FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Klassifikation Urbaner Punktwolken Mittels 3D CNNs

In Kombination mit Rekonstruktion von Gehsteigen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Visual Computing

eingereicht von

Lisa Maria Kellner, BSc
Matrikelnummer 01428183

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.-Prof. Dipl.-Ing. Dr.techn. Dr.h.c. Werner Purgathofer
Mitwirkung: Dipl.-Ing. Dr. Stefan Maierhofer

Wien, 23. Februar 2021

Lisa Maria Kellner	Werner Purgathofer

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Classification of Urban Point Clouds Using 3D CNNs

In Combination with Reconstruction of Sidewalks

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Visual Computing

by

Lisa Maria Kellner, BSc
Registration Number 01428183

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.-Prof. Dipl.-Ing. Dr.techn. Dr.h.c. Werner Purgathofer
Assistance: Dipl.-Ing. Dr. Stefan Maierhofer

Vienna, 23rd February, 2021

Lisa Maria Kellner	Werner Purgathofer

Erklärung zur Verfassung der Arbeit

Lisa Maria Kellner, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 23. Februar 2021

Lisa Maria Kellner

Danksagung

Es war eine lange Reise vom Beginn meines Studiums bis zu dieser Diplomarbeit und ich habe währenddessen sehr viel Unterstützung von Familie, Freunden und Kollegen erhalten. Deshalb möchte ich mich an dieser Stelle bedanken.

Zuerst, möchte ich mich bei VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH bedanken, welches mir diese Diplomarbeit ermöglicht hat. Die VRVis Forschungs-GmbH wird im Rahmen von COMET – Competence Centers for Excellent Technologies (854174, 879730) durch BMK, BMDW, Land Steiermark, Steirische Wirtschaftsförderung – SFG, Land Tirol und Wirtschaftsagentur Wien – Ein Fonds der Stadt Wien gefördert. Das Programm COMET wird durch die FFG abgewickelt. All meinen Kollegen am VRVis sei ebenfalls ein Dank ausgesprochen, für die fachliche Hilfe, Diskussionen, Anregungen und Unterstützung während meines Studiums und dieser Arbeit. Dabei ist ein ganz besonderer Dank an Stefan, Attila, Michi, Harri, Georg, Andi, Thomas, Lui, Rebecca, Janne, Maria und Dani gerichtet.

Ebenfalls möchte ich mich bei meinen Eltern bedanken, welche mich während des gesamten Studiums unterstützt haben. Hier sei auch ein großer Dank an meine restliche Familie und Freunde gerichtet, die mir während meines Studiums zur Seite gestanden haben.

Außerdem möchte ich mich bei CycloMedia Deutschland, LiDAR Point Cloud und Stadtentwässerungsbetriebe Köln, AöR für das Bereitstellen der Daten aus Köln bedanken.

Acknowledgements

It has been a long journey from the beginning of my studies to this thesis and I have received a lot of support from family, friends and colleagues during this time. Therefore, I would like to take this opportunity to thank them.

First, I would like to thank VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH for making this thesis possible. VRVis is funded by BMK, BMDW, Styria, SFG, Tyrol and Vienna Business Agency in the scope of COMET - Competence Centers for Excellent Technologies (854174, 879730) which is managed by FFG. I would also like to thank all my colleagues at VRVis for their professional help, discussions, suggestions and support during my studies and this thesis. Special thanks go to Stefan, Attila, Michi, Harri, Georg, Andi, Thomas, Lui, Rebecca, Janne, Maria and Dani.

I would also like to thank my parents, who supported me throughout my studies. I would also like to thank the rest of my family and friends, who have supported me during my studies.

Furthermore, I would like to thank CycloMedia Deutschland, LiDAR Point Cloud and Stadtentwässerungsbetriebe Köln, AöR for providing the data from Cologne.

Kurzfassung

LiDAR-Geräte sind in der Lage die physische Welt sehr genau zu erfassen. Daher werden sie häufig für die 3D-Rekonstruktion verwendet. Leider können solche Daten sehr schnell sehr groß werden und meist ist nur ein Bruchteil der Punktwolke tatsächlich von Interesse. Daher wird die Punktwolke vorher gefiltert, um Algorithmen nur auf die für sie relevanten Punkte anzuwenden. Eine semantische Information über die Punkte kann für eine solche Filterung verwendet werden. Die semantische Segmentierung einer Punktwolke ist ein beliebtes Forschungsgebiet und auch hier gibt es in den letzten Jahren einen Trend in Richtung Deep Learning. Jedoch sind Punktwolken, im Gegensatz zu Bildern, nicht strukturiert. Daher werden Punktwolken oft rasterisiert, was jedoch in einer Weise geschehen muss, sodass die zugrunde liegende Struktur gut erhalten bleibt.

In dieser Arbeit wird ein 3D Convolutional Neural Network für die semantische Segmentierung einer LiDAR-Punktwolke entwickelt und trainiert. Dabei wird eine Punktwolke mittels einer Octree-Datenstruktur repräsentiert, welche es ermöglicht, nur relevante Teile zu rasterisieren. Denn nur dichte Teile der Punktwolke, in denen sich wichtige Informationen über die Struktur befinden, werden immer weiter unterteilt. Das ermöglicht es, die Knoten eines gewissen Levels im Octree zu nehmen und diese als Daten zu rasterisieren.

Es gibt viele Anwendungsgebiete für 3D Rekonstruktionen basierend auf Punktwolken. In einem städtischen Szenario können das zum Beispiel ganze Stadtmodelle oder einzelne Gebäude sein. Jedoch wird in dieser Arbeit die Rekonstruktion von Gehwegen untersucht. Da bei Überflutungssimulationen in Städten eine Erhöhung von ein paar Zentimetern einen großen Unterschied machen kann und Wissen über die Bordsteinegeometrie hilft die Simulation genauer zu machen. Bei der Gehwegrekonstruktion wird zuerst, basierend auf der semantischen Information des 3D CNNs, die Punktwolke gefiltert und dann werden Punktmerkmale berechnet, um die Punkte des Bordsteins zu erkennen. Mit diesen Bordsteinpunkten wird die Geometrie der Bordsteins, Gehweges und der Straße berechnet.

Insgesamt wird in dieser Arbeit ein Proof-of-Concept Prototyp für semantische Punktwolken-Segmentierung mittels 3D CNNs und basierend auf dieser ein Detektions- und Rekonstruktionsalgorithmus für Bordsteine entwickelt.

Abstract

LiDAR devices are able to capture the physical world very accurately. Therefore, they are often used for 3D reconstruction. Unfortunately, such data can become extremely large very quickly and usually only a small part of the point cloud is of interest. Thus, the point cloud is filtered beforehand in order to apply algorithms only on those points that are relevant for it. A semantic information about the points can be used for such a filtering. Semantic segmentation of point clouds is a popular field of research and here there has been a trend towards deep learning in recent years too. However, contrary to images, point clouds are unstructured. Hence, point clouds are often rasterized, but this has to be done, such that the underlying structure is represented well.

In this thesis, a 3D Convolutional Neural Network is developed and trained for a semantic segmentation of LiDAR point clouds. Thereby, a point cloud is represented with an octree data structure, which makes it easy to rasterize only relevant parts. Since, just dense parts of the point cloud, in which important information about the structure is located, are subdivided further. This allows to simply take nodes of a certain level of the octree and rasterize them as data samples.

There are many application areas for 3D reconstructions based on point clouds. In an urban scenario, these can be for example whole city models or buildings. However, in this thesis, the reconstruction of sidewalks is explored. Since, for flood simulations in cities, an increase in height of a few centimeters can make a great difference and information about the curb geometry helps to make them more accurate. In the sidewalk reconstruction process, the point cloud is filtered first, based on a semantic segmentation of a 3D CNN, and then point cloud features are calculated to detect curb points. With these curb points, the geometry of the curb, sidewalk and street are computed.

Taken all together, this thesis develops a proof-of-concept prototype for semantic point cloud segmentation using 3D CNNs and based on that, a curb detection and reconstruction algorithm.

Contents

CHAPTER 1

Introduction

1.1 Motivation

3D reconstruction is rebuilding the physical world based on captured data.

Reconstructions have a broad field of applications, like preservation of cultural heritage as statues from Michelangelo, relicts from Egypt or Buddha statues [GBS14] or serious games about it [MCB+14]. Further use cases are underwater robotic surveys [JRPWM10] or indoor scenes like KinectFusion, which can be used for augmented reality [IKH+11]. Also reconstructions of cities are of great interest, because they can be used for planning, city climate or noise propagation applications [Bre00] as well as emergency management, disaster control and for movies or computer games [MWA+13].

Such simple city models are used in Visdom [Vis], a simulation and visualization application for disaster management of floods. A screenshot of this software can be seen in Figure 1.1. They assist decision making in emergencies of floods caused by heavy rainfall, tsunamis or other disasters. The impact of short-term protections, like sandbags, or long-term protections can be simulated in advance to support decisions about safety arrangements. Based on this, emergency personnel can be trained accordingly and possible evacuation plans can be explored beforehand. In addition, the sewer system of a city can be added to the simulation to take their capacity and potential bottlenecks into account.

In case of floods, an increase in height of a couple of centimetres or a house driveway with a slope down can make a great difference in safety arrangements. Therefore, a geometric information about driveways or sidewalks would be of great interest for the Visdom-team to make their simulation even more accurate. Therefore, this thesis addresses the detection and reconstruction of sidewalk surfaces.

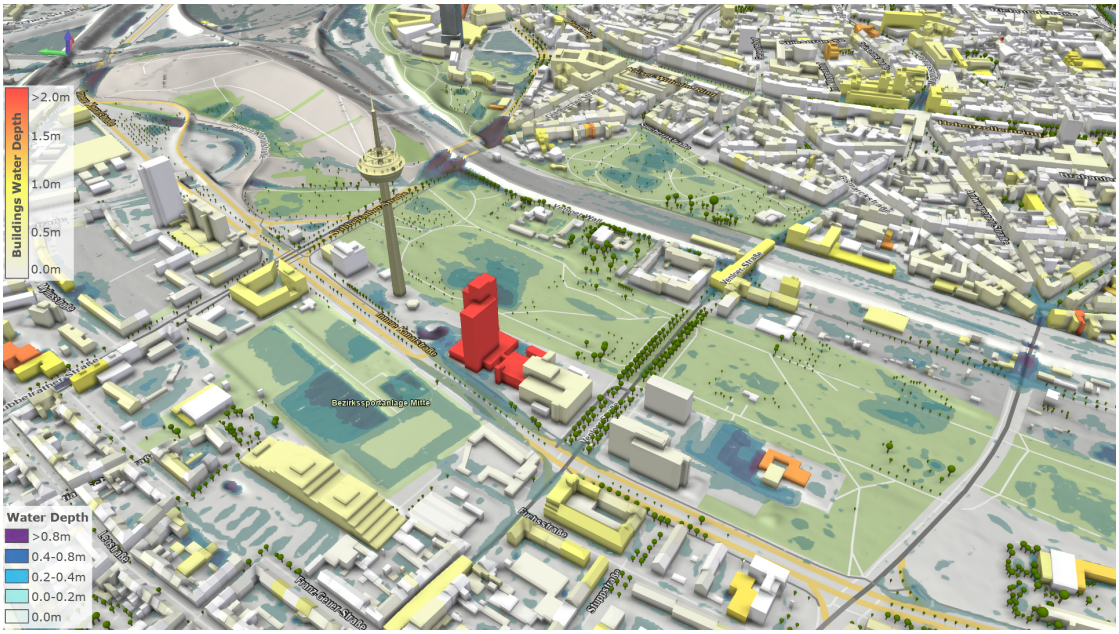


Figure 1.1: Screenshot of the Visdom flood simulation software. (Source: [Vis])

1.2 Problem Statement

Acquiring respective data for a 3D reconstruction is nowadays probably as easy as never before. Various technical instruments are available. Mobile phones are able to take high-resolution photos with meta-data like camera parameters or GPS location. Moreover, photos are not limited to the ground, because drones allows us to capture the world from above too. LiDAR devices scan their surrounding with high precision and yield dense point clouds. This data can then be combined with other information like cadastre data or with data from Open Streets Maps, an open source map with further geospatial tags.

Each type of data has its own advantages and disadvantages. Everyone can take photos with their smartphone, but the quality of photos strongly depends on the lighting conditions and are therefore prone to over- as well as underexposure. Airborne photography is not able to capture small structures and has to cope with occlusions from trees. Although, they are a good extension for terrestrial laser scanning data to close gaps. Photogrammetric point clouds are not as fine-grained as LiDAR data, but on the other hand, LiDAR data can become so huge that it can be challenging to process it efficiently. Geospatial data does not provide enough information for reconstruction, but is a good enhancement to improve algorithms. Therefore, to solve a problem it is inescapable to use the type of data as well as data structure fitting best to the problem and to combine different sources of data.

The reconstruction of sidewalks is based on LiDAR data, because laser scanners are able to capture subtle structures like an elevation of ten to fifteen centimetres. Laser scan

data of a whole city has up to several billion points and curbs are just a tiny part of them. It is therefore not feasible to search curb points in the whole point cloud. This would not only take many computational resources, but also a satisfying curb detection result would be hard to achieve. For the simple reason that by applying this problem to the whole city, it can either be over-modelled, which would result in many false positives or under-modelled, which would not detect many curb points. Since sidewalks are located on the ground and besides streets, the majority of points can be filtered beforehand to reduce the search area. This could either be achieved with some kind of ground plane filtering or by using semantic information. With semantic information applied to points, they can be filtered more precisely. For example, a car is on the ground and at or nearby a street too, but during the detection process, those car points would lead to many false positive curb points. Therefore, it would be great to remove car points from the point cloud beforehand too. Another advantage is, that this would not only help to limit the search area for this problem, since the semantic information could also be used for further reconstructions, like buildings, or for filtering vegetation and other unwelcome points. Hence, sidewalks are located near roads; information about the street network would be another great source of data. OpenStreetMap [Opea] provides it and many further geospatial information. However, the location of those tags is saved in another coordinate system as the LiDAR data and has to be converted beforehand. Due to the semantic filtering of the point cloud and the combination with further data sources, the search area is insomuch restricted, that detecting curb points can be achieved with basic spatial features and geometry can be generated with simple algorithms.

1.3 Aim of this Thesis

The aim of this thesis is to develop a proof-of-concept prototype for semantic point cloud segmentation and based on that, a curb detection and 3D reconstruction algorithm.

Convolutional neural networks are probably the most common deep learning method nowadays. They are used in a wide range of fields, such as image or speech recognition, face detection, text or video analysis and self-driving cars. They are outperforming other supervised classification methods by far, because during the convolutional operations not only low-level features are detected, but also high-level features and essential structures are distinguished. The essential processes like convolution or pooling can be easily adapted to the three-dimensional space. Therefore, a semantic segmentation of point clouds is achieved by building and training 3D convolutional neural networks. To decrease memory consumption and training time, the point cloud will not just be represented in a uniform grid structure, but with an octree. The cells of the octree in a certain level are then voxelized and used for the segmentation.

For curb detection, just ground points are used, which can easily be selected based on the semantic information. Multiple geometric features are calculated for each point and if a point fulfils all features, it is labelled as “curb”. Afterwards a filtering step removes false positive “curb” points and finally geometry is generated for sidewalks, curbs and street.

Therefore, the work of this thesis can be divided into two parts. First, for a semantic segmentation of a point cloud, a 3D convolutional neural network will be developed and trained. Secondly, curb points will be calculated with geometric point cloud features and based on them, geometry for the curb, sidewalk and street will be computed.

1.4 Structure of this Thesis

First, an overview about related work in point cloud segmentation as well as curb detection is given in Chapter 2, where section 2.1 discusses semantic segmentation based on handcrafted features, artificial neural networks and convolutional neural networks. Convolutional neural networks are categorized in 2D image segmentation networks, where the point cloud is projected into images first, and 3D convolutional neural networks, which are directly acting on the point cloud. Section 2.2 describes the curb detection process in point clouds on which the curb detection steps in this thesis are based on. In the course of this process, the point cloud is prefiltered, then spatial features are calculated and based on them a point is classified as curb or not. Afterwards, false positive labelled points are filtered and finally geometry is computed.

Chapter 3 describes semantic point cloud segmentation and thereby addresses data extraction in Section 3.1, the dataset used for training in Section 3.2, some network learning basics in Section 3.3, as well as the final network architecture in Section 3.4. Furthermore, network classification and segmentation of an unknown point cloud is shortly depicted in Section 3.5.

Chapter 4 is dedicated to the different steps of the curb detection and fitting process. Section 4.1 describes prefiltering of a point cloud and Section 4.2 explains and visualizes spatial point cloud features, including the integration of OpenStreetMap data. Section 4.3 addresses false positive filtering and finally, Section 4.4 discusses geometry fitting.

Chapter 5 addresses the network training process and its results in detail, with the composition of the training set in Section 5.1 and the hyperparameter tuning in Section 5.2. Furthermore, three networks with different grid sizes are evaluated: a network using a small grid size in Section 5.3, a network using a middle grid size in Section 5.4 and a network using a big grid size in Section 5.5.

Chapter 6 shows and discusses the results of the network training and semantic segmentation of the training set in Section 6.1 as well as the semantic segmentation and curb fitting results of real-world data in Section 6.2. A critical reflection on the work of this thesis is given in Section 6.3 and finally, Chapter 7 gives a conclusion and short outlook into future work.

CHAPTER 2

Related Work

This chapter gives an overview about related work in point cloud segmentation and curb detection. Section 2.1 addresses semantic point cloud segmentation based on handcrafted features as well as artificial neural networks. Thereby a focus is given to 2-dimensional convolutional neural networks, which are mapping the point cloud into images, 3-dimensional convolutional neural networks, which process the point cloud directly, as well as special for point clouds developed optimizations, like tree-based networks for example. Section 2.2 describes research in road boundary and curb detection of self-driving vehicles. Emphasis is given to the pipeline used in this thesis, where the point cloud is filtered first, then candidate points are detected and filtered for false positives, and finally geometry is fitted to the detected curb points.

2.1 Semantic Point Cloud Segmentation

Point clouds are a set of unstructured points either calculated by photogrammetry from several photos or captured directly by a scanning device.

The basic principle of generating photogrammetric point clouds is computing 2D image-features and determining their 3D position by finding the same feature in multiple overlapping photos. The most commonly used feature is the Scale Invariant Feature Transform (SIFT), which is invariant to translation, scaling and rotation of the image [Low04]. Structure-from-Motion calculates the location and orientation of the photos as well as scene geometry, without any prior knowledge about camera positions, by using bundle adjustment and such image-features. 3D points, and thus the scene geometry, are computed by knowledge acquired in this process about the same feature in several photos and by using triangulation [WBG+12]. Figure 2.1 shows such a photogrammetric point cloud.

Figure 2.1: A photogrammetric point cloud with location and orientation of the photos. (Source: adapted from [SKMW17])

Light Detection and Ranging (LiDAR) devices, or colloquially called laser scanners, are able to capture the physical world by shooting out laser beams. When a laser beam hits an object, it is reflected and returns to the scanning device. The elapsed time, until the return of the beam, is measured and based on that, the distance from the object to the laser scanner is obtained and therefore the 3D position of the hit-point as well. Point clouds captured with LiDAR devices are invariant to illumination, more accurate and denser than photogrammetric point clouds. Such devices are able to capture huge areas, but the resulting point clouds could become very large very quickly.

Semantic point cloud segmentation is the assignment of discrete class labels for each point in the point cloud individually.

In this thesis the main focus is on supervised learning techniques, where a classifier learns the correlation of set $X := \{x_1, x_2, ..., x_n\}$ of n data samples to the corresponding set $Y := \{y_1, y_2, ..., y_n\}$ of n ground truth labels. A trained classifier takes a data sample x_i as input and predicts a class label y_i, where i is the index of the data sample and its according ground truth value in the sets X and Y respectively.

As mentioned above, the set X is consisting of n data samples, where each data sample is an m-dimensional feature vector by itself. Therefore, X is nm-dimensional. The set Y encodes the affiliation of data samples to c classes and is commonly represented by a one-hot encoding of the categorical class values. Thus, Y is nc-dimensional.

One-hot encoding is the mapping of categorical values to binary feature vectors. An example is given in Table 2.1, where three categories “building”, “vegetation” and “terrain” are each encoded into a three-dimensional binary feature vector, where every category is assigned to a fixed position in the vector. The values in a feature vector are zero everywhere, except for the position of the represented class, which has value one. The according feature vectors of our example are $[1, 0, 0]$, $[0, 1, 0]$ and $[0, 0, 1]$ respectively. This method is not restricted to class labels; it is used to encode categorical values in data samples as well. Since a numerical encoding for categories, like in our example 1, 2

and 3, would cause a machine learning algorithm to rate "terrain" higher as "building", because of $3 > 1$ and one-hot encoding solves this problem.

categorical name	numerical encoding	one-hot encoding
building	1	1 0 0
vegetation	2	0 1 0
terrain	3	0 0 1

Table 2.1: An example of mapping categories to numerical values and an according one-hot encoding of them.

2.1.1 Handcrafted Features

Handcrafted features are calculated for each single point individually. They usually consider information about a point's neighbourhood, which can be defined by the number of the nearest k points or a certain radius.

Commonly used features describing the geometric shape of a point's neighbourhood are based on the eigenvalues λ_i, where $\lambda_1 \geq \lambda_2 \geq \lambda_3$, of the covariance matrix. The covariance matrix Σ of a point p and its neighbourhood consisting of k points is defined by:

$$\begin{aligned} \Sigma &= \frac{1}{k}\sum_{i=1}^{k}(p_i - \mu)(p_i - \mu)^T \\ \mu &= \frac{1}{k}\sum_{i=1}^{k} p_i \end{aligned} \tag{2.1}$$

Such features - amongst many others - are linearity, planarity, sphericity, omnivariance, sum of eigenvalues, curvature and eigenentropy, which are listed in Table 2.2 [WJHM15] [HWS16] [TGDM18].

Weinmann et al. [WUH+15] are optimizing their point cloud features by choosing the best neighbourhood size for a single point. This makes features better distinguishable and therefore improves the classification. They are calculating the optimal amount of k nearest neighbours for a point p individually by building the covariance matrix of the neighbourhood around p with an altering k. The best k is chosen based on the minimum Shannon entropy [Sha48] of the normalized eigenvalues or features. A downside of this approach is the increased runtime, since the optimal neighbourhood is computed for each point individually. Hackel et al. [HWS16] are approximating the neighbourhood by iteratively downsampling the point cloud to generate a multi-scale pyramid. The features are then computed per scale with a fixed k. This method computes features very fast and concurrently achieves high classifications results. Thomas et al. [TGDM18] are using a spherical neighbourhood based on a radius instead. They use a spherical neighbourhood instead of a fixed number of neighbours, because features with such a neighbourhood have a consistent geometrical meaning and the number of points in the neighbourhood

become a feature by themselves. Besides, classifier with those features are performing better than with features computed with a fixed amount of points.

Linearity	$\frac{\lambda_1-\lambda_2}{\lambda_1}$
Planarity	$\frac{\lambda_2-\lambda_3}{\lambda_1}$
Sphericity	$\frac{\lambda_3}{\lambda_1}$
Omnivariance	$\sqrt[3]{\lambda_1\lambda_2\lambda_3}$
Sum of eigenvalues	$\lambda_1+\lambda_2+\lambda_3$
Curvature	$\frac{\lambda_3}{\lambda_1+\lambda_2+\lambda_3}$
Eigenentropy	$-\sum_{i=1}^{3}\lambda_i \ln \lambda_i$

Table 2.2: Equations of some point cloud features.

2.1.2 Artificial Neural Networks

Besides descriptive features, a classifier is needed to map those features to class labels. Such classifier could be decision trees [BFOS84], random forests [Bre01], support vector machines (SVM) [CV95] or artificial neural networks (ANN) for example. Artificial neural networks, or short neural networks (NN), are inspired by mechanics of mammal's brains, where signals are transported between neurons.

The basis for neural networks was Frank Rosenblatt's perceptron algorithm [Ros58], which could linearly separate binary data. It calculates a decision boundary between two classes, which is an $(n-1)$-dimensional hyperplane. A weight vector w together with a bias b represent this hyperplane, where w is normal to the plane and therefore defines its orientation and b determines its position. The class affiliation of a new data sample x is defined by $y(x) = w^T x + b$, where $y(x) > 0$ means that x lies in front of the separation plane, respectively $y(x) < 0$ states that x is located behind. Based on that, the classifier assigns x to one of the two classes [Bis06].

Feedforward neural networks, also known as multilayer perceptrons (MLPs), are directed acyclic graphs, where each edge in pointing forward. They are able to approximate a mapping function f^* from an input x to its according class label y. This approximated mapping function is defined as $y = f(x;\theta)$, where θ is searched during the training, such that f becomes as similar to f^* as possible. This is achieved by composing multiple functions, where the layerwise graph structure describes how those functions are assembled. For example, if there is a network with three layers and some functions representing each layer, say f_1 for the first, f_2 for the second and f_3 for the third layer, the assembled

functions would look like $f(x) = f_3(f_2(f_1(x)))$. If we want to model a linear function, θ would consist of w and b, resulting in $f(x; w, b) = x^T w + b$. A loss function, like the mean squared error (MSE), shown in Equation 2.2, is then minimized with respect to w and b [GBC16]. Note, Equation 2.2 is adapted from [GBC16].

$$J(\theta) = \frac{1}{|X|} \sum_{x \in X} (f^*(x) - f(x; \theta))^2 \tag{2.2}$$

However, common problems are not linear. In order to be able to learn nonlinear models, an affine transformation and a nonlinear function, named activation function, are used. Consider a network with a single hidden layer consisting of $h = f_1(x; W, c)$, where W are weights and c are biases, and $y = f_2(h; w, b)$, resulting in $f(x; W, c, w, b) = f_2(f_1(x))$. By applying this method to h, this would result in $h = g(W^T x + c)$, where g is the activation function and the final network would be defined as $f(x; W, c, w, b) = w^T g(W^T x + c) + b$ [GBC16]. This example is visualized in Figure 2.2.

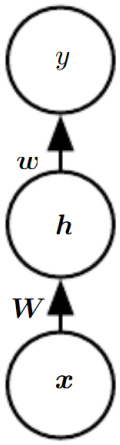


Figure 2.2: An example of a feedforward neural network with a single hidden layer mapping an input x to an output y with an intermediate layer h, where W maps x to h and w maps h to y. (Source: adapted from [GBC16])

As mentioned above, problems are commonly not linear and by using an activation function, neural networks are able to learn such a nonlinear mapping. There are many different activation functions, the most common ones are the binary step, linear, sigmoid, tanh, ReLu and softmax function, which are shown in Table 2.3. There are so many different ones, because each activation function has its advantages and disadvantages. Besides that, the problem a network should solve or in which layer the activation function is used influences which kind of function is used too. For example, a classification problem performs better with sigmoid functions, but on the other hand, sigmoid as well as tanh functions tend to cause gradients to become zero, which we want to avoid. The binary step function can be used for binary classifier, but its gradient is zero. Linear activation

functions do not have zero gradients, but the network will not perform well on hard problems. ReLU is widely used and performs better in most cases than other functions, but it could cause gradients to become zero too. In addition to that, ReLU can only be used in hidden layers, where sigmoid and tanh are not used there. Furthermore, the sigmoid function is used for binary classification problems only and for multi-class problems, the softmax activation function is used instead [SSA20].

The basic graph structure of a feedforward neural network consists of an input layer, one or multiple hidden layers and one output layer. Figure 2.3 visualizes the simplest possible network with three layers: an input layer depicted with green nodes, one hidden layer in blue and the output layer in red.

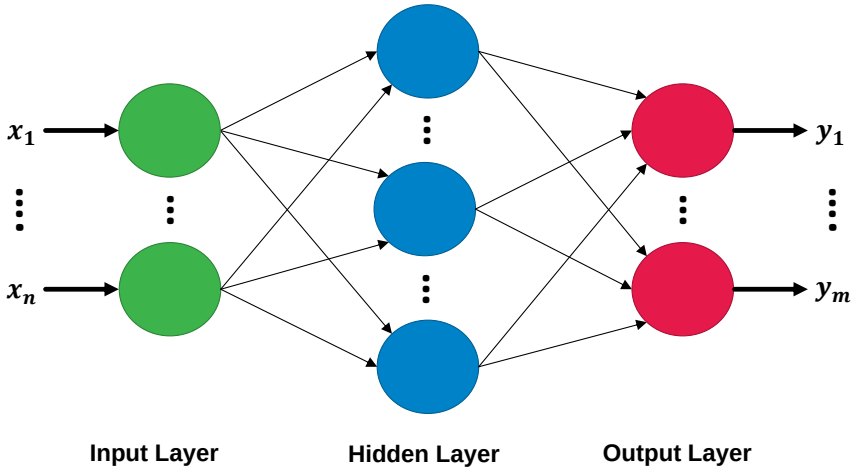


Figure 2.3: A visualization of a feedforward neural network, also known as multilayer perceptron (MLP).

Pioneering work in 3D point cloud classification and segmentation using feedforward neural networks was accomplished by Qi [QSMG17] et al. with the PointNet architecture. They describe in their work that PointNet processes the whole point cloud directly at once and returns either a classification for the whole point cloud or a pointwise segmentation result. Its architecture consists of two transformation networks, which are subnetworks predicting and applying an affine transformation matrix to make the network invariant for transformations of the point cloud, followed by a MLP respectively, ensued by a max pooling layer (see: Figure 2.5) to extract a global feature vector. For point cloud classification, this global feature is directly processed by a MLP. For segmentation, all point features are combined with the global feature by concatenation and new point features are extracted from this combination first. With this method, these new features are encoding local as well as global information. They achieve with their PointNet architecture equal or even better classification results than state of the art at that time.

