CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

3D Object Classification using Point Clouds and Deep Neural Network for Automotive Applications

Master's thesis in Computer Science and Engineering

Christian Larsson
Erik Norén

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Master's thesis 2019: DATX05

3D Object Classification using Point Clouds and Deep Neural Network for Automotive Applications

Christian Larsson
Erik Norén

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2019

3D Object Classification using Point Clouds and Deep Neural Network for Automotive Applications
Christian Larsson
Erik Norén

Supervisor: Knut Åkesson, Department of Electrical Engineering, Chalmers
Advisor: Gunnar Ragnarsson, QRTECH
Examiner: Elad Schiller, Department of Computer Science and Engineering, Chalmers

Master's Thesis 2019
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Point cloud of a traffic environment obtained by a LIDAR.

Typeset in LaTeX
Gothenburg, Sweden 2019

Christian Larsson
Erik Norén
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Object identification is a central part of autonomous cars and there are many sensors to help with this. One such sensor is the LIDAR which creates point clouds of the cars surrounding. This thesis evaluates a solution for object identification in 3D point clouds with the help of a neural network. A system named DELIS (DEtection in Lidar Systems), which takes a point cloud generated from a LIDAR as input, is designed. The system consists of two subsystems, one non-machine learning algorithm which segments the point cloud into clusters, one for each object, and a neural network that classifies this clusters. The final output is then the classes and the coordinates of the objects in the point cloud. The result of this thesis is a system named DELIS that can identify between pedestrians, cars, and cyclists.

Keywords: LIDAR, Neural Networks, Object Identification, Segmentation, Automotive applications.

Acknowledgements

We would like to thank all of our colleagues at QRTECH for having us there (the breakfast was awsome). A very special thank you to our advisor Gunnar Ragnarsson and supervisor Knut Åkesson. We would also like to thank our families and our friends, wherever they are, for their support. The training data in this project are supplied from KITTI, which we are grateful to have taken part of.

Christian Larsson and Erik Norén
Göteborg, Sweden, March, 2019

Dictionary

Adam - Adaptive Moment Estimation. An optimization algorithm that is used in different kinds of *ANN*.

ANN - Artificial Neural Networks. Computing system inspired by biological systems.

CNN - Convolutional Neural Network. A special kind of ANN, that utilizes convolution.

HDBSCAN - Hierarchical Density-Based Spatial Clustering of Applications with Noise. An algorithm designed to find stable clusters.

KITTI - Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago. A project between two technological institutions, that have captured a lot of data from traffic environments that is available for the academy.

LIDAR - Light Imaging, Detection, And Ranging. A sensor that is used to detect distances.

Contents

List of Figures

List of Tables

1
Introduction

Autonomous driving is something that most car manufacturers are working on these days and a few of them already have autonomous vehicles out driving for testing purposes.
Self-driving vehicles need to be able to navigate safely and to do this they need to be able to identify objects in the traffic correctly. To achieve this, different sensors and classification algorithms, which have different properties, may be used. To exemplify, when the light condition is bad, during nights, etc., sensors such as cameras have a hard time working since they use light from their surrounding. There are however sensors that are unaffected by the level of light in its surroundings. One such sensor is the LIDAR (Light Imaging, Detection, And Ranging) sensor. A LIDAR creates a point cloud which is a set of points representing the surroundings of the vehicle. This point cloud can then be used to detect and identify objects in the vicinity of the vehicle, which is needed to ensure safe navigation. In [2] it is claimed that one of the advantages of LIDAR is its independence of light from the environment. This means it works the same no matter the light conditions. An additional benefit is that the LIDAR is able to create a 3D map of the car's surroundings. This can be used by the car to make decisions about what to do next and plan a route accordingly. On the other hand, LIDAR is an expensive technology and is therefore not as widespread as cameras. Figure 1.1 shows a car with a LIDAR on its roof.
In order to classify different kinds of objects, various strategies have been developed. Artificial Neural networks, (ANN), is one of them. Researchers in [3] have developed networks that have, depending on input data and architecture, accuracy over 90%, when it comes to identification of different objects. That is 90% of the input data that are sent through the ANN is correctly classified. This is however for 2D-image from a standard RGB-camera, which is a well-researched area. Object classification with 3D data as input, on the other hand, is less researched. A LIDAR sensor has the advantage that it can provide a single complete view of the environment around

Figure 1.1: Car with a LIDAR mounted on its roof

the car. Although it is possible to use multiple cameras around the vehicle and fuse the different views together to achieve something similar this needs extra computer processing, whereas with the LIDAR you get the complete view immediately.

1.1 Purpose and Objective

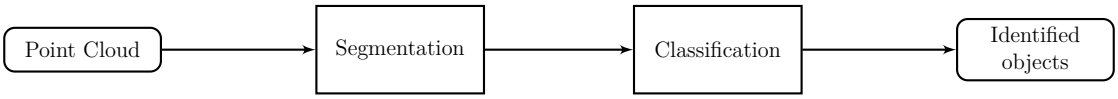


Figure 1.2: An overview of the system called DELIS (DEtection in LIDAR Systems). The boxes with soft corners are input and output and the boxes with hard edges are the parts of the system. Throughout the report, this figure illustrates which part of DELIS that is explained.

The purpose of this thesis is to examine if it is possible for a machine learning method to classify objects obtained from LIDAR data that have been segmented using non-machine learning methods.
This work covers object classification of PC obtained by a LIDAR. The data should come from real and simulated traffic environments. To be able to answer the Research Questions stated in chapter 1.3 DELIS, which be viewed in figure 1.2, will be created.
DELIS should be able to work in real time and high classification accuracy. This, since it is vital for applications like autonomous vehicles to quickly know if there might be a dangerous situation.
One possible way for a vehicle to safely navigate is to use a machine learning method to recognize different objects from the LIDAR data.
In this thesis, LIDAR data from the KITTI library [4] is used to train the ANN. The system is going to be evaluated using both the KITTI data and simulated data created with ESI Pro-SiVIC [5].
The created system, DELIS, is as a proof of concept for object detection in a 3D environment using non-machine learning methods. It may be viewed in Figure 1.2.

1.2 Related Works

One common method to classify objects in traffic environments is to use images obtained from cameras, an example of this is *You Only Look Once: Unified, Real-Time Object Detection* [6]. However, this method sacrifices accuracy for speed. This method uses a convolutional neural network (CNN) to both create the bounding boxes and to calculate the class probability.
One approach to classification of objects in point clouds was introduced in *VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection* [7]. In the paper, the researchers created voxels from point clouds and then ran them through an ANN.

A quicker way of obtaining a point cloud is to use a LIDAR. The paper in [8], gives an idea of how CNN could be used in order to directly identify a road using LIDAR-data as input. However the method used here does not include any identification of moving objects in a traffic environment such as cars and pedestrians, but it shows that it is possible to use LIDAR data in a CNN.
There is also proposed hybrid solutions such as *Multi-view Convolutional Neural Networks for 3D Shape Recognition*[9]. In that paper, the idea was to convert the 3D point clouds into 2D images and train a neural network to identify the objects from this images. One downside with this method is that the system loses the advantages gained from using 3D data in the first place. This includes tasks such as point classification, shape completion and scene understanding [10], [11], all which can be very useful for a car in order to understand its surroundings.
There have been some papers that deal with object identification directly from point clouds. For example, PointNet [10], where the researchers trained a neural network to be able to identify an object based on the raw data points collected. This sounds like a promising way to go, however, the data used was collected with a Matterport 3D camera, the camera takes 30 seconds to capture the depth and would thus be a poor choice to use as a sensor on vehicles [12].
The project differs from others in the use of LIDAR data as input when classifying an object with a neural network without using voxels. It also differs from other work by the usage of non-machine learning segmentation methods. The project further plans to include simulated data from a simulated traffic environment in order to see if this data gives the same result as real-world data. Simulated data is interesting because it is possible to generate a large amount of labeled data with minimal effort. Simulated data is however problematic since it is difficult to make it realistic, hence it often differs greatly from real-world data.

1.3 Research Questions

The research questions are as follows.

RQ1 How well does machine learning methods, trained on real traffic environments perform on simulated environments?
In order to make data acquisition more efficient one can use simulated data. However, the simulated data will not be available in the project until the evaluation phase of the project.
This question is answered by creating a system and using KITTI data for testing and training of the chosen machine learning method. First, when the system is working with the KITTI data, the simulated data will be used for testing and verification.

RQ2 Is it possible to use machine learning algorithms with non-machine learning algorithms in order to make a competitive system for object classification?
The common procedure nowadays is to use an end to end neural network in order to classify objects. However, this project will try to decrease the gap between classical algorithms and machine learning methods, by using them

together.
This will be answered when the evaluation of the system itself and its subsystems, segmentation algorithm and machine learning methods, are performed.

1.4 Summary of Contributions

This thesis proposes a method to identify objects directly from a LIDAR scan. The first step of two is to segment the point cloud into clusters, one per object. The second is the classification of the objects with the help of a neural network. The fact that the segmentation is done as its own algorithm and not in a neural network makes this approach unique. As far as we know, this has never been done before. This is opposed to doing the segmentation and classification in one neural network, as is done in [8]. One other unique aspect is in the usage of both real-world and simulated data. The simulated data shall be used to verify the network that is trained with real-world data. Briefly before this project started one neural network called PointNet were published, which is the neural network which DELIS is built upon.
The evaluation of the segmentation was done by counting the points in the segmented clusters and how many of them that were correctly labeled with the help of the ground truth data. For the segmentation, the cluster creation time is measured in order to obtain the performance time. This is relevant information for real-time system applications. Among other things, it was concluded that the HDBSCAN, see chapter 2.2.3, were faster than the Nearest Neighbour algorithm, although neither is fast enough for a real-time application. Also, by counting the number of points in the clusters that have the correct label and compare that number with the number of points that are in the cluster, the accuracy for the segmentation is obtained. The accuracy for the segmentation varied between different scenes and HDBSCAN showed better performance overall.
As for the classification, the accuracy is obtained by dividing the number of correctly classified objects with the total number of objects. It was apparent that the transform nets used in PointNet were of great use when it comes to classifying objects detected in point clouds. The validation accuracy of the network where around 95% when using the training dataset.
The system itself is evaluated by manually inspecting the scene and counting the number of identified objects of each relevant class and divide it by the total number of objects in that class. Thus the system's accuracy is obtained. The final system where able to identify a few items depending on the scene, still the system has much room for improvement. Our conclusion is that the segmentation part of the system has the most to improve.

1.5 Scope and Boundaries

The types of objects to identify has been restricted to cars, cyclists, and pedestrians. This is to prove that the suggested method works for multiple sets of classes.

The three object limitation is also to simplify the data collection since these objects are some of the most common objects in a traffic environment and therefore the easiest to get data off. By focusing on three classes the dataset can build with more data points of this classes, which will result in a more accurate identification.
The problems with identifying objects from LIDAR data is that the objects are only scanned from one angle, which makes the input 2.5D and, the further away the object is from the sensor, the fewer points are able to be found. As a result, the classification requires a lot of training data in order to perform well.
The ANN being designed is based on a network called *PointNet*, [10], which is designed to classify computer generated objects, and thus on 3D objects. The network is trained on PC that contains 128 points, which is half as many as *PointNet* was designed to read as a minimum. The limiting factor was the relatively few points in the objects from the KITTI ground truth that was labeled as pedestrians. A higher number of points makes it easier for the network to recognize an object, but this would mean fewer objects to train on.
The system needs to separate objects from the point cloud to be able to identify them, therefore some kind of segmentation is needed. No new segmentation methods will be developed instead already existing algorithms will be implemented in the system. Furthermore, the system will be designed to deal with a flat landscape, thus no slopes or speed bumps are to be included in the project.

2

Theory

In order to get a better view of this thesis, this chapter will provide the theory necessary to get an understanding of the project. It will first give an understanding of how to create point clouds and then it will provide a description of tools that were used. The last two parts concern segmentation and classification, which are the two most important parts of the theory. Figure 2.1 gives a picture of the system that will be created.

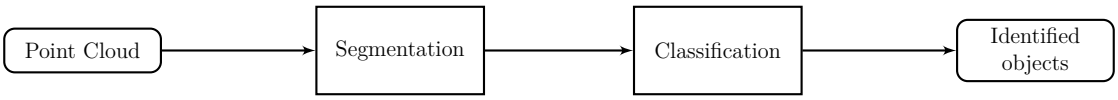


Figure 2.1: The figure illustrates the system broken down to its main components and how the data flows inside DELIS. This is also how the theory will be covered.

2.1 Generation of Point Clouds

A point cloud is a unified and simple structure that has none of the complexities and combinatorial irregularities that meshes normally have, [10]. One way to view point clouds is as points of data in a space. According to Mathworks [13], a point cloud is normally used to measure distances in the physical world. Normally a 3D Cartesian coordinate system is used to visualize the point cloud, which could be a map or an object [14]. Its application may be everything between, robot vision and boat building, [15], to medical applications like 3D tomographic reconstruction [16]. Point Clouds are also used in order to prevent robots in factories from colliding. An example of a point cloud can be seen in Figure 2.2.

Organized vs Unorganized

A point cloud can be either *organized* or *unorganized*. When plotting them, both kinds of point clouds look like one other. The difference is how they are stored. According to [17], an *organized* point cloud has an organized structure resembling an image where the data is sorted into columns and rows, e.g. $n \times m$. Such data most often comes from stereo cameras. In knowing the relationship between points in an organized point cloud the computational cost of certain algorithms can be lowered, for example, the nearest neighbor algorithm. *Unorganized* point clouds are on the other hand without structure, they are stored as a $1 \times M$ matrix. The only thing you know is the number of points in the point cloud.

Figure 2.2: Point Cloud of a Hare.

2.1.1 LIDAR

Normally cameras are used to identify objects in an automotive application, but they have some limitations, both NASA and Chalmers University of Technology, [2, 8], agrees on that cameras are very dependent on the surrounding light, which makes it difficult for cameras to work in poor light conditions. The LIDAR, on the other hand, does not have this problem which makes it an excellent tool in such conditions. However, these sensors are expensive and often breaks, since there are a lot of moving parts in them.
The LIDAR works by emitting a laser beam at a known direction. At the same time, a clock is activated and measure the time it takes for a detector to detect the laser, which has bounced on an object. The distance for the object is then obtained through equation 2.1.

$$d = c\frac{t}{2} \tag{2.1}$$

Where d is the distance, c is the speed of light and t is the measured time. It is divided by two since it is half the time it takes for the laser to be emitted and detected which is the time it takes from the emitter to the object. This is then done in every different direction in order to get a full scene.
In [18] it is claimed that there are different kinds of LIDARs, which have different applications. A *coherent LIDAR* is one example, it measures the velocity of an object. The *range LIDAR* is one other example of a LIDAR. It measures the time it takes for light to travel from the LIDARs emitter to an object and then back to the sensor, from this the distance to the object may be obtained a diagram for the vital parts can be seen in Figure 2.3.
To solve the cost and reliability issues with the LIDAR, a new kind of LIDAR called *Solid State LIDAR* have been developed. These new type of LIDAR have a more limited field of view but have the advantage of being cheaper to produce and they

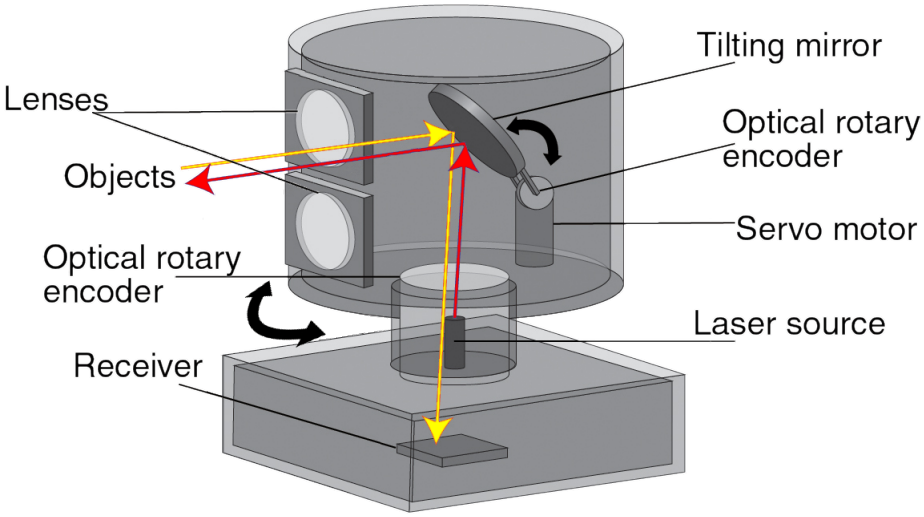


Figure 2.3: Diagram of the LIDAR. The red arrows is the laser that is being emitted and hits an object. The reflected light is now symbolized as the yellow arrows.

contain no moving part which increases the lifetime of the sensor. The lack of moving parts also enables the LIDAR to have a lower weight and a smaller size. According to [19] one additional advantage with the solid-state LIDAR is that the lower cost makes it possible to combine two or more sensors in order to make the system more fault tolerant.

