Large-scale Machine Learning for Point Cloud Processing

Author(s):
Hackel, Timo

Publication Date:
2018

Permanent Link:

https://doi.org/10.3929/ethz-b-000264691 →

DISS. ETH NO. 25105

LARGE-SCALE MACHINE LEARNING FOR POINT CLOUD PROCESSING

A dissertation submitted to attain the degree of

DOCTOR OF SCIENCES OF ETH ZURICH
(DR. SC. ETH ZURICH)

presented by

TIMO HACKEL
DIPL. ING. KARLSRUHE INSTITUTE OF TECHNOLOGY

BORN ON 21.03.1988
CITIZEN OF GERMANY

accepted on the recommendation of

Prof. Dr. K. Schindler, examiner
Prof. Dr. A. Nüchter, co-examiner
Dr. J. D. Wegner

2018

IGP Mitteilungen Nr. 120
Large-scale Machine Learning for Point Cloud Processing
Timo Hackel

Published by:
Institute of Geodesy and Photogrammetry
ETH ZURICH
CH-8093 Zurich

ISBN 978-3-03837-006-2

Abstract

Machine learning has made phenomenal progress in the past decades. This work has a focus on the challenges and opportunities that come with machine learning in 3D space. We start the thesis with an efficient approach to point-wise semantic classification that can classify point clouds of significant size. The method is able to process the inhomogeneous and unstructured point distribution of typical 3D point clouds that are captured with industrial grade static terrestrial LiDAR or with photogrammetric reconstruction. The processing of large data sets with millions of points only takes a couple of minutes.

For this purpose, we rely on traditional machine learning, where we exploit a computationally efficient neighborhood definition to obtain a rich multi-scale feature set. This approach outperforms a then state-of-the-art baseline method on various benchmark data sets from mobile mapping platforms in both per-point classification accuracy and runtime by a large margin.

We then demonstrate the benefits of machine learning over traditional point cloud processing techniques on the example of 3D contour extraction; A task with a multitude of applications in graphics and mapping, e.g. by deploying contours as intermediate features for structuring point clouds and converting them into a high-quality surface. On large outdoor scenes, we vastly outperform a Canny-style edge detection.

Benchmarking plays a central role in computer vision research by making various approaches comparable. This thesis presents a novel large-scale benchmark test for point cloud classification. With this benchmark test, we publish a data set of over four billion manually labeled points; Large-scale data sets are important for data-hungry machine learning techniques like deep learning. Furthermore, we published three baseline methods: *i)* our traditional machine learning approach; *ii)* a deep learning baseline; and *iii)* an image based semantic classification approach. Our benchmark shows that recent deep learning based approaches significantly outperform traditional machine learning.

However, a problem of neural networks in 3D space remains: the massive demand for computational resources that easily reaches hardware limitations. The sparsity in common 3D data can help to overcome this issue. We introduce a deep learning framework that exploits and preserves sparsity in both the feature maps and the model parameters

of our network to reduce the memory footprint and the runtime of convolutional neural networks.

Our approach provides *i)* a sparse GPU implementation of various network layers as well as *ii)* a filter step that sets an upper bound on the computational requirements by reducing the tendency to decrease sparsity.

Kurzzusammenfassung

Im letzten Jahrzehnt hat maschinelles Lernen enorme Fortschritte erlebt. Diese Arbeit behandelt Herausforderungen sowie Möglichkeiten des maschinellen Lernens im 3D Raum. Wir beschreiben effiziente Ansätze für das punktweise Klassifizieren von Punktwolken enormen Ausmasses. Unser Ansatz ermöglicht das Prozessieren von inhomogenen und unstrukturierten Punktverteilungen, wie sie von industriellen, statischen Terrestrischen Laserscannern oder von photogrammetrischen Rekonstruktion generiert werden. Das Prozessieren von grossen Datensätzen mit Millionen von Punkten dauert nur wenige Minuten.

Zu diesem Zweck verwenden wir traditionelles maschinelles Lernen, welches auf eine effiziente Definition von Nachbarschaften zurückgreift um eine reiche Menge an Merkmalen zu erzeugen. Unser Ansatz übertrifft unsere Vergleichsmethode auf vielen Benchmarktests sowohl in die Klassifikationsqualität als auch in der Laufzeit.

Anschliessend demonstrieren wir die Vorzüge von maschinellem Lernen über traditionellen Prozessierungsmethoden für Punktwolken am Beispiel Konturextraktion; Eine Aufgabe mit vielen Anwendungen in der Computergrafik und im Mapping. Zum Beispiel verwenden viele Algorithmen Konturen als Basismerkmale um Oberflächen mit hoher Güte zu rekonstruieren. Auf grossen, natürlichen Punktwolken übertreffen wir eine Kantenerkennung, welche dem Ansatz von Canny folgt deutlich.

Benchmarktests spielen eine wichtige Rolle für die Forschung im maschinellen Sehen. Diese Arbeit stellt einen neuen Benchmarktest für das Verarbeiten grosser Punktwolken vor. Mit diesem Benchmarktest veröffentlichen wir vier Milliarden handannotierte Punkte; Grosse Datensätze sind wichtig für eine Vielzahl von Ansätzen im Bereich des maschinellen Lernens, einschliesslich Deep Lerning. Zudem publizieren wir drei Ansätze zum Klassifizieren von Punktwolken: *i)* unseren auf traditionellem Lernen basierender Ansatz; *ii)* eine Deep Learning Baselinemethode und *iii)* ein Ansatz der auf 2D Bildverarbeitung basiert. Auf unsrem Benchmarktest zeigen Deep-Learning-Ansätze eine deutlich bessere Leistung als Methoden, die traditionelles maschinelles Lernen nutzen.

Jedoch bleibt ein Problem von Neuronalen Netzen bestehen: Der hohe Bedarf an Rechenleistung und Speicherplatz. Die Spärlichkeit in typischen 3D Daten kann dazu

beitragen dieses Problem zu verringern. Daher führen wir ein Framework für Deep Learning ein, welches Spärlichkeit in Merkmalen sowie im Model ausnutzt und befördert um den Speicherbedarf und die Rechenzeit von Neuronalen Netzen zu senken. Unser Ansatz bietet sowohl *i)* eine GPU Implementierungen von vielen Netzwerklayern als auch *ii)* einen Filterschritt, welcher den Bedarf an Computerressourcen beschränkt indem die Neigung zum Auffüllen von spärlichen Daten reduziert wird.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Ever since the Stone Age, new technologies helped to avoid repetitive, laborsome or physically exhausting work by either creating better tools controlled by human operators or by a complete automatization of tasks. Until the introduction of powerful computers, sensors and actuators, automatization was limited to fairly simple tasks. However, automatization remains difficult even with modern hardware, so that a future where work is merely a hobby to the very ambitious rather than a necessity by far and large remains a utopia. Possibly, the most prominent challenge to automatizing complex and non-repetitive tasks is data processing, where often it is not possible to reach the performance of human operators; even with significant development effort.

This calls for algorithms capable of inferring information from sensor readings while approaching or even outperforming human performance. Traditionally, in computer vision, the performance of algorithms on tasks is measured with benchmark tests. These tests evaluate approaches in a qualitatively comparable manner. Such comparisons have helped to focus and accelerate the research on many important tasks. Benchmark tests cover the essential tasks in computer vision. For instance KITTI [Geiger et al., 2012] has a focus on 3D reconstruction, while MNIST [LeCun et al., 1998] or ImageNet [Russakovsky et al., 2015] provide tests for the inference of object classes in images; A classification problem that assigns object classes to images. The benchmarking of semantic classification in images is approached in the Pascal visual object classes challenge [Everingham et al., 2010], where an object class is assigned to each individual pixel. In the past decade, hand-crafted features, such as SIFT [Lowe, 1999], in combination with discriminative learning, such as Random Forests [Breiman, 2001], showed good performance on these tests. On large training datasets and with a significant amount of computational resources, purely data-driven approaches tend to achieve

more impressive results. Since the benchmark submission of [Krizhevsky et al., 2012] on ImageNet, deep learning approaches that are capable of representation learning have quickly become state-of-the-art for most computer vision problems.

For a multitude of problems, plain 2D camera images are not a suitable data representation. One such class of problems covers the measurement of actual distances in physical space, e.g. the distance of a pedestrian to an autonomous car, where 3D data is superior to 2D data[1]. Stereo reconstruction, Shape from Shading or Structure from Motion can be used to reconstruct 3D data from 2D images. Nevertheless, it can be more accurate and robust to measure 3D data directly, e.g. with laser scanners. Especially, when the environment can turn to adverse conditions, such as rain or a lack of illuminations in tunnels or during the night. Data processing in 3D space has followed a similar development as 2D image processing. However, data processing in 3D space is more challenging than 2D image processing. On the one hand, 3D data has one more dimension than 2D data and, hence, has a larger memory footprint. On the other hand, the neighborhood in 3D space is more complex than an 8-neighborhood in 2D images. For 3D neighborhoods, a multitude of definitions exists, for instance, memory hungry dense voxel grids or computationally more demanding kd-trees [Friedman et al., 1977], OcTrees [Elseberg et al., 2013] or sparse voxel grids. For this reason point cloud processing typically lags behind image processing on an algorithmic level; Often it is challenging to perform seemingly simple tasks in 3D space with good accuracy and within a reasonable amount of time. One such seemingly simple task is 3D contour extraction that turns out to be surprisingly difficult to solve. More complex tasks in 3D space, such as 3D semantic segmentation, have to deal with significant constraints regarding computational resources. Nevertheless, even for processing a 3D reconstruction generated from multiple 2D images, it can be more efficient to perform these tasks only once directly in 3D space rather than on each 2D image [Riemenschneider et al., 2014].

Even though deep learning shows outstanding performance on many 2D problems, it is extremely challenging in 3D space. Deep learning not only needs large datasets for training but is already resource hungry in 2D space. For this purpose the data processing of typical convolutional neural networks (CNN) in 3D space [Wu et al., 2015] [Maturana and Scherer, 2015] is limited to voxel grids with small resolutions to avoid large memory footprints due to their cubic memory growth. The use of measurement properties can help to reduce the need for computational resources: The line-of-sight measurements of typical 3D point clouds are sparse when represented as voxel grids. Sparse data representations, such as Compressed Sparse Blocks, can help to exploit this sparsity in the convolutional network layers, as shown in [Parashar et al., 2017] with similarities to [Engelcke et al., 2017] and [Park et al., 2017]. An-

[1] Please note, that distance relations can also be guessed from 2D images, e.g. [Richter and Roth, 2015], yet these guesses are prone to errors.

other approach is to perform space partitioning, e.g. with OcTrees [Tatarchenko et al., 2017] [Häne et al., 2017] [Riegler et al., 2017]. With such sparse approaches, it is possible to reduce the 3D data processing problem to the processing of the approximated underlying 2D surface. However, the exploitation of sparsity induces computational overhead.

1.2 Research Goals

数据裤

The backbone of this thesis is a large database of 3D point clouds, which has been recorded with industrial-grade terrestrial laser scanners. To obtain ground truth, large parts of this database were semantically annotated by humans workforce. For annotation, meaningful object classes, such as *Building*, *Car* or *Natural Ground* were defined. Subsequently, each point was labeled one of these classes. Annotation by humans is not only costly but also several orders of magnitude more time consuming than the capturing of such point clouds. For a multitude of technical applications, it is a necessity to obtain such an annotation in near-real time. Hence, it is fundamental to infer object classes automatically. The overriding research question of this thesis is: *What is the best way to infer information from large 3D point clouds?* We approach this issue from three sides.

1. **Traditional Machine Learning.** The processing of large 3D point clouds requires a significant amount of computational resources. Even though deep learning has proven to be successful on 2D data it has shown a significantly higher need for computational resources than traditional discriminative learning, e.g. with random forests. Hence, the first goal of this thesis is to perform 3D semantic segmentation with traditional machine learning techniques and demonstrate its benefits on an elemental technical application; 3D contour extraction. On application side, Contour Extraction is an obvious choice as it turned out to be a surprisingly difficult task in 3D. This raises the question if semantic information can help to improve contour extraction? Vice versa contour extraction has synergies with the semantic segmentation: can contours be used as a feature for semantic classification?

   这里主要是利用传统的机器学习方法去解决3D轮廓提取的问题还提出了，3D轮廓提取是否能解决语义分类的问题

   随机森林，是一种分类器

   被证明的固定句式

   接下来应该会主要研究这方面的课题

2. **Benchmarking.** Performance evaluation and comparison of different algorithms is essential to rapid technological advancement. Ground truth annotation commonly allows a comparison of classification algorithms with human annotation. However, the ability to identify the best-suited algorithm by deploying ground truth annotated data is limited to a comparison among those algorithms, that are tested on the same data. The second goal of this thesis is to establish standardized train and test datasets to ensure that the performance of all algorithms that

follow this standard testing procedure is comparable.

3. **Deep Learning.** Finally, we want to develop inference and learning mechanisms, which exploit and preserve sparsity in data; An essential property for deep learning in large point clouds. Note that such optimized Convolutional Networks are not limited to point cloud processing, but could also be applied to sparse image processing tasks or higher dimensional problems.

All of the above research goals have the same objective: The development of efficient and accurate inference mechanisms for large point clouds.

1.3 Structure and Contribution

This thesis is provided as a *cumulative dissertation* and complies with the ETH Zurich Doctorate Ordinance of 2013. The contributions of this thesis have been successively published across several papers. Chapter 4, 5 and 6 include the key papers.

这里理解为 引进

Approach. In the beginning of this thesis, we introduce our approach to semantic classification as well as contour detection (Chapter 4, [Hackel et al., 2017b]). Our semantic classification is based on a traditional machine learning approach that has a small memory footprint, fast runtime and is used to generate low-level evidence for contour extraction. Subsequently, the points are connected into a graph to identify long-range interactions with the help of a second discriminative classification stage.

For the task of semantic segmentation, we created a ground truth annotated dataset of significant size. This dataset was turned into a benchmark test (Chapter 5, [Hackel et al., 2018a]), where we provide not only data but also three baseline methods covering a broad range of different classification approaches. Furthermore, the test set labels are held back, and we provide an automatic evaluation system to make submissions to the benchmark as convenient as possible.

Recent submissions to the benchmark test show that deep learning is hitherto the most promising approach to reach high classification accuracies. However, neural networks have large requirements on computational resources. The last significant contribution of this thesis (Chapter 6, [Hackel et al., 2018b]) introduces techniques to exploit the sparseness in 3D data to reduce memory footprint and runtime of a convolution network while guaranteeing that available computational resources are not exceeded.

These three stages cover all research questions from Chapter 1.2 and contribute to the rapid development in the field of computer vision.

Contribution in Chapter 4. We contribute to point cloud processing with approaches and applications that deploy traditional machine learning:

1. We introduce a novel method for 3D semantic segmentation to efficiently extract a rich set of multi-scale features in large point clouds by deploying a pyramid of successively coarser sub-sampled search trees. Subsequently, a discriminative learner is trained to infer information from this feature set. Similar to [Weinmann et al., 2015a] we deploy a Random Forest for discriminative learning.

2. After public announcement of the benchmark test, our submitted baseline method remained top in the leaderboard for more than half a year. Please note that additional experiments in [Hackel et al., 2016b] outperform our baseline [Weinmann et al., 2015a] both in runtime and overall accuracy on the *Rue-Madame* and *Rue-Casette* datasets.

3. We introduce a 3D contour extraction algorithm which uses semantic segmentation to generate lowlevel evidence in the form of class conditional probabilities. This semantic classification helps to disambiguate the definition of *contours* by providing a definition in form of training data. To be more robust to noise, an over-complete contour graph is constructed, and false connections are pruned by deploying a second discriminative classification stage.

4. The evaluation successfully demonstrates the advantages of our semantically informed contour extraction on pointwise labeled point clouds *(contour, no contour)*, where we significantly outperform a 3D canny like baseline method.

Contribution in Chapter 5. Furthermore, we contribute to benchmarking of 3D classification tasks as described below:

1. We provide a large dataset of over four billion ground truth annotated points for eight classes; much larger than any previously published dataset. The points are hand-labeled to ensure high label quality by avoiding a bias towards segmentation algorithms. Such a large set of points is a necessity in times of representation learning, where it is unclear if improvements to an algorithm couldn't be learned by just providing more training data.

2. We provide baseline methods: The semantic segmentation algorithm from Chapter 4, a traditional machine learning method that operates on cube map images and a deep learning method that deploys dense voxel grids as data representation. These three methods cover a wide range of possible approaches to 3D classification.

3. An automated submission system allows to conveniently submit results to our benchmark server. This server evaluates submissions against a held back test label set so that a fair comparison of all submissions is possible. So far more than 100 unique users have registered to our benchmark while the results of 14 approaches are publicly available on the leaderboard.

Contribution in Chapter 6. Moreover, this thesis contributes to the field of deep learning with a focus on the processing of large and sparse datasets:

1. We introduce a GPU implementation of a convolutional layer for the processing of sparse data, which exploits both sparse feature maps and sparse model parameters to have a small memory footprint and a fast runtime.
2. To the best of our knowledge, we are the first to introduce upper bounds to limit the requirements of computational resources. Furthermore, our resource management is complemented by sparsity encouraging regularization techniques and parameter pruning.
3. Experiments show that we are significantly faster in runtime while having a significantly lower memory footprint than dense *Tensorflow* for very large and very sparse tensors.

Even though we are constrained by copyright due to industrial cooperations, many datasets are published online on our benchmark test [2]. Furthermore, the source code for our neural network is publicly available on github[3].

1.4 Relevance to Science and Economy

Science

Classification algorithms and large-scale machine learning have become an integral part of 3D vision that shows synergies with a multitude of applications. On the one hand, traditional hand-crafted approaches can be improved by deploying semantic cues, e.g. 3D reconstruction [Häne et al., 2013] or our work on contour detection. On the other hand, semantic classification can help to design end-to-end learning approaches that can learn how to best solve a task, e.g. object tracking [Ondruska et al., 2016] or stereo reconstruction [Zbontar and LeCun, 2016]. For these tasks, efficiency plays an important role. The work presented contributes to the topic of machine learning for 3D semantic segmentation and derived applications by introducing novel efficient approaches to 3D data processing. Our contributions have a focus on efficient feature extraction in traditional machine learning and on the exploitation of sparsity, e.g. by avoiding unnecessary computations. Furthermore, our benchmark test helps to analyze and compare point cloud classification techniques. These contributions are published in the following papers:

[2] `www.semantic3d.net`
[3] `https://github.com/TimoHackel/ILA-SCNN`

1. *Hackel, Timo, et al. "Contour detection in unstructured 3d point clouds." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.*

2. *Hackel, Timo, et al. "Fast semantic segmentation of 3D point clouds with strongly varying density." ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences 3.3 (2016).*

3. *Hackel, Timo, et al. "Semantic3D.net: A new large-scale point cloud classification benchmark." ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences 4 (2017).*

4. *Hackel, Timo, et al. "Joint classification and contour extraction of large 3D point clouds." ISPRS Journal of Photogrammetry and Remote Sensing 130 (2017): 231-245.*

5. *Hackel, Timo, et al. "Large-scale supervised learning for 3D point cloud labeling: Semantic3D.net." accepted in PE&RS Journal*

6. *Hackel, Timo, et al. "Inference, Learning and Attention Mechanisms that Exploit and Preserve Sparsity in Convolutional Networks" submitted.*

These publications tackle the challenging problem of 3D data processing as described in Section 1.3.

Economy

A multitude of technical domains can benefit from semantic segmentation, contour extraction or generic machine learning. Some of the most promising applications are discussed below:

Intelligent vehicles & autonomous robots. For autonomous vehicles, scene understanding plays an integral role to avoid accidents and to plan their behavior, e.g. in heavy traffic. Sensors that capture 3D data can help to determine the distance of obstacles to a car or robot, while semantic cues are a necessity to distinguish between *i)* dangerously moving obstacles, such as other cars, *ii)* slowly moving obstacles, such as pedestrians, which must not be harmed and static objects, such as houses. For this purpose, it is a necessity to infer semantic information in real time with high accuracies. Tasks in robotics typically aim to not only navigate in the world but also to manipulate the world. For instance, the task to "pour milk into a bowl of cornflakes" has an even higher dependency on semantic scene understanding than the task of navigating to a point on a map. The computationally efficient semantic segmentation introduced in [Hackel et al., 2016b] can help to infer semantic classes for such applications.

Surveying. In surveying, one of the primary goals is to capture important information quickly. For instance, the process of scanning a bridge with terrestrial laser scanners could be performed by: *i)* create a coarse scan, *ii)* perform semantic classification in the coarse scan, so that man-made structures are identified, *iii)* scan only the man-made structure with high resolution and accuracy. Such an approach would not only reduce the time of field operations but also compress the data by capturing fine-grained details only when they are needed. Additionally, contour extraction could be used as a lossy data compression, which reduces memory requirements significantly by just storing the most essential points [Hofer et al., 2015]; such a compact representation can be beneficial when working with extraordinary large datasets. Moreover, high-quality semantic information can be deployed to survey construction sites conveniently. In general, the ability to distinguish between man-made structure and vegetation can help to automate many processes in airborne scanning and urban planning.

语义分类的应用之二，可以迅速提取重要的信息，忽略次要信息

Virtual reality & Augmented reality. Recently, a multitude of VR and AR devices and applications have been introduced, including but not limited to Microsoft's Hololens, Facebook's Oculus Rift or Google's ARCore. Typical applications in VR and AR heavily make use of 3D representations, which can be obtained from the real world by deploying 3D-cameras, *e.g.* time-of-flight cameras, or stereo vision. For a good user experience, it is desirable to obtain a 3D reconstruction of high quality. To improve the quality of stereo reconstruction [Häne et al., 2013] proposes to considering semantic cues. To obtain such semantic cues, 3D semantic segmentation is a suitable approach, e.g. [Hackel et al., 2016b]. Another typical problem of 3D reconstruction is smoothed edges in triangle meshes. The effects of smoothing can be reduced by identifying edges directly in point clouds before mesh generation [Hackel et al., 2016a] and deploying these edges in combination with edge-preserving reconstruction methods, e.g. [Bódis-Szomorú et al., 2015]. To augment the world, two questions have to be answered: *i) what* kind of objects should be augmented and *ii) where* is the location of these objects. Apparently, one possible solution to these questions is given by 3D semantic segmentation; where the location of objects is provided in form of 3D points and the object type is inferred as class label.

Chapter 2

Related Work

The source of human thoughts has been a motivation to generations of philosophers and thinkers. As early as the fifth century B.C.E ancient Greek philosophers like Alcmaeon started to search for *"the seat of sensation and understanding"*; a question that still inspires modern neuroscience [Crivellato and Ribatti, 2007]. James J. Gibson, one of the most cited psychologist, refined this question to *"How do we see the world around us?"* by considering the information content of the environment when a person interacts with it [Gibson, 1950, Gibson, 1966]. A subject, which encouraged Hubel and Wiesel to perform their Nobel Prize-winning experiments on the neuron activations in the cat's visual cortex [Hubel and Wiesel, 1959, Hubel and Wiesel, 1962]. One key component of human visual perception is *attention*; the mechanism that manages the limited resources of the human brain and sense organs. Despite being a focus of research [Posner and Petersen, 1990, Lavie et al., 2004, Knudsen, 2007, Baldauf and Desimone, 2014], there still does not exist a widely accepted theory on how human attention is controlled. Nevertheless, multiple projects try to reverse engineer the human brain, and it's learning mechanisms, including Numenta [Hawkins and Ahmad, 2016, Cui et al., 2017] or the Human Brain Project [Amunts et al., 2016]. Due to the lack of a coherent theory of the human brain, such reverse engineering approaches show limited success.

A more appropriate approach to machine learning is to develop coherent learning theory rather than trying to reverse engineer the human brain [Russell et al., 2003]. Among the earliest approaches to machine learning are neural networks. Perceptron, a work by [Rosenblatt, 1957], is considered to be one of the earliest implementations of neural networks and implements a linear classifier. A bottleneck of the original perceptron is the training step, especially, when combining linear classifiers with non-linear activation functions and stacking them together to form a multi-layer perceptron. This bottleneck is addressed by Rumelhart's error propagation [Rumelhart et al., 1985] and LeCun's backpropagation [LeCun et al., 1989], a mechanism to propagate error gradi-

ents through networks of differentiable functions by *i)* deploying chain rule to compute error gradients and *ii)* perform optimization with variants of gradient descent. An essential extension to the multi-layer-perceptron are *shift-invariant* layers [Fukushima, 1980,LeCun et al., 1998]; Such convolutional layers are the basis for most modern neural networks. One problem of gradient descent based optimization is it's convergence to a local minimum. This gave rise to a class of so-called traditional machine learning algorithms, which typically exploit convex optimization, including maximum-margin classifiers, e.g. Support-Vector-Machines [Cortes and Vapnik, 1995], or ensemble methods like boosting [Freund et al., 1996] and random forests [Breiman, 2001]. Until recently, these traditional techniques played a dominant role in machine learning. However, with large datasets and modern computational hardware, neural networks, like Alexnet [Krizhevsky et al., 2012], often show superior performance over traditional machine learning techniques and hand-crafted features.

The contributions of this thesis are in the field of machine learning for point clouds. This chapter relates our main contributions to the literature, including sections on: *i)* Point cloud classification techniques; *ii)* Contour detection in unstructured data; *iii)* Benchmark tests; and *iv)* Efficient deep learning with sparse data.

2.1 Point Cloud Classification

Applications. Traditionally, 3D semantic segmentation plays an important role for autonomous robots, mobile mapping systems and the processing of airborne images as well as point clouds. Early works perform classification in airborne LiDAR data to distinguish between buildings and natural vegetation with a focus on building reconstruction. By converting point clouds to grid data in the form of regular raster heightfields it is possible to apply image processing algorithms to the grid data, such as edge and texture filters [Hug and Wehr, 1997] in combination with maximum likelihood classification [Maas, 1999] or iterative bottom-up classification rules [Haala et al., 1998, Rottensteiner and Briese, 2002]. Another strategy to extract buildings from point clouds is to fit geometric 2D [Schnabel et al., 2007, Pu and Vosselman, 2009] or 3D [Li et al., 2011,Xiao and Furukawa, 2014] shape primitives to the data. However, it turns out that a fixed shape library is insufficient to model most complex natural scenes, e.g. typical airborne data. To reduce the modeling error, [Lafarge and Mallet, 2012] proposes to fill gaps with unstructured triangle meshes that are triangulated from the point cloud. More recent work follows a more generic strategy: *i)* use discriminative learners to infer semantic information for each individual point as low-level evidence in the form of a class label; *ii)* deploy the low-level evidence for high-level modeling [Charaniya et al., 2004, Chehata et al., 2009, Niemeyer et al., 2011, Yao et al., 2011]. This strategy is not limited to building extraction. Rather, it is applicable to generic problems such

可以学习这种表达的方法

这里说最早的航空数据分类问题主要是用来区分建筑和植被的。

as the classification of tree species in forestry [Ørka et al., 2009, Dalponte et al., 2012] or the semantic classification of point clouds from mobile mapping systems.

Applications for 3D semantic classification with mobile mapping systems include road extraction [Boyko and Funkhouser, 2011], street furniture detection [Golovinskiy and Funkhouser, 2009], building identification [Pu et al., 2011], the extraction of trees [Monnier et al., 2012, Weinmann et al., 2017], or disaster damage detection [Vetrivel et al., 2017]. For navigation tasks with autonomous robots, it is desirable to separate obstacles like trees from the ground [Lalonde et al., 2006, Bogoslavskyi and Stachniss, 2017]. Most recent approaches aim to jointly infer all relevant object classes at point level by attaching a class label to individual points [Weinmann et al., 2013, Dohan et al., 2015].

这一段主要阐述了3D语义分类的移动构图方面，道路检测，树的检测，现在研究的方法主要集中在给单独的点进行标签，这一段所提到的文章要多看看

Features. Traditional machine learning depends on hand-crafted features that can discriminate between relevant object classes. Such 3D features encode geometric properties including surface variations, curvature, normal orientation or the height to a ground plane. A multitude of descriptors has been developed that encode such geometric properties, namely spin images [Johnson and Hebert, 1999], fast point feature histograms (FPFH) [Rusu et al., 2009], signatures of histograms [Tombari et al., 2010], *etc.*. These descriptors are computationally demanding since they have originally been designed for sparse keypoints. This limits their usefulness in applications that perform dense, pointwise classification in large point clouds. Alternatively, the NARF operator [Steder et al., 2010, Steder et al., 2011] can be used as computational lightweight descriptor that explicitly models object contours to gain robustness to viewpoint changes. However, the operator deploys range images rather than actual 3D point clouds. A more efficient set of 3D features exploits the eigenvalues and eigenvectors of the covariance tensor of a point's neighborhood [Demantké et al., 2011]; often found in combination with cylindrical features [Monnier et al., 2012] that encode the distribution along the z-axis [Weinmann et al., 2013]. Randomized histograms can help to avoid the analysis of the covariance tensor, as shown in [Blomley et al., 2014] and enrich the discriminative power of the feature set when combined with covariance based features [Blomley et al., 2016].

Spatial smoothing. Random field models are a common strategy to consider correlations between object classes, e.g. by enforcing smoothness or by encouraging frequently neighboring object classes, where the class conditional probabilities of each point are used as a unary term. One such approach [Shapovalov et al., 2010] to point semantic cloud labeling first performs a pre-segmentation followed by a non-associative Markov random field with pairwise potentials to express interactions between objects. In [Najafi et al., 2014] higher-order cliques are used to represent long-range correlations. This approach performs well on small outdoor point clouds captured by aerial or mobile mapping systems. Another strategy that works well on small indoor LiDAR

scans is to deploy FPFH in combination with a conditional random field (CRF) as demonstrated by [Rusu et al., 2009]. In [Niemeyer et al., 2011, Schmidt et al., 2014] the class conditional probabilities of a random forest are used as a unary term in a CRF to detect land-cover in shallow coastal areas.

Deep learning. Recently, convolutional neural networks (CNNs) show successes, not only on dense voxel grids but also on generic point clouds. To the best of our knowledge, the earliest attempt that deploys voxel grids as data representation for 3D-CNNs is [Prokhorov, 2010]. The author improves classification accuracy for classification tasks in LiDAR point clouds. To overcome the limited amount of training data, supervised and unsupervised training is combined. [Lai et al., 2014] deploys CAD models and dense voxel grids to train CNNs for semantic segmentation of small synthetic scenes. Similarly, [Wu et al., 2015, Maturana and Scherer, 2015] train a VGG-like network [Simonyan and Zisserman, 2014b] that encodes CAD data in voxel grids for object classification. [Huang and You, 2016] deploy a dense VGG-like architecture for semantic segmentation in outdoor point clouds. Neural networks with dense voxel grids have a cubical memory growth *w.r.t.* resolution and, hence, are limited to coarse resolutions. In order to reduce the computational demand of the convolutional network, an image like 2D representation can be generated from 3D representations by deploying anisotropic probing kernels as shown in [Qi et al., 2016b]. However, such generated 2D representations [Dai et al., 2017] make the wrong assumption that all voxels in the same column have the same class label. In [Tchapmi et al., 2017] a residual network architecture is combined with a CRF to label large indoor and outdoor scenes. For deep learning, point clouds can also be represented as meshes rather than voxel grids as demonstrated by [Guo et al., 2015]. More sophisticated approaches avoid cubical memory growth as discussed in Section 2.4.

这句话可以借鉴

这里主要阐述了深度学习在分类中的作用，重点可以关注胰一下voxel

2.2 3D Contour Detection

2D contours. Line detection in 2D images is strongly related to the extraction of contours in unstructured 3D point clouds. Even in 2D images the concept of a contour is ill-defined: At what point is a rough and curved surface considered to be a contour? Canny-like edge detectors [Canny, 1986] are de facto standard to identify and link adjacent high-contrast pixels. Nevertheless, contour and line detection is challenging in 2D images and remains active research, including the research fields of topographic mapping [Türetken et al., 2011, Türetken et al., 2012], stereo reconstruction [Fabbri and Kimia, 2010] and medical imaging [Türetken et al., 2011, Türetken et al., 2012]. These more recent works follow a similar hypothesize-and-verify approach that relies on three stages: First, pixels that have a high probability of being contour pixels are identified. For this purpose, low-level image processing techniques or discriminative

learning is deployed. Second, pixels that are likely to be on a contour are linked to form contour candidates. Typically, these contour candidates are created by *i)* identifying the most likely contour pixels, seedpoints, with non-maximum suppression and *ii)* linking the seedpoints with shortest-path search or similar techniques. Third, the contour candidates are filtered to obtain an optimal subset of contour candidates as a contour graph. For this filter step, context features that encode the entire length of the resulting contour and CRF-type connectivity priors help to retain an optimal contour graph with few mistakes. For stereo images, a standard approach is to identify line features in 2D images followed by a triangulation step to obtain 3D lines. Typically, these approaches generate a set of disconnected 3D line segments [Schmid and Zisserman, 1997, Ok et al., 2012, Hofer et al., 2015]. Recent works approach the problem of contour detection with convolutional neural networks. HED [Xie and Tu, 2015] introduces a neural network that learns multi-scale and multi-level features to identify edges and object boundaries. UberNet [Kokkinos, 2016] infers multiple tasks at once including object boundary detection and semantic boundary detection. In [Maninis et al., 2017] a multiscale boundary representation is efficiently generated in a single forward pass of their network. These multiscale boundaries are then deployed for hierarchical segmentation.

3D contours. Unstructured 3D point clouds are significantly more challenging than 2D images with grid structure. Early works perform contour extraction in airborne LiDAR scans by deploying rule-based expert systems, for example [Briese, 2004]. Such hand designed expert systems try to cover all possible types of contours that can occur in the scenes. Long-range airborne scans are intrinsically less complex than point clouds of natural scenes captured at close range. This high complexity in close-range natural scenes makes the design of hand-crafted rules of expert systems infeasible. A more recent approach is to avoid the challenges induced by unstructured 3D point clouds, e.g. by transforming the point cloud into a triangle mesh, typically by deploying algorithms similar to Poisson reconstruction [Kazhdan and Hoppe, 2013]. Though it is easier to detect edges in triangle meshes than in point clouds, the process of mesh generation typically smooths sharp edges. This smoothing induces an offset into the position of the detected contours while making it more challenging to detect non-sharp edges. Multiple works propose to preserve sharp edges by identifying edges before or during triangulation [Briese, 2004, Fleishman et al., 2005, Öztireli et al., 2009, Weber et al., 2010, Boulch and Marlet, 2012]. These approaches deploy simple features for edge detection and lack a sophisticated mechanism to consider long-range interactions of the edges.

Another approach is to fit 2D or 3D shape primitives to the point cloud. Such shape primitives can be combined to form more complex geometries by applying simple set operations [Schnabel et al., 2007, Li et al., 2011, Xiao and Furukawa, 2014] similar to CAD or CSG (constructive solid geometry), e.g. union, intersection or differences.

As with triangle meshes, fitting with shape primitives benefits from explicitly known edges and contours as demonstrated in [Lafarge and Mallet, 2012]. For CAD-like applications, it is highly desirable to obtain a wireframe-like representation, e.g. a graph of contours. [Hammoudi et al., 2010] propose to fit line segments to point clouds by deploying robust fitting algorithms, such as RANSAC, and then link these segments into a line-segment-graph. Such an approach can only extract a crude approximation of wireframes due to it's restriction to simple parametric models in the form of straight lines. To process large point clouds, [Xia and Wang, 2017] propose an efficient algorithm that deploys covariance based features to generate low-level evidence and a linked graph structure that exploits the long-range interactions. Beside surface reconstruction, one popular application of 3D contours is lossy compression of large point cloud data, as shown in [Hofer et al., 2015, Chen et al., 2017].

2.3 Benchmark Tests

Images. Both in computer vision and in remote sensing, benchmarking plays a fundamental role to compare algorithms fairly. These benchmark tests often cover fundamental tasks, including High Density Aerial Image Matching [Haala, 2013, Cavegn et al., 2014], semantic segmentation and reconstruction of 3D objects in aerial images [Rottensteiner et al., 2013] or (multiview-) stereo reconstruction [Geiger et al., 2012, Seitz et al., 2006]. Certainly, the most prominent change of paradigms in recent years has happened with a benchmark submission of [Krizhevsky et al., 2012] on the *ImageNet* challenge, a large-scale image classification task. Due to the outstanding results of this benchmark submission, deep learning plays a central role in current computer vision research. To pose a challenge, benchmark tests often rely on large datasets. Large datasets help reducing over-fitting, e.g. algorithms that are tuned to one specific dataset while showing significantly worse results on other data. Furthermore, large datasets are mandatory for a multitude of statistical learning approaches. *Tinyimages*, an early challenge to large-scale object detection, published over 80 million small images ($32 \times 32\ px$) [Torralba et al., 2008]. The *ImageNet 2010* challenge contains more than one million images with ground-truth annotation for 1000 object classes [Deng et al., 2009, Russakovsky et al., 2015]. These datasets benefit from the availability of images on the web. It is easier and cheaper to download images with automated web-based scripts than to organize a measurement campaign that often requires the use of expensive sensors. With advances on algorithmic level, it is mandatory to adapt the data to pose a challenge continuously. For this purpose, some benchmarks organize challenges as annual events over several consecutive years e.g. *ImageNet, Pascal VOC*. Many of today's state-of-the-art algorithms are tested on benchmarks, e.g. *Pascal VOC* [Everingham et al., 2010] for semantic image segmentation or *MSCOCO* [Lin

et al., 2014] for object recognition and image captioning. In recent years, depth imaging has improved both in quality and price. For semantic segmentation in RGB-D data the *NYU* Depth Dataset [Silberman et al., 2012] or the *SUN RGB-D* [Song et al., 2015] offer several thousand pixel-wise annotated depth images. *ScanNet* [Dai et al., 2017] provides over 2.5 million views in 1500 RGB-D scans of small indoor scenes including crowdsourced ground truth annotation.