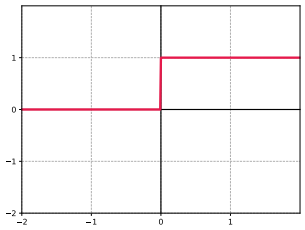
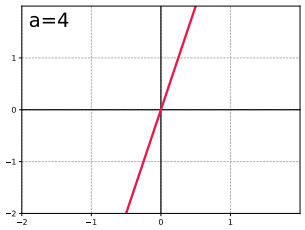
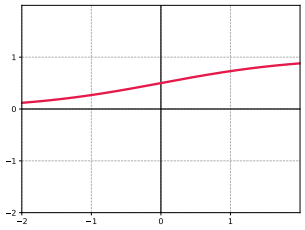
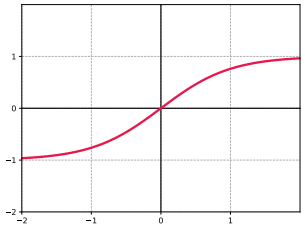
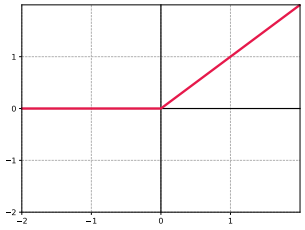
Binary Step Function	$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	
Linear Function	$f(x) = ax$	
Sigmoid Function	$f(x) = \frac{1}{(1+e^{-x})}$	
Tanh Function	$f(x) = tanh(x)$	
ReLU Function	$f(x) = max(0, x)$	
Softmax Function	$f(x)_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}}$	

Table 2.3: Most commonly used activation functions. (left) Name of the function, (middle) function $f(x)$ itself and (right) plot of the function (Source: [SSA20]).

Qi et al. [QYSG17] improved this work, calling it PointNet++, by developing a hierarchical neural network, which is able to learn local structures as well. They describe that PointNet++ divides the point cloud into small overlapping regions and extracts local features by using PointNet. Local features are then grouped together to form higher-level features and this process is repeated until the whole point cloud is encoded. Furthermore, to address different densities within the point cloud they introduce a multi-scale grouping (MSG) and a multi-resolution grouping (MRG) by concatenating features from different scales or resolutions. Those features are generalizing in different point densities. Engelmann et al. [EKHL17] also extended PointNet to increase its classification performance by adding larger spatial context to the network. They achieve this by either using multi-scale or neighbouring regions in combination with either so-called Consolidation Units (CU) or Recurrent Consolidation Units (RCU). CU's and RCU's are encoding information about neighbouring features into a feature.

A further method using deep learning is the graph-based approach from Landrieu et al. [LS18]. They depict in their work that the vertices of their graph are called superpoints, which are representing basic shapes of the point cloud, and the edges are describing the relationship of those superpoints. For segmentation, the superpoints are converted to a descriptor using PointNet and then classified by a convolutional network, which is specialized for graphs. They chose a superpoint graph-based approach because of several advantages: objects are easier to classify than single points, relationships between those objects can be used for the classification as well and the size of the graph is bounded by the amount of different shapes, which is rather small in comparison to the amount of points in a point cloud. Furthermore, they are outperforming with their approach the state of the art at that time.

2.1.3 Convolutional Neural Networks

Handcrafted features in combination with neural networks are a powerful tool for mapping data samples to their class labels. However, in image classification exist a more sophisticated method called convolutional neural networks (CNNs). CNNs are combining feature extraction and classification into a single neural network, where specific operations are extracting high-level features, which are then mapped by an appending neural network to class labels. Since CNNs are current state of the art in image classification and segmentation, this approach is used in various ways for point cloud segmentation too.

2D Convolutional Neural Networks

The main parts of a CNN are convolution and pooling operations to extract features from an image, as well as a neural network, which maps those features to distinct classes or labels.

The *convolution operation* consists of a matrix called kernel, which shifts like a sliding window over the image matrix and computes new image values by performing a convolution. A convolution is the multiplication of each kernel matrix value with each according

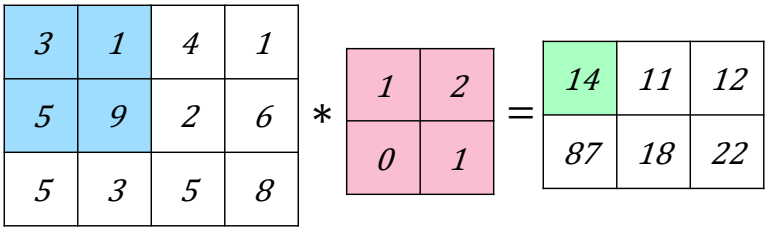


Figure 2.4: An example of a convolution operation. Such an operation $conv = m * k$ consists of an image matrix m and a kernel k. The kernel shifts like a sliding window over the image matrix and computes new image values by multiplying and adding according values. The position of the sliding window is visualized in blue, the kernel in pink and the resulting value in green, which is obtained as follows: $3*1+1*2+5*0+9*1=14$.

image value inside the sliding window, followed by summing up these values. Figure 2.4 shows an example of a convolution. The image part to be convoluted is coloured blue, the kernel pink and the resulting value green. The convolution is calculated as follows: $3*1+1*2+5*0+9*1=14$.

As mentioned above, the second main part of CNNs are pooling operations. The most commonly used pooling operation is *max-pooling* and its purpose is to reduce image size. It uses a sliding window too, where the biggest value inside is transferred into the downsampled image matrix. An example with a pooling size of 2×2 is shown in Figure 2.5.

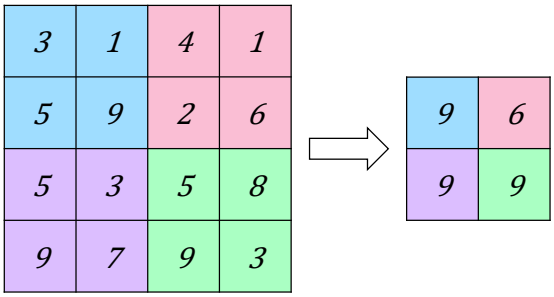


Figure 2.5: An example of a max-pooling operation. Such an operation downsamples the image matrix by using a sliding window, which only transfers the highest value inside into the reduced image matrix. (left) Image matrix with four positions of an 2×2 sliding window and (right) the downsampled image matrix with the transferred values in according colors.

CNNs are assigning a single class to the whole image, but for semantic segmentation, a pixel-wise labelling is needed. Segmentation networks, like the U-Net from Ronneberger et al. [RFB15] use convolution and pooling operations to downsample the image first and so called up-convolutions (or sometimes also referred to as deconvolutions) to upsample

the image again, whereas a high-resolution segmentation map is received. The down- and upsampling approach is often denoted as encoding and decoding, like in the SegNet from Badrinarayanan et al. [BKC17], which is visualized in Figure 2.6. There it can be seen, that their encoder network consists of 13 convolutional as well as 5 pooling operations and the decoder network of the same amount of upsampling and convolutional operations. Those inverse pooling and convolution operations are also called unpooling and deconvolution respectively [ZTF+11].

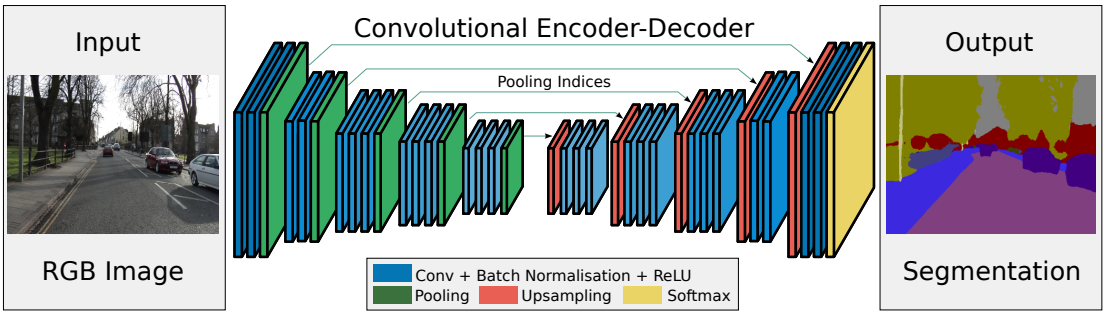


Figure 2.6: Semantic segmentation network architecture with encoder-decoder structure. (Source: [BKC17])

Boulch et al. [BGSA18] use such image segmentation methods for semantic point cloud segmentation and introduce a novel and efficient network named SnapNet. First, they thin the point cloud and generate a mesh from it. After that, they generate RGB and depth composite image views in 2D, which they call snapshots, either in a random or multi-scale way. Then, they segment those images using their own implementations of SegNet and U-Net. The pixelwise classifications are back-projected to the mesh-vertices, which can be done easily, because they saved the information if the according mesh-faces are visible from the snapshot. After that, the classifications are mapped back to the original point cloud by using the labels of the nearest mesh-vertices. This method performs as good or even outperforms the state of the art at that time, especially in indoor scenes.

Lawin et al. [LDT+17] have a similar approach, they create multiple views from a point cloud by rendering it into 2D images. They argue that a 2D image based segmentation approach results in a better accuracy, because images can have higher resolutions than voxelized point clouds, due to the increased memory consumption of voxelized data. As further benefit, they state already existing labelled datasets for image segmentation. They use RGB, depth and normal images and project the point cloud into images by using a modified version of point splatting to eliminate noisy foreground and non-visible points. Convolutional neural networks process all different image types and provide a semantic segmentation. The final prediction is the sum of those segmentation outputs. These pixelwise predictions are mapped back into the point cloud and the sum over all views gives the final classification for the points. This approach outperforms the state of the art of that time.

Besides semantic point cloud segmentation, a further field of research is dedicated to 3D object classification using similar techniques. Su et al. [SMKLM15] are introducing such a method to classify 3D objects, by rendering a 3D shape into 2D images from multiple views and classify those images by using a CNN.

3D Convolutional Neural Networks

Since convolution and pooling operations can easily be adapted for 3-dimensional space, another field of research for point cloud segmentation as well as 3D object classification has emerged based on 3D convolutional neural networks (3D CNNs).

Ji et al. [JXYY13] introduced 3D convolutions in 2013 for human action recognition in videos. They wanted to capture motion of people over multiple frames to not only include spatial, but temporal information too. Thus, they stacked multiple consecutive video frames and so time became the third dimension. For 3-dimensional convolution operations, the kernel is 3D as well and the convolution is executed over all three dimensions.

Wu et al. [WSK+15] introduced 3D ShapeNets as well as ModelNet, a 3D CNN and a large CAD model dataset respectively. They depict data with a 3D volumetric grid of size $30 \times 30 \times 30$, where each voxel holds a binary value, representing if it is inside or outside the mesh. To transform these binary values to a probability distribution - the representation of a 3D shape - Convolutional Deep Belief Networks are introduced. Furthermore, they use no pooling layers within this network, because they are not only classifying objects but also reconstructing them and pooling operations would increase uncertainty regarding the reconstruction. Additionally, they pre-train the network first and fine-tune it afterwards to enhance the network's performance. Besides introducing a new model dataset, which has by far more categories and objects per category than datasets at that time, they surpass the state of the art in object classification at that time too.

A similar approach for object recognition presented by Maturana et al. [MS15] is VoxNet, which is also a 3D CNN working with voxelized point cloud data, but it is not restricted to CAD models. In their work they distinguish between free, occupied and unknown voxels and use ray tracing to calculate these three different types of occupancy values: A binary value, stating if the voxel is occupied or not; a density value, which is the probability that a voxel would block a ray and a hit value, which does not distinguish between free and unknown space. The grid size of the voxelized data is $32 \times 32 \times 32$ and the network consists of two convolutional, one pooling and two fully connected layers.

Garcia-Garcia et al. [GGG+16] build their work on ShapeNet [WSK+15] as well as VoxNet [MS15] and introduce a 3D CNN named PointNet[1] for object class recognition. As they [GGG+16] describe in their work, they use a volumetric occupancy grid to represent the data and simply map the points into voxels. The occupancy value of a voxel

[1] Despite the same name, the PointNet from [GGG+16] is a different PointNet than the one from [QSMG17].

is the normalized number of points in it and the network is composed of a convolutional followed by a pooling layer, again a convolutional as well as a pooling layer and two fully connected layers. They are as good as the state of the art at that time, but their classification is faster, which enables real-time object classification. Qi et al. [QSN+16] are generating different orientations of the 3D object as input for a 3D CNN and combine their information within the network to enhance the classification result.

Huang et al. [JS16] focus on segmentation of large point clouds and use 3D CNNs with binary occupancy values and a resolution of $20 \times 20 \times 20$ for this purpose. As they depict in their work, for training, the point cloud is voxelized at a set of center points within a certain radius. The center points are randomly selected, but balanced across all classes available in the dataset. For classification, the point cloud is first voxelized as a whole and those voxels are the input for the network. The class label of a voxel grid is a majority vote of the labels of all points within it. SEGCloud, introduced by Tchapmi et al. [TCA+17], is also voxelizing the point cloud before a 3D fully convolutional neural network is used to classify each voxel. They describe that they apply a trilinear interpolation to interpolate the voxel prediction to the original point cloud. This interpolation is then combined with other point features using a fully connected CRF [KK11] to classify each point.

Improved Convolutional Neural Networks for Point Clouds

Conventional CNNs are working with rasterized data, since 2D images are already rasterized this does not cause any issues. However, 3D point clouds are not rasterized and vary much in point density, from regions with a very high density to regions with no points at all. Rasterizing point clouds not only extend the dimension from 2D to 3D, which is a cubic growth in memory consumption, but empty space is rasterized as well, which does not include any information and is therefore unnecessarily allocating memory and slowing down the network. Therefore, many different approaches are developed to reduce memory consumption and computation time, as for example tree-based methods. Klokov et al. [KL17] are using feed forward neural networks based on kd-trees instead of rasterized data and calling it Kd-network. The input to such a network is the point cloud's kd-tree and the network is implemented based on a kd-tree data structure instead of a grid structure. Riegler et al. [ROUG17] are working with shallow unbalanced octrees and their encoding with a bit-string, stating if a node is split or not. Based on that data structure they implemented convolutional, pooling and unpooling operations. Wang et al. [WLG+17] introduced another approach based on octrees, where they use shuffle keys and labels to store the neighbour- and parenthood of the nodes. Hence, this information is needed to perform convolution, pooling, unpooling and deconvolutional operations based on octrees. Recent research tackles this problem by improving the convolution operation itself in regard of point clouds, like Thomas et al. [TQD+19] or Boulch et al. [Bou19].

2.2 Curb Detection

The detection of curbs and road-boundaries is a popular research topic in the field of autonomous driving vehicles for a long time, since knowledge about street borders is essential to ensure a safe ride. Over years, various different sensors and techniques were explored to perceive the road and its properties. Hillel et al. [BHLLR14] described in their survey that monocular vision, stereo imaging, radar, the combination of GIS, GPS along with inertial measurement units and LiDAR are among them. However, as they state further, camera based approaches are prone to occlusions, as from other vehicles, shadows, weather conditions and illumination. In addition to that, the calculated 3D information of stereo images is not as accurate as LiDAR data. Radar devices have a low resolution and are not able to detect fine structures. Therefore, they are used for obstacle detection and on-/off-road differentiation. The positioning information is just used as additional data, since they are not as accurate and would need precise as well as up-to-date maps. Since, LiDAR does not depend on illumination and directly provides 3D data; it is often used for curb detection.

Already in 2010, Zhang et al. [Zha10] presented their work on road-edge detection based on 2D LiDAR data. Their algorithm contains five stages: candidate point-set selection, feature extraction, classification, false alarm mitigation and road curb detection. The basic framework for the curb detection pipeline is formed by these steps, except for the classification, and can be still found in recent research as well as in this thesis.

2.2.1 Point Cloud Pre-Filtering

Since the following steps in the curb detection pipeline are applied pointwise, the point cloud is roughly filtered beforehand to reduce the amount of points to process and therefore computation time. For this case, the filtering does not need to be highly accurate, because its purpose is to omit regions of the point cloud, where definitely no curb is located.

Zhao et al. [ZY12] form a regular voxel grid from the point cloud and define the voxel with lowest altitude as ground data. They use a voxel grid to quickly access points inside a voxel and to model neighbourhood relations between points. Then, they fit a plane to points stored in these voxels and the fitting points are determined as ground points. Zhang et al. [ZWWD18] are using a plane-based method too, which is applied on points below the scanning device. They use the RANSAC [FB81] algorithm to fit a plane and ground points are determined based on a loose threshold. Another plane-based method is proposed by Wang et al. [WWHY19], they are first splitting the point cloud into multiple segments and fit a plane in all segments, by also using the RANSAC method. They are using such a segment based approach to be able to handle slopes on the road.

2.2.2 Feature Extraction

Curbs have very distinctive spatial properties; they are between 10 to 15 centimetres high, roads and curbs are orthogonal as well as parallel to one another and features based on the covariance in a certain neighbourhood can describe curbs too. Furthermore, the laserline of a LiDAR device has a different pattern when it hits an obstacle. This and many more properties are used to find curb points in the pre-filtered point cloud. Usually, several features are computed per point and if a point satisfies all these conditions, it is identified as a curb point. Some of these features are described below.

Height difference is probably the most common used spatial feature to describe curbs in autonomous driving. Zhang et al. [ZWW+15] are using a sliding window approach to speed up the feature calculation. For each sliding window the maximum and minimum in height is searched and if the difference between them is greater than a certain threshold, it probably contains curb points. Zhao et al. [ZY12] are computing the difference in height for each voxel and are restraining it with an upper and lower threshold too. Wang et al. [WWHY19] are using, besides the difference between the highest and lowest height values, also the standard deviation of height values of points in the same laser line. They constrain the difference with an upper as well as lower and the standard deviation with an upper threshold.

The spatial structure of curbs can also be described with the *distance between two neighbouring points* of the same laser line. Zhang et al. [ZWWD18] are calculating the horizontal and vertical distance and are bounding them with a lower threshold. Wang et al. [WWHY19] are using the same method, but they are only verifying the horizontal distance.

Since roads and curbs are usually orthogonal to each other, the angle between them is another well describing feature. Therefore, Zhang et al. [ZWWD18] introduced a feature describing this property, by calculating the *angle between two vectors* starting at the same point. They create those vectors by summing up n previous or following points respectively, along the laser line from the starting point and divide the result by n. If the starting point lies on a curb, the angle between those two vectors would be smaller than the angle between vectors originating from a road point. Wang et al. [WWHY19] are using this method too, but they are not normalizing the sum with the amount of used points and define the upper threshold with 170 degrees.

Another set of features is based on the *eigenvalue decomposition of the covariance matrix* in a certain neighbourhood of points. Zhao et al. [ZY12] compute the normal vector of a point by using the according eigenvector of the smallest eigenvalue. This normal is then checked for perpendicularity to the car and if so, the point is probably part of a curb. Whereas, Fernández et al. [FILS14] are using the eigenvalues of a weighted covariance matrix to estimate the curvature in the area around a point. They do this by dividing the smallest eigenvalue through the sum of all eigenvalues and bounding the result with a lower and upper threshold. Furthermore, they state that, curbs can be categorized into very small, small, regular and big obstacles with increasing threshold values.

Smoothness of points in a certain neighbourhood is a further curb descriptor. Wang et al. [WWHY19] describe how smooth a surface is by taking adjacent points on both sides of a point and build the norm of the sum of differences between those point and its neighbours. This norm is then divided through the norm of the point times the amount of neighbours. The resulting smoothness value is bounded by a lower threshold.

2.2.3 False Positive Filtering

In an urban scenario, besides the self-driving vehicle itself, other cars, pedestrians, buildings, vegetation and many more obstacles are inside or nearby the road too and therefore part of the LiDAR point cloud. Some of them are having the same spatial properties as well as features as curbs and will thus be falsely detected as curb points in the previous steps. For this reason, a false positive filtering is sometimes applied before the actual curb fitting to remove those misclassified points. This eliminates possible sources of error for the following step and can therefore enhance the curb detection results.

Zhao et al. [ZY12] are applying a short-term memory technique on predicted curbs to filter wrongly classified points. They incorporate curb points of the current frame with those of some previous frames. Their curb detection algorithm works on voxels and if a voxel has already been labelled as road, any vertical unsteadiness will not be predicted as curb anymore. Fernández et al. [FILS14] are using Conditional Random Fields [LMP01] with a 4-connected neighbourhood for the filtering process. They are using the previous result of the Conditional Random Field as prior and curbs found with spatial features as likelihood.

Another approach splits false positives into inside and outside the road and filters both individually using methods suitable for them. For example, Hata et al. [HOW14] are detecting curb points based on the distance between neighbouring laser rings, because when a laser hits an object this distance becomes smaller. Therefore, they are using a differential filter to keep points with a height variance by using a convolution operation. Then, they apply a distance filter to omit curb-like points besides the road, because curbs are the nearest points to the car. Finally, they use a regression filter to omit obstacles inside the road, like other cars or pedestrians. This filter estimates the shape of the curb and removes points not fitting in. Wang et al. [WWHY19] also perform their filtering with two steps to remove false positives inside and outside the road. They apply a distance filter to eliminate wrongly classified points outside the road and a RANSAC filter for inside. The RANSAC filter calculates the parameters for a quadratic polynomial model, which fits curb points as good as possible. All points further away from the model than a certain threshold are false positives and therefore removed from the curb point set.

2.2.4 Geometry Fitting and Tracking

Finally, a geometric model is generated from the calculated and filtered curb points. For this, several different approaches are used regarding representation and computation of the model, which are discussed below.

For the representation of the curb as geometric model either a parabolic model [ZY12] [ZWW+15] [WWHY19] or a B-spline [SZX+19] is used. Zhao et al. [ZY12] achieve the fitting with the combination of RANSAC followed by a linear least squares method. Zhang et al. [ZWW+15] are using a root mean square error method to compute the geometric model and Wang et al. [WWHY19] a least square method. To improve the curb further, it is tracked over multiple frames. Zhao et al. [ZY12] use therefore a particle filter [DdFG01] and Wang et al. [WWHY19] as well as Zhang et al. [ZWW+15] are using a Kalman filtering method [Kal60]. Zhao et al. [ZY12] state, that they are using a particle filter and not a Kalman filter to avoid a bias towards the cars movement.

Nevertheless, not all research uses this pipeline or this kind of features, for instance Han et al. [HKLS12] are computing line segments in the untransformed polar coordinates of the LiDAR sensor to detect the borders of the road. Kumar et al. [KMLM13] are using parametric active contour models to detect road boundaries in the converted 2D raster surfaces of elevation, reflectance and pulse width values of the LiDAR. A further method, introduced by Wang et al. [WLW+15] uses saliency maps and salient points to detect curbs and Rodríguez-Cuenca et al. [RCGCOA15] base their curb detection algorithm on 2D raster images of the point cloud, holding the height difference and the amount of points in each cell.

CHAPTER 3

Semantic Point Cloud Segmentation

When humans look at a point cloud, they can instantly identify objects like houses, trees or cars, but for a computer a point cloud is just a mass of unstructured points with no semantic information. An urban point cloud consists of buildings, vegetation, pedestrians, scanning artefacts and a lot more. However, for most tasks, points of just a certain category are actually needed. Let us assume the goal is to reconstruct buildings, it would be great if we already knew where in the point cloud the buildings are located, to apply further reconstruction algorithms specifically there. This would decrease the reconstruction time, since many points do not need to be processed. On the other hand, in some applications it would be an advantage to filter points of a specific class beforehand, like scanning artefacts or vegetation, to enhance the stability of algorithms.

For this and many other reasons, semantic point cloud segmentation is a well-liked research topic with a long history. Since CNNs are a great tool to extract and learn high level features and reach a very high accuracy in various applications as well, it seems natural to apply the knowledge and algorithms from image classification with CNNs to the 3D world of point clouds. Therefore, this thesis uses a 3D Convolutional Neural Network for the point cloud segmentation and this chapter is dedicated how this is achieved. Section 3.1 explains the process of data extraction of a point cloud and Section 3.2 describes the used dataset. Then, Section 3.3 gives an overview about neural network learning basics and Section 3.4 discusses the network architecture. Finally, Section 3.5 shortly depicts the segmentation process of a point cloud.

3.1 Data Extraction

The convolution and pooling operations in conventional machine learning environments are working on rasterized data and since those operations can be easily adapted from 2D to 3D, they are already supported. Nevertheless, point clouds do not have a regular structure and the point density can differ from very dense to extremely sparse. In order that the neural network can process the data, it needs to be rasterized in a way that the underlying structure of the point cloud is represented well. Furthermore, to train a well-generalized classifier a great amount of training data is needed, which requires a lot of memory - especially in the case of 3D data - but graphic cards do not have such an amount of memory available. Therefore, the data needs to be extracted such that it contains important information, but regions with too few points should be omitted.

For this reasons as well that LiDAR data can become extremely huge, an appropriate data structure is needed to visualize and process a point cloud in a proper way. Commonly used data structures for handling spatial data fast are spatial subdivision trees. Based on that, an octree is used as underlying data structure. As the name indicates, an octree node can have up to eight child nodes. In case of LiDAR data, the whole point cloud is the root node. The root node and recursively all other nodes are subdivided further, if they contain a feasible amount of points. Figure 3.1 shows a 2D visualization of an octree, where the root node is coloured black. Dense regions are subdivided more often and therefore located in lower levels of the tree, like the red nodes of Figure 3.1. Because of that, the size of those nodes is getting smaller with each subdivision. Contrary to that, sparse regions are not subdivided further and therefore located in bigger nodes on a higher level in the tree, as the blue nodes in Figure 3.1. Such data-structures give information about point density and location of relevant information in a point cloud, which is a good basis to extract training data.

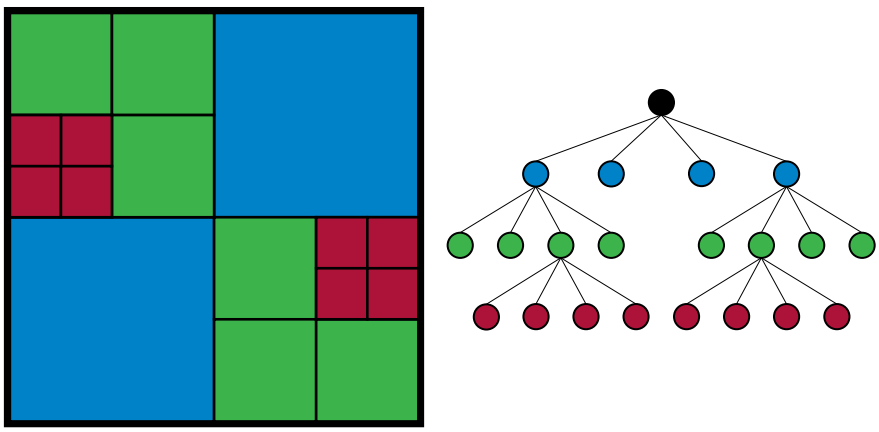


Figure 3.1: 2D visualization of an octree.

The visualization and processing of point clouds is done with the algorithms and data-structures from the Aardvark Platform [Aar]. The out-of-core octree implementation can handle point clouds very fast and is able to process humongous datasets. In this octree, the nodes are called cells and they have a property named "exponent" n, which indicates a fixed cell-size of 2^n meters. That makes it is easy to traverse a tree to the cells with the desired size in meters, particularly because all cells with the same exponent are on the same level of tree. Since point clouds are stored with a fixed measurement - a meter has always the same unit - it is possible to extract different datasets of various sizes with the same procedure. This is important, because the size of the nodes is crucial to represent the underlying structure of a point cloud. The cells must not be too big, because it would not be possible to represent fine structures after the rasterization, since they would be aggregated in the same grid cell. On the other hand, too small nodes would lose information about the greater context of an object and the classifier would not be able to distinguishing them well. In this work, the exponent $n = 0$, which is one meter, was used, since this exponent established as best cell size in prior attempts.

During data extraction, the octree is traversed to the level with the desired exponent and the cells of this level are used for training. Nonetheless, if a cell is already a leaf, but still contains a feasible amount of points, it is subdivided further. For the training the minimum amount of points in a cell are ten, for the classification, on the other hand, all cell with any points inside are used. The cells are then rasterized with a fixed grid size. Used grid sizes are 16 and 32, for example, if a cell with exponent $n = 0$, which is one meter, is rasterized with a grid size of 32, then each grid cell has a size of $\frac{1}{32}m = 0.03125m = 3.125cm$. Each grid cell is assigned a value, which represents the information about the points occupying the cell. In this thesis, three different values where explored:

- a **binary occupancy-value**, where the grid cell contains 1 if there are any points inside and 0 if not,
- the **absolute amount of all points** inside the cell
- and the **ratio of the number of points inside the grid cell to the total amount** that can be inside a node.

Besides the data itself, each training sample needs a ground truth to be learned by a neural network. Since, not all points in a node must have the same class-label; a majority vote on the class distribution is performed. By empirically exploring the class distribution for both used grid sizes, it transpired that, either just one class is contained in a node, or one class is highly represented in comparison to the others. That shows us, a majority vote is sufficient to portray the correct class of a node and by considering the rather small size of the nodes of just 1 meter, this seems reasonable.