A scene from a point cloud may be viewed in Figure 2.4

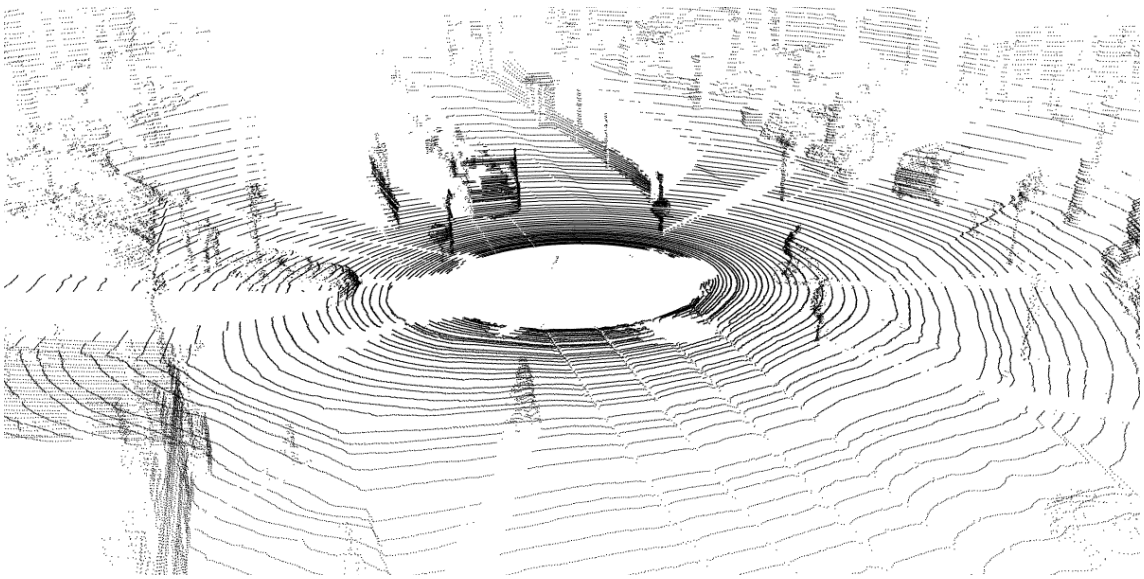


Figure 2.4: A point cloud created with the help of a LIDAR

2.1.2 Stereo Vision

There are a number of different ways to create point clouds. One way is to use stereo vision [20], which basically is two cameras that takes photos at the same time, with a known distance between each other. Through triangulation, it is then possible to calculate the distance to objects. As illustrated in Figure 2.5 an overview of how stereo vision works may be viewed. The blue circle is not detected through the left camera, which is illustrated by how the lines are dashed from the left camera. Since it is known where the cameras are mounted in respect to each other, it is possible to find out the distance from the objects to the cameras.

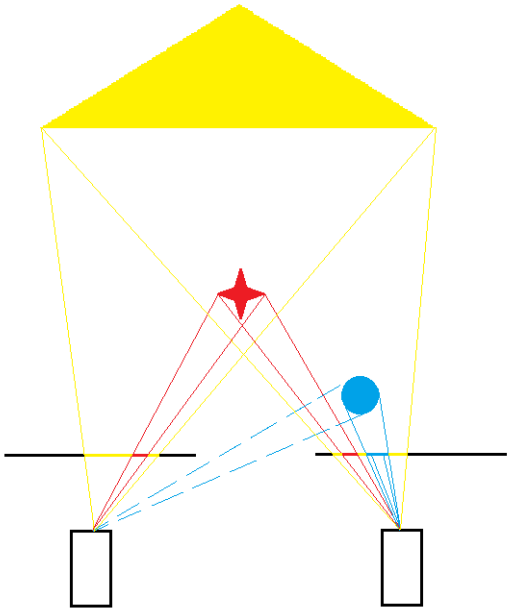


Figure 2.5: Illustration of stereo vision. The black boxes in the bottom are the cameras that capture the images, which are the black and colored horizontal lines. The blue circle, red star and yellow triangle are objects that the cameras detect, thus the different colors on the images.

2.1.3 Synthesized Point Clouds

In [21] there are many ways to generated point clouds with the help of a computer. One way is by using computer-aided design to create 3D models of objects, this model can then be imported and transformed into a point cloud. By using this method it is possible to get a symmetrical point cloud with all sides represented instead of just the one side that is turned toward the sensor. Another way to create point cloud is to simulate the real world and sensors, such as a LIDAR. This approach has many advantages, the main ones being the possibility to generate an almost infinite amount of labeled data, as well as the lack of noise which is present in all point cloud created by sensors, especially when the sensor is moving, e.g the LIDAR sensor on a car.

2.2 Segmentation

Segmentation is done in order to separate the points of the different objects, in the point cloud, into a smaller cluster that can be identified. There are a number of ways of doing segmentation for point clouds [22]. Edge-based such as Normal Segmentation [23] can provide good results if you find the right parameters, however, it has difficulties if the data contains noise. Another method is described in [24]. It uses the HDBSCAN algorithm which is a Region Growing method.

2.2.1 K-Nearest Neighbor Graph

The nearest neighbor graph is a useful algorithm when finding clusters and is illustrated in Figure 2.6. The algorithm calculates the distance from one point to the surrounding neighboring points and depending on the threshold, represented by the circle, the point is seen as a neighbor or not. These calculations are done for every point and result in points with common neighbors being classified as the same cluster. The final result of a 2D dataset can look something like Figure 2.7. The same idea can be applied to 3D points.

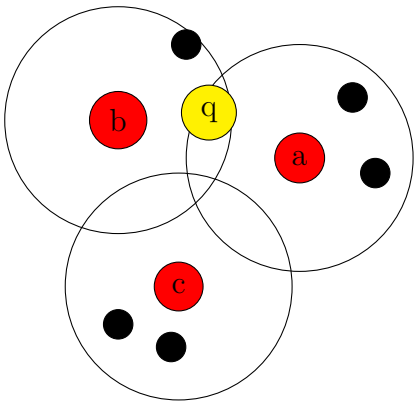


Figure 2.6: Node a and b share a neighbor labeled q, i.e. they belong to the same cluster. Node c has no common nodes with a or b i.e. it belongs to separate cluster.

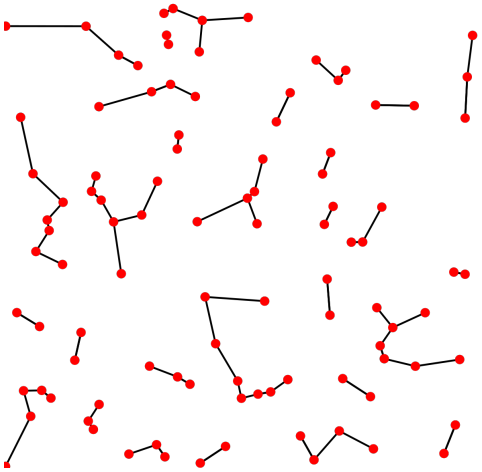


Figure 2.7: Clustering of 2D-points with the use of K-Nearest neighbour graph algorithm. The red circles are the points and the black lines symbolizes which points that are in the same cluster.

Algorithm 1 Pseudo code for K-Nearest Neighbour Graph

```
1: for i ← 1, ..., n do
2:     if (hasCluster(u_i)) then
3:         continue;
4:     end if
5:     NN ← findNeighborsInRadius(x_i, r)
6:     for all u_j ∈ NN do
7:         if hasCluster(u_i) ∧ hasCluster(u_j) then
8:             if clusterOf(u_i) ≠ clusterOf(u_j) then
9:                 mergeClusters(clusterOf(u_i), clusterOf(u_j));
10:            end if
11:        else
12:            if hasCluster(u_j) then
13:                clusterOf(u_i) ← clusterOf(u_j);
14:            else
15:                if hasCluster(u_i) then
16:                    clusterOf(u_j) ← clusterOf(u_i);
17:                end if
18:            end if
19:        end if
20:    end for
21:    if ¬hasCluster(u_i) then
22:        clusterOf(u_i) ← createNewCluster();
23:        for all u_j ∈ NN do
24:            clusterOf(u_j) ← clusterOf(u_i)
25:        end for
26:    end if
27: end for
28: for all C_i ∈ Clusters do
29:    if ‖C_i‖ < nMin then
30:        delete(C_i);
31:    end if
32: end for
33: returnClusters;
```

Description of K-Nearest Neighbor Graph Algorithm

- Step through each point.
- If the current point belongs to a cluster go to next point.
- For each point
  - Find all neighbors within distance r.
  - If any of its neighbors is in a cluster, attach the current point to the same cluster, then attach all neighbors without a cluster to the same cluster.
  - If the point belongs to a cluster and there is neighbours belonging to different clusters, merge all this clusters.

2.2.2 K-D Tree

K-D tree is a binary tree with k-dimensions. All none leaf nodes generate a hyperplane that divides the space into two parts, known as half-spaces. Points to the left are represented by the left subtree and the right side is represented by the right subtree. Depending on the direction order of the hyperplanes the K-D tree can look different. To construct the tree one cycle through the axes, x, and y in this case. In the left part of Figure 2.8 the data is first split in the x-direction, at (7,2), the median of the points, then the y-axis, and is then repeated. This method leads to the balanced tree in the right part of Figure 2.8, where each leaf is approximately the same distance from the root.

K-D tree is a common data structure used to lower the time complexity for nearest neighbor searches due to its fast range search. Range search can be used, to great effect in finding all neighbors within a specific distance of the queried point. Thanks to the tree structure where the neighboring points are in the same sub-branch, large portions of the tree can be eliminated and thereby speed up the process. This results in a relatively low time complexity, which can be seen in equation 2.2 [25].

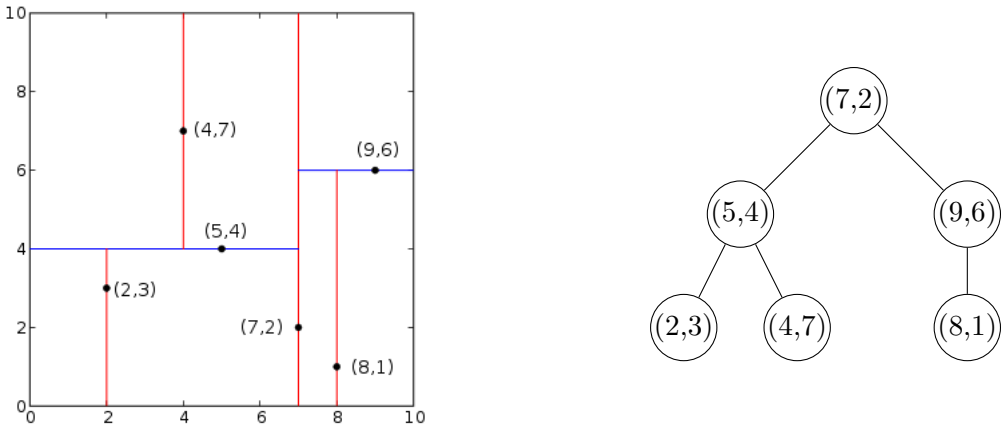


Figure 2.8: The left picture is the K-D data structure where the red lines symbolizes splits in the x-dimenstion and the blue lines are splits in the y-dimensions. In the right picture the same data is visualized as a tree.

$$t_{worst} = O(k \cdot N^{1-\frac{1}{k}}) \tag{2.2}$$

2.2.3 HDBSCAN

According to [24], one way to cluster data efficiently is by using Hierarchical-based Density-Based Spatial Clustering of Applications with Noise (HDBSCAN). Basically, the algorithm connects the points in the point cloud with each other and prunes the points that are farthest away. A more detailed description is listed below.

- Transform the space in order to distinguish clusters from each other and reveal the outliers.

$$d_{mr,k}(a,b) = max\Big(core_k(a), core_k(b), d(a,b)\Big) \tag{2.3}$$

  where $d_{mr,k}$ is the mutual reachability distance, a and b are two points located in the space, $d(a,b)$ are the euclidean distance between a and b. $core_k(x)$ is the core distance. That is the distance between point x and the kth neighbour. This is to make the algorithm more robust against noise. An example of this is seen in Figure 2.9.
- Build a Minimum Spanning Tree from the Mutual Reachability Graph where only the shortest edges are in. This will create a hierarchy of connected components, where some are completely connected and others are disconnected.
- From the data obtained in the Minimal Spanning Tree, a dendrogram is created. From this form, it is possible to eliminate clusters which contain too few elements.
- This dendrogram is then condensed so each of the clusters contains a specific number of elements. The condensation is done by pruning away the edge which has the lowest value if two or more edges have the same value both edges are removed simultaneously. At each split, the number of pruned nodes are compared with a variable, minimum cluster size. If the pruned nodes are more then this variable, they are stored as one new cluster, else they are seen as noise.
- In order to obtain stable clusters, the inverse of the distance is used, $\lambda = \frac{1}{distance}$, where λ is the stability. $\lambda_{p,max}$ is the value when the specific node was pruned, died. λ_{min} is when the cluster was created, i.e, when it got split off. Stability for these clusters are then calculated as:

$$\sum_{p \in cluster} \Big(\lambda_{p,max} - \lambda_{min}\Big) \tag{2.4}$$

- The final step is then to extract the clusters. This is done by comparing the child nodes stability with its parents. If the children stability is higher then the parent, the parent is discarded. Figure 2.10 illustrates this. The text in the circles indicates the label of the cluster and the stability of it.

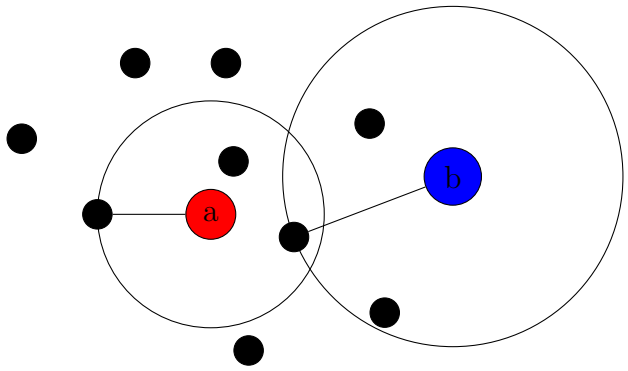


Figure 2.9: Illustration of the core distance, where core distance is 3, $k = 3$. The circle with red and blue color are the points that are being examined. Black circles are points that are to be evaluated.