Point clouds. While there exist large databases of point clouds, only a few are publicly available, and less are ground truth annotated. The limited availability of 3D point clouds with ground truth annotation has prevented the generation of large-scale 3D benchmark tests that are as large and rich as vision benchmarks. Many recent neural network approaches avoid this problem by deploying synthetic data in the form of CAD models for training and testing. Typically, these CAD datasets, like ModelNet [Wu et al., 2015], are rather small so that recent deep learning approaches tend to overfit. For example [Brock et al., 2016] records a performance of over 97% on ModelNet10; An indicator that the data does not pose a challenge anymore. The availability of cheap indoor 3D scanning devices, like the Kinect, makes it possible to capture large 3D indoor scenes, e.g. the Stanford Large-Scale 3D Indoor Spaces (S3DIS) published 215 million points from various buildings with ground truth annotation for 13 object classes [Armeni et al., 2016, Armeni et al., 2017].

这里说了好多点云数据集

Beside low-resolution airborne scans, datasets for 3D outdoor scenes mostly stem from mobile mapping systems or robots, e.g. *DUT1* [Zhuang et al., 2014], *DUT2* [Zhuang et al., 2015], or *KAIST* [Choe et al., 2013]. These datasets are not publicly available and have less than ten million points. Other databases like *Oakland dataset* [Munoz et al., 2009a], the *Sydney Urban Objects dataset* [De Deuge et al., 2013] or the *Paris-rue-Madame database* [Serna et al., 2014] are publicly available but contain less than 30 million points. The *IQmulus & TerraMobilita Contest* [Vallet et al., 2015] provides one hierarchically annotated training set of 12 million points and 10 small test sets. The ground truth of the 10 test sets is held back to guarantee a fair evaluation. *Paris Lille*, a dataset, that has captured 143 million points with a mobile mapping system [Roynard et al., 2017], is annotated with the same classes as [Vallet et al., 2015]. The *TUM City Campus* dataset contains 1.5 billion 3D points captured by a mobile mapping device. The dataset is partly ground truth annotated for 8 classes [Gehrung et al., 2017]. Most point cloud datasets lack a thorough evaluation system with public availability, held back ground truth and listed benchmark submissions.

Submissions. Submissions to our benchmark cover a broad range of approaches. Possibly, the most straightforward approach is to render 2D images and perform classification with standard techniques from image processing. The works from [Lawin et al., 2017] and [Boulch et al., 2017] follow this approach by *i)* rendering virtual images in the scene and *ii)* deploying convolutional neural networks for semantic segmentation

in the images. These approaches certainly benefit from a large amount of training data that is available for images. However, it turns out that classification in 3D space can achieve better results. In [Tchapmi et al., 2017] an end-to-end learning method is introduced that combines a 3D CNN with a CRF. Both the CNN and the CRF are trained jointly. The data is represented by coarse voxel grids that have a resolution of 5cm and a maximum volume of $100 \times 100 \times 100$.

The so far most successful benchmark submission [Landrieu and Simonovsky, 2017] heavily relies on deep learning with graph convolutions. In the first step, the point cloud is partitioned into simple parts by deploying graph-cut; The paper refers to these simple shapes as superpoints. Adjacent superpoints are linked into an oriented graph structure. For classification of the superpoints, a PointNet architecture is combined with graph convolutions in the form of a gated recurrent unit (GRU). While PointNet generates descriptors for each individual superpoint, the GRU allows considering long-range interactions.

2.4 Sparse Deep Learning

Data sparsity. Possibly, the primary factor for today's success of deep learning is an efficient implementation of neural networks that makes good use of the available computational power of modern graphics cards. In order to avoid cubical memory growth in dense voxel grids, most recent approaches to deep learning with 3D data either try to avoid voxel grids as data representation or exploit the sparseness in voxel grids. One such method, [Qi et al., 2016a, Qi et al., 2017a], avoids voxel grids by working on raw point clouds. For this purpose, they extract global feature vectors with a multi-layer-perceptron and select interesting points with pooling operations. Their approach works well on small 3D datasets. However, it lacks classification accuracy on large-scale problems. Another strategy is to exploit sparseness in voxel grids, e.g. Graham *et al.* [Graham, 2014] uses a sparse data representation but lacks mechanisms to encourage sparsity; Encouraging sparsity is a central challenge in sparse neural network design, since common convolutional and pooling layers significantly decrease sparsity.

Other works exploit the sparseness of voxel grids with a spatial decomposition, e.g. [Riegler et al., 2017,Tatarchenko et al., 2017] spatially decompose voxel grids with OcTrees to avoid cubical memory growth. While [Riegler et al., 2017] encourage sparsity by using a fixed decomposition based on the structure of the input data [Tatarchenko et al., 2017] learn the structure of the OcTree such that it is applicable to generative networks. In Häne *et al.* [Häne et al., 2017] a coarse-to-fine scheme is deployed for hierarchical prediction for blocks of voxels in a voxel block OcTree. In [Graham and

这点是不是与老提的要求有关系

van der Maaten, 2017, Graham et al., 2017] sparsity in data is enforced by enforcing zero valued features in the input of a layer to stay zero. However, their network implementation relies on hash tables, which come with a worst case complexity of $O(n)$.

Parameter sparsity. Beside sparsity in data, parameter sparsity plays a central role for runtime and memory savings. Early works by Denil *et al.* [Denil et al., 2013] predict model parameters with low-rank matrix factorization so that they can compress the memory to 5% of the dense requirements without a significant drop in classification accuracy. Similarly, Liu *et al.* [Liu et al., 2015] exploit the decomposition of matrices to reduce the redundancy in parameters. In [Jaderberg et al., 2014] and [Denton et al., 2014] the convolutional filters are approximated with linear structures, e.g. with a linear combination of a set of small basis filters. Han *et al.* [Han et al., 2015] use a three-stage connection pruning technique to reduce the parameters of a neural network by *i)* training a network, *ii)* pruning small weights, *iii)* retraining the pruned network for fine-tuning. Recently, the works from [Gray et al., 2017] and [Narang et al., 2017] deploy sparse blocks to represent filter weights and exploit these block sparse filter weights for runtime gains. While [Narang et al., 2017] finds that block sparsity decreases classification accuracy, [Gray et al., 2017] shows that on their tasks block sparsity increase classification accuracy. Rather than relying on block sparsity within channels, [Changpinyo et al., 2017] introduces a channel-wise sparse connection structure for 2D convolutions.

Joint data and parameter sparsity. Three recent and independent works [Park et al., 2017, Engelcke et al., 2017, Parashar et al., 2017] propose a novel approach to compute sparse convolutions efficiently while heavily relying on atomic operations; atomic operations are executed within one processor cycle so that they are thread-safe without requiring locking mechanisms. This direct convolutional approach is capable of exploiting parameter sparsity jointly with data sparsity. Park *et al.* [Park et al., 2017] describe techniques to exploit parameter sparsity by deploying compressed rows as a sparse tensor format. Engelcke *et al.* utilize data sparsity with a non-standard sparse tensor format in a slow CPU-implementation. Parashar *et al.* [Parashar et al., 2017] exploit the sparsity of both parameters and data with the sparse blocks format on custom-designed hardware, that is both more energy efficient and faster than the dense baseline. Convolutional layers with kernel larger than $1 \times 1 \times 1$ significantly decrease sparseness in data. None of these works deploy techniques that preserve sparsity in the convolutional layers.

Chapter 3

Background

Some of the earliest and most well known optical instruments are Galileo's telescopes. With these devices came observations of the mountainous moon, stars, and planets that inspired generations of researchers. Ever since, the development of better optical instruments went hand in hand with more advanced models capable of interpreting and explaining the obtained data, so that a broad range of sensors and models have been developed. Nowadays, common optical instruments exploit the photoelectric effect to generate digital data, e.g. passive CCD and CMOS cameras or active laser scanners. Both cameras and laser scanners are suitable to create 3D reconstructions of a scene. This Chapter introduces some standard techniques for the processing of 3D data that are important to understand this work. First, we introduce 3D data representations followed by sections on machine learning algorithms that can be used to infer information from 3D data.

3.1 3D data representations

Point Clouds. Typical 3D sensors deploy line-of-sight measurement principles to capture point clouds, e.g. 3D points on the 2D surface of a car or house as shown in Figure 3.1. To store two-dimensional data in a three-dimensional space, often a sparse data format in the form of a coordinate list (COO) is deployed. In this format, the coordinates of each 3D point are stored as floating point numbers complemented by additional information, like rgb colors, intensity values or class labels, e.g. $\{x, y, z, r, g, b\}$. Coordinate lists for point clouds have large memory requirements since they store all coordinates as floating point numbers. Hence, data compression can help to reduce the memory requirements for storing large point clouds. Due to the floating point coordinates, well-known compressed sparse tensor formats like compressed sparse rows (CRS) or compressed sparse blocks [Buluç et al., 2009] are not applicable since they

三维坐标以浮点数的格式储存，非常占用存储空间

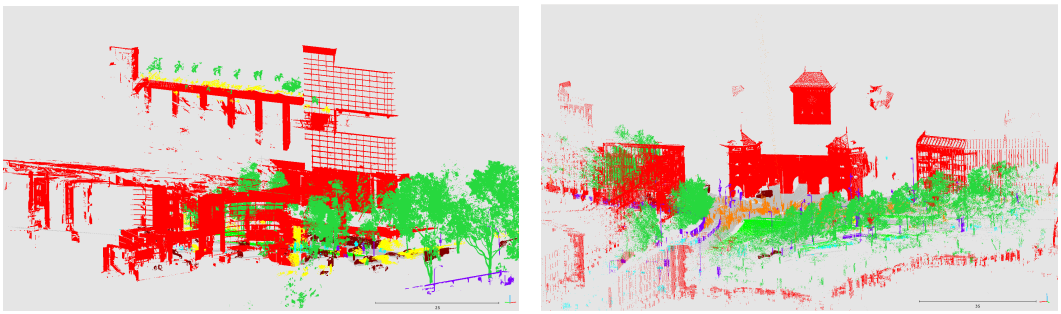
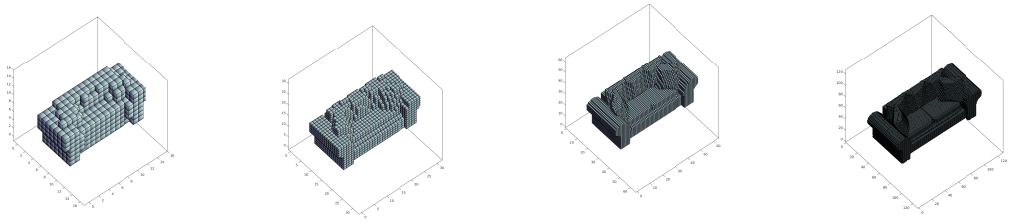


Figure 3.1: Example point clouds from *left)* office buildings in Singapore and *right)* the Isartor in Munich with human ground truth annotation for 8 object classes: man-made terrain, natural terrain, high vegetation, low vegetation, buildings, remaining hard scape and scanning artifacts, cars.

depend on a finite set of integer coordinates. However, it is possible to obtain a more memory efficient representation by removing noise and by deploying entropy-based compression functions, for instance as specified in the LAS and LAZ data format [Isenburg, 2011]. One challenge that comes with COO is the absence of neighborhood information. In order to identify neighbors in COO search trees like kd-trees [Friedman et al., 1977, Muja and Lowe, 2009] are deployed. Typically, these search structures come with a high time complexity, e.g. $O(log(n))$ that is significantly higher than the $O(1)$ of a neighborhood query in 2D image grids. Especially, for large point clouds with a large number n of points, these neighborhood queries can turn into a computational bottleneck.

这是两种数据格式

Voxel grids. Another common 3D data representation, dense voxel grids, discretizes



(a) $r_1 = 16^3, \rho_1 = 15.5\%$ (b) $r_2 = 32^3, \rho_2 = 9.1\%$ (c) $r_3 = 64^3, \rho_3 = 5.3\%$ (d) $r_4 = 128^3, \rho_4 = 2.7\%$

Figure 3.2: Effects of voxelization r on density ρ in voxel grid on an example from Modelnet40 dataset.

the 3D coordinates of measurements into a grid structure. Voxel grids have an efficiently encoded neighborhood, e.g. neighborhood queries have a complexity of $O(1)$. However, they come with two main drawbacks as illustrated in Figure 3.2: *i)* Voxel grids intrinsically come with quantization errors that are larger with coarser resolutions of the voxel grid; *ii)* Voxel grids do not exploit sparsity: The memory require-

有伴随的意思

学会这种方式

以。。。增长，学习这种方式

ments grow cubically with the resolution. Voxel grids can be interpreted as 3D tensors so that sparse tensor formats, e.g. CSR, can exploit the sparsity in voxel grids. However, these sparse formats come at the cost of higher time complexity for neighborhood queries.

OcTree. While generic sparse tensor formats do not encode neighborhood informa-

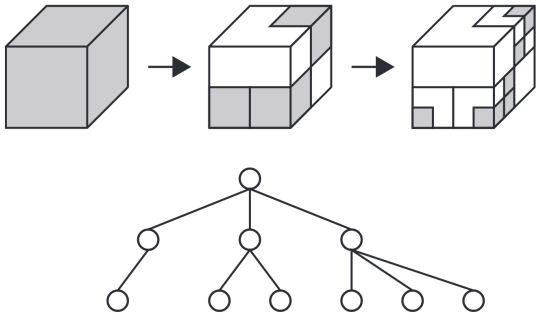


Figure 3.3: Example of spatial decomposition of sparse data with a three-layer OcTree. (Source [Elseberg et al., 2013])

tion, OcTrees encode neighborhood information and exploit the sparsity in 3D space by decomposing the space into blocks of free space and non-free data blocks, see Figure 3.3. Neighborhood queries can be performed fast by storing neighborhood pointers in the leaf nodes of the tree. This comes at the cost of high memory requirements. Alternatively, the tree can be traversed. In general, it is not known which search structure is best suited to find point neighborhoods quickly. For some problems, OcTrees turn out to be better than kd-trees [Elseberg et al., 2013]. For other problems, approximate nearest neighbor approaches are faster and preferable [Muja and Lowe, 2009]. However, the construction of an OcTree has a high complexity, e.g. adding or deleting points comes with $O(log(n))$, where n is the number of 3D points in the tree.

3.2 Supervised Classification

自动对焦

Object detection with classifiers plays an important role in many popular technological devices and applications, e.g. the autofocus of mobile phone cameras relies heavily on face detection. In general, these classifiers are functions that map input data $\mathbf{x}$ to a class label $y = f(\mathbf{x}, \mathbf{w}) \,|\, y \in \mathcal{C}$, where $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ is a set of discrete class labels and $\mathbf{w}$ are the parameters of the classifier. Supervised learning approaches deploy a training data set of m training samples $\mathbf{X}_{train} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ that have known class labels $\mathbf{Y}_{train} = \{y_1, \ldots, y_m\}$ to find a set of parameters $\mathbf{w}$ that has a small prediction error. When training a classifier the objective is *not* to minimize the prediction error on the training set: $\mathcal{L}(f(\mathbf{X}_{train}, \mathbf{w}), \mathbf{Y}_{train})$, where $\mathcal{L}$ is a loss function

短语

map...to
映射

that defines the prediction error. Instead, the objective is to minimize the generalization error, e.g. perform good predictions on unseen data that are not part of the training set $\mathbf{X}_{train}$. Since previously unseen data cannot be used for training, the minimization of the generalization error is an ill-posed problem. For this purpose, the generalization error is approximated. A common approximation of the generalization error is the validation error that deploys a validation set $\{\mathbf{X}_{val}, \mathbf{Y}_{val}\}$ of samples that are not in the training set: $\mathcal{L}(f(\mathbf{X}_{val}, \mathbf{w}), \mathbf{Y}_{val})$. On small datasets, it is beneficial to use all samples in the training set for validation: *K-fold cross-validation* randomly splits the training set into k subsets. Each subset is used as validation set once while the remaining $k-1$ subsets are used to train the classifier. The mean validation loss over all k iterations turns out to be a good approximation to the generalization loss.

Within this thesis, we deploy two classification approaches:

Traditional machine learning. In traditional machine learning the input data to a classifier are hand-crafted features: $\mathbf{x} = g(\mathbf{x}_{data})$, where $\mathbf{x}_{data}$ denotes raw data, e.g. points from a 3D point cloud. These features $g(.)$ map $\mathbf{x}_{data}$ into feature space where it is better possible to discriminate between object classes than in raw data. Section 3.2.1 introduces AdaBoost. A variant of AdaBoost is deployed in Chapter 5. Furthermore, Chapter 4 and Chapter 5 make use of a Random Forest. Random Forests are described in Section 3.2.2;

Representation learning. The design of hand-crafted features $g(\mathbf{x}_{data})$ that can discriminate between different object classes often is non-trivial. Neural networks avoid hand-crafted features by deploying differentiable parametric functions that learn how to best distinguish between object classes. Typically, these networks learn representations that consider a large context around the actual value. In practice, the input to neural networks often is raw data: $\mathbf{x} = \mathbf{x}_{data}$. Section 3.2.3 describes principles of neural networks that are used in Chapter 5 and Chapter 6.

3.2.1 AdaBoost

Among the simplest and weakest classifiers are decision stumps and linear classifiers. AdaBoost, a technique introduced by [Freund et al., 1996], combines many weak classifiers to form a strong ensemble. Classifier ensembles converge to perfect classifiers the more independent weak classifiers are combined; even if the performance of the weak classifiers is only slightly better than random predictions. Furthermore, ensembles can learn highly non-linear decision boundaries even when using linear classifiers. For this reason, an iterative three-stage approach is introduced: *i)* Train a new weak classifier with a weighted loss function, where those training samples that are often misclassified have a higher weight than accurately classified training samples; *ii)* Cal-

culate the loss of the current classifier on the training set; *iii)* Recalculate the weights of the training samples based on the current loss. Algorithm 1 describes this training routine.

Algorithm 1 AdaBoost training routine

1: initialize data weights ϕ uniformly: $\phi_i^{(1)} = \frac{1}{m}$
2: **for** $b \in [0 : \textit{ensemble\_size}]$ **do**
3: *i)* train a weak classifier $f(\mathbf{x}, \mathbf{w}_b)$ *w.r.t.* the weighted sum of the zero-one-loss:

$$\mathcal{L}(\mathbf{x}, y, \phi^{(b)}, \mathbf{w}_b) = \sum_{i=1}^{m} \phi_i^{(b)} I(f(\mathbf{x}, \mathbf{w}_b)_i \neq y_i). \tag{3.1}$$

$I(f(\mathbf{x}_i, \mathbf{w}_b) \neq y_i)$ is the indicator function that is 1 for false predictions and 0 for correct predictions. $\{\mathbf{x}_i, y_i\}$ is the training data.
4: *ii)* evaluate and normalize the weighted loss:

$$\epsilon_b = \frac{\mathcal{L}(\mathbf{x}, y, \phi^{(b)}, \mathbf{w}_b)}{\sum_{i=1}^{m} \phi_i^{(b)}}. \tag{3.2}$$

Compute the importance of the current classifier:

$$\alpha_b = ln\Big(\frac{1 - \epsilon_b}{\epsilon_b}\Big). \tag{3.3}$$

5: *iii)* Update the data weights:

$$\phi_i^{(b+1)} = \phi_i^{(b)} exp(\alpha_b \cdot I(f(\mathbf{x}_i, \mathbf{w}_b) \neq y_i)). \tag{3.4}$$

6: **end for**

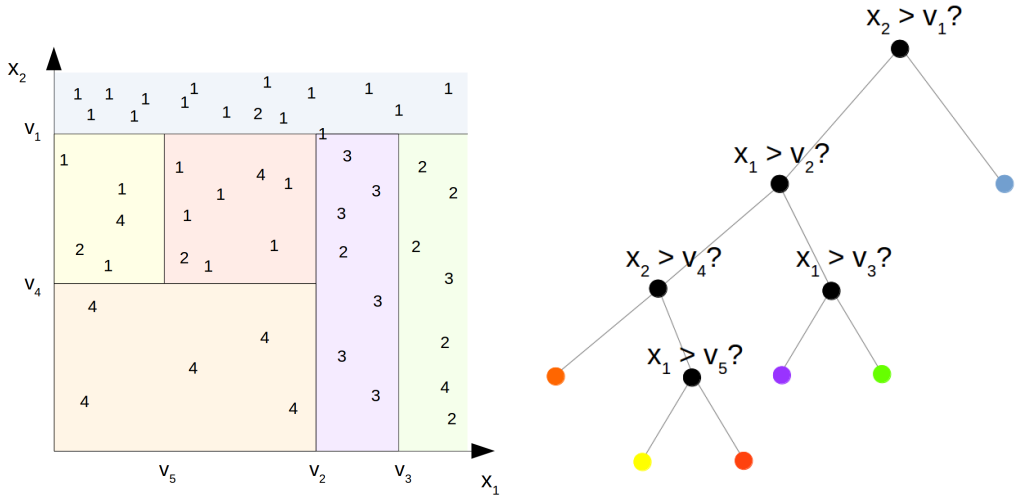
Predictions of ensembles are a weighted sum over the predictions of all weak classifiers:

$$\tilde{y}_i = sign\Big(\sum_b \alpha_b \cdot f(\tilde{\mathbf{x}}_i, \mathbf{w}_b)\Big) \tag{3.5}$$

The signum function $sign(.)$ is 1 for positive values and -1 for negative values. One of the main problems of AdaBoost is the limitation to solve binary classification problems that only have two classes. For more information on AdaBoost and it's multi-class-extensions refer to [Bishop, 2006, Hastie et al., 2009].

3.2.2 Random Forest

Decision Tree. Decision trees are another common approach to combine classifiers



(a) (b)

Figure 3.4: Example of cuboid decision boundaries of a random forest: *a)* One possible partitioning of the feature space; *b)* The underlying tree structure. Each leaf node stores class conditional probabilities per cuboid, e.g. the relative class frequency in the cuboid.

by deploying weak learners as non-leaf nodes [Breiman et al., 1984]. Though, the nodes consist of weak learners, such as decision stumps that split the data into two subsets, the trees can define highly non-linear decision boundaries, compare Figure 3.4. Predictions with decision trees simply traverse those nodes that are decided by the weak learners until a leaf node is reached. The leaf nodes store class conditional probability distributions that are learned during training. However, training of decision trees turns out to be challenging.

Training. The learning of the tree structure consists of two important components: *i)* identification of an optimal number of nodes, e.g. by providing an optimal depth; *ii)* selection of a discriminative feature for each node, including the estimation of a threshold parameter for the split. Due to a large number of possible solutions, there is no computational feasible strategy that is able to solve both problems in an optimal manner. For this reason, the features are selected with a greedy approach that only considers single nodes starting with the root node rather than optimizing the entire tree at once [Bishop, 2006]. To jointly select the discriminative feature and finding the threshold, an exhaustive search over the feature space is performed for each node individually. For many problems a search over the entire feature space is computationally too demanding so that random subsets of the feature space are deployed. This optimization is commonly performed with either of the following two splitting criteria as error metric:

i) The cross-entropy

$$Q_\tau(v) = \sum_{i=1}^{n} p_{\tau\mathcal{C}_i} ln(p_{\tau\mathcal{C}_i}); \tag{3.6}$$

ii) The gini-index

$$Q_\tau(v) = \sum_{i=1}^{n} p_{\tau\mathcal{C}_i}(1 - p_{\tau\mathcal{C}_i}), \tag{3.7}$$

where $p_{\tau\mathcal{C}_i}$ denotes the probability of points in cuboid τ to be of class $\mathcal{C}_i$ and v denotes the threshold. In this thesis, we identify the depth of the decision tree by deploying an exhaustive grid search in combination with k-fold cross-validation.

Bootstrap aggregation. A common problem of decision trees is overfitting, e.g. with a too large depth of the tree, there is only a single training sample per cuboid so that there are no more training errors. Typically, decision trees are pruned with heuristic rules to reduce overfitting. The works from [Breiman, 1996, Amit and Geman, 1997] introduce bootstrap aggregation (Bagging), a more sophisticated approach that complements pruning heuristics. For this purpose, the training is randomized, e.g. by training on a randomly selected subset of training data or by considering a random subset of features per node. Random forests combine many randomized decision trees with model averaging; a technique to reduce the generalization errors of individual trees.

3.2.3 Convolutional Neural Network

Forward pass. In general, neural networks are a technique to approximate arbitrary complex functions with connected chains of simple, differentiable functions $\mathbf{x}^{(i+1)} = f^{(i)}(\mathbf{x}^{(i)}, \mathbf{w})$, e.g. $\mathbf{x}^{(5)} = f^{(4)}(f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x}^{(1)}, \mathbf{w}_1), \mathbf{w}_2), \mathbf{w}_3), \mathbf{w}_4)$. In neural networks these simple, differentiable functions are referred to as layers. Among the most important layers are convolutional layers and dense layers: Dense layers [Rosenblatt, 1957] are linear classifiers that connect each output value with each input value:

$$f^{(i)}_{dense}(\mathbf{x}^{(i)}, \mathbf{w} = \{\mathbf{W}, \mathbf{b}\}) = \mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}, \tag{3.8}$$

where the weights $\mathbf{W}$ and biases $\mathbf{b}$ are the trainable parameters $\mathbf{w}$ of the layer. Convolutional layers [LeCun et al., 1998] perform convolutions on the layer's input to extract shift invariant information:

$$f^{(i)}_{conv}(\mathbf{x}^{(i)}, \mathbf{w}) = \sum_j \mathbf{w}_j \mathbf{x}^{(i)}_j + \mathbf{w}_0, \tag{3.9}$$

where $\mathbf{w}_0$ is a bias and the remaining parameters are filter weights that are used to compute the convolutions. Typically, these layers are combined with non-linear functions, so-called activation functions. The most common activation functions are applied to each output value of a layer individually and are not trainable, e.g. $\mathbf{w} = \{\}$. Among these activation functions are rectified linear units:

$$f_{ReLU}(\mathbf{x}_j^{(i)}) = max(0, \mathbf{x}_j^{(i)}), \tag{3.10}$$

sigmoid functions:

$$f_{sigmoid}(\mathbf{x}_j^{(i)}) = \frac{1}{1 + e^{-\mathbf{x}_j^{(i)}}}, \tag{3.11}$$

or the hyperbolic tangent:

$$f_{tanh}(\mathbf{x}_j^{(i)}) = tanh(\mathbf{x}_j^{(i)}). \tag{3.12}$$

Gradient-based learning. The objective of supervised learning is to find a set of parameters $\mathbf{w}$ that minimizes the error of the loss function $\mathcal{L}(.)$. In classification tasks the test set error is computed with a zero-one loss:

$$\mathcal{L}_{0-1}(\mathbf{x}^{(i)}, \mathbf{y}_j, \mathbf{w}) = I(f(\mathbf{x}^{(i)}, \mathbf{w})_j \neq \mathbf{y}_j), \tag{3.13}$$

where $I(.)$ is the indicator function. However, while training a network it turns out to be beneficial to also consider probabilities of correct and misclassified predictions, e.g. with the cross-entropy loss:

$$\mathcal{L}_{cross}(\mathbf{x}^{(i)}, \mathbf{y}_j, \mathbf{w}) = -\sum_{c=1}^{n} I(\mathcal{C}_c = \mathbf{y}_j) \cdot log(p(\mathcal{C}_c|\mathbf{x}^{(i)}, \mathbf{w})) \tag{3.14}$$

In order to minimize these error functions, the works of [Rumelhart et al., 1985] and [LeCun et al., 1989] propose to propagate error gradients through the network by deploying chain rule:

$$\frac{\partial L(\mathbf{x}^{(i)}, \mathbf{y}_j, \mathbf{w}_i)}{\partial \mathbf{x}^{(i)}} = \frac{\partial L(\mathbf{x}^{(i+1)}, \mathbf{y}_j, \mathbf{w}_{i+1})}{\partial \mathbf{x}^{(i+1)}} \frac{\partial f^{(i)}(\mathbf{x}^{(i)}, \mathbf{w}_i)}{\partial \mathbf{x}^{(i)}} \tag{3.15}$$

$$\frac{\partial L(\mathbf{x}^{(i)}, \mathbf{y}_j, \mathbf{w}_i)}{\partial \mathbf{w}_i} = \frac{\partial L(\mathbf{x}^{(i+1)}, \mathbf{y}_j, \mathbf{w}_{i+1})}{\partial \mathbf{x}^{(i+1)}} \frac{\partial f^{(i)}(\mathbf{x}^{(i)}, \mathbf{w}_i)}{\partial \mathbf{w}_i} \tag{3.16}$$

Given the error gradients, the trainable parameters of the network can be learned by deploying gradient descent based optimization. To account for the large size of training data sets, variations of stochastic gradient descent are commonly deployed, like adam [Kingma and Ba, 2014] or adagrad [Duchi et al., 2011]. For further reading refer to [Goodfellow et al., 2016].

Chapter 4

Joint Classification and Contour Extraction of Large 3D Point Clouds

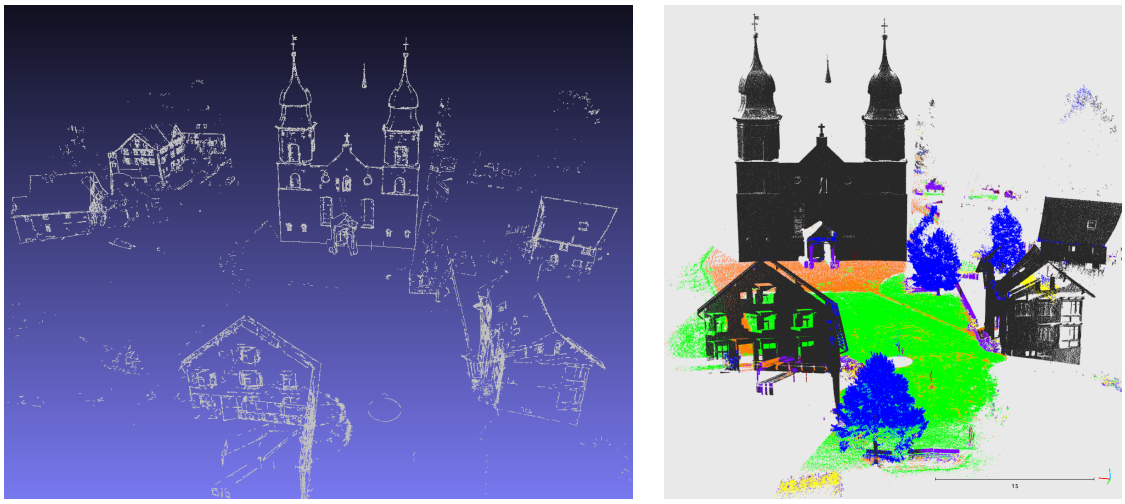
Abstract

We present an effective and efficient method for point-wise semantic classification and extraction of object contours of large-scale 3D point clouds. What makes point cloud interpretation challenging is the sheer size of several millions of points per scan and the non-grid, sparse, and uneven distribution of points. Standard image processing tools like texture filters, for example, cannot handle such data efficiently, which calls for dedicated point cloud labeling methods. It turns out that one of the major drivers for efficient computation and handling of strong variations in point density, is a careful formulation of per-point neighborhoods at multiple scales. This allows, both, to define an expressive feature set and to extract topologically meaningful object contours.

Semantic classification and contour extraction are interlaced problems. Point-wise semantic classification enables extracting a meaningful candidate set of contour points while contours help generating a rich feature representation that benefits point-wise classification. These methods are tailored to have fast run time and small memory footprint for processing large-scale, unstructured, and inhomogeneous point clouds, while still achieving high classification accuracy. We evaluate our methods on the semantic3d.net benchmark for terrestrial laser scans with $> 10^9$ points.

4.1 Introduction

With advanced measurement technologies, like terrestrial laser scanners, it takes fairly little effort and time to collect millions to billions of 3D points in various environments



(a) (b)

Figure 4.1: Examples of (a) extracted contours and (b) point-wise semantic classification; *gray:* buildings, *orange:* man made ground, *green:* natural ground, *yellow:* low vegetation, *blue:* high vegetation, *purple:* hard scape, *pink:* cars

and for many different applications. Yet, large collections of 3D data are hard to process due to their lack of structure. This is especially true in natural scenes, where not only the captured data itself is challenging but also the many possible variations of objects within the scene.

Most applications avoid working directly on these challenging data sets by transferring point clouds into different representations, such as triangle meshes or more CAD-like structures, e.g. [Schnabel et al., 2007, Lafarge and Mallet, 2012, Xiao and Furukawa, 2014]. The most common CAD model is Constructive Solid Geometry, where parametric 3D solids are fitted to the data as shape primitives. Such an approach quickly reaches its limits in complex scenes, where large portions of the geometry are not covered by the primitive library. A more generic approach is to extract contours directly in point clouds and use them as representation (Fig. 4.1(a)). At first glance this step might appear of little advantage, since edges can also be extracted from a triangle mesh or parametric model. But, arguably, it is advantageous to already know the contours as an input for mesh triangulation or surface fitting, so as to avoid modelling errors that frequently occur around contours. Especially mesh triangulation is known to smooth out edges.

Intuitively, contours should thus be detected *before* surface reconstruction to support segmentation. Interactive modelling systems in graphics and vision [Taylor et al., 1996] and in commercial mapping [Trimble, Inpho Geo-Modeling Module, 2016, Eos Systems Inc., Photomodeler, 2017] actually operate in this way. In practice it is challenging to detect contours in point clouds. The main reasons are not only the unstruc-

tured data but also the unclear definition of what constitutes a contour. While high curvature values play an important role there are also other factors, which cannot be neglected. For instance, the curvature where two building walls meet can in some cases be quite a bit lower than on rough surfaces, e.g. meadows. Obviously, a human operator uses additional contextual cues to identify what constitutes a contour. This situation, where one must exploit diffuse knowledge that is hard to make explicit and formalise, calls for statistical machine learning.

Much the same can be said of semantic segmentation, i.e., the task to assign each individual data point a semantic class label (Fig. 4.1(b)). Early work on semantic segmentation was concerned with airborne data. The point data was converted to a 2.5D range image (rastered height field), so that it can be treated with image processing techniques [Hug and Wehr, 1997, Haala et al., 1998]. Newer methods, especially those concerned with terrestrial datasets consisting of multiple scans, follow a more general approach and work directly on 3D points [Chehata et al., 2009, Yao et al., 2011, Golovinskiy and Funkhouser, 2009, Weinmann et al., 2013]. Very recently, Convolutional Neural Networks (CNN) have shown promising results also for 3D data, following their success in image interpretation and speech recognition [Maturana and Scherer, 2015, Song and Xiao, 2016, Shen et al., 2015].

It is easy to see that semantic segmentation and contour detection are related, both because contours often coincide with object boundaries and because the geometric properties of contours may vary depending on the object class (c.f. "breaklines" on natural terrain or on man-made objects). In this paper we extend preliminary work [Hackel et al., 2016b, Hackel et al., 2016a] on contour detection and semantic classification by showing experimentally that it is beneficial to couple the two tasks and solve them jointly.

4.2 Related Work

Contour detection is closely related to edge and boundary detection in 2D images, which is still an active field of research, in spite of long-standing classics like the Canny edge detector [Canny, 1986]. Supervised machine learning has brought about important advances in 2D contour detection, where recent algorithms approach human performance [Konishi et al., 2003, Shen et al., 2015, Xie and Tu, 2015].

One option to construct 3D lines in the multi-view setting is to triangulate them from 2D line segments extracted in images [Schmid and Zisserman, 1997, Ok et al., 2012, Hofer et al., 2015]. The outcome are mostly short, disconnected, straight line segments in 3D that can certainly support certain modelling tasks, but are a long way from connected, topologically meaningful wireframe. In many cases point clouds are

not generated via multi-view matching or the source images are not available; such that contour detection must start directly in 3D space. Early work in computer graphics has confirmed the intuition that 3D contours can be identified better with more complex feature sets, which go beyond simple curvature values [Pauly et al., 2003]. Our contour detector relies on a hypothesize-and-verify strategy: an over-complete set of line candidates is extracted via shortest path search (based on point-wise features), followed by a global pruning step that selects those candidates that fulfill appropriate topological conditions like long-range connectivity. The technically most related work we are aware of comes from the fields of medical imaging [Türetken et al., 2011, Türetken et al., 2012] and topographic mapping [Tupin et al., 1998, Montoya-Zegarra et al., 2014, Wegner et al., 2015]. A recurrent finding in these works is that the local evidence, i.e., the likelihood that an individual point of a line lies on a contour, is best learned from a large set of labeled training data (rather than guessed and hand-coded).

Early work on semantic point cloud segmentation transformed the points (recorded from airborne platforms) into other representations such as regular raster height maps, in order to simplify the problem and benefit from the comprehensive toolbox of image processing functions [Hug and Wehr, 1997, Maas, 1999, Haala et al., 1998, Rottensteiner and Briese, 2002]. Recent work follows a more generic approach that can also deal with true 3D data. Learning and prediction operate directly on 3D points [Charaniya et al., 2004, Chehata et al., 2009, Niemeyer et al., 2011, Yao et al., 2011], such that one can also process data which cannot be reduced to height maps in a straightforward manner, in particular terrestrial data generated from multiple imaging or scan positions, and mobile mapping data [Weinmann et al., 2013, Dohan et al., 2015].