However, not only the nodes themselves contain important information, their neighbourhood can be revealing as well. Therefore, for all six faces of the node, adjacent cells are

created and rasterized as well. Figure 3.2 shows the middle-node (left) and the according six side-cells (right) in matching colours. These six side-cells have the same exponent and therefore the same size, but the grid size used for rasterization could differ. In this thesis, grid sizes of 8 and 16 were used for the side-cells.

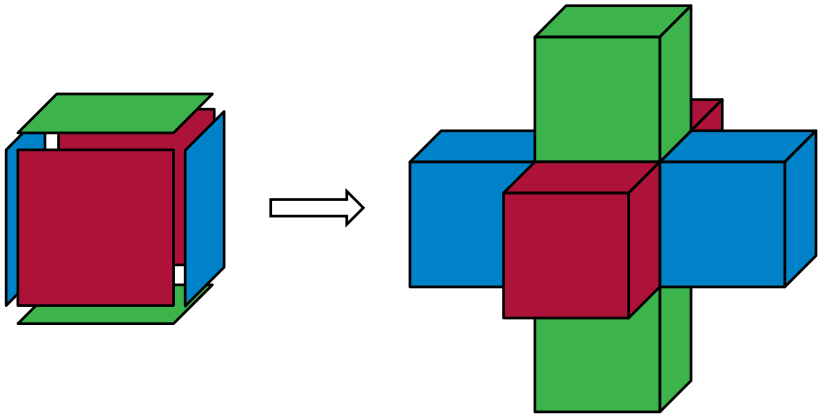


Figure 3.2: Left: middle-node with coloured faces. Right: adjacent side-cells coloured according to the faces

3.1.1 Data Augmentation

The generalization of a classifier is the ability to perform as good on unseen as on known data. That means that the classifier does not suffer from performance loss if it is applied to new data. A classifier may be afflicted with a bad generalization if it under- or overfits the training data. The training of a classifier approximates a function $y = f^*(x)$, which should depict the distribution of the training data well, but in case of under- or overfitting this function poorly represents the data. Figure 3.3 visualizes under- and overfitting in the case of polynomial curve fitting from Bishop et al. [Bis06], where the green curve shows the searched function, the blue dots are the training samples, the red curve is the fitted function and M denotes the order of the fitted polynomial. In case of $M = 1$ the fitted function does not capture the searched function at all (it is too simple) and this is called underfitting. $M = 9$ models a function fitting exactly all training samples, but gives a poor approximation of the searched function, this is called overfitting. Overfitting will cause a high training accuracy, whereby the validation or test accuracy is low.

To improve the generalization of a classifier, each class should have a nearly equal amount of training samples. Nevertheless, usually some classes are represented well, like vegetation or buildings, where others, like for instance natural terrain as meadows, are not so common in an urban point cloud dataset. To create an equal distributed training set, the amount of data samples with the least frequent class-label could be used as maximum size for all other classes too. However, for this method the dataset must have a certain size, because otherwise the training set would become too small

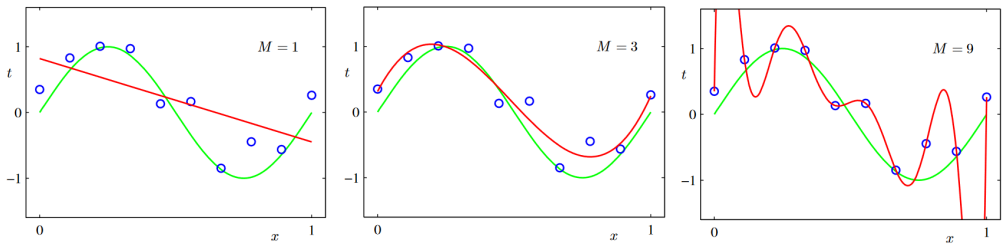


Figure 3.3: Example of under- and overfitting (red curves) in the case of polynomial curve fitting. M is the order of the fitted function. Left: underfitting. Middle: good representation. Right: overfitting. (Source: adapted from [Bis06])

to train a good classifier. Another method would be to create more data samples from underrepresented classes artificially. This is called data augmentation and a common procedure in machine learning. Data augmentation increases the variety of different data samples inside a class, which leads to a better generalization. It is not restricted to some classes and can be applied to the whole training set to increase its size and diversity. However, the training set used in this work is highly unbalanced; some classes have more samples than could be used for training, where others are underrepresented. Therefore, data augmentation is used in this work to create an equal distributed training set by creating various data samples of those underrepresented classes. To create data samples artificially, random samples from the training set are taken and then rotated, scaled, horizontally as well as vertically flipped, translated or a combination of some of them. Since the used augmentation-methods are depending on the use-case of the classifier, a rotation around the z-axis up to 360 degrees, an arbitrary translation in all directions about max. 75% of the node-size or a horizontal flip are applied in this work. These data sample augmentation methods are visualized in Figure 3.4.

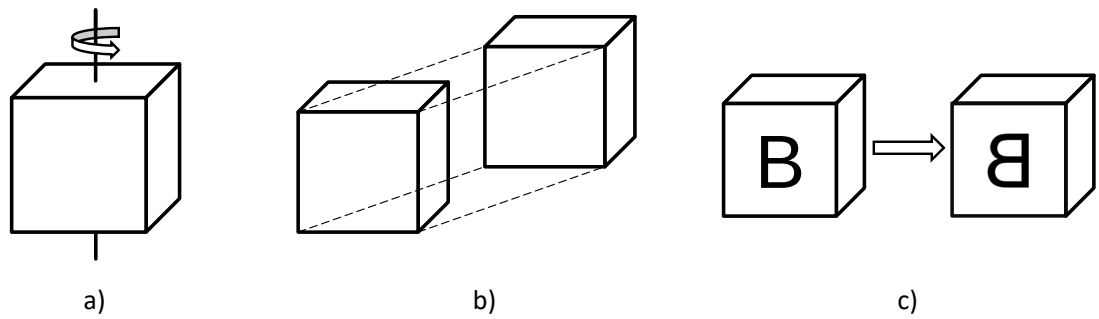


Figure 3.4: Data sample augmentation methods: a) rotation around z-axis, b) translation in all directions, c) horizontal flip.

3.2 Training Dataset

As mentioned above, the training set needs to be of a certain size and the data samples must have a correct ground-truth. For the purpose of this thesis, these would be point clouds of villages or cities with a certain amount of points, point density and an according labelling. The labelling should represent classes like the ground, vegetation, buildings, cars and suchlike.

Therefore, the Semantic3d.net [HSL+17] dataset was used for the training of the classifier. This is a dataset with around four billion points of urban and rural cities, which was captured with a laser scanner. This dataset comprises the following classes:

- man-made terrain
- natural terrain
- high vegetation
- low vegetation
- buildings
- hard scape
- cars
- scanning artefacts

The dataset contains 30 laser scans and is equally divided into a training and a test set, where only the training set is already labelled [HSL+17]. A point cloud of this training set, captured in a rural area, is shown in Figure 3.5.

The class affiliation of the points is visualized with colours, where each colour represents one class. Unlabelled points are white, “man-made terrain” is grey, “vegetation” is coloured with different shades of green, “buildings” are blue, “hard scape” is orange, “cars” are pink and “artefacts” are visualized with a brown-pink colour. Figure 3.6 shows the dataset from Figure 3.5 coloured according to the class membership. There it can also be seen that “man-made terrain” is asphalted terrain and “natural terrain” are meadows. The difference between high and low vegetation is that trees are referred as “high vegetation” and bushes or flowers as “low vegetation”. “Hard scape” are man-made structures like low walls, posts or wells.

Exporting the 15 Semantic3d training set point clouds with a cell-size of one meter results in over 200 000 training samples. Since, this is a dataset captured in the real world the class distribution is very unbalanced. “Buildings” is the most common class with nearly 75 000 data samples, followed by “high vegetation” with round 56 000, “natural terrain” and “man-made terrain” with 39 000 as well as 31 000 samples respectively.

Figure 3.5: A point cloud of the Semantic3d dataset, captured with a laser scanner in a rural area.

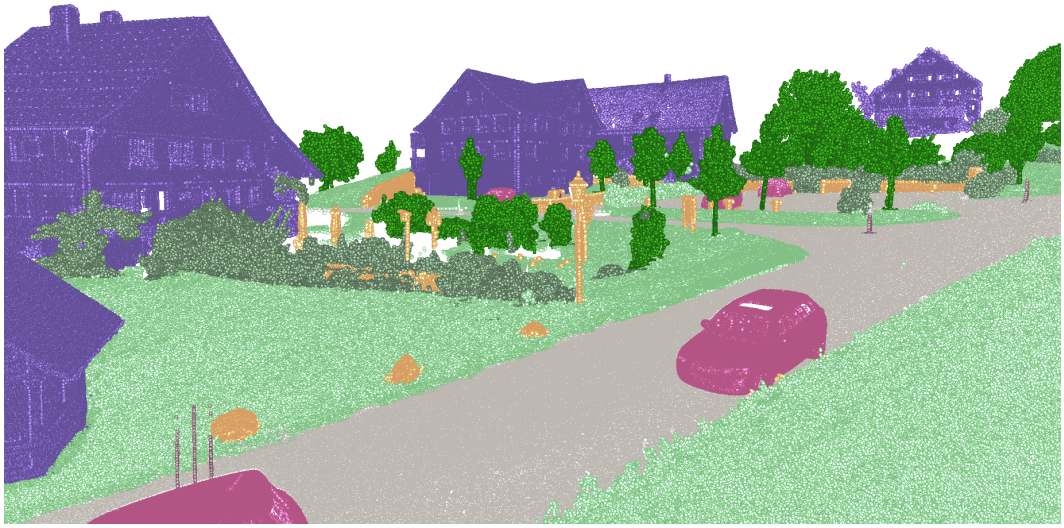


Figure 3.6: A point cloud of the Semantic3d dataset coloured according the class affiliation of the points.

Whereas “hard scape”, “low vegetation”, “scanning artefacts” and in particular “cars” are extremely underrepresented. However, for a good generalization all classes should be equally distributed, otherwise it could occur that the underrepresented labels are prone to misclassifications. For curb detection, especially cars need to be detected well, because they are usually placed in or nearby the street and are the most common cause for false positive curb classifications. Therefore, the classes with too few samples are artificially increased with data augmentation.

3.3 Network Learning Basics

As mentioned earlier, the training of a neural network approximates a function $y = f^*(x)$ by learning the mapping $y = f(x;\theta)$ with respect to θ [GBC16]. To achieve this, a loss function and an optimization algorithm are needed.

Loss Function

A loss function $L(\theta)$ measures the cost of misclassifications and is therefore also known as cost or objective function. It compares the network predictions with the ground truth based on the selected loss function. The probably best-known loss function is the mean squared error (MSE), where the squared distance between the predicted class probability $\tilde{y}$ and the ground truth y is calculated as follows:

$$L(\theta) = \frac{1}{N}\sum_{i=0}^{N}(y_i - \tilde{y_i})^2. \tag{3.1}$$

However, the probably most used loss function is the categorical or binary cross-entropy for multiple or two classes respectively. The entropy H was introduced by C. E. Shannon [Sha48] and is the measure for the average information content. In use as loss function, the cross-entropy is defined as follows:

$$L(\theta) = -\sum_{n=1}^{N}\sum_{c=1}^{K} y_i \; log \; \tilde{y_i}, \tag{3.2}$$

where N is the amount of data samples and K is the number of classes [Bis06]. Note, Equation 3.2 is adapted from [Bis06].

The goal of the training is to minimize the loss function, because a smaller loss means a smaller classification error. This is achieved by using optimization algorithms like gradient descent.

Gradient Descent

Gradient descent was introduced by Cauchy in 1847 [Cau47] and is an iterative optimization algorithm, which searches for the minimum of a function. Starting at a point x_k in a function $f(x)$, it looks where the greatest slope is and makes a step in the opposite direction. This results in a new point x_{k+1} and is one iteration of the algorithm. This process is repeated until a flat region is found. The greatest slope is the derivative $f'(x)$ or also known as gradient $\nabla f(x)$ of the function $f(x)$. The mathematical expression for one iteration is as follows:

$$x_{k+1} = x_k - \alpha f'(x_k), \tag{3.3}$$

where α is the step size, which is usually known as learning rate. When the gradient converges near zero $\nabla f(x) = 0$ the algorithm terminates, because a critical point is found.

Many improvements of the gradient descent algorithm were developed, the probably most known is adding a momentum term to improve the speed of convergence [Qia99]. Further methods are Adam or AdaMax, which are using adaptive learning rates [KB14].

Back-Propagation

The training of a network can be broken down into two parts: The *forward pass*, where an input x is propagated through the network to receive a prediction $\tilde{y}$. Then, the cost between the output $\tilde{y}$ and the ground truth y is calculated. In the *backward pass*, the cost is propagated backwards through the network to compute the gradients for the gradient descent optimization. The back-propagation algorithm [RHW86] achieves this in a computational inexpensive way [GBC16].

As Rumelhart et al. [RHW86] described in their work, the gradients are the partial derivatives of the total network error E with respect to the networks weights w and is denoted as $\frac{\partial E}{\partial w}$. Considering, $x_j = \sum_i y_i w_{ji}$, where x_j is the input to a neuron j, y are the outputs of the previous i neurons of j and w_{ji} are the according weights. The partial derivative of the error E with respect to the outputs y is computed first, resulting in $\frac{\partial E}{\partial y}$. Then, the chain rule is used to compute the partial derivative of the error E with respect to the input x: $\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial x}$. With this technique, the influence of an input x to an output y, and therefore to the error E, is known. Finally, $\frac{\partial E}{\partial w} = \frac{\partial E}{\partial x}\frac{\partial x}{\partial w}$ is determined by using the chain rule again. All this together results in:

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w}. \tag{3.4}$$

The network is updated by using $\frac{\partial E}{\partial w}$ to update the weights. As Rumelhart et al. [RHW86] stated further in their work, they aggregate all results until the update is performed and they modify the weights by using Equation 3.5, where t is increased by one after each iteration and $\alpha = [0, 1]$ stating the influence of the gradients to the weights.

$$\Delta w(t) = -\epsilon \frac{\partial E}{\partial w}(t) + \alpha \Delta w(t-1) \tag{3.5}$$

3.3.1 Regularization Methods

Regularization prevents a neural network from overfitting. The most used methods are L1- or L2 regularization, dropout layer and early stopping, which are shortly discussed below.

L^2 Parameter Regularization

The L^2 parameter regularization method adds a norm penalty $\Omega(\theta) = \frac{1}{2}\|w\|_2^2$ to the loss function $L(\theta)$. The regularized loss function $\tilde{L}(\theta)$ is therefore defined as shown in Equation 3.6, where $\alpha \in [0, \infty)$ controls the strength of the regularization. This method is also knows as weight decay, because it shifts the weights closer to zero [GBC16]. Note, Equation 3.6 is adapted from [GBC16] to the notation used in this thesis.

$$\tilde{L}(\theta) = \frac{\alpha}{2}\|w\|_2^2 + L(\theta) \tag{3.6}$$

L^1 Regularization

The L^1 regularization also adds a norm penalty term $\Omega(\theta) = \|w\|_1$ to the loss function $L(\theta)$. The regularized loss function $\tilde{L}(\theta)$, including the weight factor α, is defined with this method as shown in Equation 3.7. The L^1 regularization has the effect that parts of the weights become zero and that is why it was used as feature selection method [GBC16]. Note, 3.7 is adapted from [GBC16] to the notation used in this thesis too.

$$\tilde{L}(\theta) = \alpha\|w\|_1 + L(\theta) \tag{3.7}$$

Dropout

Another approach to reduce overfitting was introduced by Srivastava et al. [SHK+14] and is called dropout. As they describe in their work, this method randomly omits neurons as well as their connections from the network during training, which should hinder them from becoming too co-adapted. The probability, that a neuron is dropped is defined by $1 - p$. They stated, that a dropout value of $0.5 \leq p \leq 0.8$ is a good value for hidden neurons, whereas for input neurons p should be near 1. In the tests, they often used $p = 0.5$ and $p = 0.8$ for the hidden and input layer respectively.

Early Stopping

An overfitted classifier achieves a high accuracy on the training set, but generalizes poorly. This can already be seen during the training by observing the training and validation set loss. At some point the validation set loss will start to increase, while the training set loss still decreases - this is the beginning of overfitting. Early stopping traces those values and terminates the training, if the validation set loss does not decrease over a pre-defined amount of epochs. In course of this, the parameters of the network are saved at every improvement of the validation loss and after the training reloaded for classification.

3.4 Network Architecture

The architecture of a CNN can be broken down into two main blocks, which can be seen in Figure 3.7. The first block performs feature extraction as well as dimensionality reduction and is called *feature learning block*. The second one is a neural network and referred as *classification block*.

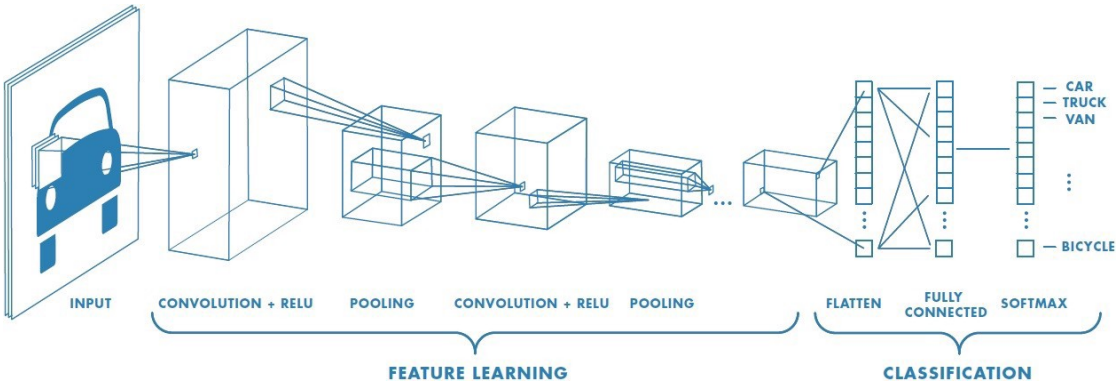


Figure 3.7: Exemplary structure of a CNN illustrating its two-stage structure with the *feature learning* and *classification blocks*. (Source: [Sah])

3.4.1 Feature Learning Block

The *feature learning block* is composed of multiple components: A 3D convolution operation (CONV), followed by a batch normalization (BN), a ReLU activation function and a 3D pooling operation (POOL). Batch normalization was introduced by Ioffe et al. [IS15] and is a regularization method, which conducts a normalization for each mini-batch during the training. They state that this method allows higher learning rates and one not have to be so careful about initialization. Their normalization transforms for each activation function x the mean to 0 and the variance to 1 by using Equation 3.8.

$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}} \tag{3.8}$$

First, two convolutional operations followed by batch normalization and a ReLU are stacked. Then, a pooling operation is applied. This sequence of operations is referred as a *feature learning component* in this work and can be repeated multiple times to form a deeper network. Figure 3.8 visualizes the structure of such a component.

The 3D convolution is performed with a kernel-size of 3×3 and an L2-regularization for the kernel and bias. The 3D max-pooling is executed with a pooling-size as well as stride of 2×2.

This block can be repeated multiple times, until a data sample is scaled down to the desired size. This is performed onto the middle-cell as well as all six neighbouring

side-cells individually. The results are then flattened by a global average pooling and concatenated afterwards. This final vector is fed into the classification block.

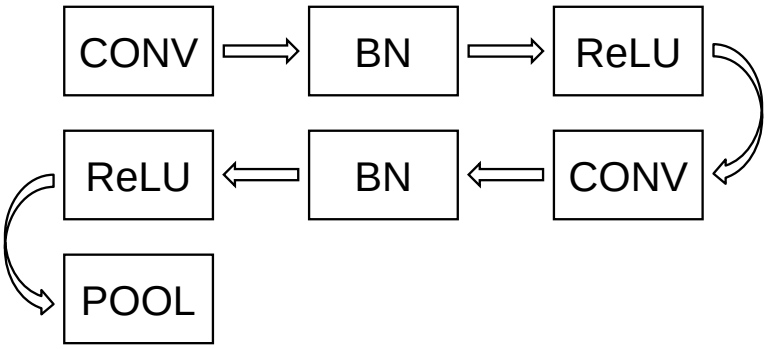


Figure 3.8: A *feature learning component* is composed of two convolution operations (CONV), batch-normalization (BN) and ReLU activation functions as well as a pooling operation (POOL).

3.4.2 Classification Block

The *classification block* is composed of multiple components too: It consists of a dropout-layer (DO), a fully-connected layer (FC), a ReLU activation function and then again, a dropout-layer as well as a fully-connected layer and a ReLU activation function. This structure is visualized in Figure 3.9 and is called *neural network component* it this thesis.

The fully-connected layers are regularizing the kernel and the bias with an L2-regularizer. The dropout-layers are first randomly excluding 20% and then 50% of the neurons.

After this block, the final output-layer is appended, which is a fully-connected layer with a softmax activation function and the number of classes as amount of neurons.

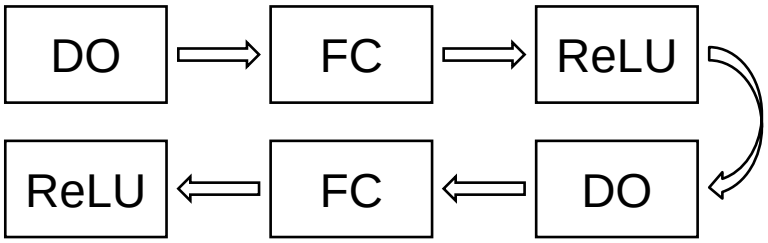


Figure 3.9: The *neural network component* of the network with fully-connected (FC), dropout (DO) layers and a ReLU activation function.

Figure 3.10 shows the combination all these part resulting in a neural network architecture with multiple convolution and pooling components, followed by fully-connected and dropout layers and the final layer with a softmax activation function.

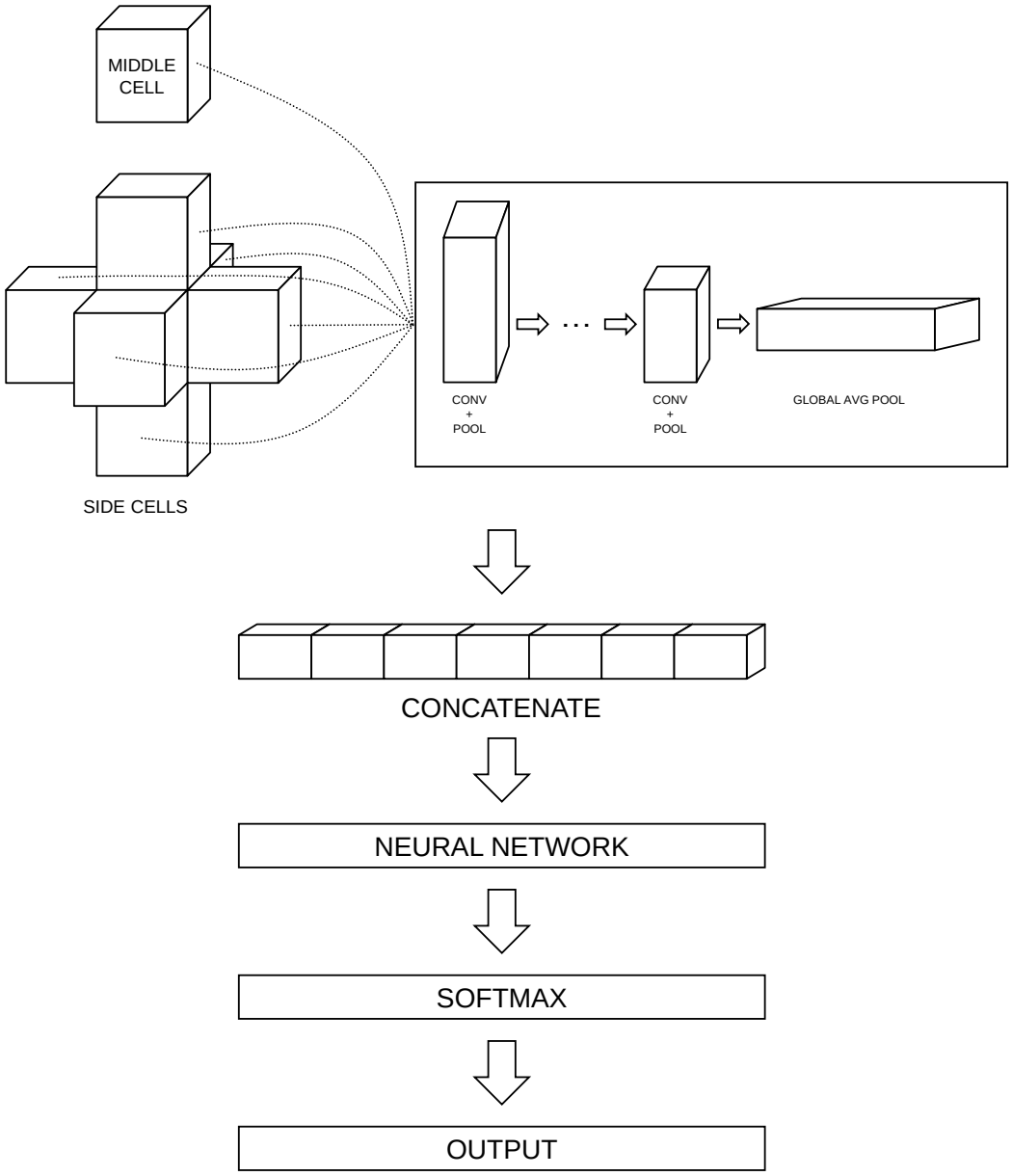


Figure 3.10: The entire network architecture.

3.5 Point Cloud Segmentation

To apply the trained network onto a point cloud, it needs to be exported the same way as the training data. However, in this case the minimum amount of points inside a cell is one, because all points of the dataset must be represented in the exported data to become classified. The data samples are then fed into the neural network and a class label is applied. Since the octree nodes are axis-aligned, the min- and max-position of the cells are exported too. By using that, the classified cell can be mapped back into the point cloud to its initial position. All points inside this cell are then labelled with the class prediction of the cell. This results in a pointwise segmentation of the point cloud.

CHAPTER 4

Curb Detection and Fitting

Due to the previous segmentation of the point cloud by using a 3D CNN a semantic information of the individual points is available. This information can be used for various applications and we are interested in extraction and geometry reconstruction of sidewalks. To achieve that, curb points are searched in the pre-filtered point cloud first and based on them, polygons of the sidewalk including the street are calculated. For searching curb points, only points labelled as "terrain" are used, since curbs are located on the ground. This chapter addresses the curb detection and geometry fitting process. The curb detection contains multiple steps: First, a point cloud is roughly filtered in Section 4.1, then simple geometric point cloud features are calculated and based on them points are classified as "curb" or "no curb" in Section 4.2. Some of those features need information about the street. Therefore, this Section covers data we use for that too. Since, every machine learning method lead to some false positive classifications, which would impair the final curb fitting, they are filtered in Section 4.3. Finally, the actual polygons are generated in Section 4.4.

4.1 Point Cloud Prefiltering

To identify curb points in a point cloud, features are calculated for each single point. Since LiDAR datasets are usually very large, the area in a point cloud for the curb detection must be narrowed down, to be able to detect curbs in a reasonable time. An obvious approach would be limiting the search area to the ground, because curbs are usually located there. Unfortunately, this would result in many false positive curb classifications, because houses, cars and other objects near or inside the street have similar geometric properties as curbs. However, with an understanding of the scene, it is possible to filter

these specific points beforehand and restrict the search area even further. Filtering these high-risk false positive points, will improve the curb detection result. Therefore, a point cloud is prefiltered according to the result of the point cloud segmentation and just points labelled as "terrain" are used.

4.2 Feature Extraction

Curbs have certain geometric characteristics; they are perpendicular to the street, follow the course of the road and have a particular elevation as well as shape. These properties are encoded into handcrafted features and computed for each single point of the (remaining) point cloud. To ensure a proper curb detection result, multiple different spatial features are used per point and if the point fulfils all descriptors, it is classified as curb. Three features are used in this thesis: difference in height along with the standard deviation of the height, curvature and angle of the point normal to the road.

4.2.1 Height Difference and Standard Deviation

The height of curbs usually do not exceed a certain limit and the elevation changes within a particular range. Therefore, features describing the height or the change of it, within a certain neighbourhood, are commonly used.

This feature was proposed by Wang et al. [WWHY19] and it calculates the absolute difference in height in a certain neighbourhood of the current contemplated point p and looks if this value lies within a certain range. Moreover, it also computes the standard deviation of the height values and restricts it with a lower bound. If both conditions are fulfilled, this feature states the point p as curb point.

The nearest neighbours n within half a meter of the point p are used and the height difference is calculated between the absolute minimum h_{min} and maximum h_{max} height value of the neighbourhood. Hence, the height difference is defined as:

$$T_{lower} \leq h_{max} - h_{min} \leq T_{upper} \tag{4.1}$$

where the range is bounded with a lower and upper threshold, T_{lower} and T_{upper} respectively. Figure 4.1 depicts this feature.

