to add additional non-linearity to the network architecture.

2-3 Fusion methods

In order to overcome the limitations of individual sensors, the systems should be able to make use of more sources of information. An example for that can be radar data, which is able to operate at night and is a range measurement device, and a camera, which can detect colours and in general reflects how humans perceive a scene. However, a camera is not able to perform well with insufficient lighting conditions or to easily determine the distance and velocity of an object. Thus, combining the measurements from both sensors allows for improved perception.

There are three conceptionally different fusion approaches, namely *early fusion*, *late fusion*, and *deep fusion*. For the sake of comparison to deep fusion, early and late fusion can also be collectively named as *shallow* fusion.

2-3-1 Early fusion

Early fusion, also called value fusion, combines raw data (measurements) or feature vectors from individual sources into a joint, fused feature vector. This vector is then an input to the network. An advantage of this approach is the fact that the employed classifier makes use of more information what allows for better performance. However, some of the techniques can be computationally demanding (also regarding the data storing) and in some cases the data alignment can be challenging.

Methods that can be used in this task are based on various kinds of pre-processing and projections. Data from two sources have to be properly aligned and transformed in order to be comparable and usable for further fusion.

An example of early fusion can be found in [17]. Thomanek and Wanielik combine data from two vision sensors in order to enhance the performance of pedestrian detection. At first, data from each source is pre-processed (image registration) and the features are extracted (edge maps). Subsequently, two extracted edge maps are combined using a probabilistic fusion rule (see Figure 2-8). Later on, the method uses a cascade of Adaboost classifiers to perform the pedestrian detection.

2-3-2 Late fusion

Late fusion is also called the decision fusion. In this approach, data from each source is processed individually, possibly using different methods/classifiers. Subsequently, the resulting

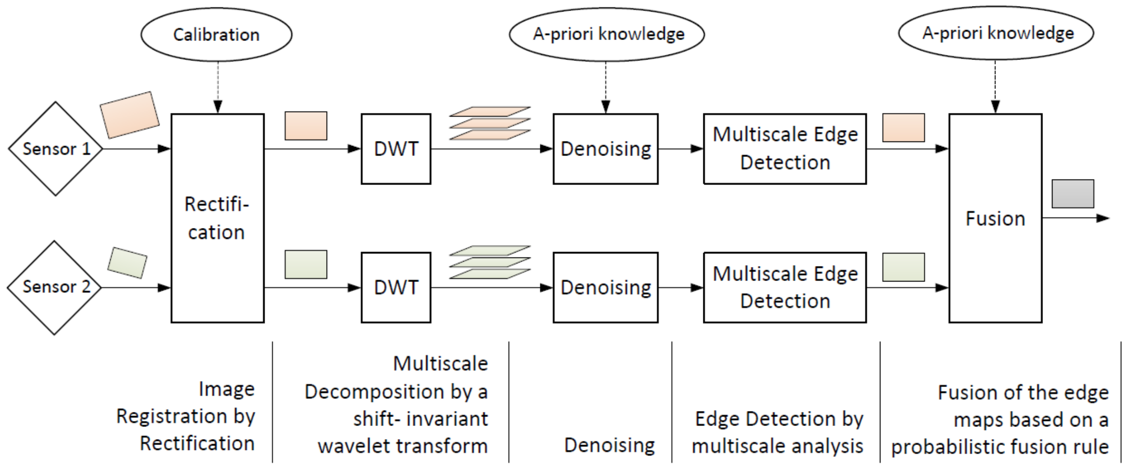

Calibration
A-priori knowledge
A-priori knowledge
Sensor 1
Sensor 2
Rectifi-cation
DWT
DWT
Denoising
Denoising
Multiscale Edge Detection
Multiscale Edge Detection
Fusion
Image Registration by Rectification
Multiscale Decomposition by a shift- invariant wavelet transform
Denoising
Edge Detection by multiscale analysis
Fusion of the edge maps based on a probabilistic fusion rule

Figure 2-8: Early fusion for pedestrian detection presented in [17]. After some basic processing of the individual sensor signals, early fusion is performed on the obtained features (edge maps). Subsequently, the fusion output is used for further detection algorithm.

outcomes are fused together to obtain a new, potentially more accurate or reliable, decision. In the late fusion it is possible to combine classifiers that are specialized and efficient in a specific class. However, in this approach, the already pre-processed data are combined and it can be hence prone to errors.

There are many different approaches to the decision fusion. Among the popular techniques are fuzzy inference systems [18], Bayesian classifiers [19], and evidential theory (Dempster-Shafer theory) [20].

Other popular decision fusion techniques are opinion pools and voting [21]. Opinion pool combines probabilities from multiple sources (classifiers, models, etc.). The coefficients are the corresponding weights, which give an indication about the contribution of a given source and can reflect the expert knowledge about the reliability of each source. The examples are a linear opinion pool and the logarithmic opinion pool. In voting, each source has to produce a decision regarding each class. Subsequently, the decisions are combined in majority voting.

An example of a work using the decision level fusion is [20]. Oh et al. combine the classification results of unary classifiers applied to each sensor for the task of object detection and classification. Two independent convolutional neural networks (CNNs) are processing the data from lidar and from camera (see Figure 2-9). The method uses basic belief assignment (within the aforementioned evidential theory) to fuse the resulting object bounding boxes and classes. In result, the performance of the detection algorithm is better than of the two methods individually.

2-3-3 Deep fusion

Deep fusion approach was introduced in [22]. The main idea of deep fusion is to fuse data or feature vectors multiple times within the neural network. The features extracted with the part of each network used in the architecture serve as an input to the remaining piece of each

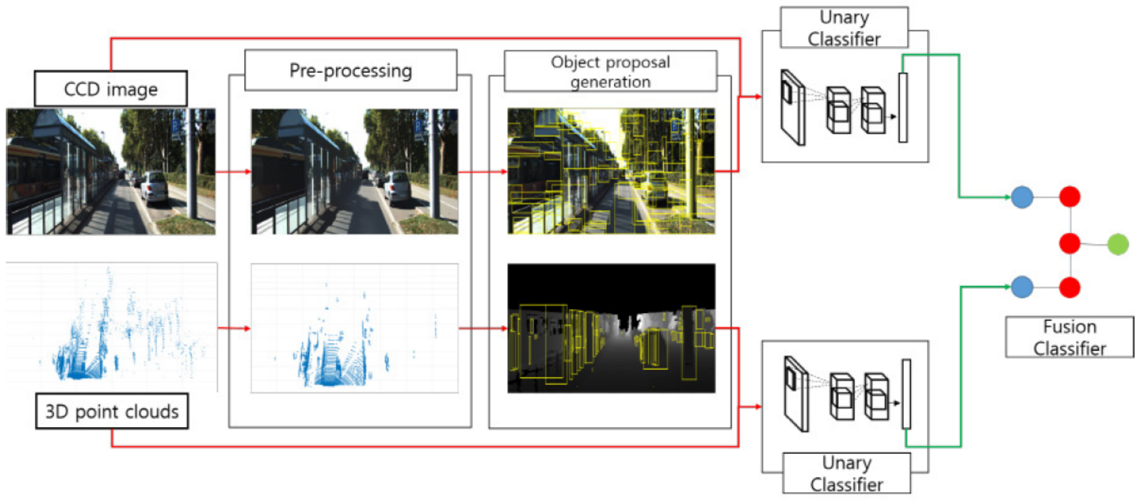


Figure 2-9: Decision level fusion of two CNNs [20]. Data from a camera and a lidar are processed independently. Both are fed into separate CNNs and only the network outputs are fused together.

network. One can notice, that deep fusion reminds an early fusion approach in which feature vectors are fused multiple times. Figure 2-10 depicts the idea behind deep fusion.

The authors of [22] claim that deep fusion has many advantages over a traditional shallow fusion, such as improved flow of the information or the capability to use multiple networks to learn multi-scale representations. A comparison of shallow and deep fusion can be found in [22].

The methods that utilize this approach are, for instance, [10, 23, 24].

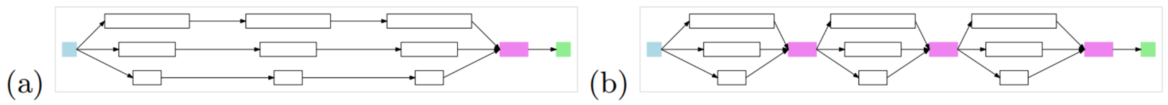


Figure 2-10: Shallow fusion (a) in comparison with deep fusion (b) [22].

Chapter 3

Related work

The task of object detection is crucial for any autonomous driving application. In order to be able to classify or, for instance, track an object, it has to be first localized in the scene. There are different approaches available, from the classical ones from early 2000s to the very recent deep learning methods. Nowadays, deep learning is the mostly applied, as it outperforms the classical approaches. An example outcome of one of the novel object detection networks can be seen in Figure 3-3a.

There are different sensors that can be used as data source, such as cameras, lidars, radars, ultrasonic sensors, or infrared sensors. Here, only methods which make use of camera and lidar data are considered.

Below, 2D object detection algorithms are described. Later on, the object detection in 3D is discussed. Three groups of 3D methods are presented, namely, the ones using cameras, the ones using lidars, and the ones using both of these sensors. Also, multiple options for the representation of 3D bounding-boxes are discussed, as they are an inherent part of the object detection task.

3-1 Classical approaches for object detection

Before the great success of deep learning techniques in the computer vision domain, most scientists were using hand-crafted features and shallow classifiers for object detection.

The most known detector of its time is an algorithm for object detection developed by Viola and Jones in 2001 [25]. Their method makes use of the Haar features which are obtained using a set of rectangular regions and the difference between the sum of pixels within them. Later on, AdaBoost is used for a cascade classification. The advantages of Viola and Jones detector are its robustness and speed, it is able to process a 384×288 pixel image in 0.067 seconds, using the computational power commonly available back in 2001.

Another remarkable work in the object detection field has been done by Dalal and Triggs in 2005 [26]. They introduced the Histogram of Oriented Gradients (HOG) as feature descriptors (see Figure 3-1). Later on, in their specific application of pedestrian detection, a linear SVM is employed for a classification between 'person' and 'non-person'.

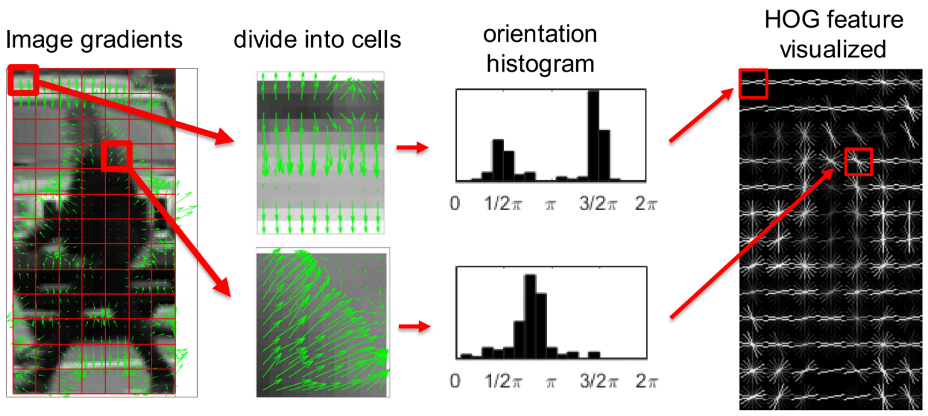


Figure 3-1: The principle behind the Histogram of Gradients (HOG) feature descriptor [26]. At first, an image is divided into cells. Then, a histogram of intensity gradient directions is calculated. Image [27].

An approach that is also based on HOG features is the Deformable Part Models (DPM) detector [28], primarly developed to detect people. Felzenszwalb *et al.* exploit the fact that the object is composed of deformable parts which are at some specific relative locations (see Figure 3-2). Therefore, instead of using only a model of a whole human body (as in [26]), the authors also use models of particular parts of the body, such as head, arm, or leg. An advantage of this approach is that it can handle situations where an object is partly occluded or, for instance, deformed. A classifier that is used is a latent SVM.

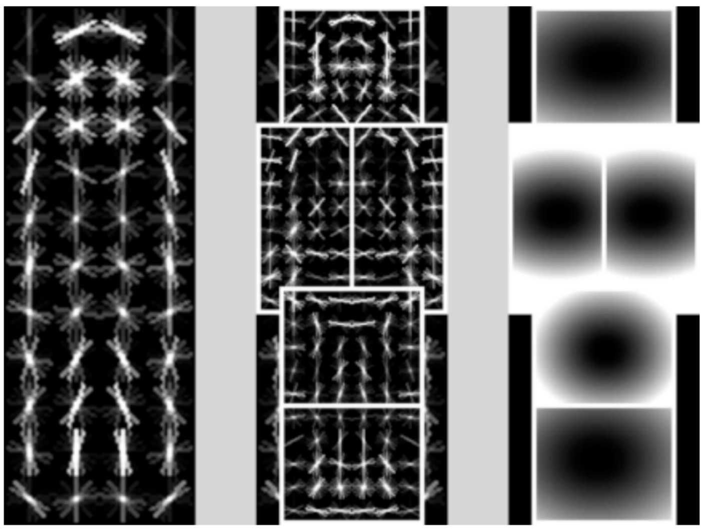


Figure 3-2: A scheme presenting the DPM approach [28].

3-2 Deep learning based object detection

In the deep learning field, there are two different approaches on how to perform the task of the object detection, namely two-stage and single-stage detection.

Two-stage detectors

The general rule of a two-stage detector is the following: at first, region proposals in which an object is expected are generated, then, in the second stage these regions are classified. This approach allows to use fast and not very demanding networks in the first stage to generate proposals. Then, in the second stage, an accurate and more complex classifier has to deal only with generated proposals rather than the whole input data. Such an architecture helps to keep satisfactory runtime of the method while obtaining high accuracy. Examples of methods adopting this approach are a Region-based Convolutional Network (R-CNN) [29] together with its improvements, namely Fast R-CNN [5] and Faster R-CNN [6], as well as a Region-based Fully Convolutional Network (RFCN) [30].

In R-CNN, at first, a selective search algorithm is used for generating region proposals. Then, each region is fed into a CNN and the output is classified with SVMs. A downside of this method is that it is extremely slow (single image inference time is approx. 50 seconds).

The performance of the R-CNN has been improved in the Fast R-CNN. There, the core CNN uses a whole image as an input instead of individual region proposals and the selective search is applied to obtained feature maps. Later, together with the Faster R-CNN (see Figure 3-3b, a Region Proposal Network (RPN) was presented, which was able to directly generate the proposals and predict the bounding boxes using a CNN feature map [6]. In the Faster R-CNN the RPN has a set of common convolutional layers with the Fast R-CNN, what makes the whole algorithm faster compared to using the selective search separately. The average inference time for a single image is 2.0 s and 0.2 s for the Fast R-CNN and the Faster R-CNN, respectively. Example detections obtained from Faster R-CNN are depicted in Figure 3-3b.