Training a good model requires an expressive feature set. A large number of 3D point descriptors has been developed, which typically encode geometric properties within the point's neighborhood, like surface curvature, surface normal orientation, *etc.* . Popular descriptors are for example spin images [Johnson and Hebert, 1999], fast point feature histograms (FPFH) [Rusu et al., 2009] and signatures of histograms (SHOT) [Tombari et al., 2010]. One drawback of these rich descriptors is their high computational cost. While computation time is not an issue for small point sets (e.g., sparse key points), it is a crucial bottleneck when *all* points in a large point cloud shall be classified. A faster alternative if working on range images instead of 3D point clouds is the NARF operator that is a widely used for key point extraction and description in the robotics community [Steder et al., 2011, Steder et al., 2010]. In order to achieve robustness against viewpoint changes, it explicitly models object contour information. A computationally cheaper alternative for full 3D point cloud features is derived from the structure tensor of a point's neighbourhood [Demantké et al., 2011], and the point distribution in oriented (usually vertical) cylinders [Monnier et al., 2012, Weinmann et al., 2013]. Neural networks (usually of the deep, convolutional network flavour) offer the possibility to completely avoid heuristic feature design and feature selection. They

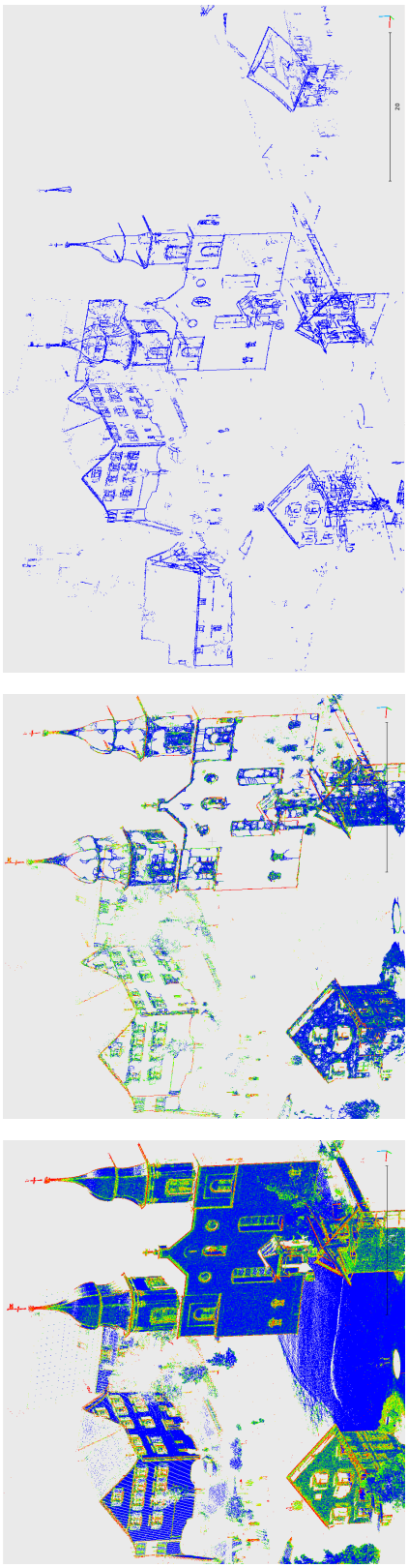


Figure 4.2: Illustration of our contour detection pipeline. (left) A binary classifier predicts point-wise contour scores (red: high contour probability, blue: low contour probability); (middle) Seed points with high contour scores are linked into an over-complete graph of contour candidates; candidates are re-scored with another round of classification; (right) candidates are pruned to an optimal set of contours by MRF inference.

are at present immensely popular in 2D image interpretation, recently deep learning pipelines have been proposed for voxel grids [Lai et al., 2014, Wu et al., 2015, Maturana and Scherer, 2015] and RGB-D images [Song and Xiao, 2016], too. These deep learning techniques inherently capture appearance based features as well as geometric object properties. Yet, they are computationally demanding and have so far been limited to comparatively small point clouds.

Our solution is in some sense related in that we aim to use correlations between object classes and the geometric properties of contours. In both cases the hope is that mutual feedback between semantics and geometry will benefit the overall interpretation. Our approach is simpler and less elegant, but much more efficient to compute and more generally applicable in case of limited training data.

4.3 Approach

Requirements for our processing pipeline are efficiency in terms of both run-time and memory footprint, such that the algorithm can be applied to point clouds of realistic size. In fact, the bottleneck in terms of efficiency are the large number of 3D nearest-neighbor queries to construct the neighborhood structure for feature and contour computation. We build on the following, simple key insight to gain speed: to (approximately) characterise a larger neighbourhood, it is sufficient to use a proportionally smaller subset of points. It may seem that this strategy sacrifices accuracy for speed. But the computational savings are so large that one can side-step scale selection and instead exhaustively compute features over a large range of scales. Empirically, this multi-scale coverage *increases* performance. For details, refer to Section 4.4.

For contour detection, we start with computing per-point contour probabilities with a binary semantic classifier, to distinguish between points on contours and non-contours (Fig. 4.2). Here we are not trying to strictly detect semantic or geometric transactions. Instead for us contours are more abstract and ill defined. In our framework users can define contours by providing a training set for the semantic classifier.

Second, it tries to find regularly spaced points with high likelihoods, and link them into an over-complete graph of candidate contours. In a final step, an optimal subset of candidates is extracted with a higher-order random field, as described in section 4.5. This procedure is in line with recent computer vision and medical imaging research [Türetken et al., 2011, Montoya-Zegarra et al., 2014].

Please note that we explicitely integrate information from semantic multiclass classification into the contour classification problem. The reason for this is, that labeling ground truth contours in 3D space is tedious work. Thus the training set is smaller than

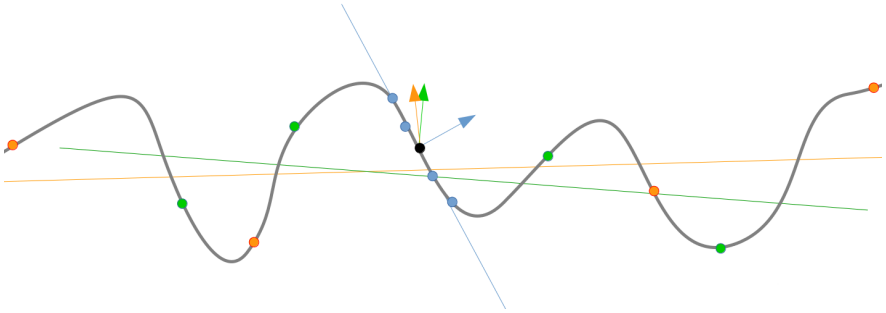


Figure 4.3: Sampling a point cloud at different scales (denoted by colors) reveals different properties of the underlying surface – here different surface normals (colored arrows).

the entire semantic3D.net training set used to train the initial per-point, multi-class classifier. Therefore, we extend the feature set of the semantic classifier for contour detection with the class conditional probabilities of an additional multi-class semantic classifion stage. For the multiclass classification contour information is used only implicitly by integrating contour features into the feature set without an additional classification stage.

4.4 Semantic Classification

Semantic classification of large point sets is a computationally demanding problem. The main bottleneck turns out to be the computation of the point neighborhoods, which is required to compute geometric features. We approximate neighborhoods with multi-scale pyramids, so that small volumes can be described with a small number of neighbors at a fine scale and large volumes with a small number of neighbors at a coarse scale. The feature set described in section 4.4.2 extends the work of [Weinmann et al., 2013] to multi-scale, and adds new features that aim at contour extraction. Details of our training routine are given in section 4.4.3.

4.4.1 Neighborhood approximation

Typically, search methods are either of geometrical nature, like radius search, or select the k nearest neighbours. From a conceptual point of view radius search has advantages because a constant radius guarantees that always the same volume is covered. Yet, radius search is impractical for large radii in very dense point clouds due to its high computational cost. Another disadvantage of radius search is the varying number of neighbours (in the worst case none), which can be a problem for further computations. The alternative, k nearest neighbors, can be seen as an adaptive version, in which the

radius is increased until it includes exactly k neighbors. For constant point density, both search strategeis give the same output.

Here, we prefer k nearest neighbors over radius search, and approximate constant point density in sufficiently dense parts of the point cloud by performing uniform down-sampling with a voxel grid filter. A range of different voxel sizes are used for voxel grid filtering, so as to enable efficient neighbour search across different scales [Brodu and Lague, 2012]. The resulting set of search trees forms a multi-scale pyramid, similar in spirit to a discrete image scale space. It has been shown that features extracted with a small neighborhood of $k = 10$ are more discriminative than larger neighborhoods [Weinmann et al., 2015b]. We apply this insight to our problem and use $k = 10$ to search within each of the scales of our pyramid. While fine scales capture detailed information of the surface, coarser scales cover a larger search volume by selecting $k = 10$ representative samples. This strategy greatly improves the computational efficiency. Another beneficial effect is the reduced size of the search trees. Uniform down-sampling reduces the point cloud size, and hence also the search tree size, which significantly speeds up search queries especially for coarse scales[1].

4.4.2 Feature extraction

Our set of features is designed for purely point-based classification, given points and their nearest neighbors. We refrain from using colour or intensity information, which is not always available, and empirically did not improve the classification in our tests. We work with local neighborhood $\mathcal{P}^N$ of point $\mathbf{p}$ because for large point clouds it becomes quickly infeasible to generate features based on much larger neighborhoods ("neighbors of neighbors") due to memory and computational limitations. We follow [Weinmann et al., 2013] and use 3D features based on eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$ and corresponding eigenvectors $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ of the covariance tensor $C = \frac{1}{k}\sum_{i\in\mathcal{P}}(\mathbf{p}_i - \overline{\mathbf{p}})(\mathbf{p}_i - \overline{\mathbf{p}})^\top$, where $\mathcal{P}$ is the set of k nearest neighbors and $\overline{\mathbf{p}} = \mathrm{med}_{i\in\mathcal{P}}(\mathbf{p}_i)$ is its medoid. The original feature set covers object boundaries only weakly. For this reason we introduce two additional features. Our first addition to the feature set is the normalized first order moment around the largest eigenvector, which we use to identify boundaries between surfaces with different point density.

$$O = \frac{\mathsf{m}_\uparrow^2}{\mathsf{m}_{\Uparrow}} \qquad \text{with} \qquad \begin{aligned} \mathsf{m}_\uparrow &= \sum_{n\in\mathcal{P}_i^N} \langle \mathbf{p}_n - \mathbf{p}_i, \mathbf{e}_2 \rangle \ , \\ \mathsf{m}_{\Uparrow} &= \sum_{n\in\mathcal{P}_i^N} \langle \mathbf{p}_n - \mathbf{p}_i, \mathbf{e}_2 \rangle^2 \ . \end{aligned} \tag{4.1}$$

[1] Conceptually similar alternatives like hierarchical octrees [Elseberg et al., 2013] exist, but are not investigated here.

covariance	Sum	$\lambda_1 + \lambda_2 + \lambda_3$
	Omnivariance	$(\lambda_1 \cdot \lambda_2 \cdot \lambda_3)^{\frac{1}{3}}$
	Eigenentropy	$-\sum_{i=1}^{3} \lambda_i \cdot \ln(\lambda_i)$
	Anisotropy	$(\lambda_1 - \lambda_3)/\lambda_1$
	Planarity	$(\lambda_2 - \lambda_3)/\lambda_1$
	Linearity	$(\lambda_1 - \lambda_2)/\lambda_1$
	Surface Variation	$\lambda_3/(\lambda_1 + \lambda_2 + \lambda_3)$
	Sphericity	λ_3/λ_1
	Verticality	$1 - \lvert\langle[0\,0\,1], \mathbf{e}_3\rangle\rvert$
contour	1st order	O
	2nd order	$\mathsf{m}_{\Uparrow}$
	surface orientation	R
	line feature	Q
height	Vertical range	$z_{\text{max}} - z_{\text{min}}$
	Height below	$z - z_{\text{min}}$
	Height above	$z_{\text{max}} - z$

Table 4.1: Our basic feature set consists of geometric features based on eigenvalues of the local structure tensor, moments around the corresponding eigenvectors, as well as features computed in vertical columns around the point. Features in the center row are explicitly designed to capture contour information.

Strictly, the momentum around the axis $\mathbf{e}_1$ is a 2D moment in the $\mathbf{e}_2\mathbf{e}_3$ plane. However, on smooth surfaces (with an object boundary) there is little variation along the $\mathbf{e}_3$ axis so that we neglect $\mathbf{e}_3$ to obtain a 1D moment. This feature is especially suited to identify occluding and occluded edges following the nomenclature of [Choi et al., 2013], which defines occluding edges as edges, which cast a shadow and dubs edges caused by a shadow occluded edges.

Another property of edges are different orientations of the separated surfaces. In order to identify points on different surfaces, we use the direction of the first eigenvector, expected to point in the direction of the possible contour, and split the neighborhood into two disjoint sets on either side of the contour: $d_{i,n} = \langle\ p_n - p_i,\ \mathbf{e}_2\ \rangle$. The first set $\mathcal{P}^+$ consists of points with a distance $d_{i,n} \geq 0$, while the remaining points are collected in $\mathcal{P}^-$. We take the medoid within each of these subsets and use their scalar product as additional feature:

$$\begin{aligned} &\mathcal{P}^- = \{\mathbf{p}_n \in \mathcal{P}^N | \langle \mathbf{p}_n, \mathbf{e}_2 \rangle < 0\}\ ,\ \mathcal{P}^+ = \mathcal{P}^N \setminus \mathcal{P}^- \\ &R = \left| \left\langle \operatorname*{med}_{n \in \mathcal{P}^-} (\mathbf{n}_n), \operatorname*{med}_{n \in \mathcal{P}^+} (\mathbf{n}_n) \right\rangle \right| . \end{aligned} \tag{4.2}$$

Furthermore we introduce a line feature, where we again make use of the distance of neighboring points to the eigenvector $d_{i,n}$. Points are divided into "near" neighborhood

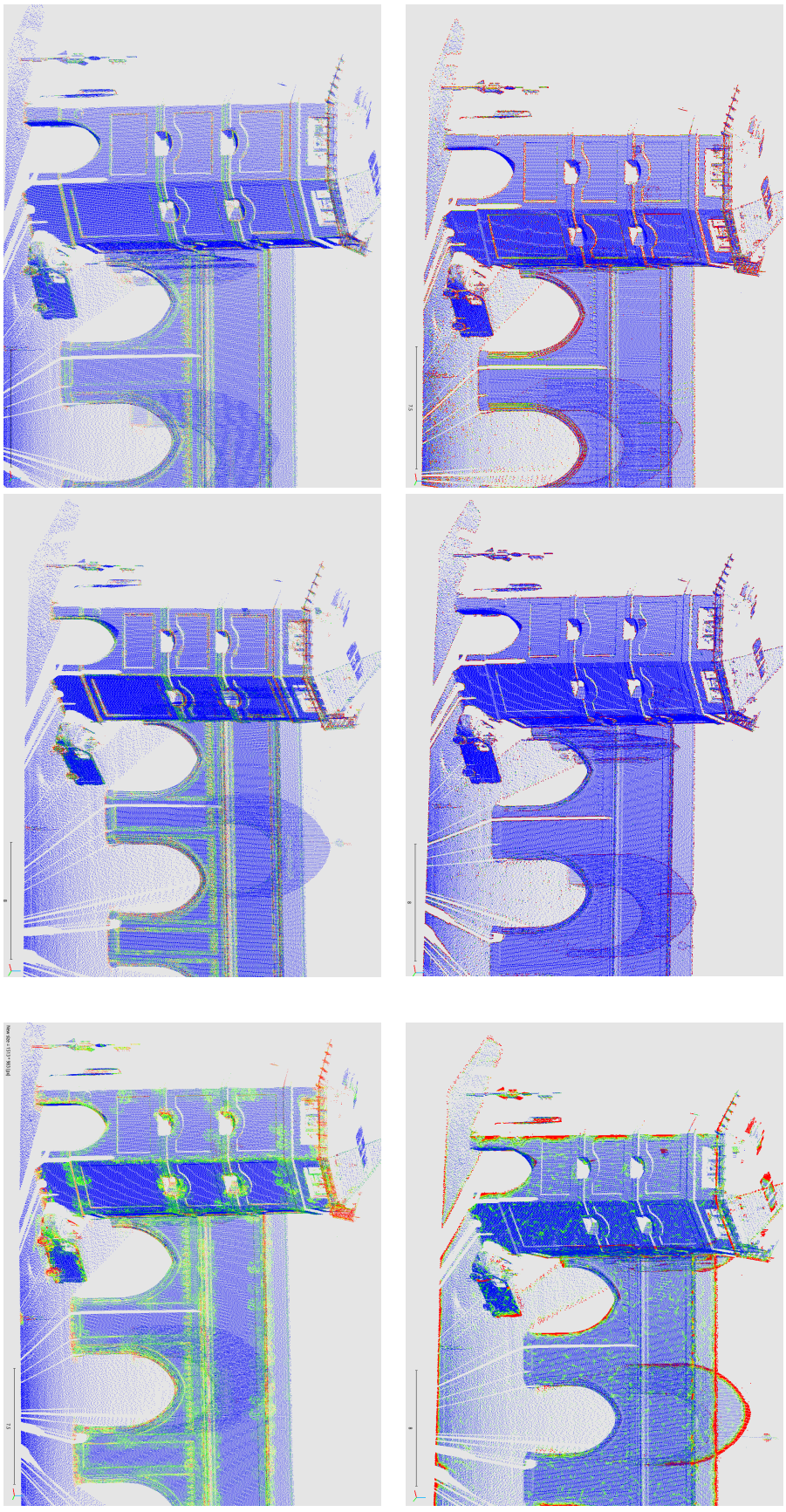


Figure 4.4: *Top Row:* Feature values for O *Bottom Row:* Feature values for $1 - R$ on example point cloud computed at 3 different scales: $\{0.025$ m, 0.05 m, 0.4 m$\}$ from left to right.

$\mathcal{C}_{\mathsf{near}}$ of the $\lceil \alpha \cdot N \rceil$ closest points to the tangent and a "far" neighborhood $\mathcal{C}_{\mathsf{far}}$ consisting of the remaining points. For each point we examine the local surface variation $\gamma_n = \lambda_3/(\lambda_1 + \lambda_2 + \lambda_3)$. The ratio between the average surface variations of the two subsets serves as feature.

$$Q = \frac{\text{mean}_{n \in \mathcal{C}_{\mathsf{near}}}(\gamma_n)}{\text{mean}_{n \in \mathcal{C}_{\mathsf{far}}}(\gamma_n)} \ . \tag{4.3}$$

The local structure tensor is computed from $k = 10$ points (current point plus 9 neighbors) over 9 scale levels, corresponding to voxels of side-length $[0.025, 0.05, 0.1, 0.2, 0.4, 0.8, 1.6, 3.2, 6.4]$ m. The feature set is summarised in table 4.1. A visualization of some features at different scales is given in Fig. 4.4.

4.4.3 Classification and training

For each point $\mathbf{p}_i \in \mathcal{P}$ we compute a feature vector x_i and train a discriminative learner to predict the class probabilities $p(c_i|x_i)$. By taking the $\arg\max_{c_i} \big(p(c_i|x_i)\big)$ we obtain the most likely class for point $\mathbf{p}_i$. We deploy a Random Forest classifier, which is directly applicable to multi-class problems, by construction delivers probabilities, and has been shown to yield good results in reasonable time on large point clouds [Chehata et al., 2009, Weinmann et al., 2015b]. We run grid search with 5-fold cross validation to find the best parametrization of the Random Forest; in our case 50 trees and a depth of approximately 30, with the Gini-index as splitting criterion.

Training on large data sets in 3D space is challenging due to memory restrictions. Furthermore, for polar recordings like those of laser scanners or cameras, the class frequency distribution depends heavily on the instrument's position in the scene. Surfaces near the sensor and perpendicular to the ray direction will contribute much more training samples than distant or slanted objects of the same physical dimensions, hence the distribution learned from a few scans can be severely biased. To resolve this issue in 3D space, a simple solution is to weight errors depending on their distance and angle to the sensor, but to do so the sensor position for every point must still be known. We follow a more pragmatic approach; training data is uniformly down-sampled with voxel grid filtering, so that the number of training samples on objects are approximately independent of the instrument position. As a side effect the number of training samples in densely sampled regions is reduced. Consequently, the training set can cover a larger region (in object space) with the same memory footprint, which is preferrable to ensure the classifier generalises well.

In order to increase computational efficiency at test time, we also scan the Random Forest to identify features that are never used and therefore do not influence the result. These features are not extracted from the test data. Usually almost all covariance-based features are used. Most importantly, the classifier uses at least some features from each

scale level. Since the most expensive computation (even if it is done analytically) is to solve for eigenvector and eigenvalues once per scale, the gains from feature selection are negligible for the basic features from Table 4.1.

Importantly, the memory footprint at test time is small, and independent of the dataset size, since there is no need to store feature vectors. The features x_i of each point $\mathbf{p}_i \in \mathcal{P}$ can be computed on the fly and discarded after classification.

4.5 Contour Extraction

Our contour extraction is a three-stage approach in the spirit of [Türetken et al., 2011, Montoya-Zegarra et al., 2014, Guo et al., 2014]. *(i)* We define what we understand as contour by providing labeled training data and use this data to train a discriminative classifier as described in Section 4.4. This discriminative learner is used to predict class probabilities for a point being a contour or non-contour (background), which we use as low-level evidence for identifying points, which are likely to be a contour; *(ii)* We transition from a set of individual points to connected contour segments by selecting regularly spaced points with high contour scores as seed points, and link them into a connected graph as described in Section 4.5.1; *(iii)* Lastly, a (higher-order) MRF on the over-complete graph of putative contour segments helps to filter out false positives, Section 4.5.2.

4.5.1 Contour candidate generation

An important property of contours is their connectivity over large distances. We follow a hypothesize-and-verify strategy in order to discover long-range interactions. We first generate an over-complete set of contour candidates, and in a second step prune the set to optimal size. This strategy delivers a contour graph in form of many linked line segments, similar to wire-frame models, which do not only consist of line segments but are a collection of linked models, including line segments, b-splines or circles.

We generate contour candidates by selecting points of high contour probabilities $p(c_i = 0|\mathbf{x}_i)$ as seed points, and connecting with 3D shortest path search.

Voxel-grid non-maxima suppression

We uniformly sample seed points in the point cloud, and reduce that initial set with a conservative heuristic for Non-Maxima Suppression (NMS). Non-uniform sampling would risk missing the true contour segment, unless shortest paths are computed in a

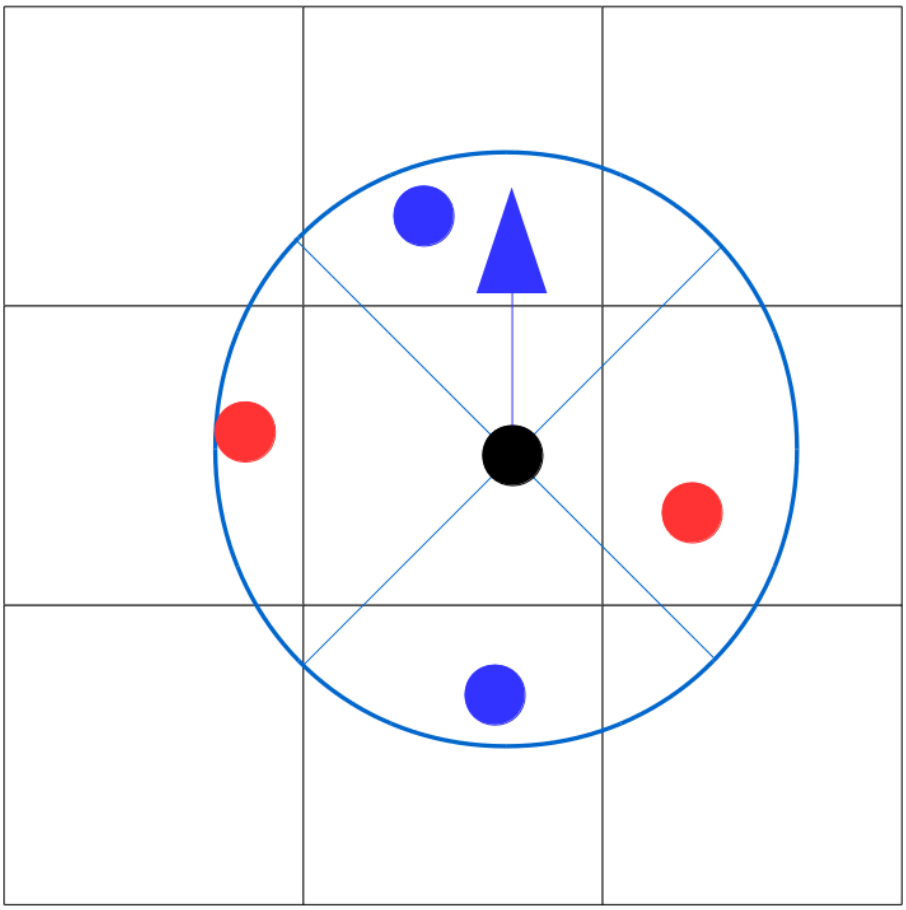


Figure 4.5: Voxel-Non-Maxima Suppression: For each voxel the point with the highest probability is stored. If nearby (blue circle) neighboring seed points have a smaller probability they get suppressed (red points) unless they are located in the direction of the expected contour $\mathbf{e}_1$ and are not too close (red circle).

large neighbourhood around each seed point; which would, in turn, inflate the set of incorrect contour candidates and complicate the classification in Section 4.5.2. For uniform sampling we once again rely on a (sparse) voxel grid (Fig. 4.5), with $s = 0.1\ [m]$ spacing.[2] In a first stage we perform simple thresholding $p(c_i = 0|\mathbf{x}_i) \geq 0.5$ on the most likely point per voxel, so that we find an initial set of potential seeds. In a next step, seed points are pruned that have a neighbour with higher contour likelihood in $\mathcal{P}^N$, unless they are close to the tangent of the contour, assumed to be the eigenvector $\mathbf{e}_1$ of the covariance tensor at the neighbor with the highest likelihood. Finally,

[2]Throughout this section, a number of model parameters need to be chosen. These are mostly distances in metric object coordinates, which are easy to derive for a given application. Values quoted refer to terrestrial laser scans of settlement areas.

points are also removed if they have a more likely neighbour along the tangent within a distance of $< 0.5 \cdot s$.

Graph construction

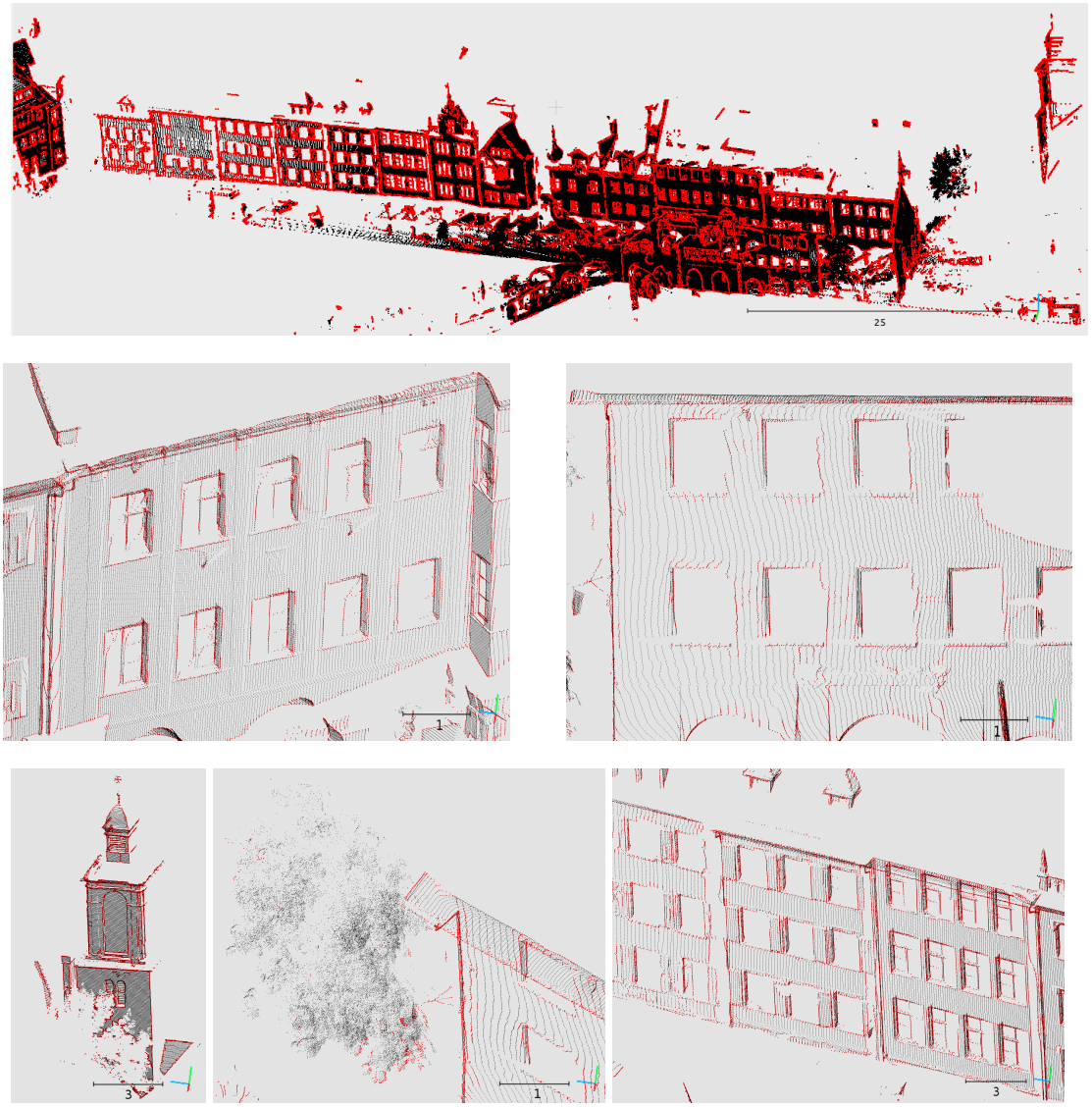


Figure 4.6: "Market Square" Laser-scan. While the horizontal point density decreases rapidly due to grazing scan rays, the vertical density decreases much slower and "scan lines" become visible on the facades on the left. Various zooms are given. Please note that we suppressed responses on vegetation in our training set, so that we intentionally do not want to detect contours in tree crowns.

In a next step a neighborhood graph is generated in order to extract shortest paths between seed points. At this point one must take care to correctly handle anisotropic vari-

ations of the point density, most notably large differences between horizontal and vertical point spacing, which stem from the polar scanning of slanted surfaces, c.f. Fig. 4.6.

Our target is to keep the number of connections, and consequently the memory footprint of the neighborhood graph, small. For this reason we split the creation of the neighborhood graph into two tasks. First, a k-nearest neighbor graph with a small $k_1 = 5$ neighborhood is extracted, which is able to connect uniformly sampled areas but fails in the anisotropic case. Second, disconnected regions are identified and connected. For this purpose a larger neighborhood $k_2 = 50$ is computed for each point. These neighbors are then connected by using their precomputed k_1 neighborhood and reduced to a minimum spanning tree (MST) with Prim's algorithm. The MST edges are then merged with the edges obtained by k_1. Candidate contours are extracted within this neighborhood graph by linking each seed point with a maximum of $\beta = 15$ neighboring seed points in a radius $r_{\text{link}} = 0.85\,\text{m}$ via Dijkstra shortest paths.

The graph edge costs depend on the contour likelihoods of the segment's endpoints, and on the segment's length (to avoid unjustified short-cuts):

$$e_{ij} = 2 - \frac{p(c_i|\mathbf{x}_i) + p(c_j|\mathbf{x}_j)}{2} - \frac{d^*}{\|\mathbf{p}_i - \mathbf{p}_j\|} \,, \tag{4.4}$$

with $\|\mathbf{p}_i - \mathbf{p}_j\|$ the Euclidean length of the edge between $\mathbf{p}_i$, $\mathbf{p}_j$, and d^* the smallest distance from either of the two endpoints $\mathbf{p}_i$, $\mathbf{p}_j$ to all other points. In practice it is unnecessary to perform shortest path search in the full graph containing also nodes with a large distance. Instead, we run it only within a local neighborhood $2 \cdot r_{\text{link}}$. We note that more sophisticated methods exist to construct an optimal graph, for instance using ant colony optimization [Türetken et al., 2011]. In our experience shortest path search is sufficient and yields results of similar quality, while being a lot faster to compute.

4.5.2 Contour edge labeling

The extracted set of contour candidates is designed for high recall, therefore it is overcomplete and contains many false positives. To prune false positives from the contour graph we set up a MRF, where contour scores and spatial arrangement of contour candidates are used for graph labeling. The connectivity of the graph naturally leads to higher-order terms that suppress short isolated contours, minimise the number of free endpoints and encourage closed contours.

Unary Term

Standard MRFs consist of a unary term and one or several prior terms. Our formulation follows this design. The unary term encodes properties of the individual contour candidates, such as statistics over the point wise contour scores or the shape of the candidate. We develop a new descriptor, coined *cumulative shape context*, which adapts the ideas of the original shape context (SC, [Belongie et al., 2002]) to our problem. The main idea of SC is to encode relative positions $\mathbf{v}_{ij} = \mathbf{p}_j - \mathbf{p}_i$ of a point $\mathbf{p}_i$ and its neighbors $\mathbf{p}_j$ in a polar histogram is directly applicable to 3D space. However, contours do not consist of a single point $\mathbf{p}_i$ but of many, therefore all points on a contour should be considered in order to obtain a discriminative descriptor of a putative contour's shape. An additional requirement for our descriptor is scale and rotation invariance. To achieve scale invariance and at the same time simplify the descriptor to 1D, we discard relative distance and encode only directions, by normalising the vectors. Rotation invariance is achieved by projecting onto the canonical direction $\mathbf{v}_{se} = \mathbf{p}_e - \mathbf{p}_s$ defined by the start point $\mathbf{p}_s$ and end point $\mathbf{p}_e$ of the contour candidate.

Information from all points along the contour candidate is merged by simply summing their individual histograms into a single one: we visit every point $\mathbf{p}_i$ in turn, and generate a vector v_{ij} to each other point of the contour:

$$v_{ij} = \left\| \left\langle \frac{\mathbf{v}_{se}}{\|\mathbf{v}_{se}\|}, \frac{\mathbf{v}_{ij}}{\|\mathbf{v}_{ij}\|} \right\rangle \right\| \quad \forall\, i,j \in \{s,\ldots,e\} : i \neq j \ . \tag{4.5}$$

v_{ij} is a 1D value and is added to a histogram with 5 equally spaced bins. To encode overall straightness / curvature of the contour candidate, we compute the same histogram again, but this time normalised by projection onto tangent (first eigenvector $\mathbf{e}_{1,i}$):

$$w_{ij} = \left\| \left\langle \mathbf{e}_{1,i}, \frac{\mathbf{v}_{ij}}{\|\mathbf{v}_{ij}\|} \right\rangle \right\| \quad \forall\, i,j \in \{s,\ldots,e\} : i \neq j \ , \tag{4.6}$$

The bin counts of both histograms $\mathbf{z}_1(v_{ij})$ and $\mathbf{z}_2(w_{ij})$ form the first part of our feature vector to discriminatively learn the unary term. Moreover, the feature vector also includes the contour likelihoods $p(c_i|\mathbf{x}_i)$ along the sequence of points $\mathbf{p}_i$ along the contour segment, again in the form of two discrete distributions. The first one quantises the class probabilities into a global histogram with 5 bins centered at the probabilities $\{0.1, 0.3, 0.5, 0.7, 0.9\}$. The second one retains some spatial information, by chopping the contour sequence into 5 segments with equal number of points, and computing their mean probabilities. Given these features $\mathbf{z}_i$ we again employ a random forest to learn the probabilities $p(g|\mathbf{z})$ that a candidate segment is part of a true contour ($g_i = 1$) or a false alarm ($g_i = 0$), and pass the log-likelihoods $h_i = -\log p(g_i|\mathbf{z}_i)$ as unary term into the MRF.

Method	$\overline{IoU}$	OA	$t[s]$	IoU_1	IoU_2	IoU_3	IoU_4	IoU_5	IoU_6	IoU_7	IoU_8
Our	0.494	0.850	38421	0.911	0.695	0.328	0.216	0.876	0.259	0.113	0.553
HarrisNet	0.623	0.881	unknown	0.818	0.737	0.742	0.625	0.927	0.283	0.178	0.671
DeepSegNet	0.516	0.884	unknown	0.894	0.811	0.590	0.441	0.853	0.303	0.190	0.050
Image Based	0.391	0.745	unknown	0.804	0.661	0.423	0.412	0.647	0.124	0.	0.058

Table 4.2: Semantic3d benchmark results on the full data set with a training set of $\sim 1.7 \cdot 10^9$ points and a test set of $\sim 2.3 \cdot 10^9$ points.

Method	$\overline{IoU}$	OA	$t[s]$	IoU_1	IoU_2	IoU_3	IoU_4	IoU_5	IoU_6	IoU_7	IoU_8
Our	0.542	0.862	1800	0.898	0.745	0.537	0.268	0.888	0.189	0.364	0.447
DeepNet	0.437	0.772	64800	0.838	0.385	0.548	0.085	0.841	0.151	0.223	0.423
Image Based	0.384	0.740	unknown	0.726	0.73	0.485	0.224	0.707	0.050	0.0	0.15

Table 4.3: Semantic3d benchmark results on the reduced data set with the same training set as the full challenge and a test set of $\sim 8 \cdot 10^7$ points.