The standard deviation of the height is calculated as follows:

$$\begin{gathered} \mu = \frac{1}{N} \sum_{i=1}^{N} n_i \\ \sqrt{\frac{\sum_{i=1}^{N} (n_i - \mu)^2}{N}} \geq T_{std} \end{gathered} \tag{4.2}$$

where N is the amount of nearest neighbours and T_{std} is the upper threshold for the standard deviation. Note, Equations 4.1 and 4.2 are adapted from [WWHY19] to the

notation used in this work. The thresholds used for this feature are denoted in meters and have the following values:

$$T_{lower} = 0.03$$
$$T_{upper} = 0.3$$
$$T_{std} = 0.01$$

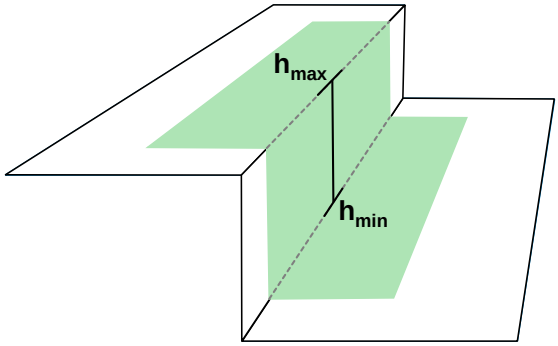


Figure 4.1: Height feature: The difference between h_{max} and h_{min} indicates weather the point is a curb or not.

4.2.2 Curvature

The shape of curbs is distinctive and therefore another good feature to detect sidewalks in point clouds. The curb shape can be transformed into a feature by analysing the point distribution within a certain neighbourhood.

This feature was introduced by Fernández et al. [FILS14] and it computes the curvature in the neighbourhood of a point p by creating a weighted covariance matrix with the nearest neighbours and applying an eigenvalue decomposition. The curvature is then computed by dividing the smallest eigenvalue by the sum of all eigenvalues. If this value lies within a certain range, the point is marked as "curb".

The nearest neighbours n within half a meter of the point p are used in this thesis to create the weighted covariance matrix, which is computed as follows:

$$\begin{aligned} \sigma &= \frac{1}{N}\sum_{i=1}^{N} n_i \\ \mu &= \frac{1}{N}\sum_{i=1}^{N} |\sigma - n_i| \\ w_i &= exp\left(-\frac{|p-n_i|^2}{\mu^2}\right) \\ COV &= \sum_{i=1}^{N} w_i (n_i - p)^T (n_i - p) \end{aligned} \tag{4.3}$$

where N is the amount of nearest neighbours n and w_i is the weight value for the neighbour n_i.

Then an eigenvalue decomposition is applied to this weighted covariance matrix, which results in three eigenvalues, where $\lambda_0 \leq \lambda_1 \leq \lambda_2$. Finally, the curvature value is defined as:

$$T_{lower} \leq \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \leq T_{upper} \tag{4.4}$$

where a lower and an upper threshold, T_{lower} and T_{upper} respectively, defies the range within a point is classified as "curb". Note, Equations 4.3 and 4.4 are adapted from [FILS14].

Visually spoken, the covariance matrix encodes the point distribution of the neighbourhood around the point of interest. The eigenvalue decomposition of this matrix computes a coordinate system representing the orientation of this distribution. The eigenvectors are the three orthogonal principal axis of this coordinate system and the eigenvalues are the length of these vectors, which encodes the underlying point variance along an axis. The curvature of the neighbourhood then can be described by putting these eigenvalues in relation to each other. Figure 4.2 illustrates the eigenvectors of a neighbourhood in their actual length in 2D. Comparing the result from a neighbourhood of a curb point (left) to the result from a neighbourhood of a road-point (right), it can be seen that the length of the eigenvectors and therefore the variance in the z-axis is way smaller for the flat neighbourhood.

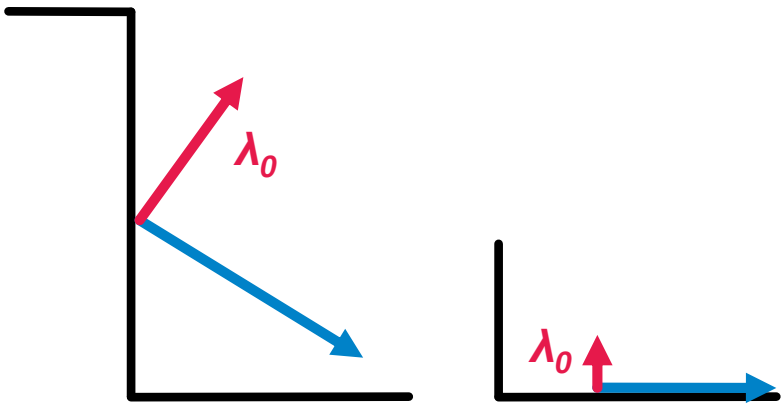


Figure 4.2: Result of an eigenvalue decomposition, once from a curb point (left) and once from a street-point (right) in 2D. The eigenvalue λ_0 from a curb point is larger than the eigenvalue λ_0 from a street-point.

The threshold values used for this feature are the following:

$$T_{lower} = 0.009$$

$$T_{upper} = 1$$

4.2.3 Perpendicularity

Curbs are always perpendicular to the street. Therefore, this characteristic is used to detect curbs too.

This feature was proposed by Zhao et al. [ZY12] and it verifies if the normal of the point is perpendicular to the road. In this work, the road is simplified with a continuous line-strip. Therefore, the direction of the nearest line-segment to the point is taken to represent the road. The dot product between the point normal n and the direction of the road d is calculated and bound by an upper threshold. Hence, the perpendicularity feature is defined as:

$$n \cdot d \leq 0.2 \tag{4.5}$$

This corresponds to a minimum bound of 78.46 degrees between the road direction and the point normal. Figure 4.3 illustrates this feature.

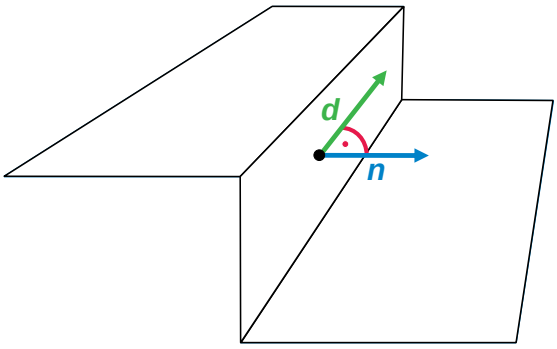


Figure 4.3: Perpendicularity feature, where the angle between the point normal n and the direction of the road d is computed.

Open Street Maps

Since we are dealing with LiDAR point clouds, although they have information about terrain points, we do not know the course of the road. However, there is a great interest in collecting geodata - including street data - with projects like OpenStreetMap [Opea]. The geodata is depicted with tags assigned to nodes, ways or relations. Streets are ways, which are formed by a list of nodes, represented with a key-value tag, like for example `highway=*`, where `highway` is the key. The value describes the type of the road and can have various classifications, like for example `primary`, `secondary`, `residential`, `pedestrian`, `footway` or `cycleway`. They can have further tags, describing properties such as name, speed limit, number of lanes or if the road is in a tunnel or a bridge [Opeb] and many more. Figure 4.4 shows an overview of Cologne in OpenStreetMap.

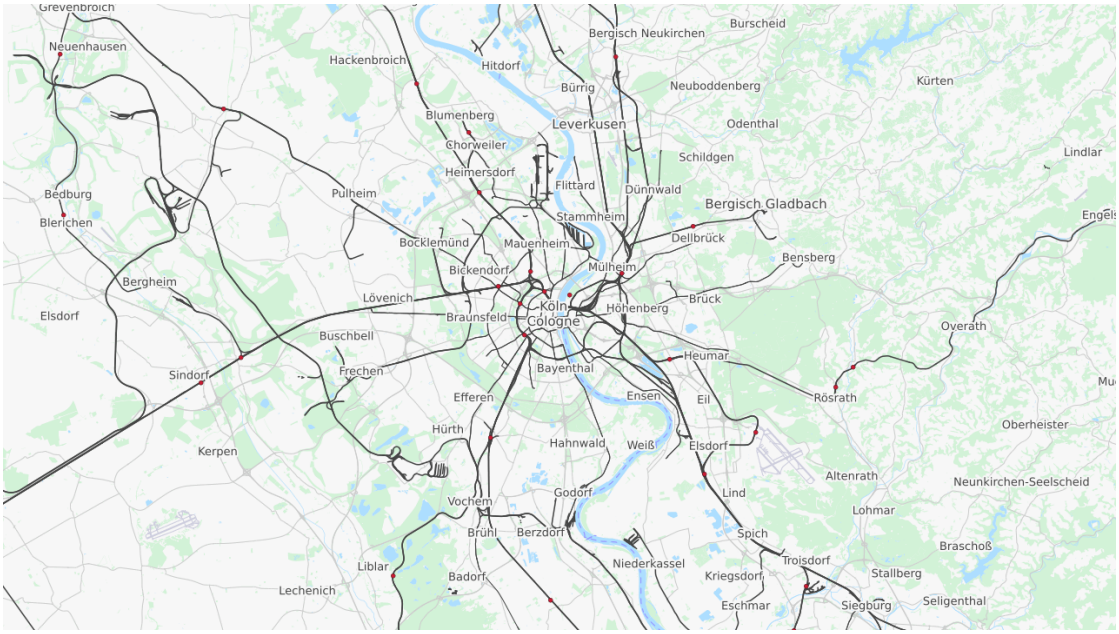


Figure 4.4: Overview of Cologne in OpenStreetMap. (Source: [Opea])

The OpenStreetMap geodata from an arbitrary area can be exported as an XML. Figure 4.5 shows a small area of Cologne including the Christian-Gau-Straße, which is a way with five nodes, highlighted in red. The according XML-representation of this street is as follows:

```
<way id="110666603">
   <nd ref="1263272353"/>
   <nd ref="1591118578"/>
   <nd ref="1591118581"/>
   <nd ref="1591118583"/>
   <nd ref="1591168365"/>
   <nd ref="250108119"/>
   <tag k="cycleway" v="opposite"/>
   <tag k="highway" v="residential"/>
   <tag k="lit" v="yes"/>
   <tag k="maxspeed" v="30"/>
   <tag k="name" v="Christian-Gau-Strasse"/>
   <tag k="oneway" v="yes"/>
   <tag k="oneway:bicycle" v="no"/>
   <tag k="sidewalk" v="both"/>
   <tag k="surface" v="sett"/>
</way>
```

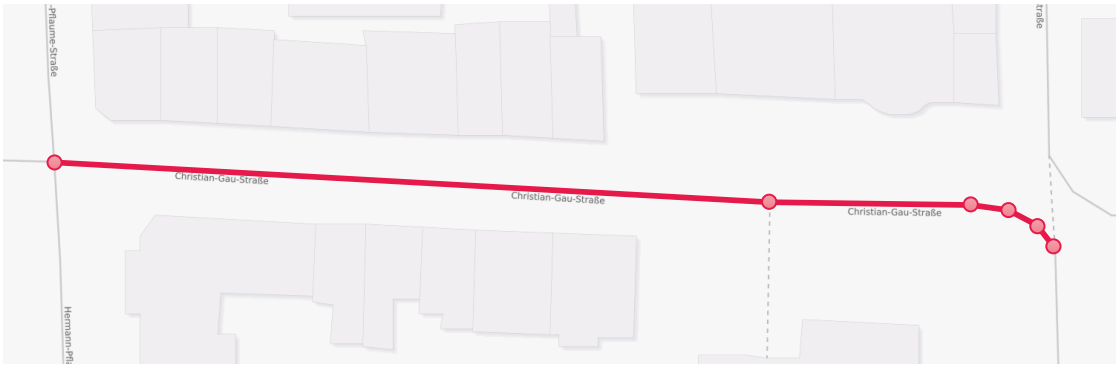


Figure 4.5: Christian-Gau-Strasse in Cologne with its OpenStreetMap tag coloured in red. (Source: adapted from [Opea])

This street is represented as a way and has an `id` as meta-information. Attributes of a way are the nodes `nd` with an id as reference `ref` to them and key-value pairs called `tag`, stating the name of the road as `Christian-Gau-Strasse`, the type as `residential`, a speed limit of 30 and among other tags also that this street is an one-way. A referenced node can look like this:

```
<node id="1591118578" lat="50.9360153" lon="6.8985179"/>
```

The node has also an `id` and GPS coordinates, `lat` and `lon` respectively, as meta-attributes. The latitude and longitude are represented in WGS84, World Geodetic System 1984, but LiDAR data have usually a local coordinate system, like in case of the data used in this thesis, the EPSG:31467, also known as 3-degree Gauss-Kruger zone 3. Therefore, the GPS coordinates of the OpenStreetMap nodes must be converted into the other coordinate system and this is achieved by using the library ProjNET4GeoAPI [Net].

To find streets in an exported OpenStreetMap XML, the file is searched for the tag `way` and this tag is then checked for the attribute `highway`. If the tag contains this attribute, the value of this key-value pair is inspected to filter ways like `footway`, `steps`, `cycleway`, `pedestrian` or `service`, because such ways are not of interest for us since they are not streets. Assuming that the tag contains a `highway`-attribute with a permitted value, the referenced nodes in this tag are searched in the XML. The GPS positions of them are extracted, converted into the 3-degree Gauss-Kruger zone 3 coordinate system and saved in a file named with the id of the way. When those roads are needed later on, the positions of the according file are loaded into the system, where they form the line-strip of the road. Figure 4.6 shows the LiDAR point cloud of the Christian-Gau-Straße with its imported OpenStreetMap road data in red.

Figure 4.6: Christian-Gau-Strasse in Cologne with its imported OpenStreetMap road data.

4.3 False Positive Filtering

Machine learning is not perfect, the classifier is not performing well on some data samples. Points misclassified as "curb", although they are not, are called false positives. Such false positives can origin from nearby cars, buildings and other objects with a similar geometric structure to curbs. Hence, this step tries to identify and filter such false positive classified curb points. The false positives are filtered by these three steps:

- First, the curb points are clustered with a density-based clustering algorithm.
- Then, a line-filter is applied on each cluster and
- finally, the remaining points per cluster are used to fit lines. These lines are checked for parallelism to the road.

Whereby the first two steps are inspired from the work of Wang et al. [WWHY19]. They are using a density-based clustering to split the points into the right and left side and a RANSAC-filter to detect false positives inside the road.

4.3.1 DBSCAN-Clustering

The points classified as curbs are clustered with the density-based clustering algorithm DBSCAN [EKS+96], which is able to detect cluster of an arbitrary shape and noise. The following paragraph describing this algorithm including Definition 4.6 is cited from [EKS+96]. The DBSCAN-algorithm differentiates between three types of points: core points, border points and noise. *Noise* are points p of the dataset D with no cluster C_i assigned and defined as $\{p \in D \mid \forall i : p \notin C_i\}$. *Core* and *border* points are both belonging to a cluster, whereby the latter ones are located at the border of the cluster. The main difference between them is the amount of neighbours found within the ϵ-neighbourhood, which is specified as $N_\epsilon(p) = \{q \in D \mid dist(p,q) \leq \epsilon\}$, where $dist(p,q)$ is a distance function between two points p and q. *Border* points usually have less ϵ-neighbours than *core* points. To form a cluster some constraint must be fulfilled, namely in a cluster every point p must be part of the ϵ-neighbourhood of q with a minimum amount of point $minPts$. This ***directly density-reachability*** is formal denoted as $p \in N_\epsilon(q)$ and $\mid N_\epsilon(q) \mid \geq minPts$, where the second one is the core point condition. The next constraint is the ***density-reachability*** of two points p and q, where p is reachable from q over the points $p_1, ..., p_n$, thus every step from p_i to p_{i+1} must be directly density-reachable. The last constraint is the ***density-connection*** between two points p and q, where both of them are density-reachable from another point g. Finally, they define a cluster C by using Definition 4.6.

$$\begin{gathered} \forall p, q : p \in C \text{ and } q \text{ is density-reachable from } p \text{ then } q \in C \\ \forall p, q \in C : p \text{ is density-connected to } q \end{gathered} \tag{4.6}$$

The curb points are clustered by using this algorithm, with $\epsilon = 0.2$ and $minPts = 50$, which indicates that an ϵ-neighbourhood of 20 centimetres must contain 50 points. This parameters are set relatively low, because LiDAR data are usually very dense and therefore curbs have more than enough points within these ranges. It is possible that the DBSCAN-algorithm splits a curb into multiple clusters. This could origin from "holes" in the point cloud caused by occlusions from other objects during the scanning process. For example, in an urban point cloud curbs are often occluded by parking cars besides the street, which cause a disconnected sidewalk. However, this algorithm is used to filter false positive classified curb points by identifying them as noise. Therefore, it does not matter if or in how many clusters a curb is split by this algorithm. The resulting clusters are used in the next steps of the false positive filtering, but discarded afterwards. Besides that, this step and the following ones are performed only with the xy-position of the points, since the line-filtering and the validation of the parallelism to the road are performed in 2D, which simplifies the problem, but is as accurate as in 3D.

4.3.2 Line-Filtering

The resulting clusters from the previous step are now filtered further with a line-filter. This filter uses the RANSAC algorithm [FB81], where a random set S of size N is selected from the dataset D and a model M is fitted with the samples of S. The size N depends on the minimum amount of points the model needs. Then, the model is verified by calculating an error function between all points of D to M and all points within a certain tolerance range are forming the consensus set S_c. If the size of S_c is greater than some threshold, a new model is computed with the consensus set. If not, this process is repeated until another random subset S fulfils the condition. In case no random subset S is greater than the threshold, the algorithm terminates after a defined number of iterations, there either the best model so far is returned or the algorithm fails.

The RANSAC implementation of this thesis uses a 2D line as model, which needs two points to be formulated. The problem is transformed from the 3D to the 2D space by omitting the z-axis and just using x- and y-coordinates of the points. Visually spoken, it is like looking at the problem from a bird's eye perspective. This dimensionality reduction simplifies the problem, but does not influence the overall result in a negative way. As error function, the distance from the points in D to the line is used and the points in the consensus set are called *inlier*. If the number of inlier is greater than the best result so far, a new model based on the consensus set and a score are computed. The score gives a measurement about the accuracy of the model and is based on the *normalizes squared error of inlier* from Choi et al. [CKY97] denoted as seen in Equation 4.7, where $\mathcal{I}$ is the set of inlier, Err is an error function between a point p_i and the estimated model M or true model M^*.

$$NSE(M) = \frac{\sum_{p_i \in \mathcal{I}} \mathrm{Err}(p_i; M)^2}{\sum_{p_i \in \mathcal{I}} \mathrm{Err}(p_i; M^*)^2} \tag{4.7}$$

The score is then calculated as in Equation 4.8, which originates from the RANSAC implementation of the scikit-learn python library [PVG+11] [BLB+13].

$$\begin{gathered} NSE(M) = \frac{\sum_{p_i \in \mathcal{I}} (p_i - M(p_i))^2}{\sum_{p_i \in \mathcal{I}} (p_i - \mu)^2} \\ Score(M) = 1 - NSE(M) \end{gathered} \tag{4.8}$$

The RANSAC-filtering process performs the following steps for each cluster. Considering that the course of the road is approximated with a line-strip, the cluster is split according to the nearest line-segments of the road. For the reason that, if the road bends, the points of the curve will be part of the same cluster, since the conditions of the DBSCAN algorithm are fulfilled, but a line cannot approximate a curve well. The RANSAC algorithm is applied to each sub-cluster with a distance threshold of 15 centimetres. If the resulting model is accurate enough, which is achieved with a $Score(M) \geq 0.8$, the inlier and the model are saved. This again filters points not complying the distance threshold of the RANSAC algorithm.

4.3.3 Parallelism to the Road

All fitted lines of the sub-clusters are combined into a single continuous line-strip by interpolating the corresponding endpoints of them. This new cluster line-strip is then checked for parallelism to the road by using the dot product between them, which is 1 if two vectors are parallel. If any dot product for a line-segment in the cluster is ≥ 0.99 the remaining inlier points of this cluster are marked as true positives and not filtered.

All remaining possible curb points are labelled as "curb". In the next and last step, the geometry is fitted through these points.

4.4 Curb Fitting

The curb fitting is done in multiple steps, where the process orients itself on the course of the road line-strip. First, each curb point is assigned to its nearest road line-segment and then allocated to the left or right side of it. Through both sides, a model of a mathematical function is fitted with the RANSAC algorithm, where it is represented by a Lagrange polynomial. The standard case uses a polynomial of second degree, but for streets with a stronger curve, a polynomial of a higher degree is applied. This step uses polynomial of a higher degree and not a line to represent the course of the road, because we need a good representation of the curb in this step, since we are now computing the final geometry. The error function is the distance from the point to the model and the distance threshold is 0.1 meter. Furthermore, the function only uses the x- and y-coordinates, because this step calculates the pathway of the curb and not the location of the height. However, a function of higher degree is not so handy to work with afterwards. Therefore, it is subdivided into a continuous line-strip with a fixed step-size of two meters in the standard case. This depicts the actual road even with curves well.

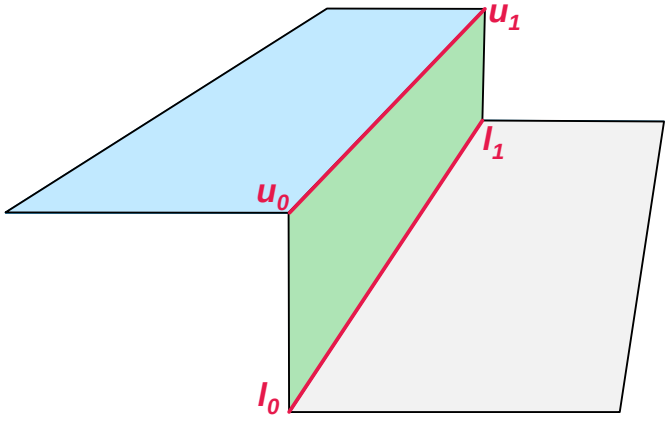


Figure 4.7: A curb segment with the fitted lower and upper edges. u_0 and u_1 denote the endpoints of the upper edge and l_0 and l_1 denote the endpoints of the lower edge.

To calculate the height of the curb, the best-fit lines themselves and their inlier points are used. For each curb line-segment, the nearest inlier are taken to find the minimum and maximum height value in the area of the ends of the line. The point cloud could have holes in the area of the curb, caused by parking cars for example, but we want to close those holes and create a continuous curb geometry. Therefore, we start searching within the neighbourhood of 0.1 meter and as long as we do not find any curb points, this area is enlarged by 0.1 meter each iteration. With the line-segments representing the course of the curb and the detected minimum as well as maximum height values of points constituting the ends of the curb, the upper and lower edges of the curb are constructed, which can be seen in Figure 4.7.

The final geometry model includes polygons for the curbs, sidewalk and street. Figures 4.7 to 4.10 are showing the curb in green, the sidewalk in blue and the road in grey. For each side of the road, the polygon line-strip with the upper and lower edges, u as well as l respectively, are processed. The curb polygons are constructed by the edge points of the according lines. The endpoints of the lower edge as well as the endpoints of the upper edge form a curb polygon C_i with the vertices l_0, l_1, u_0 and u_1. Figure 4.8 visualizes a section of a curb with three final curb polygons C_0, C_1 and C_2.

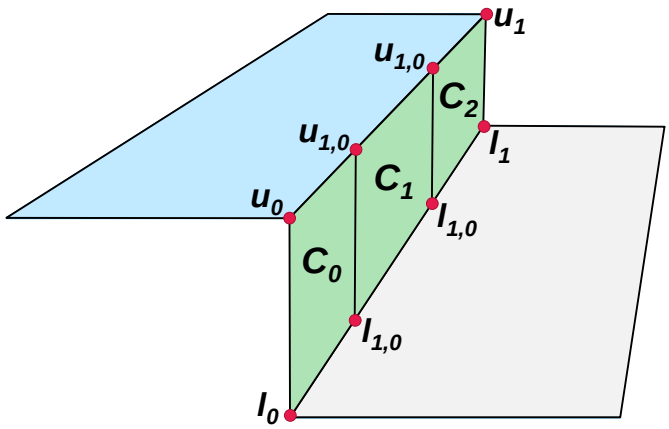


Figure 4.8: A section of a curb with the final polygons C_0, C_1 and C_2 of three curb segments.

For each sidewalk-polygon, the terrain points in a neighbourhood of two meters are split into pavement and street points by using the middle height of the lower and upper edges. The points above this height value are sidewalk points and the points below are street points. A plane is fitted through these points using the RANSAC algorithm with a distance threshold of 0.005 meter. The sidewalk-polygons are constructed by using the endpoints of the upper edge, and positions two meters perpendicular to them, resulting in a polygon SW_i with vertices u_0, u_1, s_0 and s_1. To create a continuous sidewalk geometry, the perpendicular points are then interpolated accordingly. The resulting sidewalk polygons SW_0, SW_1 and SW_2 of this step are visualized in Figure 4.9.

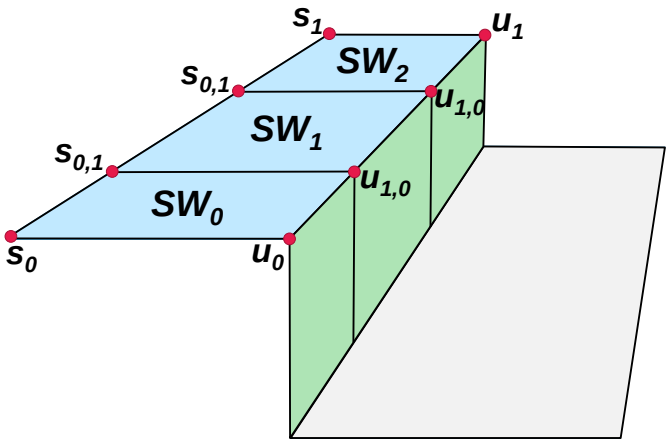


Figure 4.9: A section of a curb with the final polygons of three curb segments with the according sidewalk-polygons SW_0, SW_1 and SW_2.

Finally, the road-polygon is created by using the end-points of the lower edge-lines of the curbs. Figure 4.10 shows this polygon with vertices $r_0, ..., r_1$ in pink created from the according endpoints of the line-segments l_0 and l_1 respectively.

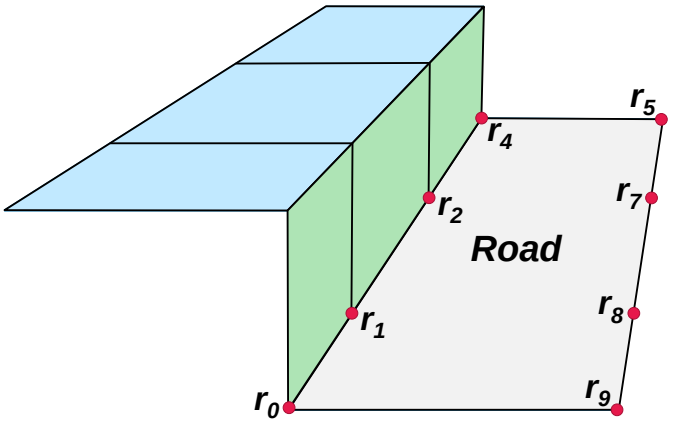


Figure 4.10: The final polygons of a curb section from three curb as well as sidewalk segments and the road polygon, which is constituted by the vertices r_i.

CHAPTER 5

Neural Network Training

Training a Convolutional Neural Networks needs a lot of fine-tuning. Therefore, a lot of work was put into finding the best datasets sizes, network architecture and hyperparameters. The results of the network training are shown in this chapter, whereby Section 5.1 gives a short overview about the assembly of the used dataset, Section 5.2 describes the used hyperparameters and why they were chosen and finally, Sections 5.3, 5.4 as well as 5.5 discuss the training results of three different networks respectively.

The following system setup was used for the training and classification: Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz, 64 GB RAM and GeForce RTX 2080 Ti. The network was implemented with Keras [C+15] and Tensorflow [AAB+15] in Python.

5.1 Datasets

The Semantic3d training set is used for the network training. It is split into an actual training, validation and test set, where the training set is used to learn the data distribution and the validation set to tune the network's hyperparameters. The test set is used to verify the performance of the resulting classifier on unseen data. Hence, for this chapter we refer to the Semantic3d training set as dataset.

The dataset is split into a training, a validation and a test set. The distribution of the classes should be equal in size. If there are not enough data samples of a class available, data augmentation is used to create more samples artificially. In the training set, the amount of data samples per class is 10.000, in the validation set 3.000 and in the test set 630. This results in a total set size of 80.000 for the training set and 24.000 as well as 5.040 for the validation and test set respectively. The composition of these sets are

visualized in Figure 5.1 with the training set including data augmentations in blue, the validation set as well as its data augmentations in green and the test set in red.

