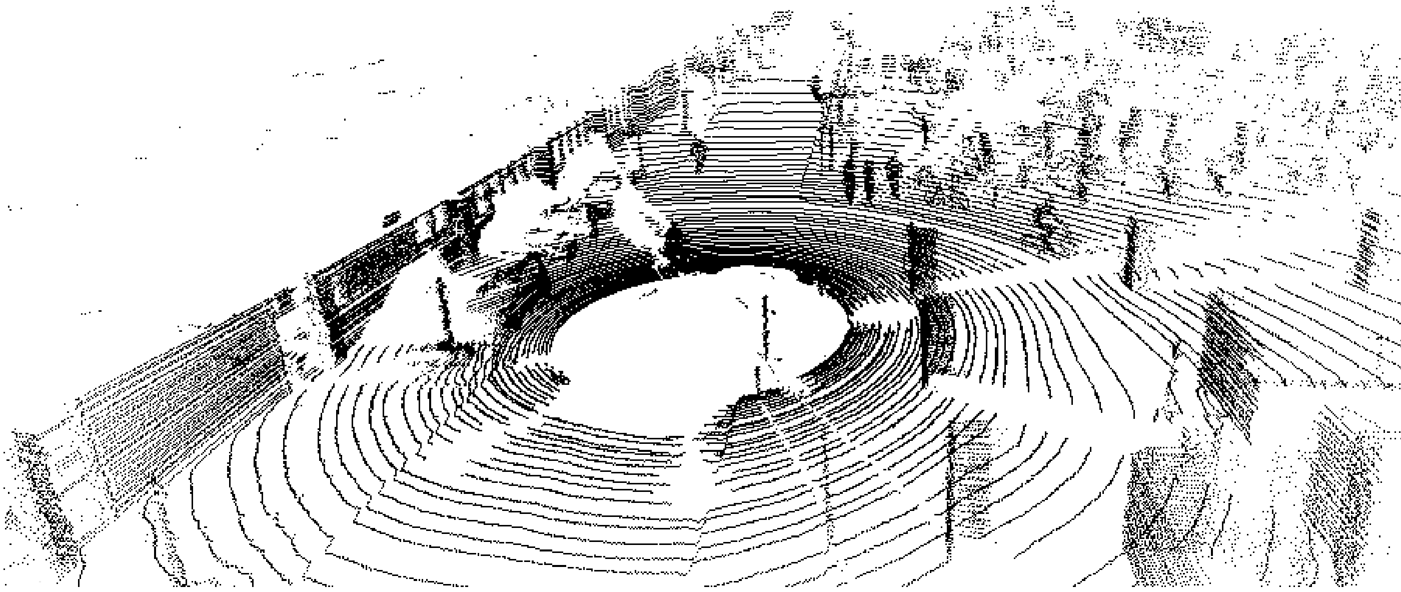


A Comparative Study of Segmentation and Classification Methods for 3D Point Clouds

Master's thesis

PATRIK NYGREN and MICHAEL JASINSKI

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2016

A Comparative Study of Segmentation and Classification Methods for 3D Point Clouds

Patrik Nygren and Michael Jasinski

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY and
UNIVERSITY OF GOTHENBURG
Göteborg, Sweden 2016

A Comparative Study of Segmentation and Classification Methods for 3D Point Clouds

PATRIK NYGREN, MICHAEL JASINSKI

Supervisor: Marco Fratarcangeli, Department of Applied Information Technology
Examiner: Graham Kemp, Department of Computer Science and Engineering

Master's Thesis 2016
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 taken from a street with a Velodyne laser scanner.

Typeset in LaTeX

Abstract

Active Safety has become an important part of the current automotive industry due to its proven potential in making driving more joyful and reducing number of accidents and causalities. Different sensors are used in the active safety systems to perceive the environment and implement driver assistance and collision avoidance systems. Light detection and ranging (LIDAR) sensors are among the commonly utilized sensors in these systems; a LIDAR produces a point cloud from the surrounding and can be used to detect and classify objects such as cars, pedestrians, etc. In this thesis, we perform a comparative study where several methods to both segment *Region Growing* and *Euclidian Clustering*) and classify (*Support Vector Machines*, *Feed Forward Neural Networks*, *Random Forests* and *K-Nearest Neighbors*) point clouds from an urban environment are evaluated. Data from the KITTI database is used to validate the methods which are implemented using the PCL and Shark library. We evaluate the performance of the classification methods on two different sets of developed features. Our experiments show that the best accuracy can be obtained using SVMs, which is around 96.3% on the considered data set with 7 different classes of objects.

Keywords: machine learning, neural networks, support vector machines, random forest, k-nearest neighbours, segmentation, classification, features, point cloud.

Acknowledgements

We would mainly like to thank our supervisor at Volvo, Nasser Mohammadiha, for all prestigeless discussions we had with him, for the help he gave us writing our proposal, and for all valuable input he gave us throughout the work. We would also like to thank our supervisor at Chalmers, Marco Fratarcangeli, and our examiner Graham Kemp, both of whom always found the time to help us whenever we felt lost. An additional thanks to Jonathan Ahlstedt and Johan Villysson for their guidance. Finally, we would like to thank the community behind PCL and Shark for providing the resources that made this work possible.

Patrik Nygren and Michael Jasinski, Gothenburg, June 2016

Contents

List of Figures

List of Tables

1 Introduction

The car manufacturers of today compete for the consumers by providing *Active Safety* systems that prevent accidents without the need of the driver's control. As car manufacturers incorporate more and more computer systems into cars, the competitive advantages of Active Safety functions such as adaptive cruise control and collision avoidance systems are growing in importance. According to a study [1] carelessness and negligence of the driver is not the major reason for car accidents but the drivers inert perception, resulting in a delayed response to rapidly changing traffic conditions. Nearly 1.3 million people die in road accidents globally each year, and on average 3,287 deaths occur each day [2].

Active safety systems utilize many sensors to produce data. One such sensor is LIDAR (Light Detection and Ranging), an optical remote sensing technology which produces point clouds of the surroundings. The LIDAR is comprised of a number of spinning laser beams and is usually mounted on the roof of a car. The LIDAR spins 360 degrees around its axis while sending out laser beams. The reflection of each laser beam is registered in the sensor which measures the light intensity and distance to each reflected point. The final product is a point cloud image that is produced for each rotation of the sensor (see front page).

In order to extract information from the point cloud it normally goes through a segmentation and classification process. The segmentation algorithm is used to cluster different points of the point cloud to smaller clusters according to some similarity criterion. These smaller point cloud clusters are subsequently labelled by a trained classifier into different categories, such as cars, pedestrians, etc. The final result is a point cloud with labeled objects. This offline result finally provides an accuracy measure that can be compared with the cars online sensor accuracy, to establish a threshold that can act as metric of the online sensor's performance.

1.1 Purpose

The aim of this thesis is to implement and evaluate a segmentation-classification pipeline for classifying objects in urban environments and compare different methods with respect to their accuracy. Figure 1.1 shows an overview of the pipeline.

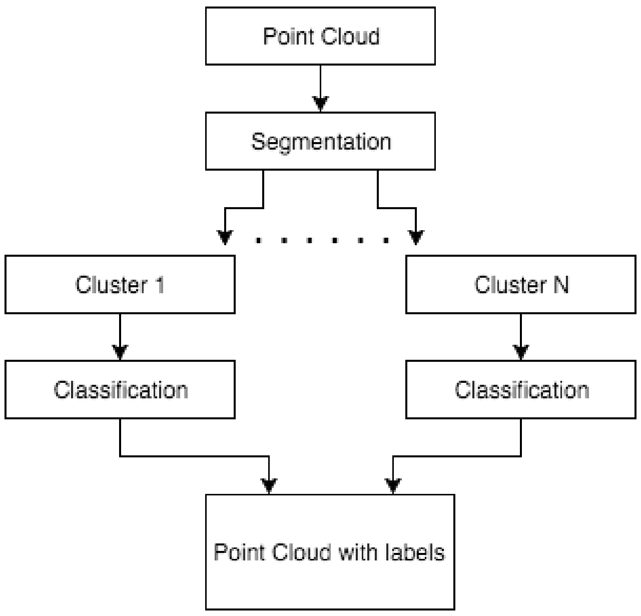


Figure 1.1: *The input to the pipeline is a sparse point cloud. The point cloud is segmented into different objects referred to as clusters, 1 to N. The clusters are then classified and the predicted class label is assigned to each cluster. Lastly, the original point cloud is visualized, showing the relevant objects in bounding boxes and their respective label.*

To enable a feasible study the performance of four different classifiers are compared, namely *Support Vector Machines* (SVM), *Feed Forward Artificial Neural Networks* (FFNN), *Random Forests* (RF), and *K-Nearest Neighbours* (KNN). A comparative study of two Segmentation methods, *Region Growing* and *Euclidean Clustering* is made. In addition to comparing classifiers two different feature sets (Feature set A and Feature set B) will be composed to evaluate how the Classifiers performance gets affected by changing their input.

The following classes are the subjects for the classification:

1. Cars
2. Trucks
3. Vans
4. Pedestrians
5. Bicyclists / Motorcyclists
6. Signs / Poles
7. Unknowns

1.2 Limitations

The accuracy of the segmentation algorithms will be evaluated visually and will be considered satisfactory if it manages to segment the mentioned object classes in section 1.1. This thesis provides the comparison of two different types of segmentation algorithms, Euclidian clustering and Regional growing, both of which are implemented in PCL[3].

The implementation and evaluation of the classification is limited to comparing four different discriminative machine learning algorithms, namely SVM, FFNN, RF and KNN. The literature presents another type of algorithms that will not be considered here, namely generative algorithms. The main difference between these types of algorithms will not be covered in great detail in this thesis, the reader is instead referred to the literature. However, according to [4] discriminative classification models are almost always preferred to generative ones, since they solve the classification problem directly instead of solving a more general problem such as modeling the probability distributions for the classes.

1.3 Thesis Outline

The thesis is divided into five main chapters. The **Background** gives an overview of used methods in the field of urban point cloud segmentation and classification. The used methods gets a deeper explanation in **Theory**, where the algorithms in focus will be explained to the reader. An explanation of the process of the work and the methods used for evaluating the algorithms are given in **Implementation** and also some details regarding the segmentation-classification pipeline implementation. **Results** accounts for the measured results of the pipeline, and how the different segmentation and classification methods performed. **Discussion** reflects over the results and provides some thoughts on possible future work. The major findings in the thesis are finally presented in **Conclusion**.