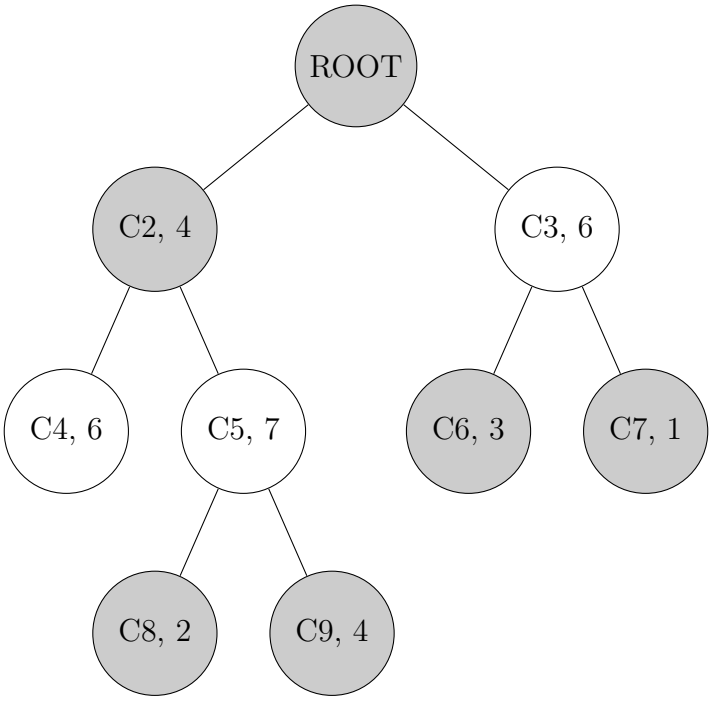


Figure 2.10: Graph over the selection of the clusters that are obtained. The text in the circles indicates the label of the cluster and the stability of it. The white nodes are the ones which are kept and the gray are discarded.

2.3 Classification

In supervised learning, it is common practice to use labeled data sets to train the system. The KITTI [4] project contains LIDAR scans of traffic environments, performed with an autonomous car. The data is labeled and can therefore easily be sorted into categories. This data is then divided into a test set, a training set, and a validation set. One common rule of thumb is the “80-20” rule.

2.3.1 Machine Learning Algorithms

There are a number of different ways to learn computer systems to classify data. In the article *Analysis Of Supervised Classification Algorithms*, [26], different supervised learning algorithms, which mainly are used to classify and regress, are being discussed.

Sliding Window Algorithm

The sliding window algorithm is used in image object detection. Basically, the method moves a window around the image and checks if the window contains the object that is requested. This algorithm is limited in that it is very computational expensive [27].

Naïve Bayes Classifier

Naïve Bayes Classifier is obtained by using Bayes Theorem. The classifier uses two variables, t, and $\mathbf{X}$. k is the total number of classes, t is the class for one specific and $\mathbf{X}$ is the set of predicative variables. Given the class variable, the Naïve Bayes assumes all of the predicative variables are conditionally independent.

$$p(x_{new}|t_{new} = k, \mathbf{X}, \mathbf{t}) = \prod_{j=1}^{D} p(x_j|t_{new} = c, \mathbf{t}) \tag{2.5}$$

Where x_j is the is the value of the j:th variable. t is the unknown parameters. D is the number of features.[28]

Support Vector Machine

Support Vector Machines, SVM, separates on data by "drawing lines" through them. It classifies data that are in two dimensions easily through 2.6

$$y_{new} = sgn\Big(\hat{w_0} + \hat{\mathbf{w}}^{\mathbf{T}}\mathbf{x}\Big) \tag{2.6}$$

Where $\hat{w_0}$ is the bias, $\hat{w}$ is the weights, y is the class and x is the data that is evaluated. This way, depending on which side of the line the data is on it either gets classified as 1 or -1. In order to get the system to validate as good as possible, the margin between the two different classes needs to be as large as possible in order to safely classify new data. To find that, it is necessary to find a point as close to the boundary as possible, thus optimizing as in 2.7.

$$min_{w,w_0}\frac{1}{2}|\mathbf{w}|^2$$

$$s.t \qquad y_i(\hat{w_0} + \hat{\mathbf{w}}^{\mathbf{T}}\mathbf{x_i}) \geq 1, i = 1 : N \tag{2.7}$$

N is the total number of data and i is the data being evaluated. There are a lot of different techniques to change the binary SVM into a multi-class classifier. However according to [28], these either takes a lot of computational time or are inaccurate, both are attributes required in automotive applications.

2.3.2 Artificial Neural Networks

Computer researchers consider ANN (Artificial Neural Networks) as systems that constitute a number of nodes with different linear functions. These nodes might be considered as a way to approximate basically every mathematical function [29]. Since ANN is a versatile way of examining data, it is known to be in various fields like psychology, engineering, and economics [30]. They can be used in image analysis for medical purposes [31] and in computer vision [32], but also in speech recognition [33]. According to [34] ANN may even be used to analyze the toughness of ferritic steel welds.
In [35] it is written that the nodes or the artificial neurons are organized in layers and each node can send a signal to one or more nodes in the next layer. When training an ANN, each weight is updated in order to give the input to next layer a higher or lower value. The output from the final layer is used to determine which class the objects is a member of.
From a point cloud obtained by a LIDAR, an ANN should be able to classify the surrounding objects, [36, 37, 10]. However, the development of these kinds of methods is a new research field and not very developed. This project's ANN will be based on the ANN from *Pointnet* which will be modified to fit this purpose.

Optimization Algorithms

Optimization algorithms are used to find the optimum for the objective function. There are a lot of different optimization algorithms, Adadelta, Nesterov Accelerated Gradient are two examples of this [38].

Stochastic Gradient Descent One other example of an optimization algorithm is the Stochastic Gradient Descent. Over a convex domain, SGD may optimize any convex function, this through the usage of subgradients [39].
The update function for SGM is as follows:

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{2.8}$$

[40] Where $J(\theta)$ is the objective function, θ are the models parameters. η is the learning rate, which is the same as step size. t is the current round, $x^{(i)}$ is one training example and $y^{(i)}$ is the corresponding label.

ADAM In order for an ANN to classify objects, it needs to be trained. In this project, Adam [40], were used in order to learn the network to receive a value for the optima. As seen in [40] there are a lot of other methods that also may be used. The reason why Adam is used is because it calculates adaptive learning rates for every parameter. Adam keeps the mean, m_t, and the uncentered variance, v_t of the gradients, which normally makes the learning rate pretty large in the beginning and lower when it gets closer to the optima. Thus Adam is preferable towards many other adaptive learning algorithms. The equation for how the mean and variance updates are in 2.9 and 2.10.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.9}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.10}$$

Both m_{t-1} and v_{t-1} are normally initialized as a zero-vectors and the decay rates are normally $\beta_1 = 0.9$ and $\beta_2 = 0.999$.
To avoid that the algorithm gets biased to zero, the bias-corrected mean and variance are calculated as in 2.12:

$$\hat{m_t} = \frac{\hat{m_t}}{1 - \beta_1^T} \tag{2.11}$$

$$\hat{v_t} = \frac{\hat{v_t}}{1 - \beta_2^T} \tag{2.12}$$

These are the variables used in the Adam update rule, 2.13.

$$\theta_{t+1} = \theta_t - \frac{\eta\ \hat{m}_t}{\sqrt{v_t} + \epsilon} \tag{2.13}$$

where η is the learning rate, normally set around 0.001 and epsilon is a very small positive value, 10^{-1} that prevents division by zero.

Over/Underfitting

Depending on how the accuracy of the validation set, different scenarios might occur. The model should fit the training data in order to let the ANN decide on which class the new object belongs to. However, if the model fits the training data too well, the model is not going to be able to make a good evaluation of the new data. The validation data have a large variance when speaking about the accuracy. This is called *overfitting*, [41, 42]. In an ANN with many nodes, the ANN is likely to follow a different path when making small adjustments to the input. One way to get control of it is by adding dropout when training the network. Basically, the dropout means that you disable a portion of the nodes in order to let the ANN learn with fewer nodes, thus forcing the data to develop fewer, stronger paths based on the same data. [42] When there are too few parameters the model will not be able to fit the data and the bias for the validation data get high. One good remedy for this is to add more nodes to the ANN. This is called *underfitting* [42], it is when the system will not model the training data and therefore will not be able to make any valid estimates of new data. An increment of the training data is what most researchers recommend.
Examples of how the over, under and the perfect fit looks like, can be found in Figure 2.11. The blue line indicates the perfect fit of the sampled data, which is the red dots. The green line is the underfitted system response and the yellow line is the overfitted output.

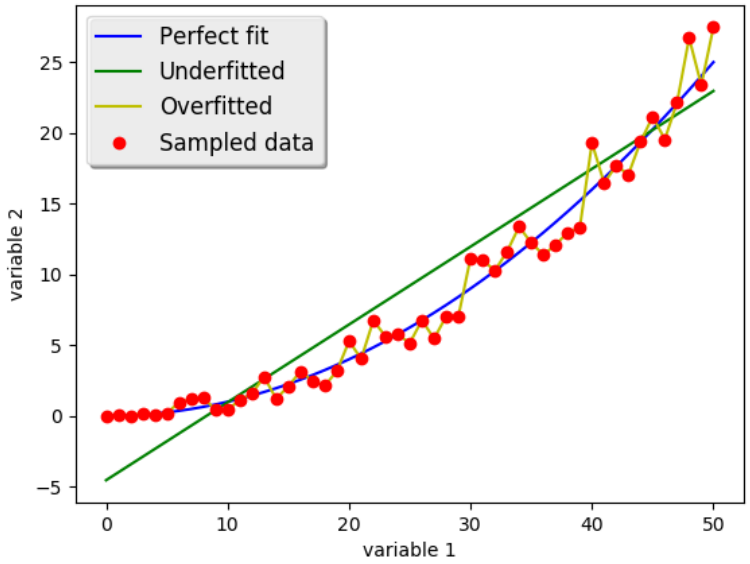


Figure 2.11: Over (yellow), under (green) and perfect (blue) fit of the sampled data (red dots). NOTE, the image is a toy example, and is specifically made to exemplify the different types of fitting.

Convolutional Neural Networks

Fully connected layers require a lot of nodes and therefore many parameters to be tuned. According to [35], one way reduce these parameters is to use convolutional layers, Figure 2.12. The output from the convolutional layer is calculated by convolving the weight of the node with its input. If it is already known how the gradient of the loss function with respect to the computational nodes in H^2, the backpropagation becomes

$$\frac{\Delta L}{\Delta w_i} = \frac{\Delta H_0^2}{\Delta w_i}\frac{\Delta L}{\Delta H_0^2} + \frac{\Delta H_1^2}{\Delta w_i}\frac{\Delta L}{\Delta H_1^2} + \frac{\Delta H_2^2}{\Delta w_i}\frac{\Delta L}{\Delta H_2^2}. \tag{2.14}$$

Where Δ symbolizes the derivate-operand, L is the loss, H is the layer, w is the weight. The superscript is what layer it belongs to and the subscript is to what node in the current layer. By assigning: $\delta_i = \frac{\delta L}{\delta H_i}$, rearranging 2.14 and calculate with respect to each individual weight we obtain:

$$\begin{aligned}
\frac{\delta L}{\delta w_0} &= h_0\delta_0 + h_1\delta_1 \\
\frac{\delta L}{\delta w_1} &= h_1\delta_1 + h_2\delta_2 \\
\frac{\delta L}{\delta w_2} &= h_2\delta_2 + h_3\delta_3
\end{aligned} \tag{2.15}$$

When $h^1 = [h_0, h_1, h_2, h_3]$, which are the output from each node in H^1, and $\delta^2 = [\delta_0, \delta_1, \delta_2, \delta_3]$ are the vector of the gradients in H^1. With this you can show that 2.14

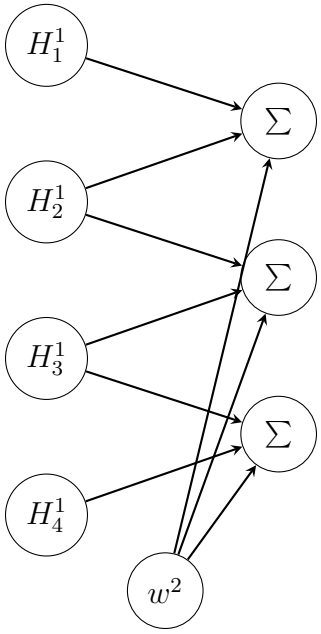


Figure 2.12: A piece of a CNN. H symbolize a node and the superscript gives its layer. The subscript indicate where in the layer the node is located in. w is the weight.

is actually a convolution, 2.16.

$$(h^1 * \delta^2)(t) = \int h^1(\tau)\delta^2(t-\tau)d\tau \tag{2.16}$$

Thus 2.17 is obtained by combining 2.15 with 2.16.

$$\frac{L}{W} = h^1 * \delta^2 \tag{2.17}$$

The quota $\frac{L}{W}$ shows the gradient of loss with respect to each weight. By noting that $w = \frac{H_i^2}{h_j}$ is then possible to obtain 2.18

$$\begin{aligned} \frac{\delta L}{\delta h_0} &= h_0\delta_0 + h_1\delta_1 \\ \frac{\delta L}{\delta h_1} &= h_1\delta_1 + h_2\delta_2 \\ \frac{\delta L}{\delta h_2} &= h_2\delta_2 + h_3\delta_3 \end{aligned} \tag{2.18}$$

The same calculations are used for the quota, $\frac{L}{h_i}$ where i is one node. This is needed in order to compute the backpropagation.
One other layer common in the CNN is the pooling layers. As described in [35], the pooling layer is a way to reduce the data in different ways depending on the information required. One common pooling is the *max pooling layer*, where it is the maximum of a region that is kept. An example of the max pooling with stride 2×2

is in 2.19, where *patch* is the patch, from where the maximum, is obtained. A patch is a subsection of the input image.

$$patch = \begin{bmatrix} 9 & 4 & 1 & 5 \\ 3 & 7 & 2 & 8 \\ 1 & 5 & 9 & 2 \\ 3 & 6 & 1 & 5 \end{bmatrix} \qquad maxpool(patch) = \begin{bmatrix} 9 & 8 \\ 6 & 9 \end{bmatrix} \tag{2.19}$$

PointNet

One type of convolutional neural network able to identify objects in 3D space is the *PointNet* [10]. It is designed to classify and segment detailed scanned objects and computer-generated point clouds. The scanner that *PointNet* utilizes is the Matterport scanner [12, 43], which takes more than 30 seconds for the scan to completed and more than one hour to upload to a computer. Needless to say, this would not function in an online application for automotive.
One property that sets PointNet apart from other ANN is that it does not change the point clouds into voxels or images. It does this by the use of three essential ideas. The first idea is to approximate a general function of the point cloud by utilizing a symmetric function on the transformed sets.

$$f(x_1, ..., x_n) \approx g(h(x_1), ...h(x_n) \tag{2.20}$$

In Equation 2.20 $f : 2^{R^N} \rightarrow R, h : R^N \rightarrow R^K$ and $g : R^K \times ... \times R^K \rightarrow R$ is a symmetric function. h is approximated by a multi-layer network and g by a max pool and a single variable function. The second idea is for the segmentation part of the PointNet which is not used in this project and is therefore disregarded. The third idea concerns the rotation and translation of the point cloud. The PointNet creators have designed transform nets which predicts an affine transform matrix by a sub-network and then use this transformation directly on the input point cloud.
The input point cloud needs to be a subset of points from a Euclidian space. Thus it needs to be able to be invariant under transformations, you should be able to both rotate and translate the whole point cloud. Further, each point in the set is not alone, together with its neighbors they make up important subsets of the total point cloud. The network learns to select interesting points from the cloud according to a set of criteria and encodes this into the layers for the network. One interesting feature with PointNet is that it summarizes a point cloud by a few key points.
PointNet, which is an ANN that has a unified architecture, use a alignment/transformation network to make the data more comprehensible. This alignment/transformation network is basically the same as the back end of PointNet. The difference between the alignment/transformation nets and the normal one concerns that the first transform net outputs a 3 × 3 matrix and the second a 64 × 64 matrix and the output from the system itself is the probability for each of the classes. Both of the transformations gets multiplied with the input to these nets. The output from PointNet is either the label for one segment or it is the label for the entire input.
One downside with PointNet is that it does not consider nonuniform sampling density of the point cloud. Models that are trained for dense point clouds might have a problem with sparse clouds and the other way around.