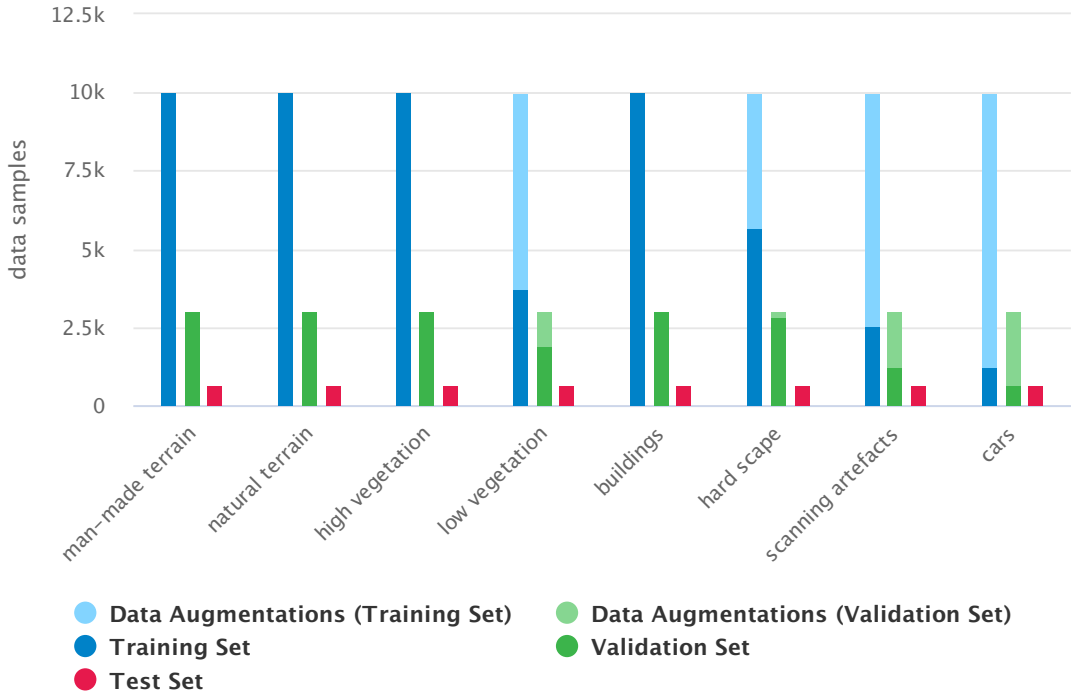


Figure 5.1: Class Distribution of the training, validation and test set including data augmentations.

5.2 Hyperparameter-Tuning

A convolutional neural network includes many hyperparameter, which need to be tuned individually. This begins with the actual architecture of the network, the amount of neurons in each layer, grid size, up to dropout or learning rates. To get the very best out of a network many different combinations were evaluated. Furthermore, in order to be statistically representative, the network was trained and evaluated ten times with each setting. The results are an average value of them. Note, that the precise results of these hyperparameter-tuning tests are listed in Appendix A.

However, some parameters were the same for all different experiments: To improve the generalization of the network, early stopping was used. It stops the training process prematurely if the validation loss has not improved over the last seven epochs. Categorical cross-entropy is used as loss function and every time it improves, the weights of the network are saved. Therefore, once the training is finished, the weights of the best epoch

are used for the classification. As optimizer AdaMax with its default parameters, 0.002 for the learning rate and $\beta_1 = 0.9$ as well as $\beta_2 = 0.999$ were used.

The most comprehensive tests were made with a grid size of 16 for the middle node and 8 for the side cubes. For the simple reason that this is the smallest dataset, according to memory consumption, and essential things, like the occupancy values, can be quickly tested and probably discarded in further experiments.

A feature learning component consists of convolutional and pooling operations and has the following structure: CONV - BN - ReLU - CONV - BN - ReLU - POOL. First, the impact of the amount of such components on the middle nodes as well as side cubes was explored. The pooling operation has a kernel size of 2×2; this scales the input down by half. That means for a input size of 16, one pooling operation results in a grid size of 8, two pooling operations result in a grid size of 4 and so on.

To explore the impact of the amount of feature learning components on the middle nodes, the network was trained just on them. One component achieved a slightly better performance, than two or three stacked components. In particular, it was about 0.86% better than two and 1.26% better than three were. This is also the case for a different amount of feature learning components in combination with side cubes. Two stacked components perform slightly worse than just one. The reason for that is probably, that more components down scale the input too much and loose information in this process. However, the increase in accuracy between a network trained just on middle nodes and a network also including the neighbourhood information is 6.47%. In further tests, the amount of feature learning components was set to one for the middle cube and one for the side cubes, since this combination works best.

Three different occupancy values were evaluated:

- A *binary* occupancy value, where the grid cell either contains 1 or 0, depending if there are any points inside the cell or not,
- the *absolute amount of points* inside a cell and
- the *ratio* of the number of points inside the grid cell to the total amount of points in the node.

Note that the previous tests were performed with binary occupancy values, as it has been determined very earlier that this mode works best. The total amount of points as occupancy value performs about 3.75% worse on the test set and the fractional occupancy value does not work at all. The network is not able to minimize the loss and therefore cannot learn anything; the accuracy is around 33%. The reason behind this is that the average amount of points inside a cell is 1.76 and 15.92 for middle and side cells respectively. The average total amount of points is 7221.31 for middle and 8149.96 and side cells. Building the fraction with the according values results in nearly zero, which is probably too small to learn a mapping from data samples to class membership.

Furthermore, additional features describing properties of the middle cell were concatenated to the resulting feature vector of the feature learning phase and evaluated as well. Those features involve the absolute height of a cell, average height of all points inside a cell, average point normal and average point colour. The additional information about the height of the box center increases the accuracy about 1.49% and the average height of points inside a box results in a better performance about 1.68%. The average normal of points in a box does not increase the result much, just about 0.07%. However, the average point colour has a much higher impact on the accuracy of the network; it is increased by 2.37% by adding this feature.

Further hyperparameters are the dropout value and amount of neurons in the classification part of the network. A dropout value of 0.2% for the input and 0.5% for the hidden layer with 2048 and 4096 neurons works best. Furthermore, 128 neurons in the convolutional block works best for the middle cube and 64 neurons for the side cubes. L2-Regularization with a value of 0.0001 is used. This value does not reduce overfitting of the network much, but on the other hand, it also does not decrease the accuracy too much (about 1%, instead of 7.5% with a regularization value of 0.01).

5.3 Network Using Small Grid Size

This network architecture has a grid size of 16 for the middle cube as well as 8 for the side cubes and consists of the following layers: CONV - BN - ReLU - CONV - BN - ReLU - POOL - DO - FC - DO - FC. The convolutional layers have 128 neurons for the middle and 64 for the side cubes. The first fully connected layer has 2048 neurons and the second one has 4096. The first dropout layer has a dropout rate of 0.2 and the second one of 0.5. Furthermore, the binary mode and additional feature of average point colour were used. The architecture of this network is visualized in Appendix B.1.

The network performance was validated with the `metric.py`-script from the Semantic3d dataset [Sem] on the test set. It calculates the Intersection over Union (IoU), Overall Accuracy (OA) and also a confusion matrix. This network achieves an overall IoU of 72.12% and an OA of 80.87%, which is shown together with the IoU of the eight single classes in Table 5.1. There it can be seen that the classes “man-made terrain” and “natural terrain” are classified quite well, but the other classes perform not so good, especially “low vegetation” and “hard scape”. The confusion matrix is shown in Table 5.2, where ***MMT*** is “man-made terrain”, ***NT*** is “natural terrain”, ***HV*** is “high vegetation”, ***LV*** is “low vegetation”, ***B*** are “buildings”, ***HS*** is “hard scape”, ***A*** are “artefacts” and ***C*** are “cars”. By taking a closer look on this confusion matrix, it can be seen, that a lot of misclassification occur between “high vegatation” and “low vegetation”, “buildings” and “hard scape” as well as “cars” and “hard scape”. Even, if the terrain classes perform quite well, most misclassification are made between those two classes.

classes	***8 classes***	***6 classes***
	IoU in %	
man-made terrain	83.29	90.6
natural terrain	84.55	
high vegetation	67.10	82.04
low vegetation	54.14	
buildings	65.34	66.02
hard scape	51.52	53.4
artefacts	74.93	74.55
cars	68.24	66.09
average over all classes	68.64	72.12
	OA in %	
overall accuracy	80.87	85.69

Table 5.1: IoU of the single classes, average IoU and OA of the test set classified with the network using small grid size. Once with the original eight classes and once with the combined six classes.

	MMT	***NT***	***HV***	***LV***	***B***	***HS***	***A***	***C***
MMT	91.75	4.60	0.00	0.32	0.95	1.59	0.16	0.63
NT	3.17	92.06	0.00	3.49	0.63	0.48	0.00	0.16
HV	0.00	0.16	81.27	14.76	1.27	1.11	0.95	0.48
LV	0.79	2.70	15.40	70.63	2.70	4.44	0.95	2.38
B	1.59	0.16	0.95	1.90	78.41	9.84	3.65	3.49
HS	1.59	0.79	1.59	5.56	10.16	67.30	3.17	9.84
A	0.79	0.16	1.75	1.90	2.54	5.87	83.02	3.97
C	2.22	0.32	1.43	2.54	1.75	7.30	1.90	82.54

Table 5.2: Confusion matrix of the test set classified with the network using small grid size. (Note: values are in %.)

Since, for the purpose of this thesis, the distinction between the classes “man-made terrain” and “natural terrain” as well as “high vegetation” and “low vegetation” is not relevant, they are combined into a single class, called “vegetation” and “terrain”. Another network was trained on these six classes, which performs better than the network using eight classes. For this network the IoU is 72.12% and the OA is 85.69%, which are 3.48% as well as 4.82% better as with eight classes. The IoU for the single classes are also visualized in Table 5.1. By looking at this table, it can be seen that the single class “terrain” performs 6.68% better than the average of the classes “man-made terrain” and “natural terrain” and the single class “vegetation” performs 21.42% better than the average of the classes “high vegetation” and “low vegetation” for the IoU. The other classes perform a little bit better or worse, but in a range that can be accepted. By looking into the confusion matrix of this network using six classes in Table 5.3, it can

be seen that the single class “terrain” performs 3.72% better than the average of the classes “man-made terrain” and “natural terrain”. The single class “vegetation” performs 16.83% better than the average of the classes “high vegetation” and “low vegetation” too. The accuracy for the other classes is up to 3% better or up to 4% worse, but also this can be accepted due to the nearly 17% performance increase of the “vegetation” class. Furthermore, it should be mentioned, that for this network using six classes the data samples of the terrain and vegetation classes were combined, which results in an amount of data samples twice as big as for the other classes. Moreover, also a network using six classes with an equal class distribution as well as a network using six classes with an unequal class distribution, but with class weights were trained. Nevertheless, those networks performed slightly worse. Therefore, the further networks using six classes in this thesis are trained with an unequal class distribution without class weights.

	terrain	***vegetation***	***buildings***	***hard scape***	***artefacts***	***cars***
terrain	95.63	2.22	0.56	0.87	0.08	0.63
vegetation	1.43	92.78	0.95	2.14	1.43	1.27
buildings	1.90	3.97	75.87	10.32	4.13	3.81
hard scape	3.17	8.25	7.46	68.57	5.40	7.14
artefacts	0.48	3.17	2.06	4.60	86.03	3.65
cars	2.70	6.35	2.38	7.46	2.86	78.25

Table 5.3: Confusion matrix of the test set classified with the network using small grid size and six classes.

5.4 Network Using Middle Grid Size

This network architecture has a grid size of 16 for the middle as well as side cubes, apart from that it is the same as for the network with grid sizes 16 and 8 respectively. Except for the amount of neurons in the dense layer as well as the batch size, they were reduced due to memory limitations. They are decreased to 1024 for the first and 2048 for the second dense layer as well as 64 for the batch size. Moreover, the additional feature of average point colour was used in this network too. The architecture of this network is visualized in Appendix B.2.

For the network using eight classes, the overall accuracy is 82.14%, which is 1.27% better than with the network using small grid size and the average IoU is 1.63% better with the increased grid size for the side cubes. The IoU values for each class are shown in Table 5.4, they are all better with this network except for the class “natural terrain”; this one performs 1.27% worse. The confusion matrix for the network using eight classes is displayed in Table 5.5, there it can be seen that the classes “man-made terrain”, “high vegetation”, “hard scape”, “artefacts” and “cars” perform better than with the network using small grid size. Especially the two classes “high vegetation” and “artefacts”

increased about 4.44% and 5.55% respectively. On the other hand, the classes “natural terrain”, “low vegetation” and “buildings” performed slightly worse, with a maximum of 2.69% for the class “natural terrain”. However, the most misclassifications occur between the two terrain and the two vegetation classes as well as between the classes “buildings” and “hard scape”.

classes	***8 classes***	***6 classes***
	IoU in %	
man-made terrain	83.67	90.00
natural terrain	83.28	
high vegetation	69.95	83.76
low vegetation	56.08	
buildings	66.90	67.57
hard scape	53.42	53.91
artefacts	78.59	76.30
cars	70.25	72.06
average over all classes	70.27	73.93
	OA in %	
overall accuracy	82.14	86.69

Table 5.4: IoU of the single classes, average IoU and OA of the test set classified with the network using middle grid size. Once with the original eight classes and once with the combined six classes.

	MMT	***NT***	***HV***	***LV***	***B***	***HS***	***A***	***C***
MMT	92.70	3.65	0.00	0.00	0.32	1.90	0.32	1.11
NT	4.76	89.37	0.32	3.17	0.63	0.79	0.16	0.79
HV	0.00	0.00	85.71	11.43	0.32	0.79	0.48	1.27
LV	0.63	2.22	18.10	70.32	1.43	2.70	1.43	3.17
B	1.90	0.48	1.59	1.11	77.30	10.63	3.02	3.97
HS	1.27	0.48	0.95	5.87	9.52	68.10	5.08	8.73
A	0.95	0.16	0.63	1.75	2.06	3.81	88.57	2.06
C	1.27	0.32	0.95	2.06	1.27	6.83	2.22	85.08

Table 5.5: Confusion matrix of the test set classified with the network using middle grid size. (Note: values are in %.)

This network was also trained with six classes and performs 4.55% better in the OA and 3.66% better in the average IoU. The IoU for single classes are also increased: About 6.53 % for the class “terrain” compared to the average of the classes “man-made terrain” and “natural terrain” and about 20.75% for the single class “vegetation” compared to the average IoU values of the classes “high vegetation” and “low vegetation”. Moreover, “buildings”, “hard scape” and “cars” have a slightly better IoU, up to 1.81% than with the network using eight classes. The class “artefacts” performs 2.29% worse compared

to the network using eight classes. However, the increase in IoU for all other classes outweigh this rather small decrease in the class "artefacts". The precise values of the OA and IoU are shown in Table 5.4.

The confusion matrix of the network using six classes is displayed in Table 5.6, there it can be seen that this network performs 4.68% better with the combined class "terrain" and 14.52% better with the combined class "vegetation" compared to the network using eight classes. The other classes perform either slightly better or worse, to be precise the classes "buildings" as well as "artefacts" perform both 2.22% worse and the classes "hard scape" and "cars" have an increased accuracy of about 1.9% and 0.48% respectively. The most misclassifications of the network using six classes occur between the classes "buildings" and "hard scape", which is probably caused of the similar structure of buildings and low walls. Also between the classes "hard scape" and "cars" is the misclassification rate high, precisely about 9%.

Compared to the accuracies of the network using small grid size and six classes the network using middle grid size has an increase in the OA about 1% and in the average IoU about 1.81%. By comparing the confusion matrices of those two networks, it can be seen that the accuracy is also increased for most classes: The classes "terrain" and "artefacts" just perform slightly better than with the network using middle grid size, about 0.08% and 0.32% respectively. The class "hard scape" performs 1.43% better too and the class "cars" has an increase about 7.31% in correct classifications. The remaining two classes, "vegetation" and "buildings", perform marginal worse than with the network using small grid size, namely about 0.24% and 0.79% respectively. However, this decrease in the accuracy is negligible in comparison to the increase for all other classes and especially the increase about 7% for the class "cars".

	terrain	***vegetation***	***buildings***	***hard scape***	***artefacts***	***cars***
terrain	95.71	1.90	0.32	1.19	0.24	0.63
vegetation	1.67	92.54	0.95	1.98	1.19	1.67
buildings	2.86	4.13	75.08	11.90	3.49	2.54
hard scape	2.86	6.67	6.19	70.00	5.24	9.05
artefacts	0.79	3.02	1.43	5.87	86.35	2.54
cars	2.86	3.33	0.95	5.71	1.59	85.56

Table 5.6: Confusion matrix of the test set classified with the network using middle grid size with six classes.

5.5 Network Using Big Grid Size

Here, the grid size for the middle cube is 32 and for the side cubes 16. This network is composed of two stacked feature learning components with 64 and 128 neurons respectively for the middle cube and one feature learning component with 64 neurons for the side

cubes. The first dense layer consists of 1024 and the second one of 2048 neurons, like in the network using middle grid size, and a batch size of 64 was used too. The rest of the network and hyperparameters stayed as in the networks before and the additional feature of average point colour was used as well.

The network using big grid size achieves for the eight class problem about the same overall accuracy as the network using middle grid size, it is just 0.7% better. By looking at the IoU values, which are shown in Table 5.7, it can be seen that, in comparison to the network using middle grid size, this network performs for the most classes slightly better, with up to 3.97% for the class "cars". On the other hand, the classes "high vegetation" and "artefacts" perform worse, with 0.14% and 0.74% respectively. Furthermore, the average IoU of all classes for the network using big grid size is about 1.06% better than with the network using middle grid sized.

classes	***8 classes***	***6 classes***
	IoU in %	
man-made terrain	84.58	90.26
natural terrain	84.09	
high vegetation	69.81	84.38
low vegetation	57.99	
buildings	67.88	66.19
hard scape	54.20	51.91
artefacts	77.85	77.42
cars	74.22	73.82
average over all classes	71.33	74.00
	OA in %	
overall accuracy	82.84	86.81

Table 5.7: IoU of the single classes, average IoU and OA of the test set classified with the network using big grid size. Once with the original eight classes and once with the combined six classes.

The confusion matrix of the network using big grid size and eight classes is shown in Table 5.8, there it can be seen that most misclassifications still occur between the classes "high vegetation" and "low vegetation" as well as "buildings" and "hard scape". In comparison to the network using middle grid size and eight classes the accuracies of the single classes is slightly worse for "man-made terrain", "high vegetation" and "buildings", with about 1.27%, 2.38% and 0.16% respectively. On the other hand, the classes "natural terrain", "low vegetation", "hard scape", "artefacts" and "cars" perform better with the network using big grid size, with at most 3.97% for the class "low vegetation".

In addition, the network using big grid size was trained with six classes instead of eight. As it can be seen in Table 5.7, the network using six classes has an overall accuracy of 86.81%, which is 4.57% better than with the network using eight classes. The IoU-value for the combined class "terrain" is 5.93% better than the average of the classes "man-

	MMT	***NT***	***HV***	***LV***	***B***	***HS***	***A***	***C***
MMT	91.43	4.44	0.00	0.00	0.79	1.59	0.63	1.11
NT	3.17	90.63	0.00	4.44	0.48	1.11	0.00	0.16
HV	0.00	0.00	83.33	13.02	0.63	1.43	1.59	0.00
LV	0.32	2.38	15.56	74.29	1.27	3.49	1.27	1.43
B	1.43	0.32	1.11	2.54	77.14	10.95	3.49	3.02
HS	1.27	0.32	1.27	5.24	7.94	69.68	5.71	8.57
A	0.95	0.16	0.32	1.11	1.27	4.29	89.84	2.06
C	0.95	0.16	1.11	1.75	1.27	5.71	2.70	86.35

Table 5.8: Confusion matrix of the test set classified with the network using big grid size. (Note: values are in %.)

made terrain" and "natural terrain". The IoU-value of the combined class "vegetation" is 20.48% better than the average of the classes "high vegetation" and "low vegetation". The IoU-values of the remaining classes are slightly worse than with the network using six classes, with at most of 2.29% for the class "hard scape". However, the average of all IoU-values is increased 2.67% with the network using six classes. This can also be seen by comparing the two confusion matrices (see Table 5.8 for the network using eight classes and Table 5.9 for the network using six classes), the accuracy of the combined class "terrain" performs 5.32% better than the average accuracy of the classes "man-made terrain" and "natural terrain". The combined class "vegetation" has an increase in the accuracy of 14.21% compared to the average of the classes "high vegetation" and "low vegetation". Furthermore, the class "buildings" performs 3.34% better too, but the classes "hard scape", "artefacts" and "cars" perform worse, with 5.08% for the class "hard scape". Moreover, it also can be seen that misclassifications between the classes "buildings" and "hard scape" are still very high for networks using eight and six classes.

	terrain	***vegetation***	***buildings***	***hard scape***	***artefacts***	***cars***
terrain	96.35	1.59	0.56	1.03	0.24	0.24
vegetation	2.14	93.02	1.43	1.51	1.11	0.79
buildings	2.70	4.13	80.48	7.30	3.81	1.59
hard scape	3.02	7.30	13.33	64.60	5.24	6.51
artefacts	1.27	2.06	3.02	4.13	88.73	0.79
cars	2.22	3.81	1.27	7.94	2.86	81.90

Table 5.9: Confusion matrix of the test set classified with the network using big grid size with six classes.

By comparing the network using big grid size and six classes with the network using middle grid size and six classes, it can be seen that the network using big grid size has an increase in accuracy of just 0.12% and also the average IoU performs just 0.06% better. The IoU-values for the network using big grid size are slightly better for the classes "terrain", "vegetation", "artefacts" and "cars", but the classes "buildings" and "hard

scape" perform worse, with at most about 2% for the class "hard scape". By comparing the accuracies of the single classes in the confusion matrices of Table 5.6 for the network using middle grid size and of Table 5.9 for the network using big grid size, it can be seen that the classes "terrain", "vegetation", "buildings" and "artefacts" perform better with the network using big grid size. They do this with at most 5.40% for the class "buildings", but the remaining classes "cars" and "hard scape" perform worse, with at most 5.40% for the class "hard scape".

As it can be seen in Table 5.10 the overall accuracy increases with an increased grid size, whereby the largest increase is between the networks using small and the middle grid sizes. This is probably due to the increased grid size of the neighbouring cells. Besides that, the networks using six classes are performing better than the networks using eight classes with all three grid sizes, whereby the greatest difference between the accuracies is from the network using small grid size to the network using middle grid size too. Since, the network using big grid size performs better than the networks using small and middle grid sizes, it is used for the semantic segmentation of the point cloud. Figure 5.2 visualizes the architecture of this network in more detail.

	8 classes		**6 classes**		difference
	OA	increase	OA	increase	
network using small grid size	80.87	-	85.69	-	4.82
network using middle grid size	82.14	1.27	86.69	1.00	4.55
network using big grid size	82.84	0.70	86.81	0.12	3.97

Table 5.10: Overall accuracies of the three different networks, increase in the OA between the networks as well as difference between six and eight class networks (Note: values are in %).

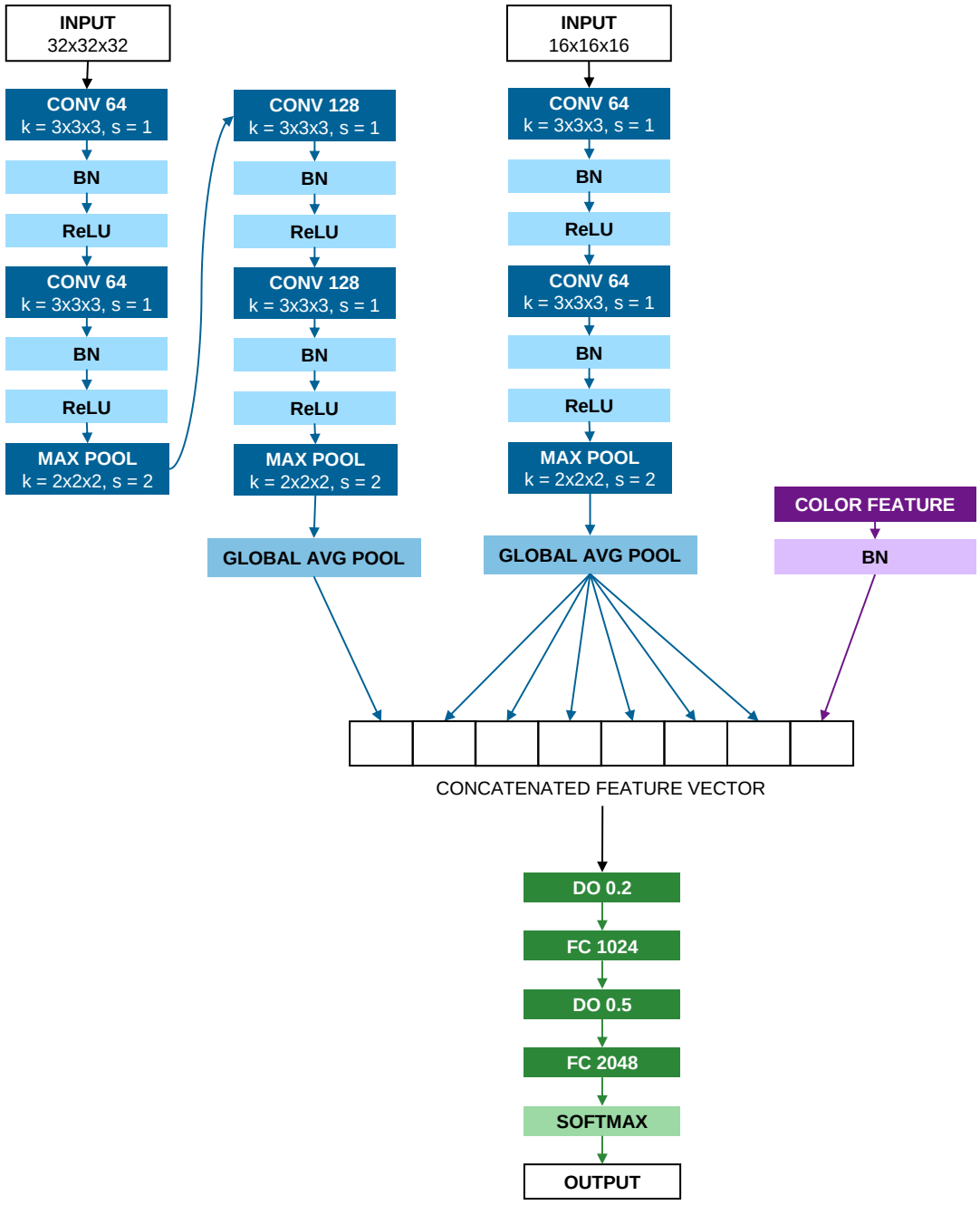


Figure 5.2: Architecture of the network using big grid size.

CHAPTER 6

Results

This chapter is dedicated to results of the semantic segmentation using 3D CNNs as well as results of the curb fitting. Section 6.1 shows results of the semantic segmentation on the Semantic3d training as well as test set. Section 6.2 presents the outcome of the semantic segmentation as well as of the curb fitting on a real world dataset of a city in Germany. Finally, a critical reflection is given in Section 6.3, whereby 3D CNNs as well as curb fitting will be discussed shortly.

6.1 Semantic Segmentation of the Semantic3d Dataset

A segmented point cloud can either be evaluated categorically, if ground truth class labels are available, like for the Semantic3d dataset, or visually, if no ground truth class labels are existing, like for the Cologne dataset. The numerical evaluation is performed with the same metrics as in Chapter 5, but pointwise instead of boxes.

For visual evaluation, the points of the point cloud are coloured according to their class affiliation. “Man-made terrain” is grey, “natural terrain” is light-green, “high-vegetation” is green, “low vegetation” is dark-green, “buildings” are blue, “hard scape” is orange, “cars” are pink and “artefacts” are brown-pink. In case of six classes instead of eight, the combined “terrain” class is coloured grey and the combined “vegetation” class is coloured green. The classes and their colours are listed in Table 6.1, whereas on the left side are the colours for eight classes and on the right side are the colours for six classes visualized.

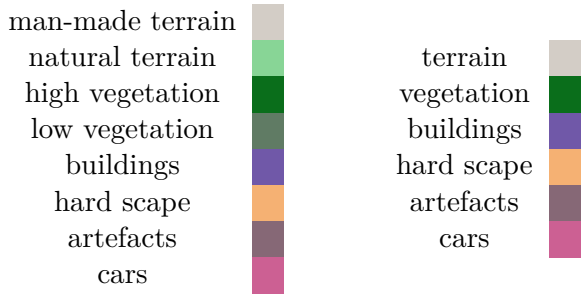


Table 6.1: The points of the point cloud are coloured according their assigned classes. Left: Colours for eight classes. Right: Colours for six classes with combined "terrain" and "vegetation" classes.

6.1.1 Semantic3d Training Set

The 15 point clouds of the Semantic3d training set were used for training of the 3D CNN. However, not the entire point clouds were used for training, but only parts of them. Since, the positions of the boxes, which were used for training, are known, the points of a point cloud can be split into two sets: *used for training* and *not used for training*. These two sets can then be numerically evaluated separately.