2
Background

This chapter aims to give some background to the general problems that exist in urban point cloud segmentation and classification. Moreover, it aims to give some insight into related segmentation approaches from the literature and an understanding of how different combinations of classification algorithms, feature vectors and class types may affect the final classification accuracy.

2.1 Object segmentation

When performing segmentation on a sparse point cloud the main problems that arise are: *The under-segmentation problem* (how to handle spatially close objects), and *the over-segmentation problem* (how to handle objects inherent surface variations). When sections of the point cloud that represent individual objects are too close it becomes significantly more difficult to segment these correctly since they appear to be one object. The opposite problem arise when a single object can be perceived as several objects. The main reason for these problems comes from the fact that sparse point clouds represent a small sample of all possible points and hence results in an information loss. Different methods address these problems in different ways. In general segmentation methods can be divided into three approaches: Model fitting, Region growing, and Clustering [5]. Since Model fitting combines segmentation and classification in one step and the scope of the thesis is to evaluate segmentation and classification algorithms as separate steps the reader is referred to the literature for further details about Model fitting.

2.1.1 Region Growing

Region growing methods start by using random seed points and compares them to neighbouring points based on a similarity criterion, if they are similar according to the criterion they get merged into the segment otherwise they get rejected and can later be incorporated in other segments. The algorithm continues to grow the region this way to form a segment. Although Region growing algorithms can be used successfully for segmentation and are generally fast [6] the random choice of seed points usually results in different segmentations and are therefore not considered as robust as others methods [7]. In [8] they use Region growing to segment building features from a facade. The similarity criterion used in this work was proximity and perpendicular distance. This way walls, windows, doors, etc. could be satisfactory segmented since these classes shift in perpendicular distance compared to each other.

2.1.2 Clustering

Similar to Region growing, clustering relies on similarity criteria for subdivisions to be grouped. The simpler clustering techniques use a spatial similarity criterion such as the Manhattan distance or the Euclidean distance [9]. In [10] Euclidean clustering is used to find objects placed on a table in a robotic context. The segmentation was done after ground removal (removal of the table). Focusing the segmentation to certain regions of the point cloud the performance varied significantly depending on whether several objects of interest were included in that region or not, with better accuracy when only a single object of interest was included.

2.1.3 Noise Filtering and Ground removal

A typical pre-processing step before segmentation is to extract the ground from the point cloud in order to focus the segmentation to objects of interest and not ground segments. Example of such work include [11], where objects are derived after the ground has been removed. Furthermore, it has been shown that removing noise by doing statistical outlier filtering before the ground removal increases the performance of the ground removal [12]. Noise filtering also allows for a more accurate segmentation later, and therefore for a more accurate classification since the features that describe the object can be more precise.

According to [13], the *RANdom SAmple Consensus* (RANSAC) algorithm is a common method used to detect planes in point clouds applications. Due to its robustness to outliers RANSAC is a popular model fitting method that was presented in [14]. Given a geometrical template (i.e. model) RANSAC randomly samples a subset of points and computes the model parameters of each subset. The best fitted model is decided on the estimated parameters that contain the largest number of inliers (points). One of the challenges applying RANSAC consist of determining the "inliers threshold", i.e the variance allowed in the model for points to be considered belonging to the model [5].

2.2 Classification

A set of discriminative features in the point cloud needs to be chosen that either describe the object locally or globally in order to classify point cloud clusters. In a general 3D point cloud context there are two basic alternatives for labeling data. The first alternative is to use a discriminative 3D point feature descriptor/vector and train a machine learning algorithm to predict the correct labels for different classes of objects. The second alternative is to use geometric reasoning techniques to fit geometric primitive shapes (i.e cylinders, spheres, etc) to the data. Generally, the former approach will outperform the latter geometric approach in most cases [9].

One important question using the former method is the choice of feature vector, such that representative but also discriminative information is extracted from the point cloud. The goal is to find features that are similar for objects in a given class and that vary as much as possible between different classes.

2.2.1 Feature extraction

Whichever classification algorithm is used, the results of the classification will in almost all cases depend on what type of features are used. Features can be described as the attributes of an object, e.g. the volume or the height of an object. The goal is to find features that are similar for objects in a given class and that vary as much as possible between different classes.

There are a number of different approaches to feature extraction in urban environments. For example, in [15] a total of 15 different geometrical features are used to classify pedestrians from non-pedestrians, while [16] use a feature vector of 28 features, including various intensity features, to classify cars from non-cars. Three of the features used by the latter authors include eigen-features computed by performing a principal component analysis (PCA).

2.2.2 Support Vector Machines

A common technique in point cloud classification is to use Support Vector Machines (SVM). Given a set of training examples, the SVM algorithm attempts to find an optimal separation of sets of training examples belonging to the same class. In [17] SVM is used to classify different point clouds according to two data sets, of which one contains: cars, lampposts, lights, posts, and trees, another data set contains: cars, poles and trees. In the work they combined geometrical features with contextual features, where contextual features is defined as neighbouring regions touching the object. They recorded a best overall accuracy of 88% on the latter set. On a different data set of 20 classes they achieved a best overall accuracy of 82%. In [18] they use SVM to classify: cars, pedestrians, cyclists, poles and unknown objects with an overall accuracy of 96%.

2.2.3 Artificial Neural Networks

Artificial Neural Networks relies on a network of nodes, where each node, given an input, performs a computation with some learned parameters. During training all the nodes computations result in a prediction, and depending on the error rate the nodes can refine their individual parameters through a process of backpropagation. [19] uses a Feedforward Artificial Neural Network (FFNN) implementation to classify the following objects in a point cloud: vehicles, people, tree trunks, light poles and buildings. The features used in this work is described in section 4.4.2. With this setup they are able to achieve an accuracy of 94.61%.

2.2.4 Random Forests

Random forest (RF) is a technique that build up multiple decision trees during training. Each tree is based on a random subset of the data where the decision nodes are randomly picked from the feature set. Each node in the tree forms a binary decision about an individual feature on an example that categorises the example and continues categorising until a leaf is reached, where the leafs represent unique classes. Each tree gets to make a vote on an example and the class that gets the majority vote decides the class. The authors of [20] use RF to classify a point cloud captured from an airborne LIDAR sensor. The following classes are subjects for classification: natural ground, artificial ground, buildings, and vegetation. Height features, eigen-features, local plane features and full-waveform LIDAR features are used in this work to classify objects. They achieved an overall accuracy of 94.35%.

2.2.5 K-Nearest Neighbours

K-Nearest Neighbours (KNN) is a technique that in contrast to the other mentioned classification methods does not produce a model after training. It simply stores all the examples in memory and picks out the K neighbours that lie closest to the example in question in the feature space. Similar to RF it lets all the neighbouring training examples make a vote on the class and classifies the example based on the majority vote. In [21] they use KNN to classify a point cloud. The classes are: cars, poles, trees and walls. In the work they compare eigen-features applied locally, with some other local shape descriptors such as spin images, and a global descriptor, i.e a descriptor that describes the whole point cloud, called spherical harmonic descriptor. They achieved an overall accuracy of 92.1%.

3 Theory

This chapter will give the reader a deeper explanation of the algorithms used in the implementation of the pipeline. The details of each algorithm will be limited to the relevance of this work. First, the two main segmentation algorithms will be explained on a high level, thereafter the classification algorithms will be explained in some detail.

3.1 Region growing

For surface based segmentation methods two approaches are possible: bottom-up and top-down [6]. Bottom-up approaches start from some seed-point/points and grow the segments from these by comparing these seed points to new candidate points, based on a given similarity property that acts as a threshold. If the candidate points gets incorporated into the segment they become new seed points. Top-down does the opposite by first assigning all points to a segment and then trying to fit a surface to it and as long as the figure to fit does not comply to a certain threshold it keep subdividing the segment. Region growing is an instance of the former where the algorithms grows regions (i.e segments) from a seed point by using a similarity property (i.e smothness, color etc), which determines whether the candidate point belong to the cluster or not [9].

3.2 Euclidean clustering

Euclidean clustering, in the context of point clouds, processes points by searching for the nearest points to a candidate point according to a distance threshold, and clusters points together as long as they fall inside the range of the threshold. The distance threshold, as the name implies, is simply the euclidean distance $d = \sqrt{x^2 + y^2 + z^2}$ between neighboring points. A common way to find the neighbors of a point is to construct a *KD-tree* [9]. A KD-tree is a data structure that separate the dimension space to k sub-dimensions which helps to organize and localize the nearest points to a certain point.

3.3 Support Vector Machines

Support Vector Machines (SVMs) are generalizations of the more simple *maximal margin classifier* [22]. The basic idea of SVMs is to separate classes in the training data by hyperplanes. The hyperplanes then act as decision boundaries for unlabeled examples. Since SVM only can do binary classification, *one-versus-one* or *one-versus-all* is often used on multiclass problems.

3.3.1 Linear SVMs

Linear SVMs does not apply to all data sets since they require that the training examples in the data set are linearly separable by some hyperplane so that each division is able to isolate a class, as seen in figure 3.1.

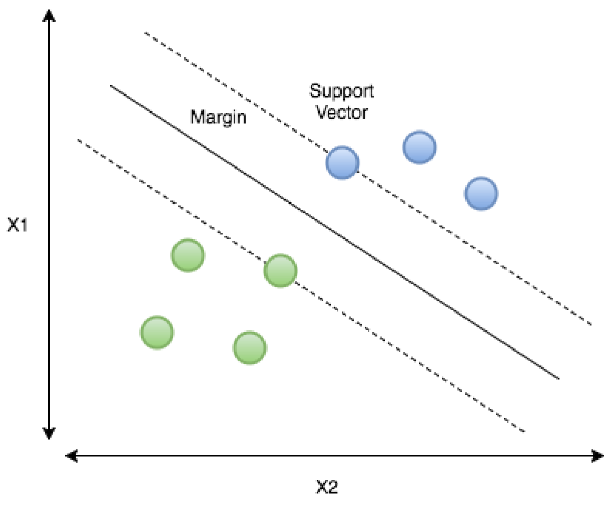


Figure 3.1: *Two separable classes with two features. The hyperplane is illustrated by the solid line, while the distance to a dashed line represents the margin. The circles represent training examples and the ones that lie on the dashed lines are the support vectors.*

A hyperplane is defined as: $\beta_0 + \beta_1 x_1 + ... + \beta_n x_n = 0$. Depending on if $\beta X < 0$ or $\beta X > 0$ (the sign of the inner product) the class of X is decided. The data examples that lie closest to the hyperplane are called *support vectors* since they support the maximal margin to the hyperplane (the descision boundary). The maximal margin classifier chooses the hyperplane by picking the one that maximizes the margin between the classes; this way the method reformulates the classification problem to the following optimization problem:

$$\underset{\beta_0,\ldots,\beta_n}{\text{maximize}}\, M \tag{3.1}$$

$$\text{subject to} \sum_{j=1}^{n} \beta_j^2 = 1 \tag{3.2}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_n x_{in}) \geq M, \,\forall i = 1, \ldots, m, \tag{3.3}$$

where M is the margin and the class labels $y_i, ..., y_n \in \{-1, 1\}$. Equation 3.1 is the *objective function* that maximizes the margin, and the two following equations, 3.2 and 3.3, are the *constraints*. The first constraint enforces the solution to be a hyperplane, and the last constraint in the optimization problem guarantees that each example is on the correct side of the hyperplane.