The RFCN also uses the RPN in order to generate the Regions of Interest (RoIs). The core CNN is using a whole image as an input and outputs a set of the so called position-sensitive score maps, which are specialized in detecting a particular class at a specific location. Later on, the RoIs are compared with the obtained feature maps and depending on the activation, the collective vote for a whole RoI is given (yes or no) and an object is detected accordingly.

Single-stage detectors

On contrary, there are single stage detectors that perform a detection task in one forward pass. In these methods the region proposals generation is eliminated and all computation is performed in a single network. The most popular single-stage detectors are You Only Look Once (YOLO) [4] and the Single Shot MultiBox Detector (SSD) [3]. Figure 3-4 presents an overview of both architectures [31].

In YOLO an input image is divided into a grid. Each grid cell is predicting a fixed number of bounding boxes with an associated confidence score. The classification is performed over all of the predicted bounding boxes. Moreover, unlike previously described detectors, YOLO

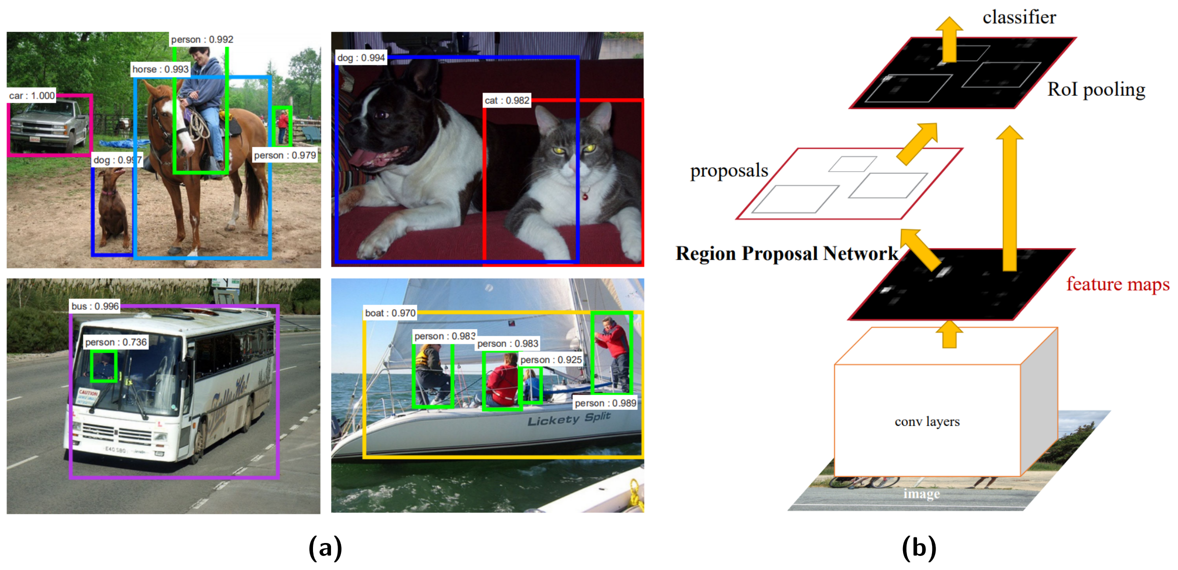


Figure 3-3: Examples of detections in the images, together with bounding boxes, labels, and scores obtained with the Faster R-CNN [6] (a), the architecture of the Faster R-CNN [6] (b)

uses a custom CNN architecture. This rather simple approach allows YOLO to be very fast and operate in real-time. Currently, also improved versions of the algorithm are available [32, 33].

A different approach is adopted in SSD. At first, SSD uses a base CNN to extract feature maps, namely the VGG-16 [34]. Then, additional convolutional layers which allow to predict detection at multiple scales are added to the end of the base network. In contrast, YOLO only uses one scale feature map. Each layer produces a number of detection predictions, based on default bounding boxes associated to each feature map cell.

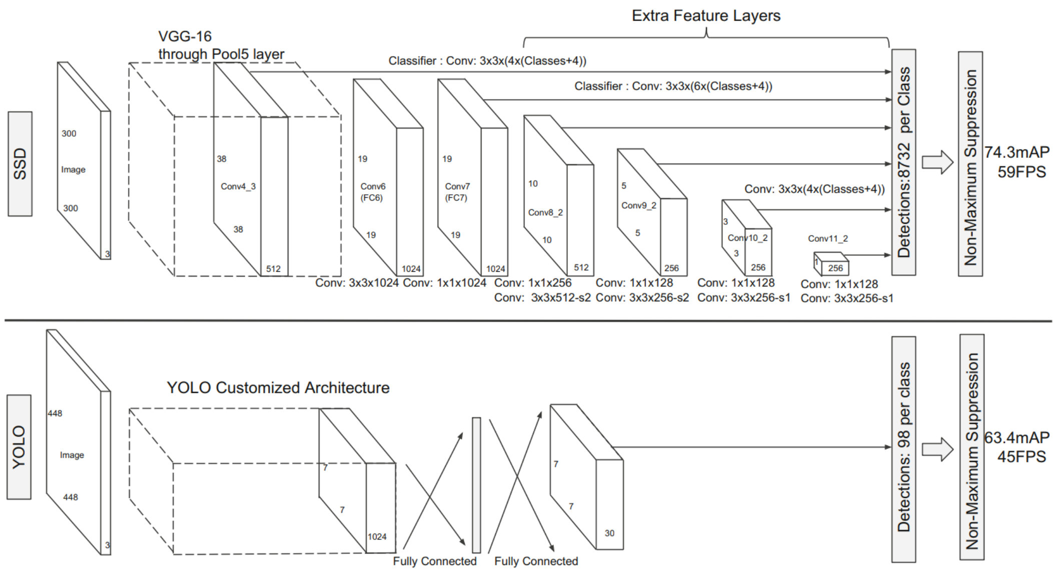

SSD
Extra Feature Layers
VGG-16 through Pool5 layer
Classifier : Conv: 3x3x(4x(Classes+4))
Classifier : Conv: 3x3x(6x(Classes+4))
Conv: 3x3x(4x(Classes+4))
300 Image 300
Conv4_3
Conv6 (FC6)
Conv7 (FC7)
Conv8_2
Conv9_2
Conv10_2
Conv11_2
Conv: 3x3x1024
Conv: 1x1x1024
Conv: 1x1x256 Conv: 3x3x512-s2
Conv: 1x1x128 Conv: 3x3x256-s2
Conv: 1x1x128 Conv: 3x3x256-s1
Conv: 1x1x128 Conv: 3x3x256-s1
Detections:8732 per Class
Non-Maximum Suppression
74.3mAP 59FPS
YOLO
YOLO Customized Architecture
448 Image 448
Fully Connected
Fully Connected
Detections: 98 per class
Non-Maximum Suppression
63.4mAP 45FPS

Figure 3-4: An overview of SSD (top) and YOLO (bottom) architectures [31].

3-3 3D object detection in the context of autonomous driving

While 2D object detection is a well developed and mature field of science, 3D object detection has still many open and unsolved problems. To motivate why one needs to perform 3D object detection, one can cite the example of an autonomous vehicle. In order to properly react to dynamic situations, such a vehicle needs to reason about which objects are present in the scene as well as about their 3D location, orientation, and 3D extent.

3-3-1 3D bounding-box representation

The easiest way to describe a 3D object is a bounding box that covers object's full volume. The smallest number of parameters for a unique representation of a 3D bounding box equals to nine, i.e. three spatial coordinates of the bounding box's centre, three dimensions (length, width, height), and three angles (roll, pitch, yaw). However, many other redundant representations are also popular. Some of the examples are given in Figure 3-5.

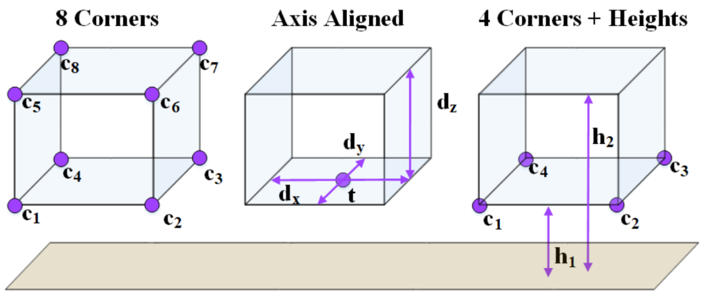


Figure 3-5: Different 3D bounding box representations used in the literature [23]. Left: 8 corners representation [10], middle: axis aligned representation [35], right: 4 corners + 2 heights representation [23].

Apart from the bounding box representation, some methods try to fit the best 3D models (e.g. CAD models in [36]) to the objects in regions of interest.

3-3-2 Camera based methods

Regarding the object detection methods that are using cameras, one can divide them into the ones using a single camera and the ones using a stereo camera setting. Using a setup of two cameras has an advantage of being able to provide more information about the observed environment, e.g. a depth map. A downside of this approach is the need of a precise calibration. Another problem is that, most of the mono camera methods for 3D object detection prove to be too slow for real time applications [37]. For instance the best performing method in KITTI benchmark for 2D object detection with orientation estimation [37] is around 0.7 seconds. For many of the methods in this section the performance in 3D benchmark [1] is not available, thus, the 2D benchmark [37] scores are used for comparison.

Mono camera

One of the approaches is to first generate object proposals and then pass them to a CNN. This is used in Mono3D [38], where the authors generate 3D object proposals in three different sizes of the bounding boxes, corresponding to cars, cyclists, and pedestrians. They take into account segmentation, contextual features, as well as location priors. Subsequently, obtained proposals are fed into a CNN to generate final labels and oriented bounding boxes. The network used by the authors is the same as the one used for a stereo setting in [39], written by the same authors.

A similar approach is taken by Mousavian *et al.* in [40], who propose Deep3DBox, where an already well-performing 2D object detector is extended by training a deep CNN to additionally regress oriented 3D bounding box. The method makes use of the fact that a 2D bounding box fully encloses the perspective projection of a 3D bounding box of the same object.

Whereas other methods use only 3D bounding-boxes, a different approach is taken in Deep MANTA (Deep Many-Tasks) introduced by Chabot *et al.* in [36]. This method additionally uses a special 3D mask vehicle dataset for the 3D detection of cars. In Deep MANTA, at first, convolutional layers provide feature maps, which are later fed into a network outputting object proposals with associated vehicles' parts, parts visibility, and 3D mask template similarity. After removing any redundant detections, the best fitting 3D model is chosen from the vehicle dataset and 2D/3D matching is performed in order to obtain the position and orientation of the car in the image plane.

Deep MANTA is currently the best performing method in KITTI benchmark for 2D object detection with orientation estimation [37]. However, its runtime in this benchmark is around 0.7 seconds which is not fast enough for a real time application.

Stereo camera

A stereo camera system makes it possible to compute depth and therefore, it can be used for the 3D detection. The depth information can be further used to generate a point cloud. This approach is adopted by Chan *et al.* in the 3DOP method [39] and by Pham and Jeon in [41]. Both methods share the same approach to the initial proposal generation and the detection. The proposal generation problem is formulated as energy minimization task, using Markov Random Field with energy function. The generated bounding boxes are scored with Fast R-CNN [5] with added context branch after the last convolutional layer and orientation regression loss for simultaneous location and orientation learning. Additionally, in [41] the authors introduce a re-ranking algorithm with a two-stream CNN called a DeepStereoOP, that exploits features such as a disparity map and a 'distance to the ground' map.

In the KITTI benchmark, the DeepStereoOP outperforms 3DOP by a small margin in the 2D car detection with orientation estimation, but is still slightly worse in pedestrian and cyclists classes. Both methods have a runtime of more than 3 seconds per image, which is not applicable to real time detection tasks.

3-3-3 Lidar based methods

There are two substantially different approaches to tackle lidar data. One of them is based on voxels and the other one is based on projection maps. Detailed description of both representation methods can be found in Section 2-1-2.

Voxels

One of the ways to work with voxels is to use voting. Examples of methods using this approach are Vote3D [42] by Wang *et al.* and Vote3Deep by Engelcke *et al.* [43].

In Vote3D, the authors use a sliding window in a point cloud, but first, in order to avoid huge computational expense, the point cloud is discretized into a voxel grid and voting is used to determine whether each cell is occupied or empty. Vote3D, apart from voting, uses exhaustive search for detection and SVM for classification.

On the contrary to the Vote3D, Vote3Deep processes a full 3D point cloud using a neural network. Depending on the class (car, pedestrian, or cyclist), the network's last layer's kernel has a corresponding size. This method outperforms the one that uses sliding window detection and SVM classification in terms of accuracy (only 2D detection results are available), but runs noticeably slower with a runtime of 1.5 seconds per point cloud.

A method using a different approach to process voxels is presented in [9]. Zhou and Tuzel introduce VoxelNet, which consists of three main parts: feature learning network, convolutional middle layers, and region proposal network (RPN), see Figure 4-2. The key innovation proposed by the authors is a voxel feature encoding (VFE) layer that is able to learn features from a raw point cloud. A more detailed description of the VoxelNet architecture is given in Section 4-2.

In [44] Yan *et al.* present the network SECOND (Sparsely Embedded CONvolutional Detection), which introduces significant improvements to the VoxelNet approach. The authors use sparse convolutions to deal with the sparse lidar data. This approach allows them to achieve much faster detection than the original VoxelNet, with the speed of the detector of 20 FPS.

Projection maps

A different way of handling lidar point cloud data is to use projection maps instead of voxels, thus bringing point cloud data in an image-like structure.

A projection map approach is adopted, i.a. in LMNet [11] using front-view projection and in RT3D [45] or [46] using BEV.

In the LMNet, Minemura *et al.* use front view maps obtained from the lidar point clouds. A cylindrical projection is used to obtain a sparse 2D point map. At first, five different feature maps are generated representing reflection, range, longitudinal and lateral coordinate values, and height. Then, these maps are fed into the LMNet network. There are two outputs of the network, one is the objectness classification with a softmax loss function and the second one is the bounding box's corners offset regression with smooth L1 loss. The inference time for one point cloud is 0.02 seconds, however, the method achieves relatively poor results in 3D object detection benchmark.

Regarding the works using bird-eye-view projections, in the Real-Time 3-D Vehicle Detection (RT3D) network by Zeng *et al.* [45], a depth map is encoded by taking maximum, average, and minimum height of projected points in each grid cell. The authors propose pre-RoI pooling convolution technique for better computation efficiency and obtain the detection time of 0.09 s. However, this speed increase is achieved at the cost of accuracy of the detection.