Markov Random Field

By definition, contours should be long connected line features rather than independent, short segments. We treat the contour candidates as variables (nodes) in a graph and encourage long chains of contour segments with a simple Potts-type prior, which aims to to suppress segments that cannot be connected, and encourage segments that close a gap between their start and end nodes, linking up other candidates into longer contours ("start" and "end" are used for linguistic convenience, the graph is undirected). Each contour candidate l_i defines a clique ℓ_i that also includes all other candidates with which it shares either the start node or the end node. Denoting by $\mathcal{L}_i^s$ the set of contours that connect to l_i at the start node, and by $\mathcal{L}_i^e$ those which meet l_i at the end node, we have

$$\ell_i = \begin{cases} \gamma & g_i = 1 \textbf{ AND } \forall\, \mathrm{l}_j \in \mathcal{L}_i^s, \mathcal{L}_i^e : g_j = 0 \\ & \text{(isolated contour)} \\ \delta & g_i = 1 \textbf{ AND } (\exists\, \mathrm{l}_j \in \mathcal{L}_i^s : g_j = 1 \textbf{ XOR } \exists\, \mathrm{l}_j \in \mathcal{L}_i^e : g_j = 1) \\ & \text{(continuation only on one side)} \\ 0 & g_i = 1 \textbf{ AND } \exists\, \mathrm{l}_j \in \mathcal{L}_i^s : g_j = 1 \textbf{ AND } \exists\, \mathrm{l}_j \in \mathcal{L}_i^e : g_j = 1 \\ & \text{(continuation on both sides)} \\ 0 & g_i = 0 \\ & \text{(candidate is not a contour)} \end{cases} \tag{4.7}$$

These cliques are higher-order terms over a variable number of segments, but they can be computed efficiently, since they only depend on direct neighbors. Higher-order render MRF inference expensive, fortunately it turns out that satisfactory minima of

the energy

$$E = \sum_i h_i + \sum_i \ell_i. \tag{4.8}$$

can be found with the Iterated Conditional Modes (ICM) algorithm [Besag, 1986], which we use for its computational efficiency [Kappes et al., 2013].

Our graph is constructed such that overlapping candidates are not penalized. Overlaps are needed to correctly recover T-junctions and crossings that do not coincide with a seed point. They are easily resolved in post-processing, since the overlapping segments share the same vertices.

4.5.3 Postprocessing

The desired output of the contour detector is a wireframe with few vertices linked by long, straight segments, which can be further processed with conventional CAD software. To get closer to such a minimal representation, while at the same time reducing measurement noise, we employ line simplification. Following [Mackaness and Mackechnie, 1999] we separate the simplification of junctions from the simplification of edges. Junction simplification is implicitly handled in our framework, via the MRF prior (Sec. 4.5.2). For contour simplification between junctions, we found that the classic Douglas-Peuker algorithm yields good results [Shi and Cheung, 2006].

4.6 Experiments

We have conducted experiments on the *semantic3d.net* semantic 3D labeling challenge (`www.semantic3d.net`). Since contour annotations are not available for that dataset, we also test on a smaller set of terrestrial laser scans with manually labeled contours [3].

The *semantic3d.net* benchmark provides terrestrial laser scans from a variety of different urban and rural scenes, as well as hand-labeled ground truth for 8 classes. The point clouds are large: they have been recorded with state-of-the-art terrestrial instruments and have a realistic (i.e., high) point density. The training set consists of $\approx 1.7 \cdot 10^9$ points, and the test set of $\approx 2.3 \cdot 10^9$ points[4].

The qualitatively similar, but smaller scan data set from [Hackel et al., 2016a] also features high-density terrestrial scans from various geographic locations, and includes

[3] The same dataset has alreadt been used in [Hackel et al., 2016b].

[4] A reduced test set with $\approx 79 \cdot 10^6$ points is also available, to allow submissions from computationally very demanding algorithms. We run on both versions for completeness

ground truth contour dense annotations for one scan, which is exclusively used for evaluation of the entire framework as well as 14 sparsely labeled point clouds, of which five were used for training while the rest was used for the evaluation of point-wise contour evidence.

Our software is implemented in *C++* using the *Point Cloud Library* (`pointclouds.org`), *FLANN* (`github.com/mariusmuja/flann`) for nearest-neighbor search and the *ETH Random Forest Templates*[5] for discriminative learning. All experiments were run on a standard desktop machine with Intel Xeon E5-1650 CPU (hexa-core, 3.5 GHz) and 64 GB RAM. Timings always refer to that hardware configuration. We do parallelize across multiple cores with the help of *OpenMP* (`http://openmp.llvm.org`), but note that a significant speed-up would still be possible with extreme parallelism on the GPU.

4.6.1 Experiments on the Semantic3d.net Benchmark

We reduce the very large number of training samples in two ways: *(i)* The sub-sampling as described in section 4.4.3 is applied with a spacing of 1cm; *(ii)* We distort the class frequencies so that the largest class has at maximum $4\times$ the number of training samples of the smallest class, by randomly picking training samples per class, which leads to a total of 10^6 samples for training.

The depth of the Random Forest is estimated with Grid Search and 5-Fold Cross-Validation over a parameter range from 30 to 60 with a stride length of 3. In general more trees yield slightly better results, but also linearly increase the computational cost. The number of trees was empirically set to 40 and the Gini index was used as splitting criterion for interior tree nodes. Training and grid search took less than one day for the 1 million training samples. The time given is the time needed to classify the test set and includes loading data from hard disk, feature extraction, classification and saving of results.

The main metric of the benchmark is Intersection over Union (IoU), and its average $\overline{IoU}$ over all classes. IoU is defined as follows:

$$IoU_i = \frac{c_{ii}}{c_{ii} + \sum_{i \neq j} c_{ij} + \sum_{i \neq k} c_{ki}}, \tag{4.9}$$

where c_{ij} is the count of class i being predicted as j. IoU is a a more sensitive error measure that does not saturate as quickly as alternative metrics. It gives all classes equal weight independent of their point count, and decreases faster than the F_1-score or

[5] `https://edit.ethz.ch/igp/photogrammetry/downloads/rforest.zip`

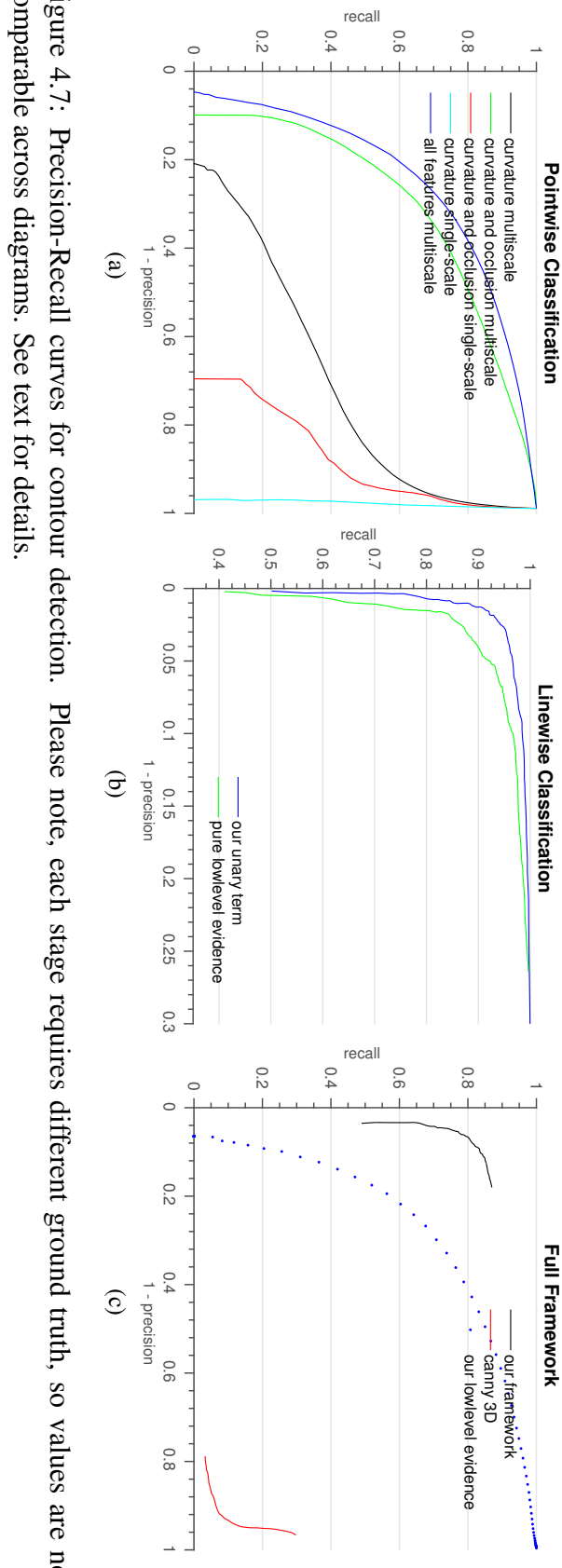


Figure 4.7: Precision-Recall curves for contour detection. Please note, each stage requires different ground truth, so values are not comparable across diagrams. See text for details.

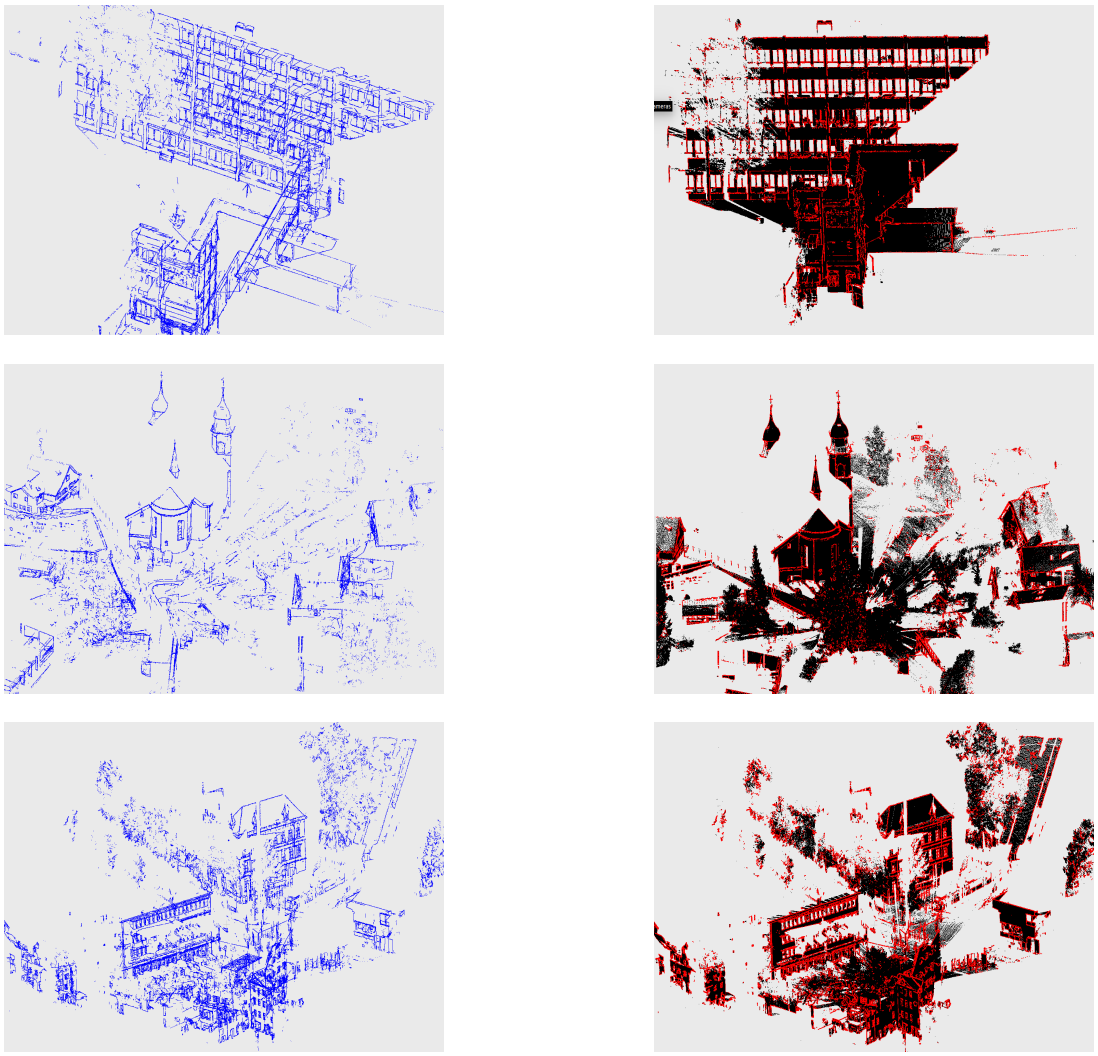


Figure 4.8: Visual evaluation 1 to 3: *left column:* detected points on contour (blue); *right column:* original point cloud (black) with detected contour (red).

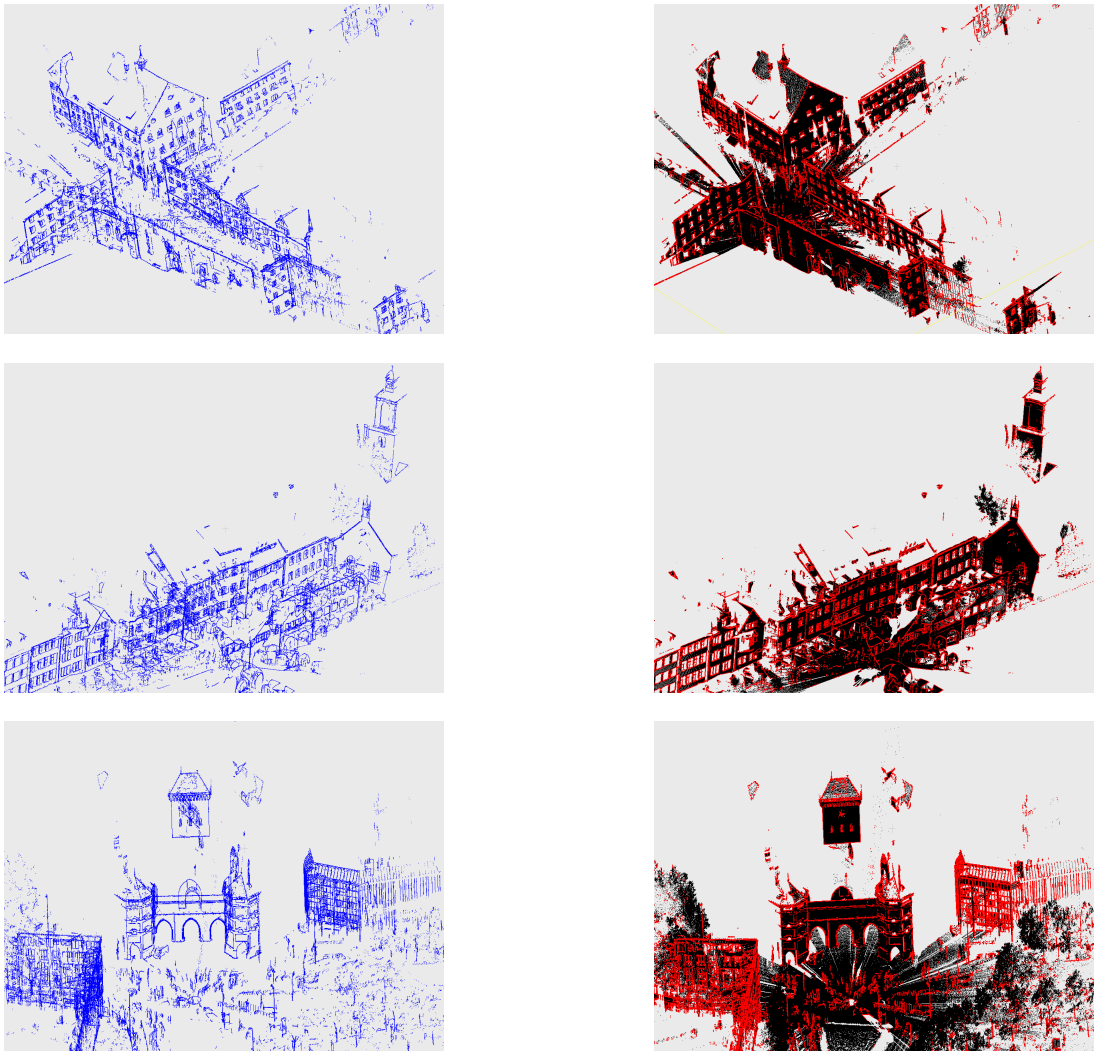


Figure 4.9: Visual evaluation 4 to 6: *left column:* detected points on contour (blue); *right column:* original point cloud (black) with detected contour(red).

average class accuracy as the number of false positives (type-1 errors) or false negatives (type-2 errors) increases. Additionally, the overall accuracy (OA) and the runtime for the complete test set are given as auxiliary measures.

Table 4.2 shows the results of our method on the full data set, while table 4.3 compares our method with other baselines on the reduced data set. Recently, there have been submission based on deep neural networks (HarrisNet, DeepNet) with a higher $\overline{IoU}$, which comes with high computational costs and their methods are not published, yet. Nevertheless, we reach higher $\overline{IoU}$ and faster runtimes than competitors based on traditional machine learning, as shown in [Hackel et al., 2016b]. We note that our method not only achieves good quantitative results, but also offers practically viable runtimes even with the current prototype software: classifying new point clouds takes less than

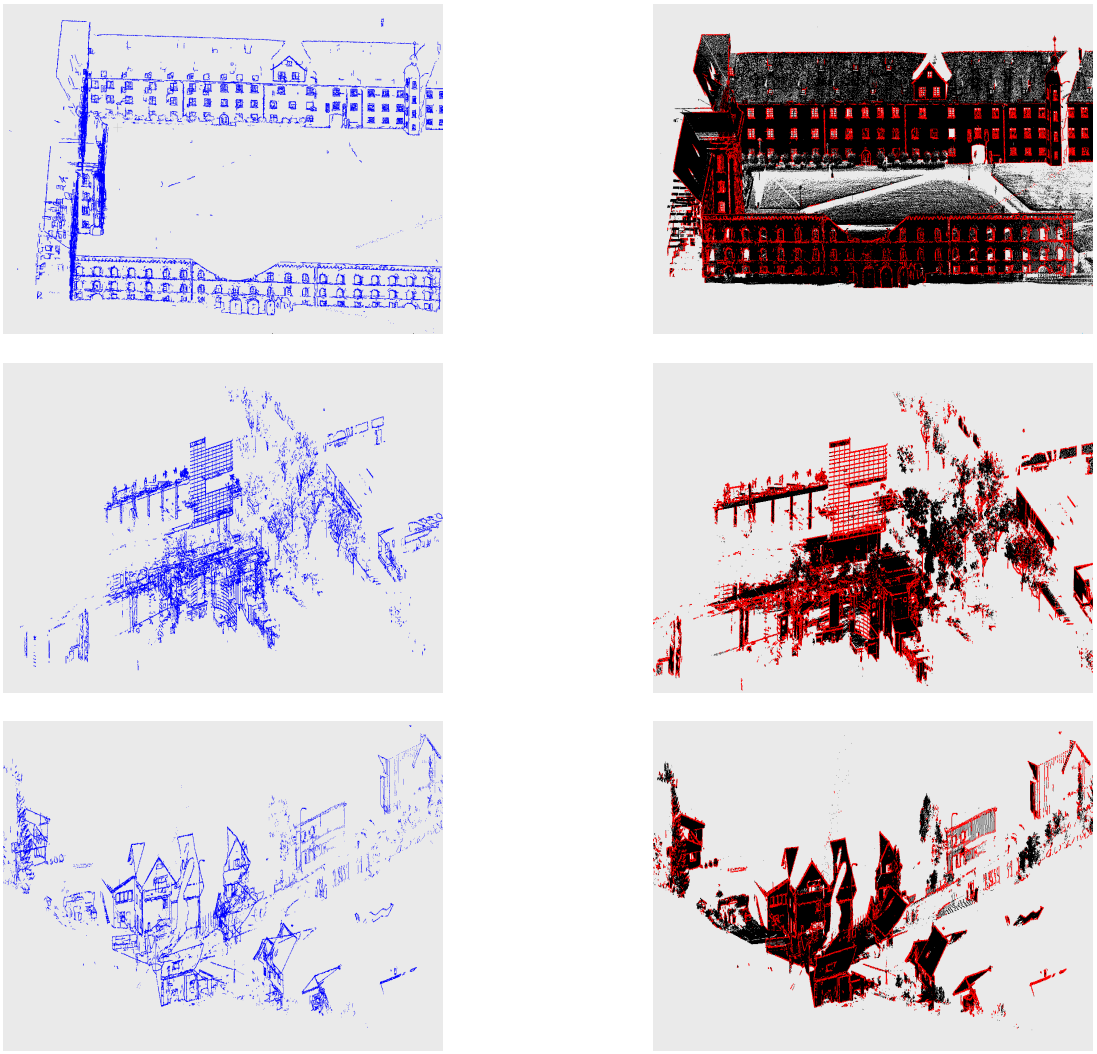


Figure 4.10: Visual evaluation 7 to 9: *left column:* detected points on contour (blue); *right column:* original point cloud (black) with detected contour (red).

3 minutes per 10 million points.

4.6.2 Contour Extraction

A database of 16 laser scans of urban scenes in different countries were used for qualitative evaluation of the contour extraction framework. One scan was manually labeled to serve as ground truth, with $101'614$ points on contours and $7'681'061$ background points. The only baseline for which we could find a 3rd-party implementation is the Canny-style 3D edge detector in the *Point Cloud Library*. In order to facilitate a meaningful comparison, the *tree* and *clutter* classes are excluded. We are forced to do this to avoid a complete failure of the baseline, which otherwise generates huge amounts of

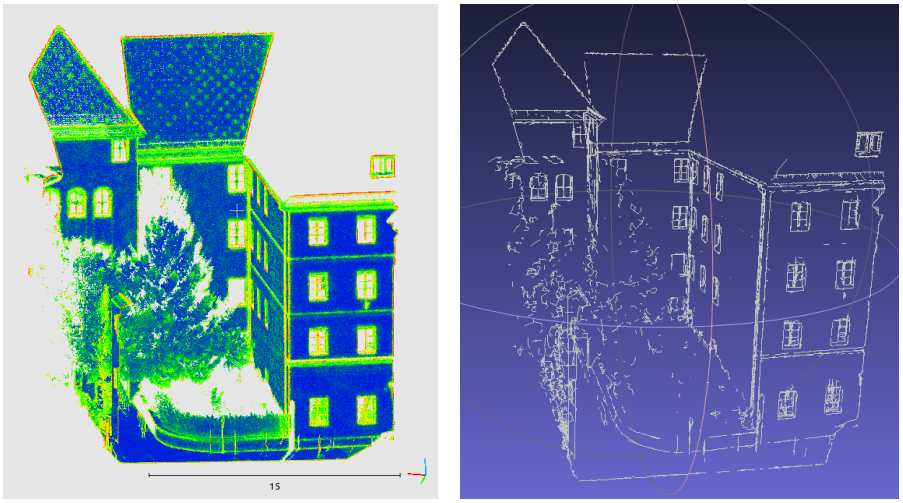


Figure 4.11: Point wise score and final result.

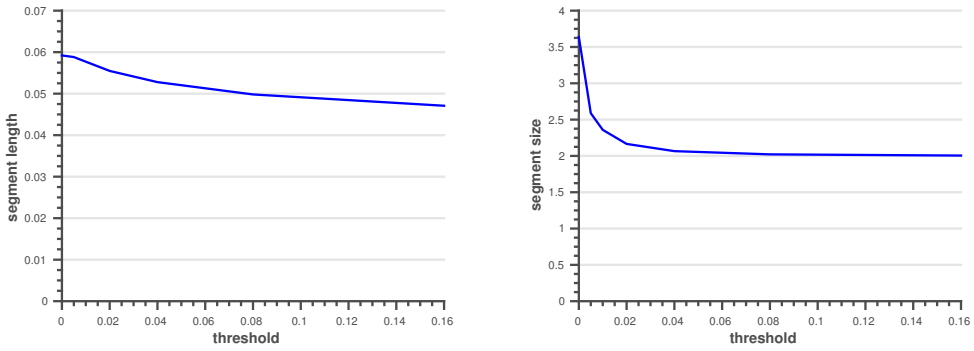


Figure 4.12: The statistics show how the average length of contour segments changes by applying douglas peucker. Both graphs show that the length decreases slowly while the number of points on the contour is reduced significantly, which is a desired behavior. The threshold is the parameter of the Douglas-Peucker-algorithm.

false positives on those classes. Making the task easier by excluding the worst distactors is a significant bias *against* our integrated method: it has access to the object class likelihoods, precisely *because* it jointly addresses contours and semantic segmentation.

Pointwise Classification

In a first experiment the influence of semantic classification on the detection of contours was tested with a focus on different feature sets. We also evaluated the performance of multi-scale feature sets versus single-scale features. We trained our Random Forest on $48,964$ positive and $85,853$ negative examples, captured in different cities, and use precision-recall curves for the evaluation. The performance of the different

feature sets is shown in Fig. 4.7a. Our feature subsets cover the features used in related methods, which either use curvature as feature or combine curvature with occlusion features. In particular, single-scale curvature and occlusion form the basis of RGB-D edge detection in [Choi et al., 2013], while multi-scale curvature is the descriptor used by [Pauly et al., 2003]. In our comparison, we train a Random Forest for each of the feature subsets, which can be expected to yield at least as good results as a single threshold per feature dimension. This experiment shows that single-scale detection does not perform well on large and complex point clouds. Multi-scale curvature alone still misses many clear and unambiguous contours, particularly along occlusion boundaries. Our full feature set performs better than the best feature subset and significantly better than the baseline method.

Linewise Classification

We also assess the performance of the unary term for contour segment classification. The classifier was trained on a set of 1862 true contour candidates and 832 negatives. Note that each of these candidates consists of multiple points. We again used grid search with the same settings as in the previous experiment for training. The evaluation was performed on a test set with 2227 positive and 1013 negative samples. Here, we compare against the naive baseline feature set consisting only the per-point contour likelihoods $p(c_i = 0|x_i)$, where we use the histograms from the full feature vector. The results are shown in Fig. 4.7b. It can be seen that adding information about the shape of contour candidates improves the identification of actual contours up to ≈20 percent points, especially in the high-precision regime.

Full Framework

Terrestrial Laser Scans	All features			no contour features		
	Recall	Precision	IoU	Recall	Precision	IoU
Man made terrain	0.8758	0.9433	0.9083	0.8757	0.9627	0.9171
Natural terrain	0.8809	0.8921	0.8864	0.9064	0.8774	0.8916
High vegetation	0.9129	0.8102	0.8585	0.8086	0.9254	0.8631
Low vegetation	0.6496	0.4021	0.4967	0.6356	0.5364	0.5818
Buildings	0.9592	0.9878	0.9733	0.9726	0.9813	0.9769
Remaining hard scape	0.7879	0.4878	0.6025	0.8419	0.4743	0.6068
Scanning artefacts	0.5127	0.4595	0.4847	0.3995	0.5039	0.4457
Overall accuracy		0.9215			0.9187	
$\overline{IoU}$-score		0.6514			0.646	

Table 4.4: Quantitative results for terrestrial laser scans.

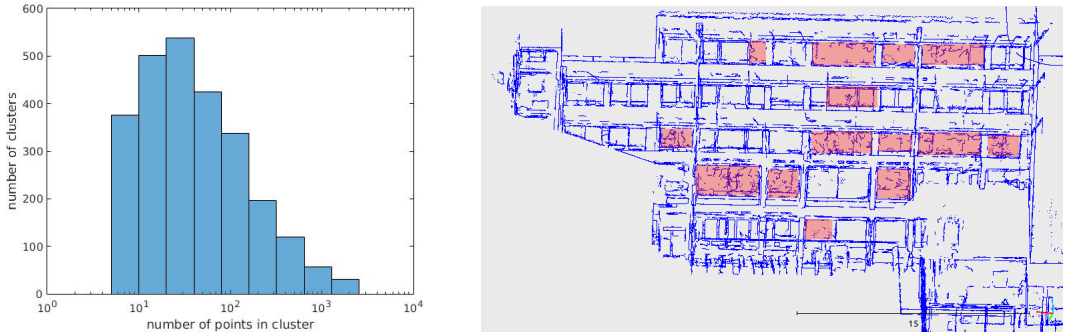


Figure 4.13: *left)* Histogram over points per contour segments, not broken up at junctions. *right)* A failure case: if objects with strong geometric structure did not appear in the training set, then they tend to get high contour scores. Red highlights windows with sun blinds. Sun blinds were not covered in our training set and cause many false positives.

We evaluate our complete contour detector against the 3D Canny of [Choi et al., 2013], which is the only published competitor in recent literature, and the only method for which we found an accessible implementation (in *PCL*). This method requires "organised point clouds", which are essentially the same as height maps, i.e. they have a 2.5D grid structure. We convert individual scans to "organised clouds" by cube-mapping, noting that 3D Canny in the proposed form is not applicable if the sensor position is unknown, or when the point cloud has been recorded dynamically from a moving platform.

At this point the evaluation faces a subtle, but important problem: the Canny baseline does not return line segments, but only a list of all points that form part of a line. For this reason, our evaluation is only based on point-wise scores. However, our full framework is designed to extract contours in form of a graph, and (during NMS and shortest path search) intentionally discards many point that are not needed to trace the contour. We emulate this sub-sampling of the points along detected contours by ignoring all false negatives with lie within $3\,\text{cm}$ of either a true positive or an already counted false negative. With this filtering we account for the dropped points. Gaps larger than $6\,\text{cm}$ still count as false negatives, and alarms where there is no ground truth contour still count as false negatives. As a consequence of the filtering, the graph in Fig. 4.7c is not a true precision-recall curve over the complete, original point cloud, but it displays the most meaningful comparison we could come up with. One can see that the proposed per-contour scores based on contour likelihoods and shape consistently beat the pure likelihoods , and that our higher-order context model (including the connectivity prior) greatly outperforms the discriminatively trained low-level evidence. The latter already works dramatically better than the 3D Canny, although we made our best effort to tune the baseline for maximum performance.

The final output of our contour detector is a graph of contour edges which connect vertices that are 3D points from the original point cloud as shown in (Fig. 4.11). In that graph, the average length of a single contour is 187 points, respectively $3.7\,\mathrm{m}$ in metric world units for our specific urban outdoor scan data. The median is 28 points, respectively $0.5\,\mathrm{m}$. The full histogram of the extracted contours' length is also shown in Fig. 4.13. For post-processing we split up the graph at junctions and smooth individual segments with the Douglas Peucker algorithm. The effect on the segment length between two junctions is shown in (Fig. 4.12). It can be seen that line simplification has only a small impact on the segment length but a large impact on the average number of vertices and quickly converges to 2, the minimum number of vertices.

Additionally, several point clouds with in total more than $3 \cdot 10^8$ points were evaluated visually. Representative results are shown in Fig. 4.8 to Fig. 4.10.

Failure cases

Errors of our detector are mainly caused by two effects: *(i)* if there are independent contours with a small distance ($< s$) the framework will hallucinate erroneous connections between them; *(ii)* if the classifier encounters unusual point configurations dissimilar to those seen in the training data, it tends to produce false positives.

4.6.3 Impact of contour features on multi-class classification

In order to show the impact of contours on the performance of a multi class classifier we perform an experiment based on the data set of [Hackel et al., 2016b]. The training set consists of $\sim 2 \cdot 10^8$ points with 7 classes, while the test set consists of $\sim 3 \cdot 10^8$ points. We train two different classifiers. The first version contains all features designed for contour detection and multi class classification. The second version uses only the subset of features, which have not been explicitly designed for contour detection. This includes $\mathcal{O}$, R, Q and $\mathsf{m}_{\Uparrow}$. We again use grid search and 5-fold cross-validation to estimate the depth of the Random Forest and train with Gini-impurity as splitting criterion. On this data set the full feature set performs slightly better than the one without contour features; $\sim 0.5\%$ in $\overline{IoU}$ amd $\sim 0.3\%$ in Overall Accuracy. This shows that multi-class classification slightly benefits from features, which encode contour information.

and semantic classification (pretrained classifiers).

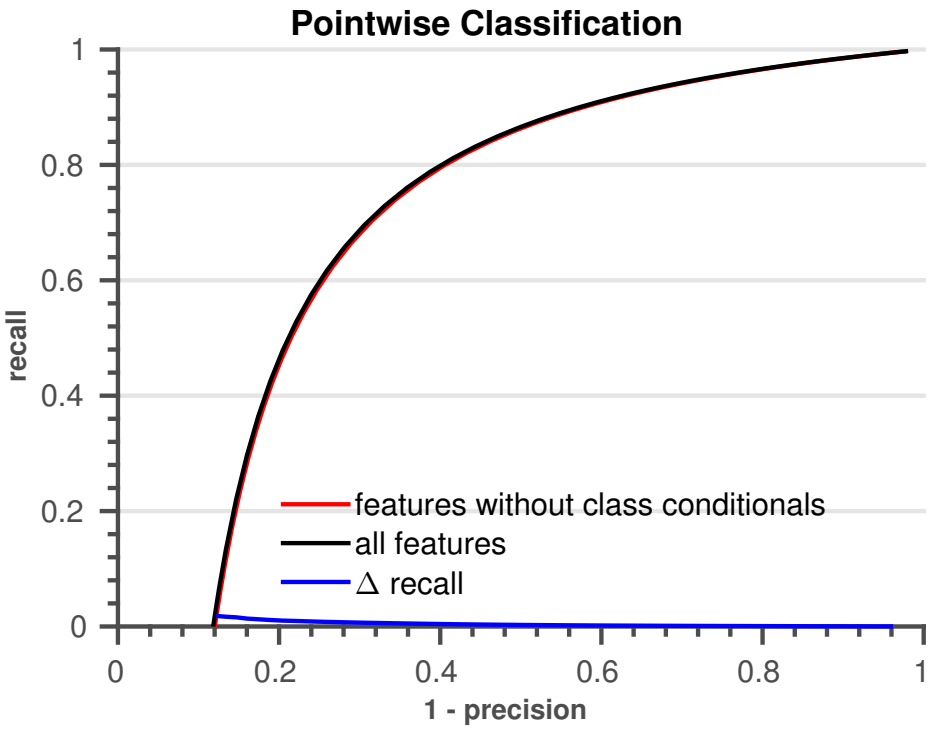


Figure 4.14: Precision Recall for pointwise contour labeling.

4.6.4 Impact of multi-class classification on contours

Recall that we have labeled contours (to train the contour classifier) only for a relatively small set of data compared to the size of the dataset used for multi-class classification, because labeling contours in 3D space is tedious work. In order to still make use of the information contained in the entire multi-class training data, the feature vector of the contour detection was extended with class conditional probabilities of the multi-class classifier. The assumption is that (i) these additional probabilities from the full, much larger training data set might help the contour classifier to better generalize and that (ii) transitions between specific classes (e.g., roof and facade) might be a strong hint at contours.

To verify the benefit of adding probabilities from the multi-class classifier, we perform an experiment similar to Fig.4.7a using the data sets from section 4.6.2. It turns out that, in fact, contour detection benefits only slightly from the multi-class classifier. Small performance improvement by few percent points is recognizable in the high precision region in Fig. 4.14, whereas the gain is marginal in low precision regions. Multi-class scores from the full training dataset seem to add only few extra evidence in our case. This outcome shows that, although rather small, the training dataset seems to be sufficiently large to implicitly learn semantic classes for contour detection even without any direct feedback from the multi-class classifier. Although it is only a bi-

nary classification problem, the contour detector learns to neglect potential contours on vegetation, e.g. trees, while firing on contours at building edges.

4.7 Conclusion

We have described a novel approach for semantic classification and contour extraction, which is able to handle large data sets of unstructured point clouds very efficiently while achieving state-of-the-art performance. Because contours are ill-defined and not easy to explicitly describe with a set of ad-hoc rules, we prefer to also cast contour detection as a machine learning problem and let the classifier learn relevant contour evidence directly from the data. Contour detection benefits significantly from prior point-wise semantic classification. Our methods are fast enough for point clouds of realistic size. Moreover, they reach practically interesting accuracies: precision *and* recall of semantic segmentation are $> 85\%$ for most major object classes, which is well above the often quoted usability threshold of 80%. And contour detection reaches $\approx 85\%$ recall at 90% precision, which also is a healthy basis for subsequent interative modelling in a CAD system. We thus hope to soon see widespread deployment of automatic interpretation methods in point cloud processing software.

Chapter 5

Large-scale Supervised Learning for 3D Point Cloud Labeling: Semantic3D.net

Abstract

In this paper we review current state-of-the-art in 3D point cloud classification, present a new 3D point cloud classification benchmark data set of single scans with over four billion manually labelled points, and discuss first available results on the benchmark. Much of the stunning recent progress in 2D image interpretation can be attributed to the availability of large amounts of training data, which have enabled the (supervised) learning of deep neural networks. With the data set presented in this paper, we aim to boost the performance of CNNs also for 3D point cloud labelling. Our hope is that this will lead to a breakthrough of deep learning also for 3D (geo-)data. The *semantic3D.net* data set consists of dense point clouds acquired with static terrestrial laser scanners. It contains 8 semantic classes and covers a wide range of urban outdoor scenes, including churches, streets, railroad tracks, squares, villages, soccer fields and castles. We describe our labelling interface and show that, compared to those already available to the research community, our data set provides denser and more complete point clouds, with a much higher overall number of labelled points. We further provide descriptions of baseline methods and of the first independent submissions, which are indeed based on CNNs, and already show remarkable improvements over prior art. We hope that *semantic3D.net* will pave the way for deep learning in 3D point cloud analysis, and for 3D representation learning in general.

5.1 Introduction

Neural networks have made a spectacular comeback in image analysis since the seminal paper of [Krizhevsky et al., 2012], which revives earlier work of [Fukushima, 1980, LeCun et al., 1989]. Especially deep convolutional neural networks (CNNs) have quickly become the core technique for a whole range of learning-based image analysis tasks. The large majority of state-of-the-art methods in computer vision and machine learning now include CNNs as one of their essential components. Their success for image-interpretation tasks is mainly due to (i) easily parallelisable network architectures that facilitate training from millions of images on a single GPU and (ii) the availability of huge public benchmark data sets like *ImageNet* [Deng et al., 2009, Russakovsky et al., 2015] and *Pascal VOC* [Everingham et al., 2010] for rgb images, or *SUN RGB-D* [Song et al., 2015] for rgb-d data.