2.4 Summary of the Theory

In this chapter different ways of generating point clouds were covered. Also, a short description of KITTI, PointNet and also Pro-SiVIC are being presented.
This section also covered different algorithms and methods used to segment the data. Further, it also covered different ways to classify data. It was decided to use HDBSCAN because it showed good results when performing segmentation. The accuracy of the algorithm was about the same as Nearest Neighbor but HDBSCAN was much faster. Another advantage with HDBSCAN was that it was easy to implement in our code due to preexisting libraries. Different objects in nature have complex shapes and they would, therefore, have complex equations in order to be drawn in a computer, it was therefore decided to use ANN. ADAMs are being used since we do not have that much data to use.

3
Methods

This chapter describes the methods used to conclude this project. The first section states the research questions, followed by useful tools that were used in this project. The third section covers the first subsystem, the segmentation. The fourth section describes the classification. And the chapter is finished by the evaluation methods of the system and its parts.

3.1 The Research Questions Revisited

The research questions are as follows.

RQ1 How well does machine learning methods, trained on real traffic environments perform on simulated environments?

RQ2 Is it possible to use machine learning algorithms with classical algorithms in order to make a competitive system for object classification?

RQ1 is answered by creating a system and using KITTI data for testing and training the machine learning algorithm. When the system is working well with the KITTI data, the simulated data will be used for testing and verification. RQ2 will be answered when the evaluation of the system is performed.
From these questions, the system that is viewed in Figure 1.2 were designed. This figure makes it is clear how the data flows in the system.

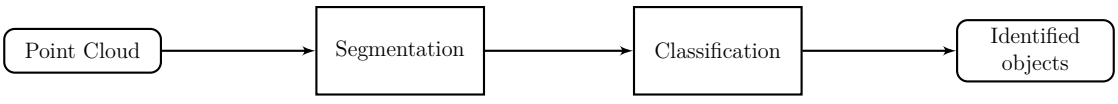


Figure 3.1: An overview of the to be designed system called DELIS. The boxes with soft corners are input and output and the boxes with hard edges are the parts of the system. As this section covers the methodology, the red box describes in which part of DELIS where the method is used.

3.2 DELIS

DELIS were created with the alternative to switch between the two different segmentation algorithms. After the segmentation, each cluster was randomly sampled

for 128 points so that they fit into the neural network. The clusters and the points are then labeled and stored in a file which can later be displayed. Delis also estimate the center of an object by calculating the mean value for each axis in the cluster.

3.3 Useful Tools

This project has utilized a couple of different tools, this subsection gives a brief description of KITTI, and ESI-Pro-SiVIC both are tools from where the data used in this project was obtained from.

3.3.1 KITTI

KITTI [4] is a project carried out by Karlsruhe Institute of Technology in collaboration with Toyota Technological Institute at Chicago, where the aim was to create a benchmark for real-world computer vision. The researchers performed data collection of traffic environments with the help of a car equipped with the following sensors.

- Velodyne HDL-64E Laserscanner
- 2 Grayscale cameras, 1.4 Megapixels
- 2 Color cameras. 1.4 Megapixels
- 4 Varifocal lenses, 4-8 mm

The sensor used for this project is the Velodyne Laser scanner that creates point clouds of the surrounding environment. The scanners spin at 10 frames/second and obtain approximately 100k points per cycle.

3.3.2 ESI Pro-SiVIC

A company called ESI, [5] have created a 3D simulation tool for sensors, like LIDAR, GPS, and cameras in a realistic environment. The environment is not just buildings, roads, and other vehicles, but also different kinds of weather and lighting conditions. According to *Esi Pro-SiVIC - 3D simulations of environments and sensors*, [44], the simulator allows a good way to try algorithms for self-driving vehicles, perform safety studies, analysis of sensor robustness and system performance.

3.4 Point Cloud

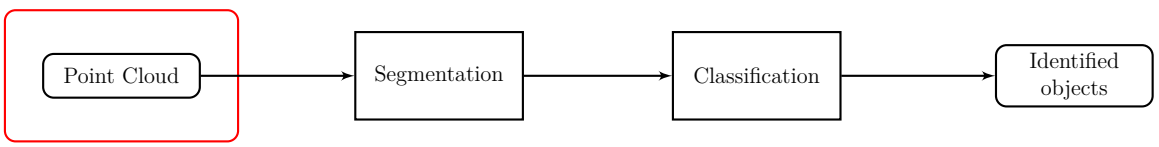


Figure 3.2: The red box indicates what part of DELIS that is currently being discussed, in this case the input data, point cloud.

The data used in this thesis were downloaded from the *KITTI Vision Benchmark Suite* homepage [4] and the point clouds were divided into text files with x, y and z coordinates for each point. There were also labeled data that provided information about what objects each point cloud contained and the position of the identified object. The labeling of the data was presented in the form of bounding boxes placed around the objects of interest, e.g. cars, pedestrians, and cyclist. All points inside such a box could be seen as belonging to the same object. With the help of this information, the points were gathered into clusters and put into arrays with a label of which object it was identified as. The clusters contained a varying number of points depending on the distance to the LIDAR and the size of that object. The cars contained the most points and the pedestrians the fewest.

These clusters were also transformed into their own 3D space in order for the ANN to be able to understand the data, e.g normalized and centered around the origin. This because the position of the object in the point cloud should not play a part in the classification process.

Visually it was possible to determine what kind of object that a cluster is when it contains more than 90 points. Because of this, and the fact that graphics processing units work most efficient in base two, it was decided to use clusters that contained 128 points. These points were randomly sampled from clusters with more than 128 points and clusters that contained less then 128 points were discarded. The data sets were further enlarged by sampling multiple times in the objects with the most points. With this method, a larger data set was created. Since the dataset should be balanced with an equal number of cars, pedestrians and cyclists the final size of the data set was 800 for each type of object. The limiting factor in the process was the number of pedestrians present in the KITTI data set. They were both the fewest clusters and the smallest, hence the number of times that it was possible to sample the clusters and create objects from the pedestrian clusters were not as many as it was for the cars and cyclists. The data were finally divided into training, testing and validation sets which are needed for different parts of the development of DELIS.

The other type of data used in the project was point clouds from the ESI simulator[44], which was used in the final evaluation of the system. This data was unlabeled and it was therefore not possible to create a large dataset entirely of simulated data. The data was created by building an environment that is as realistic as possible, with the help of the simulator tool. This means a simulation with parked and moving cars, buildings and pedestrians. Unfortunately, cyclists were unavailable in the simulator which limits the results when comparing simulated data and the real world data. The simulator allowed for the placement of LIDAR sensors in the environment created, in this case, a car. It was also possible to change the properties of the LIDAR, for examples resolution of the sensor and the aperture angle. In this project, the properties were set to the same as Velodyne HDL-64E [45] since that is the sensor used in the KITTI data collection. With the sensor in place, the simulator allowed for the car to be driven around in the environment and collect data. The data obtained were segmented and put into the ANN in the same way as the KITTI data.

3.5 Segmentation

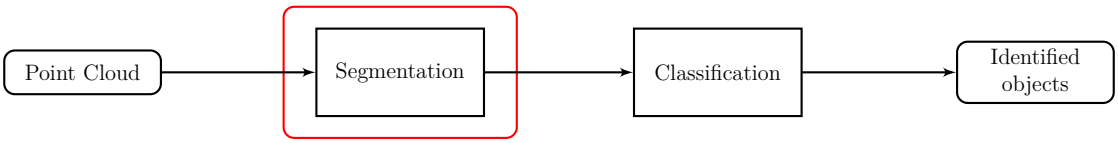


Figure 3.3: This chapter is about the segmentation. The arrow that goes from Point Cloud indicates that the segmentation part is being fed with Point Clouds. The output from the segmentation is smaller point clusters of objects. These clusters are sent to the classification part of DELIS

In order to separate the objects that are to be identified, from the point cloud, some form of segmentation is needed. There are different ways of segmenting point clouds, one way is to perform it with the help of an ANN that is also used to identify objects, but this would not answer the research questions. In this project, the segmentation was performed before the machine learning algorithm i.e. segmented clusters are the input to the machine learning algorithm.
The segmentation was performed with two different algorithms, the first one was an algorithm that used the nearest neighbor algorithm and graph building, the second was the HDBSCAN segmentation algorithm [24]. The nearest neighbor algorithm was chosen because it is one of the classical methods and the first that came to mind. Our nearest neighbor-algorithm was designed and implemented by the authors. HDBSCAN was decided because it is a fast and stable method.
The purpose of both algorithms was to divide the point cloud into clusters, that consists of points which belong to an object in the scene. These objects could be cars, pedestrians, trees or lamp posts, just to mention a few. Using one of these algorithms on each scene, resulted in each scene being split into multiple objects, with varied size and number of points. These are the objects that needs to be classified.
The first algorithm, K-Nearest Neighbor graph, started by building a KD-Tree of all the points. This KD-Tree were then used in order to speed up the querying of the nearest neighbor in the algorithm. A description of the algorithm is seen below and the pseudocode can be seen in 1

1. Step through each point.
2. If the current point belongs to a cluster go to next point.
3. For each point
   (a) Find all neighbors within distance r.
   (b) If any of its neighbors is in a cluster, attach the current point to the same cluster, then attach all neighbors without a cluster to the same cluster.
   (c) If the point belongs to a cluster and there are neighbors belonging to different clusters, merge all these clusters.

The second algorithm used in the project was HDBSCAN [24]. This algorithm was implemented and the results were compared to the previously described algorithm. Further explanation of this algorithm is written in the theory chapter. To improve the results of the algorithms the ground were removed from the point cloud. This

was done by removing all points below a certain height. This operation helped in separating the objects in the point cloud that were previously connected to the ground. All points beyond a certain distance were also removed in order to speed up the segmentation.

3.6 Classification

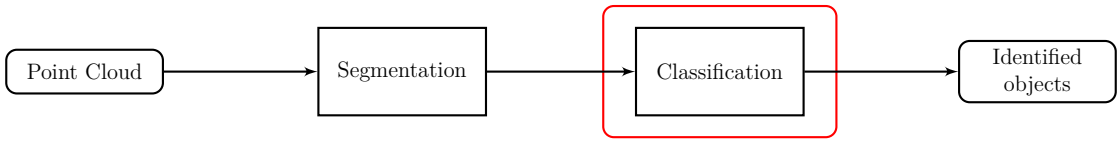


Figure 3.4: This section describes the classification part of DELIS. Point clusters from the segmentation part is the input. The ANN identifies the point cluster and label it. This labels together with the point clusters are the final output from DELIS.

ANN was used in this project as a method to classify the different objects because they have a fast execution time, and when properly trained, they have high accuracy. They are also able to handle a large amount of data. The reason for using *PointNet*[10] is that it is able to work on unordered point clouds without voxelization or rendering, this makes it faster than many other networks. *PointNet* uses the *Tensorflow*[46] framework to build their network. The code is open source and easily modifiable.
It was decided to use the "80-20 rule" to split the data set into training and testing, but since a validation set was also required the sets ratio became test 20%, validation 16% and training 64%.
PointNet is already programmed to use either *SGM* with momentum or *Adams* as its optimizer.
According to the paper *An overview of gradient descent optimization algorithms*, [40], different learning algorithms should be used depending on how much training data that is available. If the input data is sparse, then one should use Adam.

3.6.1 Architecture

A common approach to creating an architecture is trial and error with a few common guidelines. The common architecture for all the convolutional neural network is to start with a couple of convolutional layers followed by a max-pooling layer. Then, there normally is a few fully connected layers that might be connected to a layer of dropout. The last layer use to be a fully connected layer with as many nodes as there are classes to identify [10, 47, 35, 48, 6]. A script written to generate different net structures were implemented. The different structures were evaluated and the best one's execution time and accuracy were compared in order to find an ANN as good as possible. The networks that got evaluated were trained 100 epochs.
In order to produce a good architecture design, the computer randomly generates a couple of hundred different designs and evaluated each one. The architecture generation follows the algorithm seen in Figure 3.5. The purpose of the algorithm

is to make sure the generated design is a valid one and is possible to execute, e.g there cannot be two transforms after each other. The box which is labeled Exp is used to change the dimensions of the point cloud in order to fit the ANN.
The architectures were implemented in tensorflow.