While RT3D uses only depth map as the input to the feature extractor, Wirges *et al.* in [46] use features such as intensity maps, or height difference maps. They also remove the estimated ground plane. As the meta architecture the Faster R-CNN [6] is employed. Both RT3D and the work by Wirges *et al.* use two-stage detectors and use Resnet [47] as an initial feature extractor.

3-3-4 Camera and lidar based methods

Among the top performing methods in the KITTI 3D Object Detection Benchmark most of the algorithms fuse the data from camera and lidar (see Table A-1). This allows to benefit from the advantages of both sensors, i.e. the human-like perception of the scene provided by a camera and the depth information provided by lidars.

There are different approaches to data fusion. More information about fusion methods are presented in Section 2-3.

In general, one can discriminate between early, late, and deep fusion. In the case of early fusion, what is being combined is the raw data from two sources or the initially processed data, for instance, feature maps. In the late fusion, the data from two sources is first fully processed separately and then, the outcomes are accordingly combined. Deep fusion approach fuse data or feature vectors multiple times within the neural network. Deep fusion is similar to an early fusion approach in which feature vectors are fused multiple times.

Among the available methods, some of them use the early fusion approach, e.g. F-PC_CNN [48]. Deep fusion can be found in the MV3D method [10], AVOD [23], and in the work by Liang *et al.* [24].

Additionally, some methods use data from different sources sequentially and the data from camera and lidar are not directly fused. To this group belong, i.a. F-PointNet [49] or RoarNet [50]. In this thesis such approaches are referred to as sequential fusion.

Early fusion

An example of a 3D object detection method using early fusion can be F-PC_CNN by Du *et al.* [48], which focuses on car detection only. In this method, a 2D detection network generates a bounding box of the candidate cars and the result is fused with a 3D point cloud to obtain a 3D information. To detect subset of points that belong to the car Du *et al.* make use of the generalized car models. After that, a 3D bounding box of the vehicle is generated and a two-stage CNN is used to perform the fine tuning of the detected bounding box and to assign *objectness* score. The inference time for each scene is equal to 0.5 seconds which is not satisfactory for real-time applications.

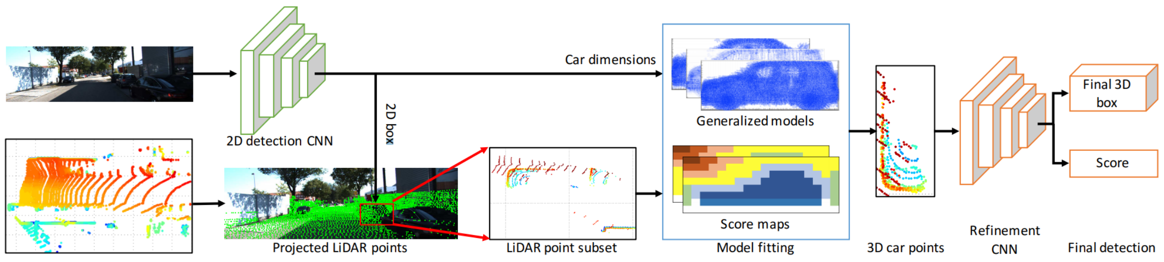


Figure 3-6: General pipeline for fusion of any 2D detector with 3D point cloud [48]. In the image all point clouds are in 3D, but shown from the top. The color indicate height and red encode the ground. A part of point cloud is selected based on 2D detection. A specially designed model fitting algorithm is applied to find vehicle points. At this point a bounding box can be assigned to the car. In the last step, a two-stage CNN is used to tune both the 3D bounding box and an objectiveness score.

Deep fusion

Several methods for 3D object detection perform deep fusion of data obtained from a camera and from a lidar. Among them are MV3D by Chen *et al.* [10], AVOD by Ku *et al.* [23], and a method proposed by the researchers from Uber by Liang *et al.* [24].

The MV3D method [10] is applied to car detection only. It takes three inputs, i.e. an RGB camera image, and two images of projections of a point cloud. The first projection is a bird-eye view (BEV), which is encoded by height, intensity, and density. The second projection of the point cloud is a front view (FV) with height, distance, and intensity maps. The network for the proposals generation is inspired by RPN [6]. The proposals are generated from BEV map and projected to all views. As features from different sources can have nonuniform sizes, a ROI pooling [5] is applied to each region in each view. The obtained fixed-size feature maps are an input to deep fusion network which generates the final representation. Deeply fused features are used for objectness classification and oriented 3D box regression. MV3D is rather slow in comparison to other state-of-the-art detectors, with an inference time for one scene equal to 0.36 seconds. An overview of the MV3D approach is shown in Figure 3-7.

In the work by Liang *et al.* [24], a deep continuous fusion method for the camera image and the BEV image is proposed. Fusion is performed on multiple scales. The method makes use of the continuous convolution concept to construct the continuous fusion layers, in which the camera image and the BEV image are combined. In some cases the pixels of the BEV image cannot be directly matched with the ones from a camera image, as they may be not observable. The presented approach uses k nearest lidar points and their projection onto the camera plane to find the missing feature pixel via interpolation and then proceeds further with the feature fusion (see Figure 3-8). The method is very fast, it runs at more than 15 FPS.

Moreover, the Aggregate View Object Detection (AVOD) by Ku *et al.* [23] uses a fusion method called the Feature Pyramid fusion. In this case, the fusion is performed multiple times between intermediate feature maps. The method takes as an input an RGB image and a bird-eye view (BEV) projection of a lidar point cloud. Then, each input is fed into a VGG-16 [34] based network for feature extraction. Subsequently, its output is passed through upsampling layer to obtain higher resolution feature maps. Later on, an RPN regresses the difference

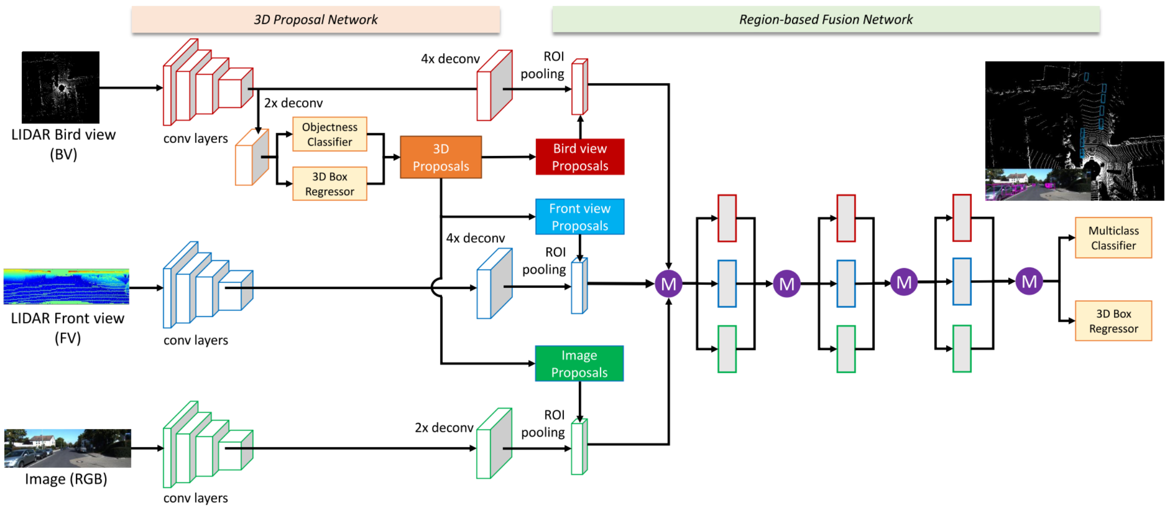


Figure 3-7: MV3D architecture [10]. The method takes three inputs to the network. At first, 3D object proposals are generated from BEV map and projected to all three views. ROI pooling is performed to obtain region-wise features of the same size. A deep fusion network is used to combine the region-wise features, which are used to regress an oriented 3D bounding box and predict the class of the object.

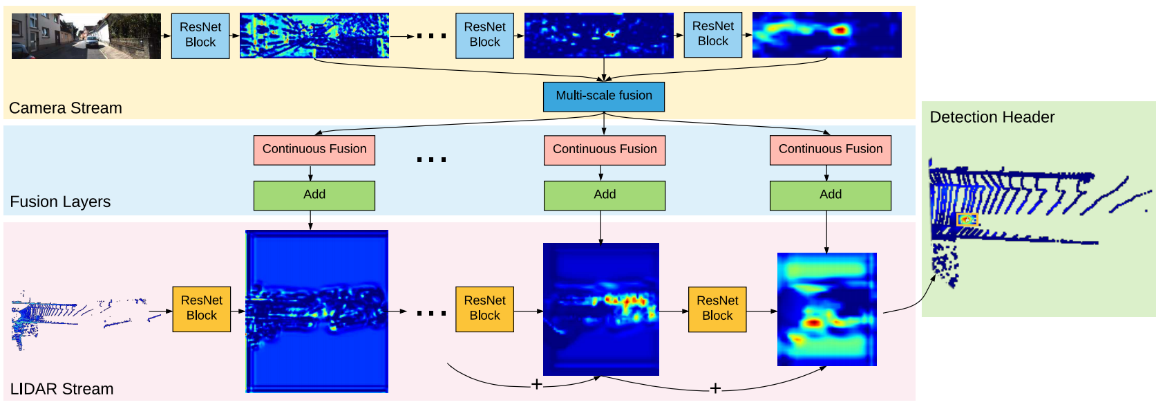


Figure 3-8: Architecture of the model presented in [24]. The camera and lidar streams are fused using continuous fusion layers.

between prior 3D boxes (anchors) and the ground truth. Given an anchor in 3D, two regions of interest are obtained by projection of the anchor onto the BEV and the image feature maps and they are used to extract crops from the maps. These are later fused via element-wise mean. Two branches of fully connected layers of size 256 are used to regress object proposal boxes and to obtain the objectness score. Best proposals are projected onto feature maps, cropped, resized, and fused using an element-wise mean. More detailed description of this method can be found in Section 4-1.

Sequential fusion

One of the methods using sequential fusion is F-PointNet by Qi *et al.* [49]. This method first detects the object in a 2D image and then searches for this object in a 3D space. It is based on 3 modules: frustum proposal, 3D instance segmentation, and 3D amodal bounding box estimation. At first, a 2D object detector proposes 2D object regions in RGB images and the 2D box is subsequently lifted to a frustum defining a 3D space for the object, see Figure 3-9.

Then, an instance segmentation is performed in 3D point cloud using PointNet [51]. Given the segmented object points, the last module estimates the object's amodal oriented 3D bounding box. It contains the full object even if occluded or truncated. F-PointNet achieves very high accuracy in the 3D object detection benchmark for cars, cyclists, and pedestrians. The runtime of the method is 0.17 seconds per scene.

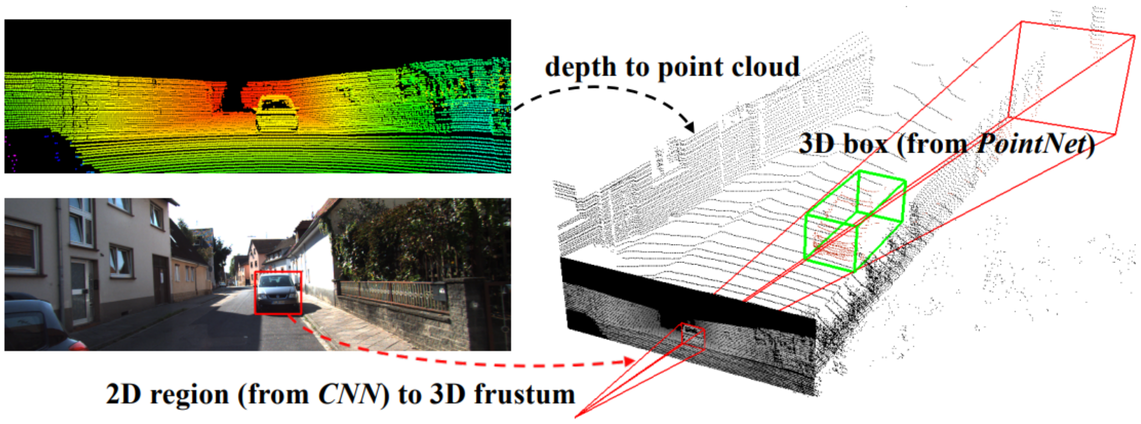


Figure 3-9: Object detection pipeline for F-PointNet [49]. At first, 2D object region proposals are generated in the RGB image using a CNN and for each 2D region a 3D frustum is extracted. Each frustum contains points from depth data obtained from lidar point cloud. Finally, PointNet [51] uses points within each frustum to predict an oriented 3D bounding box of the object.

A similar concept is implemented in the Region Approximation Refinement Network (RoarNet) by Shin *et al.* in [50]. The authors first use mono camera image to estimate the 3D poses of objects in the scene and, in result, choose potential regions with objects to be detected. This allows to reduce the search space significantly. Further, the second part of the network processes the point cloud data from the proposed regions. This part is a two-stage detector based on the PointNet [51]. According to the authors, RoarNet performs well not only on the synchronized data but also can handle cases in which the camera and lidar synchronization is not very precise.

Chapter 4

Method

Based on the analysis of the already existing methods a new approach to address the problem of 3D object detection is proposed in this thesis. The proposed architecture uses data from an RGB camera and a lidar.

One of the methods that performs well in the KITTI benchmark [1] is AVOD [23], see Table A-1. AVOD is a deep learning approach that uses data from both a camera and a lidar for detecting objects. The method projects the point cloud collected from the lidar into the BEV images and processes them in the same way as the input from camera. The fact that data from two substantially different modalities are processed with the same approach is the factor that can potentially worsen the performance. Additionally, with BEV projection, some height information is lost before even entering the trainable network. Thus, improvements can be made in the AVOD architecture that could help to achieve better performance of the detection, especially regarding how the method deals with the point cloud data.

A method that uses only point cloud data and still achieves a very good results is VoxelNet [9], see Table A-1. In this approach, features are learned from points in their raw form, meaning that all 3D information is being kept. VoxelNet discretizes the space into voxels of equal sizes in order to apply 3D convolutional networks that learn features. Even though, the method achieves good results, it does not utilize data from camera which potentially could improve the performance.

A method to leverage the advantages of both AVOD and Voxelnet is the main contribution of this work, i.e. using the two sensor modalities of camera and lidar in a deep fusion framework with learned features from camera and lidar data.

In this chapter, at first, the details of both AVOD and VoxelNet that are crucial for this work are explained. Subsequently, the proposed network architecture for 3D object detection is presented, together with a detailed description of the design and the training processes.

4-1 Aggregated View Object Detection

Aggregated View Object Detection (AVOD) takes as inputs an image and a point cloud. The image input modality is taken as RGB tensors which are preprocessed in two steps. At first, images are bilinearly resized to the same shape. Later on, for each RGB channel a mean of a particular channel across all images is subtracted from each image.