While CNNs have been a great success story for image interpretation, they have not yet made a comparable impact for 3D point cloud interpretation. What makes supervised learning hard for 3D point clouds is the sheer size of millions of points per data set, and the irregular, not grid-aligned, and in places very sparse distribution of the data, with strongly varying point density (Figure 5.1).

While recording point clouds is nowadays straight-forward, the main bottleneck is to generate enough manually labeled training data, needed for contemporary (deep) machine learning to learn good models, that generalize well across new, unseen scenes. Due to the additional dimension, the number of classifier parameters is larger in 3D space than in 2D, and specific 3D effects like occlusion or variations in point density lead to many different patterns for identical output classes. This makes it harder to train good classifiers, so it can be expected that even more training data that in 2D is needed[1]. In contrast to images, which are fairly easy to annotate even for untrained users, 3D point clouds are harder to interpret. Navigation in 3D is more time-consuming and the strongly varying point density aggravates scene interpretation.

In order to accelerate the development of powerful algorithms for point cloud processing[2], we provide the (to our knowledge) hitherto largest collection of individual, non-overlapping terrestrial laser scans with point-level semantic ground truth annotation. In total, it consists of over $4 \cdot 10^9$ points, labelled into 8 classes. The data set is split

[1]The number of 3D points of *semantic3d.net* (4×10^9 points) is at the same scale as the number of pixels of the *SUN RGB-D* benchmark ($\approx 3.3 \times 10^9$ px) [Song et al., 2015], which aims at 3D object classification. However, the number of 3D points per laser scan ($\approx 4 \times 10^8$ points), and thus the variability in point density, object scale etc. is considerably larger than the number of pixels per image ($\approx 4 \times 10^5$ px).

[2]Note that, besides laser scanner point clouds, it is also sometimes preferred to classify point clouds generated via structure-from-motion directly instead of going back to the individual images and then merging the results [Riemenschneider et al., 2014].

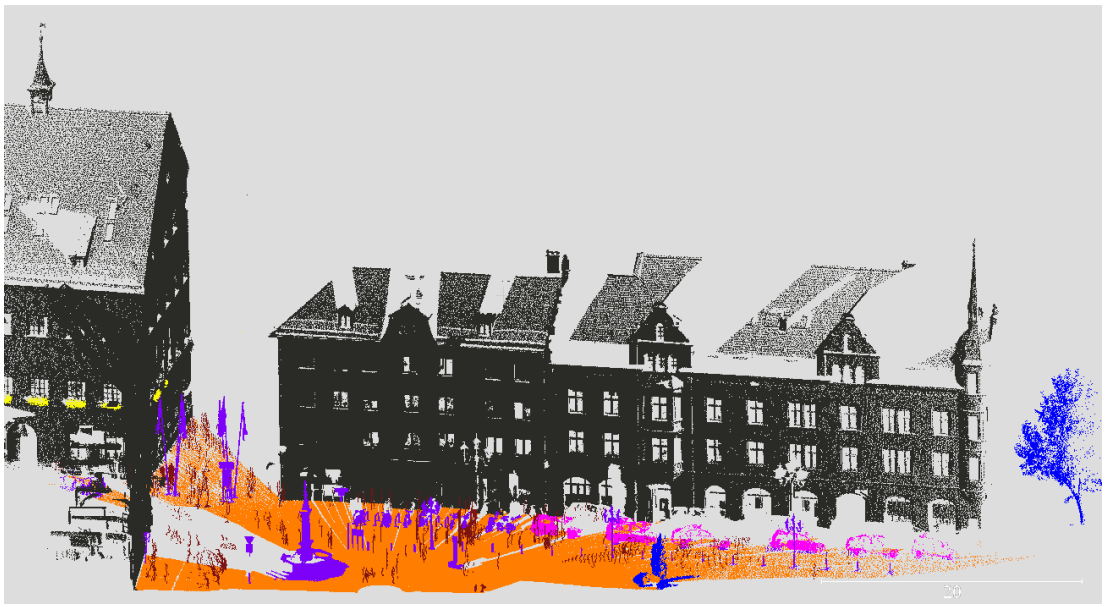


Figure 5.1: Example point cloud from the benchmark dataset, where colours indicate class labels.

into training and test sets of approximately equal size, without any overlap between train and test scenes. The scans are challenging, not only due to their realistic size of up to $\approx 4 \cdot 10^8$ points per scan, but also because of their high angular resolution and long measurement range, leading to extreme density changes and large occlusions. For convenient use of the benchmark, we provide not only freely available data and ground truth, but also an automated online submission system, as well as evaluation tables for the submitted methods. The benchmark also includes baselines, both for the conventional pipeline consisting of eigenvalue-based feature extraction at multiple scales followed by classification with a random forest, and for a basic deep learning approach. Moreover, we briefly discuss the first submissions to the benchmark, which so far all employ deep learning. This article is an extended version of the conference paper [Hackel et al., 2017a]. Here, we add a more thorough review of related work, with emphasis on the most recent 3D-CNN methods. We also provide descriptions of the two latest CNN-based submissions, which lead the comparison by a significant margin, and seem to confirm that, also for point cloud anaysis, deep learning is the most powerful technology developed to date.

5.2 Related Work

Here, we first review traditional methods for point cloud segmentation before discussing novel deep learning-based methods for this task. Finally, we review existing

benchmark activities and motivate the introduction of our new 3D point cloud benchmark for semantic segmentation.

5.2.1 Point cloud segmentation

Early work on semantic point cloud segmentation transformed the points (recorded from airborne platforms) into other representations such as regular raster height maps, in order to simplify the problem and benefit from the comprehensive toolbox of image processing functions [Hug and Wehr, 1997, Maas, 1999, Haala et al., 1998, Rottensteiner and Briese, 2002, Lodha et al., 2006]. Much of the pioneering work on true 3D (i.e., not 2.5D) point cloud processing was developed to guide autonomous outdoor robots [Vandapel et al., 2004, Manduchi et al., 2005, Montemerlo and Thrun, 2006, Lalonde et al., 2006, Munoz et al., 2009b] that rely on laser scanners to acquire data of their surroundings.

In general, it is advantageous if scene interpretation directly operates on 3D points, both for aerial [Charaniya et al., 2004, Chehata et al., 2009, Niemeyer et al., 2011, Yao et al., 2011, Lafarge and Mallet, 2011, Lafarge and Mallet, 2012, Niemeyer et al., 2014, Yan et al., 2015] and for terrestrial data [Brodu and Lague, 2012, Weinmann et al., 2013, Dohan et al., 2015]. Full 3D processing can handle data which cannot be reduced to height maps in a straight-forward manner, in particular terrestrial data generated from multiple scan positions, and mobile mapping data.

Training a good model requires an expressive feature set. A large number of 3D point descriptors has been developed, which typically encode geometric properties within the point's neighborhood, like surface normal orientation, surface curvature, *etc.* . Popular descriptors are for example spin images [Johnson and Hebert, 1999], fast point feature histograms (FPFH) [Rusu et al., 2009] and signatures of histograms (SHOT) [Tombari et al., 2010]. One drawback of these rich descriptors is their high computational cost. While computation time is not an issue for small point sets (e.g., sparse key points), it is a crucial bottleneck when *all* points in a large point cloud shall be classified. A faster alternative – again for range images rather than true 3D point clouds – is the NARF operator, which is popular for key point extraction and description in the robotics community [Steder et al., 2010, Steder et al., 2011]. In order to achieve robustness against viewpoint changes, it explicitly models object contour information. A computationally cheaper alternative for full 3D point data are features derived from the 3D structure tensor of a point's neighbourhood [Demantké et al., 2011], and from the point distribution in oriented (usually vertical) cylinders [Monnier et al., 2012, Weinmann et al., 2013].

5.2.2 Deep learning for point cloud annotation

Neural networks (usually of the deep, convolutional network flavour) offer the possibility to completely avoid heuristic feature design and feature selection. They are at present immensely popular in 2D image interpretation. Recently, deep learning pipelines have been adapted to voxel grids [Lai et al., 2014, Wu et al., 2015, Maturana and Scherer, 2015] and RGB-D images [Song and Xiao, 2016], too. Being completely data-driven, these techniques have the ability to capture appearance (intensity) patterns as well as geometric object properties. Moreover, their multi-layered, hierarchical architecture has the ability to encode a large amount of contextual information. Deep learning in 3D has been proposed for a variety of applications in robotics, computer graphics, and computer vision. To the best of our knowledge, the earliest attempt that applies a 3D-CNNs on a voxel grid is [Prokhorov, 2010]. The author classifies objects in LiDAR point clouds and improves classification accuracy despite limited amount of training data, by combining supervised and unsupervised training. More recent 3D-CNNs that operates on voxel grids include [Maturana and Scherer, 2015] for landing zone detection in 3D LiDAR point clouds, [Wu et al., 2015] for learning representations of 3D object shapes, and [Huang and You, 2016] to densely label LiDAR point clouds into 7 different object categories. A general drawback when directly applying 3D-CNNs to dense voxel grids derived from originally sparse point clouds is the huge memory overhead for encoding empty space. Computational complexity grows cubically with respect to voxel grid resolution, although high detail would only be needed at object surfaces.

Therefore, more recent 3D-CNNs exploit the sparsity commonly found in voxel grids. One strategy is to resort to an octree representation, where empty space (and potentially also large, geometrically simple object parts) are represented at coarser scales than object details [Riegler et al., 2017, Engelcke et al., 2017, Tatarchenko et al., 2017]. Since the octree partitioning is a function of the object at hand, an important question is how to automatically adapt to new, previously unseen objects at test time. While [Riegler et al., 2017] assume the octree structure to be known at test time, [Tatarchenko et al., 2017] learn to predict the octree structure together with the labels. This allows generalization to unseen instances of a learned object category, without injecting additional prior knowledge.

Another strategy is to rely only on a small subset of the most discriminative points, while neglecting the large majority of less informative ones [Li et al., 2016, Qi et al., 2017a]. The idea is that the network learns how to select the most informative points from training data and aggregates information into global descriptors for object shapes via fully-connected layers. This allows for both shape classification and per-point labeling, while using only a small subset of points, resulting in significant speed and memory gains.

5.2.3 Benchmark initiatives for point clouds

Benchmarking efforts have a long tradition in the geospatial data community and particularly in ISPRS. Recent efforts include, for example, the *ISPRS-EuroSDR benchmark on High Density Aerial Image Matching*[3] that evaluates dense matching methods for oblique aerial images [Haala, 2013, Cavegn et al., 2014] and the *ISPRS Benchmark Test on Urban Object Detection and Reconstruction*, which contains several different challenges like semantic segmentation of aerial images and 3D object reconstruction [Rottensteiner et al., 2013, Rottensteiner et al., 2014].

In computer vision, very large benchmark datasets with millions of images have become standard for learning-based image interpretation. A variety of datasets have been introduced, many tailored for specific tasks, some serving as basis for annual challenges for several consecutive years (e.g., *ImageNet, Pascal VOC*). Datasets that aim at boosting research in image classification and object detection heavily rely on images downloaded from the internet. Web-based imagery has been a major driver of benchmarks because no expensive, dedicated photography campaigns have to be accomplished for dataset generation. This makes it possible to scale benchmarks from hundreds to millions of images, although often weakly annotated and with a considerable amount of label noise, that has to be taken into account when working with the data. Additionally, one can assume that internet images constitute a very general collection of images with less bias towards particular sensors, scenes, countries, objects etc.. This mitigates overfitting, and enables the training of rich, high-capacity models that nevertheless generalize well.

One of the first successful attempts to object detection in images at very large scale is *tinyimages*[4] with over 80 million small ($32 \times 32\ px$) images [Torralba et al., 2008]. A milestone and still widely used dataset for semantic image segmentation is the famous Pascal VOC[5] dataset and challenge [Everingham et al., 2010], which has been used for training and testing many of the well-known, state-of-the-art algorithms today like [Long et al., 2015, Badrinarayanan et al., 2015]. Another, more recent dataset is *MSCOCO*[6], which contains $300{,}000$ images with annotations that allow for object segmentation, object recognition in context, and image captioning. One of the most popular benchmarks in computer vision today is *ImageNet*[7] [Deng et al., 2009, Russakovsky et al., 2015], which made Convolutional Neural Networks popular in computer vision [Krizhevsky et al., 2012]. It contains $> 14 \times 10^6$ images organized ac-

[3] http://www.ifp.uni-stuttgart.de/ISPRS-EuroSDR/ImageMatching/index.en.html
[4] http://groups.csail.mit.edu/vision/TinyImages/
[5] http://host.robots.ox.ac.uk/pascal/VOC/
[6] http://mscoco.org/
[7] http://www.image-net.org

cording to the semantic WordNet hierarchy[8], where words are grouped into sets of cognitive synonyms.

The introduction of the popular, low-cost range sensor Microsoft Kinect gave rise to several large rgb-d image databases. Popular examples are the *NYU Depth Dataset V2*[9] [Silberman et al., 2012] and *SUN RGB-D*[10] [Song et al., 2015] that provide labeled rgb-d images for object segmentation and scene understanding. Compared to laser scanners, low-cost, structured-light rgb-d sensors have much shorter measurement range, lower resolution, and work poorly outdoors, due to interference of the sunlight with the projected infrared pattern.

To the best of our knowledge, no publicly available dataset with laser scans at the scale of the aforementioned vision benchmarks exists today. Thus, many recent Convolutional Neural Networks that are designed for Voxel Grids [Brock et al., 2016, Wu et al., 2015] resort to artificially generated data from the CAD models of ModelNet [Wu et al., 2015], a rather small, synthetic dataset. As a consequence, recent ensemble methods, e.g., [Brock et al., 2016], reach performance of over 97% on ModelNet10, which clearly indicates that the dataset is either too easy, or too small and already significantly overfitted.

Those few existing laser scan datasets are mostly acquired with mobile mapping devices or robots like *DUT1* [Zhuang et al., 2014], *DUT2* [Zhuang et al., 2015], or *KAIST* [Choe et al., 2013], which are small ($< 10^7$ points) and not publicly available. Public laser scan datasets include *Oakland* [Munoz et al., 2009a] ($< 2 \times 10^6$ points), the *Sydney Urban Objects* [De Deuge et al., 2013], *Paris-rue-Madame* [Serna et al., 2014] and data from the *IQmulus & TerraMobilita Contest* [Vallet et al., 2015]. All have in common that they use 3D LIDAR data from mobile mapping vehicles, which provides a much lower point density than static scans, like ours. They are also relatively small and localised, and thus prone to overfitting. The majority of today's available point cloud datasets comes without a thorough, transparent evaluation that is publicly available on the internet, continuously updated and that lists all submissions to the benchmark.

With the *semantic3D.net* benchmark presented in this paper, we attempt to close this gap. It provides a much larger labelled 3D point cloud data set with approximately four billion hand-labeled points, comes with a sound evaluation, and continuously updates submissions. It is the first dataset that allows fully-fledged deep learning on real 3D laser scans, with high-quality, per-point supervision.

[8] `https://wordnet.princeton.edu/`
[9] `http://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html`
[10] `http://rgbd.cs.princeton.edu`

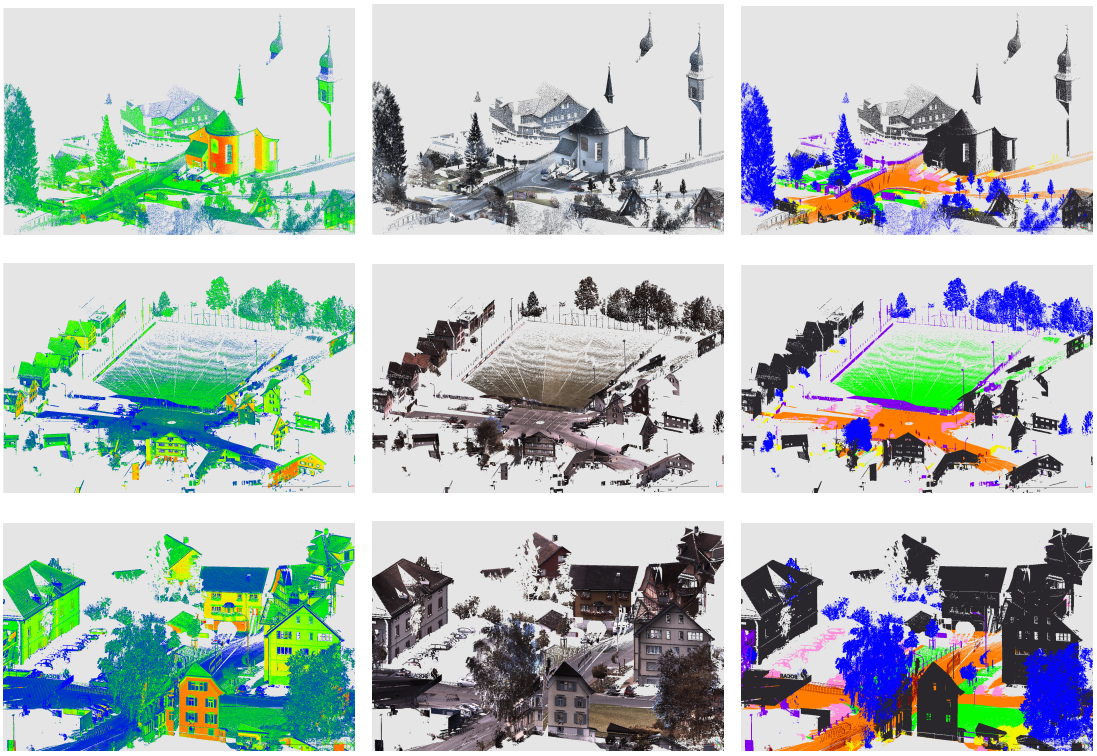


Figure 5.2: Intensity values (left), rgb colors (middle) and class labels (right) for example data sets.

5.3 Data

Our 30 published individual, non-overlapping terrestrial laser scans consist of in total ≈ 4 billion 3D points. Although we would have many more scans that overlap largely with the ones in our benchmark data set and would facilitate co-registration for large scenes, we prefer to keep this for a later extension. The main reason for publishing only individual scans is the huge size per scan (2.72 GB for the largest scan). For the same reason, we did not record multiple echoes per pulse. The data set is split into 15 scans for training that come with labels and 15 scans for testing, where labels are not publicly released and kept by the organizers (see parameters in Tab. 5.1 & 5.2). Submitted results on the test set are evaluated completely automatically on the server and repeated submissions are limited to discourage overfitting on the test set. Train and test data sets are always from different scenes to avoid biasing classifiers and ensure that we verify generalization capability. The data set contains urban and rural scenes, like farms, town halls, sport fields, a castle and market squares. We intentionally selected various different natural and man-made scenes to prevent overfitting of the classifiers. All of the published scenes were captured in Central Europe and depict urban or rural European architecture, as shown in Figure 5.2. Surveying-grade laser scanners were used for recording these scenes. Colorization was performed in a post processing

Train data set	Number of points	Scene type	Description	Download size [GB]
bildstein1	29′302′501	rural	church in bildstein	0.20
bildstein3	23′765′246	rural	church in bildstein	0.17
bildstein5	24′671′679	rural	church in bildstein	0.18
domfountain1	35′494′386	urban	cathedral in feldkirch	0.28
domfountain2	35′188′343	urban	cathedral in feldkirch	0.25
domfountain3	35′049′972	urban	cathedral in feldkirch	0.23
untermaederbrunnen1	16′658′648	rural	fountain in balgach	0.17
untermaederbrunnen3	19′767′991	rural	fountain in balgach	0.17
neugasse	50′109′087	urban	neugasse in st. gallen	0.32
sg27_1	161′044′280	rural	railroad tracks	1.87
sg27_2	248′351′425	urban	town square	2.72
sg27_4	280′994′028	rural	village	1.59
sg27_5	218′269′204	suburban	crossing	1.25
sg27_9	222′908′898	urban	soccer field	1.22
sg28_4	258′719′795	urban	town square	1.40

Table 5.1: Parameters of the full resolution semantic-8 training data set. Identical names (left column) with different IDs identify scans of the same scene (but with very low overlap). All ground truth labels together have size 0.01 GB for download. All parameters are also provided on the benchmark website `http://www.semantic3d.net/view_dbase.php?chl=1`

Test data set	Number of points	Scene type	Description	Download size [GB]
stgallencathedral1	28′181′979	urban	cathedral in st. gallen	0.22
stgallencathedral3	31′328′976	urban	cathedral in st. gallen	0.22
stgallencathedral6	32′342′450	urban	cathedral in st. gallen	0.22
marketsquarefeldkirch1	23′228′738	urban	market square in feldkirch	0.17
marketsquarefeldkirch4	22′760′334	urban	market square in feldkirch	0.15
marketsquarefeldkirch7	23′264′911	urban	market square in feldkirch	0.15
birdfountain1	36′627′054	urban	fountain in feldkirch	0.25
castleblatten1	152′248′025	rural	castle in blatten	0.24
castleblatten5	195′356′302	rural	castle in blatten	0.70
sg27_3	422′445′052	suburban	houses	2.40
sg27_6	226′790′878	urban	city block	1.27
sg27_8	429′615′314	urban	city center	2.08
sg27_10	285′579′196	urban	town square	1.56
sg28_2	170′158′281	rural	farm	0.94
sg28_5	269′007′810	suburban	buildings	1.35

Table 5.2: Parameters of the full resolution semantic-8 testing data set. Identical names (left column) with different IDs identify scans of the same scene (but with very low overlap). All parameters are also provided on the benchmark website `http://www.semantic3d.net/view_dbase.php?chl=1`

step, by generating high-resolution cubemaps from co-rgistered camera images. In general, static laser scans have a very high resolution and are able to measure long distances with little noise. Especially compared to point clouds derived via structure-from-motion pipelines or Kinect-like structured light sensors, laser scanners deliver superior geometric data quality.

Scanner positions for data recording were selected as usually done in real field campaigns: only little scan overlap as needed for registration, so that scenes can be recorded in a minimum of time. This free choice of the scanning position implies that no prior assumption based on point density and on class distributions can be made. We publish up to 3 laser scans per scene that have small overlap. The relative position of laser scans at the same location was estimated from targets.

The choice of output classes in a benchmark, independent of downstream applications, is not obvious. Based on feedback from geo-spatial industry experts, we use the following 8 classes, which are considered useful for a variety of surveying applications: *(1) man made terrain:* mostly pavement; *(2) natural terrain:* mostly grass; *(3) high vegetation:* trees and large bushes; *(4) low vegetation:* flowers or small bushes which are smaller than 2 m; *(5) buildings:* Churches, city halls, stations, tenements, etc.; *(6) remaining hard scape:* a clutter class with for instance garden walls, fountains, benches, etc.; *(7) scanning artifacts:* artifacts caused by dynamically moving objects during the recording of the static scan; *(8) cars and trucks*. Some of these classes are ill-defined, for instance some scanning artifacts could also go for cars or trucks and it can be hard to differentiate between large and small bushes. Yet, we prefer not to alter the class nomenclature in a way that might reduce ambiguities, but departs from the requirements of the data providers and users. Note also, in many application projects class *7*, scanning artifacts, is filtered out in pre-processing with heuristic rule sets. Within the benchmark we prefer to also include that additional classification problem in the overall machine learning pipeline, and thus do not perform any heuristic pre-processing.

In our view, large data sets are important for two reasons: *a)* Typically, real world scan data are large. To have an impact on real problems, a method must be able to process large amounts of data. *b)* Large data sets are especially important for modern machine learning methods that involve representation learning (i.e., extracting discriminative low- to high-level features from the raw data). With too small data sets, good results leave strong doubts about possible overfitting; unsatisfactory results, on the other hand, are hard to interpret as guidelines for further research: are the mistakes due to shortcomings of the method, or simply caused by insufficient training data?

5.3.1 Point Cloud Annotation

In contrast to common strategies for 3D data labelling that first compute an automatic over-segmentation and then label segments, we manually assign each point a class label individually. Although this strategy is more labor-intensive, it avoids inheriting errors from the segmentation; and, perhaps more importantly, it ensures that the ground truth does not contain any biases from a particular segmentation algorithm, that could be exploited by the classifier and impair its use with other training data. In general, it is more difficult for humans to label a point cloud by hand than images. The main problem is that it is hard to select a 3D point on a 2D monitor from a set of millions of points without a clear neighbourhood/surface structure. We tested two different strategies:

Annotation in 3D: We follow an iterative filtering strategy, where we manually select a couple of points, fit a simple model to the data, remove the model outliers and repeat these steps until all inliers belong to the same class. With this procedure it is possible to select large buildings in a couple of seconds. A small part of the point clouds was labeled with this approach by student assistants at ETH Zurich.

Annotation in 2D: The user rotates a point cloud, fixes a 2D view and draws a closed polygon which splits a point cloud into two parts (inside and outside of the polygon). One part usually contains points from the background and is discarded. This procedure is repeated a few times until all remaining points belong to the same class. In the end, all points are separated into different layers corresponding to classes of interest. This 2D procedure works well with existing software packages [Daniel Girardeau-Montaut, CloudCompare, 2016] such that it can be outsourced to external labelers more easily than the 3D work-flow. We used this procedure for all data sets where annotation was outsourced.

5.4 Methods

Given a set of points (here: dense scans from a static, terrestrial laser scanner), we want to infer an individual class label per point. We provide three baseline methods that are meant to represent typical categories of approaches recently used for the task, covering the state of the art at the time of creating the benchmark.

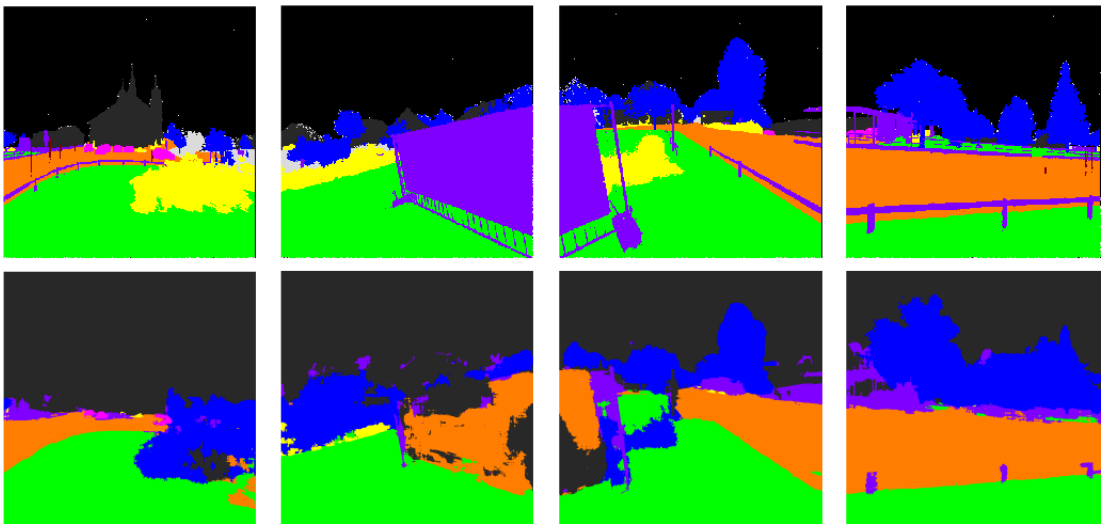


Figure 5.3: *Top row:* projection of ground truth to images. *Bottom row:* results of classification with the image baseline. *White:* unlabeled pixels, *black:* pixels with no corresponding 3D point, *gray:* buildings, *orange:* man made ground, *green:* natural ground, *yellow:* low vegetation, *blue:* high vegetation, *purple:* hard scape, *pink:* cars

5.4.1 2D Image Baseline

We convert color values of the scans to separate images (without depth) with cube mapping [Greene, 1986]. Cube maps are centered on the origin of the laser scanner and we thus do not experience any self-occlusions. Ground truth labels are also projected from the point clouds to image space, such that the 3D point labeling task turns into a purely image-based semantic segmentation problem in 2D (Figure 5.3). We chose the associative hierarchical random fields method [Ladicky et al., 2013] for semantic segmentation because it has proven to deliver good performance for a variety of tasks (e.g., [Montoya et al., 2014, Ladický et al., 2014]) and was available in its original implementation.

The method works as follows: four different types of features – textons [Malik et al., 2001], SIFT [Lowe, 2004], local quantized ternary patters [Hussain and Triggs, 2012] and self-similarity features [Shechtman and Irani, 2007] – are extracted densely at every image pixel. Each feature category is separately clustered into 512 distinct patterns using standard K-means clustering, which corresponds to a typical bag-of-words representation. For each pixel in an image, the feature vector is a concatenation of bag-of-word histograms over a fixed set of 200 rectangles of varying sizes. These rectangles are randomly placed in an extended neighbourhood around a pixel. We use multi-class boosting [Torralba et al., 2004] as classifier and the most discriminative weak features are found as explained in [Shotton et al., 2006]. To add local smoothing without loosing sharp object boundaries, the model includes soft constraints that favor

constant labels inside superpixels and class transitions at their boundaries. Super-pixels are extracted via mean-shift [Comaniciu and Meer, 2002] with 3 sets of coarse-to-fine parameters as described in [Ladicky et al., 2013]. Class likelihoods of overlapping superpixels are predicted using the feature vector consisting of a bag-of-words representation for each superpixel. Pixel-based and superpixel-based classifiers with additional smoothness priors over pixels and superpixels are combined in a conditional random field framework, as proposed in [Kohli et al., 2008]. The maximum a-posteriori label configuration is found using a graph-cut algorithm [Boykov and Kolmogorov, 2004], with appropriate graph construction for higher-order potentials [Ladicky et al., 2013].

5.4.2 3D Covariance Baseline

The second baseline was inspired by [Weinmann et al., 2015b, Hackel et al., 2016b]. It infers the class label directly from the 3D point cloud using multiscale features and discriminative learning. Again, we had access to the original implementation of [Hackel et al., 2016b]. That method uses an efficient approximation of multi-scale neighbourhoods, where the point cloud is sub-sampled into a multi-resolution pyramid, such that a constant, small number of neighbours per level captures the multi-scale information. The multi-scale pyramid is generated by voxel-grid filtering with uniform spacing.

The feature set extracted at each level is an extension of the one decribed in [Weinmann et al., 2013]. It uses different combinations of eigenvalues and eigenvectors of the covariance per point-neighborhood to represent geometric surface properties. Furthermore, height features based on vertical, cylindrical neighbourhoods are added to emphasize the special role of the gravity direction (assuming that scans are, as usual, aligned to the vertical). Note that we do not make use of color values or laser intensities. We empirically found that they did not improve the point cloud classification, moreover color or intensity information is not always available. As classifier, we use a random forest, for which optimal parameters (number of trees and tree depths) are found with grid search and five-fold cross-validation.

5.4.3 3D CNN Baseline

We design our baseline for the point cloud classification task following recent ideas of VoxNet [Maturana and Scherer, 2015] and ShapeNet [Wu et al., 2015] for 3D encoding. The pipeline is illustrated in Fig. 5.4. Instead of generating a global 3D voxel-grid prior to processing, we create $16 \times 16 \times 16$ voxel cubes per scan point[11]. We do this at 5

[11] This strategy automatically centers each voxel-cube per scan point. Note that for the alternative approach of a global voxel grid, several scan points could fall into the same grid cell in dense regions of the scan. This would require scan point selection per grid cell, which is computationally costly and

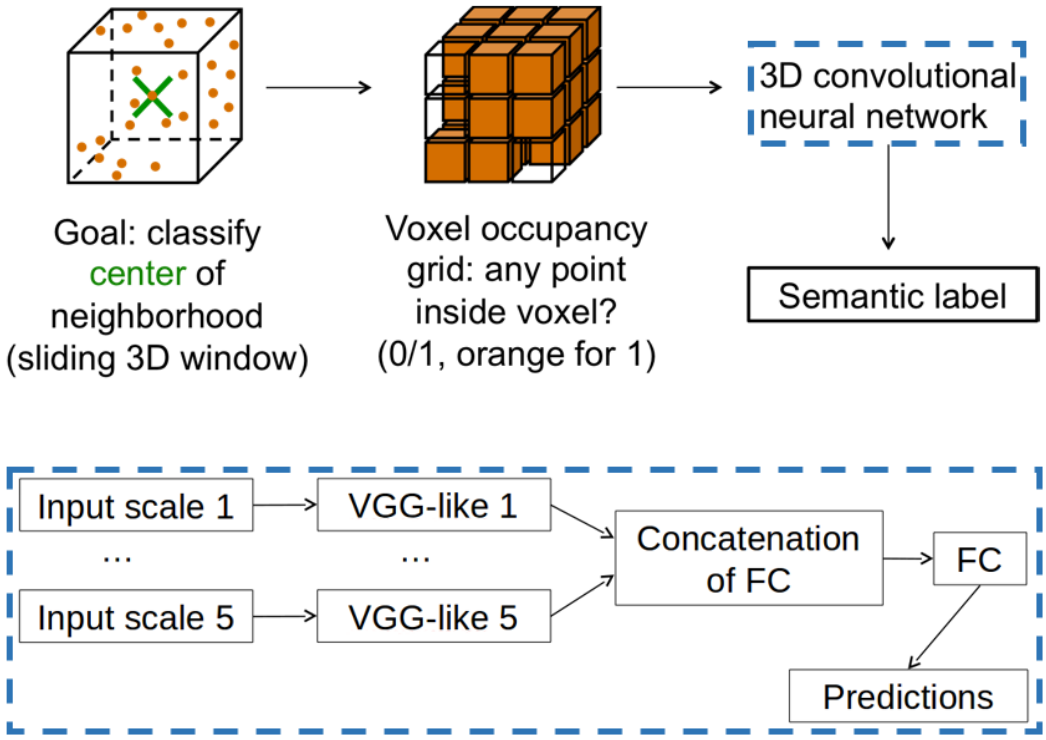


Figure 5.4: Our deep neural network baseline.

different resolutions, with voxel sizes ranging from 2.5 cm to 40 cm (multiplied by powers of 2) and encode empty voxel cells as 0 and filled ones as 1. The input to the CNN is thus encoded in a multidimensional tensor with $5 \times 16 \times 16 \times 16$ cube entries per scan point.

Each of the five scales is handled separately by a VGG-like [Simonyan and Zisserman, 2014b] network branch that includes convolutional, pooling and ReLU layers. The 5 separate network paths are finally concatenated into a single representation, which is passed through two fully-connected layers. The output of the second fully-connected layer is an 8-dimensional vector, which contains the class scores for each of the 8 classes in this benchmark challenge. Scores are transformed to class conditional probabilities with the soft-max function.

Before describing the network architecture in detail we introduce the following notation:

$c(i, o)$ stands for convolutional layers with $3 \times 3 \times 3$ filters, i input channels, o output channels, zero-padding of size 1 at each border and a stride of 1. $f(i, o)$ stands for fully-connected layers. r stands for a ReLU non-linearity, m stands for a volumetric max-pooling with receptive field $2 \times 2 \times 2$, applied with a stride of 2 in each dimension, d stands for a dropout with 0.5 probability, and s stands for a $softmax$ layer.

results in (undesired) down-sampling.

Our 3D CNN architecture assembles these components to a VGG-like network. We choose the filter size in convolutional layers as small as possible ($3 \times 3 \times 3$), as recommended in recent work [He et al., 2016], to have the least amount of parameters per layer and, hence, reduce both the risk of overfitting and the computational cost. Each of the 5 separate network paths, acting at different resolutions, has the sequence:

$$(c(1, 16), r, m, c(16, 32), r, m, c(32, 64), r, m).$$

The output is vectorized, concatenated across all branches (scales), and fed through two fully-connected layers to predict the class responses:

$$(f(2560, 2048), r, d, f(2048, 8), s).$$

The network is trained by minimising the standard multi-class cross-entropy loss, with stochastic gradient descent (SGD, [Bottou, 2010]). The SGD algorithm uses randomly sampled mini-batches of several hundred points per batch to iteratively update the parameters of the CNN. We use the popular *adadelta* [Zeiler, 2012] variant of SGD. We use a mini-batch size of 100 training samples (i.e., points), where each batch is sampled randomly and balanced to contain equal numbers of samples per class. We run training for $74{,}700$ batches and sample training data from a large and representative point cloud with 259 million points (scan sg28_4). A standard pre-processing step for CNNs is data augmentation to enlarge the training set and to avoid overfitting. Here, we augment the training set with a random rotation around the z-axis after every 100 batches. During experiments it turned out that additional training data did not improve performance. This indicates that in our case we rather face underfitting (as opposed to overfitting), i.e., our model lacks the capacity to fully capture all the evidence in the available training data[12]. We thus refrain from further possible augmentations like randomly missing points or adding noise. The network is implemented in C++ and Lua and uses the Torch7 framework [Collobert et al., 2011] for deep learning. Code and documentation are available at `https://github.com/nsavinov/semantic3dnet`.

5.4.4 Submissions to the benchmark

The two top-performing approaches [Boulch et al., 2017,Lawin et al., 2017] submitted to the benchmark so far[13] both project 3D point clouds to 2D images, so as to harness the strength of well-established CNN models in 2D space. Their strategy is to: *(i)* render virtual 2D images from viewpoints in the 3D point cloud; *(ii)* perform semantic classification on the 2D images; *(iii)* lift the results back into 3D space, and merge the

[12]Our model reaches the hardware limits of our GPU (TitanX with 12GB of RAM), we thus did not experiment with larger networks at this point.

[13]as of August 28, 2017

predictions from different 2D views. In the following, we provide a brief overview of both methods. Schematic work-flows are shown in Fig. 5.5 & 5.6.