The results of this evaluation are shown in Table 6.2 and there it can be seen that the the network using eight classes has nearly the same accuracy for points *used* and *not used for training*, with about 90% and 91% respectively. Most classes perform similar in those two sets, except of "low vegetation" and "hard scape"; they have a deviation of about 10% in favour of points *used for training*. During the training, it became apparent that the network has problems distinguishing "high vegetation" and "low vegetation" due to its similarity. However, most misclassifications of "low vegetation" in points *not used for training* occur at "hard scape". The class "hard scape" does not perform as good as other classes in the training set as well. Misclassifications occur in both directions, which is evident in the IoU of this class with about 63% too. For points *not used for training* this problem does have stronger effects and the IoU is about 24%. This is probably caused by many different things being labelled as "hard scape", such as walls, fences, flagpoles, streetlights and fountains to name a few. All these things have a diverse structure in the point cloud and the network is probably confused due to this. A targeted relearning and refinement of this class cloud probably solve this problem. Furthermore, the classes "artefacts" and "cars" differ of about 4-5% between *used* and *not used for training*. Whereby, "artefacts" perform better in *not used for training* and "cars" in *used for training*. "Cars" have an accuracy around 98% and 93% in *used* and *not used for training* respectively. This class may be slightly overfitted during the training. "Artefacts" are not performing well in both sets. They have an accuracy of approximately 33% for points *used* and 38% for points *not used for training*. In points *used for training*, most misclassifications occur between this class and "buildings" as well as "hard scape". In points *not used for training*, most misclassifications occur between this class and

"man-made terrain" as well as "hard scape". The misclassifications between "buildings" and "hard scape" may be caused by the similar structure of exterior walls of buildings and low walls or fences. The misclassifications between "man-made terrain" and "hard scape" probably results from the cube data structure. By looking at the classified point cloud, it becomes apparent that many "artefacts" are caused by passing people and are thereby located at the ground. If a cube is classified as "man-made terrain" but has "artefacts" in it, those "artefacts" are wrongly labelled as "man-made terrain".

	used for training		*not used for training*	
	8 classes	6 classes	8 classes	6 classes
	Accuracy in %			
man-made terrain	94.05	96.76	94.33	97.71
natural terrain	85.43		89.24	
high vegetation	92.10	98.69	89.48	96.74
low vegetation	96.43		87.27	
buildings	93.54	95.39	90.36	92.56
hard scape	81.01	78.02	68.62	59.50
artefacts	33.45	24.39	37.82	40.33
cars	97.71	95.35	92.72	91.64
overall accuracy	90.25	93.73	91.07	95.51
	IoU in %			
man-made terrain	89.22	95.74	90.78	96.30
natural terrain	81.05		84.54	
high vegetation	87.32	93.86	89.05	92.39
low vegetation	88.11		43.07	
buildings	80.43	76.12	88.90	90.89
hard scape	63.87	65.02	24.27	28.95
artefacts	30.26	22.85	20.38	24.99
cars	65.35	68.60	6.84	8.99
average over all classes	73.20	70.36	55.98	57.09

Table 6.2: Final semantic segmentation results of the Semantic3d training set.

Furthermore, it can be seen in Table 6.2, that the network using six classes has nearly the same accuracy for points *used* and *not used for training* too, with about 94% and 96% respectively. "Terrain", "vegetation", "buildings" and "cars" are performing almost equal is both sets, with about 1-4% difference. "Hard scape" and "artefacts" have a big difference in accuracy too. Whereby "hard scape" performs about 19% better for points *used for training* and "artefacts" performs about 16% worse. "Hard scape" points are misclassified through all classes and have corresponding IoU values of about 65% and 29% for points *used* and *not used for training* respectively. The reason for this is most probably the same as for the network using eight classes. Points labelled as "hard

scape" belong to different objects with diverse structures and this problem is enhanced for unknown points. "Artefacts" are performing in the network using six classes bad too, with just about 24% and 40% accuracy for points *used* and *not used for training* respectively. Most misclassified "artefacts" in points *used for training* occur between this class and "buildings" as well as "hard scape". However, in points *not used for training* most misclassifications occur between this class and "terrain". The reason for those misclassifications are most probably the same as for the network using eight classes.

In addition to that, Table 6.2 shows that the network using eight classes and the network using six classes have a similar overall accuracy, where the network using six classes performs a couple percent better. In both sets, *used* and *not used for training*, the combined "terrain" and "vegetation" classes outperform the respective single classes. "Buildings" and "cars" are performing approximately equal in those sets. "Hard scape" performs similar in points *used for training*, but in *not used for training* the network using six classes performs worse than the network using eight. "Artefacts" are performing for points *not used for training* better for both networks, but with a greater difference in the network using six classes. This difference between the two networks in those classes may be caused by the diversity in "hard scape" objects too and the problem with "artefacts" in general.

The semantic segmentation of the Semantic3d datasets "bildstein station 5" and "sg27 station 1" are shown in Figure 6.1 and 6.2 respectively. The top images display the according scene, the second row shows the ground truth labels, the third row the segmentation predictions and the bottom row visualizes if the semantic segmentation is correct (blue points) or not (orange points). The left column depicts a classification with eight classes and the right column with six classes. In Figure 6.1, it can be seen that, the overall classification works quite well for eight as well as six classes, whereby the six class classification is more accurate. Most misclassifications occur due to the representation of the data in the 3D CNN, since cubes with one meter side length are used as data samples. For example, if a cube contains mostly car and some terrain points, it will be predicted as "cars" from the neural network. Thus, all points in the cube are labelled as "cars" - also terrain points, which is incorrect. Another frequent source of misclassifications are the confusion between the classes "buildings" and "hard scape", which can be seen at the leftmost building and at the barn in Figure 6.2. Furthermore, in Figure 6.2 it can also be seen that misclassifications between the classes "high vegetation", "low vegetation" and "natural terrain" occur commonly. In addition to that, it can also be determined that in this scenario a classification with six classes performs better than with eight classes.

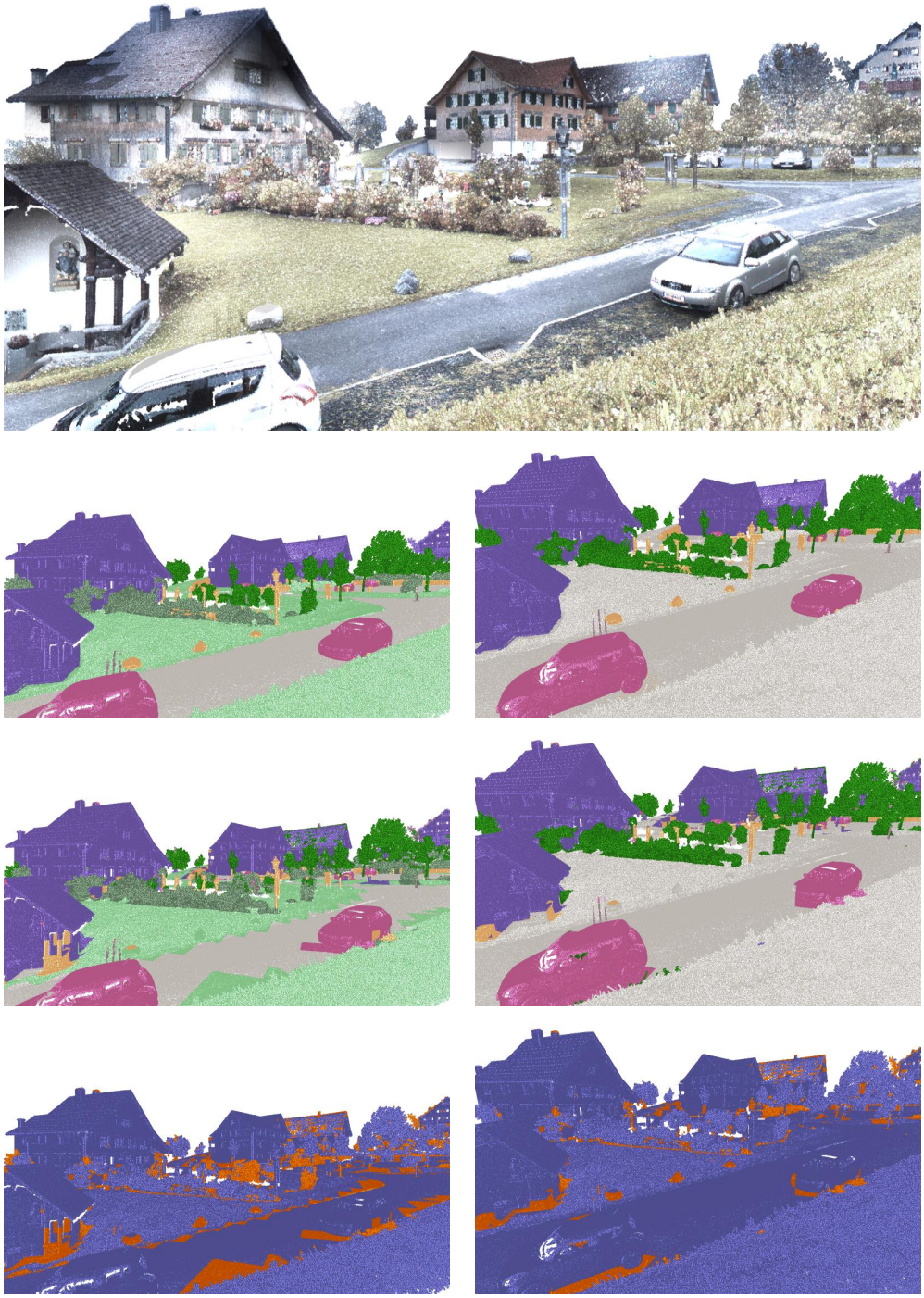


Figure 6.1: Semantic3d training dataset “bildstein station 5”. Second row: ground truth, third row: network predictions, bottom row: correct predicted (blue) or not (orange). The left column depicts a classification with eight classes and the right with six.

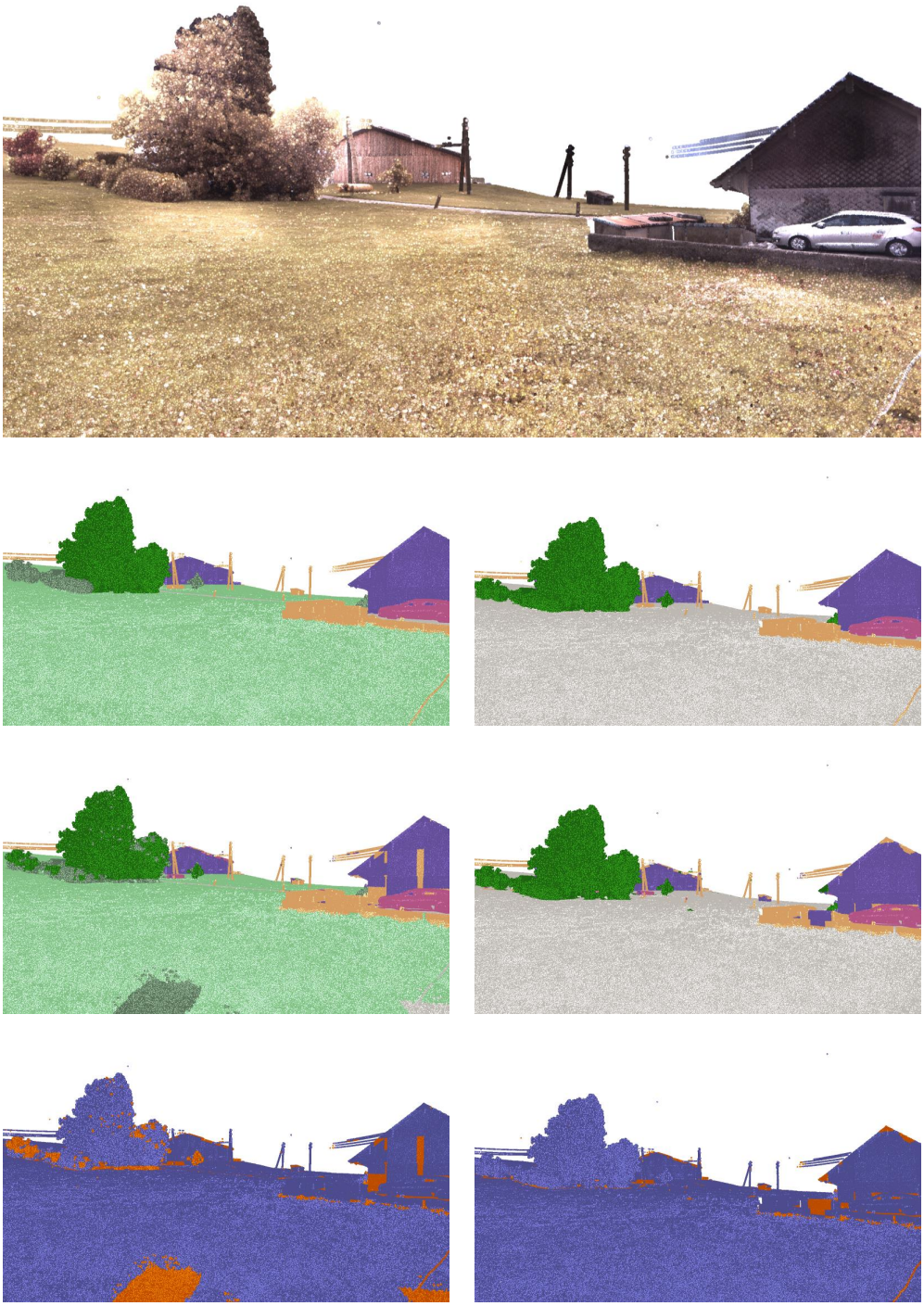


Figure 6.2: Semantic3d training dataset “sg27 station 1”. Second row: ground truth, third row: network predictions, bottom row: correct predicted (blue) or not (orange). The left column depicts a classification with eight classes and the right with six.

6.1.2 Semantic3d Test Set

By looking at the semantic segmentation of the Semantic3d test set with eight classes in Figure 6.3, it can be seen that the overall classification works quite well too, with the same weaknesses as in the Semantic3d training set. Most misclassifications occur between the classes "buildings" and "hard scape" and besides that, the vegetation classes as well as "natural terrain" are being mixed up. This can be seen quite well in the pictures of the second row. Furthermore, in the top row it can be determined that the shopping windows are labelled as "natural terrain" and not as part of the building; this might be probably caused by reflections of the opposite meadow. The dataset of the third row is segmented quite well, but the class affiliation of the vegetation in the background might not be correct and also some misclassifications between the classes "natural terrain" and "man-made terrain" might occur due to the cubic representation of the point cloud in the 3D CNN. The bottom row may also suffer from this, whereby here misclassifications between the classes "man-made terrain" and "artefacts" occur and the classes "buildings" and "hard scape" are mixed up at the building on the right side.

These classification results were also submitted to the Semantic3d benchmark, where an overall accuracy of 85.1% and an average IoU of 51.4% was achieved. By taking a look at the exact results, which are shown in Table 6.3, it can be seen that the classes "low vegetation", "artefacts" and "hard scape" are performing the least well. This has already become apparent in Table 6.2 at *points not used for training*, where those classes produced some of the weakest results too.

	Accuracy in %
overall accuracy	85.1
	IoU in %
man-made terrain	89.6
natural terrain	77.2
high vegetation	47.7
low vegetation	27.8
buildings	85.5
hard scape	24.7
artefacts	24.5
cars	34.2
average over all classes	51.4

Table 6.3: Final semantic segmentation results of the Semantic3d test set.

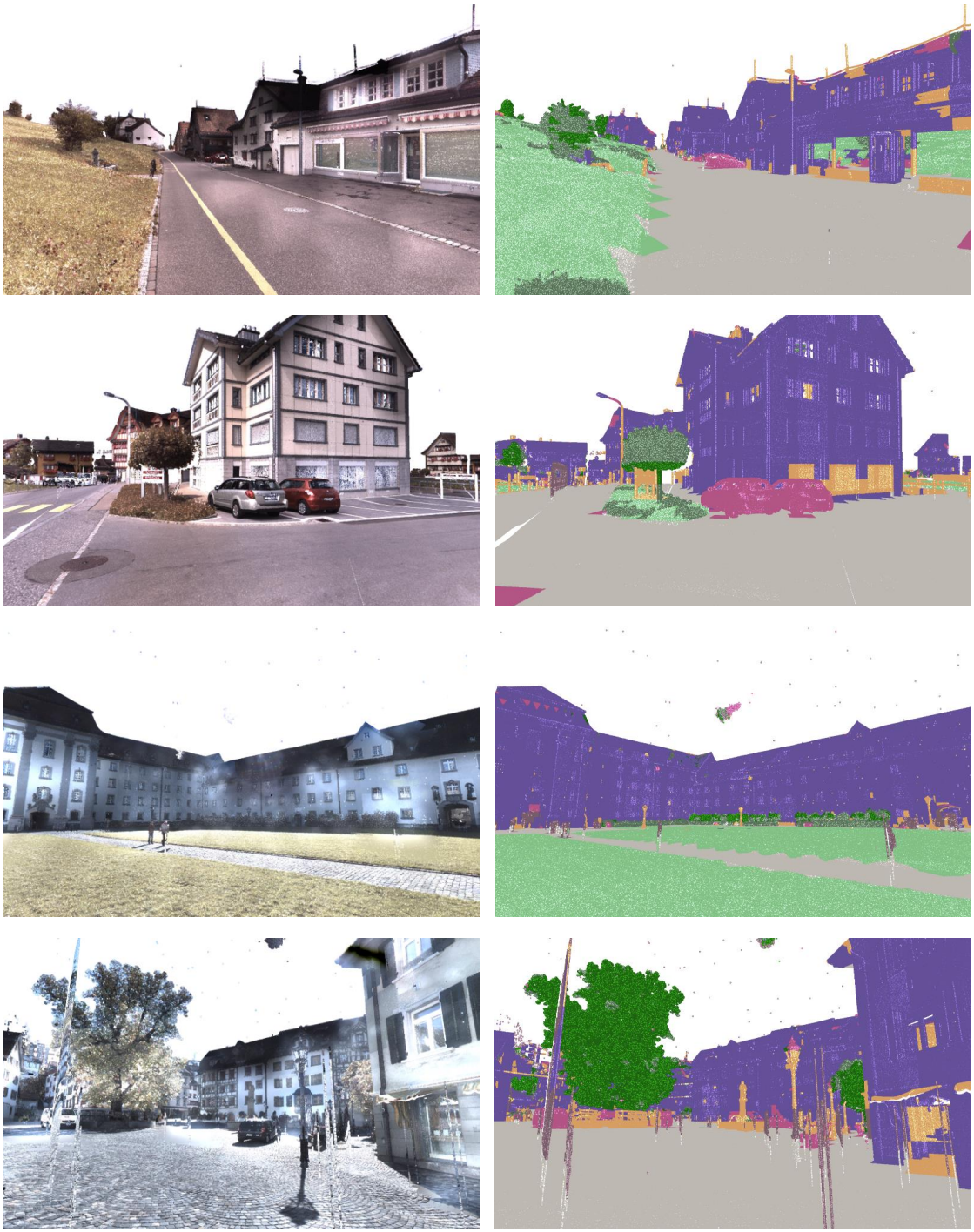


Figure 6.3: Semantic segmentation of datasets from the Semantic3d test set. First row: “sg27 station 3”, second row: “sg27 station 6”, third row: “st gallen cathedral station 1”, bottom row: “st gallen cathedral station 6”.

6.2 Semantic Segmentation and Curb Fitting of the Cologne Dataset

Cologne is a city in Germany with about 1.1 million inhabitants and an area of around 405 square kilometres, whereby 66 square kilometres are public thoroughfare [Kö]. *CycloMedia Deutschland, LiDAR Point Cloud* and *Stadtentwässerungsbetriebe Köln, AöR* have scanned the streets of the whole city with LiDAR devices, resulting in raw data of over 1.5 TB. However, for this thesis, not the entire point cloud was processed, but only individual streets were evaluated. These streets were chosen carefully to provide representative urban scenarios and are called as follows:

- An der Schanz
- Auenweg
- Augustinerstraße
- Äußere Kanalstraße
- Kleiner Griechenmarkt.

As mentioned above, those streets were chosen carefully to provide meaningful use cases for a city. They include urban areas with many buildings and parking cars next to the streets, rural areas with lots of vegetation, roads with curves as well as straight streets. With this selection, the semantic segmentation with a network using six classes can be tested thorough, because there are a lot of terrain, vegetation, buildings and cars present. Furthermore, with this data the network is tested on a different dataset than the training data, which is evaluated visually, since there is no ground truth available. Besides that, those selected streets cover the use case of simple straight streets, curves and partly occluded sidewalks. This is a nice set to evaluate how well this method works in the simplest (straight streets) and harder use cases (curves) as well as how it manages missing information about curbs.

For all five streets, the result of a semantic segmentation with a 3D CNN using the big grid size and six classes, the detection of curb points with geometric point cloud features and the final reconstructed geometry of the street, curbs and sidewalks are shown and discussed below. Something that can be said in advance is that, the segmentation with the trained 3D CNNs works quite well, keeping in mind, that those Cologne data are different from the training data. The detection of curb points on terrain points of the semantic segmentation is robust and the final reconstructed geometry is quite accurate with an error of a few centimetres.

6.2.1 An der Schanz

The first street of the Cologne dataset is called “An der Schanz” and it is a road with two separated driving lanes, which are architecturally divided by a railway track. However, both lanes were evaluated in this thesis and the point cloud of those two lanes are shown in Figure 6.4.

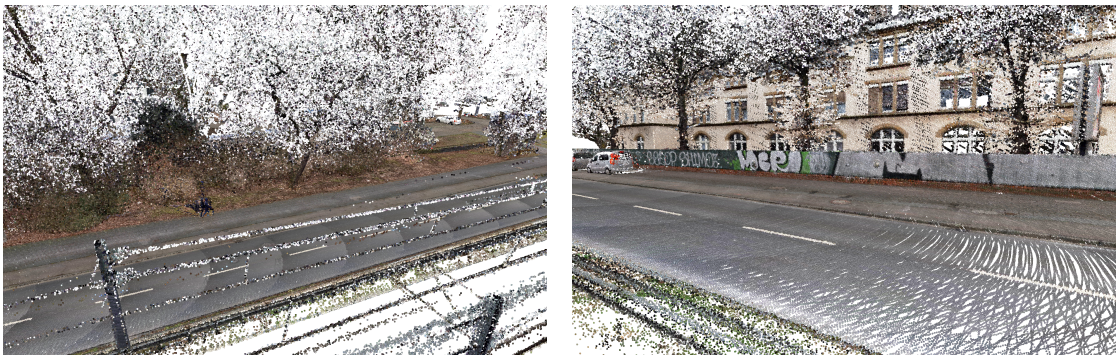


Figure 6.4: The LiDAR point cloud of the two driving lanes of the street in Cologne named “An der Schanz”.

A semantic segmentation was applied to the point cloud with the network using big grid size and six classes. The result is visualized in Figure 6.5. There it can be seen that the segmentation works quite well for the classes “terrain”, “vegetation” and “buildings”. However, as it can be seen in the left image of Figure 6.5, there are some misclassifications at the barriers and the rail wires in the middle of the two lanes.

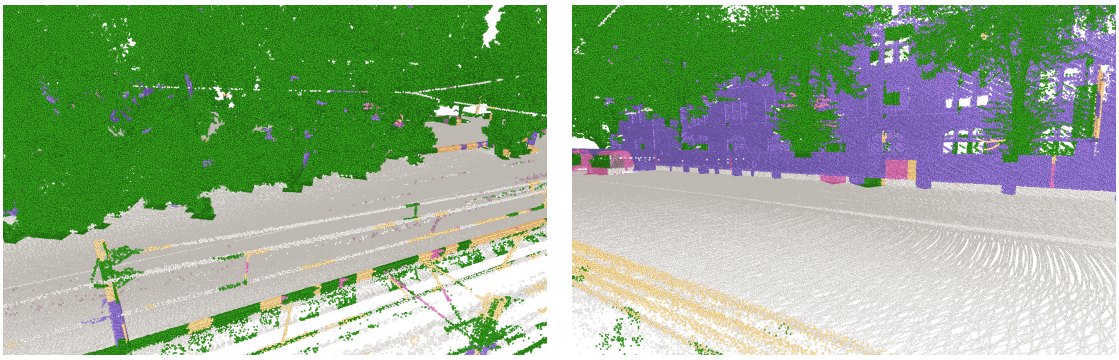


Figure 6.5: The result of the semantic segmentation of the network using big grid size and six classes of the street named “An der Schanz”.

Next, the curb points are calculated by using geometric features. Figure 6.6 shows the resulting curb points in magenta. In the left image not much curb points are detected, this might be caused by the rather low curb height. On the other hand, the curb point computation of the right image works well, but it can be seen that points of the wall in the background, which are labelled as terrain, are also detected as curb points due to the similar geometric features.

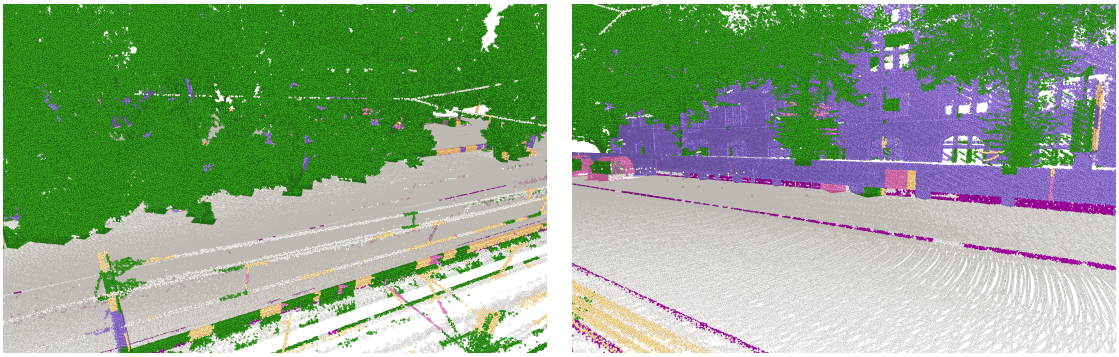


Figure 6.6: Detected curb points coloured magenta of the geometric feature calculation of the street named “An der Schanz”.

However, the next steps of the algorithm nevertheless manage to compute reliable geometry. The geometry of the curbs, street and sidewalk have a mean error of $0.0169m$, $0.0146m$ and $0.0251m$ respectively. The mean error is the average distance from points of the point cloud to the final geometry within a distance of $25cm$. The points within this threshold are filtered according to their normal vector to avoid mixing street points into the calculation of the curb error and vice versa. If the normal is pointing up, the points belongs either to the street or sidewalk; if not, the points belongs to the curb. The results can be seen in Figure 6.7, where the reconstructed street is visualized in green, the curb itself in red and the sidewalk in blue. Furthermore, Figure 6.8 shows the final calculated geometry of both lanes from a bird’s eye perspective.

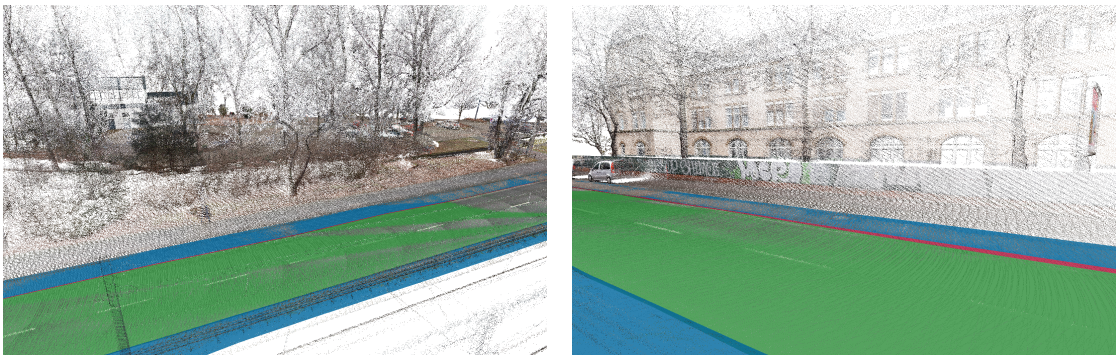


Figure 6.7: The reconstructed geometry of the street named “An der Schanz”.

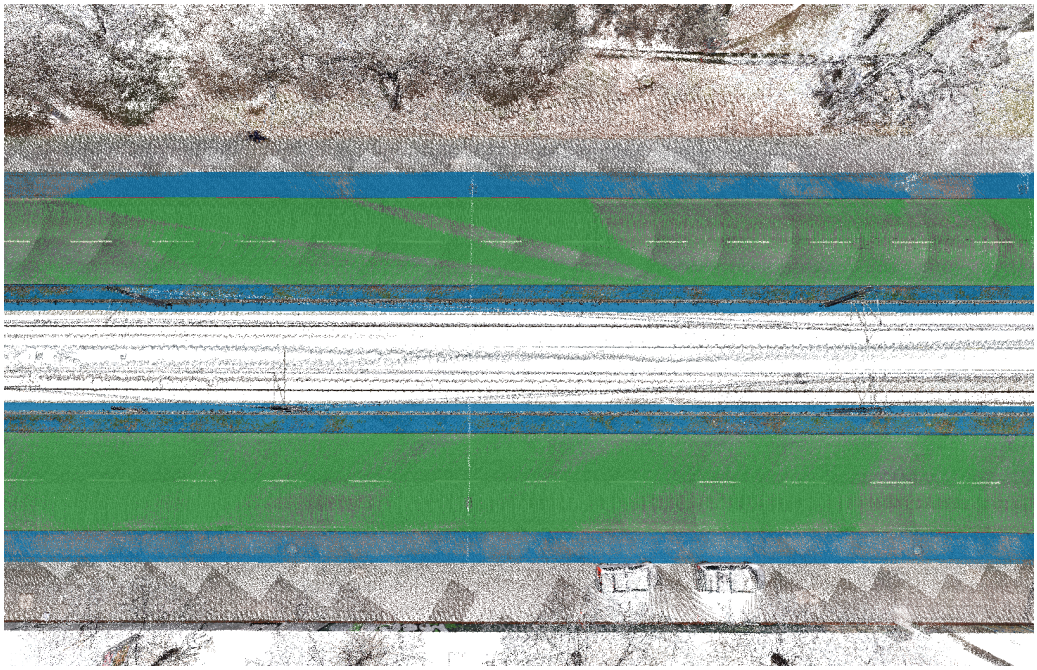


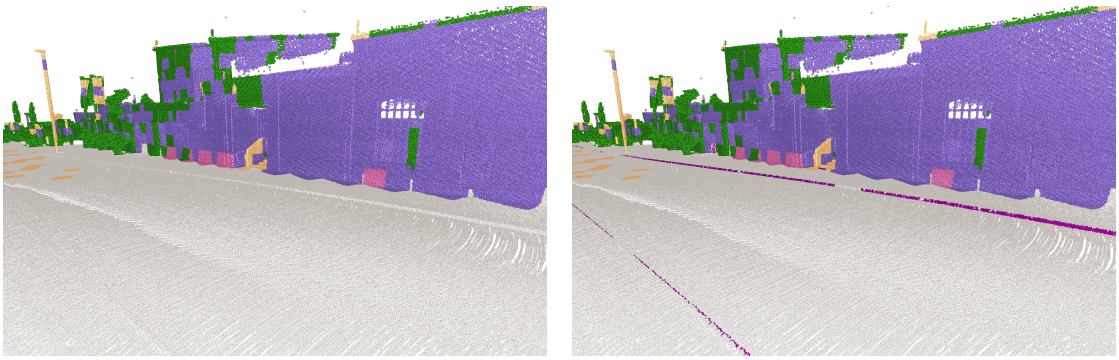
Figure 6.8: Top view of the reconstructed geometry of the street named “An der Schanz” of both lanes.