The point cloud is also preprocessed to obtain a dense structure that can be fed into a feature extraction network composed of several convolutional layers. In particular, the point cloud is cropped in the area of camera's range in the lateral and longitudinal directions, from -40 to 40 meters and from 0 to 70 meters, respectively, relative to the origin of camera coordinate system. This space is discretized with the resolution of 0.2 meters resulting in a feature map of size 400×350. For each cell of the feature map there are six channels obtained as follows. First, to keep the information about the height, the point cloud is cropped from -0.2 to 2.3 meters in the horizontal dimension, measured from the ground plane. The authors do not provide details on the ground plane estimation algorithm. The height is divided into cells of 0.5 meters. The first five channels are the maximum height of the points contained in each cell. The last channel contains point density information and is defined as $\min(1.0, \frac{\log(N+1)}{\log(16)})$ where N is the number of points in each cell. This results in a *preprocessed point cloud*, represented as a tensor of $400 \times 350 \times 6$.

An overview of the AVOD architecture can be found in Figure 4-1. The architectures of feature extractors are the same for both the image and the preprocessed point cloud and they are based on the VGG16 network [34] with some modifications. In particular, the number of channels in each layer is reduced by half and some layers at the end of the original network are cut [23, 34].

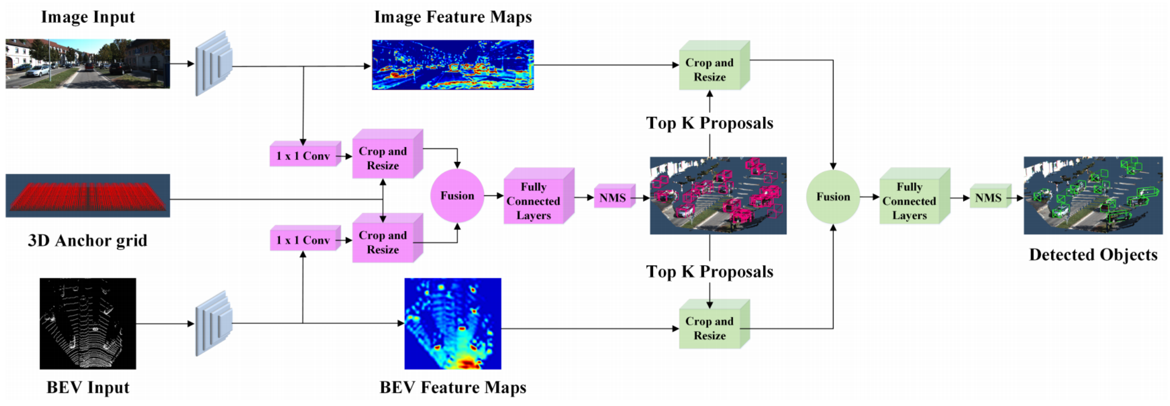


Figure 4-1: AVOD architecture [23]. Feature extractors are depicted in blue, the RPN in pink, and the second stage network in green.

AVOD is structured as a two stage detector (see Section 3-2) and is using a 3D anchor grid to obtain object proposals in the first stage. The anchors are placed on the ground plane with intervals of 0.5 meters in longitudinal and lateral dimensions. The size of the anchors is determined by clustering the sizes of ground truth boxes for each class that is being used.

Feature maps from both sources are passed to Region Proposal Network (RPN). To reduce the memory demand, feature maps are passed through 1×1 convolutional layer (see Section 2-2-4). Subsequently, an anchor grid is used in order to take crops from these reduced feature maps. The crops are further bilinearly resized, such that the spatial size of crops from both feature maps is the same. The fusion of the features from both modalities is implemented by taking the mean of the corresponding values. Fused data from crops is passed through a fully connected neural network to regress boxes and obtain *objectness* score with smooth L1 and cross-entropy losses, respectively [5, 6]. During training, the best 1024 proposals are kept.

In the second stage, crops from full, non-resized feature maps are taken, but only from those 1024 best proposals obtained with RPN. Similarly to what is done in RPN, they are later resized, fused, and passed through fully connected network to obtain final results for bounding box and *objectness* score.

4-2 VoxelNet

In opposite to AVOD, VoxelNet uses only point cloud data from lidar to perform 3D object detection. Unlike in most of the other approaches, in VoxelNet, features are learned from the raw points. An overview of the VoxelNet architecture can be found in Figure 4-2.

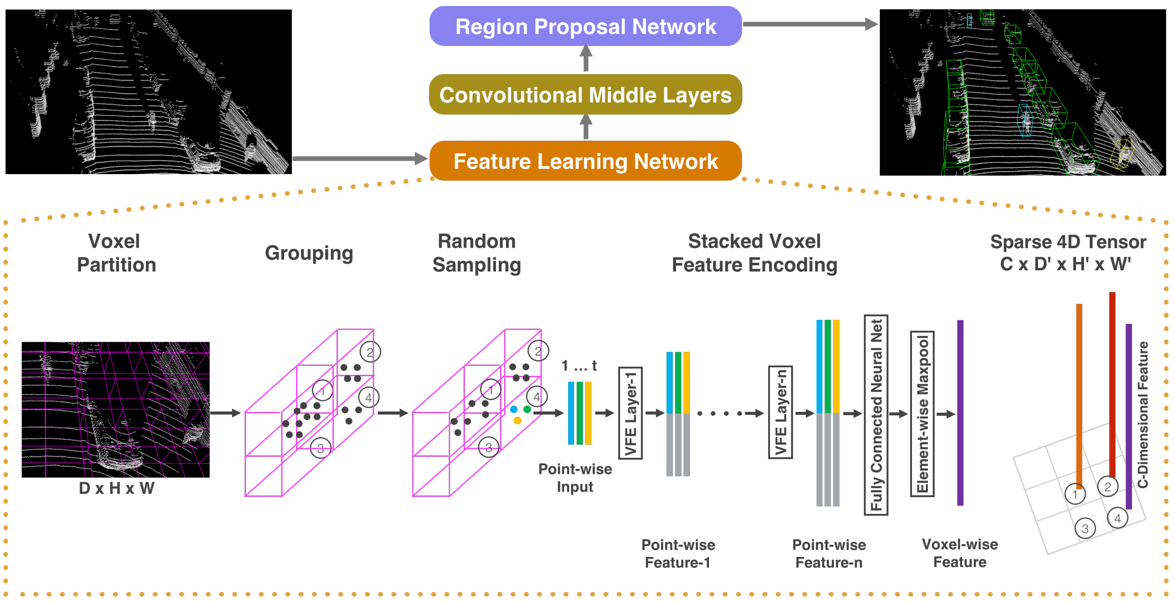


Figure 4-2: The overview of VoxelNet architecture with the emphasis on the Feature Learning Network [9].

The point cloud is cropped in the range of $[x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [z_{min}, z_{max}]$ meters in the longitudinal, lateral, and horizontal dimensions, respectively, with an origin at the lidar coordinate system. The cropped point cloud is discretized into voxels of equal sizes. The height of each voxel is v_z meters, the width is v_y meters and the length is v_x meters. For the purpose of evaluation, the authors use KITTI dataset [7] and benchmark [1]. In the training specifications for the *car* class, the following values (in meters) for the above variables are given:

$$[x_{min},\ x_{max}] \times [y_{min},\ y_{max}] \times [z_{min},\ z_{max}] = [0,\ 70.4] \times [-40,\ 40] \times [-3,\ 1] \tag{4-1}$$

$$[v_x,\ v_y,\ v_z] = [0.2,\ 0.2,\ 0.4] \tag{4-2}$$

Thus, the considered 3D space has the size of 352 × 400 × 10 voxels. For each non-empty voxel, the maximum number of N points residing in the voxel is sampled. This serves the purpose of increasing computation efficiency and having more uniform distribution of points. Additionally, for each non-empty voxel, a mean of the points is computed, and for each point within the voxel, the mean is subtracted and concatenated creating the input to the network.

A point cloud grouped in the described way is passed to the Voxel Feature Encoding (VFE) layer (Figure 4-3) that learns features from points in each non-empty voxel. Specifically, all points in the voxel are passed to the fully connected neural network that transforms the input data into a point-wise feature. Subsequently, the element-wise pooling is applied to compute a locally aggregated feature. Finally, each point-wise feature is augmented with the locally aggregated feature.

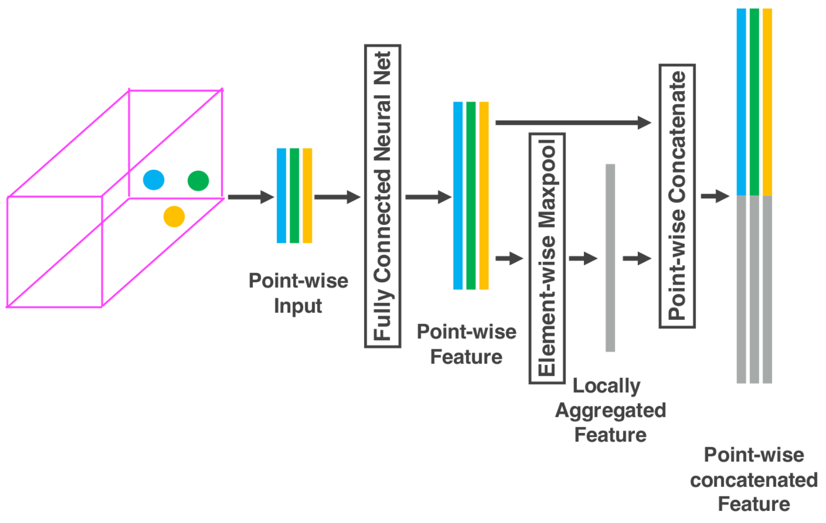

Point-wise Input
Fully Connected Neural Net
Point-wise Feature
Element-wise Maxpool
Locally Aggregated Feature
Point-wise Concatenate
Point-wise concatenated Feature

Figure 4-3: Voxel feature encoding layer introduced in VoxelNet [9]. At first, for each point, a fully connected neural network transforms the input data into the point-wise feature. Then, the locally aggregated feature is computed with element-wise max pooling. At last, the locally aggregated feature is concatenated to each point-wise feature.

Each VFE layer can be denoted as VFE(x_{in}, y_{out}), meaning that the input feature of size x_{in} is transformed into the output feature of size y_{out}. Multiple VFE layers can be stacked to learn more complex feature maps (see Figure 4-2). For the training on the KITTI dataset, two VFE layers are used, namely VFE(7, 32) and VFE(32, 128). The output from the last VFE layer is passed to another fully connected neural network followed by element-wise max pooling operation which transforms the point-wise feature into a voxel-wise feature.

After the process of voxel-wise feature extraction is done, each voxel is described with one learned vector. For the KITTI dataset training, the space is described with a tensor of shape 352 × 400 × 10 × 128.

To further process the data, three layers of 3D convolutions are applied to the voxel-wise features (see Table 4-1 for KITTI dataset implementation details). Subsequently, 2D convolutions are used.

At the last stage, similarly to AVOD, VoxelNet also uses the smooth L1 loss to regress boxes and cross-entropy loss to obtain *objectness* score [5, 6].

4-3 Proposed method

The 3D object detector introduced in this thesis is inspired by the architecture of AVOD. The proposed method uses camera images and lidar point clouds as inputs for a deep learning based object detection network.

As mentioned before, AVOD preprocesses the point cloud to obtain a dense structure before passing it to the feature extraction network. BEV projections that are used discard considerable amount of information about the height. Due to that fact, some information is lost in the preprocessing step preceding a trainable object detection network. Additionally, in AVOD the data from two substantially different modalities is processed with the same feature extractor. In particular, the feature extraction network treats the projected point cloud input as images and processes them with identical 2D convolutional networks.

In opposite, VoxelNet extracts features directly from points in 3D, keeping more spatial information. VoxelNet discretizes the point cloud into voxels in order to impose a dense structure on the data. For each non-empty voxel, a feature that represents the structure of points is learned with fully connected neural network. A dense structure allows to use a 3D convolutional network on voxel-wise features.

In the proposed architecture, the network layers which use preprocessed lidar point clouds in AVOD are replaced with the voxel feature encoding layers as introduced in VoxelNet. This allows for an end-to-end training that avoids any kind of hand-crafted features.

See Figure 4-4 for an overview of the proposed architecture.

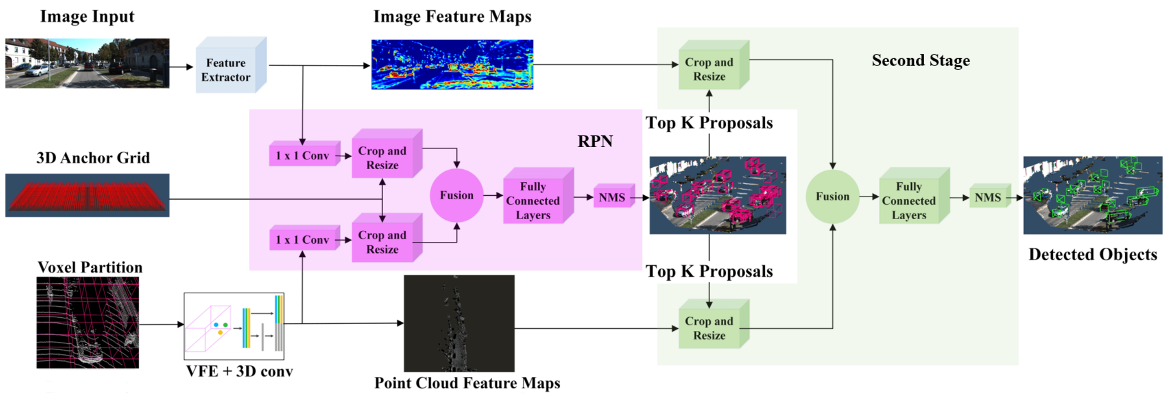


Figure 4-4: The architecture of the proposed method. Image features are extracted by an adapted VGG16 network. Point cloud features are extracted from voxel partitions by applying Voxel Feature Encoding (VFE) layers and 3D convolutions. In a Region Proposal Network (RPN), 1×1 convolution is applied to the feature maps to reduce their size. The anchor grid is used to crop the proposals from the reduced feature maps. After resizing to a common size, the features from both modalities are fused and a fully connected neural network outputs the best proposals. In the second stage, the best proposals from RPN are cropped from the full feature maps and fused. Object detection layers are implemented by fully connected layers operating on the fused crops. This allows for an end-to-end network which detects 3D objects from camera and lidar sensor data. Image adapted from [23, 9].

4-3-1 Coordinate system for the point cloud

In AVOD all of the data being processed is represented in the camera coordinate system. For the proposed architecture, a new coordinate system is introduced, hereinafter referred to as *voxel coordinate system.* Both coordinate systems are pictured in Figure 4-5.