3.3.2 Linear SVMs with soft margin

In a training data set with outliers the maximal margin classifier may overfit (see figure 3.2) and with non-separable examples it may even be unable to create a linear decision boundary (see figure 3.3).

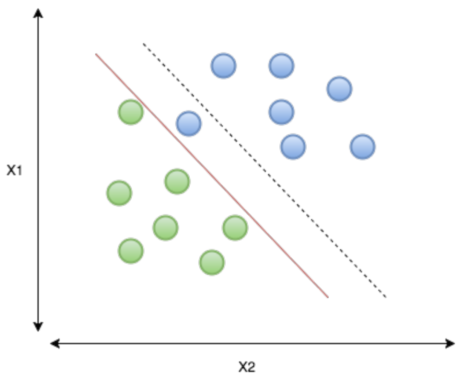


Figure 3.2: *Case of overfitting*

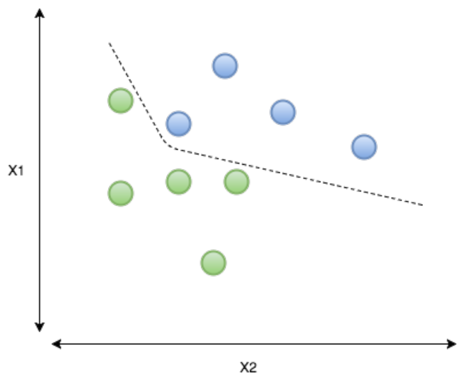


Figure 3.3: *Case of non-linear separable classes*

To alleviate the problem the *Linear SVMs with soft margin* may ignore some examples to achieve a better generality and a higher confidence, therefore it is sometimes referred to as a *Soft Margin Classifier*. In order to allow some missclassifications the optimization problem is reformulated:

$$\underset{\beta_0,\dots,\beta_n,\epsilon_0,\dots,\epsilon_n}{\text{maximize}} \; M$$

$$\text{subject to} \sum_{j=1}^{n} \beta_j^2 = 1$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \dots + \beta_n x_{in}) \geq M(1 - \epsilon_i),\; \epsilon_i \geq 0,\; \sum_{i=1}^{n} \epsilon_i \leq C,$$

where $\epsilon_0, \dots, \epsilon_n$ are slack variables that allow some examples to be on the wrong side of the hyperplane. If, for instance, $\epsilon_i > 0$ for the ith training example, the example ended up violating the margin boundary in the solution, and if $\epsilon_i > 1$ it even ended up on the wrong side of the hyperplane. The role of C is to bind the sum of $\epsilon_0, \dots, \epsilon_n$ and therefore it determines the tolerance of violations that the optimizer will allow. C is a tuning parameter that is set by the user of the algorithm, whose optimal value usually is searched for experimentally. When C is small larger margins are allowed and when it is large more errors are allowed.

The support vector classifier involves inner products of the training examples (examples are vectors of features). After the optimization the support vector classifier can be represented as:

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i \langle x, x_i \rangle,$$

where $\alpha_1, ..., \alpha_n$ are the resulting optimization parameters. Since only the training examples that support the margin are needed to classify a unlabeled example, only a minor set of the examples (support vectors) are used in the classifiers. In the formula the indexes of the support vectors are denoted as S.

3.3.3 Kernels

Sometimes the classes can not be linearly separated. In these cases normally quadratic, cubic or other higher-order polynomial terms are introduced to allow for more complex functions to describe the decision boundary, but this way the feature space gets enlarged. If the data examples only can be separated by a non-linear hyperplane the examples gets mapped to a higher dimensional feature-space where they are linearly separable. If the function is very complicated the optimization problem becomes very hard to solve, since the feature space to search is much bigger. *Kernels* are an extension to Linear SVMs that allows for a implicit enlargement of the feature space.

A *kernel* is function that generalizes the inner product in a computation so that the inner product calculation instead is implicit and can save the method using the inner product some time.

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i K(x, x_i)$$

A popular example of a non-linear kernel is the gaussian *radial basis function* kernel (RBF):

$$K(x_i, x_{i'}) = exp\left(-\gamma \sum_{j=1}^{n} (x_{ij} - x_{i'j})^2\right)$$

γ is often a tuning parameter in implementations of SVMs just as C in support vector classifiers. A small gamma gives a low bias and high variance while a large gamma gives a higher bias and lower variance. To find appropriate values of γ and C a grid search is often applied, where different combinations of the parameters are tested by doing cross-validation.

3.4 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a collection of machine learning techniques that are supposed to mimic the behavior of a biological brain [23]. The basic building blocks used to model a neural network are nodes and edges, both of which are illustrated in figure 3.4. The corresponding terminology used in biology is neurons and synapses, respectively. These terms will be used interchangeably in this thesis. There are many different types of neural networks but they all have certain traits in common, namely, they all consist of a set of nodes that are interconnected via edges, which together form a network. A single node usually consists of a number of input edges and a number of output edges and it is the computational unit of a

Neural Network. In some cases a bias value $b = 1$ is added to its input. The typical computations performed by a node is to add its inputs by a weighted sum:

$$z = \sum_{j=1}^{m} w_j x_j \tag{3.4}$$

and apply a so called *activation function* (typically a sigmoid function) to the sum:

$$y = \varphi(z + b) \tag{3.5}$$

where x_j is the input for input link j, w_j is the weight for the corresponding input, $\varphi()$ is an activation function, b is the optional bias parameter and y is the output of the neuron.

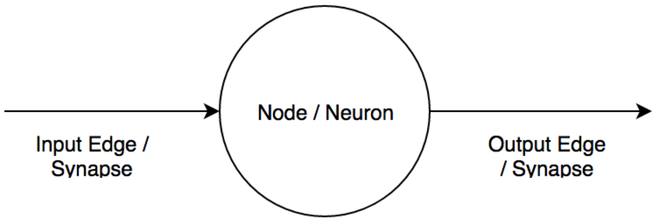


Figure 3.4: *A simple model of a single neuron. A neural network consists of many such neurons connected to each other via their input/output edges*

3.4.1 Feed Forward Neural Networks

A special class of artificial neural networks commonly used is the *Feed Forward Neural Network* or FFNN (Also called *Multilayer Perceptrons* or MLPs) which consists of multiple layers of fully connected neurons [23]. Figure 3.5 shows a FFNN with four layers.

A FFNN typically has three different types of layers, one input layer, one output layer and a number of hidden layers. The input layer does not perform any computations, it simply takes all its inputs, $x = [x_1, ..., x_n]$, and outputs them to the next layer. Each neuron in the hidden layers computes the equations 3.4 and 3.5 on the received inputs and outputs its results to the next layer. A neuron in the output layer performs the same computations as the the neurons in the hidden layer, and the result of an output neuron is then passed as the network's response, $y = h(x)$. This process is called *forward propagation.*

Determining the number of hidden layers and the number of neurons in each layer is not a trivial task and often depends on the amount of available training data as well as the properties of the training data. The number of neurons in the output layer is determined by the number of possible classes used in the specific context of the problem at hand. There is usually one neuron per class and the predicted label can be extracted from the output vector. Note that this is only true when classification is considered while regression problems are usually solved with only one output neuron.

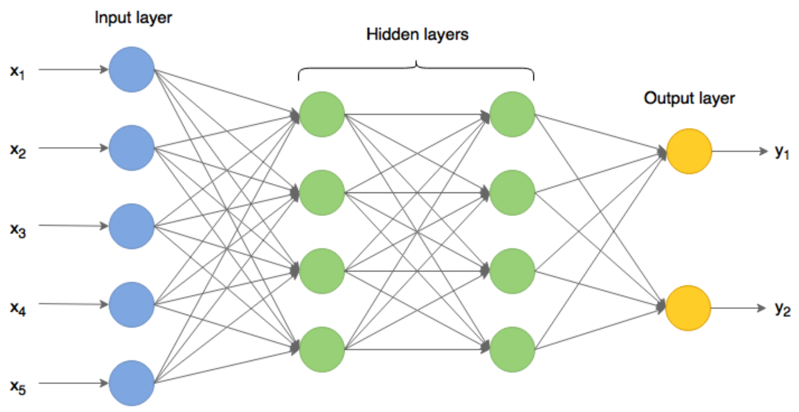


Figure 3.5: *A Feed forward Neural Network with four layers, one input layer with five neurons (blue), two hidden layers with 4 neurons each (green) and one output layer with two neurons representing two classes (yellow). The neurons between two consecutive layers in a FFNN are always fully connected.*

3.4.1.1 Training a FFNN

A FFNN is often trained using backpropagation or BackProp, which is a popular learning algorithm that trains a FFNN by adjusting the weights in each layer to give better future predictions. BackProp is usually used in conjunction with an optimization algorithm such as *gradient descent.* In contrast to the feedforward step, which propagates a training example from the "back" to the "front" (From left to right in figure 3.5), the BackProp algorithm propagates its values in the opposite direction, hence its name. The following steps give a high level overview of how the algorithm works:

1. Each training example x^i goes through a forward propagation step where a network response is given, as explained earlier.

2. The error of the output layer is computed by comparing the networks response with the actual class label of the training example, i.e. a cost function

$$J = \frac{1}{2}\sum_{i=1}^{n} e_i^2$$

$$e_i = y_i - y_{pi}$$

   is computed, where n is the number of neurons in the output layer, y_i is the actual label of the training example for output neuron i and y_{pi} is the response of neuron i.