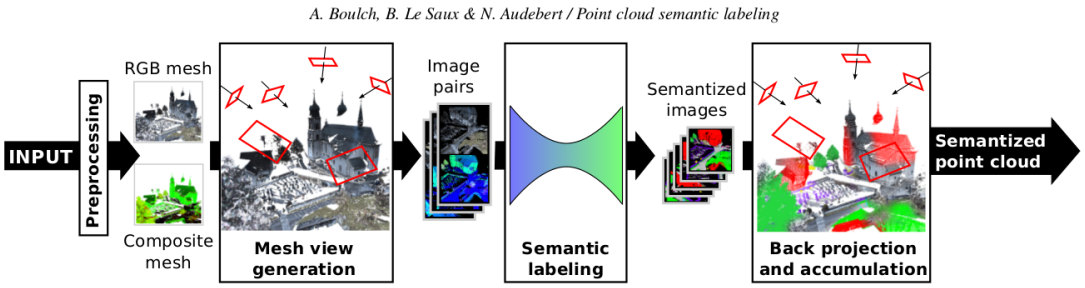


Figure 5.5: Work-flow of the *SnapNet* approach, figure taken from the original paper [Boulch et al., 2017].

The currently top-performing method is *SnapNet* [Boulch et al., 2017]. The processing pipeline consists of four main parts (Fig. 5.5):

1) Point clouds are down-sampled with a voxel grid filter, 3D features are extracted (e.g., the deviation of surface normals to a vertical vector, sphericity etc.), and 3D meshes are generated by running the surface reconstruction approach of [Marton et al., 2009];

2) Virtual images are rendered from meshes at a high number (400 per point cloud for training) of different camera positions. RGB images as well as composite images with a channel for depth, the deviation of surface normals and sphericity are computed. For training and validation sets also virtual ground truth images are rendered. The authors propose to select camera view points either randomly in the bounding box of the scene (altitudes vary between 10 and 30 meters above ground) or to apply a multi-scale strategy, where three camera poses are generated for a subset of points that vary in distance to the selected point. A 3D mesh viewer renders virtual 2D images from the mesh.

3) Two different encoder-decoder CNNs, SegNet [Badrinarayanan et al., 2015] and U-Net [Ronneberger et al., 2015], are compared for semantic labeling of the rendered virtual images. Moreover, different strategies to combine RGB and depth information are tested, for example, model averaging and adding a shallow network to the output of the two separate depth and RGB networks.

4) Class responses of the neural network are back-projected to the mesh and averaged over the different virtual views. Finally, a kd-tree is used to assign the class label with the highest class response in the mesh to close points in the point cloud. The overall best results (i.e., those reported for the benchmark, cf. Tab. 4.2 & 4.3) are obtained with a combination of U-Net, shallow network for depth and RGB fusion, and multi-scale view generation.

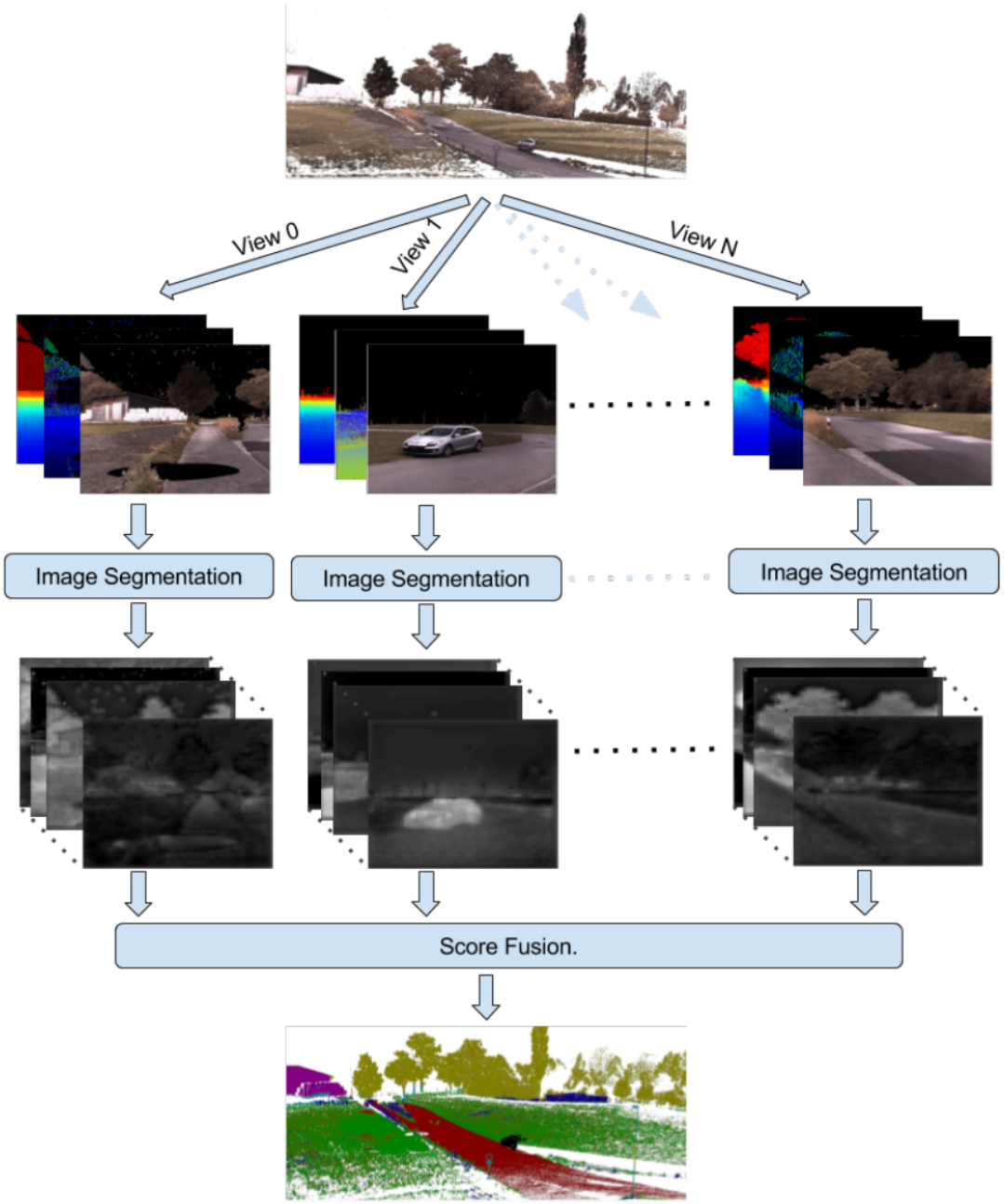


Figure 5.6: Work-flow of *DeePr3SS*, figure borrowed from the original paper [Lawin et al., 2017].

The second-best submission at present is *DeePr3SS* [Lawin et al., 2017] (Fig. 5.6), which follows a coneptually similar strategy as [Boulch et al., 2017]:

1) Virtual images with RGB channels as well as channels for depth and surface normals are rendered directly from the point clouds by point splatting [Zwicker et al., 2001] (which, unlike [Boulch et al., 2017], works without an intermediate mesh generation step). In total, 120 camera views are rendered per point cloud by rotating the camera around four vertical axis in the scene. Low quality images are discarded by using two filter strategies: First, images with a coverage below a threshold are removed. Second, views which are too close to large objects are neglected by thresholding the percentage of small depths.

2) Semantic segmentation is performed using fully convolutional networks, where the different inputs are fused by using a multi-stream architecture [Simonyan and Zisserman, 2014a] that averages the output of the different streams. The authors use pre-trained VGG16 networks [Simonyan and Zisserman, 2014b] for each stream and experiment with different combinations of streams for RGB, depth and normal channels. As often done, pre-training is performed on the ImageNet dataset [Russakovsky et al., 2015].

3) Finally, class responses of the CNN are back-projected to the point cloud. Mapping between 3D points and pixels in the virtual images is given by rendering with point splatting. Class responses of the CNN for all pixels which correspond to the same 3D point are summed up, and the maximum average class response is used as final class label. The authors report that the multi-stream architecture with streams for all RGB, depth and normal channels works best (that workflow is used to produce numbers shown in Tab. 4.3 for the benchmark) for *DeePr3SS*.

Method	$\overline{IoU}$	OA	$t[s]$	IoU_1	IoU_2	IoU_3	IoU_4	IoU_5	IoU_6	IoU_7	IoU_8
SnapNet	**0.674**	**0.910**	unknown	0.896	**0.795**	**0.748**	0.561	0.909	**0.365**	**0.343**	**0.772**
HarrisNet	0.623	0.881	unknown	0.818	0.737	0.742	**0.625**	**0.927**	0.283	0.178	0.671
TMLC-MS	0.494	0.850	**38421**	**0.911**	0.695	0.328	0.216	0.876	0.259	0.113	0.553
TML-PC	0.391	0.745	unknown	0.804	0.661	0.423	0.412	0.647	0.124	0.0*	0.058

Table 5.3: Semantic3d benchmark results on the full data set: 3D covariance baseline *TMLC-MS*, 2D RGB image baseline *TML-PC*, and first submissions *HarrisNet* and *SnapNet*. *IoU* for categories (1) man-made terrain, (2) natural terrain, (3) high vegetation, (4) low vegetation, (5) buildings, (6) hard scape, (7) scanning artefacts, (8) cars. * Scanning artefacts were ignored for 2D classification because they are not present in the image data.

Method	$\overline{IoU}$	OA	$t[s]$	IoU_1	IoU_2	IoU_3	IoU_4	IoU_5	IoU_6	IoU_7	IoU_8
SnapNet	**0.591**	0.886	3600	0.820	0.773	**0.797**	0.229	**0.911**	0.184	**0.373**	**0.644**
DeePr3SS	0.585	**0.889**	unknown	0.856	**0.832**	0.742	**0.324**	0.897	0.185	0.251	0.592
TMLC-MSR	0.542	0.862	**1800**	**0.898**	0.745	0.537	0.268	0.888	**0.189**	0.364	0.447
DeepNet	0.437	0.772	64800	0.838	0.385	0.548	0.085	0.841	0.151	0.223	0.423
TML-PCR	0.384	0.740	unknown	0.726	0.73	0.485	0.224	0.707	0.050	0.0*	0.15

Table 5.4: Semantic3d benchmark results on the reduced data set: 3D covariance baseline *TMLC-MSR*, 2D RGB image baseline *TML-PCR*, and our 3D CNN baseline *DeepNet*. *TMLC-MSR* is the same method as *TMLC-MS*, the same goes for *TMLC-PCR* and *TMLC-PC*. In both cases *R* indicates classifiers on the reduced dataset. *IoU* for categories (1) man-made terrain, (2) natural terrain, (3) high vegetation, (4) low vegetation, (5) buildings, (6) hard scape, (7) scanning artefacts, (8) cars. * Scanning artefacts were ignored for 2D classification because they are not present in the image data.

5.5 Evaluation

We follow the Pascal VOC challenge [Everingham et al., 2010] and choose the *Intersection over Union* (IoU), averaged over all classes, as our principal evaluation metric.[14]. Let the classes be indexed with integers from $\{1, \ldots, N\}$, with N the number of different classes. Let C be an $N \times N$ confusion matrix of the chosen classification method, where each entry c_{ij} is a number of samples from ground-truth class i predicted as class j. Then the evaluation measure per class i is defined as

$$IoU_i = \frac{c_{ii}}{c_{ii} + \sum_{j \neq i} c_{ij} + \sum_{k \neq i} c_{ki}}. \tag{5.1}$$

The main evaluation measure of our benchmark is thus

$$\overline{IoU} = \frac{1}{N} \sum_{i=1}^{N} IoU_i. \tag{5.2}$$

We also report IoU_i for each class i and overall accuracy

$$OA = \frac{\sum_{i=1}^{N} c_{ii}}{\sum_{j=1}^{N} \sum_{k=1}^{N} c_{jk}} \tag{5.3}$$

as auxiliary measures and provide the confusion matrix C. Finally, each participant is asked to specify the time T it took to classify the test set as well as the hardware

[14] IoU compensates for different class frequencies as opposed to, for example, *overall accuracy* that does not balance different class frequencies, thus giving higher influence to large classes.

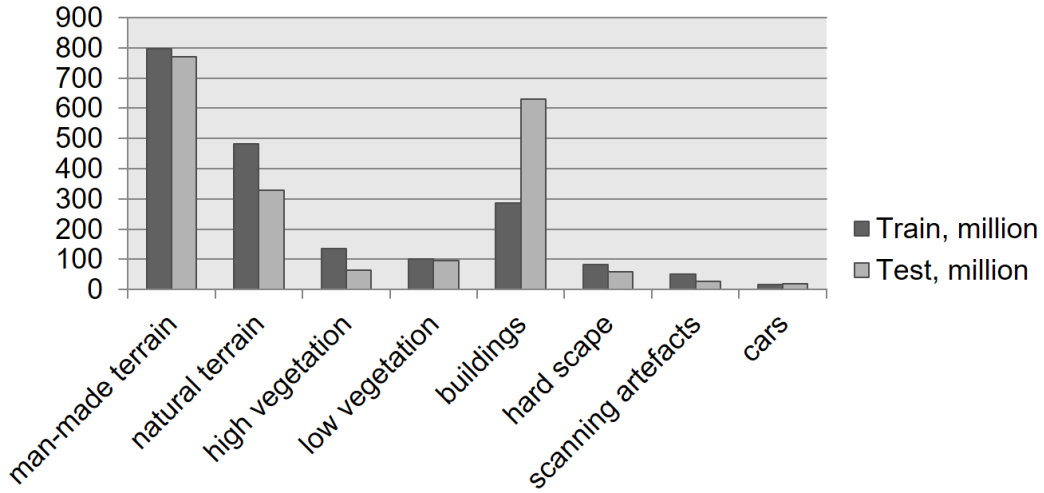
used for experiments. The computation time (if available) is important to understand how suitable the method is in real-world scenarios, where usually billions of points are required to be processed.

For computationally demanding methods we additionally provide a reduced challenge, consisting of a subset of the original test data. The results of our baseline methods as well as submissions are shown in Table 5.3 for the full challenge and in Table 5.4 for the reduced challenge. Of the three published baseline methods the classical machine learning pipeline with hand-designed, covariance-based features performs better than simplistic color image labeling without 3D information, and it also beats our simple CNN baseline, *DeepNet*. Due to its computational cost we could only run the *DeepNet* on the reduced data set. We note that *DeepNet* is meant as a baseline for "naive" application of CNNs to point cloud data, we do expect a more sophisticated, higher-capacity network to perform significantly better. Both *SnapNet* and *DeePr3SS* comfortably beat all baselines.

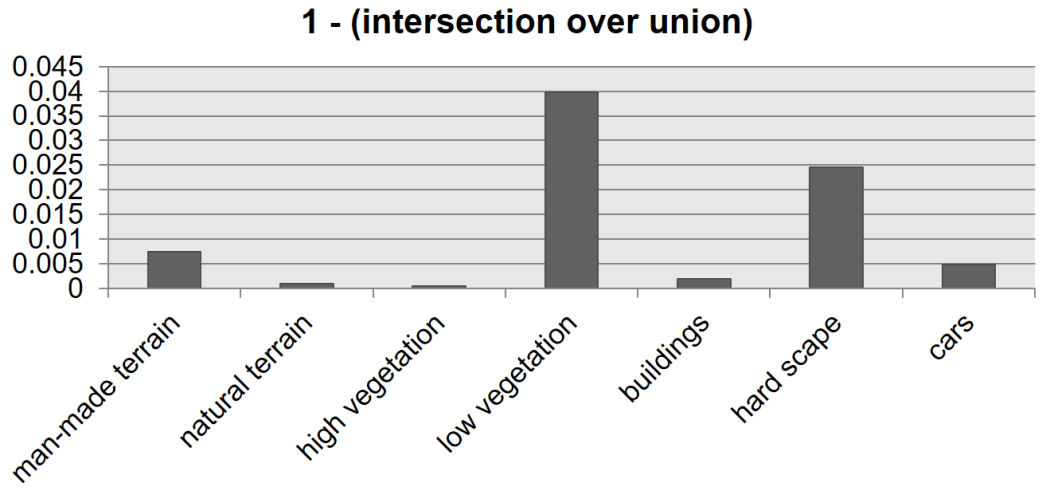
On the full challenge, two CNN methods, *SnapNet* and *HarrisNet* (unfortunately unpublished), already beat our best baseline by a significant margin (Table 5.3) of 12 respective 18 percent points. This indicates that deep learning seems to be the way to go also for point clouds, if enough training data is available. However, it should be noted that both *SnapNet* and *HarrisNet* are no true 3D-CNN approaches in the sense that they do not process 3D data directly. Both methods side-step 3D processing and cast semantic segmentation of point clouds as a 2D image labeling problem. For the future of the benchmark it will be interesting how true 3D-CNN approaches like [Riegler et al., 2017, Tatarchenko et al., 2017, Qi et al., 2017a] will perform. As a lesson learned, a future update of the benchmark should include multi-station point clouds that challenge the reprojection strategy.

5.6 Benchmark Statistics

Class distributions in the test and training sets are rather similar, as shown in Figure 5.7a. Interestingly, the class with most samples is *man-made terrain* because, out of convenience, operators in the field tend to place the scanner on flat and paved ground. Recall also the quadratic decrease of point density with distance to the scanner, such that many samples are close to the scanner. The largest difference between samples in test and training sets occurs for class *building*. However, this does not seem to affect the performance of the submissions so far. The most difficult classes, *scanning artefacts* and *cars*, have only few training and test samples and a large variation of possible object shapes. *Scanning artefacts* is probably the hardest class because the shape of artefacts mostly depends on the movement of objects during the scanning process. Note that, following discussions with industry professionals, the class *hard*



(a)



(b)

Figure 5.7: (a) Number of points per class over all scans and (b) ground truth label errors estimated in overlapping parts of adjacent scans.

scape was designed as a sort of "clutter class" that contains all sorts of man-made objects except for buildings, cars and the ground.

In order to quantify the quality of the manually acquired labels, we also checked the label agreement among human annotators. This provides an indicative measure how well different annotators agree on the correct labeling, and can be viewed as an internal check of manual labeling precision. To estimate the label agreement between different human annotators, we inspect areas where different scans of the same scene overlap (recall that overlaps of adjacent scans can be established precisely, via artificial markers placed in the scenes). Since we cannot rule out that some overlapping area might have been labeled twice by the same person (labeling was outsourced and

we thus do not know exactly who annotated what), the observed consistency might in the worst case be slighty too optimistic. Even if scan alignments would be perfect without any error, no exact point-to-point correspondences exist between two scans, because scan points acquired from two different locations will not fall exactly onto the same 3D location. We thus have to resort to nearest-neighbor search to find point correspondences. Moreover, not all scan points have a corresponding point in the adjacent scan. A threshold of 5 cm on the distance is used to ignore those points where no correspondence exists. Once point correspondences have been established, it is possible to transfer the annotated labels from one point cloud to the other and compute a confusion matrix. Note that this definition of correspondence is not symmetric, "forward" point correspondences from cloud A to cloud B are not in all cases the same as "backward" correspondences from cloud B to cloud A. For each pair, we calculate two intersection-over-union (IoU_i) values, which indicate negligible differences between forward and backward matching, an overall disagreement $< 3\%$, and a maximum label disagreement for the worst class (*low vegetation*) of $< 5\%$, see Figure 5.7b. Obviously, no correspondences between asynchronously acquired scans can be found on moving objects, so we ignored the class *scanning artefacts* in the evaluation.

5.7 Conclusion and Outlook

The *semantic3D.net* benchmark provides a large set of high quality, individual terrestrial laser scans with over 4 billion manually annotated points and a standardized evaluation framework. The data set has been published recently and the first results have been submitted. These already show that deep learning, and in particular appropriately adapted and well-engineered CNNs, outperform the leading conventional approaches, such as our covariance baseline, on large 3D laser scans. Interestingly, both top-performing methods SnapNet and HarrisNet are no true 3D-CNN approaches in the sense that they do not process 3D data directly. Both methods cast semantic point cloud segmentation as a 2D image labeling problem. This leaves room for methods that directly work in 3D and we hope to see more submissions of this kind in the future.

We are confident that, as more submissions appear, the benchmark will enable objective comparisons and yield new insights into strengths and weaknesses of different classification approaches for point clouds, and that the common testbed can help to guide future research efforts. We hope that the benchmark meets the needs of the research community and becomes a central resource for the development of new, more efficient and more accurate methods for semantic data interpretation in 3D space.

Acknowledgement

This work is partially funded by the Swiss NSF project 163910, the Max Planck CLS Fellowship and the Swiss CTI project 17136.1 PFES-ES.

Chapter 6

Inference, Learning and Attention Mechanisms that Exploit and Preserve Sparsity in Convolutional Networks

Abstract

While CNNs naturally lend themselves to densely sampled data, and sophisticated implementations are available, they lack the ability to efficiently process sparse data. In this work we introduce a suite of tools that exploit sparsity in both the feature maps and the filter weights, and thereby allow for significantly lower memory footprints and computation times than the conventional dense framework when processing data with a high degree of sparsity. Our scheme provides *(i)* an efficient GPU implementation of a convolution layer based on direct, sparse convolution; *(ii)* a filter step within the convolution layer, which we call *attention*, that prevents fill-in, i.e., the tendency of convolution to rapidly decrease sparsity, and guarantees an upper bound on the computational resources; and *(iii)* an adaptation of the back-propagation algorithm, which makes it possible to combine our approach with standard learning frameworks, while still exploiting sparsity in the data and the model.

6.1 Introduction

Deep neural networks are nowadays the most popular and most successful tool for a wide spectrum of computer vision problems [Krizhevsky et al., 2012, Long et al., 2015, Ren et al., 2015]. A main reason for their spectacular comeback, perhaps even the single most important factor besides much larger training data sets, is the enormous gain in computational efficiency brought about by highly efficient parallel computing on GPUs. Both the response maps (feature maps) within the neural network and the parameters (filter weights) of the network form regular grids that are conveniently stored

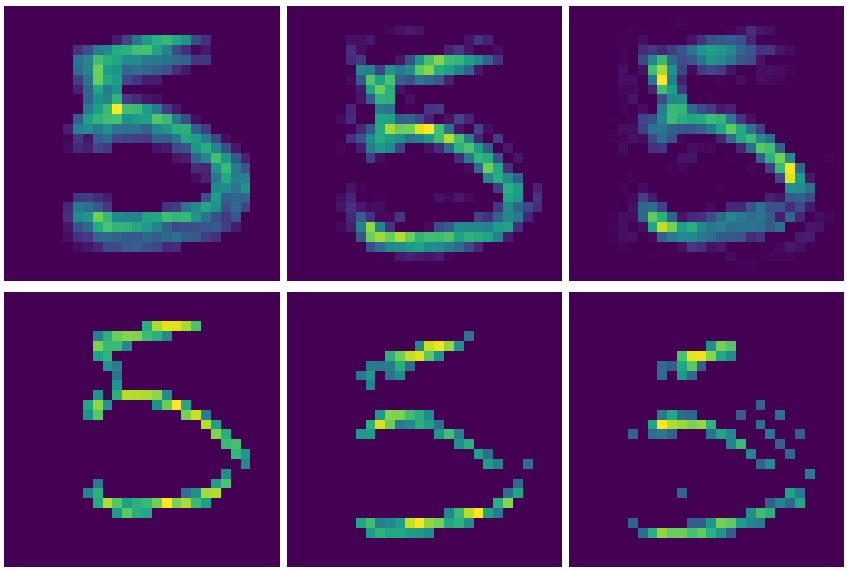


Figure 6.1: Activations for one channel of the first, second and third convolutional layer after one training epoch, in a neural network for MNIST digit classification. *Top:* dense network; *Bottom:* sparse network with upper bound of $\rho_{up} = 15\%$ density per channel

and processed as tensors. However, when going beyond standard image processing tasks, regular grids can be a somewhat unnatural and suboptimal representation, for instance for decentralised multi-sensor systems, or for unstructured 3D point clouds.

Typically, point clouds are acquired with line-of-sight instruments, thus the large majority of points lies on a small number of 2D surfaces. When represented as a 3D voxel grid they therefore exhibit a high degree of sparsity: most voxels are empty; while at the same time 3D data processing with CNNs is seriously challenged by the high memory demands [Brock et al., 2016, Wu et al., 2015, Maturana and Scherer, 2015, Huang and You, 2016]. A counter-measure is to make explicit the sparsity of the feature maps, (see example in Figure 6.1). One way to do this is to change the data representation, e.g. , [Riegler et al., 2017] use an octree instead of regular voxels. Another possibility is to store the voxel grids as sparse tensors, usually implemented as compressed sparse rows (CRS) or index-value pairs.

It can also be beneficial to represent the CNN parameters in a sparse fashion. This can potentially improve both runtime and – perhaps more important for modern, deep architectures – memory footprint; especially if the sparsity is promoted already during training through appropriate regularisation. It is obvious that, in a sufficiently sparse setting, a significant speed-up can be achieved by performing convolutions *directly*, incrementally updating a layer's output map only where there are non-zero entries in the input map as well as non-zero filter weights. This has recently been confirmed independently by two concurrent works [Engelcke et al., 2017, Park et al., 2017]. Direct

convolution guarantees that only the minimum number of necessary operations is carried out. However, the selective, incremental updating only at indexed locations makes parallelisation harder. This may be the reason why, to our knowledge, no practical implementation with sparse feature maps exist (in [Park et al., 2017], a smaller gain is achieved by storing only the filter weights in CRS format). In this work we develop a framework to exploit both sparse feature maps and sparse filter parameters in CNNs. To that end *(i)* we provide a sparse *Direct Convolution Layer*, as well as sparse versions of the *ReLU* and *max-pooling* layers; *(ii)* we extend the back-propagation algorithm to preserve sparsity and make our sparse layers usable with existing optimization routines that are available in modern deep learning frameworks, which have been designed for dense data; *(iii)* we propose to add a density-dependent regulariser that encourages sparsity of the feature maps, and a pruning step that suppresses small filter weights. This regularisation in fact guarantees that the network gets progressively faster at its task, as it receives more training. All these steps have been implemented on GPU as extensions of *Tensorflow*, for generic n-dimensional tensors. Source code and data are publicly available[1]. We test our sparse CNN structure both on a sparse version of MNIST and on realistic 3D data sets, and show that it outperforms its dense counterpart in terms of both runtime and memory footprint when processing data with a high degree of sparsity.

6.2 Related Work

Dense CNN for sparse data. Neural networks, usually of the deep, convolutional network flavour, offer the possibility to completely avoid heuristic feature design and feature selection. They are at present immensely popular in 2D image interpretation. Recently, deep learning pipelines have been adapted to voxel grids [Prokhorov, 2010, Lai et al., 2014, Maturana and Scherer, 2015, Wu et al., 2015] ,RGB-D images [Song and Xiao, 2016] and video [Karpathy et al., 2014], too. Being completely data-driven, these techniques have the ability to capture appearance (intensity patterns) as well as geometric object properties. Moreover, their multi-layered, hierarchical architecture is able to encode a large amount of contextual information. A general drawback when directly applying 3D-CNNs to (dense) voxel grids derived from (originally sparse) point clouds is the huge memory overhead for encoding empty space. Computational complexity grows cubically with respect to voxel grid resolution, but in fact high resolution would only be needed at object surfaces.

[1] `https://github.com/TimoHackel/ILA-SCNN`

Data sparsity. Therefore, more recent 3D-CNNs exploit the *sparsity of occupied voxels* prevalent in practical voxel grids. Graham *et al.* [Graham, 2014] introduced a sparse CNN, which takes into account sparsity but is limited to small resolutions due to decreasing sparsity in convolutional layers. In their publication they consider resolutions of up to 80^3 voxels. Another strategy is to resort to an octree representation, where empty space (and potentially also large, geometrically simple object parts) are represented at coarser scales than object details [Riegler et al., 2017, Tatarchenko et al., 2017]. Since the octree partitioning is a function of the object at hand, an important question is how to automatically adapt to new, previously unseen objects at test time. While [Riegler et al., 2017] assume the octree structure to be known at test time, [Tatarchenko et al., 2017] learn to predict the octree structure together with the labels. This allows generalization to unseen instances of a learned object category, without injecting additional prior knowledge. Häne *et al.* [Häne et al., 2017] uses a coarse-to-fine scheme to hierarchically predict the values of small blocks of voxels in a voxel block octree. Another strategy is to rely only on a small subset of the most discriminative points, while neglecting the large majority of less informative ones [Li et al., 2016, Qi et al., 2017a, Qi et al., 2017b]. The idea is that the network learns how to select the most informative points from training data and aggregates information into global descriptors of object shape via fully-connected layers. This allows for both shape classification and per-point labeling, while using only a small subset of points, resulting in significant speed and memory gains. Recently, Graham *et al.* [Graham and van der Maaten, 2017, Graham et al., 2017] advocate the strategy to perform convolutions only on non-zero elements in the feature map and find correspondences with the help of hash tables. However, to limit activations to non-zero elements of the input can increase the error and slow down learning.

Parameter sparsity. Several works address the situation that the model *parameters* are sparse. Denil *et al.* [Denil et al., 2013] reduce the network parameters by exploiting low rank matrix factorization. Liu *et al.* [Liu et al., 2015] exploit the decomposition of matrices to perform efficient convolutions with sparse kernel parameters. Jaderberg *et al.* [Jaderberg et al., 2014] as well as Denton *et al.* [Denton et al., 2014] approximate convolutional filters to achieve a faster runtime. Han *et al.* [Han et al., 2015] use connection pruning to reduce the number of network parameters. Wen *et al.* [Wen et al., 2016] reduce the parameters of a pre-trained network with structured sparsity learning.

Direct Convolutions. The works of Park *et al.* [Park et al., 2017], Engelcke *et al.* [Engelcke et al., 2017] and Parashar *et al.* [Parashar et al., 2017] are the most related ones to our approach, in that they also perform convolutions in a direct manner to efficiently exploit sparseness in network parameters and feature maps. While

Park *et al.* use compressed rows as sparse tensor format for the filter parameters, neither [Engelcke et al., 2017] nor [Park et al., 2017] uses a sparse tensor format for both filter parameters and feature maps. Parashar *et al.* [Parashar et al., 2017] implement sparse convolutions on a custom-designed hardware to achieve an energy- and memory-efficient deep CNN. Note that even though all three works follow a similar idea, only Parashar *et al.* exploit sparsity in both the parameters and the data, with compressed sparse blocks, but require dedicated, non-standard hardware.

6.3 Method

At a conceptual level, it is a general theme of computing to speed up computations and reduce memory usage by exploiting sparsity in the data. In the following section, we propose a number of ways to do the same for the specific case of neural network layers, always keeping in mind the specific requirements and limitations of modern GPU architectures. Throughout, sparse tensors are represented and manipulated in a format similar to *Coordinate List*[2], which stores indices into the sparsely populated grid and the corresponding data entries in separate tensors, and is available in the "SparseTensor" implementation of *Tensorflow*. To minimise memory overhead, the indices of the form $\{batch, index_x, index_y, ..., channel\}$ are compressed into unique $1D$ keys and only expanded when needed.

In order to achieve coalesced memory access, which permits efficient caching and thus fast memory access, the tensors for feature maps are sorted w.r.t. batches and within each batch w.r.t. channels. Likewise, filter weights are sorted w.r.t. the output channels and within each channel *w.r.t.* the input channels. Compared to dense tensors, the sparse representation naturally adds some overhead. For instance, in our implementation we use 64 bit keys, and 32 bit depth for feature maps. Consequently, storing a dense feature map (100% density) required $3\times$ more memory. For densities $<33\%$ the sparse representation is more efficient, and at low densities the savings can be quite dramatic, e.g., at density 1% the sparse version uses 97% less memory.

6.3.1 Sparse Convolution

Our convolutional layer is designed to work with sparse tensors for both feature maps and filter weights. Feature maps are updated incrementally with *atomic operations*, c.f. algorithm 2, where atomic operations are small enough to be thread-safe, even if no locking mechanism is used. In that respect it is similar to two concurrent works

[2]We have also experimented with other sparse formats, like compressed sparse blocks; but found none of them to work as well, mainly due to limitations and idiosyncrasies of current GPU hardware.

[Parashar et al., 2017, Park et al., 2017]. In practice, the incremental update is limited by the current hardware design, since atomic operations are slightly slower than non-atomics: at present, off-the-shelf GPUs do not offer native support for atomic floating point operations in shared memory, although they do for more costly CAS instructions. Yet incremental updating is significantly faster, because it performs only the minimum number of operations necessary to obtain the convolution, while avoiding to multiply or add zeros.

Algorithm 2 Direct Sparse Convolution with Attention

```
decompress filter and data indices from 1D to kD
for b ∈ [0 : batch_count] do
    for oc ∈ [0 : out_channel_count] do
        initialize dense buffer with 0
        for ic ∈ [0 : in_channel_count]) do
            for {id, val} ∈ data(b, ic) do
                for {fid, fval} ∈ filter(oc, ic) do
                    compute uid with get_update_id(id, fid)
                    atomically add val · fval to buffer at uid
                end for
            end for
        end for
        get non-zero entries from buffer
        add bias to non-zero entries in buffer
        select k largest responses from non-zero entries
        compress ids of k largest responses from kD to 1D
        write k largest features and ids as sparse output
    end for
end for
```

The sparse convolution is computed sequentially per output channel and batch, but in parallel across input channels, features and filter weights. Its result is stored in a temporary, dense buffer with dimensions for batches and output channels set to 1, e.g., $\{d_b = 1,\ d_x,\ d_y,\ ...,\ d_c = 1\}$. This buffer increases with the resolution of the data, i.e., quadratically for 2D images, cubically for 3D volumes, *etc.* . Still, it is in practice a lot smaller than a typical dense tensor with correct dimensionality for batches and channels, such that volumes up to 512^3 can be processed on a single graphics card (Nvidia Titan Xp, 12 GB).

6.3.2 Preserving sparsity with attention

Convolution with kernels larger than (1×1) generates fill-in, i.e., it reduces the sparsity of a feature map, by construction. C.f. Figure 6.2. This "smearing out" of the sparse in-

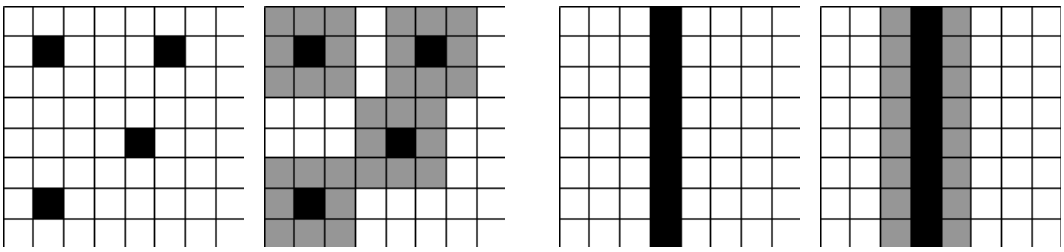


Figure 6.2: The fill-in (decrease in sparsity) due to convolutions depends on the data distribution. Uniformly distributed data is affected most strongly, e.g., in 3D data every $3 \times 3 \times 3$ filter would increase the density by a factor of 27, until data is dense.

puts usually only has a small influence on the output of the network, see section 6.4.2. But it considerably increases memory consumption and runtime, especially when occurring repeatedly over multiple layers. In order to guarantee upper bounds on the memory footprint and runtime of the network, we propose to apply a k-selection filter on each output channel, keeping only the k strongest responses. This can be seen as an approximation of the exact convolution output where small responses are suppressed, but using an adaptive threshold that supresses only as many values as necessary to maintain a desired degree of sparsity.

The parameter k controls the sparsity, and thus the memory consumption, of the convolutional layers. Processes that aim to optimally direct and manage the limited resources available for some cognitive task are commonly referred to as *attention*. We have implemented two versions of our simple attention mechanism via k-selection [Alabi et al., 2012]: *(i)* acts on the raw responses, so it prefers large positive responses, making it similar to a rectified linear unit; *(ii)* picks the k values with the largest absolute values, expressing a preference for responses with large magnitude. The time complexity of this layer is given by:

$$O\big((\rho_d \cdot \rho_f \cdot s_f^k \cdot c_{in} + log(s_d^k)) \cdot s_d^k \cdot c_{out} \cdot b\big), \tag{6.1}$$

where k is the dimension of the data, s_d its resolution and ρ_d its density, s_f denotes the size of the filter and ρ_f the density of the filter. The number of batches is b and the number of input and output channels of the layer are c_{in} and c_{out}, respectively.

6.3.3 Pooling Layer

Our sparse pooling layer goes through three straight-forward stages. First, features are assigned to an output (hyper-) voxel, by dividing the data channels of their index by strides. Second, the data is sorted w.r.t. voxels, so that all responses within the same voxel are clustered together. Third, the pooling operator is applied separately to each

cluster. So far, we have implemented only max-pooling. Clearly, the time complexity of the pooling layer is

$$O\big(\rho_d \cdot s_d^k \cdot log(\rho_d \cdot s_d^k) \cdot c_{in} \cdot b\big) \ . \tag{6.2}$$

6.3.4 Direct Sparse Backpropagation

Our target for back-propagation is again to skip operations that can be avoided due to sparsity. We must propagate error gradients only to all those features which have produced evidence, in the form of non-zero responses during the forward pass. Yet, the same performance issues already discussed for the forward pass apply also to the backward pass: back-propagation through a convolutional layer is itself a convolution and therefore produces fill-in, increasing memory use and runtime.