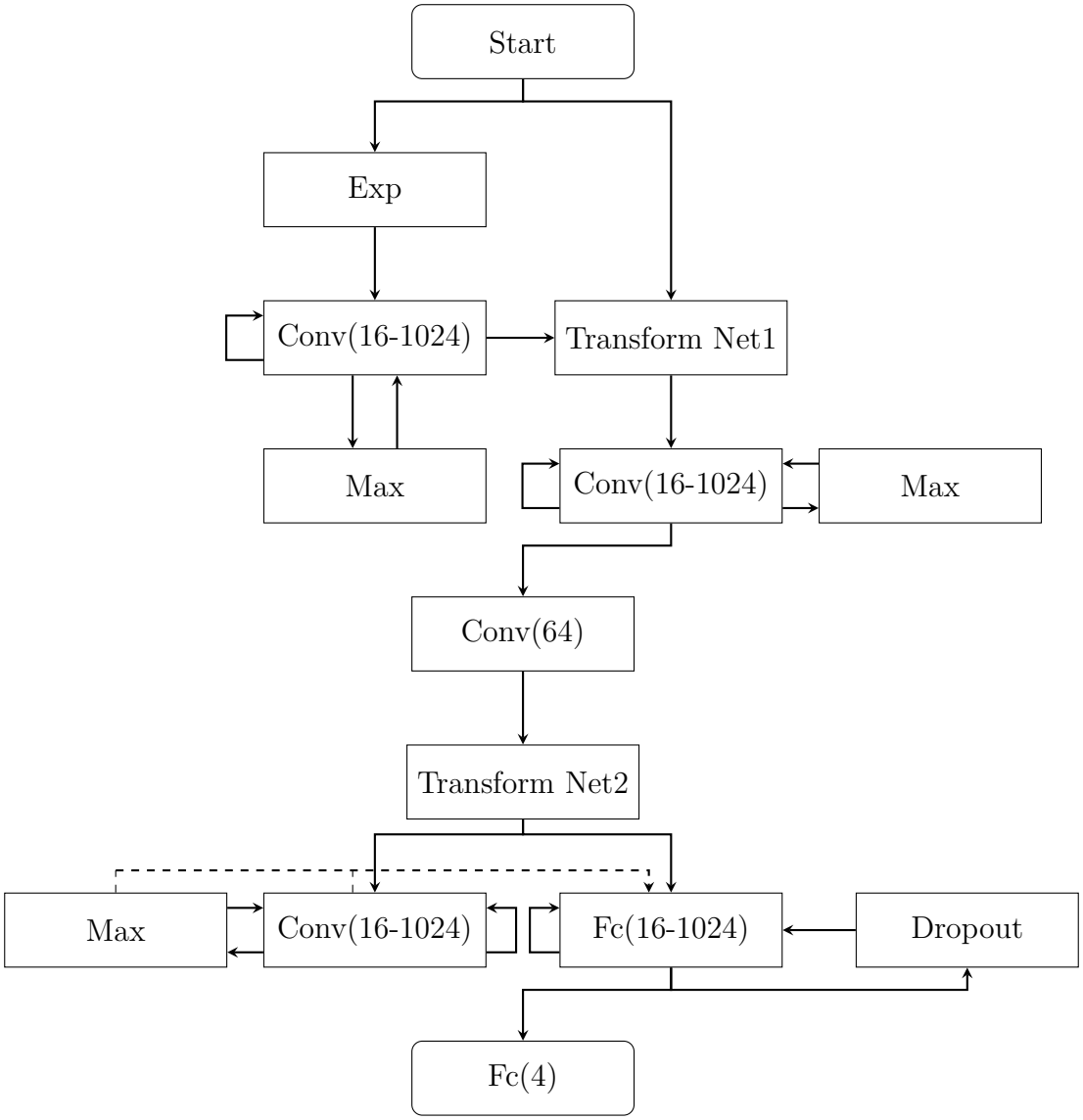


Figure 3.5: Flowchart of the Net-Generator. It was decided that we wanted at least four layers, two convolutional and two fully connected layers. This because the ANN needs a lot of nodes in order to avoid underfiting. Further, from [1], it is proven that shallow ANN needs exponentially more nodes then deep networks. First the generator decides if the ANN should start with a transform or with a convolutional block. Afterwards there are at least two convolutional layers, followed by the second transform. From the last max pooling and the convolutional layers there is a dotted arrow pointing to the box labeled FC(16-1024), this is solely to indicate that the arrows cross each other. There are at least two fully connected layers in the end. And, of course, the last layer have as many nodes as there are classes. Branching is equally weighted.

3.6.2 Tensorflow

There are a lot of different frameworks to choose from when working with ANNs, such as Tensorflow, Caffe, and Theano. In this project, Tensorflow were used thanks to the fact that it has a built-in support for deep neural networks and Convolutional Neural Network, (CNN) [46]. Since this project is just a proof-of-concept, it has more focus on fast implementation and less on the execution time, which suits Tensorflow. Tensorflow is also a good choice when it comes to designing a network for Python.

3.7 Evaluations

In order to know how well DELIS is compared with other existing systems, it is vital to evaluate it. Each part was evaluated individually as well as the complete system.

3.7.1 The Segmentation

The evaluation of the segmentation was done in part by measuring the performance and in part by measuring the accuracy. The performance was evaluated by measuring the execution time of the segmentation and the cluster creation time over a few different point clouds. The accuracy evaluation was done by counting the number of points in the clusters from the segmentation and how many of them were correctly labeled with the help of the ground truth data.
The segmentation evaluation is performed on both the KITTI data and the simulated data. The performance was easily measured in the same way for both data types. However, the accuracy evaluation was only possible on the KITTI data since no ground truth data for the objects in the simulated data were available.

3.7.2 The Classification

For the classification, the accuracy for different neural network architectures are tested and evaluated. The test set consists of KITTI data that is labeled, with a size of 480 objects from each of the three classes. This allows for an average accuracy to be calculated for each class and architecture.

3.7.3 DELIS

DELIS is evaluated with point clouds from both ESI Pro-SiVIC and KITTI. A graphic display of the result was created, where the points were displayed in different colors depending on what the neural network identified the object as. This visualization was used when the system where evaluated. The evaluation was done by manually inspecting the different point clouds that were feed into the system and count the cars, pedestrians, and cyclists in the image from the visualization program.

3.8 Summary of the Methods

In this chapter, we argue why we chose KITTI and ESI Pro-SiVIC as data sources for our system. We briefly describe the different segmentation methods and have some arguments on why ANN are chosen as the classification method. This is followed by how the ANN we chose to use were obtained. Lastly, the evaluation of the segmentation, the ANN, and the whole system is described.

4
Results

The results of the project are listed in this chapter. It starts with the results for the segmentation and continues with the classification results. Finally, the results from the total system are covered. This chapter ends with a discussion where this system is compared with similar applications, i.e. YOLO [6] and VoxelNet [7].
It was concluded that the Nearest Neighbour algorithm was slower then the HDBSCAN, the problem is that neither is the execution speed is too slow for a real-time application. The accuracy for the segmentation differed from one scene to one other, however, HDBSCAN showed better performance overall.
The resulting network had a validation accuracy of around 95% when using the training dataset.
The final system where able to identify a few items depending on the scene, still the system has much room for improvement. Our conclusion is that the segmentation part of the system has the most to improve.

4.1 The Individual Parts

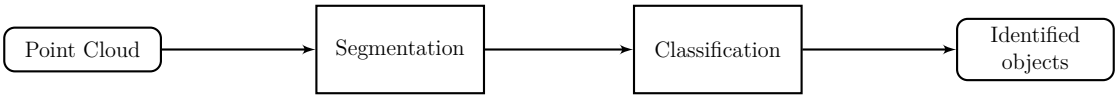


Figure 4.1: An overview of the to be designed system. The boxes with soft corners are input and output and the boxes with hard edges are the parts of the system.

DELIS is constructed by two different parts, one part that segments and one that identifies. The way the different parts interacts may be viewed in Figure 4.1.
DELIS, seen in Figure 4.1, takes the scan from a LIDAR as a whole, transform the input point cloud into Euclidean coordinates and segment it. Thus the input for the ANN is only clusters that have been segmented from the scene, which have more than 128 points.
The evaluation of DELIS is done in three steps. First, the segmentation, then the ANN, and finally the parts together.

4.1.1 Segmentation

This section contains the results of the segmentation evaluation which was done by measuring the execution time of the segmentation and the truthfulness of the segments.

Algorithm	Dataset	Time
Nearest Neighbour with KD-tree	Dataset 1	30.74
	Dataset 2	15.72
	Dataset 3	24.76
HDBSCAN	Dataset 1	7.80
	Dataset 2	3.61
	Dataset 3	4.39

Table 4.1: Execution time for the segmentation of each dataset in seconds

In a real-time system e.g. a car, the execution time of a program is of utmost importance. As can be seen in 4.1, HDBSCAN is significantly faster than the alternative. Even though the HDBSCAN is faster, it is still far from being fast enough to be used in a real-time system. The segmentation was run on an Intel Core i7-5930K CPU @ 3.50GHz.

The truthfulness of the segmentation was evaluated by comparing the segmented clusters with the ground truth data in the KITTI dataset. The clusters compared where manually identified in the scene and then compared to the ground truth. The results can be seen in 4.2, 4.3, the tables show the number of points in the ground truth cluster and in the segmented cluster. It also shows the number of points the clusters have in common i.e the points that were correctly labeled. the table also displays if the segmented object was identified correctly in the Neural Network.

	Object	GT	Seg	Common points	Identified
Dataset 1	Car 1	2703	1756, 802	1752, 802	No
	Car 2	967	2994	861	No
	Car 3	470	722	398	No
Dataset 2	Pedestrian	307	269	269	Yes
	Cyclist	853	808	805	Yes
Dataset 3	Car 1	3298	3105	3104	Yes
	Car 2	1290	1203	1168	Yes
	Car 3	435	281	273	No
	Car 4	3859	4889	3273	Yes

Table 4.2: Segmentation of KITTI Data with HDBSCAN. The table shows number of points in the segmented cluster and the number in the ground truth. It also displays number of points the clusters have in common. The table also shows if the segmented clusters where successfully identified. (Car 1 in dataset 1 was segmented into two clusters)

4.1.2 Classification

This section shows results from the ANN that was constructed, starting with the ANN-generator and ends with the how the selected ANN behaved on the test-set.

	Object	GT	Seg	Common points	Identified
Dataset 1	Car 1	2703	121208	2687	No
	Car 2	967	121208	967	No
	Car 3	470	2186	468	No
Dataset 2	Pedestrian	307	269	269	Yes
	Cyclist	853	808	805	Yes
Dataset 3	Car 1	3298	3171	3171	Yes
	Car 2	1290	1216	1181	Yes
	Car 3	435	409	401	No
	Car 4	3859	4889	3273	Yes

Table 4.3: Segmentation of KITTI Data with Nearest Neighbor. The ground truth is the same as above the only difference to the table is the type of segmentation algorithm used

Architecture

The only thing that is kept from *PointNet* are the transforms.
In order to choose between the different obtained architectures, obtained from 3.6.1, every ANN that has over 80 % accuracy for the last 10 epochs on the validation set was put in a list where the ANN with the lowest computational time was selected. This net was trained an additional 1200 epochs before getting evaluated against the test set. After retraining, almost all the cars were correctly classified. The class with the lowest accuracy is Pedestrians with 0.88 % accuracy. This is when only the ANN were run. i.e the segmentation was done by *KITTI*. The accuracy is to low to use in a real car, however, as a proof of concept, it is acceptable.
Figure 4.2 is a graph for the networks structure. The arrow indicates the path the data is processed. The input starts getting processed in Transform Net1, which the first layer is a convolutional layer with 64 nodes, displayed on the left side. The right side is Transform Net2, which is the same as Transform Net1, except for the final layer. At the end of the of the main network, the middle part, there is one box with the label “Dropout, 0.75 P" that means that each node in the layer before has a 75% chance of being shut down. This has been introduced in chapter 2.3.2.

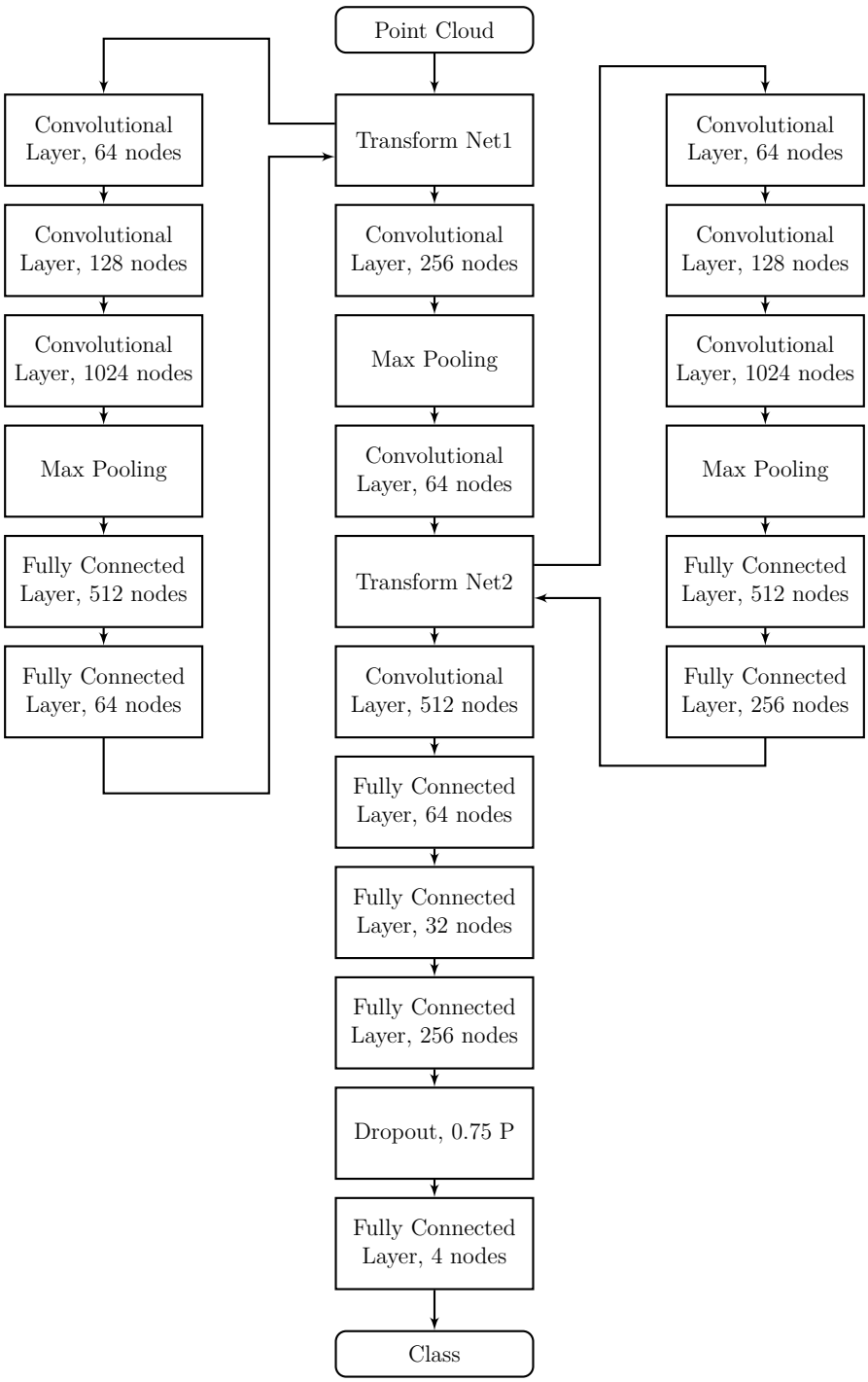
ANN on the Test Set

As mentioned previously, the data is split so three different categories of data are obtained. The training and validation set is there to improve the training of the ANN. The third set, (the test set), is used in order to show how well a system behaves on new, previously unseen data. The results published was obtained when running the test set after training for 1300 epochs:

Test on the Usage of transformations

When the system altogether is executed, during 100 epochs, the average accuracy for the last ten epoch became 92.832. The values for the different epochs are listed in 4.5

Figure 4.2: The resulting ANN. The left wing is what is inside Transformation Net1 and the right wing is Transformation Net2. In the middle the main network is described. The rectangles are the layers and the text inside the box tells what kind of nodes that are in that layer. The number tells ho many nodes that are present inside the current layer. Max Pooling have no number since it have no nodes. Dropout 0.75 P indicates that during training there is a 75 % chance that a node is turned of.



Number of examples	660
Number of correctly assigned	625
Eval mean loss	0.168
Eval accuracy	0.947
Eval avg class accuracy	0.948
Cyclist	1.0
Car	1.0
Pedestrian	0.881
Misc	0.911

Table 4.4: Table over the behaviour for the ANN. The top five entries tells how many examples that were in the test-set and some numbers that are comparable with other nets. The bottom four is the percentage of the correct evaluated objects.

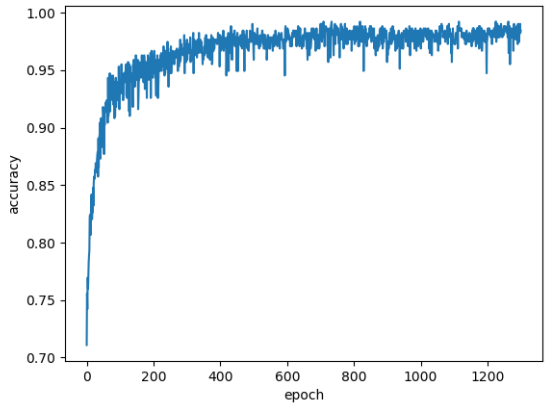


Figure 4.3: The final training for the chosen network with 1300 epochs.