3. Gradient descent is then used to minimize this cost function by computing the gradients $\frac{\partial J}{\partial w_{ij}^l}$ for each synapse (link) going from neuron i to neuron j in each

layer l. For each weight, the gradients are calculated by using the chain rule as follows:

$$\frac{\partial J}{\partial w} = -e\frac{\partial y_p}{\partial \varphi}\frac{\partial \varphi}{\partial z}\frac{\partial z}{\partial w}$$

4. Moreover, these gradients are then used to adjust the weights in the network with the following computation:

$$\Delta w_{ij}^l = -\alpha\frac{\partial J}{\partial w_{ij}^l}$$

$$w_{ij,new}^l = w_{ij,old}^l + \Delta w_{ij}^l$$

where α is a constant called the *learning rate*, $w_{ij,old}^l$ is the old value of the weight w_{ij^l} and $w_{ij,new}^l$ is the new, updated weight value.

3.4.1.2 Learning rate

Gradient descent algorithms use a constant α which is called the *learning rate* to determine how fast it steps. The choice of learning rates is often determined experimentally by trying different values to see which perform best. The authors of [24] have proposed an approach they call Rprop where the learning rate is calculated locally for each weight.

3.5 K-Nearest Neighbors

K-nearest neighbors or KNN is a relatively simple algorithm that doesn't require any training. The algorithm takes a new example, finds its K nearest neighbors, and applies a voting strategy where the predicted label of the example is set to be the class of the majority of its neighbors. Although there are other, a very common strategy used to determine which neighbors are nearest is the euclidian distance

$$d = \sqrt{\sum_{i=1}^{N}(x_p - x_i)^2}$$

where N is the dimensionality of the feature space, x_p is the test point and x_i is a neighbor of x_p. Figure 3.6 shows a simple example of a classificaiton problem having two classes, circles and triangles. The new test point is being classified as belonging to one of these classes depending on the majority vote of its neighbors.

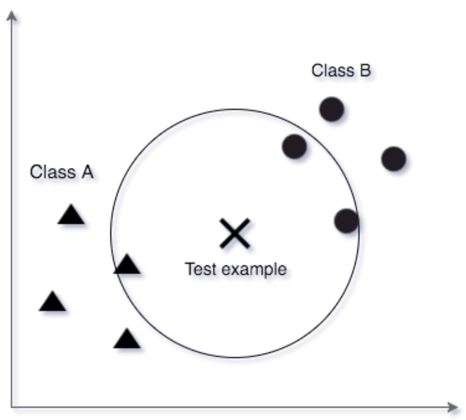
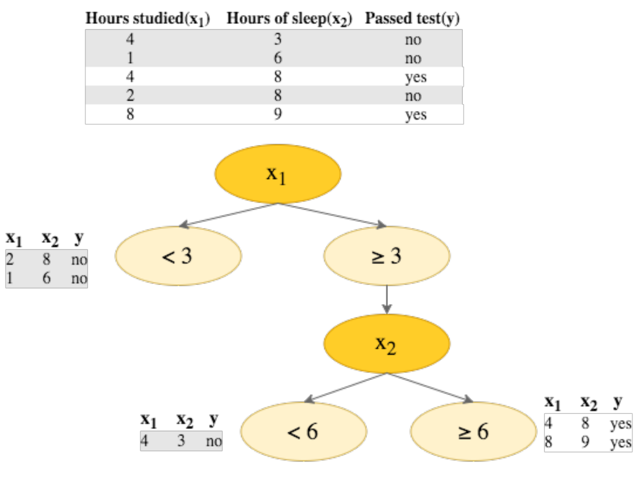


Figure 3.6: *The new test example is classified by finding its three ($K = 3$) nearest neighbors by calculating the euclidian distance between the new test example and each of its three neighbors. A majority vote then determines that the new example belongs to the circles class (2 votes against 1).*

3.6 Random Forests

The Random Forests algorithm, or RF, belongs to a family of algorithms that are based on building decision trees to classify objects based on features. Figure 3.7 presents a simple example of a decision tree that tries to predict whether a student will pass his/her exam depending on two features, namely, how many hours the student has studied for the test and how many hours of sleep the student had the night before the test. Each leaf (The end node of any branch) shows a subset of the training examples with the same type of output, passing the test or not and these types of sets are called *pure*.



Hours studied(x_1)	Hours of sleep(x_2)	Passed test(y)
4	3	no
1	6	no
4	8	yes
2	8	no
8	9	yes

x_1	x_2	y
2	8	no
1	6	no

x_1	x_2	y
4	3	no

x_1	x_2	y
4	8	yes
8	9	yes

Figure 3.7: *An example of a simple decision tree trying to predict whether a student will pass his/her next exam depending on how many study hours and how many hours of sleep the student had the day before the exam. Each leaf node shows a pure set of examples.*

A nice property of most tree based algorithms is that it is fairly intuitive to understand how they work compared to many other machine learning algorithms. A drawback however, is that the simplest decision tree algorithms usually perform worse in comparison to the more advanced algorithms [22]. However, the RF algorithm manages to mitigate the negative aspects of simple decision trees by sacrificing some of these simple properties to increase its performance significantly.

The general principle of RFs are to create multiple trees each with a random subset of the training examples (This technique is referred to as bagging) for each tree. Moreover, for each node in the tree, the algorithm picks a random subset of the features, and chooses the best one. Choosing random subsets of features helps to make the trees less correlated by giving more importance to less significant features. For example, if a training set has a feature that has a lot more importance in predicting a certain class then a random subset of features in one of the trees might not include this feature, giving the less important features a chance of giving "their" view of the prediction. Typically, the number of random features chosen for a specific tree is given by

$$r \approx \sqrt{n}$$

where n is the total amount of features in the training set. Finally, to classify a new test example, a majority vote for all trees is performed.

4 Implementation

This chapter describes third-part dependencies used to build the pipeline, and provides details of the evaluation setup for the segmentation and classification. Furthermore, an overview of the pipeline itself is provided. Four different classifiers, and three segmentation algorithms where implemented for this thesis.

4.1 Third party dependencies

This section provides an overview of the external resources and libraries used in this thesis.

4.1.1 PCL

The Point Cloud Library (PCL) is a large scale open project for 2D/3D image and point cloud processing[3]. The PCL framework contains numerous state-of-the art algorithms including filtering and segmentation algorithms, it also contains various methods for visualization and processing of point clouds. The library was used for easier handling and manipulation of point cloud data during the implementation, mainly for filtering (see 4.3.1), ground extraction (see 4.3.2), visualization and for segmentation (see 4.3).

4.1.2 KITTI

The KITTI vision benchmark suite is a cooperative project between Karlsruhe Institute of Technology and Toyota Technological Institute in Chicago. The goal of the project is to provide benchmarks with novel difficulties to the computer vision community[25]. The data benchmarks were collected by driving a vehicle in different urban settings recording the surroundings of the vehicle with a Velodyne HDL-64E unit. The benchmark data consist of raw data point clouds collected from city-, residential-, road-, and campus settings. This data is available online (The KITTI Vision Benchmark Suite) and was used during the implementation.

4.1.3 Shark

SHARK is a C++ library for machine learning algorithms. It comprises methods for single- and multi-objective optimization (e.g., evolutionary and gradient-based algorithms) as well as kernel-based methods, neural networks, and other machine learning techniques[26]. This library was used for the implementation of the classification module (see 4.4).

4.2 Overview of the pipeline

The pipeline is comprised of two main processes, segmentation and classification. The input to the pipeline consist of a point cloud object, which essentially is a wrapper to a vector of points with x,y,z-coordinates and intensity. The point cloud is pre-processed by a filtering and ground removal module before the segmentation. From the resulting point cloud clusters features are extracted, which are then used by a pre-trained classifier, which returns the labels of these clusters. Figure 4.1 depicts the whole process.

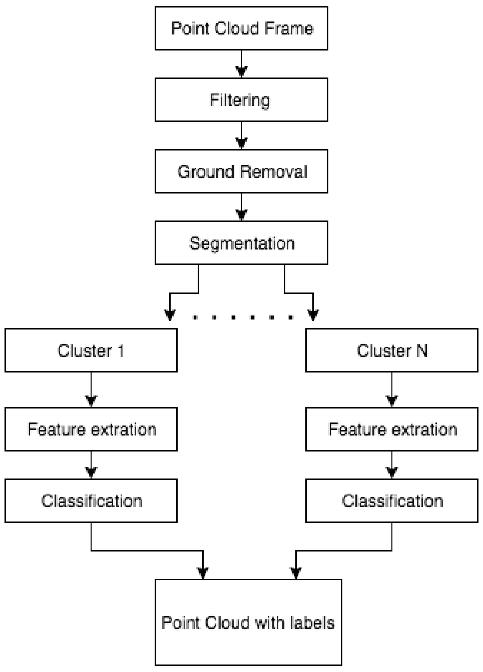


Figure 4.1: *Overview of the whole segmentation-classification pipeline.*

4.3 Segmentation

In the segmentation module, *Region growing* and *Euclidean clustering* was used, both of which are available in PCL. To find the closest neighbors the Euclidean clustering implementation in PCL uses a *KD-tree* representation of the points, which subdivides the 3D-space into a treelike structure for faster search for nearest neighbors. Euclidean clustering was used with a 45cm distance threshold which was chosen by experimenting and visualizing the results repeatedly. The implementation of Regional growing in PCL uses *smoothness* as the *similarity property.* To use Region growing estimating normals to all the points in the point cloud were necessary. A method for this in PCL were used. The smoothness property compares the angle between the estimated normals of the points to some threshold, in this case it was set to 32°, which also was chosen by experimenting and visualizing the results.

Before the segmentation the point cloud is pre-processed by a filtering and ground removal module. It has been empirically proven that ground removal improves the segmentation[11], and that filtering allows for more accurate segmentation and is preferably done before the ground removal[12].

4.3.1 Filtering

Noise in point clouds occur because registered points often will be unevenly dense after a scan due to varying distances, other reasons might include wrongly registered points due to weather conditions such as haze, or due to system noise To remove noise in the point cloud statistical outlier removal is applied which removes points that lie outside a given threshold from the distance mean of neighbouring points. PCL includes functionality for this method which was used in the implementation, and was done prior to the ground segmentation.

4.3.2 Ground removal

To apply ground removal the model fitting method RANSAC was used in the implementation. To compute whether a point is an inlier or outlier to the ground a threshold is set, which specifies how far away the point can be to be considered an inlier. In our instance we use a threshold of 30 cm for the most efficient performance. This led to to some parts of the clusters to be cut of such as the feet of the pedestrians.