Contrary to the forward pass, it is not advisable to bound the fill-in with the k-selection technique, since this will not prevent the back-propagated error gradients from spreading to zero activations and vanishing, while smaller gradients flowing towards non-zero activations might be missed. It is evident that this effect could seriously slow down the training process. Hence, we propose to use a stricter back-propagation, which only propagates errors L to non-zero features x and model parameters w:

$$\frac{\partial L}{\partial x_i} = \begin{cases} 0 & \text{for } x_i = 0 \\ \frac{\partial L}{\partial y} \frac{\partial y}{\partial x_i}, & \text{else} \end{cases} \tag{6.3}$$

$$\frac{\partial L}{\partial w_i} = \begin{cases} 0 & \text{for } w_i = 0 \\ \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_i}, & \text{else} \end{cases} \tag{6.4}$$

Here, weights are considered equal to zero only if they have been explicitly removed by pruning, so as to avoid suppressing the gradients of weights that pass through $w_i = 0$ while changing sign. Note the similarity of our approximated back-propagation to the back-propagation through an arbitrary layer with $ReLU$ activation function:

$$\frac{\partial L}{\partial x} = \begin{cases} 0 & \text{for } y_i \leq 0 \\ \frac{\partial L}{\partial y_i'} \frac{\partial y_i}{\partial x}, & \text{else} \end{cases} \ , \tag{6.5}$$

where y_i denotes the output of the layer before applying the ReLU and y_i' denotes the output of the ReLU. Conventional back-propagation sets values to zero in function of the layer output y_i, whereas we do so as in function of the input features x_i.

Neglecting zero-elements slightly reduces the efficiency per learning iteration, since not all error gradients are propagated anymore. However, it has a number of advantages: *(i)* The tensors used for back-propagation have fixed size and shape. Therefore,

Algorithm 3 Backpropagation for Convolutional Layer

initialize *bp_data* with shape(*input_values*) and 0
initialize *bp_filter* with shape(*filter_weights*) and 0
decompress filter and data indices from 1D to kD
for $b \in [0 : \textit{batch\_count}]$ **do**
  for $oc \in [0 : \textit{out\_channel\_count}]$ **do**
    initialize dense *buffer* with gradients(b, oc)
    for $ic \in [0 : \textit{in\_channel\_count}]$) **do**
      for $\{id, val\} \in$ data(b, ic) **do**
        for $\{fid, fval\} \in$*filter*(oc, ic) **do**
          compute uid with get_update_id(id, fid)
          get gradient g from *buffer* at uid
          atomically add $g \cdot fval$ to *bp_data* at id
          atomically add $g \cdot val$ to *bp_filter* at fid
        end for
      end for
    end for
  end for
end for

one can still use optimization frameworks that have been designed for dense data, and expect fixed and known array dimensions; *(ii)* By considering only gradients on non-zero elements of the forward pass, back-propagation can be implemented in a clean and transparent manner. E.g. for convolutional layers one obtains Algorithm 3, which is very similar to Algorithm 2; *(iii)* Once a filter weight has been set to zero (see section 6.3.6 on pruning weights during learning), it will remain zero. Below, we will describe how this property can be used to guarantee that the network gets progressively faster at its task as the learning proceeds and it sees more training data.

6.3.5 Adaptive density regularisation

The methods in Sections 6.3.2 and 6.3.4 directly impose upper bounds on network density. There is, however, a computationally more efficient way to encourage sparsity of the feature maps, such that the sparsity thresholds are rarely exceeded in the first place, and the more costly k-selection step is avoided. That strategy makes use of the properties of the rectified linear unit ($ReLU$), which is the most popular activation function in modern CNNs. The $ReLU$, by definition, truncates negative activations to zero while leaving positive ones unchanged. This means that we only have to include a regularisation that pushes down the values (not magnitudes) of filter weights (and biases). By doing so, more weights will be driven into the negative region, where they are extinguished by the subsequent $ReLU$. Moreover, the same idea can be used to

reduce sparsity when it is not needed, and optimally use the available resources: when too few activations are > 0, one drives the filter weights up, so that fewer of them are suppressed by the $ReLU$. To achieve the desired effect, we simply add a bias b to the L_2-regularisation, so that the regulariser becomes $\sum(w+b)^2$. The scalar b is positive when the density ρ is too large, and negative when it is overly small:

$$b = \begin{cases} o + b_1 \cdot (\rho - \rho_{up}) & \rho > \rho_{up} \\ & \text{(exceeds available resources)} \\ -b_2 \cdot (\rho_{up} - \rho) & \rho \leq \rho_{up} \\ & \text{(not using available resources)} \end{cases} \tag{6.6}$$

with ρ_{up} the upper bound implied by the k-selection filter, and o, b_1, $b_2 \geq 0$ control parameters. The offset o adds an additional penalty for exceeding the available resources, since this case requires the use of the k-selection filter and, hence, increases the computational load.

6.3.6 Parameter Pruning

As explained above, our training algorithm has the following useful properties: *(i)* The regulariser in section 6.3.5 encourages small model parameters. *ii)* The sparse back-propagation in section 6.3.4 ensures that, once set to zero, model parameters do not reappear in later training steps. Together, these two suggest an easily controllable way to progressively favour sparsity during training: At the end of every training epoch we screen the network for weights w that are very small: $|w_i| < \epsilon$. If the magnitude of a weight w_i stays low for two consecutive epochs (meaning that it was already close to zero before, and that did not change during one epoch of training) we conclude that it has little influence on the network output and prune it (one-warning-shot pruning). We note that a small weight should not be pruned when first detected, without the "warning shot": it could have a large gradient and just happen to be at its zero-crossing from a large positive to a large negative value (or vice versa) at the end of the epoch. On the contrary, it is very unlikely to observe a weight exactly at its zero-crossing twice in a row in consecutive epochs.

Since a weight, once set to zero, will not reappear with our sparse back-propagation, every pruning reduces the number of non-zero weights, unless all remaining weights have relevant magnitude, in which case their number stays the same. It is thus guaranteed that the network become sparser, and therefore also faster at the task it is learning, as it sees more and more training data, c.f. equation (6.1). We note, it is well-documented that biological systems get faster at a specific task with longer training [Robertson, 2007, Nissen and Bullemer, 1987].

6.4 Evaluation

In this section we evaluate the impact of upper bounds and regularization on runtime and classification accuracy. The sparse network was implemented into the Tensorflow framework and programmed in C++/CUDA with a python interface. Our experiments were run on PCs with Intel Core i7 7700K processors, 64GB RAM and Titan Xp GPUs. For the evaluation of memory footprint and runtime of our convolutional layer a synthetic dataset of sparse random tensors is used and compared against the dense layers of tensorflow 1.4, which was compiled with Cuda 9.0 and CuDNN 6.0.

We conduct different experiments to evaluate the effects of our sparse network on classification accuracy: First, the impact of upper bounds on classification is evaluated by performing a grid search on the upper bound ρ_{up} in the convolutional layers. For this experiment the MNIST data set [LeCun et al., 1998] is used, as it is small enough to perform grid search in a reasonable amount of time and can be interpreted as sparse data (1D lines in 2D images). Second, the effects of pruning on runtime and classification accuracy are shown using the ModelNet data set [Wu et al., 2015], by varying the strength λ of the regularisation. Modelnet40 provides 3D CAD models of 40 different classes. Furthermore, the classification results of different baseline methods are compared on this data set. Training on Modelnet40 is performed for 90 epochs with a learning rate of 0.001. Minimisation is done by stochastic gradient descent with the $adagrad$ optimizer [Duchi et al., 2011].

6.4.1 Runtime and Memory Footprint

For the evaluation of runtime, convolutions are performed on a sparse voxel grid filled with random numbers. The resolution of the voxel grid r^3 is varied between $r = 16$ and $r = 256$. To achieve the expected data density of a 2D surface in a 3D voxelgrid the data density ρ as well as the upper bound on the per-channel density ρ_{up} are set to $\rho = \rho_{up} = \frac{1}{r}$. In order to compute the high resolution $r = 256$ with dense convolutional layers, the mini-batch size and channel size had to be set to one[3], while the number of output channels was set to 8. The density ρ_f of the filter weights is varied between 0.1 and 1. As baseline we use the convolutional layer of Tensorflow [Abadi et al., 2016], which performs convolutions via the fast Fourier transform and batched general matrix-matrix multiplication from cuBlas, as front end to cuDNN [Chetlur et al., 2014]. We note that processing only a single input channel does not play to the strength of our sparse network. Moreover, Tensorflow is able to use the full capability of the GPU, while our implementation is limited to performing operations in global memory, due to the weak support for atomic floating point operations in shared memory. These

[3]Protobuf limits the size of single tensors to 2 GB.

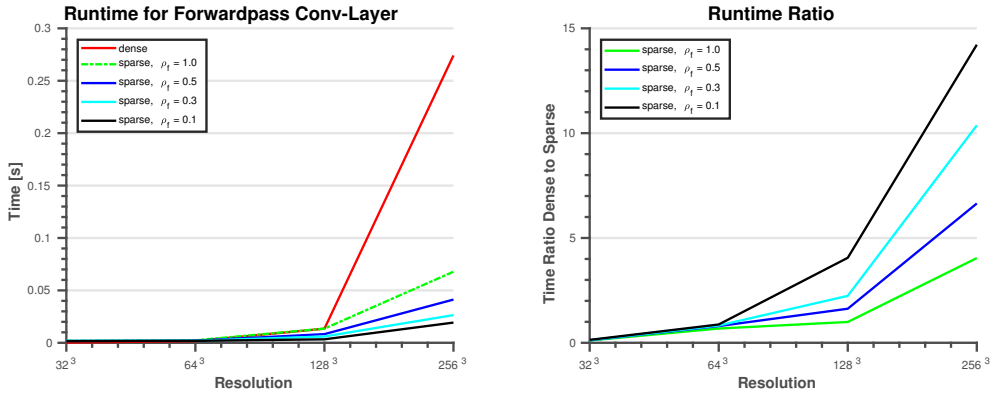


Figure 6.3: Runtime of a dense convolution layer in Tensorflow and our sparse convolution layer. The runtime *(left)* and runtime ratio *(right)* on random sparse input are plotted for different resolutions r^3. Both the density of the input and the upper bound for the sparse output are set to $\rho = \rho_{up} = \frac{1}{r}$. The density of the convolution kernel varies as $\rho_f \in \{0.1, 0.3, 0.5, 1\}$. At large resolutions the sparse layer is significantly faster.

Resolution	32^3	64^3	128^3	256^3
Dense [GB]	0.04	0.27	2.15	17.18
Sparse 32 [GB]	$2 \cdot 10^{-3}$	$8 \cdot 10^{-3}$	0.03	-
Sparse 64 [GB]	$3 \cdot 10^{-3}$	0.013	0.05	0.2
Sparse Temp [GB]	$3 \cdot 10^{-4}$	0.002	0.016	0.13

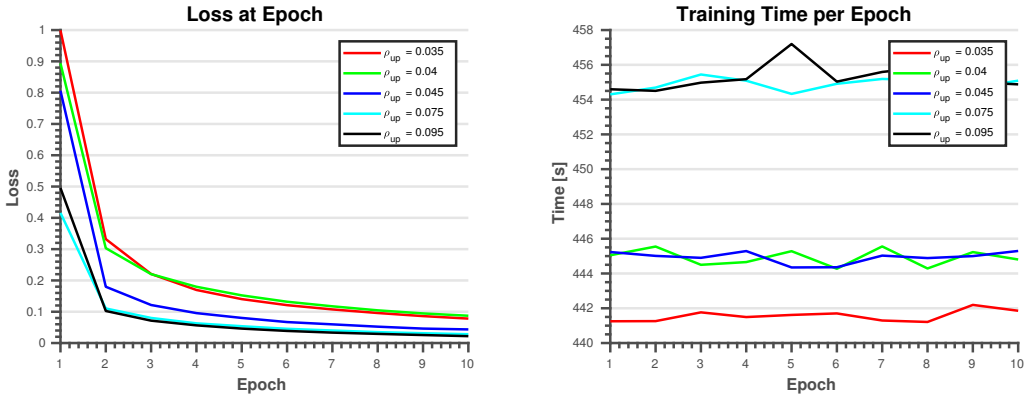
Table 6.1: Theoretical memory required at different resolutions r for a convolutional layer with minibatch size 32, output depth 8 and upper bound $\rho_{up} = \frac{1}{r}$.

advantages are seen especially at small resolutions r and large densities. In the worst case, at $r = 32$ tensorflow is $\approx 9\times$ faster than our implementation. In the best case, at $r = 256$ and $\rho_f = 0.1$, we are $\approx 14\times$ faster, as shown in Figure 6.3.

Table 6.1 shows the memory requirements for dense and sparse convolutional layers at various resolutions r, a minibatch size of 32, 8 output channels and an upper bound $\rho_{up} = \frac{1}{r}$. For our sparse representation we consider both the memory consumption with 64-bit indices and with 32-bit indices. At large resolutions our memory footprint is about two orders of magnitude smaller.

6.4.2 Contribution of small feature responses

In the context of sparsity the question arises, whether zero-valued features contribute valuable information. Two recent works tried to answer this question. On the one hand, Graham *et al.* [Graham and van der Maaten, 2017] found that they reach the same accuracies as dense networks for their application, while completely neglecting



(a) (b)

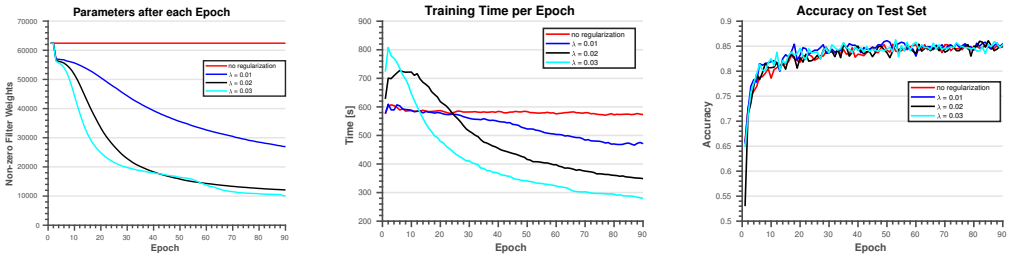
Figure 6.4: Effect of removing small features with upper bound ρ_{up}. (a) convergence of the training loss and *(b)* runtime, measured on the sparse MNIST data set. Stricter upper bounds slow down convergence, but reduce runtime.

zero-valued features. On the other hand, Uhrig [Uhrig et al., 2017] concluded that for certain tasks zero-valued features may be beneficial. For our network it is possible to assess the importance of small feature responses (not limited to exact zero-values) by training neural networks with varying upper bounds. For this experiment, CNNs are trained on MNIST for 10 epochs without regularization. The optimization is performed with $adagrad$ and a learning rate of 0.01.

The pixels in MNIST were set to zero when their value $v \in [0, 255]$ was below a threshold of $v < 50$, to obtain a sparse dataset with average density $\overline{\rho_{in}} = 0.23$, while the upper bound ρ_{up} ranges from $\rho_{up} = 0.035$ to $\rho_{up} = 0.095$. Note that even though letters can be interpreted as 1D lines in 2D images, the MNIST data has a low resolution of only 28×28 pixels. Hence, the data is still not extremely sparse. Except for few outliers, the results in Figure 6.4a show a slower converges of the training loss with lower upper bounds on the density of the layers. This reduced convergence is caused by dropping some gradients in the backpropagation, as described in section 6.3.4. However, smaller upper bounds not only guarantee a small memory footprint, but also tend to yield faster runtime, as seen in Figure 6.4b. The classification results on the test set differ by less than 1.5% between the strictest upper bound and the dense case.

6.4.3 Regularization and Pruning

With our sparsity-inducing pruning and regularisation, we expect to see faster runtime for stronger regularisers. In order to verify this behaviour, neural networks are trained on Modelnet40 with varying regularisation scales $\lambda \in \{0, 0.1, 0.2, 0.3\}$. The bias



(a) (b) (c)

Figure 6.5: Influence of adaptive density regularisation and pruning on *(a)* the number of (non-zero) parameters in the convolutional layer, *(b)* the runtime per training epoch, and *(c)* the overall accuracy on the Modelnet40 test set. Strong regularisation and pruning significantly reduce the number of parameters and the runtime, without noticeable impact on accuracy.

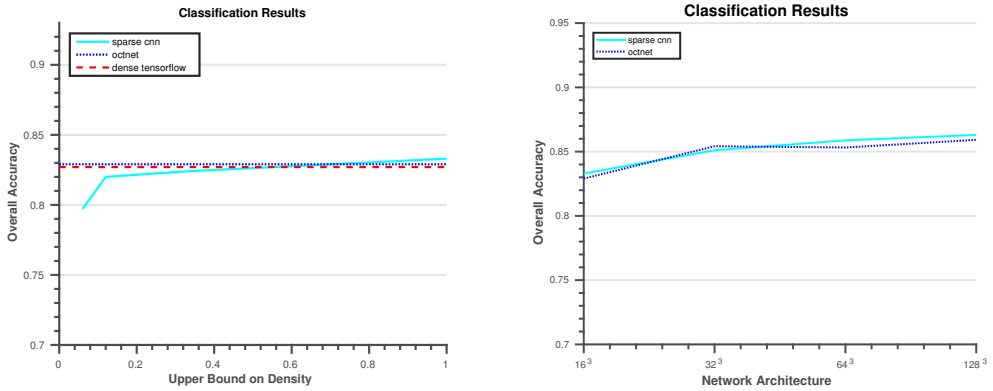


Figure 6.6: Performance of our sparse network on Modelnet40, compared to two baselines, the equivalent dense network and OctNet. *(left)* overall accuracy for different upper density bounds; *(right)* overall accuracy for different input resolutions.

for density-based regularization is computed with $b_1 = b_2 = o = 0.1$. Stronger regularisation decimates the number of (non-zero) filter weights faster, as shown in Figure 6.5a. It can also be seen that the number of parameters converges when only a fixed set of important weights is left. The drop in non-zero weights also reduces runtime, see Figure 6.5b.

After 90 epochs of training, a network regularised with $\lambda = 0.3$ is 51% faster than one trained without regularisation and pruning, even though only the first nine out of twelve convolutional layers are set to be sparse. Strong regularisation $\lambda \in \{0.2, 0.3\}$ initially causes an increase in runtime, by driving up the number of non-zero weights to use the available resources via the bias term b_2.. The classification accuracy for all tested regularization scales quickly converges to practically identical values, as shown in Figure 6.5c. We point out that pruning finds the most suitable sparsity pattern *for a*

given training set. When using a pruned model for transfer learning, it may be safer to re-initialize the removed filter weights of the sparse representation with zeros before fine-tuning.

6.4.4 Classification performance on Modelnet40

Finally, we compare our upper-bounded neural network and modified back-propagation against a conventional network. To that end we run OctNet3, a dense network without octree structure, and a sparse version of the same network on Modelnet40, see Figure 6.6. First, the input resolution is set to $r = 16^3$, while the upper bound on the density is varied between $\rho_{in} \in \{0.06, 0.12, 0.33, 1.0\}$. Both, the conventional dense network and Octnet converge to a very similar overall accuracy of ≈ 0.83. For a trivial upper bound $\rho_{in} = 1.0$ the overall accuracy of our sparse network is also practically the same (in fact slightly better at 0.833, but we attribute this to random fluctuations). Very low upper bounds up to $\rho_{in} = 0.12$ yield slightly worse results on the 16^3 inputs, for the lowest bound $\rho_{in} = 0.06$ the loss reaches ≈ 3 percent points. Second, the resolution of the input voxel grid is gradually increased: $r \in \{16^3, 32^3, 64^3, 128^3\}$. Both the sparse network and OctNet yield similar results, for all resolutions. OctNet performs slightly better on $r = 32^3$, while our bounded, sparse network has a small advantage at all other resolutions. Together the two experiments suggest that reasonable upper bounds and our approximated, sparse backpropagation do not negatively impact classification accuracy.

6.5 Conclusion

We have proposed a novel neural network mechanisms which exploit and encourage sparseness in both feature maps and model parameters. At practically useful resolutions, our novel sparse layers and back-propagation rule significantly reduce *(i)* memory footprint and *(ii)* runtime of convolutional layers for sufficiently sparse data. Moreover, our approach guarantees upper bounds on the memory requirements and runtime of the network. For classification tasks the performance of our sparse network is comparable to its dense counterpart as well as OctNet. In future work, it will be interesting to employ sparsity also for other tasks. Our implementation is fully compatible with Tensorflow and will be released as open-source code. We hope, that hardware support for sparse convolutions will improve further on future consumer GPUs, as demonstrated by [Parashar et al., 2017]; thus further boosting the performance of sparse, high-dimensional CNNs.

Chapter 7

Conclusion and Outlook

Supervised classification is one of the most popular areas in machine learning. The primary objective of supervised classification is to reach a small generalization error; *e.g.* have an excellent predictive performance on previously unseen data. At first glance, this objective bears similarities to the work of historians. Many historians try to use knowledge of the present and past to generate a good prediction of the future. One such prediction by Harari claims that mankind will reach happiness and longevity by means of technological advancement in fields like genetic engineering and machine learning, followed by an elaborate argumentation to support his claims [Harari, 2016]. This argumentation is model-based, e.g. *"developments in medicine have tried to prolong life in the past"*. Conclusions are derived by combining and extrapolating such simple models, e.g. *"hence, medical advances will continue to prolong life in the future"*. In social science, many models are often not verifiable. If possible, the verification of such long-term claims can take multiple decades. In contrast, statistical learning infers information directly from data. Typically, the performance of statistical learners is easily verifiable. However, complex reasoning remains a problem in machine learning. The inference of seemingly simple relations turns out to be surprisingly difficult to learn so that an integration of the ability to infer relation into powerful neural networks remains active research [Santoro et al., 2017, Henaff et al., 2015].

Throughout the last decade, the biggest advances in computer vision have been seen in tasks that do not depend on complex reasoning, including classification tasks in 3D vision. These successes in the field of 3D scene understanding, include: semantic classification with hand-crafted features [Blomley et al., 2014, Weinmann et al., 2015a], deep learning in voxel grids [Maturana and Scherer, 2015, Wu et al., 2015], convolutional neural networks that exploit sparseness in voxel grids with OcTrees [Riegler et al., 2017, Tatarchenko et al., 2017] and graph convolutions that allow inferring relations between different objects in a 3D scene [Landrieu and Simonovsky, 2017, Qi et al., 2017b].

The contributions made in this thesis have a focus on efficient machine learning for 3D data. Computational efficiency is mandatory when it comes to the processing of large-scale datasets, e.g. typical real-world databases of terrestrial laser scans. To this end, we provide both traditional learning techniques and neural networks that are significantly more efficient than their baseline methods. We then go one step further by demonstrating, that gains in efficiency often can be transferred to a higher precision and recall on an application side. Furthermore, we make different approaches comparable by providing a large-scale benchmark test.

7.1 Technical evolution over the thesis

This thesis is the result of a three-year study. The main contributions were completed sequentially and correspond to the three core-chapters of this thesis (Chapter 4, 5 and 6). Each subsequent contribution refines and extends the previous findings. A detailed description of the evolution of our work is given in the following.

Semantic point cloud labeling

Chapter 4 starts with an implementation of a then state-of-the-art 3D semantic segmentation approach. For this purpose we extend the work of [Weinmann et al., 2015a], a traditional machine learning approach, such that it is efficiently applicable to large point clouds with billions of points. The feature extraction of our approach is implemented with optimized *C++* code that is parallelized with *OpenMP*. For discriminative learning the *ETHZ Random Forest Template Library* is deployed. A computational bottleneck of the original work is the computation of each point's large neighborhood; a requirement for feature extraction. We extract this neighborhood efficiently by deploying a pyramid of subsequently coarser sampled kd-trees while searching for a small number of neighbors within each stage of the pyramid. As a side effect, we obtain a multi-scale feature representation.

The point clouds of the training set are subsampled to reduce the effects of laser scanner position on the class frequency in the training set. A subsampled, smaller training set ameliorates the runtime of the training step. We determine the depth of the random forest with grid search and k-fold cross-validation. With this implementation, it is possible to infer a class label for each individual point of large, real-world terrestrial laser scans. Recent developments show that neural networks can outperform traditional machine learning approaches in overall accuracy on many tasks. Nevertheless, traditional machine learning works well with a small amount of training data and requires less

computational resources than deep learning approaches. Hence, it will continue to play a role in tasks where training data or computational hardware is limited.

Contour detection in point clouds

Contour detection in 3D it is not well defined: A definition of contours that works well on buildings might yield unsatisfying results on cars or in vegetation. In Chapter 4, we introduce a convenient method to define what constitutes to contours by generating a hand-labeled training set. With a contour-definition given as training data, our semantic labeling algorithm can infer pointwise contour scores in unseen data. However, contours are elongated structures so that it is desirable to consider long-range interactions rather than individual points. We achieve to obtain a *contour graph* by applying the findings of [Türetken et al., 2011] to point clouds: *i)* find seed points that have a high contour-score with Non-Maximum Suppression, *ii)* link neighboring seed points with a shortest paths algorithm to obtain an overcomplete set of contour candidates, *iii)* filter false positives from the set of contour candidates by deploying another discriminative learner.

Benchmark for point cloud classification

The multitude of machine learning approaches to semantic segmentation raises the question if our method is the best-suited approach to infer class labels of points in large 3D point clouds. Chapter 5 gives insights on this question by comparing different approaches: We make a large dataset available in the form of a benchmark test. For this purpose, we provide three baseline methods: *i)* an approach that generates 2D images and classifies the images with context-aware features and boosting; *ii)* A deep learning based method that deploys voxel grids; *iii)* Our already introduced 3D semantic labeling algorithm that is based on traditional machine learning. To guarantee a high-quality evaluation, the ground truth of the test set is held back. Authors can submit their results through a web interface so that all submissions undergo the same evaluation procedure.

From traditional machine learning to deep learning

Submissions from the benchmark test show that deep learning tends to achieve better results than traditional machine learning on semantic segmentation tasks. However, neural networks are computationally demanding. In Chapter 6, we follow two strategies to reduce the computational requirements: *i)* Exploit the sparsity in data: Typi-

cally, sensors capture 3D point clouds with line-of-sight measurements. Hence, they represent 3D points on a 2D surface. The representation of a 2D surface in a 3D voxel grid yields sparse data. *ii)* Exploit the sparsity in the model: Many model parameters of neural networks are small so that they have an insignificant impact on the activations. Such small model parameters can be ignored. A direct, sparse convolutional neural network is implemented following recent works from [Parashar et al., 2017, Engelcke et al., 2017, Park et al., 2017] that exploits these effects. Possibly the most significant problem when exploiting sparsity in neural networks is the tendency of convolutional layers to fill-in data. This fill-in effect significantly decreases the sparseness. We propose to deploy a simple filter mechanism that only pays attention to the largest filter responses while removing small activations from the activation maps. The source code is published as open source project with optimized *C++/CUDA* functions that make use of the massive parallel computing power of modern GPUs.

7.2 Discussion of contributions

Throughout this study, we have explored the suitability of traditional machine learning techniques as well as neural networks on solving 3D vision problems. On a theoretical side, the focus is efficiency. On a practical side, we focus on semantic classification and contour detection.

Semantic point cloud labeling

Over the course of this thesis, we contributed twice to the field of semantic point cloud labeling: *i)* with a fast semantic segmentation approach in Chapter 4 and *ii)* with a benchmark test in Chapter 5. In the following, these contributions are discussed.

Fast semantic segmentation. Our baseline method [Weinmann et al., 2015a] computes covariance based features by deploying large neighborhoods. Their approach to efficiency is to select a subset of features and neighborhoods based on feature relevance assessment and only compute the subset. We demonstrate that feature relevance assessment is not the critical bottleneck to optimize by introducing an efficient way to calculate point neighborhoods for feature generation. Our neighborhood computation is based on a pyramid of kd-trees, where each subsequent kd-tree is constructed from coarser sub-sampled point clouds. In [Hackel et al., 2016b], we show that our implementation is nearly *two orders of magnitude faster* than the baseline, compare Table 7.1. Furthermore, our feature set is better suited to discriminate between classes, where we have identified two main factors. First, we do not use a subset of features and

neighborhoods for training. Instead, the full multi-scale feature set is deployed; Unnecessary features are identified and excluded only after the training, not before. Second, the subsampling in the kd-tree pyramid is based on voxel grid filters so that subsampled point clouds are closer to a uniform distribution than the original point clouds. Experiments on the Paris-rue-Madame and the Paris-rue-Cassette dataset [Serna et al., 2014, Vallet et al., 2015] show that our multi-scale feature set *outperforms* the baseline in overall accuracy and mean class recall, compare Table 7.2. On the F_1-score we are more than 18% better. Figure 7.1 visualizes these results.

Paris-rue-Cassette	[Weinmann et al., 2015b]	our work
features	23,000 s	191 s
training	2 s	16 s
classification	90 s	60 s
total	23'092 s	267 s

Table 7.1: Computation times for processing the *Paris-Rue-Cassette* database on a single PC. Our feature computation is much faster. The comparison is indicative, implementation details and hardware setup may differ. (Source [Hackel et al., 2016b])

Benchmarking. With our benchmark test, it is possible to compare a multitude of approaches to 3D point cloud processing. These approaches cover both traditional machine learning and deep learning. Our initial contribution is a large dataset with over four billion manually labeled points. Furthermore, an initial comparison of *i)* a pure image-based classification method, *ii)* a deep learning approach and *iii)* our fast semantic segmentation is provided. In the initial comparison, our traditional machine learning approach showed best results. Recently[1], there have been multiple successful submissions to our benchmark test as shown in Table 7.3 and Table 7.4. Many submissions have not described their methodology publicly, e.g. methods that are still under review or industrial submissions. However, those scientific submissions with published methodology yield interesting insights. DeePr3SS [Lawin et al., 2017] and SnapNet [Boulch et al., 2017] propose to render virtual images from point clouds and classify these images with convolutional neural networks, where they benefit from the large amount of training data available for 2D problems. SEGCloud [Tchapmi et al., 2017] introduces an end-to-end learning framework based on a CNN and CRF to classify directly in 3D space. Their approach outperforms the 2D image based approaches. This shows that 3D representations are richer than 2D representations for 3D classification problems. The currently best approach partitions point cloud into super points and performs graph convolutions with gated recurrent units [Landrieu and Simonovsky, 2017]. Their method achieves a high performance on the Intersection over

[1] State: 8th of February 2018.

Paris-rue-Madame	Our Method			[Weinmann et al., 2015b]		
	Recall	Precision	F_1	Recall	Precision	F_1
Façade	0.9799	0.9902	0.9851	0.9527	0.9620	0.9573
Ground	0.9692	0.9934	0.9811	0.8650	0.9782	0.9182
Cars	0.9786	0.9086	0.9423	0.6476	0.7948	0.7137
Motorcycles	0.9796	0.4792	0.6435	0.7198	0.0980	0.1725
Traffic signs	0.9939	0.3403	0.5070	0.9485	0.0491	0.0934
Pedestrians	0.9987	0.2414	0.3888	0.8780	0.0163	0.0320
Overall accuracy		0.9755			0.8882	
Mean class recall		0.9833			0.8353	
Mean F_1-score		0.7413			0.4812	

Paris-rue-Cassette	Our Method			[Weinmann et al., 2015b]		
	Recall	Precision	F_1	Recall	Precision	F_1
Façade	0.9421	0.9964	0.9685	0.8721	0.9928	0.9285
Ground	0.9822	0.9871	0.9847	0.9646	0.9924	0.9783
Cars	0.9307	0.8608	0.8943	0.6112	0.6767	0.6423
Motorcycles	0.9758	0.5199	0.6784	0.8285	0.1774	0.2923
Traffic signs	0.8963	0.1899	0.3134	0.7657	0.1495	0.2501
Pedestrians	0.9687	0.2488	0.3960	0.8225	0.0924	0.1661
Vegetation	0.8478	0.5662	0.6790	0.8602	0.2566	0.3953
Overall accuracy		0.9543			0.8960	
Mean class recall		0.9348			0.8178	
Mean F_1-score		0.7020			0.5218	

Table 7.2: Quantitative results for iQmulus / TerraMobilita and Paris-Rue-Madame databases. (Source [Hackel et al., 2016b])

Method	$\overline{IoU}$	OA	$t[s]$	IoU_1	IoU_2	IoU_3	IoU_4	IoU_5	IoU_6	IoU_7	IoU_8
SPGraph	**0.762**	**0.929**	**10000.00**	**0.915**	0.756	**0.783**	**0.717**	**0.944**	**0.568**	**0.529**	**0.884**
SnapNet	0.674	0.910	unknown	0.896	**0.795**	0.748	0.561	0.909	0.365	0.343	0.772
HarrisNet	0.623	0.881	unknown	0.818	0.737	0.742	0.625	0.927	0.283	0.178	0.671
TMLC-MS	0.494	0.850	38421	0.911	0.695	0.328	0.216	0.876	0.259	0.113	0.553
TML-PC	0.391	0.745	unknown	0.804	0.661	0.423	0.412	0.647	0.124	0.0	0.058

Table 7.3: Semantic3d benchmark results on the full dataset: For information on the methods refer to the text. *IoU* for categories (1) man-made terrain, (2) natural terrain, (3) high vegetation, (4) low vegetation, (5) buildings, (6) hard scape, (7) scanning artefacts, (8) cars.

Figure 7.1: Results from iQmulus / TerraMobilita dataset. *Top row:* Paris-Rue-Madame (left) and Paris-Rue-Cassette (right) with classes: façade, ground, cars, motorcycles, traffic signs, pedestrians and vegetation. *Bottom row:* Further results without ground truth. (Source [Hackel et al., 2016b])

Method	$\overline{IoU}$	OA	$t[s]$	IoU_1	IoU_2	IoU_3	IoU_4	IoU_5	IoU_6	IoU_7	IoU_8
SPGraph	**0.732**	**0.940**	3000.00	**0.974**	**0.926**	**0.879**	**0.440**	**0.932**	0.310	**0.635**	0.762
MSDeepVoxNet	0.653	0.884	115000.00	0.830	0.672	0.838	0.367	0.924	**0.313**	0.500	**0.782**
RF_MSSF	0.627	0.903	1643.75	0.876	0.803	0.818	0.364	0.922	0.241	0.426	0.566
SEGCloud	0.613	0.881	1881.00	0.839	0.660	0.860	0.405	0.911	0.309	0.275	0.643
SnapNet	0.591	0.886	3600	0.820	0.773	0.797	0.229	0.911	0.184	0.373	0.644
OctreeNet_CRF	0.591	0.899	**184.84**	0.907	0.820	0.824	0.393	0.900	0.109	0.312	0.460
DeePr3SS	0.585	0.889	unknown	0.856	0.832	0.742	0.324	0.897	0.185	0.251	0.592
3D-FCNN-TI	0.582	0.875	774	0.840	0.711	0.770	0.318	0.899	0.277	0.252	0.590
DeepVoxNet	0.571	0.848	100000.00	0.827	0.531	0.838	0.287	0.899	0.236	0.298	0.650
DLUT_SR	0.563	0.860	unknown	0.953	0.849	0.548	0.296	0.832	0.192	0.320	0.518
TMLC-MSR	0.542	0.862	1800	0.898	0.745	0.537	0.268	0.888	0.189	0.364	0.447
DeepNet	0.437	0.772	64800	0.838	0.385	0.548	0.085	0.841	0.151	0.223	0.423
TML-PCR	0.384	0.740	unknown	0.726	0.73	0.485	0.224	0.707	0.050	0.0	0.15

Table 7.4: Semantic3d benchmark results on the reduced dataset: For information on the methods refer to the text. *IoU* for categories (1) man-made terrain, (2) natural terrain, (3) high vegetation, (4) low vegetation, (5) buildings, (6) hard scape, (7) scanning artefacts, (8) cars.

Union scores, e.g. $\overline{IoU} = 0.732$ on the reduced challenge which is 19% higher than the performance of our traditional machine learning approach.

Contour detection in unstructured point clouds

In chapter 4, we provide a novel approach to contour detection in unstructured point clouds. First, we provide a definition of contours in the form of training data. This training data contains both, class specific contour definitions as well as the signal to noise ratio of the data. Second, a 3D semantic segmentation algorithm is trained to infer lowlevel evidence in the form of a probability score for individual points. Third, long-range interactions are considered by deploying overcomplete *contour graphs* and another discriminative learning stage that prunes false positives in the contour graph. The method was tested on point clouds with more than $\sim 10^7$ points where it significantly outperforms a Canny-like baseline method on complex real-world data. The processing of millions of points just takes a couple of minutes. The fast processing of point clouds is a mandatory requirement for lowlevel tasks like contour extraction: Our contour extraction approach benefits significantly from the computational efficiency of our semantic segmentation method.

Deep learning

Finally, our contribution to deep learning is two-fold: *i)* In Chapter 5, we provide a convolutional network as baseline method to 3D semantic segmentation for our bench-

mark test. Though the results of the baseline methods are unsatisfying, the limitation of our network to processing small input resolutions indicates a need for deep learning frameworks capable of exploiting sparseness in data; *ii)* Chapter 6 introduces an efficient GPU implementation of a neural network that exploits sparseness both in data and in model parameters. One problem of exploiting sparseness in neural networks is the decrease in the sparseness of data due to convolutional layers and pooling layers. We introduce a novel mechanism to preserve sparseness by deploying a filter step in the convolutional layer complemented by density-based regularization. Furthermore, a pruning step encourages sparseness in model parameters. Experiments show that our approach is faster than dense *Tensorflow* on data with a high degree of sparseness while requiring significantly less memory.

7.3 Limitations of our approach

This thesis makes contributions to the field of machine learning for large-scale 3D data. However, there exist weaknesses and shortcomings in our approaches. In this section, we describe these limitations.