When running the chosen network without Transform Net1 network the average for the last ten epochs of a 100 epochs run, was 0.631. In table 4.6, it is apparent that the system does not perform as well when the transform net is removed. Worth noticing is that that the correct guessed between epoch 92 and 98 have dropped with more than 10%. Further, the bias is low. Thus the system is neither over or under fitted.

The Transform Net2 was also disabled alone, the last ten epochs are listed in table 4.7. The average became: 92.148

The last experiment with the Transform Nets that were concluded was disabling both the transform nets. Again the average for the last ten of the 100 epochs was recorded. The result from when both transform nets are disabled are in table 4.8.

Epoch	Percentage
91	95.12
92	91.80
93	92.77
94	91.41
95	92.38
96	92.19
97	93.16
98	92.19
99	93.16
100	94.14

Table 4.5: Impact on the evaluation-set when both transform nets are active

Epoch	Percentage
91	66.60
92	67.77
93	61.33
94	64.26
95	64.45
96	62.11
97	60.35
98	57.03
99	65.63
100	65.43

Table 4.6: Impact on the evaluation-set when disabling Transform Net1

Epoch	Percentage
91	88.67
92	93.16
93	93.75
94	90.82
95	91.41
96	91.99
97	92.58
98	94.14
99	92.77
100	92.19

Table 4.7: Impact on the evaluation-set when disabling Transform Net2

Epoch	Percentage
91	64.84
92	61.72
93	65.23
94	62.11
95	63.28
96	54.30
97	51.76
98	57.03
99	53.52
100	63.28

Table 4.8: Impact on the evaluation-set when disabling both transform nets

4.2 DELIS

The evaluation is done by running several scenes of LIDAR data through the different parts, the result is then compared to manually identification of objects in the same scene. The performance of DELIS is measured in how many objects it correctly identified.
The system showed varying results depending on segmentation method and the dataset. The Figures 4.4, 4.6, and 4.8 displays the resulting identification of the network for HDBSCAN segmentation. The images 4.5, 4.7, 4.9, displays the resulting identification with Nearest Neighbor segmentation.
In order to evaluate the accuracy of DELIS two series of 50 frames where entered into DELIS. The results can be seen in 4.15, 4.16, 4.17 and 4.18.
The system where also used with simulated data. Although the network was unsuccessfully in identifying cars in the simulated data it managed to identify pedestrians, examples can be seen in the Figures 4.11 and 4.14, the pictures show the segmented and correctly labeled clusters from the KITTI dataset. 4.10 pictures a pedestrian that the network identified correctly, 4.13 pictures a segmented cluster from the simulated data that the network was unable to classify. All code for DELIS can be found on *https://github.com/ChristianLarsson91/DELIS*.

4.3 Evaluation of Identified Data

In order to verify the output of DELIS we manually look at the identified objects. The objects are counted and verified by us manually, so that they are correctly labeled, in order to get the accuracy of DELIS.

4.3.1 Dataset 1

Figure 4.15 is a diagram where the number of correct and incorrect are blue respectively red bars for each frame in the dataset 1 using nearest neighbor. The figure

Figure 4.4: Dataset 1 with HDBSCAN segmentation, one cyclist in cyan and one pedestrian in orange

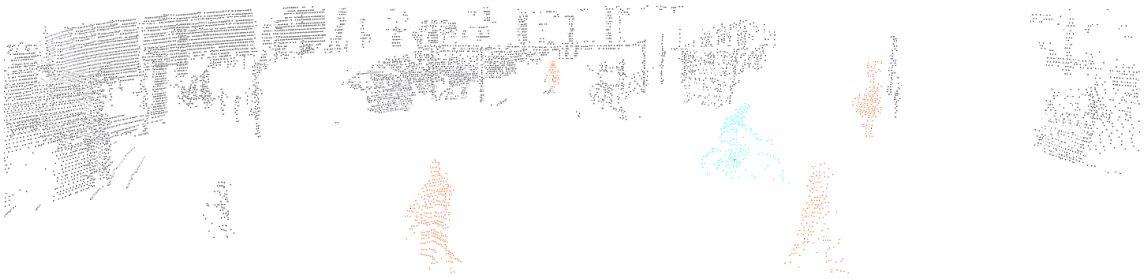


Figure 4.5: Dataset 1 with K-nearest neighbor graph segmentation, one cyclist in cyan and two pedestrian in orange and a traffic sign that has been labeled as a pedestrian

4.16 shows also shows the correct and incorrect objects identified in each frame in the dataset 1, however, now DELIS is used with HDBSCAN.

4.3.2 Dataset 5

Figure 4.17 is a diagram where the number of correct and incorrect are blue respectively red bars for each frame in the dataset 5 using nearest neighbor. The figure 4.18 shows also shows the correct and incorrect objects identified in each frame in the dataset 5, however, now DELIS is used with HDBSCAN. .

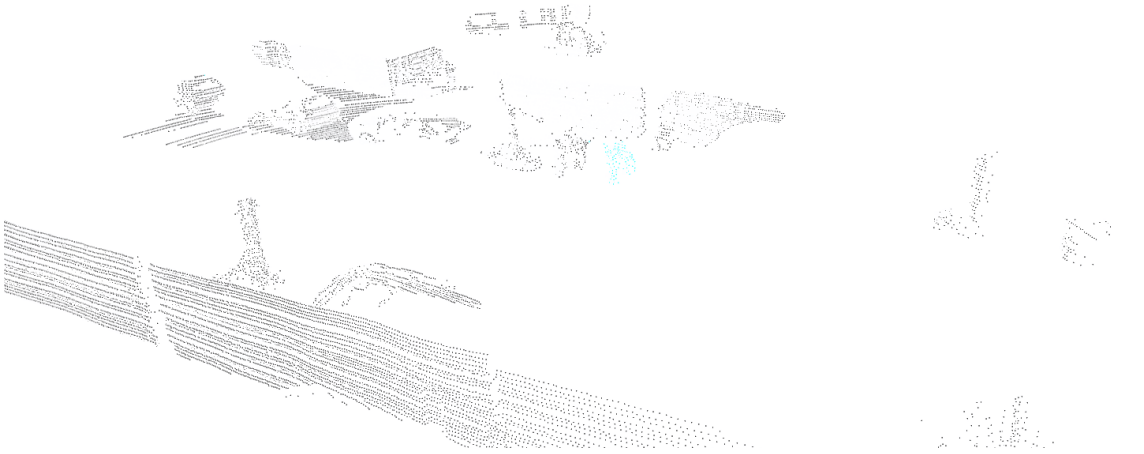


Figure 4.6: Dataset 2 with HDBSCAN segmentation, object incorrectly identified as a cyclist

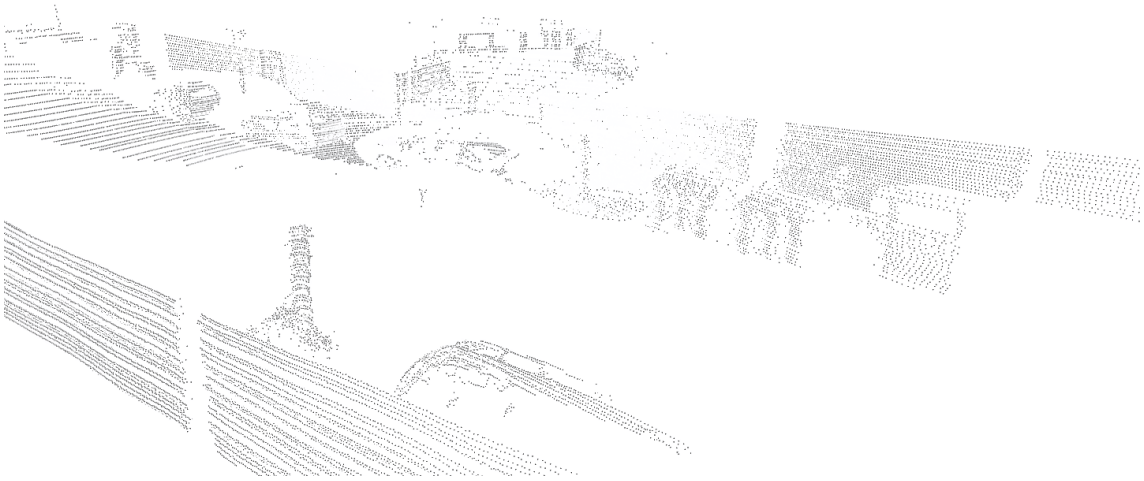


Figure 4.7: Dataset 2 with K-nearest neighbor graph segmentation segmentation, nothing identified

4.4 Summary

DELIS consists of two subsystems, one segmenting and one classifying part. The two subsystems are first evaluated independently and then also together. It is clear that HDBSCAN is much faster then Nearest Neighbor Algorithm and the results are almost the same. Further, we try the chosen ANN with and without the transform nets, where it is obvious how important the transform nets are. The chapter was concluded by trying DELIS on complete scenes.

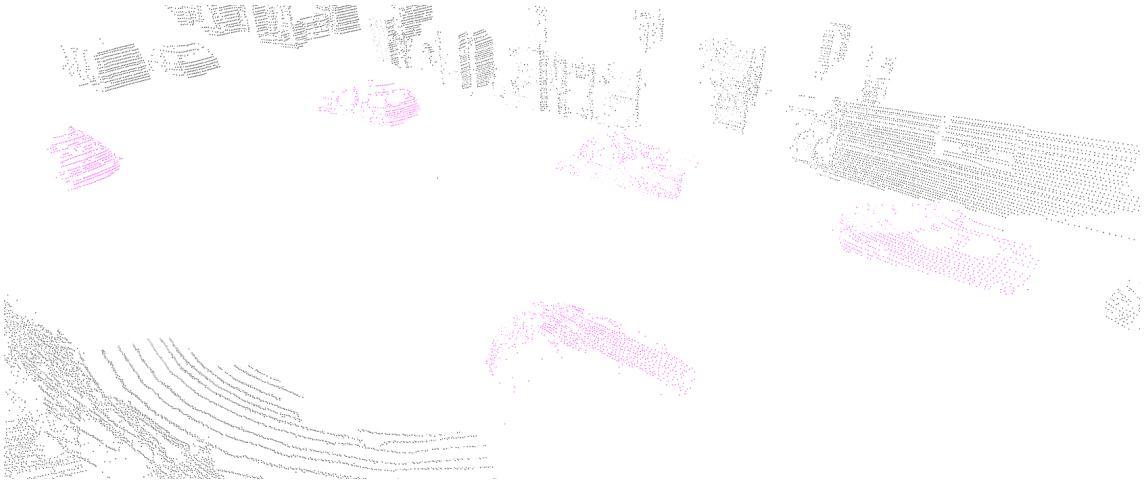


Figure 4.8: Dataset 3 with HDBSCAN segmentation, five cars in purple

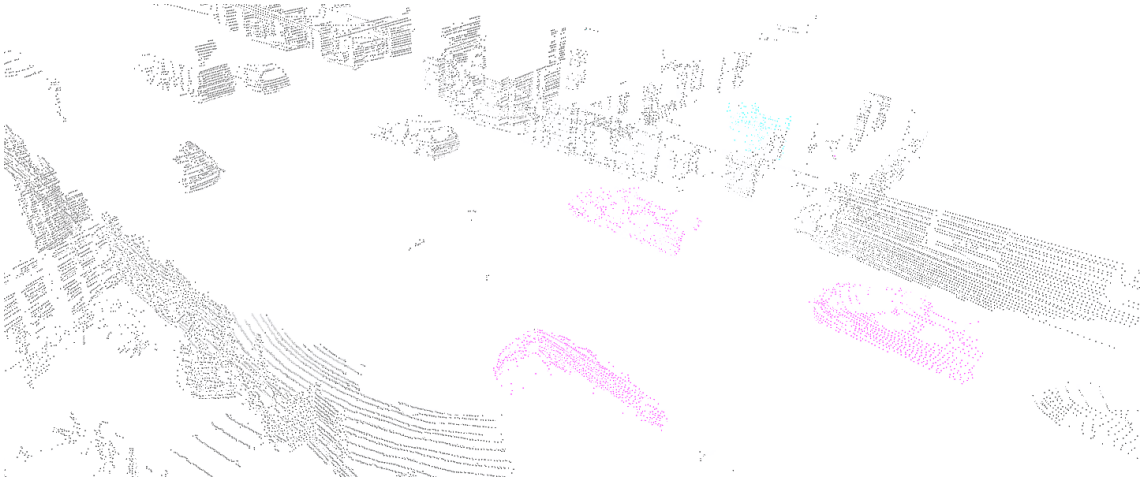


Figure 4.9: Dataset 3 with K-nearest neighbor graph segmentation segmentation, three cars in purple

Figure 4.10: Simulated data of a pedestrian

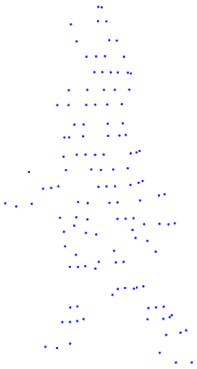


Figure 4.11: KITTI data of a pedestrian

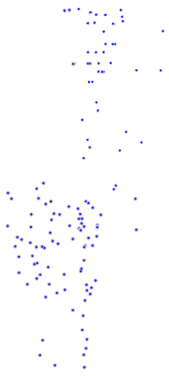


Figure 4.12: A roadside sign from KITTI that were identified as a pedestrian

Figure 4.13: Simulated data of a car that have not been identified

Figure 4.14: KITTI data of a car that have been identified

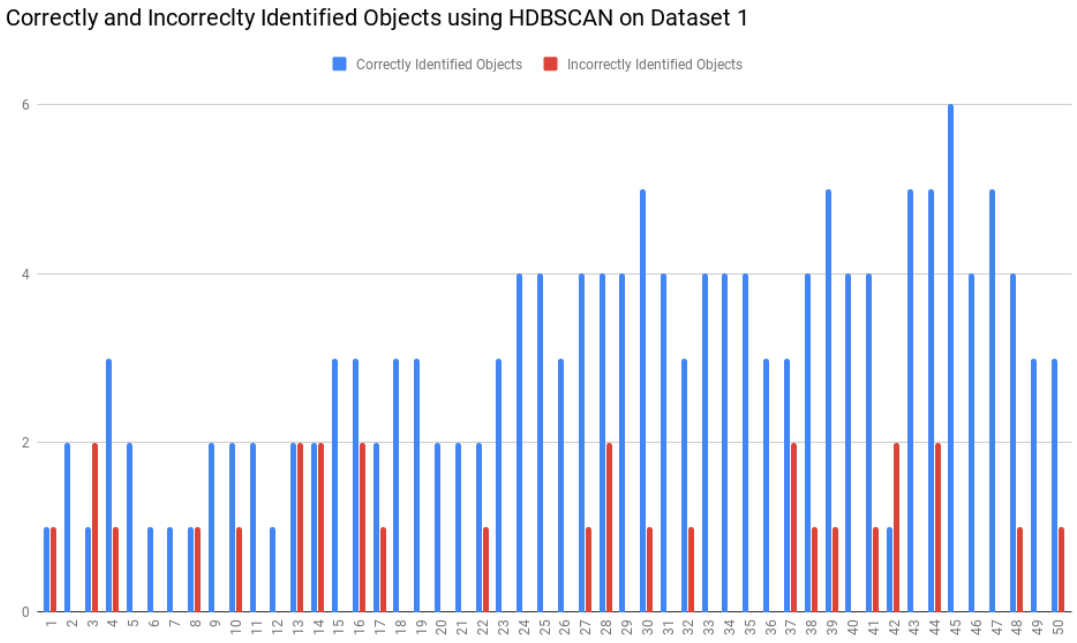


Figure 4.15: Number of object correctly and incorrectly identified by DELIS with NN