4.4 Classification

This section will give a description of how the classification details were implemented. The shark library was used to implement *KNN*, *FFNN*, *SVM* and *RF*. SVM used a One-Vs-All classifier to handle multiple classes, with a radial basis function as kernel. The following subsection will present the two feature sets (A and B) that will be used to compare the classification algorithms.

4.4.1 Feature set A

The feature set presented in table 4.1 is the original set used at Volvo Cars by the authors of [18]. The set contains ten features describing global attributes of the objects. Since the features have different units and thereby different ranges they are scaled and normalized to lie in the range $-\frac{1}{2} < f < \frac{1}{2}$. The normalization was done by subtracting each feature by its mean and the scaling was done by dividing by the range of each feature. The process is summarised by the following equation:

$$\widehat{f_i} = \frac{f_i - \mu_i}{max_i - min_i}$$

, where f_i is feature i, μ_i is the mean value of feature i and max_i and min_i are the max and min values of feature i, respectively.

Features	
Feature	Description
f_1	The height of the object
f_2	The width of the object
f_3	The depth of the object
f_4	The box-volume of the object
f_5	The length of the hypotenuse of the width and the depth of the object
f_6	The standard deviation of the distance from each point to the center of gravity of the object
f_7	The distance to the object from the vehicle multiplied with the number of points of the object
f_8	A measure of how scattered the points are in the object
f_9	A measure of the "linear-ness" of the object
f_{10}	A measure of how "surface-like" the object is

Table 4.1: *Feature set A*

4.4.2 Feature set B

Feature set B is based on [19] with some slight modifications to suit the classes used in this project. The basic idea is to project the object onto a 2D grid of fixed size (16×16) where each cell has the dimensions $0.5m \times 0.5m$. Moreover, the number of points in each cell is counted and a vector of features is given by concatenating each row in the grid to a one-dimensional vector of size 256. The resulting feature vector is given by adding the height H of the object as well as the largest value of the width W and the depth D to the vector of 256 grid values, forming a feature vector of size 258 in total.

To project the object onto the 2D grid, the vertical axis of the grid is given by the z-axis of the point cloud. To determine which of the two x-axis or y-axis should be used for the projection, the width W and the depth D of the object is calculated and compared. The horizontal axis of projection is chosen to be the bigger of the two values, as can be seen in figures 4.2 and 4.3, where the width (x-axis) is used as the horizontal projection axis.

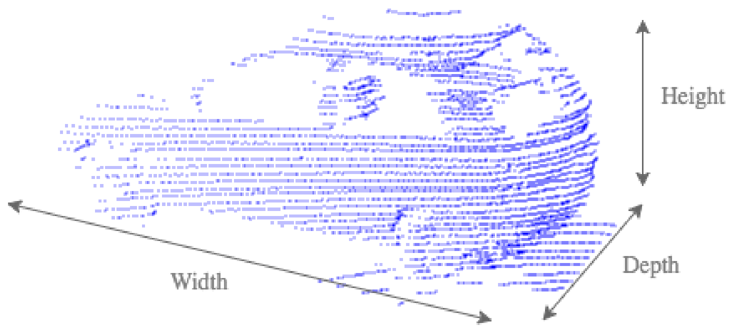


Figure 4.2: *A training example representing a car and the three dimensions (height, width and depth) are shown The height corresponds to the z axis in the point cloud, the width and the depth correspond to the x and y axes respectively. The projection surface is the height combined with the largest of the width and the depth axis.*

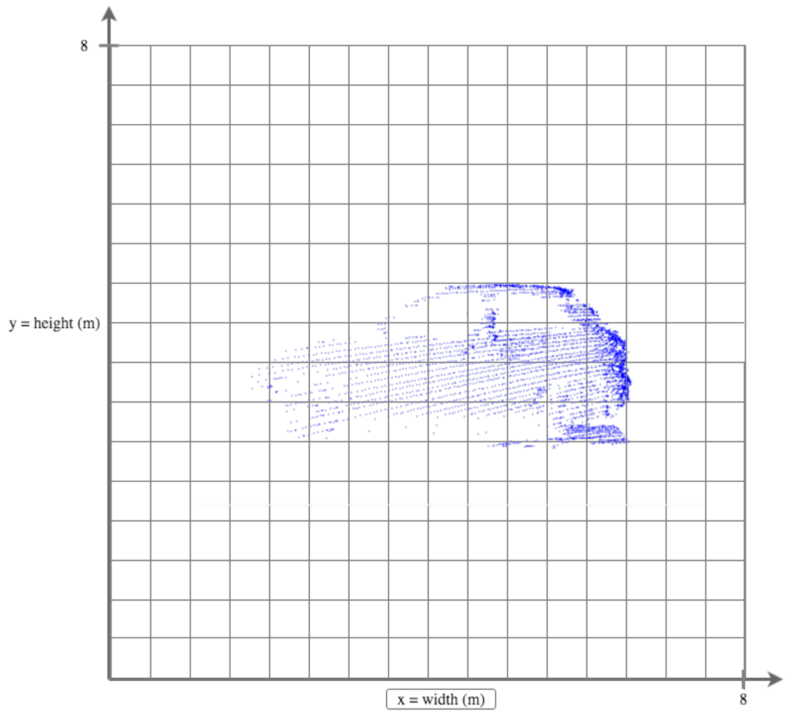


Figure 4.3: *The training example from figure 4.2 projected onto the surface of its height and width axes. The object is centered in the grid based on its "center of mass" and the surface grid is divided into 256 cells. The dimensions of each cell is 0.5m x 0.5m. The number of points of each cell is counted and a feature vector is extracted by appending each horizontal row into a vector.*

Choosing the grid size to be 16 × 16 instead of 8 × 8 as done by [19] is because of the fact that this project considers different classes. The classification of this project need to be able to differentiate between Vans and Trucks, both of which

are expected to have a horizontal size ($max(W, D)$) bigger than $8 * 0.5 = 4$ meters. $16 * 0.5 = 8$ meters was chosen by assuming that no Vans can be longer than 8 meters, thus giving the algorithms a better chance of distinguishing between Vans and Trucks.
Lastly, to normalize the numbers of each cell, the calculated count in each cell is divided by the maximum count number.

4.4.3 Data sets

The data for the training and validation for the classifiers was taken from the KITTI data set. KITTI provides *tracklets*, which basically are files of sequences of the same clusters from different time steps in a recording. Obtaining training data manually takes a long time and instead, tracklets from the data set were used to create the training and validation data. This meant that the same objects could occur multiple times in the data but from different view points. Therefore this object could be considered as different examples. But if the recording vehicle at any point stood still during the recording, for instance by some traffic light, this could have led to the tracked objects to appear from the same viewpoint multiple times.

Since the tracklets did not contain any poles these where extracted manually from some selected point clouds. After the data was obtained features for all the clusters where extracted to a file. This was done for both the training data and the validation data. The table 4.2 accounts for the number of samples used in each data set.

Class	Training	Validation
Pedestrian	1307	312
Car	8181	2144
Van	1380	138
Bicycle	658	207
Truck	212	47
Pole	39	10
Unknown	219	21
Total	11996	2879

Table 4.2: *Number of examples for each class and for each data set.*

4.4.3.1 Avoiding validation biases

In order to avoid the bias of an object reoccurring in both the training and validation sets, tracklets from completely different data sets were used for the two sets. The sets total size are approximately divided in to the proportion 70/30 (9117/2879). The offset to this proportion between the classes in the two sets is related to the fact that most of the examples originate from the tracklets files in KITTI; a clear division between 70/30 could therefore not easily be made and moving data between the sets would lead to a bias.

4.4.4 Training and Validation

In practical machine learning applications the evaluation of parameters is necessary in order to optimize the performance of the application. The performance on the training set is not a good indicator on the performance on unseen data due to the problem of over-fitting, therefore the trained model is usually evaluated with some unseen data[22].

One common approach is *k-Fold Cross-validation*, which randomly divides the data into k equally sized sets of which one becomes a validation set and the rest comprises the training set, then the error rate is computed, this is repeated until all divisions have acted as a validation set. The mean error rate of all iterations is returned finally. Another approach is *the validation set approach*, which simply separates the data in a training set and a validation set, where the training set is used to learn the parameters and the validation set to evaluate the predictive performance of the parameters[22]. In this thesis *the validation set approach* was used.

4.4.4.1 Searching optimal parameters

To find the optimal parameters a grid search was performed performed for the methods with more than one parameter. In a grid search various combination of parameter values are tried by doing a validation and the parameters with best performance are picked. For each method two searches were made, one for feature set A and one for feature set B.

For SVM a grid search for γ and C was made with $\gamma = 2^{-15}, 2^{-13}, \ldots, 2^{3}$ and $C = 2^{-5}, 2^{-3}, \ldots, 2^{7}$. For RF a grid search for optimal *nodesize* and *trees* was made with $nodesize = 1, 5, 10, 20, 30, 40, 50$ and $trees = 50, 100, 128, 256$. For KNN the optimal tree size was searched for by simply testing $K = 1, 3, \ldots, 99$.

In the Shark library the RF algorithm takes nodesize as a parameter. The nodesize is a upper bound on the node before it is classified as a leaf. Lowering this value makes the trees in the ensemble larger and increasing this value makes the trees smaller.

To determine the number of hidden layers, the number of neurons in each hidden layer and the type of activation function in the hidden and output layers for the FFNN implementation, a grid search was performed for each combination of activation functions (sigmoid, hyperbolic tangent and linear). The search was limited to at most 2 hidden layers. The network structure determined by the grid search turned out to give very unstable accuracy results after running the validation several times with the best parameters. This lead to a new approach, where the network parameters where manually selected by finding a relatively stable parameter-setup, where each consecutive run produced similar accuracies. The most stable activation function combination was to use a sigmoid function in the hidden layers and a linear function in the output layer for feature set A. For feature set B, a hyperbolic tangent function for both the hidden layer and the output layer produced the most stable combination.

Parameter search on feature set A

Method	Parameters	Values
SVM	γ, C	0.125, 128
RF	*trees*, *nodesize*	128, 30
FFNN	*hiddennodes*	5
KNN	K	9

Table 4.3: *Optimal parameters for methods searched over feature set A.*

Parameter search on feature set B

Method	Parameters	Values
SVM	γ, C	0.125, 2
RF	*trees*, *nodesize*	128, 5
FFNN	*hiddennodes*	128
KNN	K	5

Table 4.4: *Optimal parameters for methods searched over feature set B.*

5

Results

The following section will present the results obtained during the segmentation and classification steps. First, the segmentation results will be presented. Thereafter, the performance of the classification algorithms will be presented.