A transformation matrix from the camera coordinate system to the voxel coordinate system is defined as follows:

$$T_{voxelbev\_cam} = \begin{bmatrix} 0 & -1 & 0 & x_{max} \\ 1 & 0 & 0 & y_{max} \\ 0 & 0 & -1 & z_{max} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4-3}$$

Where x_{max}, y_{max}, and z_{max} correspond to the maximum object distances from the camera origin in the camera coordinate system, thus defining the origin of the voxel coordinate system.

In the voxel coordinate system, all of the points of the cropped point cloud have a positive sign. Additionally, longitudinal and lateral directions are stored in the first and the second dimensions of the point cloud, respectively.

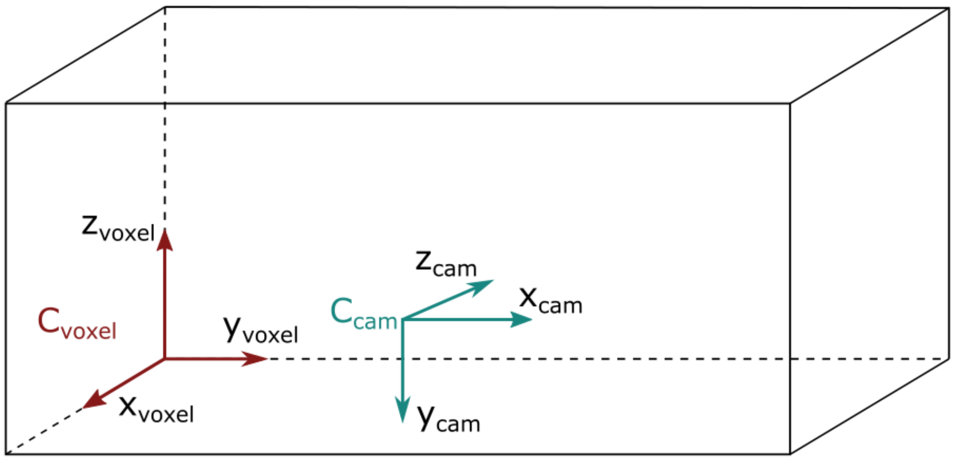


Figure 4-5: Camera and voxel coordinate systems placement.

4-3-2 Feature extraction from the point cloud

Point cloud is processed with the Feature Learning Network (FLN) adopted from VoxelNet. However, the original architecture of the network (see Figure 4-4) is adapted. In particular, the fully connected neural network after the last VFE layer is omitted and the voxel-wise feature is obtained directly from applying an element-wise max pooling on the output from the last VFE layer.

The processing layers from the raw point cloud data up to the extracted point cloud features are detailed in Table 4-1.

Table 4-1: Convolutional layers for point cloud feature extraction. The voxel-wise input obtained from stacked VFE layers is processed with 3D convolutional layers and then reshaped to obtain the final feature map from the point cloud. Stride and padding are defined in each of the three spatial dimensions which correspond to x, y, and z dimensions in the voxel coordinate system.

	layer type	kernel	stride	padding	output size
voxel-wise feature input	input	n/a	n/a	n/a	$350 \times 400 \times 10 \times 128$
	3D convolution	$3 \times 3 \times 3$	$(1, 1, 2)$	$(1, 1, 1)$	$350 \times 400 \times 5 \times 64$
	3D convolution	$3 \times 3 \times 3$	$(1, 1, 1)$	$(1, 1, 0)$	$350 \times 400 \times 3 \times 64$
	3D convolution	$3 \times 3 \times 3$	$(1, 1, 2)$	$(1, 1, 1)$	$350 \times 400 \times 2 \times 64$
feature map	reshape	n/a	n/a	n/a	$350 \times 400 \times 128$
feature map for RPN	2D convolution	1×1	1	valid	$350 \times 400 \times 1$

4-3-3 Feature extraction from the image

In the proposed method, feature maps from images are obtained with an extractor similar to the one from AVOD (see Section 4-1), which takes the RGB image as an input and consists of an adapted version of VGG16. In particular, the number of channels in each layer of the extractor from AVOD is reduced by half. This allows feature maps obtained from both the image and the point cloud to have an equal number of output channels.

The processing layers from the RGB image up to the extracted image features are detailed in Table 4-2.

Table 4-2: Image feature extractor. Image is processed by a series of 2D convolutional and max pooling layers. Finally, the features are upsampled and reduced by a 1×1 convolution.

	# of layers	layer type	kernel	stride	padding	output size
image input	1	input	n/a	n/a	n/a	$480 \times 1590 \times 3$
feature extractor	2	2D convolution	3×3	1	valid	$480 \times 1590 \times 16$
	1	max pooling	2×2	2	valid	$240 \times 795 \times 16$
	2	2D convolution	3×3	1	valid	$240 \times 795 \times 32$
	1	max pooling	2×2	2	valid	$120 \times 397 \times 32$
	3	2D convolution	3×3	1	valid	$120 \times 397 \times 64$
	1	max pooling	2×2	2	valid	$60 \times 198 \times 64$
	3	2D convolution	3×3	1	valid	$60 \times 198 \times 128$
feature map	1	upsampling	n/a	n/a	n/a	$240 \times 795 \times 128$
feature map for RPN	1	2D convolution	1×1	1	valid	$240 \times 795 \times 1$

4-3-4 Feature proposals

As in AVOD, a Region Proposal Network (RPN) which projects the proposals into the two input views. For camera images, the 3d proposals are projected into the camera image by using the pinhole camera model projection. For the point cloud input, the 3d proposals are transformed into the voxel coordinate system.

4-3-5 Dimensionality reduction and resizing

After feature extraction of the individual sensor input by VGG16 for the image input, and FLN for the point cloud input, the activations of the last feature extraction layers are reduced in dimensionality by using 1×1 convolutions. This operation helps to reduce the memory demand in the RPN stage and increase the speed of the network, as there are usually dozens of thousands of anchors to be processed there [23]. An example of the reduced feature map with the corresponding image is pictured in Figure 4-6.

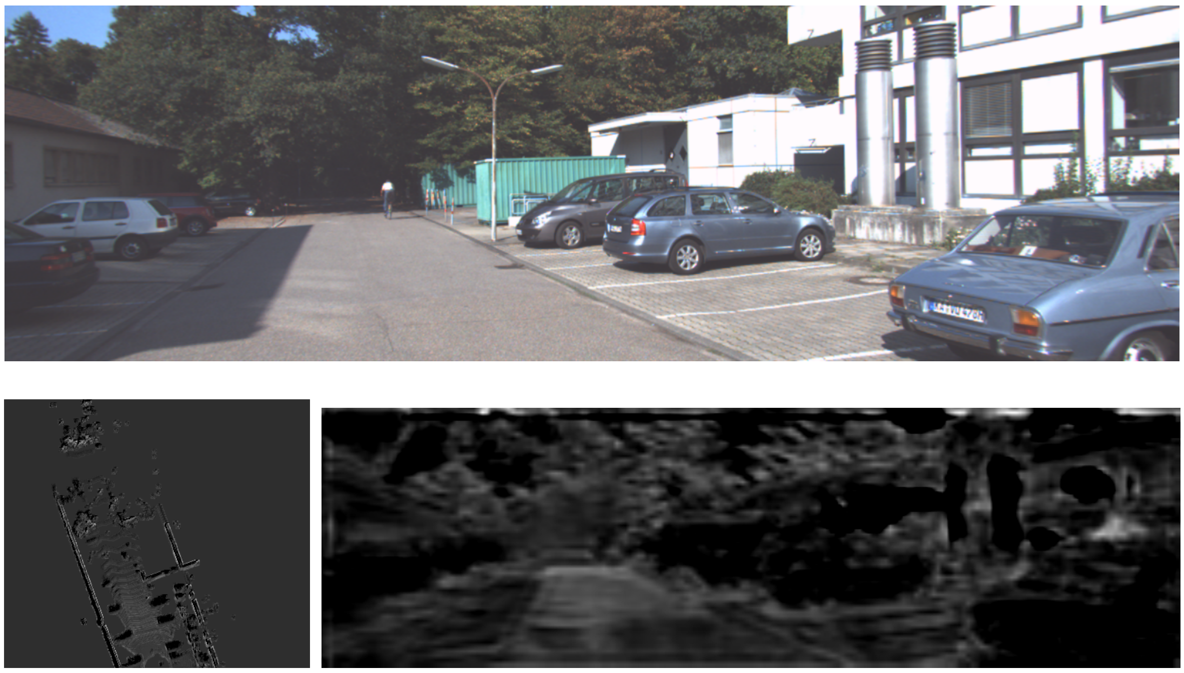


Figure 4-6: An example input image (top) and a corresponding point cloud feature map (bottom left) after 1×1 convolution and an image feature map (bottom right). The feature are maps obtained in the training process.

4-3-6 Deep fusion of image-based and point cloud-based features

Similarly to AVOD, the proposed architecture fuses resized crops from images and point cloud data via Feature Pyramid fusion (see Section 3-3-4). Fusion takes place in the RPN and in the second stage of the proposed architecture (see Figure 4-4).

For the purpose of evaluation, fusion in the second stage is performed in three different ways, which are called early, late, and deep fusion. In the early fusion features from image and point cloud are fused directly after cropping and resizing, before the very last fully connected detection network, see Section 4-3-7. In the late fusion, features are fused before the very last layer of the detection network. In the deep fusion features are fused in between all layers of the detection network. For a better overview of different ways of fusion, see Figure 4-7.

Moreover, two fusion types, namely *mean* and *concatenate*, are evaluated. *Mean* uses the element-wise mean activations of the resized crops from image and point cloud features. *Concatenate* fuses by concatenating the cropped and resized activations, thus forming a longer feature tensor.

4-3-7 Detection network

A detection network (Figure 4-7) is the very last stage of the whole architecture. It is based on three fully connected layers with 2048 neurons. For regularization purposes, dropout (Section 2-2-2) is applied with the probability of keeping the connection equal to 0.5. The feature crop from each proposal is processed to output box regression, orientation estimation and *objectness* estimation. As in AVOD, we employ a multi-task loss combining two Smooth L1 losses for the bounding box and orientation vector regression tasks and cross-entropy loss for the classification task.

Overlapping detections are removed by applying non-maximum suppression with a threshold of 0.01.

This results in the overall network architecture, as depicted in Figure 4-4.

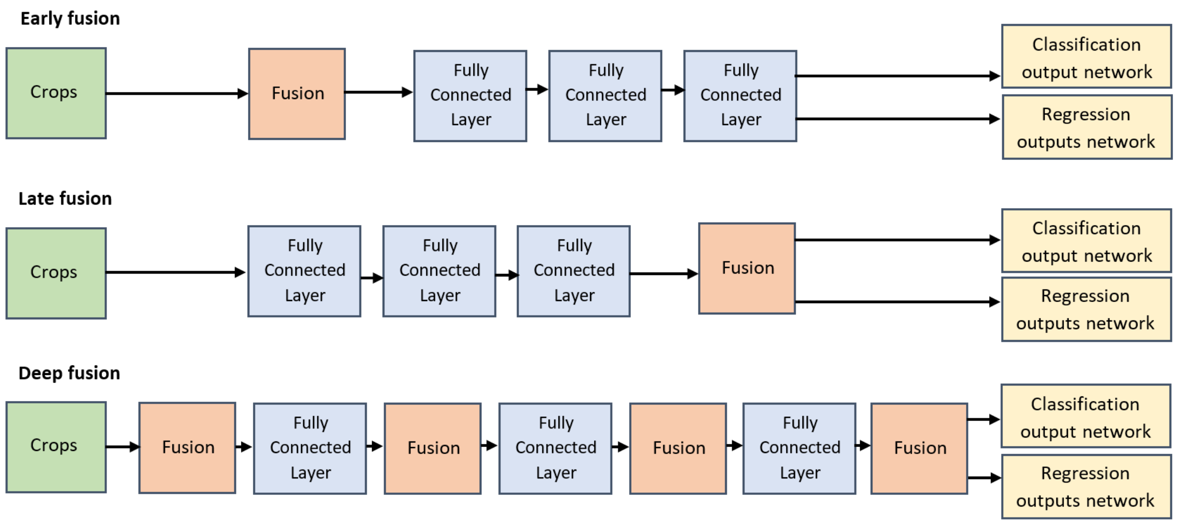


Figure 4-7: Fusion types for the proposed method. In the early fusion, the crops are first fused and then are fed into three fully connected layers. Later on, the output goes into classification and regression output networks. In the late fusion, three fully connected layers precede the fusion node. In the deep fusion, the fusion is performed both at the beginning and at the end, and also between the individual fully connected layers.

Chapter 5

Experimental results

The proposed method is evaluated quantitatively on the cars subset of the KITTI dataset, which will be presented in more detail in Section 5-1. Results are compared to the baselines AVOD and Voxelnet.

The KITTI dataset is presented in Section 5-1. The model hyperparameters are evaluated in Section 5-4. Evaluation metrics for 3D objets detection are discussed on Section 5-2 The top-performing models are analyzed more thoroughly in Section 5-5. There, a quantitative comparison to the baselines is performed alongside qualitative results showing the model performance on the KITTI validation dataset. The last section displays qualitative results on the TU Delft dataset.

5-1 KITTI dataset

KITTI dataset [52] is a dataset built by the Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago. The data were collected with a car equipped with two color and two greyscale cameras, a Velodyne lidar, and a GPS/IMU unit (see Figure 5-1). Moreover, the creators provide an online benchmark suite for such tasks as stereo matching and optical flow estimation, 3D visual odometry/SLAM, and 2D, 3D, and BEV object detection. These acpects cause that the dataset is widely used in the field of autonomous driving related research. For the object detection, KITTI dataset provides RGB camera images and lidar point clouds together with 3D annotations.

5-1-1 Dataset description

Each object in the dataset is described with a number of parameters. Among them is information regarding the class of the object (eight available), truncation and occlusion, the location and dimensions of the 3D bounding box associated with an object, as well as its orientation. A detailed description of the parameters can be reviewed in Table 5-1.

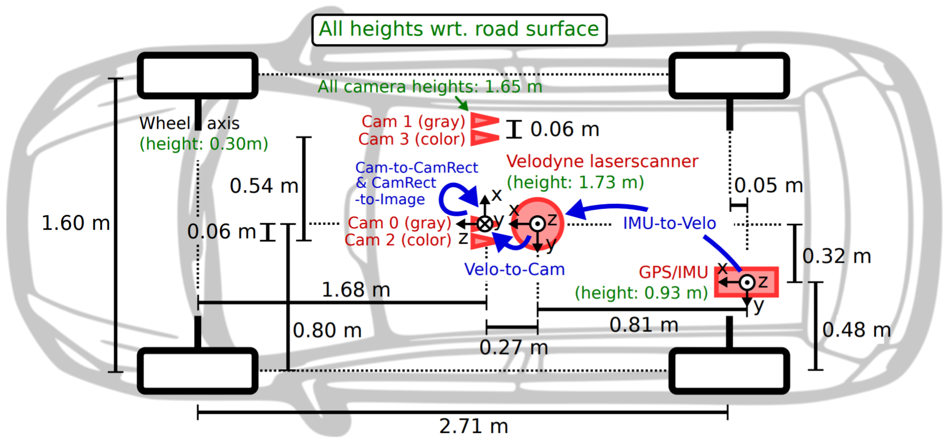


Figure 5-1: A scheme of the sensor placement in the car used to collect the KITTI dataset [7].