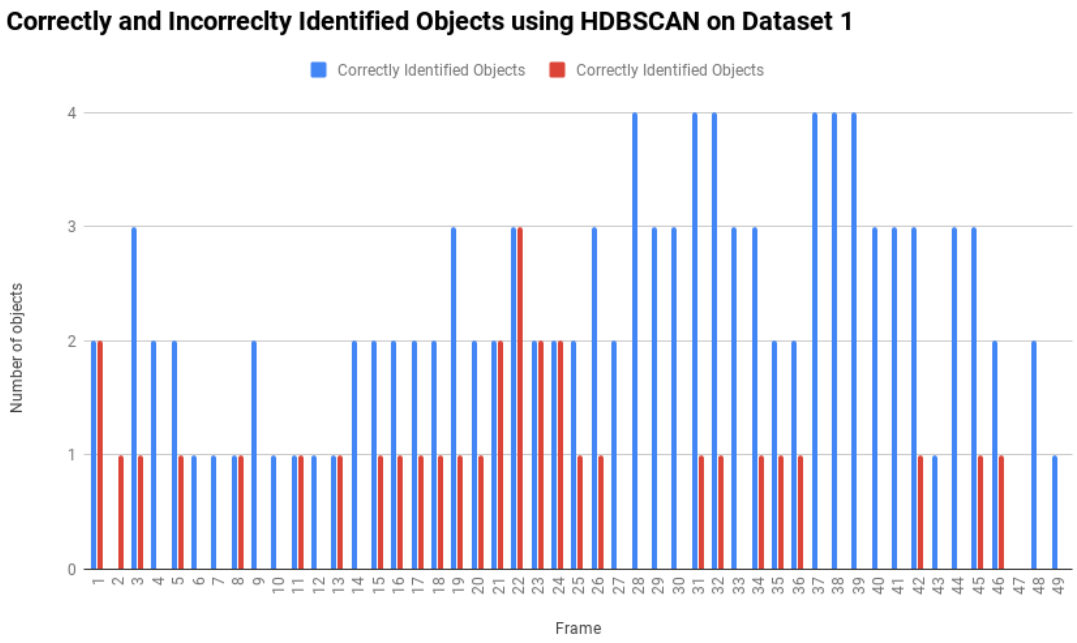


Figure 4.16: Number of object correctly and incorrectly identified by DELIS with HDBSCAN

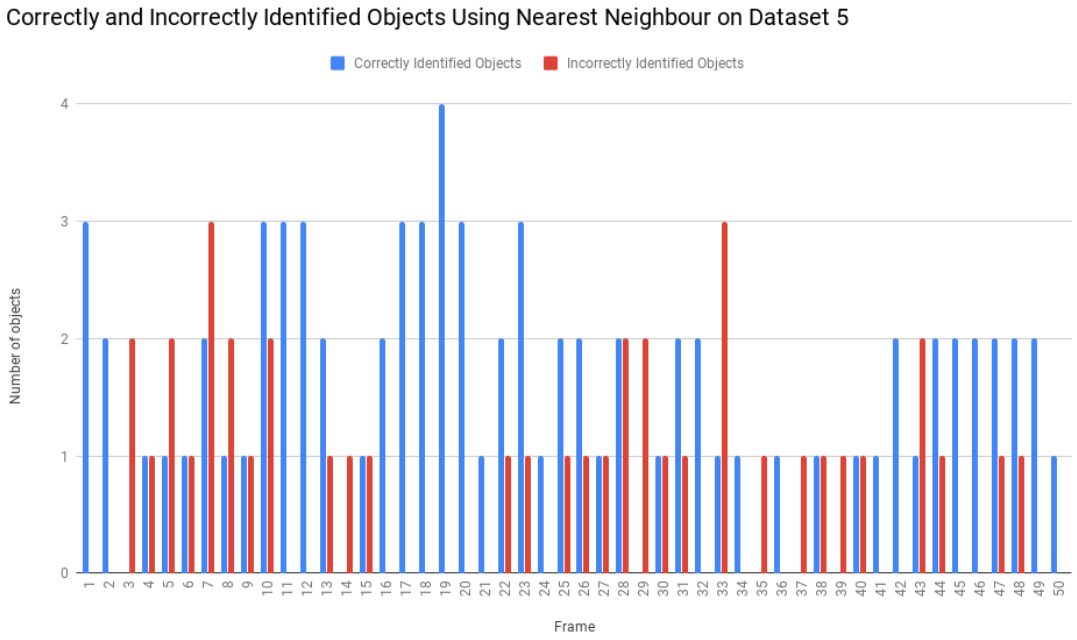


Figure 4.17: Number of object correctly and incorrectly identified by DELIS with NN segmentation

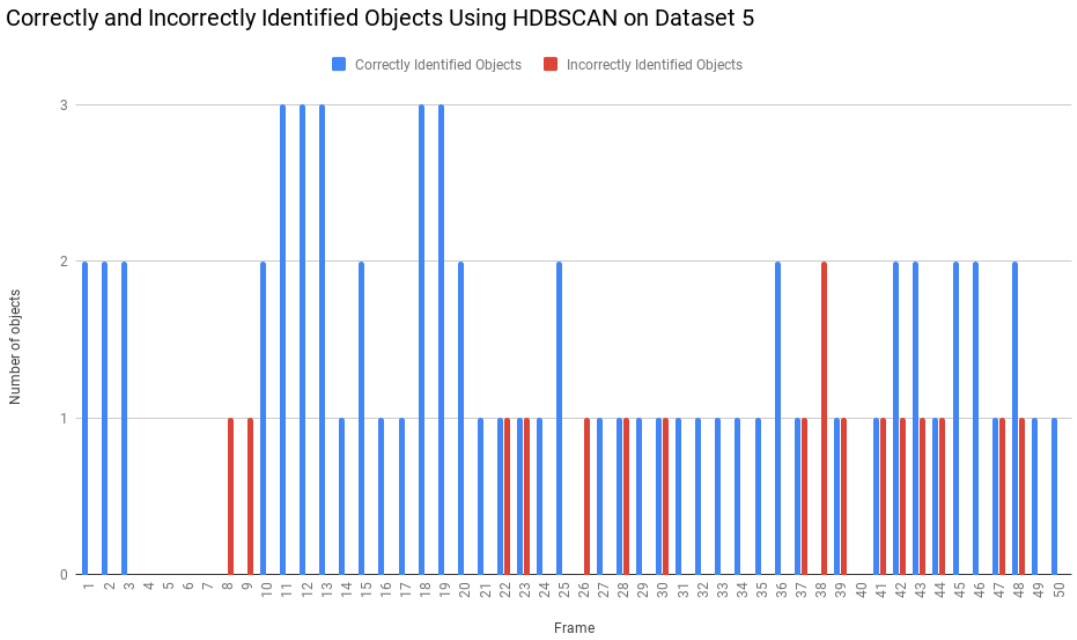


Figure 4.18: Number of object correctly and incorrectly identified by DELIS with HDBSCAN segmentation

5
Discussion

The first section discusses the segmentation. Next part concerns the classification and is then followed by a part about DELIS (DEtection In LIDAR Systems). The third part concerns comparison with similar systems that are already online in similar fields and also the answers to the research questions. The last part is about ethics and sustainability.

5.1 Segmentation

Segmentation of the point clouds was a greater challenge than first expected since the network was trained on pre-labeled objects, similar objects needed to be produced for the network input. The big problem with segmentation is to divide the points into clusters were the points belong to the same object. The thesis used the HDBSCAN[24] algorithm and nearest neighbor algorithm for segmentation with different results. Due to the computational cost of the nearest neighbor algorithm, the segmentation was very slow when it was used, the main bottleneck with the algorithm was the graph building part. A method used to speed up the algorithm was the KD-Tree, this lowered the cost of querying for neighbors. The removal of all points beyond a specified range and the cropping of the ground also helped to speed up the process since there are fewer points to process. Although cropping objects may be dangerous to do in real life environments since you can remove potholes or other obstacles. Also, the fact that it does not work if the ground is uneven is another disadvantage of this solution. Still, we considered it acceptable since we only consider scenes with a flat ground for this proof of concept.
The HDBSCAN were a lot faster which is a great advantage when it comes to real-time systems, however, the execution time for the HDBSCAN is still not fast enough for a real-time application. The truthfulness of the segmentation varied a lot as well, primarily depending on the point cloud used as input. In order to make the segmentation easier and faster, the ground was removed by a limit on the z-axis, this limit was found through experimentation. The experimentation approach is far from optimal and can likely be solved in a lot of ways. One suggestion is to sort the points in the cloud after the z-axis and just remove the lowest points. This solution ignores offsets of the point cloud in the z-axis. The drawback of this is the sorting time of the points, with large point clouds this might cause problems. However, this approach was not further developed due to that it was seen as outside of the scope of the thesis.
Since the segmentation part was a big problem and good segmentation is critical for

the network to be able to identify the objects, this problem should be investigated further. There are some existing papers that present methods for segmentation of point clouds. One of this possible alternative to our segmentation is *Difference of Normals as a Multi-Scale Operator in Unorganized Point Clouds* [23]. The paper proposes a method for segmenting unorganized point clouds by the use of surface normals created from the points in the cloud. With the use of two surface normals, calculated from surfaces with different sizes, both of the same point, changes can be noticed in the point cloud. This changes can be used to create clusters from the point cloud. The researchers in this paper managed to create promising results with this type of segmentation. This method was never implemented in our system, mainly because of time constraints. Furthermore, the segmentation part was not the primary goal of the thesis and was therefor not prioritized. However, this is something that could be interesting in investigating further.

5.2 Classification

When looking at the result of the evaluation of the ANN, see 4.4 it appeared as if the system often failed at differentiating between the classes pedestrians and misc. The cars were most often correctly labeled, the same for the cyclists.
The reason, for the low amount of correctly labeled misc, is that the system has interpreted a misc object as something else. In the KITTI data, there are a lot of objects that look like a car. Trucks and vans are some objects that are very similar to cars, that are in the misc class, this is one of the reasons for the misc class to be so low. The same goes for the pedestrians, in this case, there are lampposts and road signs that gets miss-classified. To improve the misc class it might have been good to put vans and trucks in the same category and evaluate on this new set. This might, however, lead to that the result for the car class gets worse.
Another problem could be the size of the training set which was limited in the number of pedestrians available in the point cloud data.
One interesting thing we discovered was the high result of the last ten epochs when disabling the second transform net. However, the results were a bit too unstable. Thus we decided to keep all of the transform nets, even though the execution time would increase.
One useful aspect of 3D data is that it contains depth, which is used in combination with RGB data in [49, 50]. It is shown that fusing images with LIDAR data can increase the performance of the system. One additional extension to this project would be to use this in order to further increase the accuracy of the system. However, the execution-time would probably get worse since the more nodes that are used the higher the execution-time becomes. On the other hand, there is a probability that the images would not require more nodes since they contain more data, which should make them easier to classify.
Another way to use both kinds of data is to convert the 3D point cloud into 2D images and train a neural network to identify the objects from these images as the researchers did in *Multi-view Convolutional Neural Networks for 3D Shape Recognition* [9]. The authors of *Volumetric and Multi-View CNN for Object Classification on 3D Data* [32] deem the multi-view approach to be a strong alternative to direct

classification of the point cloud. In that study the researchers compare a multi-view CNN with a volumetric CNN i.e., the data is encoded as a 3D tensor of binary values. They conclude that the limitation of volumetric CNN is the resolution of the point cloud. Although one downside with this multi-view approach is that these systems do not work well when it comes to 3D tasks such as point classification and shape completion, but also not well with scene understanding [10, 11].

One other way to classify objects, that originates from 2D data processing, is described in *Vote3Deep: Fast Object Detection in 3D Point Clouds Using Efficient Convolutional Neural Networks* [51]. In this paper they suggest to use the sliding window method but extend it to 3D, this is, however, a computationally expensive method although they argue that the empty space in the cloud would partially compensate for this.

Concerning the ANN, it worked mostly satisfactory. However, we speculate that by increasing the training set, primarily with pedestrians, the evaluation would be increased. Further, by also adding the items in the misc class that are most difficult for the system to label correctly, the misc class result would increase. This may be viewed in 4.11 and 4.12. Another possible way, that might increase the result would be to use a larger amount of data as training.

5.3 DELIS

The simulated data was problematic when used as input to the system, the segmentation worked well but the network had problems in identifying the cars. It is hard to say what the cause of this is, the first that comes to mind is that the data is too symmetrical i.e there is no noise in the data. This is also further discussed in future work, 6.1, to discover what the difficulty is. A limitation with the simulated data was that we had no ground truth for the objects, only whole point clouds. If we could retrieve the ground truth from this clouds somehow we could remove the segmentation as a factor in the identification. This would also make it possible to train a network of simulated data as previously discussed. It might also be possible that the ANN is to well train on the KITTI-data that it is unable to classify the data from the simulated environment. In order to improve the results, additional data obtained from the simulation are required.

As described in 2.3.2, there might occur problems when the PointNet is trained on dense data and then evaluated on sparse data, and the other way around. Since we got the simulator so late in the project, there was no possibility of creating enough training data to efficiently change the outcome.

An other way to improve the overall accuracy of the system could be to use sequences of frame to identify an object. If an object could be located in several frames the neural network would get multiple chances to identify it. This would greatly increase the probability that the the object where correctly identified. In our verification of the system we used sequences of frames and noticed that the sames object where identified in several sequential frames. If DELIS where able to use this the accuracy might improve. It might even be used to track and predict the path of the objects.

5.3.1 Comparison with Existing Systems

When it comes to comparisons between 2D pictures and 3D point clouds it is a difficult comparison to make. They both have their advantages and disadvantages as discussed earlier. System for 2D object detection is most common, one of this is YOLO. YOLO is interesting due to the speed it manages to identify objects. [6]. This speed comes at the cost of accuracy, the network only has an accuracy around 63.4 mAP, Mean Average Precision, on the other hand, it can analyze up to 45 FPS, frames per second. YOLO has been further developed into *Fast Yolo* which is even faster at 155 FPS, although its mAP has decreased to 52.7.
When it comes to 3D object detection, *VoxelNet* [7] is a system that analyses point clouds and identifies objects. The idea is to divide the 3D space into voxels and randomly sample a fixed number of points from each voxel. These points are then used as input for a chain of VFE layers (Voxel Feature Encoding), which are specifically designed to find features in the voxels.
With the help of these layers, 3D shape information can be gained, this information can then be further aggregated using 3D convolution. The last step is a Region Proposal Network that displays the detection result. The primary thing about VoxelNet is the ability to operate on sparse point clouds.

5.3.2 Research Questions and Their Answers

RQ1 How well does machine learning methods, trained on data from a real traffic environment, perform on simulated environments?

RQ2 Is it possible to use machine learning algorithms with classical algorithms in order to make a competitive system for object classification?