5.1 Segmentation

The segmentation results will be presented visually and compared using images for the algorithms Euclidean Clustering, Region Growing.

5.1.1 Under-segmentation

To illustrate the under-segmentation problem a point cloud with people close to each other is segmented by Euclidean clustering and Region growing. The following pictures depicts the results.

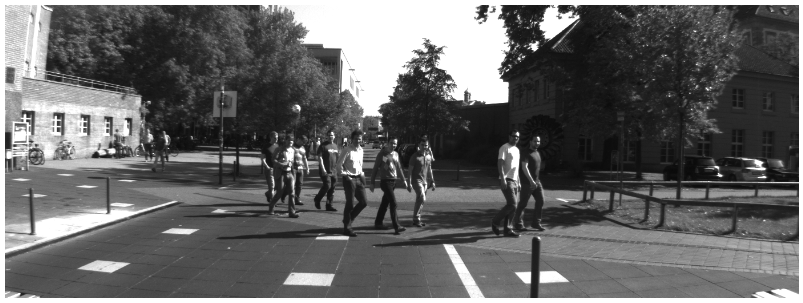


Figure 5.1: *People crossing street.*

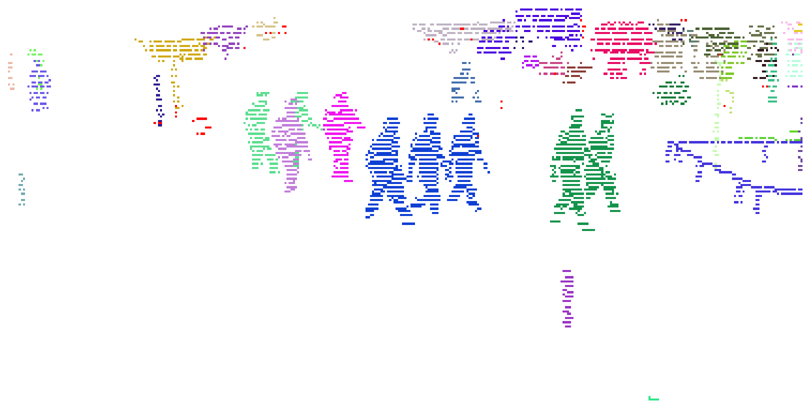


Figure 5.2: *People crossing street segmented by Euclidean clustering.*

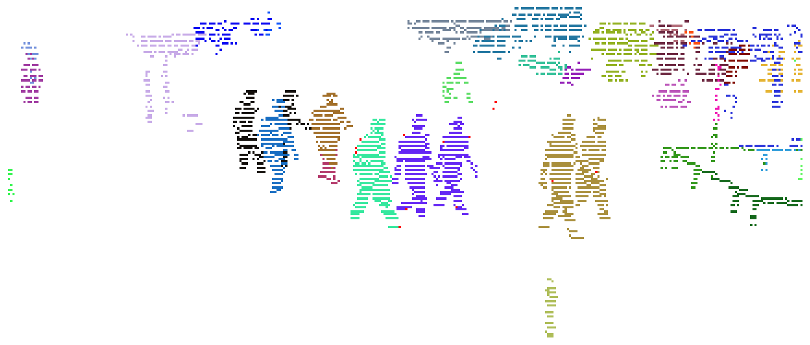


Figure 5.3: *People crossing street segmented by Region growing.*

5.1.2 Over-segmentation

To illustrate the over-segmentation problem a point cloud taken from traffic congestion with a large truck is segmented by Euclidean clustering and Region growing. The following pictures depicts the results.

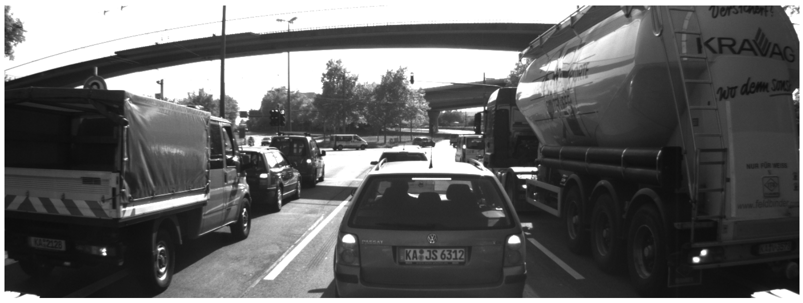


Figure 5.4: *Traffic congestion.*

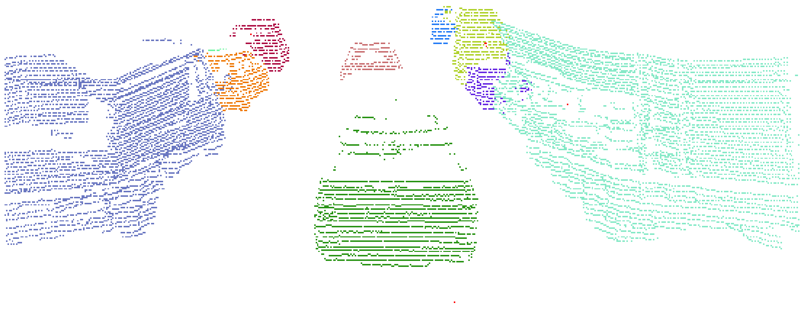


Figure 5.5: *Traffic congestion segmented by Euclidean clustering.*

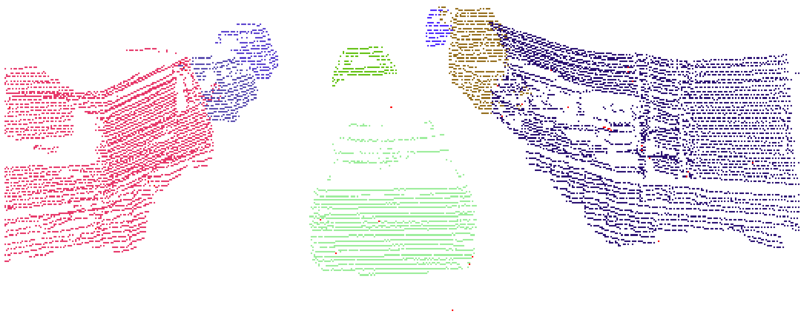


Figure 5.6: *Traffic congestion segmented by Region growing.*

5.2 Classification result

In order to evaluate the classification algorithms a comparison of KNN, FFNN, RF and SVM was made were the results are presented in a confusion matrix for the classes. Two different feature sets where tested with different properties.

5.2.1 SVM performance on Feature set A

The following section shows the accuracy results for our SVM classifier for feature set A. Table 5.1 shows the confusion matrix for each class.

	Pedestrian	Car	Van	Bicycle	Truck	Pole	Unknown
Pedestrian	298	9	0	5	0	0	0
Car	4	2138	1	1	0	0	0
Van	0	48	89	1	0	0	0
Bicycle	5	2	9	191	0	0	0
Truck	0	1	3	0	43	0	0
Pole	3	0	0	0	0	7	0
Unknown	0	21	0	0	0	0	0

Table 5.1: *Confusion matrix for SVM on feature set A. The rows are the actual class label while the columns show the predicted class label.*

5.2.2 KNN performance on Feature set A

The following section shows the accuracy results for our KNN classifier for feature set A. Table 5.2 shows the confusion matrix for each class.

	Pedestrian	Car	Van	Bicycle	Truck	Pole	Unknown
Pedestrian	278	30	0	4	0	0	0
Car	5	2123	11	3	0	0	2
Van	0	45	88	1	0	0	4
Bicycle	13	0	5	183	0	0	6
Truck	0	1	10	0	27	0	9
Pole	3	0	0	0	0	7	0
Unknown	0	21	0	0	0	0	0

Table 5.2: *Confusion matrix of KNN on feature set A. The rows are the actual class label while the columns show the predicted class label.*

5.2.3 FFNN performance on Feature set A

The following section shows the accuracy results for our FFNN classifier for feature set A. Table 5.3 shows the confusion matrix for each class.

	Pedestrian	Car	Van	Bicycle	Truck	Pole	Unknown
Pedestrian	299	1	0	11	0	0	1
Car	4	2136	2	2	0	0	0
Van	0	48	89	1	0	0	0
Bicycle	1	24	0	180	0	0	2
Truck	0	0	1	0	46	0	0
Pole	0	0	0	0	0	10	0
Unknown	0	21	0	0	0	0	0

Table 5.3: *Performance matrix of FFNN on feature set A. The rows are the actual class label while the columns show the predicted class label.*

5.2.4 RF performance on Feature set A

The following section shows the accuracy results for our RF classifier for feature set A. Table 5.4 shows the confusion matrix for each class.

	Pedestrian	Car	Van	Bicycle	Truck	Pole	Unknown
Pedestrian	307	2	0	3	0	0	0
Car	3	2106	33	1	0	0	1
Van	0	35	102	0	0	0	1
Bicycle	2	27	1	174	0	0	3
Truck	0	0	2	0	42	0	3
Pole	2	0	0	0	0	8	0
Unknown	0	21	0	0	0	0	0

Table 5.4: *Performance matrix of RF on feature set A. The rows are the actual class label while the columns show the predicted class label.*

5.2.5 Overall results on Feature set A

Table 5.5 shows the total accuracy performance for all algorithms for feature set A. The accuracies are calculated by dividing the total number of correctly classified validation exampled by the total amount of validation examples, for each classifier. The mean accuracy is calculated by taking the mean of the true positives (the diagonal from left to right, bottom down) from the confusion matrices in the previous subsections.