Table 5-1: Explanation of the annotations in the KITTI dataset.

No.	Contained information	Possible values
1	type of the object	'Car', 'Pedestrian', 'Van', 'Truck', 'Cyclist', 'Person_sitting', 'Tram', 'Misc', 'DontCare'
2	the information whether the object is 'truncated' or 'not-truncated', where 'truncated' means that the part of the object is outside of the image range	float value from 0.00 ('non-truncated') to 1.00 ('truncated'), the higher the value the smaller the visible part of the object
3	the information about the occlusion state of the object	'0' - fully visible '1' - partly occluded '2' - largely occluded '3' - unknown
4	alpha angle - the observation angle of the object	$[-\pi, \pi]$
5-8	2D bounding box of object in the image	left, top, right, bottom pixel coordinates
9-11	3D object dimensions	height, width, length (in meters)
12-14	3D object location	x,y,z in camera coordinates (in meters)
15	rotation ry around Y-axis	$[-\pi, \pi]$ (in camera coordinates)

'DontCare' label denotes that the objects in the region are not labeled (e.g. objects are outside of lidar range). This prevents the situation in which objects are incorrectly counted as false positives during evaluation. Objects detected in regions with 'DontCare' label will not be taken into consideration.

Moreover, the dataset items are divided into three groups with regard to its detection difficulty. The difficulty is higher if the object is far away and hence, its camera image has fewer pixels and the point cloud data is much sparser there. Furthermore, some of the objects are occluded or truncated, what also hinders the performance of the detection. Table 5-2 presents the conditions for each level.

Table 5-2: Conditions for three difficulty levels in the KITTI dataset [7]

Difficulty	min. bounding box height	max. occlusion level	max. truncation
Easy	40 pixels	fully visible	15%
Moderate	25 pixels	partly occluded	30%
Hard	25 pixels	difficult to see	50%

5-1-2 Dataset splits

In the 3D detection benchmark, there are 7481 training samples of RGB image + lidar point cloud provided. Together, the total number of labelled objects is 80256. Additionally, there are 7518 test samples. The labels for the test set are not publicly available and the only way to evaluate the performance on the test set is to upload the predictions to the online benchmark [1]. Therefore, MV3D [10] has provided a split of the training data that is commonly used by most methods. In particular, the training set is divided into a new training set with 3712 samples and a validation set with 3769 samples.

5-2 Evaluation metrics

An important metric used to evaluate the detection performance is the Intersection over Union (IoU). It measures the ratio between the overlapping area, i.e. intersection of a predicted and the ground truth box, and the joint area occupied by both boxes, i.e. the union. This is depicted in Figure 5-2. Analogously, the concept of IoU can be lifted to 3D, which is used in KITTI 3D object detection benchmark.

$$\text{IoU} = \frac{\text{area of overlap}}{\text{area of union}} \tag{5-1}$$

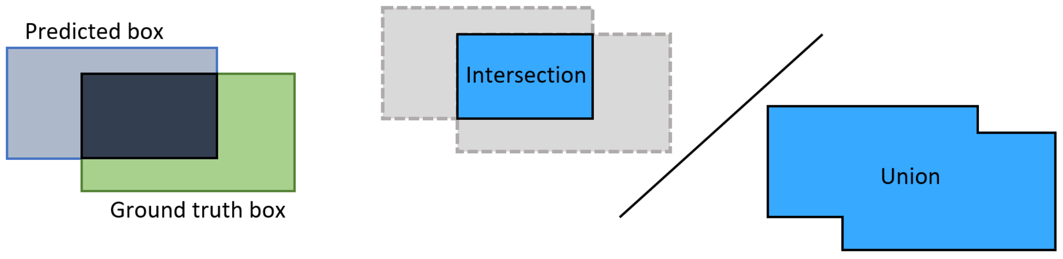


Figure 5-2: A visualization of the Intersection over Union.

In the context of object detection and KITTI benchmark a detected box is considered as a true positive (TP) if its Intersection over Union (IoU) with the ground truth is ≥ 0.7. A false positive (FP) is when the IoU between the detected box and the ground truth is < 0.7. A false negative (FN) denotes a case if there is an object but there is no box associated with it.

Precision is a measure expressing how many of selected items are actually relevant, i.e. the ratio between the true positives and all items that are predicted as positive:

$$\text{precision} = \frac{TP}{TP + FP} \tag{5-2}$$

Recall says how many of relevant items are selected, i.e. the ratio between the true positives and all ground truth positives:

$$\text{recall} = \frac{TP}{TP + FN} \tag{5-3}$$

Precision and recall, can be used to define a performance metric called an average precision (AP), first introduced in [53]. To compute the AP, eleven different confidence thresholds are used, for which the recall equals to $\{0, 0.1, \ldots, 1\}$. For a given recall value the maximum precision is used. The AP is then defined as the mean of the precision values at each of the eleven recall values:

$$AP = \frac{1}{11} \sum_{r \in \{0,0.1,\ldots,1\}} \max_{\tilde{r}:\tilde{r} \geq r} p(\tilde{r}) \tag{5-4}$$

where $p(\tilde{r})$ is measured precision for the recall value $\tilde{r}$. AP is the metric also used in the KITTI benchmark [7].

Another metric that can be defined is an average orientation similarity (AOS) [7], which is used to measure the performance of both object detection and 3D orientation estimation:

$$AOS = \frac{1}{11} \sum_{r \in \{0,0.1,\ldots,1\}} \max_{\tilde{r}:\tilde{r} \geq r} s(\tilde{r}) \tag{5-5}$$

where $s(\tilde{r})$ is the orientation similarity at recall $\tilde{r}$. For the definition of the orientation similarity refer to [7].

5-3 Data augmentation

Both methods presented in the related work section and the method proposed in this work are data-driven. Unfortunately, the amount of publicly available data with 3D annotations is limited. In order to overcome this issue, two different data augmentation techniques are used for the purpose of this project. Firstly, the data from images and point clouds is flipped horizontally. Secondly, the noise is added to the images by color perturbation. This results in a larger and slightly more diverse dataset which is beneficial for the training process. In terms of performance, it has been observed that applying data augmentation improves the average precision scores for the proposed method by around 3 percentage points.

5-4 Hyperparameters

The proposed architecture presents different hyperparameters, which largely affect the model's performance. Here, parameters that are being changed, in order to find the best performing setup, are briefly described.

- *RPN proposal RoI crop size* (RPN PRCS) - the size to which the crops from the reduced feature maps from camera and lidar are resized in RPN (see Section 4-1),
- *Second stage proposal RoI crop size* (SecS PRCS) - the size to which the crops from the full feature maps of the best proposals after RPN are resized,
- *RPN non-maximum suppression size* (RPN NMS) - the maximum number of proposals that can be kept after the non-maximum suppression step,
- *Initial learning rate* (IRL) - the initial value of the learning rate,
- *Decay steps* - the number of training iterations after which the learning rate changes (by the defined decay factor),
- *Decay factor* - the factor by which the learning rate is multiplied after particular number of iterations defined by decay steps,
- *Fusion method* - the way the fusion between the crops from an image and a lidar feature maps is performed, see Section 4-3-6,
- *Fusion type* - the fusion architecture to combine the crops from an image and a lidar feature maps, see Section 4-3-6.

Different experiments are performed by varying those hyperparameters (Section 5-5). The values for the hyperparameters used in the experiments are presented in Table 5-3.

Table 5-3: Hyperparameters used in the experiments and their values

Hyperparameters	**Values**
RPN proposal RoI crop size	$3 \times 3,\ 5 \times 5,\ 7 \times 7$
SecS proposal RoI crop size	$7 \times 7,\ 9 \times 9,\ 11 \times 11$
RPN NMS	300, 1024
initial learning rate	0.0001, 0.001
decay steps	30000, 100000
decay factor	0.1, 0.4, 0.8
fusion method	concatenate, mean
fusion type	late, early, deep

For evaluation on the KITTI dataset, the following parameters are fixated for the model: The transformation matrix from *camera coordinate system* to *voxel coordinate system* (see Section 4-3-1): $x_{max} = 70$ m, $y_{max} = 40$ m, and $z_{max} = 2.92$ m.

5-5 Evaluation

Multiple experiments are defined based on varying the hyperparameters introduced in Section 5-4. The complete list of the experiments can be found in Table B-1.

Training is performed on the training subset of KITTI dataset for which the ground truth annotations are publicly available. In order to be able to compare with other methods quantitatively, the training set is split into training and validation sets as described in Section 5-1-2.

At first, the influence of the initial learning rate (ILR), decay steps, decay factor, and RPN non-maximum suppression size (RPN NMS) on the performance is measured (Table 5-4, Table 5-5, and Table 5-6). One can see that for the experiments with the ILR equal to 0.001 the model achieve a very low performance regardless of the RPN and second stage (SecS) crop sizes. Another aspect that can be noticed is that in general, the higher the number of proposals kept after RPN, the higher the AP score for the proposed method. Thus, for all the other experiments (Table 5-7) values of 0.0001 and 1024 are used for ILR and RPN NMS, respectively. For the experiments with RPN NMS equal to 1024, the selected values of the decay steps and the decay factor do not have a very high influence on the performance. Decay factor is set to 0.8 and decay steps to 30000 for further experiments.

When comparing Table 5-4, Table 5-5, and Table 5-6, it can be observed that for the RPN and SecS crop sizes of 3×3 and 7×7, respectively, the AP score is in general lower than the score for larger crops. This can be seen especially for a *hard* difficulty Table 5-2. Due to that fact, only the larger crop sizes are being considered in the following experiments.

Table 5-7 describes experiments performed in order to analyze the influence of different fusion methods. It can be noticed, that for all the experiments the late fusion combined with concatenation of feature maps gave the best performance in terms of AP score. The most significant difference can be spotted between hard difficulty results.

After the evaluation of all the described experiments, the two best setups are chosen to be models from experiments 006 and 009. The training progress for these models, in terms of AP scores, are pictured in Figure 5-3 and Figure 5-4. The precision-recall plots for both models are depicted in Figure 5-5 and Figure 5-6.

Table 5-4: Experiments' results. Constant parameters: RPN proposal crop size: 3×3, SecS proposal RoI crop size: 7×7, fusion method: concatenate, fusion type: late.

EXP#	RPN NMS	ILR	decay steps	decay factor	AP (val)			best after # iter
					easy	moderate	hard	
000	300	0.0001	30000	0.8	76.21%	66.13%	64.83%	73000
001	1024	0.0001	100000	0.1	74.55%	64.98%	58.21%	44000
002	1024	0.0001	30000	0.8	76.71%	66.09%	58.97%	49000
003	300	0.001	30000	0.4	24.58%	21.21%	20.46%	9000

Table 5-5: Experiments' results. Constant parameters: RPN proposal crop size: 5×5, SecS proposal RoI crop size: 9×9, fusion method: concatenate, fusion type: late.

EXP#	RPN NMS	ILR	decay steps	decay factor	AP (val)			best after # iter
					easy	moderate	hard	
004	300	0.0001	30000	0.8	75.98%	66.70%	64.95%	46000
005	1024	0.0001	100000	0.1	76.24%	66.05%	64.29%	33000
006	1024	0.0001	30000	0.8	**81.38%**	67.02%	65.30%	51000
007	300	0.001	30000	0.4	71.01%	61.40%	56.14%	154000

Table 5-6: Experiments' results. Constant parameters: RPN proposal crop size: 7×7, SecS proposal RoI crop size: 11×11, fusion method: concatenate, fusion type: late.

EXP#	RPN NMS	ILR	decay steps	decay factor	AP (val)			best after # iter
					easy	moderate	hard	
008	300	0.0001	30000	0.8	76.52%	66.75%	65.23%	52000
009	1024	0.0001	100000	0.1	76.96%	**67.36%**	**65.94%**	75000
010	1024	0.0001	30000	0.8	76.32%	66.85%	65.23%	51000
011	300	0.001	30000	0.4	no conv	no conv	no conv	no conv

Table 5-7: Experiments' results. Constant parameters: RPN NMS: 1024, ILR: 0.0001, decay steps: 30000, decay factor: 0.8.