Figure 6.9: The LiDAR point cloud of the street named “Auenweg”.

6.2.2 Auenweg

The second street evaluated in this thesis is named “Aueweg”, which is near a park, the river Rhein and big congress buildings. The point cloud of this street captured with a LiDAR device is shown in Figure 6.9.

The semantic segmentation, which can be seen in Figure 6.10 (a), with the 3D CNN using big grid size and six classes, works quite well for the class “terrain”, even if there are some chunks classified as “hard scape”, which are those orange squares on the street. The classification of the building besides the street is okay, there are some pieces predicted as “cars”, the pink squares, and some areas, which are labelled as “vegetation”, which is visualized with the colour green. However, the most important parts for curb detection, namely terrain and sidewalks, are predicted correct. Thus, the calculation of the curb features works quite well, even if there is a big gap of not detected curb points, which is shown in Figure 6.10 (b).



(a) (b)

Figure 6.10: (a) The result of the semantic segmentation with the network using big grid size and six classes. (b) The result of the geometrically calculated curb feature points of the street named “Auenweg”.

Nevertheless, the algorithm for the geometry reconstruction is able to handle such gaps and produces a continuous geometry for the street, sidewalks and curbs. The mean error for this dataset is $0.0322m$, $0.0314m$ and $0.0157m$ for the curbs, street and sidewalk respectively. This final resulting geometry can be seen in Figure 6.11.

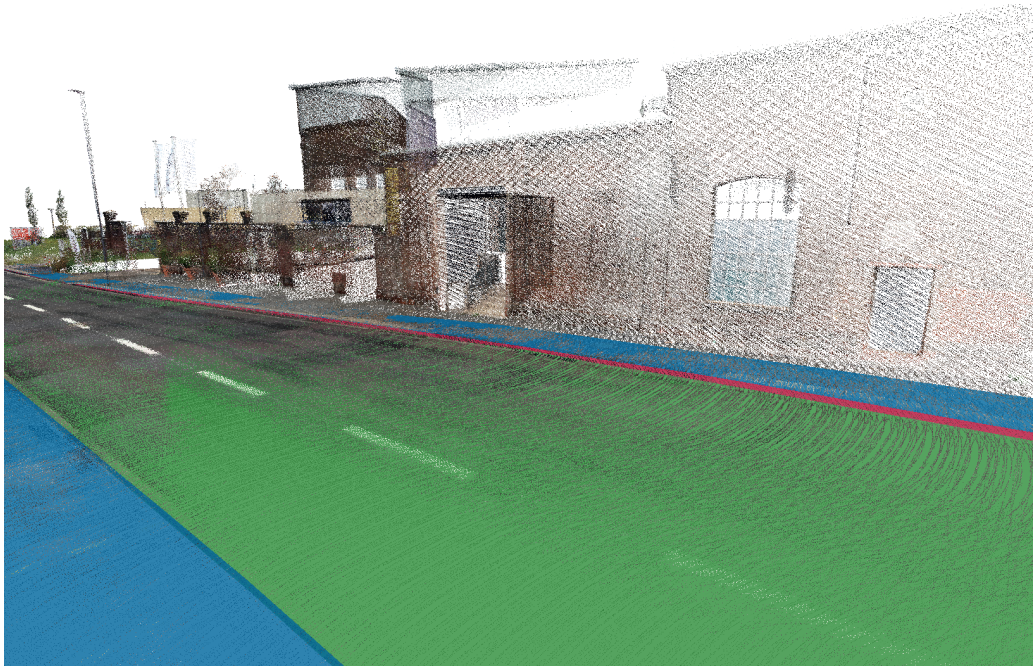


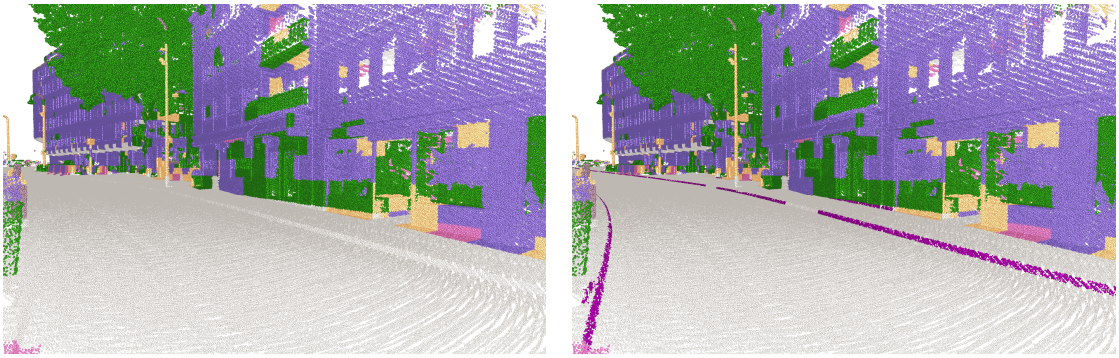
Figure 6.11: The final reconstructed geometry of the street called “Auenweg”.

Figure 6.12: The LiDAR point cloud of the urban street named “Augustinerstraße”.

6.2.3 Augustinerstraße

An urban street named "Augustinerstraße" is the next street in the dataset and its LiDAR point cloud is visualized in Figure 6.12. The semantic segmentation with the 3D CNN, which is shown in Figure 6.13 (a), works quite well for the classes "terrain" and "vegetation" too, but some parts of the buildings are wrongly classified as either "vegetation" or "hard scape". By taking a closer look, it can be noticed that this misclassifications occur most frequent in shopping windows or glass doors. This might probably be caused by reflections or geometric properties of those windows.

The detection of the curb points by using geometric point cloud features works quite well, but also in this dataset, there are some misclassifications due to incorrect labelled terrain points and holes in the curb. The result of the curb point calculation can be seen in Figure 6.13 (b).



(a) (b)

Figure 6.13: (a) The result of the semantic segmentation with the network using big grid size and six classes. (b) The result of the geometrically calculated curb feature points of the street named "Augustinerstraße".

However, also here the method used in this thesis is able to cope with that and calculated a continuous geometry, which is shown in Figure 6.14. Furthermore, it can be seen that the street makes a bend. Nonetheless, the algorithm is able to compute an accurate geometry along this curve as well. The mean error is $0.0182m$ for the curb, $0.0390m$ for the street and $0.0246m$ for the sidewalk. The resulting geometry of the whole curve of the road can be seen from above in Figure 6.15.

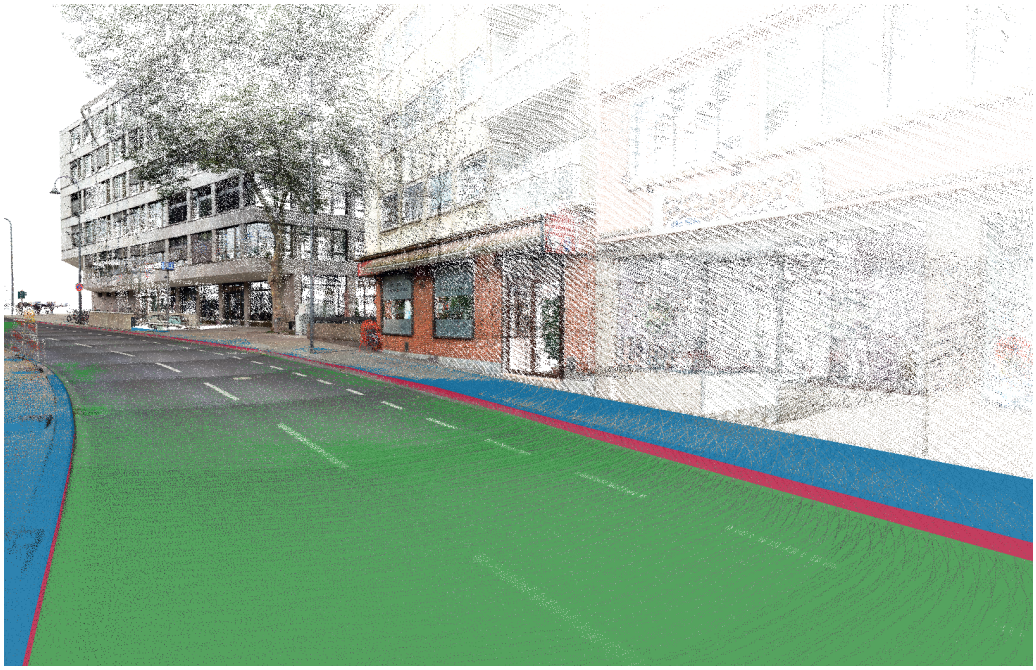


Figure 6.14: The final reconstructed geometry of the street named “Augustinerstraße”.

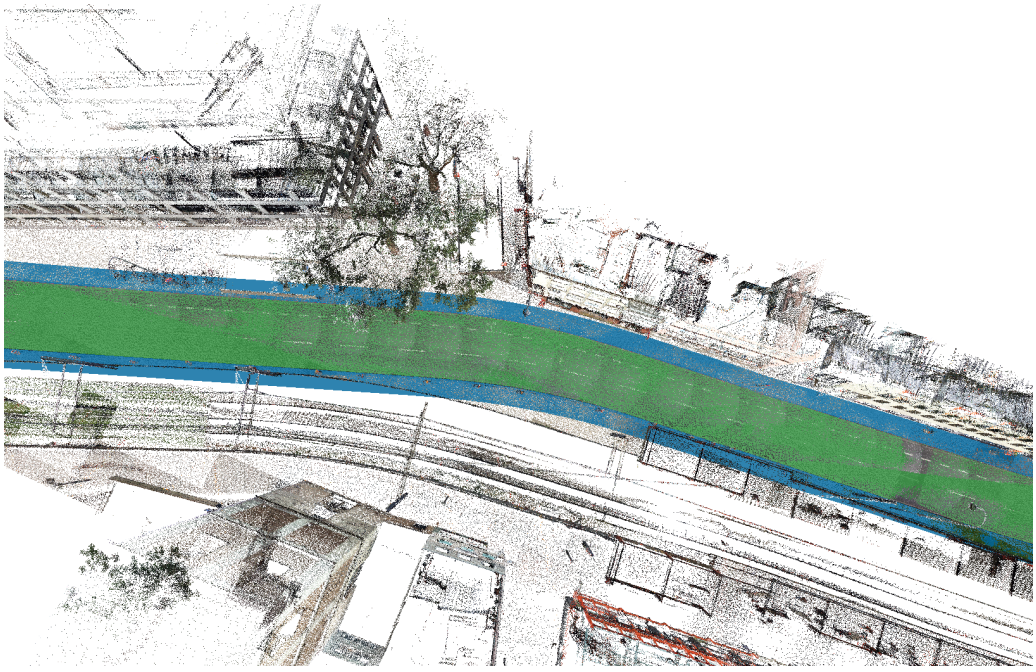


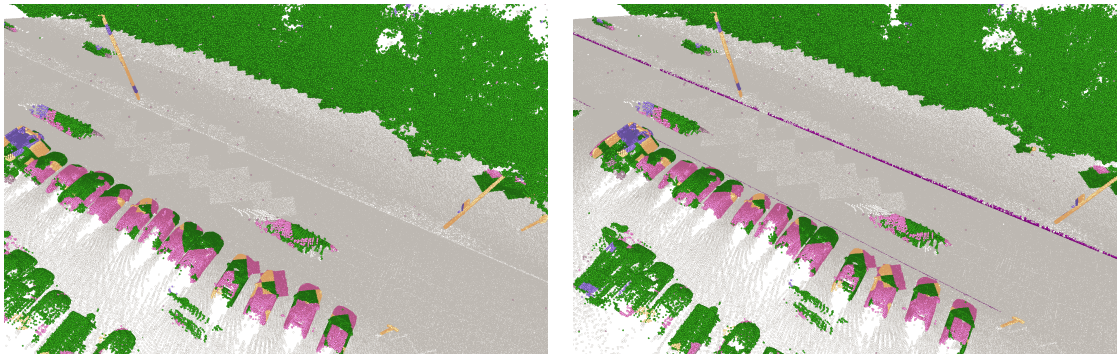
Figure 6.15: The final reconstructed geometry of the street named “Augustinerstraße” from above, where the curve of the street can be seen.

6.2.4 Äußere Kanalstraße

The second last street evaluated is named "Äußere Kanalstraße" and its LiDAR point cloud is visualized in Figure 6.16. In this dataset are a lot of parking cars besides the street. It can be seen in Figure 6.17 (a) that their classification works okay for cars parking directly besides the street, but cars in the second row are all classified as "vegetation". These LiDAR point clouds are captured by driving by on the street. By taking a closer look at the point cloud itself, it can be noticed, that this part of the point cloud is not as dense as parts directly besides the street. This is most probably the reason why those points are labelled as "vegetation", because there is no clear structure of a car in the data. Nevertheless, the terrain is segmented quite accurate and the vegetation is classified correct.

Figure 6.16: The LiDAR point cloud of the street named "Äußere Kanalstraße".

The calculation of the curb points works well in this dataset, which is shown in Figure 6.17 (b). In addition, here are some small holes present, but as already seen above, this is no problem for the rest of the algorithm. Furthermore, it should be mentioned, that left and right of the parking cars is an exit of the parking area and the driving lane is flattened to the level of the street. This is the reason why there are no curb points detected in this area.



(a) (b)

Figure 6.17: (a) The result of the semantic segmentation with the network using big grid size and six classes. (b) The result of the geometrically calculated curb feature points of the street named “Äußere Kanalstraße”.

The final reconstructed geometry is visualized in Figure 6.18 and there it can be seen that the algorithm detects the geometry of the street, curb and sidewalks accurately, with a mean error of $0.0112m$, $0.0261m$ and $0.0253m$ for the curbs, street and sidewalk respectively.

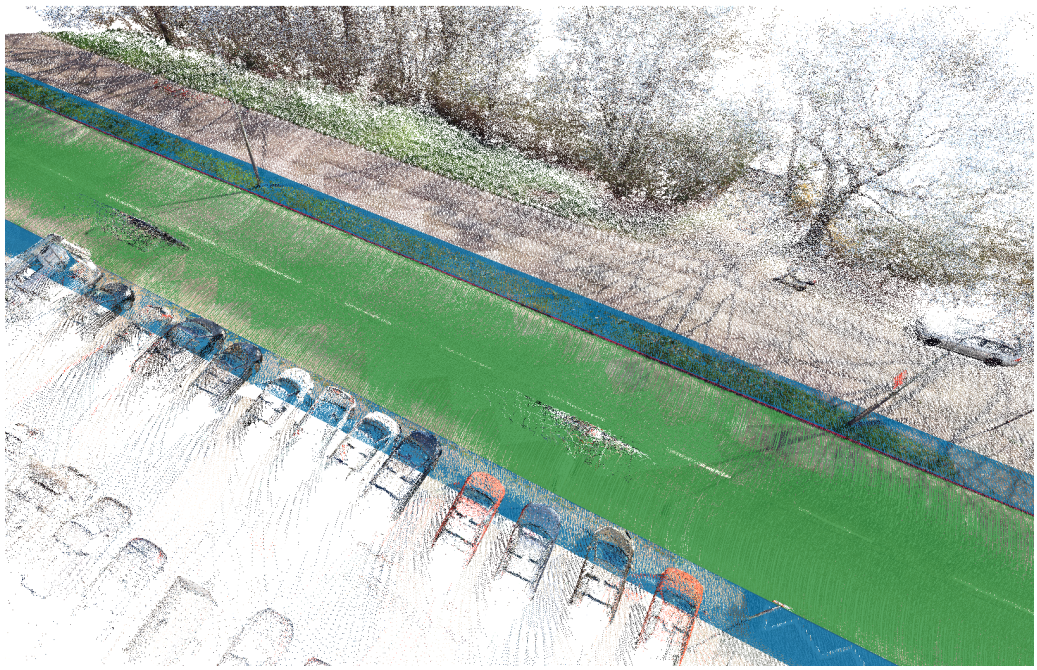


Figure 6.18: The final reconstructed geometry of the street named “Äußere Kanalstraße”.

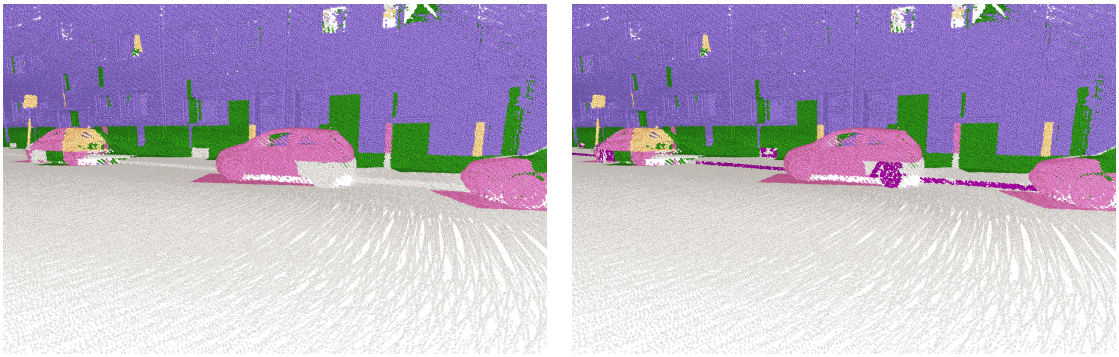
6.2.5 Kleiner Griechenmarkt

The last street evaluated in this thesis is called “Kleiner Griechenmarkt”, which is a typical urban street with a lot of parking cars besides the road. The point cloud captured with a LiDAR device is shown in Figure 6.19 and there it can be seen that he curb is covered by the cars.

Figure 6.19: The LiDAR point cloud of the street named “Kleiner Griechenmarkt”.

The semantic segmentation, which is shown in Figure 6.20 (a), with the 3D CNN works well for the class “terrain”, but some parts of the cars and buildings are misclassified as “terrain”. The cars are mostly classified correct, just the most left one is partly labelled as “hard scape”. The segmentation of the building works quite well too, but many lower parts are misclassified as “vegetation”.

The resulting curb points computed with geometric point cloud features are visualized in Figure 6.20 (b) and there it can be seen that those car and building points, which were misclassified as “terrain”, are causing false positive curb points, because those points have the same geometric features as curb points and are inside the street. Nevertheless, the actual geometry is reconstructed correctly, with a mean error of $0.0117m$ for the curbs, $0.0743m$ for the street and $0.0250m$ for the sidewalk. The mean error of the street about seven centimetres originates from the horizontal curvature of it. The calculated street polygon fits the street points well on the sides, where the curbs are, but the distance between the points and the polygon becomes higher towards the middle of the street.



(a) (b)

Figure 6.20: (a) The result of the semantic segmentation with the network using big grid size and six classes. (b) The result of the geometrically calculated curb feature points of the street named “Kleiner Griechenmarkt”.

What makes the detection of the curbs in this scene even more difficult is that there is no complete information about the curb available, because the point cloud is captured from the street. By taking a look at Figure 6.21, which shows the same scene from above, it can be seen that there are no points behind the parking cars present.

Figure 6.21: The LiDAR point cloud of the street named “Kleiner Griechenmarkt” from above.

However, the algorithm is able to compensate both problems: the false positive curb points and the missing information of the sidewalk behind the parking cars. The resulting geometries of the street, the curb and the sidewalk is shown in Figure 6.22 and in Figure 6.23 from above, where it can be seen that the geometry is calculated completely behind the parking cars.

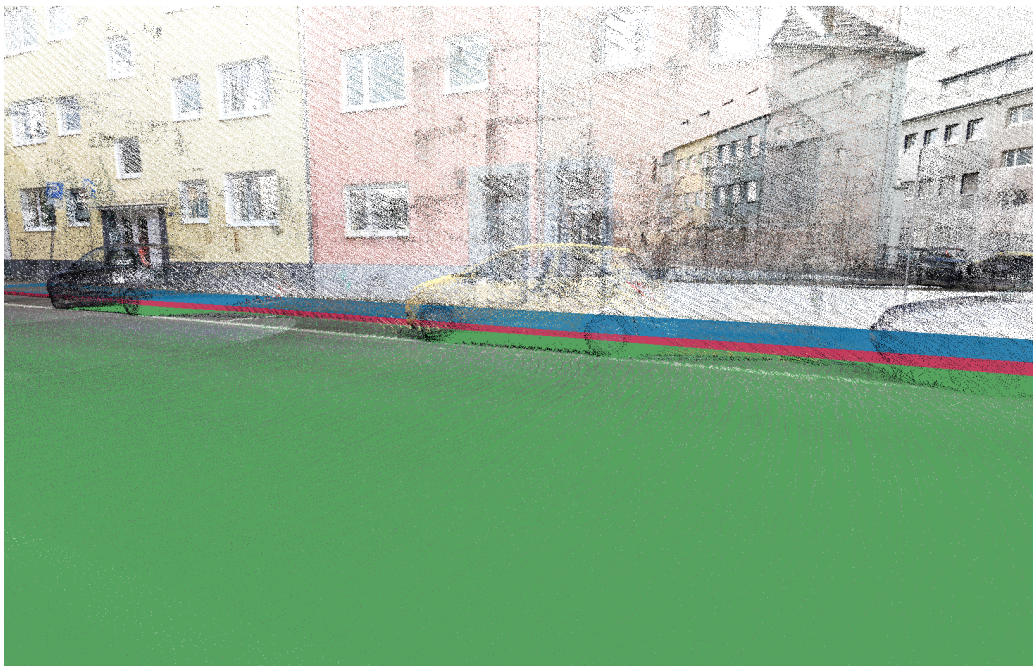


Figure 6.22: The final reconstructed geometry of the street, curb and sidewalk of the street named “Kleiner Griechenmarkt”.

The mean error of the final calculated geometry of all datasets are a couple of centimetres each. The overall mean reconstruction error is $\pm 2cm$ for curbs and sidewalks and $\pm 3cm$ for streets. An overview of the exact values for all datasets is given in Table 6.4.

	curbs	*street*	*sidewalk*
An der Schanz (1. side)	1.6	1.5	3.4
An der Schanz (2. side)	1.8	1.4	1.6
Auenweg	3.2	3.1	1.6
Augustinerstraße	1.8	3.9	2.5
Äußere Kanalstraße	1.1	2.6	2.5
Kleiner Griechenmarkt	1.2	7.4	2.5
Average	1.8	3.3	2.3

Table 6.4: Overview of mean reconstruction errors of all datasets (Note: values are in centimetres).

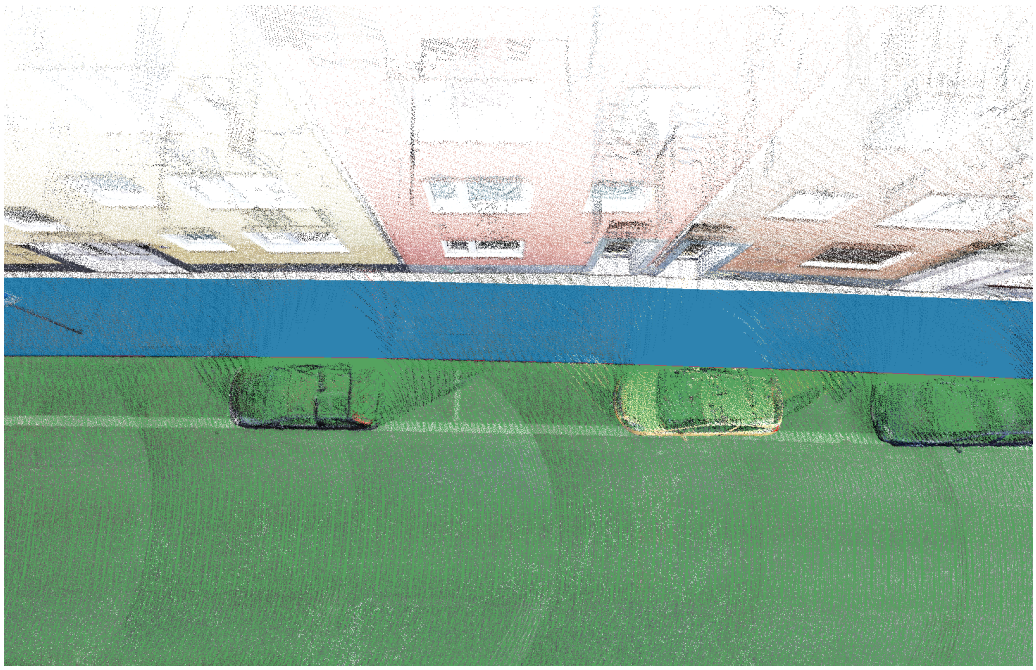


Figure 6.23: The final reconstructed geometry of the street, curb and sidewalk of the street named “Kleiner Griechenmarkt” from above.

6.3 Critical Reflection

In this section, a short discussion is conducted about what does not work so well and what could have been done better in this thesis. Whereby, Section 6.3.1 focuses on the semantic segmentation with 3D CNNs and Section 6.3.2 on the curb reconstruction process.

6.3.1 3D CNNs

Voxelizing a point cloud needs a lot of memory and it grows with an increased grid size. This is the reason that in this thesis a maximum grid size of 32 and a batch size of 32 for the network using big grid size was used. Otherwise, the network training fails due to memory exhaustion. Therefore, recent research is dedicated to find methods that are more efficient, but those approaches are not included in the Tensorflow library yet and thus not used in this work. By using more efficient 3D convolution and pooling operations, bigger grid sizes and deeper networks could be generated, which would most probably result in a more accurate point cloud segmentation. Furthermore, the segmentation is done by labelling all points in a cube with the overall label of it. This causes some

wrong classifications, for example when some terrain points are in a cell with mostly car points, then the terrain points are misclassified as “cars”. A direct segmentation with an appropriate segmentation network would not lead to such misclassifications and most probably increase the overall accuracy of the semantic segmentation.

6.3.2 Curb Fitting

Overall, the curb fitting works quite well, but the calculation of the curb points with geometric features sometimes produces many false positives, which could lead to inaccurate geometry. This could be fixed with either exploring or using more geometric point cloud features as well as enhancing the false positive filtering. Furthermore, the actual computation of the curb geometry could also be improved by automatically determining the best degree of the function fitted through the curbs points. In most cases, a parabola model is sufficient, but for streets with a stronger curve, a higher degree is required to model the curb geometry well. Moreover, the algorithm is currently just able to calculate curbs parallel to the street, but that is not always the case in real world data. Besides that, for this thesis only a handful of streets were tested, even if the selection was made with care to cover as many special cases as possible, there are certainly more cases where the algorithm fails to reconstruct the geometry correctly. Therefore, a wider range on different street scenarios would still have to be tested to find and solve those cases.

CHAPTER 7

Conclusion and Future Work

This thesis has shown a proof-of-concept work including a semantic segmentation of LiDAR point clouds by using 3D CNNs and in combination with that, a detection and reconstruction of sidewalk surfaces.

Current implementations of 3D CNNs work with rasterized data and to voxelize a point cloud, such that the underlying structure is represented well, an octree data structure was used. Therewith, too sparse regions of the point cloud can easily be identified and omitted in the training stage, which results in a better generalized classifier. Although, the semantic segmentation leaves room for improvement, it works quite well, especially for some classes, such as for example "vegetation". Furthermore, point cloud with a semantic segmentation could be useful in many applications, because a smart prefiltering, such that only points of interest are remaining, could decrease the amount of points to process enormously, especially when working with big datasets. Because of this, computation time could be reduced and results could be enhanced by eliminating high-risk points for incorrect results beforehand.

Such a prefiltering was also applied before the sidewalk detection and only points labelled as "terrain" were used. This excludes especially a lot of high-risk false positive points in advance, like cars or other objects near the street with a similar geometric structure as sidewalks. For the simple reason that, curb points are found by traditional point cloud features, like difference and standard deviation of height as well as curvature in a certain neighbourhood. Perpendicularity to the street is used too, whereby information about the road was obtained by using additional data from OpenStreetMap [Opea]. Then false positives are filtered with a combination of a density based clustering, approximated linearity and parallelism to the road. Based on those sidewalk points, the curb geometry is calculated by extracting the lower and upper edges. The upper sidewalk surface is

then computed by fitting planes into terrain points above the mean height of those edges and finally the street polygon is composed of the vertices of the lower edges.