Classifier	Accuracy	Mean Accuracy
SVM	96.3%	73.4%
FFNN	95.9%	77.8%
RF	95.1%	74.9%
KNN	94.0%	66.8%

Table 5.5: *Total and mean accuracy for each classifier on feature set A.*

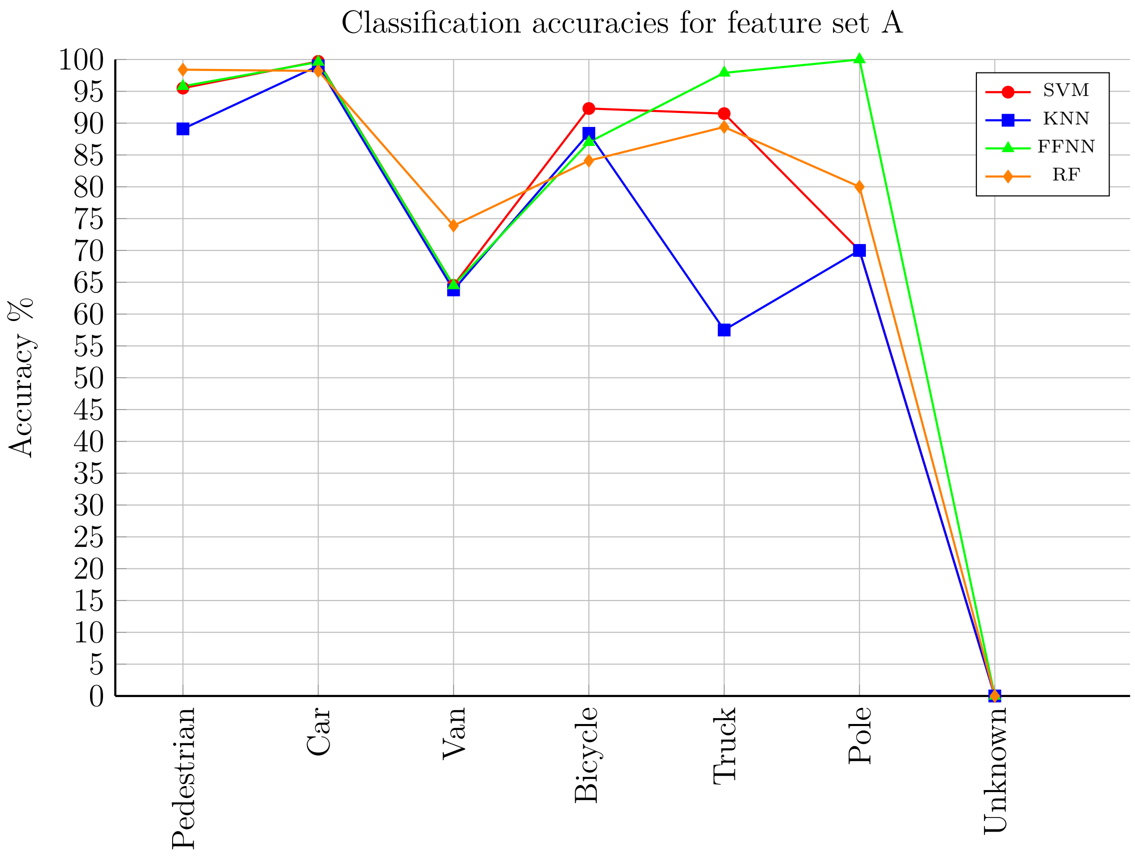


Figure 5.7: *A visual representation of the accuracies for different classes for feature set A. The horizontal axis represents the classes while the vertical axis shows the accuracies in percentages.*

5.2.6 SVM performance on Feature set B

The following section shows the accuracy results for our SVM classifier for feature set B. Table 5.6 shows the confusion matrix for each class.

	Pedestrian	Car	Van	Bicycle	Truck	Pole	Unknown
Pedestrian	311	0	0	1	0	0	0
Car	6	2122	12	4	0	0	0
Van	0	34	100	4	0	0	0
Bicycle	6	21	0	180	0	0	0
Truck	0	0	11	0	34	0	2
Pole	4	0	0	0	0	6	0
Unknown	0	21	0	0	0	0	0

Table 5.6: *Performance matrix of SVM on feature set B. The rows are the actual class label while the columns show the predicted class label.*

5.2.7 KNN performance on Feature set B

The following section shows the accuracy results for our KNN classifier for feature set B. Table 5.7 shows the confusion matrix for each class.

	Pedestrian	Car	Van	Bicycle	Truck	Pole	Unknown
Pedestrian	307	4	1	0	0	0	0
Car	8	2101	20	6	0	0	9
Van	0	40	88	9	0	0	1
Bicycle	14	4	1	184	0	0	4
Truck	0	0	26	0	19	0	2
Pole	3	0	0	0	0	7	0
Unknown	0	20	1	0	0	0	0

Table 5.7: *Performance matrix of KNN on feature set B. The rows are the actual class label while the columns show the predicted class label.*

5.2.8 FFNN performance on Feature set B

The following section shows the accuracy results for our FFNN classifier for feature set B. Table 5.8 shows the confusion matrix for each class.

	Pedestrian	Car	Van	Bicycle	Truck	Pole	Unknown
Pedestrian	297	7	2	5	0	1	0
Car	7	2054	60	3	7	13	0
Van	0	27	108	2	0	0	1
Bicycle	4	33	0	170	0	0	0
Truck	2	4	17	0	16	0	8
Pole	5	0	1	0	0	4	0
Unknown	0	21	0	0	0	0	0

Table 5.8: *Performance matrix of FFNN on feature set B. The rows are the actual class label while the columns show the predicted class label.*

5.2.9 RF performance on Feature set B

The following section shows the accuracy results for our RF classifier for feature set B. Table 5.9 shows the confusion matrix for each class.

	Pedestrian	Car	Van	Bicycle	Truck	Pole	Unknown
Pedestrian	309	2	0	1	0	0	0
Car	7	2118	19	0	0	0	0
Van	0	29	109	0	0	0	0
Bicycle	6	36	0	165	0	0	0
Truck	0	1	20	0	23	0	3
Pole	4	0	0	0	0	6	0
Unknown	0	21		0	0	0	0

Table 5.9: *Performance matrix of RF on feature set B. The rows are the actual class label while the columns show the predicted class label.*

5.2.10 Overall results on Feature set B

Table 5.10 shows the total accuracy performance for all algorithms for feature set B. The accuracies are calculated by dividing the total number of correctly classified validation exampled by the total amount of validation examples, for each classifier. The mean accuracy is the mean accuracy over the classes.

Method	Accuracy	Mean Accuracy
SVM	95.6%	70.1%
FFNN	92.0%	60.8%
RF	94.8%	66.5%
KNN	94.0%	65.6%

Table 5.10: *Total and mean accuracy for each classifier on feature set B.*

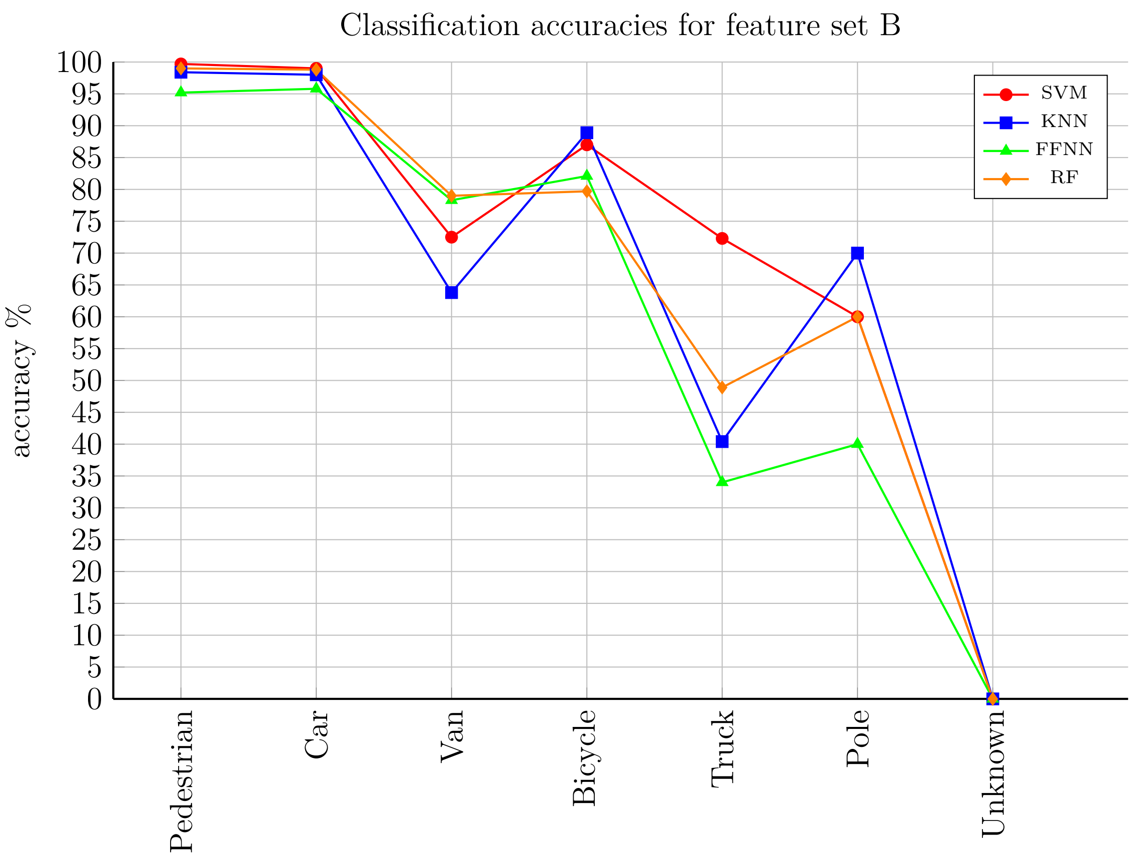


Figure 5.8: *A visual representation of the accuracies for different classes for feature set B. The horizontal axis represents the classes while the vertical axis shows the accuracies in percentages.*

6
Discussion

A comparison of two segmentation methods and four different classification methods is presented. Two feature sets (A and B) are also compared for classification results where feature set A contains 10 global features and feature set B contains 256 local features and 2 global features. Out of the four classifiers SVM has the highest overall accuracy using Feature set A. This chapter provides some thoughts on the results and gives suggestions for future work.

6.1 Segmentation

Although Region growing is generally considered to be more unstable than Euclidean clustering Region growing outperforms the Euclidean clustering approach under the *over-segmentation* problem as illustrated by the the segmentation of a truck in figure 5.6. The truck and the trailer were segmented as two objects by Region growing while Euclidean clustering segmented also the truck into two objects (figure 5.5). The reason for Euclidean clustering's over-segmenting behaviour is that the 45cm distance threshold, that was set, was too small in this case. But when experimentally searching for a threshold with reasonably good performance greater thresholds were found to under-segment too many objects so this was not changed.

Region growing also outperforms Euclidean clustering under the *under-segmentation* problem as illustrated by the segmentation of pedestrians in a group in figure 5.3. Both algorithms segment poorly and this also is due to wrong parameters in this case. A much finer grained segmentation could have been done with smaller thresholds set for both methods but this would have led to poor results in the general case. Although the two methods both segment the three pedestrians in the center wrongly Region growing is able to distinguish one from the others, while Euclidean clustering segments them all into one segment (figure 5.2).