EXP#	RPN PRCS	SecS PRCS	fusion method	fusion type	AP (val)			best after # iter
					easy	moderate	hard	
006	5×5	9×9	concat	late	**81.38%**	**67.02%**	**65.30%**	51000
014				early	75.30%	65.48%	58.48%	48000
016				deep	74.32%	64.86%	64.15%	41000
018			mean	late	76.30%	66.68%	65.03%	37000
020				early	75.28%	65.24%	58.09%	34000
022				deep	76.47%	66.08%	58.79%	77000
010	7×7	11×11	concat	late	76.32%	66.85%	65.23%	51000
015				early	74.64%	65.23%	58.42%	83000
017				deep	75.30%	64.99%	58.44%	32000
019			mean	late	76.79%	66.82%	65.10%	49000
021				early	75.74%	65.56%	58.74%	66000
023				deep	76.76%	66.50%	59.12%	71000

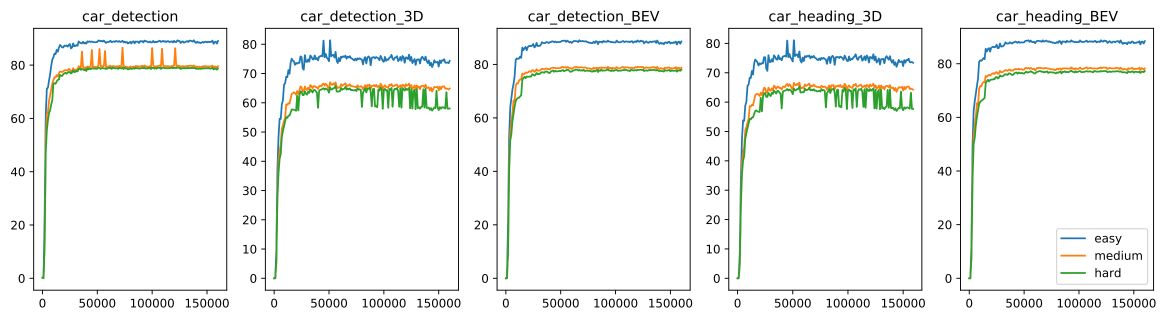


Figure 5-3: Experiment 006: Plots of average precision on the KITTI validation dataset over training iterations. From left to right: image-projection based, 3D detection based, bird eye view (BEV) based, car heading (yaw), car heading on BEV)

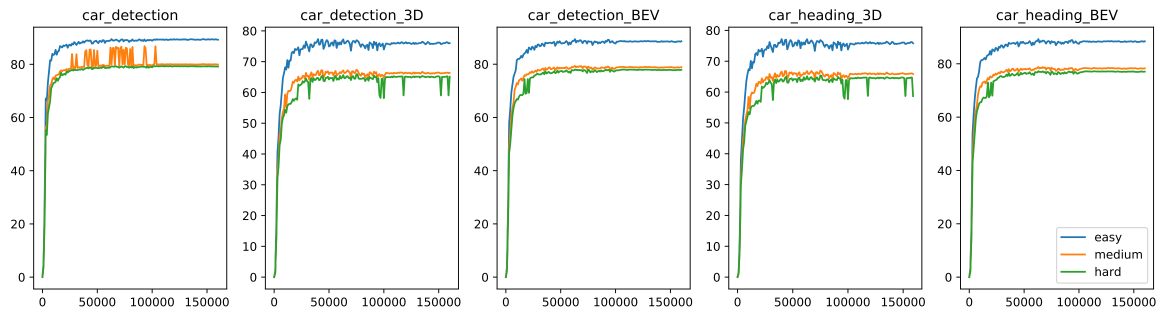
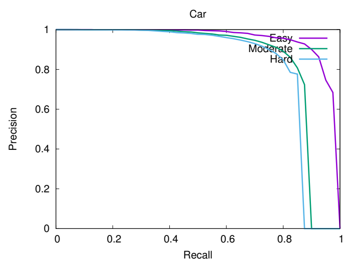
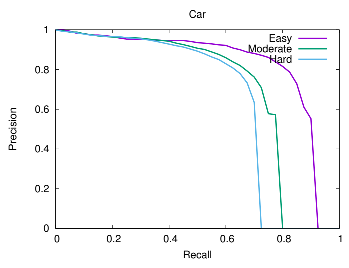


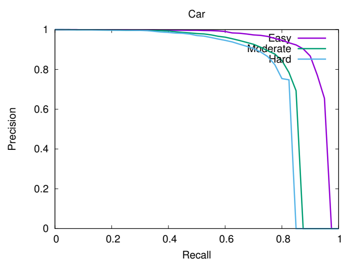
Figure 5-4: Experiment 009: Plots of average precision on the KITTI validation dataset over training iterations. From left to right: image-projection based, 3D detection based, bird eye view (BEV) based, car heading (yaw), car heading on BEV)



(a) 2d detection



(b) 3d detection



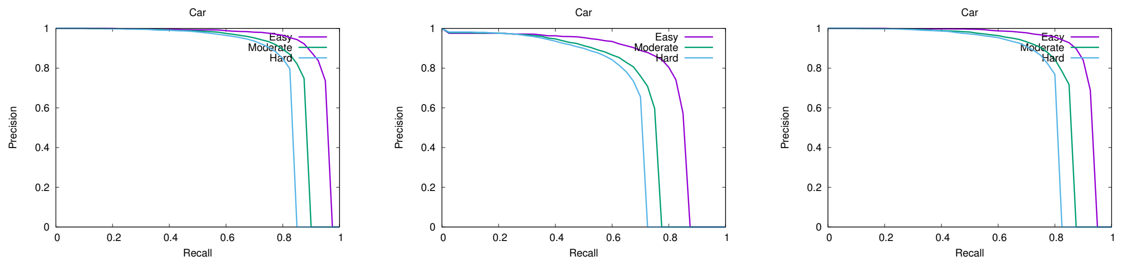
(c) Bird eye view

Figure 5-5: Precision-recall curves for the 006 model for the case of 2D, 3D, and BEV detection

5-6 Qualitative results

Qualitative results for the models of experiments 006 and 009 are depicted in Figure 5-7 and Figure 5-8. The scenes are randomly selected from the KITTI validation dataset, which is disjoint from the dataset used for training the models (see Section 5-1-2).

The models are able to predict cars with various orientations. They can handle some of the cases in which cars are partly occluded. Both models struggle with detecting cars that are far-away and create false positives for bushes, guard rails, or trash cans which occupy the



(a) 2d detection **(b)** 3d detection **(c)** Bird eye view

Figure 5-6: Precision-recall curves for the 009 model for the case of 2D, 3D, and BEV detection

volume similar to the size of a car.

5-7 Comparison

The models' performance on validation set is compared to three other methods, namely MV3D [10], AVOD [23], and VoxelNet [9]. The results are depicted in Table 5-8. The proposed architecture for 3D object detection achieves the AP score comparable to the other state-of-the-art methods.

Table 5-8: Comparison of results on the validation set for a car class. The proposed architecture achieves results comparable to the presented state-of-the-art methods.

Method	AP (val)		
	easy	moderate	hard
VoxelNet [9]	81.97%	65.46%	62.85%
AVOD [23]	83.45%	74.19%	67.78%
MV3D [10]	71.29%	62.68%	56.56%
SECOND [44]	**88.84%**	**78.43%**	**76.88%**
RT3D [45]	72.85%	61.64%	64.38%
Mono3D [38]	2.53%	2.31%	2.31%
3DOP [39]	6.55%	5.07%	4.10%
Proposed 006	81.38%	67.02%	65.30%
Proposed 009	76.96%	67.36%	65.94%

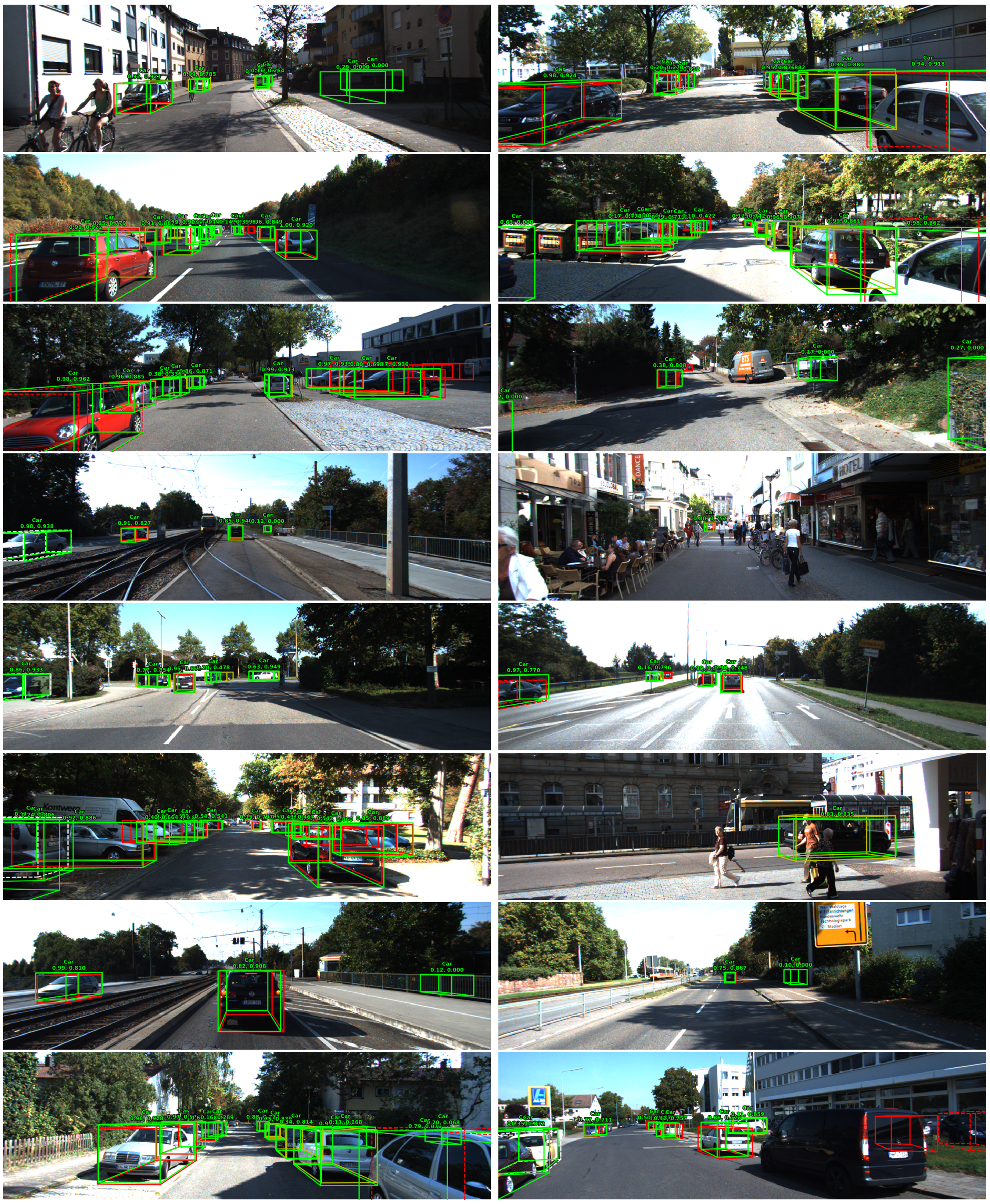


Figure 5-7: Qualitative results for the 006 model on the KITTI validation set. The model can predict cars with different orientations. Occlusions are also partly handled. The model misses far-away objects and creates false positives for guard rails and trash cans which occupy approximately the volume of a car. Ground truth: red. Predictions: green. Numbers on top of boxes: Detector objectness, IoU.

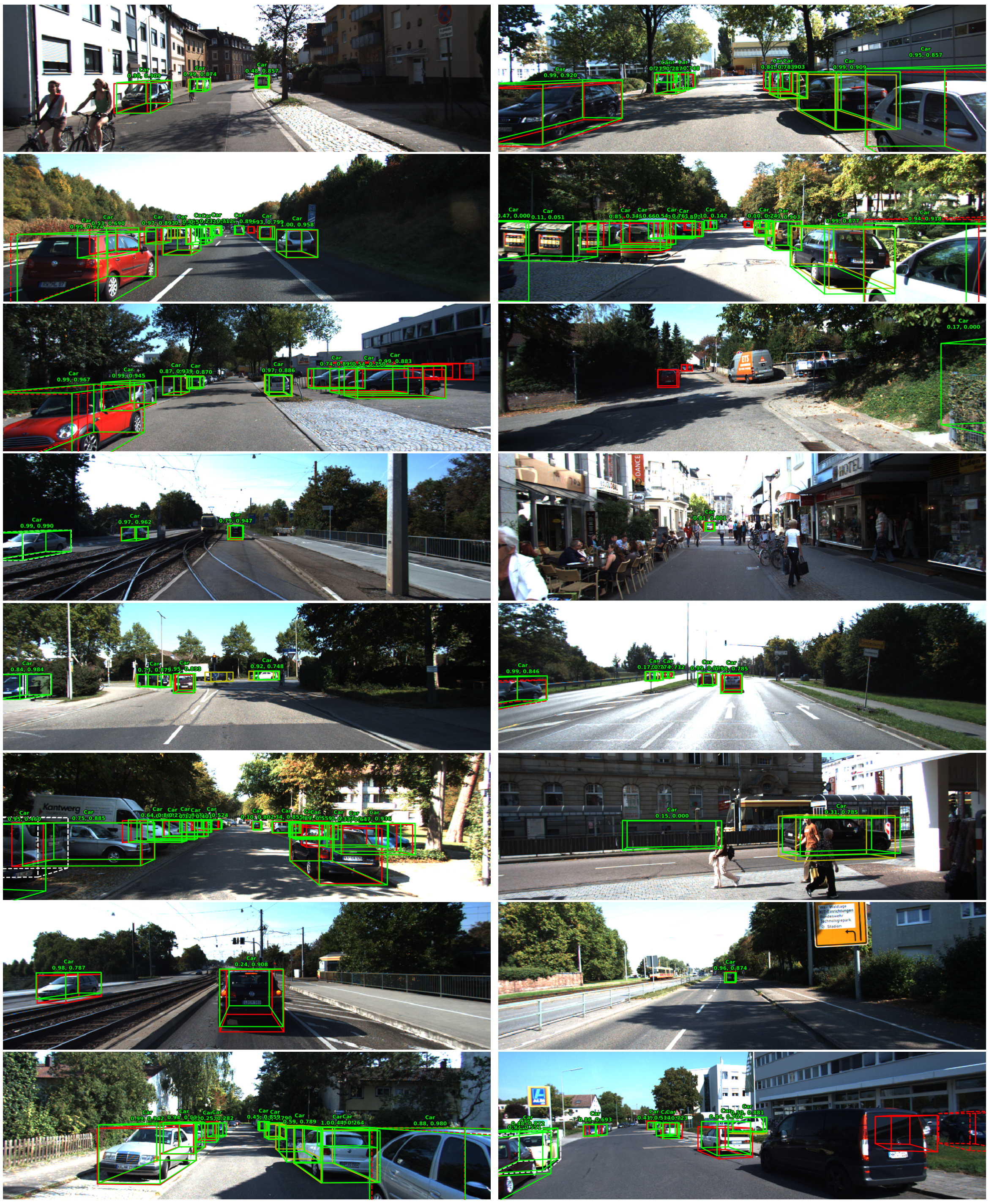


Figure 5-8: Qualitative results for the 009 model on the KITTI validation set. The model can predict cars with different orientations. Occlusions are also partly handled. The model misses far-away objects and creates false positives for guard rails and trash cans which occupy approximately the volume of a car. Ground truth: red. Predictions: green. Numbers on top of boxes: Detector objectness, IoU.

Chapter 6

Conclusions

In Section 1-1 several goals for this thesis were set. The most important objective was to: *Develop a new end-to-end deep learning based method for 3D object detection in the context of autonomous driving using camera and lidar data, with a focus on car detection.*

In this thesis a deep neural network for 3D object detection in the context of autonomous driving was presented. The network is trained in an end-to-end manner. The network takes as inputs data from two different sensors, namely an image from a camera and a point cloud from a lidar. The output of the network is an oriented 3D bounding box and the class of the detected object. The network was evaluated on the KITTI dataset [52] on the *car* class.

The most important aspect of the proposed architecture is that the network does not use any kind of hand-crafted features. Instead, it learns features from the raw sensor inputs. This approach achieves a performance comparable to the state-of-the-art methods, see Table 5-8.

Although the results of the presented method are similar to other state-of-the-art methods, the network has some limitations. The obtained inference time for one scene equals 200 ms which has limited applications in real-time systems.