Answer to RQ1

It was problematic for an ANN to perform well on both simulated and real data due to the differences in the data between them. This can be seen in 4.13 and 4.14. The main difference between the two types was that the simulated LIDAR did not take into consideration any transparent surfaces such as the windshield on the car, this resulted in a point cloud of the car without any interior points. Another difference was the lack of noise in the simulated data, it is hard to say if the lack of noise made it harder for the network to classify objects. However you can see that it causes the artificial data to differ from the real world data, and this makes it harder for the neural network. One limitation that was discovered, when training a neural network, is to gather a sufficiently large dataset. This is probably due to the fact that the equipment, such as a LIDAR, is expensive. It is also time-consuming to manually label each object in the gathered point clouds, which is needed for the data to be useful in training a neural network. This problem disappears when a computer is used to generate the data. The number of point cloud that can be generated is nearly limitless and the labeling is done automatically in the program.

The disadvantage is that it is hard to generate realistic point clouds that are similar enough to real-world data that the neural network will not notice the difference. These problems will most likely solve them self with more advanced simulation programs.

Answer to RQ2

The segmentation of point clouds turned out to be more difficult than expected. It was often hard to separate an object from the surroundings, such as the ground or a building. This resulted in that point from different objects were often clustered together, e.g points belonging to the ground being clustered together with points from a car. Two different segmentation algorithms were implemented and tried, both with varying result. Perhaps if the training data for the ANN also would contain segmented data, like a car with some road under it, the classification would be more efficient and the accuracy would be higher. The problem with this is that the segmented data would need to be manually labeled in order for it to be use-full for training, and this would be very time-consuming.

5.4 Ethics and Sustainability

Since DELIS is supposed to be mainly in self-driving cars, this part will concern autonomous vehicles. Autonomous vehicles will most likely have a smaller impact on the environment in the forms of lower emissions due to the fact that these cars will find the optimal path to traverse and will be able to drive in a more eco-friendly. According to a study from NREL, National Renewable Energy Laboratory [52], these vehicles will either save nearly 90% of fuel, when only benefits occur. On the other hand, only the increase in energy are considered, the energy consumption will increase up to 250%. We believe that the engineers developing these systems will design them to be energy efficient.
One ethical dilemma that will arise is what all the professional drivers will do. If all the vehicles are autonomous, there is no need for professions like bus-, cab- or truck drivers. The same problem was faced during the industrial revolution. People thought the machines were going to take their jobs. It turned out the other way around. The machines needed someone to maintain them. Perhaps the need for more mechanics will increase when everyone has their own personal shuttle. On the other hand the opposite can happen where you might share your car with multiple people and as a result, there will be fewer cars on the road. It is always hard to predict about future tech but we believe that self-driven vehicles are inevitable and they are going to revolutionize our everyday lives.

6
Conclusion

In this project a system called DELIS (DEtection In LIDAR Systems) was created, a system capable to identify pedestrians, cyclists, and cars from its environment. Even though DELIS did not work as well as intended, primarily considering with time and when using simulated data, we have proven that an Artificial Neural Network, is capable of identifying objects from a LIDAR (Light Imaging, Detection And Ranging) that were clustered by either the Nearest Neighbor algorithm or the HDBSCAN-algorithm.
The problem with the simulated data was that it was very different from the real world data. One difference was that it was too uniform and would therefore immediately be identified as artificial. It might be possible to add noise to the simulated data, which would make it more real. Another difference was that the LIDAR sensor in the simulation was unable to identify transparent objects such as a windshield. This resulted in a car shape, seen in 4.13, but without the common characteristics of a car, such as an absence of points on the windshield as seen in 4.14. A way to solve this problem could be to add simulated cars and pedestrians to the training set. This was never done in the project and it is therefore hard to say what kind of accuracy could be achieved and what problems would arise.
The segmentation algorithms used in this thesis performed adequately but left room for improvements. Mainly the removal of ground points, which is currently just cropped at a z-coordinate. Due to the limitation of the segmentation, it is also difficult to get an exact value of the network accuracy from the simulated data and the impact it has on the final results.

6.1 Future Work

One way to continue this work would be to train a neural network with simulated data and validate with real-world data. This would save a lot of time and money since the automotive companies, and perhaps others, would not need to collect real data to train their networks. It would be possible to just let a computer generate an almost infinite amount of data.
Another future work to continue this project could be to change the segmentation, either by performing it in the neural network or by implementing some other algorithm.
Our initial thought was that this system should be a part of a larger system. Next part of that would be to track moving object over several frames. This could help to improve the classification accuracy by giving the network multiple, slight different

clusters, that the system would know is the same object.

References

[1] Shiyu Liang and R. Srikant. Why deep neural networks? *CoRR*, abs/1610.04161, 2016.

[2] Farzin Amzajerdian Diego Pierrottet; Larry Petway Glenn Hines, Vincent Roback. Lidar systems for precision navigation and safe landing on planetary bodies. *NASA. Technical Reports*, 2011.

[3] Paul Babyn Jimmy Wang Gary Groot Mark Eramian Jianning Chi, Ekta Walia. Thyroid nodule classification in ultrasound images by fine-tuning deep convolutional neural network. *"Journal of Digital Imaging"*, Volume 30(4):477–486, 2017.

[4] Christoph Stiller Andreas Geiger, Philip Lenz and Raquel Urtasun. Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[5] Esi pro-sivic(tm) 2016: Virtually test sensors in ultra-realistic 3D scenes. `https://search-proquest-com.proxy.lib.chalmers.se/docview/1803543814?accountid=10041`, 2016. Accessed 2017-12-14.

[6] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[7] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3D object detection. *CoRR*, 2017.

[8] Luca Caltagirone, Samuel Scheidegger, Lennart Svensson, and Mattias Wahde. Fast lidar-based road detection using fully convolutional neural networks. *CoRR*, abs/1703.03613, 2017.

[9] Evangelos Kalogerakis Erik G. Learned-Miller Hang Su, Subhransu Maji. Multi-view convolutional neural networks for 3D shape recognition. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 945 – 953, 2015.

[10] Kaichun Mo Leonidas J. Guibas Charles Ruizhongtai Qi, Hao Su. Pointnet: Deep learning on point sets for 3D classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[11] E. Kim and G Medioni. Urban scene understanding from aerial and ground lidar data. *Machine Vision and Applications*, 22(4):691–703, Jul 2011.

[12] Matterport. Matterport pro2 3D camera specifications. `https://support.matterport.com/hc/en-us/articles/115004093167`, 2017. Accessed 2017-12-13.

[13] MathWorks. 3-d point cloud processing. `https://se.mathworks.com/help/vision/3-D-point-cloud-processing.html`, 2017. Accessed 2017-12-11.

[14] PointClouds.org. PCLVisualizer. http://pointclouds.org/documentation/tutorials/pcl_visualizer.php. Accessed 2018-01-22.

[15] Dong Ho Yun, Sung In Choi, Sung Han Kim, and Kwang Hee Ko. Registration of multiview point clouds for application to ship fabrication. *Graphical Models*, 90(Supplement C):1 – 12, 2017.

[16] R.H. Huesman A. Sitek and G.T. Gullberg. Tomographic reconstruction using an adaptive tetrahedral mesh defined by a point cloud. *IEEE Transactions on Medical Imaging*, 25:1172 – 1179, 2006.

[17] PointClouds.org. The pcd (point cloud data) file format. Accessed 2018-01-26.

[18] Paul McManamon. *Field Guide to Lidar*. SPIE, 2015.

[19] LeddarTech Inc. Leddarvu, 8-segment solid-state lidar sensor modules. https://leddartech.com/app/uploads/dlm_uploads/2017/05/Spec-Sheets-LeddarVu-4decembre2017-web.pdf, 2017. accesssed 2018-01-22.

[20] S. Weiming M. Yuda, Z. Xiangjun and L. Shaofeng. Target accurate positioning based on the point cloud created by stereo vision. *23rd International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, pages 1 – 5, 2016.

[21] Ankit Sharma, A.K. Jha, and Arpan Halder. Layout optimization of a robotic cell for foundry application by cad based point cloud modeling – a case study. *Industrial Robot: An International Journal*, 44(6):788–797, 2017.

[22] E. Grilli, F. Menna, and F. Remondino. A review of point clouds segmentation and classification algorithms. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W3:339–344, 2017.

[23] Yani Ioannou, Babak Taati, Robin Harrap, and Michael A. Greenspan. Difference of normals as a multi-scale operator in unorganized point clouds. *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission*, pages 501 – 508, 2012.

[24] Yang Xi Lingjuan Li. Research on clustering algorithm and its parallelization strategy. *2011 International Conference on Computational and Information Sciences*, pages 325 – 328, 2011.

[25] D. T. Lee and C. K. Wong. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica*, 9(1):23–29, Mar 1977.

[26] Omais Shafi Himani Raina. Analysis of supervised classification algorithms. *International Journal of Scientific & Technology Research*, pages 440 – 443, 2015.

[27] C. H. Lampert, M. B. Blaschko, and T. Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2008.

[28] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2014.

[29] Subana Shanmuganathan and Sandhya Samarasinghe. *Artificial Neural Network Modelling*. Springer International Publishing, 2016.

[30] J.C. Bioch ; W. Verbeke ; M.W. van Dijk. Neural networks: New tools for data analysis ? *Workshop on Neural Network Applications and Tools*, pages 29 – 38, 1993.

[31] N.A. Khovanova T. Shaikhina. Handling limited datasets with neural networks in medical applications: A small-data approach. *Artificial intelligence in medicine*, 2017.

[32] Charles Ruizhongtai Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J. Guibas. Volumetric and multi-view cnns for object classification on 3D data. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5648 – 5656, June 2016.

[33] Yangyan Li, Sören Pirk, Hao Su, Charles Ruizhongtai Qi, and Leonidas J. Guibas. FPNN: field probing neural networks for 3D data. *CoRR*, abs/1605.06240, 2016.

[34] Harshad Kumar Dharamshi Hansraj Bhadeshia. Neural networks in materials science. *ISIJ International*, 39(10):966–979, 1999.

[35] Hamed Habibi Aghdam and Elnaz Jahani Heravi. *Guide to Convolutional Neural Networks*. Springer International Publishing, 2017.

[36] Jonathan Masci Luca M. Gambardella Jurgen Schmidhuber Dan C. Ciresan, Ueli Meier. Flexible, high performance convolutional neural networks for image classification. *International Joint Conference on Artificial Intelligence IJCAI-2011*, pages 1237–1242, 2011.

[37] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-cnn: Octree-based convolutional neural networks for 3D shape analysis. *ACM Trans. Graph.*, 36(4):72:1–72:11, July 2017.

[38] Anish Singh Walia. Types of optimization algorithms used in neural networks and ways to optimize gradient descent. `https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-` 2017. Accessed 2018-01-30.

[39] Ohad Shamir and Tong Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. *CoRR*, abs/1212.1824, 2012.

[40] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[41] Geoffrey Everest Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[42] Chritian Günther H. M. W. Verbeek H. M. W. Verbeek Wil van der Aalst, Vladimir Rubin. Process mining: a two-step approach to balance between underfitting and overfitting. *Software and Systems Modeling*, 9(1):87–111, 2010.

[43] Matterport 3D media platform demo. https://vimeo.com/117514038. Accessed 2017-12-13.

[44] ESI Group 2018. Esi pro-sivic™ - 3D simulations of environments and sensors. `https://www.esi-group.com/software-solutions/virtual-environment/virtual-systems-controls/esi-pro-sivictm-3D-simulations-environments-and-sensors`, 2018. Accessed 2018-01-09.

[45] Velodyne Lidar. Hdl-64e. `https://velodynelidar.com/hdl-64e.html`. Accessed 2018-01-30.

[46] Martin Schrimpf. Should I use tensorflow? *CoRR*, abs/1611.08903, 2016.

[47] Hong-Phuc Trinh, Marc Duranton, and Michel Paindavoine. Efficient data encoding for convolutional neural network application. *ACM Trans. Archit. Code Optim.*, 11(4):49:1–49:21, 2015.

[48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[49] Joel Schlosser, Christopher K. Chow, and Zsolt Kira. Fusing lidar and images for pedestrian detection using convolutional neural networks. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2198–2205, 2016.

[50] Andreas Eitel, Jost Tobias Springenberg, Luciano Spinello, Martin A. Riedmiller, and Wolfram Burgard. Multimodal deep learning for robust rgb-d object recognition. *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 681–687, 2015.

[51] Dominic Zeng Wang Chi Hay Tong Ingmar Posner Martin Engelcke, Dushyant Rao. Vote3deep: Fast object detection in 3D point clouds using efficient convolutional neural networks. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1355–1361, 2017.

[52] Jeff Gonder Austin Brown, Brittany Repac. Autonomous vehicles have a wide range of possible energy impacts. `https://www.nrel.gov/docs/fy13osti/59210.pdf`, 2013. Accessed 2018-01-30.

A

Appendix: Randomized networks

This chapter contains some of the networks that were evaluated in chapter 3. mp means max pooling, tf is a transform net. c is a convolutional layer and fc is a fully connected layer. The first number indicates how many nodes that are in that layer. Time is the average validation time for the last ten epochs during validation. Accuracy is the average accuracy for the last ten epochs. Every net were trained for 100 epochs. This is only a sample of all the NNs that were evaluated, since the total of NNs were to many to put in here.

Architecture	Time	Accuracy	Architecture	Time	Accuracy
$[expand]$ $[mp, 1]$ $[tf, 1]$ $[c\_trans1, 32.0, 2]$ $[c, 1024.0, 3]$ $[c, 64, 4]$ $[tf, 2]$ $[c, 256.0, 5]$ $[c, 64.0, 6]$ $[c, 64.0, 7]$ $[fc, 64.0, 1]$ $[fc, 64.0, 2]$ $[fc, 512.0, 3]$ $[do, 1]$ $[fc, 256.0, 4]$ $[fc, 4, 5]$	0.87166	0.72148	$[tf, 1]$ $[c\_trans1, 128, 2]$ $[c, 512, 3]$ $[c, 64, 4]$ $[tf, 2]$ $[c, 124, 5]$ $[mp, 1]$ $[fc, 512, 1]$ $[do, 1]$ $[fc, 16, 2]$ $[do, 2]$ $[fc, 124, 3]$ $[do, 3]$ $[fc, 512, 4]$ $[do, 4]$ $[fc, 32, 5]$ $[do, 5]$ $[fc, 16, 6]$ $[fc, 64, 7]$ $[fc, 4, 8]$	0.91456	0.79238
$[tf, 1]$ $[c\_trans1, 256, 2]$ $[c, 124, 3]$ $[mp, 1]$ $[c, 64, 4]$ $[tf, 2]$ $[c, 256, 5]$ $[c, 64, 6]$ $[fc, 16, 1]$ $[do, 1]$ $[fc, 64, 2]$ $[fc, 4, 3]$	0.37595	0.76387	$[tf, 1]$ $[c\_trans1, 128, 2]$ $[c, 124, 3]$ $[c, 64, 4]$ $[tf, 2]$ $[c, 128, 5]$ $[fc, 124, 1]$ $[fc, 128, 2]$ $[fc, 512, 3]$ $[fc, 16, 4]$ $[fc, 4, 5]$	0.90975	0.71152
$[tf, 1]$ $[c\_trans1, 256, 2]$ $[c, 124, 3]$ $[mp, 1]$ $[c, 64, 4]$ $[tf, 2]$ $[c, 256, 5]$ $[c, 64, 6]$ $[fc, 16, 1]$ $[do, 1]$ $[fc, 64, 2]$ $[fc, 4, 3]$	0.61580	0.81445	$[tf, 1]$ $[c\_trans1, 64, 2]$ $[c, 64, 3]$ $[tf, 2]$ $[c, 64, 4]$ $[fc, 512, 1]$ $[fc, 128, 2]$ $[fc, 128, 3]$ $[fc, 4, 4]$	0.59992	0.71934