6.2 Classification

The classification results that were obtained can be divided into three categories, represented by the following sub sections. The first sub section covers the results that are considered to be very promising and that can be compared to results obtained by other authors in the literature. The next sub section discusses results that are not considered as good, where the classifiers are having difficulties distinguishing between several classes. The third sub section discusses mistakes that could have been avoided during the implementation.

6.2.1 The good

All of the classification methods have fairly good accuracies for pedestrians and cars on both feature sets. All algorithms except KNN have an accuracy over 95% on feature set A. These results are expected since there is a relatively large amount of training data for these classes, 1307 and 8181 examples respectively. On feature set A, SVM has a total accuracy of 96.3%, FFNN 95.9% and RF 95.1% (figure 5.5).

On feature set B, all of the algorithms, including KNN, has an accuracy performance over 95% for pedestrians and cars. An interesting observation is that all methods except FFNN have an accuracy over 98% for these classes, since the paper that feature set B originated from used FFNN to classify. This means that better performance probably could have been achieved if other methods than FFNN had been chosen in that case. On feature set B SVM had an overall accuracy of 95.6%, RF 94.8%, and KNN 93.9% (figure 5.10).

Almost all of these accuracy results for pedestrians and cars are comparable to the results obtained in the literature. Moreover, avoiding biases in the validation data was considered important, as mentioned in section 4.4.3.1. Taking this into account, our classification results for pedestrians and cars can be considered as quite good results.

6.2.2 The bad

There were some confusion between Pedestrians-Bicyclists, Bicyclists-Cars, Poles-Pedestrians, Vans-Cars, and Vans-Trucks. This was expected since these classes are similar and easy to confuse. Since there was a comparatively small set of examples in the validation data for Poles and Trucks (10, 47 examples respectively) it is more difficult to draw any conclusions about the performance of the classifiers between these classes, although the largest variations between the classifiers performance can be observed in these classes (figure 5.7, 5.8). One possible reason for these results is that there are too few training and validation examples to train a classifier to discriminate between the confused classes, especially since many of them have similar geometrical properties. An obvious solution to this problem is to add more training and validation examples. Another possible reason is that neither of the two feature sets that are used are able to discriminate between these classes satisfactory enough. A possible solution to this problem could be to find additional features that are able to differentiate between these classes.

FFNN had best mean accuracy on feature set A but it was also pretty unstable and the accuracy varied considerably between runs. The weights are randomly initialized during each training-validation run of FFNN. Since this is the only varying factor between runs, it is very likely the reason for the unstable results. If this factor could be minimized, FFNN would be a more promising candidate to use in future work.

6.2.3 The ugly

Figures 5.7 and 5.8 show that the accuracy of the unknown class is 0%, consistently for all classifiers on both feature sets. One reason for the poor result is because unknown has been trained with only unknown data examples that represent any other possible class, from small stones to large buildings. These training examples, though containing 219 examples, did only consist of 7 objects but taken from multiple frames. Therefore none of the machine learning methods were able to learn any parameters for the class unknown. A better approach that was not taken would be to compare the probabilities of each predicted class with a threshold value. If the probability is smaller than the threshold, the object would be classified as an unknown object.
Another more obvious reason for the poor performance and also an explanation for why all Unknown validation examples were classified as Cars is that the 21 validation examples in the validation set actually consisted of only one object taken from multiple frames. When visualizing this object it seemed to look like a large box or container, which meant that its geometrical properties were similar to a car which might have led the classifiers confusing it with a car.

6.3 Future work

During the thesis we have identified some areas that authors of future work could consider when evaluating different algorithms.

One of the most interesting aspects is to focus on finding new features that are better at distinguishing between objects with similar properties such as cars, vans and trucks or poles and pedestrians. Current research presents a large number of different features that looks promising. One set of features that could be considered are for example different variations of intensity features (mean intensity, std.dev of the intensity etc.). Another example is to replace the box volume with a more detailed calculation of the volume of an object. For example, a pedestrian holding out his hands horizontally will have a significantly increased box volume. Such variations in real life data may affect the variations in features like the box volume but also in features like the width and depth among others. Instead, an alternative to the box volume would be to explore other ways of finding more exact volumes of irregular objects in point clouds.

It would also be interesting to see how a segmentation-classification pipeline could be implemented by using model-fitting approaches. In this work segmentation and classification has been considered as two separate steps but by using a model-fitting approach these steps could be done in one. An example of such an algorithm to use is the Iterative Closest Point (ICP) which tries to transform and rotate an object to fit a target (class). This approach was used in [21] and gave an accuracy on par with high performing feature based classifiers. A significant point to notice is that by using this approach far less data would be needed since no training would be necessary.

An interesting approach would be to investigate if it's possible to combine all or some of the classification methods to improve the results. For example to make predictions based on majority votes of the classifiers. If the majority of the algorithms predicted a certain class with a confidence above some threshold then that would be the predicted class label. Alternatively, one could look at the confidence of each algorithms prediction and pick the one having the highest value. The benefits of these approaches is that the inherent differences between the classifiers could be exploited and combined to potentially produce better results.

7 Conclusion

Region growing, although not considered stable is generally more performant than Euclidean clustering and a strong contender to use for segmentation in the context of urban environments. There is an obvious trade-off between under-segmentation and over-segmentation. Depending on the classes for classification a good threshold should be experimentally be searched for the best performance.

The winner when it comes to total accuracy for both feature sets is SVM. This seems to go hand in hand with the literature, where most papers that we have looked at are using SVMs for classification in the context of urban point cloud classification. Although, one can argue that both FFNN and RF are good contenders to SVM for feature set A, and RF is a good contender for SVM for feature set B, based on the fact that the accuracies are relatively close.

When considering the accuracies for different classes there seems to be no clear winner since different algorithms perform better for a specific class than another algorithm. For example, RF seems to be better than the other algorithms at predicting pedestrians and vans for feature set A, but worse at predicting bicycles, for the same feature set. A conclusion that can be made is that the choice of algorithm is not as important as the choice of features since all algorithms perform better for feature set A than feature set B.

Bibliography

[1] V. Yushkov, B. Yushkov, and A. Burgonutdinov, "Active safety vehicles and reducing road accidents (in German)," *Proceedings of Moscow State University of Civil Engineering/Vestnik MGSU*, no. 10, 2014.

[2] "Annual global road crash statistics." `http://asirt.org/initiatives/informing-road-users/road-safety-facts/road-crash-statistics`. Accessed: 2016-03-30.

[3] R. B. Rusu and S. Cousins, "3D is here: Point cloud library (PCL)," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1–4, IEEE, 2011.

[4] A. Jordan, "On discriminative vs. generative classifiers: A comparison of Logistic regression and Naive Bayes," *Advances in neural information processing systems*, vol. 14, p. 841, 2002.

[5] E. H. Lim, *3D Urban Modelling*. PhD thesis, Monash University, 2004.

[6] T. Rabbani, F. Van Den Heuvel, and G. Vosselmann, "Segmentation of point clouds using smoothness constraint," *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 36, no. 5, pp. 248–253, 2006.

[7] F. Moosmann, O. Pink, and C. Stiller, "Segmentation of 3D LIDAR data in non-flat urban environments using a local convexity criterion," in *Intelligent Vehicles Symposium, 2009 IEEE*, pp. 215–220, IEEE, 2009.

[8] S. Pu, M. Rutzinger, G. Vosselman, and S. O. Elberink, "Recognizing basic structures from mobile laser scanning data for road inventory studies," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 66, no. 6, pp. 28–39, 2011.

[9] R. B. Rusu, "Semantic 3D object maps for everyday manipulation in human living environments," *KI-Künstliche Intelligenz*, vol. 24, no. 4, pp. 345–348, 2010.

[10] S. Bhatia, S. K. Chalup, *et al.*, "Segmenting salient objects in 3D point clouds of indoor scenes using geodesic distances," *Journal of Signal and Information Processing*, vol. 4, no. 03, pp. 102–108, 2013.

[11] B. Douillard, J. Underwood, N. Kuntz, V. Vlaskine, A. Quadros, P. Morton, and A. Frenkel, "On the segmentation of 3D LIDAR point clouds," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 2798–2805, IEEE, 2011.

[12] J. Palnick, *Plane Detection and Segmentation for DARPA Robotics Challenge*. PhD thesis, Worcester Polytechnic Institute, 2014.

[13] M. Y. Yang and W. Förstner, "Plane detection in point cloud data," in *Proceedings of the 2nd int conf on machine control guidance, Bonn*, vol. 1, pp. 95–104, 2010.

[14] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[15] C. Premebida, O. Ludwig, and U. Nunes, "LIDAR and vision-based pedestrian detection system," *Journal of Field Robotics*, vol. 26, no. 9, pp. 696–711, 2009.

[16] M. Himmelsbach, A. Müller, T. Lüttel, and H.-J. Wünsche, "Lidar-based 3D object perception," in *Proceedings of 1st international workshop on cognition for technical systems*, vol. 1, 2008.

[17] A. Serna and B. Marcotegui, "Detection, segmentation and classification of 3D urban objects using mathematical morphology and supervised learning," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 93, pp. 243–255, 2014.

[18] J. Ahlstedt and J. Villysson, "Classification of objects using a Velodyne LIDAR scanner," tech. rep., Department of Mathematics, Chalmers Tekniska Högskola, 2013.

[19] D. Habermann, A. Hata, D. Wolf, and F. S. Osorio, "Artificial neural nets object recognition for 3D point clouds," in *Intelligent Systems (BRACIS), 2013 Brazilian Conference on*, pp. 101–106, IEEE, 2013.

[20] N. Chehata, L. Guo, and C. Mallet, "Airborne LIDAR feature selection for urban classification using Random Forests," *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 38, no. Part 3, p. W8, 2009.

[21] B. Douillard, J. Underwood, V. Vlaskine, A. Quadros, and S. Singh, "A pipeline for the segmentation and classification of 3D point clouds," in *Experimental Robotics*, pp. 585–600, Springer, 2014.

[22] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*, vol. 112. Springer, 2013.

[23] S. Haykin, *Neural Networks: A Comprehensive Foundation.* Upper Saddle River, NJ, USA: Prentice Hall PTR, 2nd ed., 1998.

[24] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Neural Networks, 1993., IEEE International Conference on Neural Networks*, pp. 586–591, IEEE, 1993.

[25] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *The International Journal of Robotics Research*, vol. 32, pp. 1231–1237, 2013.

[26] C. Igel, V. Heidrich-Meisner, and T. Glasmachers, "Shark," *Journal of Machine Learning Research*, vol. 9, pp. 993–996, 2008.