6-1 Future work and recommendations

In order to achieve better results, several steps could be taken. First of all, more settings of the network's hyperparameters could be investigated. Secondly, for now, during the training the network handles only one scene per one forward and backward pass. To improve the performance the batch size could be increased, such that weights are updated based on more data, i.e. higher number of scenes and objects. Also, other regularization techniques such as skip links could be introduced for the network to be able to generalize better.

The presented approach was tested only on the *car* class of the KITTI dataset [52]. Nevertheless, the dataset provides the information for other classes as well, such as *pedestrians* or *cyclists*. Hence, the network could be also trained to detect objects of other types. The important remark is that the KITTI dataset has limited number of scenes and objects. Collecting more labelled data could significantly increase the performance of the method introduced in this work.

The highest amount of time during the inference is required for 3D convolutions. Thus, sparse convolutions could be implemented similarly to what is done in [44] in order to speed up the computations. Another recommendation would be to put a constraint on the maximum number of points in each point cloud, as the number of points in various point clouds highly differs.

Appendix A

KITTI benchmark results of published methods

Table A-1: 3D Object Detection KITTI Benchmark results [1] for the best performing methods. Values in bold are best results in particular category.

			Car AP [%]			Pedestrians [%]			Cyclists [%]		
Method	Data	Runtime	Moderate	Easy	Hard	Moderate	Easy	Hard	Moderate	Easy	Hard
AVOD-FPN [23]	pcl+im	0.10 s	71.88	81.94	66.38	42.81	50.80	40.88	52.18	64.00	46.61
F-PointNet [49]	pcl+im	0.17 s	70.39	81.20	62.19	44.89	51.21	40.23	56.77	71.96	50.39
VoxelNet [9]	pcl	0.23 s	65.11	77.47	57.73	33.69	39.48	31.51	48.36	61.22	44.37
MV3D [10]	pcl+im	0.36 s	62.35	71.09	55.12	N/A	N/A	N/A	N/A	N/A	N/A
F-PC_CNN [48]	pcl+im	0.50 s	48.07	60.06	45.22	N/A	N/A	N/A	N/A	N/A	N/A
LMNet [11]	pcl	0.02 s	15.24	14.75	12.85	11.46	13.64	11.57	3.23	2.84	3.28
BirdNet [54]	pcl	0.11 s	13.44	14.75	12.85	11.80	14.31	10.55	12.43	18.35	11.88
SECOND [44]	pcl	0.05 s	73.66	83.13	66.20	42.56	51.07	37.29	53.85	70.51	46.90
RoarNet [50]	pcl+im	0.1 s	73.04	83.71	59.16	N/A	N/A	N/A	N/A	N/A	N/A
UberATG-ContFuse [24]	pcl+im	0.06 s	66.22	82.54	64.04	N/A	N/A	N/A	N/A	N/A	N/A
RT3D [45]	pcl	0.09 s	21.27	23.49	19.81	N/A	N/A	N/A	N/A	N/A	N/A
TopNet-HighRes [55]	pcl	101 ms	12.58	15.29	12.25	9.66	13.45	9.64	5.98	4.48	6.18

Appendix B

All results

In this appendix, in Table B-1 all of the performed experiments are listed together with all of the varying training parameters.

Table B-1: The list of the performed experiments together with all varying training parameters.

EXP #	RPN PRCS	SecS PRCS	RPN NMS	ILR	decay steps	decay factor	fusion method	fusion type	AP (val)			best after # iter
									easy	moderate	hard	
000	3×3	7×7	300	0.0001	30000	0.8	concat	late	76.21%	66.13%	64.83%	73000
001	3×3	7×7	1024	0.0001	100000	0.1	concat	late	74.55%	64.98%	58.21	44000
002	3×3	7×7	1024	0.0001	30000	0.8	concat	late	76.71%	66.09%	58.97%	49000
003	3×3	7×7	300	0.001	30000	0.4	concat	late	24.58%	21.21%	20.46 %	9000
004	5×5	9×9	300	0.0001	30000	0.8	concat	late	75.98%	66.70%	64.95%	46000
005	5×5	9×9	1024	0.0001	100000	0.1	concat	late	76.24%	66.05%	64.29%	33000
006	5×5	9×9	1024	0.0001	30000	0.8	concat	late	**81.38%**	67.02%	65.30%	51000
007	5×5	9×9	300	0.001	30000	0.4	concat	late	71.01%	61.40%	56.14%	154000
008	7×7	11×11	300	0.0001	30000	0.8	concat	late	76.52%	66.75%	65.23%	52000
009	7×7	11×11	1024	0.0001	100000	0.1	concat	late	76.96%	**67.36%**	**65.94%**	75000
010	7×7	11×11	1024	0.0001	30000	0.8	concat	late	76.32%	66.85%	65.23%	51000
011	7×7	11×11	300	0.001	30000	0.4	concat	late	no conv	no conv	no conv	no conv
012	3×3	7×7	300	0.0001	30000	0.8	concat	deep	74.91%	65.04%	58.48%	55000
014	5×5	9×9	1024	0.0001	30000	0.8	concat	early	75.30%	65.48%	58.48%	48000
015	7×7	11×11	1024	0.0001	30000	0.8	concat	early	74.634%	65.23%	58.42%	83000
016	5×5	9×9	1024	0.0001	30000	0.8	concat	deep	74.32%	64.86%	64.15%	41000
017	7×7	11×11	1024	0.0001	30000	0.8	concat	deep	75.30%	64.99%	58.44%	32000
018	5×5	9×9	1024	0.0001	30000	0.8	mean	late	76.30%	66.68%	65.03%	37000
019	7×7	11×11	1024	0.0001	30000	0.8	mean	late	76.79%	66.82%	65.10%	49000
020	5×5	9×9	1024	0.0001	30000	0.8	mean	early	75.28%	65.24%	58.09%	34000
021	7×7	11×11	1024	0.0001	30000	0.8	mean	early	75.74%	65.56%	58.74%	66000
022	5×5	9×9	1024	0.0001	30000	0.8	mean	deep	76.47%	66.08%	58.78%	77000
023	7×7	11×11	1024	0.0001	30000	0.8	mean	deep	76.76%	66.50%	59.12%	71000

Bibliography

[1] Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago, "3d object detection evaluation 2017." http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d. Retrieved 25 May 2018. i, vii, 19, 27, 29, 39, 52

[2] "ITS & vulnerable road users." https://ec.europa.eu/transport/themes/its/road/action_plan/its_and_vulnerable_road_users_en. Retrieved on 01 December 2018. 1

[3] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015. 2, 17

[4] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. 2, 17

[5] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. 2, 17, 20, 23, 29, 30

[6] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015. v, 2, 17, 18, 22, 23, 29, 30

[7] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3354–3361, June 2012. v, vi, vii, 2, 6, 29, 38, 39, 40

[8] A. Marshall, "Waymo launches its self-driving armada." https://www.wired.com/story/waymo-launches-self-driving-minivans-fiat-chrysler/. Retrieved 15 December 2018. v, 2

[9] Y. Zhou and O. Tuzel, "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection," *arXiv*, 2017. vi, 7, 21, 27, 29, 30, 31, 45, 52

[10] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *CVPR*, 2017. v, 7, 14, 19, 22, 23, 24, 39, 45, 52

[11] K. Minemura, H. Liau, A. Monrroy, and S. Kato, "LMNet: Real-time Multiclass Object Detection on CPU using 3D LiDAR," may 2018. 7, 21, 52

[12] "CS231n Convolutional Neural Networks for Visual Recognition." http://cs231n.github.io/neural-networks-1/. Retrieved on 01 December 2018. 8

[13] N. Yadav, A. Yadav, and M. Kumar, *An Introduction to Neural Network Methods for Differential Equations*, ch. 2, pp. 13–15. Springer, 2015. 7

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. 9, 10, 11

[15] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *ArXiv e-prints*, mar 2016. 11

[16] M. Lin, Q. Chen, and S. Yan, "Network in network," *CoRR*, vol. abs/1312.4400, 2013. 12

[17] J. Thomanek and G. Wanielik, "A new pixel-based fusion framework to enhance object detection in automotive applications," in *17th International Conference on Information Fusion (FUSION)*, pp. 1–8, July 2014. v, 12, 13

[18] G. Zhao, X. Xiao, J. Yuan, and G. W. Ng, "Fusion of 3d-lidar and camera data for scene parsing," *Journal of Visual Communication and Image Representation*, vol. 25, no. 1, pp. 165 – 183, 2014. Visual Understanding and Applications with RGB-D Cameras. 13

[19] B. Khaleghi, A. Khamis, F. Karray, and S. Razavi, "Multisensor data fusion: A review of the state-of-the-art," *Information Fusion*, vol. 14, no. 1, pp. 28–44, 2013. 13

[20] S.-I. Oh and H.-B. Kang, "Object detection and classification by decision-level fusion for intelligent vehicle systems," *Sensors*, vol. 17, no. 1, 2017. v, 13, 14

[21] A. Sinha, H. Chen, D. G. Danu, T. Kirubarajan, and M. Farooq, "Estimation and decision fusion: A survey," *Neurocomputing*, vol. 71, pp. 2650–2656, Aug. 2008. 13

[22] J. Wang, Z. Wei, T. Zhang, and W. Zeng, "Deeply-fused nets," *CoRR*, vol. abs/1605.07716, 2016. 13, 14

[23] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. Waslander, "Joint 3D Proposal Generation and Object Detection from View Aggregation," 2017. v, vi, 14, 19, 22, 23, 27, 28, 31, 34, 45, 52

[24] M. Liang, B. Yang, S. Wang, and R. Urtasun, "Deep continuous fusion for multi-sensor 3d object detection," in *ECCV*, 2018. v, 14, 22, 23, 24, 52

[25] P. Viola and M. Jones, "Robust real-time object detection," in *International Journal of Computer Vision*, 2001. 15

[26] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 886–893 vol. 1, June 2005. v, 15, 16

[27] J. Kooij, "Slides from TU Delft course - Intelligent Vehicles (ME41105)," 2018. 16

[28] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, pp. 1627–1645, Sept 2010. v, 16

[29] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013. 17

[30] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: object detection via region-based fully convolutional networks," *CoRR*, vol. abs/1605.06409, 2016. 17

[31] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision – ECCV 2016* (B. Leibe, J. Matas, N. Sebe, and M. Welling, eds.), (Cham), pp. 21–37, Springer International Publishing, 2016. v, 17, 18

[32] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016. 18

[33] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. 18

[34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. 18, 23, 28

[35] S. Song and J. Xiao, "Deep Sliding Shapes for Amodal 3D Object Detection in RGB-D Images," 2015. 19

[36] F. Chabot, M. Chaouch, J. Rabarisoa, C. Teulière, and T. Chateau, "Deep MANTA: A coarse-to-fine many-task network for joint 2d and 3d vehicle analysis from monocular image," *CoRR*, vol. abs/1703.07570, 2017. 19, 20

[37] "Object detection and orientation estimation evaluation." `http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=2d`. Retrieved on 01 December 2018. 19, 20

[38] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun, "Monocular 3d Object Detection for Autonomous Driving," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2147–2156, June 2016. 20, 45

[39] X. Chen, K. Kundu, Y. Zhu, A. Berneshawi, H. Ma, S. Fidler, and R. Urtasun, "3d object proposals for accurate object class detection," in *NIPS*, 2015. 20, 45

[40] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka, "3d Bounding Box Estimation Using Deep Learning and Geometry," pp. 5632–5640, IEEE, July 2017. 20

[41] C. C. Pham and J. W. Jeon, "Robust object proposals re-ranking for object detection in autonomous driving using convolutional neural networks," *Signal Processing: Image Communiation*, 2017. 20

[42] D. Zeng Wang and I. Posner, "Voting for Voting in Online Point Cloud Object Detection," *Robotics: Science and Systems XI*, 2015. 21

[43] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner, "Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1355–1361, May 2017. 21

[44] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, 2018. 21, 45, 50, 52

[45] Y. Zeng, Y. Hu, S. Liu, J. Ye, Y. Han, X. Li, and N. Sun, "Rt3d: Real-time 3-d vehicle detection in lidar point cloud for autonomous driving," *IEEE Robotics and Automation Letters*, vol. 3, pp. 3434–3440, Oct 2018. 21, 22, 45, 52

[46] S. Wirges, T. Fischer, J. B. Frias, and C. Stiller, "Object detection and classification in occupancy grid maps using deep convolutional networks," *CoRR*, vol. abs/1805.08689, 2018. 21, 22

[47] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. 22

[48] X. Du, M. H. A. Jr., S. Karaman, and D. Rus, "A general pipeline for 3d detection of vehicles," *CoRR*, vol. abs/1803.00387, 2018. v, 22, 23, 52

[49] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum PointNets for 3D Object Detection from RGB-D Data," 2017. v, 22, 25, 52

[50] K. Shin, Y. Kwon, and M. Tomizuka, "Roarnet: A robust 3d object detection based on region approximation refinement," *arXiv preprint arXiv:1811.03818*, 2018. 22, 25, 52

[51] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," 2016. 25

[52] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013. 37, 49

[53] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2011 (VOC2011) Results." http://www.pascal-network.org/challenges/VOC/voc2011/workshop/index.html. 40

[54] J. Beltrán, C. Guindel, F. M. Moreno, D. Cruzado, F. García, and A. de la Escalera, "Birdnet: a 3d object detection framework from lidar information," *CoRR*, vol. abs/1805.01195, 2018. 52

[55] S. Wirges, T. Fischer, C. Stiller, and J. B. Frias, "Object detection and classification in occupancy grid maps using deep convolutional networks," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 3530–3535, Nov 2018. 52

Glossary

List of Acronyms

ADAS	Advanced Driver-Assistance Systems
ACC	Adaptive Cruise Control
SSD	Single Shot MultiBox Detector
YOLO	You Only Look Once
R-CNN	Region-based Convolutional Neural Network
RGB	Red Green Blue
HSV	Hue Saturation Value
BEV	Bird-Eye View
ANN	Artificial Neural Network
NN	Neural Network
DNN	Deep Neural Network
MSE	Mean Squared Error
MAE	Mean Absolute Error
ReLU	Rectified Linear Unit
CNN	Convolutional Neural Network
HOG	Histogram of Oriented Gradients
DPM	Deformable Part Models
SVM	Support Vector Machine
RFCN	Region-based Fully Convolutional Network

RPN	Region Proposal Network
ROI	Region of Interest
CAD	Computer Aided Design
VFE	Voxel Feature Encoding
FPS	Frames per second
MV3D	Multi-View 3D Object Detection Network
AVOD	Aggregate View Object Detection
TP	True Positive
FP	False Positive
FN	False Negative
TN	True Negative
EXP	Experiment
SecS	Second Stage (of the proposed network)
RPN NMS	RPN Non-maximum Suppression Size
ILR	Initial Learning Rate
AP	Average Precision
PRCS	Proposal ROI crop size
CPU	Central Processing Unit
GPU	Graphics Processing Unit