The semantic segmentation as well as detection of sidewalk surfaces are quite general approaches. Each point cloud can be stored in an octree and then segmented with a 3D CNN. Once a semantic segmentation is performed, a filtered version of the point cloud can be used for various tasks and not only for detecting curbs. With this potential, it is worthwhile to improve the semantic segmentation. Especially, those corner cases, in which a data sample consists of, for example, mostly car and also some terrain points, but due to the method used in this thesis, all points within this data sample are classified as "cars", which leads to wrong classified terrain points. This might not cause many misclassified points in a single data sample, but accumulated to all samples of the point cloud, the classification error lies in a considerable range. Such cases could be probably eliminated with a hierarchical classification approach or a segmentation network, which is able to work directly on the points, like the U-Net from Ronneberger et al. [RFB15], which applies class labels directly to the pixels of an image. Therefore, such a segmentation network will be implemented and tested in the future to improve the semantic segmentation of point clouds. Another possible next step, according the semantic segmentation, will be to use already detected curb points as well as trained 3D CNN and apply transfer learning to add a new "curb" class to the network. Therefore, the curb points could be automatically detected within the semantic segmentation step.

By looking at the calculation of the curb points and final reconstructed geometry, every step could and will be improved in the future. Beginning with testing and applying more point cloud features, enhancing the false positive filtering and automatically compute the degree of the function according to the bending of the road. In addition, a deeper look will be taken on how to fill even bigger gaps in the point cloud accurately, than the ones caused by parking cars. Furthermore, the algorithm is currently just able to create geometry parallel to the road, but sometimes parts of the curb are not parallel. The calculation of those parts will also be addressed in the future. Besides that, the sidewalk points are calculated with standard point cloud features and which can be simply adapted to other similar cases by changing some thresholds, like for example detection of railway platforms, low walls, fences etcetera. Geometrically seen are those examples just very large curbs and by extending the thresholds of the height feature they could be detected too.

In conclusion, it can therefore be said, that the development of a proof-of-concept prototype was successful and showed a lot of potential. Not only the semantic segmentation with a 3D CNN works well after this first phase of development, but also the detection of curb points as well as reconstructing the final geometry. We will therefor continue to work on these two research questions to improve them further.

APPENDIX A

Hyperparameter Tuning

This appendix shows the exact values of the hyperparameter tuning of Section 5.2 on the training, validation and test set. The *Difference* values in the tables calculates the difference in the results for the according test sets. Each result of a test is the average of ten runs.

Amount of feature learning components for the middle cube:

The amount of the feature learning components for the middle cube with a grid size of 16 were explored first. Note, that the network was not trained with any neighbourhood information at this point. The results are shown in Table A.1.

Amount of feature learning components for the side cubes:

One feature learning component for the middle cube with grid size 16 was used. The difference to the network without any neighbourhood information was calculated and the neighbouring cells had a grid size of 8. The results are listed in Table A.2 and there it can be seen, that the additional usage of the neighbouring cells gives a great increase in classification accuracy of the network.

Amount Components	*Training Set*	*Validation Set*	*Test Set*	*Difference*
1	**80.76**	**76.86**	**80.64**	
2	80.92	76.51	79.78	-0.86
3	80.42	76.01	79.37	-1.26

Table A.1: Impact of the amount of feature learning components in the middle cube.

Amount Components	*Training Set*	*Validation Set*	*Test Set*	*Difference*
0	80.76	76.86	80.64	
1	**89.42**	**84.13**	**87.11**	**+6.47**
2	89.02	83.11	86.02	+5.38

Table A.2: Impact of the amount of feature learning components for the side cubes.

Occupancy Values	*Training Set*	*Validation Set*	*Test Set*	*Difference*
binary	**89.42**	**84.13**	**87.11**	
total	82.89	80.26	83.36	-3.75
fraction	37.78	37.22	33.73	-55.38

Table A.3: Comparison of different occupancy values.

Additional Features	*Training Set*	*Validation Set*	*Test Set*	*Difference*
none	89.42	84.13	87.11	
cube center height	91.38	85.93	88.59	+1.49
average point height	91.31	85.86	88.78	+1.68
average point normal	91.09	84.65	87.18	+0.07
average point colour	**92.34**	**87.88**	**89.47**	**+2.37**

Table A.4: Impact of different additional features.

Occupancy values:

As Section 5.2 states, three different occupancy values were tested, but as discussed there, the ratio of the amount of points in a cell to the total amount of points in the cube does not work well. In fact, it worked so badly that the training was terminated early, thus the results from the fraction are just from a single run. The results of the different occupancy values are shown in Table A.3.

Additional features:

The tested additional features are the height of the center of the middle cube as well as average height, average normal vector and average colour of all points inside a cube. The results are listed in Table A.4.

Middle Cube	*Side Cubes*	*Training Set*	*Validation Set*	*Test Set*
32	32	91.29	84.60	87.33
64	32	90.93	84.84	87.57
64	64	90.86	84.72	87.87
128	**64**	**91.05**	**84.93**	**87.94**
128	128	90.66	84.73	87.58

Table A.5: Comparison of amount of neurons in middle and side cubes.

Batch Size	*Training Set*	*Validation Set*	*Test Set*
256	90.77	84.90	87.62
128	**91.05**	**84.93**	**87.94**
64	90.72	84.89	87.68

Table A.6: Comparison of different batch sizes.

Amount of neurons in a feature learning component:

The amount of neurons in a feature learning component for the middle together with side cubes were explored and the results are shown in Table A.5.

Batch size:

Furthermore, three different batch sizes were tested and the results are shown in Table A.6.

Dropout layer and amount of neurons in fully connected layer:

The dropout rate for the input, hidden and output layer as well as the amount of neurons in the fully connected layer of the classification part of the network were explored. Note, that the amount of neurons was doubled for the second fully connected layer. Therefore, an amount of neurons of 1024 is 1024 for the first and 2048 for the second layer as well as the amount of 2048 is 2048 for the first and 4096 for the second fully connected layer in the network. The results of the compared values are listed in Table A.7.

Dropout Rate			*Amount*	*Training*	*Validation*	*Test Set*
Input L.	*Hidden L.*	*Output L.*	*Neurons*	*Set*	*Set*	
-	0.2	-	1024	89.73	84.13	86.85
-	0.5	-	1024	89.63	84.37	87.68
-	0.8	-	1024	88.29	83.49	86.95
0.2	0.5	-	1024	90.60	84.75	87.66
0.5	0.5	-	1024	90.41	84.40	87.30
0.8	0.5	-	1024	83.05	80.37	84.48
-	0.2	-	2048	89.66	84.15	87.18
-	0.5	-	2048	89.68	84.35	87.64
-	0.8	-	2048	88.45	83.69	87.48
0.2	**0.5**	**-**	**2048**	**90.86**	**84.72**	**87.87**
0.5	0.5	-	2048	90.73	84.55	87.24
0.8	0.5	-	2048	81.88	79.44	83.77
0.2	0.5	0.2	2048	90.58	84.66	87.29
0.2	0.5	0.5	2048	90.68	84.61	87.31
0.2	0.5	0.8	2048	89.91	84.36	87.48

Table A.7: Comparison of different dropout rates in different layers as well as amount of neurons used in those layers.

Side Cubes	*Training Set*	*Validation Set*	*Test Set*	*Overfitting*	*Difference*
without	91.55	82.33	81.03	10.52	
0.1	66.73	65.98	64.82	1.91	-16.21
0.01	76.13	74.50	73.51	2.62	-7.52
0.001	84.02	78.98	78.47	5.55	-2.56
0.0001	**88.32**	**81.34**	**79.98**	**8.34**	**-1.05**

Table A.8: Impact of L2-regularization factors on overfitting and accuracy.

L2-Regularization:

To prevent overfitting and to achieve a good generalization of the network L2-regularization is commonly used. The impact of the regularization with different factors on overfitting as well as accuracy of the test set were explored. The results are listed in Table A.8, the *Overfitting* column shows the difference between the training and the test set of the same row, and the *Difference* column shows the difference between the test set accuracies without any regularization to the used parameters.
Note, that from this point on the tests were performed on a different assembly of the dataset than the tests above. Because, the size and composition of the training, validation and test set was updated in the course of the evaluation process.

Amount Components	*Training Set*	*Validation Set*	*Test Set*	*Difference*
1	75.05	72.64	72.00	+0.69
2	**80.25**	**74.64**	**73.87**	**+2.56**
3	80.29	74.56	73.71	+2.40
4	80.48	74.38	73.55	+2.24
network using middle grid size	75.94	71.54	71.31	

Table A.9: Comparison of amount of feature learning components for the middle cell of the network using big grid size.

First C.	*Second C.*	*Training Set*	*Validation Set*	*Test Set*	*Difference*
64	64	80.25	74.64	73.87	+2.56
64	**128**	**79.75**	**74.77**	**73.97**	**+2.66**
128	128	79.47	74.61	73.76	+2.45
network using middle grid size		75.94	71.54	71.31	

Table A.10: Comparison of amount of neurons for the two feature learning components of the middle cell in the network using big grid size.

Number of feature learning components for the middle cube of the network using big grid size:

Since in the network using big grid size, the grid size of the middle cube is as twice as big as in the network using middle grid size, the amount of feature learning components for this grid size were explored. The amount of neurons in each feature learning component were 64. In order to have a comparison not only among the amount of components themselves, the *Difference* between the accuracies of the test sets from this network to the network using middle grid size was formed. Note, that the networks in this tests where solely trained on the middle cube and the network using middle grid size consisted of a single feature learning component with 128 neurons. The results are compared in Table A.9.

Number of neurons for the middle cube of the network using big grid size:

The previous tests showed that for the network using big grid size two feature learning components are working best. Therefore, the number of neurons for each component were explored and not only compared with themselves, but also the *Difference* in accuracy of the test set to the network using middle grid size was calculated. Note, that these tests also used networks just trained on the middle cube.

APPENDIX B

Network Architectures

This appendix visualizes the architecture of the networks using small and middle grid sizes of Section 5.3 and 5.4 respectively.

B.1 Network Using Small Grid Size

The network using small grid size takes as input a middle cube with grid size 16 and side cubes with a grid size of eight. The part of the network processing the middle cube consists of one feature learning component with 128 neurons and the side cubes are processed with one feature learning component, but with 64 neurons. The flattened output of those components are then concatenated together with the feature vector of the average point colour in a data sample. The resulting feature vector is then fed into the classification part of the network, which is composed of a dropout layer, fully connected layer with 2048 neurons, again a dropout as well as a fully connected layer with 4096 neurons and finally a softmax layer. The architecture of this network is visualized in Figure B.1.

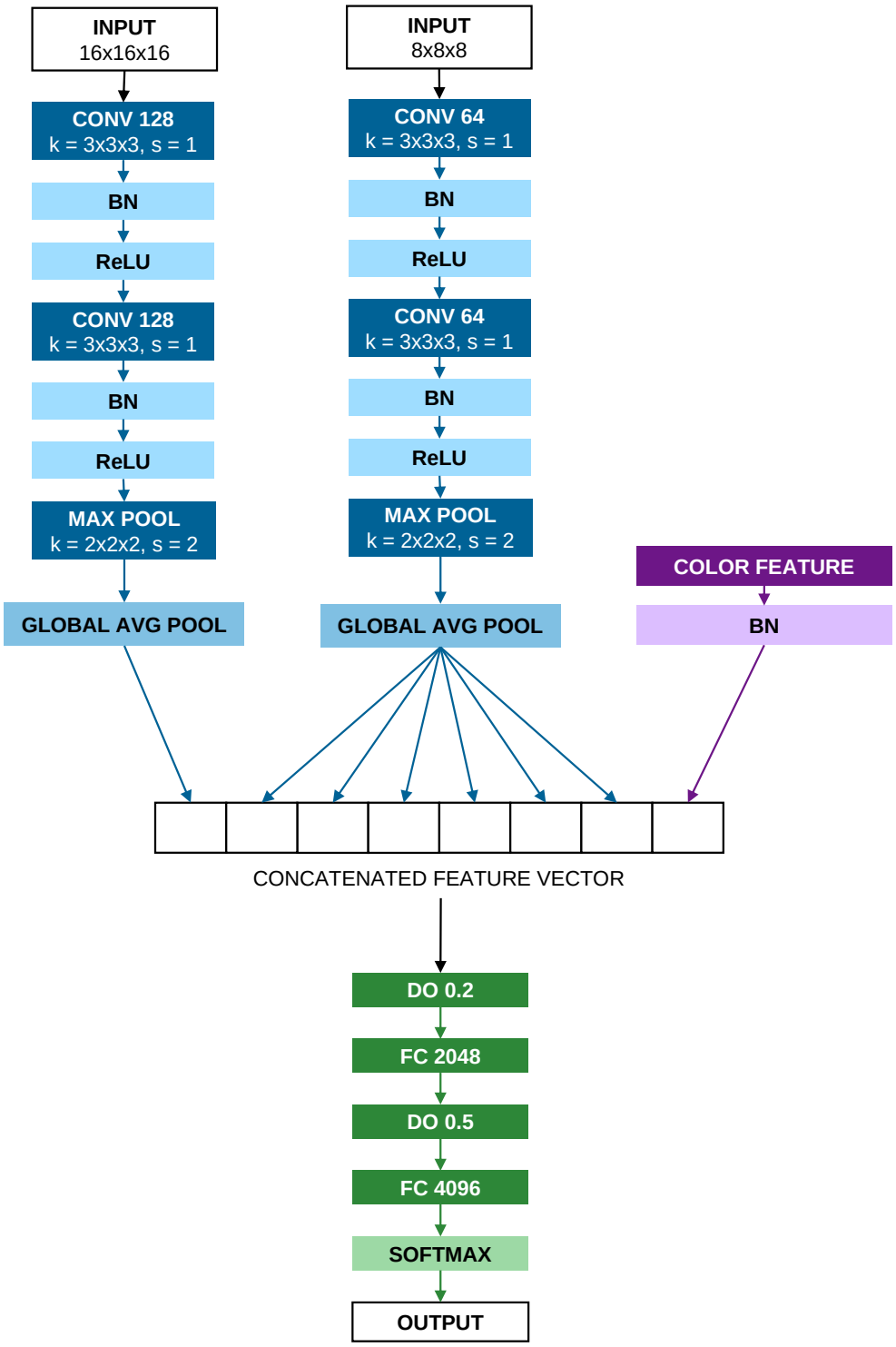


Figure B.1: Architecture of the network using small grid size.

B.2 Network Using Middle Grid Size

The network using middle grid size takes as input a middle cube as well as side cubes with a grid size of 16. The part of the network processing the middle cube consists of one feature learning component with 128 neurons too. The side cubes are also processed with one feature learning component with 64 neurons. The flatten output of those components are then again concatenated together with the colour feature. The classification part of this network consists of two dropout followed by a fully connected layer each followed by a fully connected layer too, where the fully connected layer have 1027 and 2048 neurons respectively. The last layer of this network is again a softmax layer.

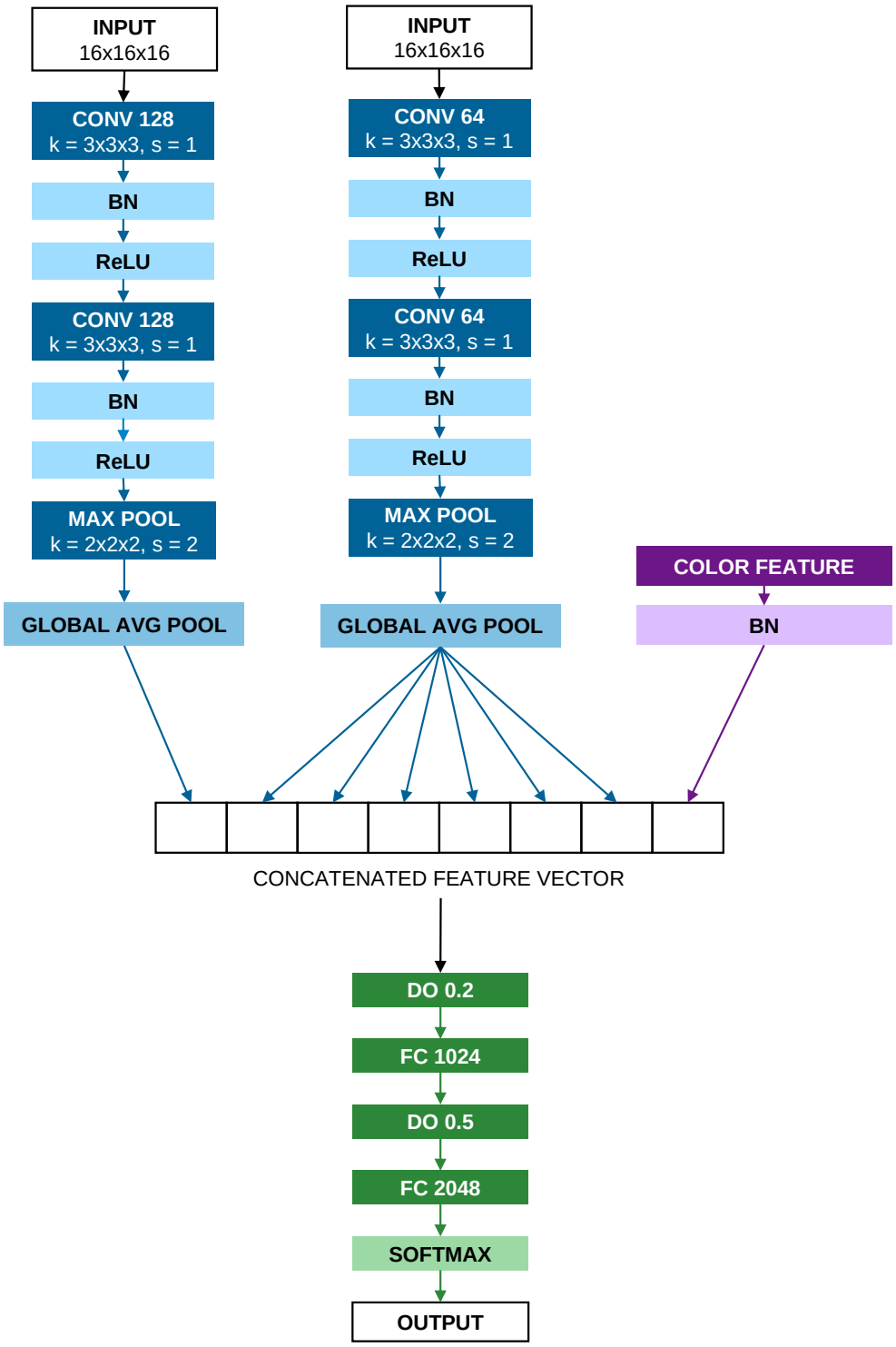


Figure B.2: Architecture of the network using middle grid size.

List of Figures

List of Tables

Bibliography

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[Aar] Aardvark. Aardvark platform. `https://github.com/aardvark-platform`. (Accessed on 04/10/2019).

[BFOS84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[BGSA18] Alexandre Boulch, Joris Guerry, Bertrand Le Saux, and Nicolas Audebert. Snapnet: 3d point cloud semantic labeling with 2d deep segmentation networks. *Computers & Graphics*, 71:189 – 198, 2018.

[BHLLR14] Aharon Bar Hillel, Ronen Lerner, Dan Levi, and Guy Raz. Recent progress in road and lane detection: a survey. *Machine Vision and Applications*, 25(3):727–745, 2014.

[Bis06] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[BKC17] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017.

[BLB+13] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre

Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[Bou19] Alexandre Boulch. Generalizing Discrete Convolutions for Unstructured Point Clouds. In *Eurographics Workshop on 3D Object Retrieval*. The Eurographics Association, 2019.

[Bre00] Claus Brenner. Towards fully automatic generation of city models. *International Archives of Photogrammetry and Remote Sensing*, 33(B3/1; PART 3):84–92, 2000.

[Bre01] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[C+15] François Chollet et al. Keras. `https://keras.io`, 2015. (Accessed on 10/10/2019).

[Cau47] Augustin Cauchy. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

[CKY97] Sunglok Choi, Taemin Kim, and Wonpil Yu. Performance evaluation of ransac family. *Journal of Computer Vision*, 24(3):271–300, 1997.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[DdFG01] Amaud Doucet, Nando de Freitas, and Neil Gordon. *Sequential Monte Carlo methods in practice*. Springer Science & Business Media, 2001.

[EKHL17] Francis Engelmann, Theodora Kontogianni, Alexander Hermans, and Bastian Leibe. Exploring spatial context for 3d semantic segmentation of point clouds. In *The IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct 2017.

[EKS+96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[FILS14] C. Fernández, R. Izquierdo, D. F. Llorca, and M. A. Sotelo. Road curb and lanes detection for autonomous driving on urban scenarios. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1964–1969, 2014.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[GBS14] Leonardo Gomes, Olga Regina Pereira Bellon, and Luciano Silva. 3d reconstruction methods for digital preservation of cultural heritage: A survey. *Pattern Recognition Letters*, 50:3 – 14, 2014. Depth Image Analysis.

[GGG+16] A. Garcia-Garcia, F. Gomez-Donoso, J. Garcia-Rodriguez, S. Orts-Escolano, M. Cazorla, and J. Azorin-Lopez. Pointnet: A 3d convolutional neural network for real-time object class recognition. In *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016.

[HKLS12] J. Han, D. Kim, M. Lee, and M. Sunwoo. Enhanced road boundary and obstacle detection using a downward-looking lidar sensor. *IEEE Transactions on Vehicular Technology*, 61(3):971–985, 2012.

[HOW14] A. Y. Hata, F. S. Osorio, and D. F. Wolf. Robust curb detection and vehicle localization in urban environments. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 1257–1262, 2014.

[HSL+17] Timo Hackel, N. Savinov, L. Ladicky, Jan D. Wegner, K. Schindler, and M. Pollefeys. Semantic3d.net: A new large-scale point cloud classification benchmark. In *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, volume IV-1-W1, pages 91–98, 2017.

[HWS16] T. Hackel, J. D. Wegner, and K. Schindler. Fast semantic segmentation of 3d point clouds with strongly varying density. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, III-3:177–184, 2016.

[IKH+11] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 559–568, 2011.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456. PMLR, 07–09 Jul 2015.

[JRPWM10] Matthew Johnson-Roberson, Oscar Pizarro, Stefan B. Williams, and Ian Mahon. Generation and visualization of large-scale three-dimensional reconstructions from underwater robotic surveys. *Journal of Field Robotics*, 27(1):21–51, 2010.

[JS16] Jing Huang and Suya You. Point cloud labeling using 3d convolutional neural network. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2670–2675, 2016.

[JXYY13] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, 2013.

[Kal60] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KK11] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. *Advances in neural information processing systems*, 24:109–117, 2011.

[KL17] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *The IEEE International Conference on Computer Vision (ICCV)*, 2017.

[KMLM13] Pankaj Kumar, Conor P. McElhinney, Paul Lewis, and Timothy McCarthy. An automated algorithm for extracting road edges from terrestrial mobile lidar data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 85:44 – 55, 2013.

[Kö] Stadt Köln. Zahlen und statistik. `https://www.stadt-koeln.de/politik-und-verwaltung/statistik/?schriftgroesse=normal`. (Accessed on 24/04/2020).

[LDT+17] Felix Järemo Lawin, Martin Danelljan, Patrik Tosteberg, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. Deep projective 3d semantic segmentation. In Michael Felsberg, Anders Heyden, and Norbert Krüger, editors, *Computer Analysis of Images and Patterns*, pages 95–107. Springer International Publishing, 2017.

[LMP01] John Lafferty, Andrew McCallum, and Fernando C.N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning 2001 (ICML 2001)*, pages 282–289, 2001.

[Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov 2004.

[LS18] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[MCB+14] Michela Mortara, Chiara Eva Catalano, Francesco Bellotti, Giusy Fiucci, Minica Houry-Panchetti, and Panagiotis Petridis. Learning cultural heritage by serious games. *Journal of Cultural Heritage*, 15(3):318 – 325, 2014.

[MS15] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928, 2015.

[MWA+13] Przemyslaw Musialski, Peter Wonka, Daniel G. Aliaga, Michael Wimmer, Luc van Gool, and Werner Purgathofer. A Survey of Urban Reconstruction. *Computer Graphics Forum*, 32(6):146–177, 2013.

[Net] NetTopologySuite. Projnet4geoapi. `https://github.com/NetTopologySuite/ProjNet4GeoAPI`. (Accessed on 18/10/2019).

[Opea] OpenStreetMap. Openstreetmap. `https://www.openstreetmap.org`. (Accessed on17/10/2019).

[Opeb] OpenStreetMap. Openstreetmap wiki. `https://wiki.openstreetmap.org/wiki/Main_Page`. (Accessed on 17/10/2019).

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Qia99] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145 – 151, 1999.

[QSMG17] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[QSN+16] Charles R. 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)*, 2016.

[QYSG17] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems 30*, pages 5099–5108. Curran Associates, Inc., 2017.

[RCGCOA15] Borja Rodríguez-Cuenca, Silverio García-Cortés, Celestino Ordóñez, and Maria C. Alonso. An approach to detect and delineate street curbs from mls 3d point cloud data. *Automation in Construction*, 51:103 – 112, 2015.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241. Springer International Publishing, 2015.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[Ros58] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[ROUG17] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[Sah] Sumit Saha. A comprehensive guide to convolutional neural networks — the eli5 way. `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53`. (Accessed on 06/12/2019).

[Sem] Semantic3D. Semantic3d - results. `http://semantic3d.net/view_results.php?chl=1`. (Accessed on 04/07/2020).

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[SKMW17] Michael Schwärzler, Lisa-Maria Kellner, Stefan Maierhofer, and Michael Wimmer. Sketch-based guided modeling of 3d buildings from oriented photos. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 9:1–9:8. ACM, February 2017.

[SMKLM15] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *The IEEE International Conference on Computer Vision (ICCV)*, 2015.

[SSA20] Siddharth Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology*, 04:310–316, 05 2020.

[SZX+19] P. Sun, X. Zhao, Z. Xu, R. Wang, and H. Min. A 3d lidar data-based dedicated road boundary detection algorithm for autonomous vehicles. *IEEE Access*, 7:29623–29638, 2019.

[TCA+17] L. Tchapmi, C. Choy, I. Armeni, J. Gwak, and S. Savarese. Segcloud: Semantic segmentation of 3d point clouds. In *2017 International Conference on 3D Vision (3DV)*, pages 537–547, 2017.

[TGDM18] H. Thomas, F. Goulette, J. Deschaud, and B. Marcotegui. Semantic classification of 3d point clouds with multiscale spherical neighborhoods. In *2018 International Conference on 3D Vision (3DV)*, pages 390–398, Sep. 2018.

[TQD+19] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, Francois Goulette, and Leonidas J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[Vis] Visdom. Visdom - combining simulation and visualization. `http://visdom.at/`. (Accessed on 05/12/2019).

[WBG+12] M.J. Westoby, J. Brasington, N.F. Glasser, M.J. Hambrey, and J.M. Reynolds. 'structure-from-motion' photogrammetry: A low-cost, effective tool for geoscience applications. *Geomorphology*, 179:300 – 314, 2012.

[WJHM15] Martin Weinmann, Boris Jutzi, Stefan Hinz, and Clément Mallet. Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers. *ISPRS Journal of Photogrammetry and Remote Sensing*, 105:286 – 304, 2015.

[WLG+17] 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, 2017.

[WLW+15] H. Wang, H. Luo, C. Wen, J. Cheng, P. Li, Y. Chen, C. Wang, and J. Li. Road boundaries detection based on local normal saliency from mobile laser scanning data. *IEEE Geoscience and Remote Sensing Letters*, 12(10):2085–2089, 2015.

[WSK+15] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[WUH+15] M. Weinmann, S. Urban, S. Hinz, B. Jutzi, and C. Mallet. Distinctive 2d and 3d features for automated large-scale scene analysis in urban areas. *Computers & Graphics*, 49:47 – 57, 2015.

[WWHY19] G. Wang, J. Wu, R. He, and S. Yang. A point cloud-based robust road curb detection and tracking method. *IEEE Access*, 7:24611–24625, 2019.

[Zha10] W. Zhang. Lidar-based road and road-edge detection. In *2010 IEEE Intelligent Vehicles Symposium*, pages 845–848, 2010.

[ZTF+11] Matthew D Zeiler, Graham W Taylor, Rob Fergus, et al. Adaptive deconvolutional networks for mid and high level feature learning. In *ICCV*, volume 1, page 6, 2011.

[ZWW+15] Y. Zhang, J. Wang, X. Wang, C. Li, and L. Wang. A real-time curb detection and tracking method for ugvs by using a 3d-lidar sensor. In *2015 IEEE Conference on Control Applications (CCA)*, pages 1020–1025, 2015.

[ZWWD18] Y. Zhang, J. Wang, X. Wang, and J. M. Dolan. Road-segmentation-based curb detection method for self-driving via a 3d-lidar sensor. *IEEE Transactions on Intelligent Transportation Systems*, 19(12):3981–3991, 2018.

[ZY12] G. Zhao and J. Yuan. Curb detection and tracking using 3d-lidar scanner. In *2012 19th IEEE International Conference on Image Processing*, pages 437–440, 2012.