Semantic Segmentation for Point Clouds

Processing speed. On a practical side, the runtime is a crucial requirement for a multitude of applications; most prominently autonomous driving. Our evaluation reveals that our approach is significantly faster than our baseline method and reaches near-real time. Nevertheless, the processing of unstructured 3D data requires demanding models, compare Chapter 4. Our implementation turns out to be too demanding to fulfill hard real-time constraints with decent hardware. Especially, feature extraction and classification are a high computational burden, compare Table 7.1. It is certainly possible to reduce the computational demand of 3D classification approaches by reducing the size of point clouds, e.g. by uniform subsampling. However, this comes at the cost that identification of small objects becomes more challenging.

Hand-crafted features. Traditional machine learning requires the design of discriminative features. The performance of such hand-crafted features has a high dependency on the tasks they are designed for. Our feature set is designed to be generic and applicable to several tasks. However, the evaluation in Chapter 4 shows a bad performance in discriminating between some object classes, e.g. *Low Vegetation* and *Natural Ground*. This calls for a more discriminative feature set; a task-specific and time-consuming development task.

Object classes. Although our feature set is generic and should be applicable to arbitrary 3D classification tasks, our experiments in Chapter 4 and our benchmark test in Chapter 5 only cover up to 8 object classes. The main problem is the availability and generation of large-scale ground truth annotated data sets. It is more challenging to discriminate between a larger variety of object classes. Hence, large-scale 3D datasets with more object classes can help to identify problems in approaches that work well on datasets with fewer object classes.

3D Contour Detection

Generalization. Probably the biggest challenge to any classification task is the generalization to unseen data, e.g. unseen object classes. In our experiments in Chapter 4 we encounter objects with a strong geometric structure that are not covered by the training set, e.g. windows with sun blinds. On these objects, our contour detection shows undefined behavior and produces poor results by causing many false positive detections.

Pruning of contour graph. While the overcomplete graph of contour candidates covers nearly all true positive contours, it also contains a multitude of false candidates. Our second classification stage finds most true positives but fails to remove a small portion of the false positives; This causes visually unappealing contours.

Sparse Deep Learning

Network layers. Modern neural networks live from a rich set of layers. Each such layer implements a differentiable function. Within our study on sparse networks, we were limited to implementing only the most important layers, e.g. Convolutional Layers and Pooling Layers. This limits the usability of our implementation to basic network architectures.

Hardware limitations. Direct convolutions exploit sparseness by performing atomic operations. However, the support for atomic operations in GPUs[2] is poor: Our GPU does not support atomic floating point additions in shared memory. Hence, we were limited to an implementation in slow global memory.

[2]Titan Xp PASCAL

7.4 Future directions

In the following, we briefly discuss potential solutions to overcome the limitations of our approaches *w.r.t.* technical features and methodology. These solutions delineate research proposals that can be used for future projects.

Semantic Segmentation for Point Clouds

Processing speed. Both *hardware* and *software* approaches can help to gain runtime performance for our semantic segmentation approach. On the hardware side, the computational power of GPUs can be used to parallelize the feature extraction and the discriminative learner massively. On the software side, feature extraction continues to be the most demanding step in our pipeline. Neighborhood queries with sophisticated search structures, like OcTrees, have not been evaluated in depth through this thesis. Such structures appear to have the potential to reduce computational complexity further.

Discriminative Features. Representation learning is the obvious approach to overcome the design of hand-crafted features. Superpoint Graph [Landrieu and Simonovsky, 2017] is one of the most promising, recent approaches to large-scale semantic segmentation. Their power comes from a combination of fixed size descriptors and graph convolutions with gated recurrent units. Our sparse convolutional network can help to generate such descriptors efficiently.

Challenging datasets. In the past year, the performance of submissions to our benchmark test has significantly improved and reaches good results. Especially, in representation learning, challenging datasets that are difficult for state of the art algorithms drive innovations. Such challenging datasets can be *i)* large in size, e.g. by deploying registered scenes rather than individual scans; *ii)* have a large number of classes, e.g. in the form of a class hierarchy; *iii)* cover different sensors.

3D Contour Detection

Lowlevel evidence. Our fast semantic segmentation algorithm is used to generate lowlevel evidence for contour detection. This lowlevel evidence in the form of a probabilistic contour score is essential for all further processing steps of our contour detection pipeline. Recently, there have been multiple improvements made to semantic segmentation [Landrieu and Simonovsky, 2017, Tchapmi et al., 2017]. This raises the question if these improvements to semantic segmentation can be passed on to contour extraction by generating better lowlevel evidence.

Pruning of contour graph. A second discriminative classification stage prunes the contour graph. While this discriminative classifier yields good results on a test set, there are still flaws left in the visual evaluation. One possible cause is the hand-labeled training set of small size; For the training of this classifier a self-supervised learning approach can help to create a larger and more representative training set automatically, e.g. as described in [Türetken et al., 2012].

Sparse Deep Learning

Network layers. Many network architectures rely on network layers that are not implemented in our sparse convolutional network yet. Hence, our network can significantly benefit from implementations of more sparse layers, most prominently skip-connections and batch normalization.

Hardware issues. Atomic operations are the limiting factor of our sparse neural network. However, there already exists hardware that can improve the performance of convolutions by improving atomic operations and their data flow [Parashar et al., 2017]. It is highly desirable that next-generation consumer products make such hardware accelerated sparse convolutions broadly available.

Outlook

Typically, machine learning approaches outperform methods that rely on hand-crafted rules for point cloud processing. Recent techniques start to approach human performance in a multitude of 3D vision tasks. While machine learning plays a central role in modern science, the industry only recently started to follow this trend. Thus, machine learning will play a more central role in many industrial products that depend on the processing of large point clouds. Certainly, for robots and autonomous cars, it is mandatory to distinguish between objects in sensor data. However, 3D machine learning will not be limited to prestigious products with media coverage like autonomous cars. Measurements on construction sites can be taken faster when the measurement devices deploy semantic cues. Furthermore, surveying plays an important role for the maintenance of infrastructure, where machine learning and semantic segmentation can help to predict faults before severe damage happens. Another field for future applications is augmented and virtual reality where semantic cues will help not just to perform random augmentations but to have a meaningful augmentation of the world.

Bibliography

[Abadi et al., 2016] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA*.

[Alabi et al., 2012] Alabi, T., Blanchard, J. D., Gordon, B., and Steinbach, R. (2012). Fast k-selection algorithms for graphics processing units. *Journal of Experimental Algorithmics (JEA)*, 17:4–2.

[Amit and Geman, 1997] Amit, Y. and Geman, D. (1997). Shape quantization and recognition with randomized trees. *Neural computation*, 9(7):1545–1588.

[Amunts et al., 2016] Amunts, K., Ebell, C., Muller, J., Telefont, M., Knoll, A., and Lippert, T. (2016). The human brain project: creating a european research infrastructure to decode the human brain. *Neuron*, 92(3):574–581.

[Armeni et al., 2017] Armeni, I., Sax, S., Zamir, A. R., and Savarese, S. (2017). Joint 2d-3d-semantic data for indoor scene understanding. *arXiv preprint arXiv:1702.01105*.

[Armeni et al., 2016] Armeni, I., Sener, O., Zamir, A. R., Jiang, H., Brilakis, I., Fischer, M., and Savarese, S. (2016). 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1534–1543.

[Badrinarayanan et al., 2015] Badrinarayanan, V., Kendall, A., and Cipolla, R. (2015). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*.

[Baldauf and Desimone, 2014] Baldauf, D. and Desimone, R. (2014). Neural mechanisms of object-based attention. *Science*, 344(6182):424–427.

[Belongie et al., 2002] Belongie, S., Malik, J., and Puzicha, J. (2002). Shape matching and object recognition using shape contexts. *IEEE transactions on pattern analysis and machine intelligence*, 24(4):509–522.

[Besag, 1986] Besag, J. (1986). On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 259–302.

[Bishop, 2006] Bishop, C. M. (2006). Pattern recognition and machine learning.

[Blomley et al., 2016] Blomley, R., Jutzi, B., and Weinmann, M. (2016). Classification of airborne laser scanning data using geometric multi-scale features and different neighbourhood types. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, 3(3).

[Blomley et al., 2014] Blomley, R., Weinmann, M., Leitloff, J., and Jutzi, B. (2014). Shape distribution features for point cloud analysis-a geometric histogram approach on multiple scales. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2(3):9.

[Bódis-Szomorú et al., 2015] Bódis-Szomorú, A., Riemenschneider, H., and Van Gool, L. (2015). Superpixel meshes for fast edge-preserving surface reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2011–2020.

[Bogoslavskyi and Stachniss, 2017] Bogoslavskyi, I. and Stachniss, C. (2017). Efficient online segmentation for sparse 3d laser scans. *PFG–Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, 85(1):41–52.

[Bottou, 2010] Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

[Boulch et al., 2017] Boulch, A., Le Saux, B., and Audebert, N. (2017). Unstructured point cloud semantic labeling using deep segmentation networks. In *Eurographics 3DOR*. The Eurographics Association.

[Boulch and Marlet, 2012] Boulch, A. and Marlet, R. (2012). Fast and robust normal estimation for point clouds with sharp features. In *Computer graphics forum*, volume 31, pages 1765–1774. Wiley Online Library.

[Boyko and Funkhouser, 2011] Boyko, A. and Funkhouser, T. (2011). Extracting roads from dense point clouds in large scale urban environment. *ISPRS Journal of Photogrammetry and Remote Sensing*, 66(6):S2–S12.

[Boykov and Kolmogorov, 2004] Boykov, Y. and Kolmogorov, V. (2004). An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *Transactions on Pattern Analysis and Machine Intelligence*.

[Breiman, 1996] Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140.

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

[Breiman et al., 1984] Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and regression trees*. CRC press.

[Briese, 2004] Briese, C. (2004). Three-dimensional modelling of breaklines from airborne laser scanner data. *ISPRS Archives*, 35(B3).

[Brock et al., 2016] Brock, A., Lim, T., Ritchie, J. M., and Weston, N. (2016). Generative and discriminative voxel modeling with convolutional neural networks. *CoRR*, abs/1608.04236.

[Brodu and Lague, 2012] Brodu, N. and Lague, D. (2012). 3D terrestrial lidar data classification of complex natural scenes using a multi-scale dimensionality criterion: Applications in geomorphology. *ISPRS Journal of Photogrammetry and Remote Sensing*, 68:121–134.

[Buluç et al., 2009] Buluç, A., Fineman, J. T., Frigo, M., Gilbert, J. R., and Leiserson, C. E. (2009). Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, pages 233–244. ACM.

[Canny, 1986] Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698.

[Cavegn et al., 2014] Cavegn, S., Haala, N., Nebiker, S., Rothermel, M., and Tutzauer, P. (2014). Benchmarking high density image matching for oblique airborne imagery. In *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, volume XL-3, pages 45–52.

[Changpinyo et al., 2017] Changpinyo, S., Sandler, M., and Zhmoginov, A. (2017). The power of sparsity in convolutional neural networks. *arXiv preprint arXiv:1702.06257*.

[Charaniya et al., 2004] Charaniya, A. P., Manduchi, R., and Lodha, S. K. (2004). Supervised parametric classification of aerial lidar data. In *Computer Vision and Pattern Recognition Workshop*, pages 30–30. IEEE.

[Chehata et al., 2009] Chehata, N., Guo, L., and Mallet, C. (2009). Airborne lidar feature selection for urban classification using random forests. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 38(Part 3):W8.

[Chen et al., 2017] Chen, S., Tian, D., Feng, C., Vetro, A., and Kovačević, J. (2017). Contour-enhanced resampling of 3d point clouds via graphs. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2941–2945. IEEE.

[Chetlur et al., 2014] Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., and Shelhamer, E. (2014). cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*.

[Choe et al., 2013] Choe, Y., Shim, I., and Chung, M. J. (2013). Urban structure classification using the 3d normal distribution transform for practical robot applications. *Advanced Robotics*, 27(5):351–371.

[Choi et al., 2013] Choi, C., Trevor, A. J., and Christensen, H. I. (2013). Rgb-d edge detection and edge-based registration. In *International Conference on Intelligent Robots and Systems*, pages 1568–1575. IEEE.

[Collobert et al., 2011] Collobert, R., Kavukcuoglu, K., and Farabet, C. (2011). Torch7: A matlab-like environment for machine learning. In *BigLearn, Advances in Neural Information Processing Systems Workshop*.

[Comaniciu and Meer, 2002] Comaniciu, D. and Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

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

[Crivellato and Ribatti, 2007] Crivellato, E. and Ribatti, D. (2007). Soul, mind, brain: Greek philosophy and the birth of neuroscience. *Brain research bulletin*, 71(4):327–336.

[Cui et al., 2017] Cui, Y., Ahmad, S., and Hawkins, J. (2017). The htm spatial poolera neocortical algorithm for online sparse distributed coding. *Frontiers in Computational Neuroscience*, 11.

[Dai et al., 2017] Dai, A., Chang, A. X., Savva, M., Halber, M., Funkhouser, T., and Nießner, M. (2017). Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1.

[Dalponte et al., 2012] Dalponte, M., Bruzzone, L., and Gianelle, D. (2012). Tree species classification in the southern alps based on the fusion of very high geometrical resolution multispectral/hyperspectral images and lidar data. *Remote sensing of environment*, 123:258–270.

[Daniel Girardeau-Montaut, CloudCompare, 2016] Daniel Girardeau-Montaut, CloudCompare (2016). `http://www.danielgm.net/cc/`.

[De Deuge et al., 2013] De Deuge, M., Quadros, A., Hung, C., and Douillard, B. (2013). Unsupervised feature learning for classification of outdoor 3d scans. In *Australasian Conference on Robitics and Automation*, volume 2.

[Demantké et al., 2011] Demantké, J., Mallet, C., David, N., and Vallet, B. (2011). Dimensionality based scale selection in 3d lidar point clouds. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 38(Part 5):W12.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE.

[Denil et al., 2013] Denil, M., Shakibi, B., Dinh, L., de Freitas, N., et al. (2013). Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pages 2148–2156.

[Denton et al., 2014] Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. (2014). Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277.

[Dohan et al., 2015] Dohan, D., Matejek, B., and Funkhouser, T. (2015). Learning hierarchical semantic segmentations of lidar data. In *International Conference on 3D Vision*, pages 273–281. IEEE.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

[Elseberg et al., 2013] Elseberg, J., Borrmann, D., and Nüchter, A. (2013). One billion points in the cloud–an octree for efficient processing of 3d laser scans. *ISPRS Journal of Photogrammetry and Remote Sensing*, 76:76–88.

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

[Eos Systems Inc., Photomodeler, 2017] Eos Systems Inc., Photomodeler (2017). `http://www.photomodeler.com/index.html`.

[Everingham et al., 2010] Everingham, M., van Gool, L., Williams, C., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338.

[Fabbri and Kimia, 2010] Fabbri, R. and Kimia, B. (2010). 3d curve sketch: Flexible curve-based stereo reconstruction and calibration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1538–1545. IEEE.

[Fleishman et al., 2005] Fleishman, S., Cohen-Or, D., and Silva, C. T. (2005). Robust moving least-squares fitting with sharp features. *ACM Transactions on Graphics*, 24(3).

[Freund et al., 1996] Freund, Y., Schapire, R. E., et al. (1996). Experiments with a new boosting algorithm. In *Icml*, volume 96, pages 148–156. Bari, Italy.

[Friedman et al., 1977] Friedman, J. H., Bentley, J. L., and Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226.

[Fukushima, 1980] Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202.

[Gehrung et al., 2017] Gehrung, J., Hebel, M., Arens, M., and Stilla, U. (2017). An approach to extract moving objects from mls data using a volumetric background representation. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 4:107.

[Geiger et al., 2012] Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE.

[Gibson, 1950] Gibson, J. J. (1950). The perception of the visual world.

[Gibson, 1966] Gibson, J. J. (1966). The senses considered as perceptual systems.

[Golovinskiy and Funkhouser, 2009] Golovinskiy, A. and Funkhouser, T. (2009). Min-cut based segmentation of point clouds. In *IEEE Workshop on Search in 3D and Video (S3DV) at International Conference on Computer Vision*.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[Graham, 2014] Graham, B. (2014). Spatially-sparse convolutional neural networks. *arXiv preprint arXiv:1409.6070*.

[Graham et al., 2017] Graham, B., Engelcke, M., and van der Maaten, L. (2017). 3d semantic segmentation with submanifold sparse convolutional networks. *CoRR*, abs/1711.10275.

[Graham and van der Maaten, 2017] Graham, B. and van der Maaten, L. (2017). Submanifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307*.

[Gray et al., 2017] Gray, S., Radford, A., and Kingma, D. P. (2017). Gpu kernels for block-sparse weights. Technical report, OpenAI.

[Greene, 1986] Greene, N. (1986). Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, 6(11):21–29.

[Guo et al., 2015] Guo, K., Zou, D., and Chen, X. (2015). 3d mesh labeling via deep convolutional neural networks. *ACM Transactions on Graphics*, 35(1):3.

[Guo et al., 2014] Guo, Y., Kumar, N., Narayanan, M., and Kimia, B. (2014). A multi-stage approach to curve extraction. In *European Conference on Computer Vision*, pages 663–678.

[Haala, 2013] Haala, N. (2013). The landscape of dense image matching algorithms. In *Photogrammetric Week 13*, pages 271–284.

[Haala et al., 1998] Haala, N., Brenner, C., and Anders, K.-H. (1998). 3d urban gis from laser altimeter and 2d map data. *International Archives of Photogrammetry and Remote Sensing*, 32:339–346.

[Hackel et al., 2017a] Hackel, T., Savinov, N., Ladicky, L., Wegner, J. D., Schindler, K., and Pollefeys, M. (2017a). 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.

[Hackel et al., 2018a] Hackel, T., Savinov, N., Ladicky, L., Wegner, J. D., Schindler, K., and Pollefeys, M. (2018a). Large-scale supervised learning for 3d point cloud labeling: Semantic3d.net. *accepted in PE&RS journal*.

[Hackel et al., 2018b] Hackel, T., Usvyatsov, M., Galliani, S., Wegner, J. D., and Schindler, K. (2018b). Inference, learning and attention mechanisms that exploit and preserve sparsity in convolutional networks. *arXiv preprint arXiv:1801.10585*.

[Hackel et al., 2016a] Hackel, T., Wegner, J. D., and Schindler, K. (2016a). Contour detection in unstructured 3d point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1610–1618.

[Hackel et al., 2016b] Hackel, T., Wegner, J. D., and Schindler, K. (2016b). Fast semantic segmentation of 3d point clouds with strongly varying density. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Prague, Czech Republic*, 3:177–184.

[Hackel et al., 2017b] Hackel, T., Wegner, J. D., and Schindler, K. (2017b). Joint classification and contour extraction of large 3d point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*, 130:231–245.

[Hammoudi et al., 2010] Hammoudi, K., Dornaika, F., Soheilian, B., and Paparoditis, N. (2010). Extracting wire-frame models of street facades from 3d point clouds and the corresponding cadastral map. *ISPRS Archives*, 38(3A).

[Han et al., 2015] Han, S., Pool, J., Tran, J., and Dally, W. (2015). Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143.

[Häne et al., 2017] Häne, C., Tulsiani, S., and Malik, J. (2017). Hierarchical surface prediction for 3d object reconstruction. *CoRR*, abs/1704.00710.

[Häne et al., 2013] Häne, C., Zach, C., Cohen, A., Angst, R., and Pollefeys, M. (2013). Joint 3d scene reconstruction and class segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 97–104.

[Harari, 2016] Harari, Y. N. (2016). *Homo Deus: A brief history of tomorrow*. Random House.

[Hastie et al., 2009] Hastie, T., Rosset, S., Zhu, J., and Zou, H. (2009). Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360.

[Hawkins and Ahmad, 2016] Hawkins, J. and Ahmad, S. (2016). Why neurons have thousands of synapses, a theory of sequence memory in neocortex. *Frontiers in neural circuits*, 10:23.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

[Henaff et al., 2015] Henaff, M., Bruna, J., and LeCun, Y. (2015). Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*.

[Hofer et al., 2015] Hofer, M., Maurer, M., and Bischof, H. (2015). Line3d: Efficient 3d scene abstraction for the built environment. In *German Conference on Pattern Recognition*, pages 237–248. Springer.

[Huang and You, 2016] Huang, J. and You, S. (2016). Point Cloud Labeling using 3D Convolutional Neural Network. In *International Conference on Pattern Recognition*, pages 2670–2675.

[Hubel and Wiesel, 1959] Hubel, D. H. and Wiesel, T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591.

[Hubel and Wiesel, 1962] Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154.

[Hug and Wehr, 1997] Hug, C. and Wehr, A. (1997). Detecting and identifying topographic objects in imaging laser altimeter data. *International archives of photogrammetry and remote sensing*, 32(3 SECT 4W2):19–26.

[Hussain and Triggs, 2012] Hussain, S. and Triggs, B. (2012). Visual recognition using local quantized patterns. In *European Conference on Computer Vision*.

[Isenburg, 2011] Isenburg, M. (2011). Laszip: lossless compression of lidar data. *European LiDAR Mapping Forum*.

[Jaderberg et al., 2014] Jaderberg, M., Vedaldi, A., and Zisserman, A. (2014). Speeding up convolutional neural networks with low rank expansions. *CoRR*, abs/1405.3866.

[Johnson and Hebert, 1999] Johnson, A. E. and Hebert, M. (1999). Using spin images for efficient object recognition in cluttered 3d scenes. *IEEE Transactions on pattern analysis and machine intelligence*, 21(5):433–449.

[Kappes et al., 2013] Kappes, J., Andres, B., Hamprecht, F., Schnorr, C., Nowozin, S., Batra, D., Kim, S., Kausler, B., Lellmann, J., Komodakis, N., et al. (2013). A comparative study of modern inference techniques for discrete energy minimization problems. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1328–1335.

[Karpathy et al., 2014] Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732. ACM.

[Kazhdan and Hoppe, 2013] Kazhdan, M. and Hoppe, H. (2013). Screened poisson surface reconstruction. *ACM Transactions on Graphics*, 32(3).

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Knudsen, 2007] Knudsen, E. I. (2007). Fundamental components of attention. *Annu. Rev. Neurosci.*, 30:57–78.

[Kohli et al., 2008] Kohli, P., Ladicky, L., and Torr, P. H. S. (2008). Robust higher order potentials for enforcing label consistency. In *Conference on Computer Vision and Pattern Recognition*.

[Kokkinos, 2016] Kokkinos, I. (2016). Ubernet: Training auniversal'convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. *arXiv preprint arXiv:1609.02132*.

[Konishi et al., 2003] Konishi, S., Yuille, A. L., Coughlan, J. M., and Zhu, S. C. (2003). Statistical edge detection: Learning and evaluating edge cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(1):57–74.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*.

[Ladicky et al., 2013] Ladicky, L., Russell, C., Kohli, P., and Torr, P. (2013). Associative hierarchical random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[Ladický et al., 2014] Ladický, L., Zeisl, B., and Pollefeys, M. (2014). Discriminatively trained dense surface normal estimation. In *European Conference on Computer Vision*, pages 468–484.

[Lafarge and Mallet, 2011] Lafarge, F. and Mallet, C. (2011). Building large urban environments from unstructured point data. In *IEEE International Conference on Computer Vision*.

[Lafarge and Mallet, 2012] Lafarge, F. and Mallet, C. (2012). Creating large-scale city models from 3d-point clouds: a robust approach with hybrid representation. *International journal of computer vision*, 99(1):69–85.

[Lai et al., 2014] Lai, K., Bo, L., and Fox, D. (2014). Unsupervised feature learning for 3d scene labeling. In *IEEE International Conference on Robotics and Automation*, pages 3050–3057. IEEE.

[Lalonde et al., 2006] Lalonde, J.-F., Vandapel, N., Huber, D., and Hebert, M. (2006). Natural terrain classification using three-dimensional ladar data for ground robot mobility. *Journal of Field Robotics*, 23(10):839–861.

[Landrieu and Simonovsky, 2017] Landrieu, L. and Simonovsky, M. (2017). Large-scale point cloud semantic segmentation with superpoint graphs. *arXiv preprint arXiv:1711.09869*.

[Lavie et al., 2004] Lavie, N., Hirst, A., De Fockert, J. W., and Viding, E. (2004). Load theory of selective attention and cognitive control. *Journal of Experimental Psychology: General*, 133(3):339.

[Lawin et al., 2017] Lawin, F. J., Danelljan, M., Tosteberg, P., Bhat, G., Khan, F. S., and Felsberg, M. (2017). Deep projective 3d semantic segmentation. In *International Conference on Computer Analysis of Images and Patterns*, pages 95–107. Springer.

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Li et al., 2016] Li, Y., Pirk, S., Su, H., Qi, C. R., and Guibas, L. J. (2016). Fpnn: Field probing neural networks for 3d data. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29, pages 307–315.

[Li et al., 2011] Li, Y., Wu, X., Chrysathou, Y., Sharf, A., Cohen-Or, D., and Mitra, N. J. (2011). Globfit: Consistently fitting primitives by discovering global relations. *ACM Transactions on Graphics*, 30(4):52:1–52:12.

[Lin et al., 2014] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer.

[Liu et al., 2015] Liu, B., Wang, M., Foroosh, H., Tappen, M., and Pensky, M. (2015). Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 806–814.

[Lodha et al., 2006] Lodha, S., Kreps, E., Helmbold, D., and Fitzpatrick, D. (2006). Aerial LiDAR Data Classification using Support Vector Machines (SVM). In *IEEE Third International Symposium on 3D Data Processing, Visualization, and Transmission*.

[Long et al., 2015] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440.

[Lowe, 1999] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157. Ieee.

[Lowe, 2004] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*.

[Maas, 1999] Maas, H.-G. (1999). The potential of height texture measures for the segmentation of airborne laserscanner data. In *Fourth international airborne remote sensing conference and exhibition/21st Canadian symposium on remote sensing*, volume 1, pages 154–161.

[Mackaness and Mackechnie, 1999] Mackaness, W. A. and Mackechnie, G. A. (1999). Automating the detection and simplification of junctions in road networks. *GeoInformatica*, 3(2):185–200.

[Malik et al., 2001] Malik, J., Belongie, S., Leung, T., and Shi, J. (2001). Contour and texture analysis for image segmentation. *International Journal of Computer Vision*.

[Manduchi et al., 2005] Manduchi, R., A, C., Talukder, A., and Matthies, L. (2005). Obstacle Detection and Terrain Classification for Autonomous Off-Road Navigation. *Autonomous Robots*, 18:81–102.

[Maninis et al., 2017] Maninis, K.-K., Pont-Tuset, J., Arbeláez, P., and Van Gool, L. (2017). Convolutional oriented boundaries: From image segmentation to high-level tasks. *IEEE transactions on pattern analysis and machine intelligence*.

[Marton et al., 2009] Marton, Z. C., Rusu, R. B., and Beetz, M. (2009). On fast surface reconstruction methods for large and noisy point clouds. In *IEEE International Conference on Robotics and Automation*, pages 3218–3223. IEEE.

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

[Monnier et al., 2012] Monnier, F., Vallet, B., and Soheilian, B. (2012). Trees detection from laser point clouds acquired in dense urban areas by a mobile mapping system. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 25:245–250.

[Montemerlo and Thrun, 2006] Montemerlo, M. and Thrun, S. (2006). Large-Scale Robotic 3-D Mapping of Urban Structures. In Ang, M. and Khatib, O., editors, *Experimental Robotics IX. Springer Tracts in Advanced Robotics*, volume 21. Springer.

[Montoya et al., 2014] Montoya, J., Wegner, J. D., Ladický, L., and Schindler, K. (2014). Mind the gap: modeling local and global context in (road) networks. In *German Conference on Pattern Recognition (GCPR)*.

[Montoya-Zegarra et al., 2014] Montoya-Zegarra, J. A., Wegner, J. D., Ladickỳ, L., and Schindler, K. (2014). Mind the gap: modeling local and global context in (road) networks. In *German Conference on Pattern Recognition*, pages 212–223. Springer.

[Muja and Lowe, 2009] Muja, M. and Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. *The International Conference on Computer Vision Theory and Applications*, 2(331-340):2.

[Munoz et al., 2009a] Munoz, D., Bagnell, J. A., Vandapel, N., and Hebert, M. (2009a). Contextual classification with functional max-margin markov networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 975–982. IEEE.

[Munoz et al., 2009b] Munoz, D., Vandapel, N., and Hebert, M. (2009b). Onboard Contextual Classification of 3-D Point Clouds with Learned High-order Markov Random Fields. In *IEEE International Conference on Robotics and Automation*.

[Najafi et al., 2014] Najafi, M., Namin, S. T., Salzmann, M., and Petersson, L. (2014). Non-associative higher-order markov networks for point cloud classification. In *European Conference on Computer Vision*, pages 500–515. Springer.

[Narang et al., 2017] Narang, S., Undersander, E., and Diamos, G. (2017). Block-sparse recurrent neural networks. *arXiv preprint arXiv:1711.02782*.

[Niemeyer et al., 2014] Niemeyer, J., Rottensteiner, F., and Soergel, U. (2014). Contextual classification of lidar data and building object detection in urban areas. *ISPRS Journal of Photogrammetry and Remote Sensing*, 87:152–165.

[Niemeyer et al., 2011] Niemeyer, J., Wegner, J. D., Mallet, C., Rottensteiner, F., and Soergel, U. (2011). Conditional random fields for urban scene classification with full waveform lidar data. In *Photogrammetric Image Analysis*, pages 233–244. Springer.

[Nissen and Bullemer, 1987] Nissen, M. J. and Bullemer, P. (1987). Attentional requirements of learning: Evidence from performance measures. *Cognitive psychology*, 19(1):1–32.

[Ok et al., 2012] Ok, A. O., Wegner, J. D., Heipke, C., Rottensteiner, F., Soergel, U., and Toprak, V. (2012). Matching of straight line segments from aerial stereo images of urban areas. *ISPRS Journal of Photogrammetry and Remote Sensing*, 74.

[Ondruska et al., 2016] Ondruska, P., Dequaire, J., Wang, D. Z., and Posner, I. (2016). End-to-end tracking and semantic segmentation using recurrent neural networks. *arXiv preprint arXiv:1604.05091*.

[Ørka et al., 2009] Ørka, H. O., Næsset, E., and Bollandsås, O. M. (2009). Classifying species of individual trees by intensity and structure features derived from airborne laser scanner data. *Remote Sensing of Environment*, 113(6):1163–1174.

[Öztireli et al., 2009] Öztireli, A. C., Guennebaud, G., and Gross, M. (2009). Feature preserving point set surfaces based on non-linear kernel regression. *Computer Graphics Forum*, 28(2).

[Parashar et al., 2017] Parashar, A., Rhu, M., Mukkara, A., Puglielli, A., Venkatesan, R., Khailany, B., Emer, J., Keckler, S. W., and Dally, W. J. (2017). Scnn: An

accelerator for compressed-sparse convolutional neural networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 27–40. ACM.

[Park et al., 2017] Park, J., Li, S., Wen, W., Tang, P. T. P., Li, H., Chen, Y., and Dubey, P. (2017). Faster cnns with direct sparse convolutions and guided pruning. In *Proceedings of the International Conference on Learning Representations*.

[Pauly et al., 2003] Pauly, M., Keiser, R., and Gross, M. (2003). Multi-scale feature extraction on point-sampled surfaces. In *Computer graphics forum*, volume 22, pages 281–289. Wiley Online Library.

[Posner and Petersen, 1990] Posner, M. I. and Petersen, S. E. (1990). The attention system of the human brain. *Annual review of neuroscience*, 13(1):25–42.

[Prokhorov, 2010] Prokhorov, D. (2010). A Convolutional Learning System for Object Classification in 3-D Lidar Data. *IEEE Transactions on Neural Networks*, 21(5):858–863.

[Pu et al., 2011] Pu, S., Rutzinger, M., Vosselman, G., and Elberink, S. O. (2011). Recognizing basic structures from mobile laser scanning data for road inventory studies. *ISPRS Journal of Photogrammetry and Remote Sensing*, 66(6):S28–S39.

[Pu and Vosselman, 2009] Pu, S. and Vosselman, G. (2009). Knowledge based reconstruction of building models from terrestrial laser scanning data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 64(6):575–584.

[Qi et al., 2016a] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2016a). Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593.

[Qi et al., 2017a] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017a). Pointnet: Deep learning on point sets for 3d classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[Qi et al., 2016b] Qi, C. R., Su, H., Nießner, M., Dai, A., Yan, M., and Guibas, L. J. (2016b). Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5648–5656.

[Qi et al., 2017b] Qi, X., Liao, R., Jia, J., Fidler, S., and Urtasun, R. (2017b). 3d graph neural networks for rgbd semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5199–5208.

[Ren et al., 2015] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*.

[Richter and Roth, 2015] Richter, S. R. and Roth, S. (2015). Discriminative shape from shading in uncalibrated illumination. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1128–1136.

[Riegler et al., 2017] Riegler, G., Ulusoy, A. O., and Geiger, A. (2017). Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE.

[Riemenschneider et al., 2014] Riemenschneider, H., Bódis-Szomorú, A., Weissenberg, J., and Van Gool, L. (2014). Learning where to classify in multi-view semantic segmentation. In *European Conference on Computer Vision*, pages 516–532. Springer.

[Robertson, 2007] Robertson, E. M. (2007). The serial reaction time task: implicit motor skill learning? *Journal of Neuroscience*, 27(38):10073–10075.

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer.

[Rosenblatt, 1957] Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory.

[Rottensteiner and Briese, 2002] Rottensteiner, F. and Briese, C. (2002). A new method for building extraction in urban areas from high-resolution lidar data. *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, 34(3/A):295–301.

[Rottensteiner et al., 2014] Rottensteiner, F., Sohn, G., Gerke, M., Wegner, J., Breitkopf, U., and Jung, J. (2014). Results of the ISPRS Benchmark on Urban Object Detection and 3D Building Reconstruction. *ISPRS Journal of Photogrammetry and Remote Sensing*, 93:256–271.

[Rottensteiner et al., 2013] Rottensteiner, F., Sohn, G., Gerke, M., and Wegner, J. D. (2013). ISPRS Test Project on Urban Classification and 3D Building Reconstruction. Technical report, ISPRS Working Group III / 4 - 3D Scene Analysis.

[Roynard et al., 2017] Roynard, X., Deschaud, J.-E., and Goulette, F. (2017). Paris-lille-3d: a large and high-quality ground truth urban point cloud dataset for automatic segmentation and classification. *arXiv preprint arXiv:1712.00032*.

[Rumelhart et al., 1985] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A., and Fei-Fei, L. (2015). Imagenet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252.

[Russell et al., 2003] Russell, S. J., Norvig, P., Canny, J. F., Malik, J. M., and Edwards, D. D. (2003). *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River.

[Rusu et al., 2009] Rusu, R. B., Marton, Z. C., Blodow, N., Holzbach, A., and Beetz, M. (2009). Model-based and learned semantic object labeling in 3d point cloud maps of kitchen environments. In *International Conference on Intelligent Robots and Systems*, pages 3601–3608. IEEE.

[Santoro et al., 2017] Santoro, A., Raposo, D., Barrett, D. G., Malinowski, M., Pascanu, R., Battaglia, P., and Lillicrap, T. (2017). A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pages 4974–4983.

[Schmid and Zisserman, 1997] Schmid, C. and Zisserman, A. (1997). Automatic line matching across views. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 666–671. IEEE.

[Schmidt et al., 2014] Schmidt, A., Niemeyer, J., Rottensteiner, F., and Soergel, U. (2014). Contextual classification of full waveform lidar data in the wadden sea. *IEEE Geoscience and Remote Sensing Letters*, 11(9):1614–1618.

[Schnabel et al., 2007] Schnabel, R., Wahl, R., and Klein, R. (2007). Efficient ransac for point-cloud shape detection. In *Computer graphics forum*, volume 26, pages 214–226. Wiley Online Library.

[Seitz et al., 2006] Seitz, S. M., Curless, B., Diebel, J., Scharstein, D., and Szeliski, R. (2006). A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 519–528. IEEE.

[Serna et al., 2014] Serna, A., Marcotegui, B., Goulette, F., and Deschaud, J.-E. (2014). Paris-rue-madame database: a 3d mobile laser scanner dataset for benchmarking urban detection, segmentation and classification methods. In *4th International Conference on Pattern Recognition, Applications and Methods ICPRAM 2014*.

[Shapovalov et al., 2010] Shapovalov, R., Velizhev, E., and Barinova, O. (2010). Nonassociative markov networks for 3d point cloud classification. the. In *Inter-*

national Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XXXVIII, Part 3A. Citeseer.

[Shechtman and Irani, 2007] Shechtman, E. and Irani, M. (2007). Matching local self-similarities across images and videos. In *Conference on Computer Vision and Pattern Recognition*.

[Shen et al., 2015] Shen, W., Wang, X., Wang, Y., Bai, X., and Zhang, Z. (2015). Deepcontour: A deep convolutional feature learned by positive-sharing loss for contour detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3982–3991.

[Shi and Cheung, 2006] Shi, W. and Cheung, C. (2006). Performance evaluation of line simplification algorithms for vector generalization. *The Cartographic Journal*.

[Shotton et al., 2006] Shotton, J., Winn, J., Rother, C., and Criminisi, A. (2006). *TextonBoost*: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *European Conference on Computer Vision*.

[Silberman et al., 2012] Silberman, N., Hoiem, D., Kohli, P., and Fergus, R. (2012). Indoor segmentation and support inference from rgbd images. In *European Conference on Computer Vision*, pages 746–760. Springer.

[Simonyan and Zisserman, 2014a] Simonyan, K. and Zisserman, A. (2014a). Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576.

[Simonyan and Zisserman, 2014b] Simonyan, K. and Zisserman, A. (2014b). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[Song et al., 2015] Song, S., Lichtenberg, S. P., and Xiao, J. (2015). Sun rgb-d: A rgb-d scene understanding benchmark suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 567–576.

[Song and Xiao, 2016] Song, S. and Xiao, J. (2016). Deep sliding shapes for amodal 3d object detection in rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 808–816.

[Steder et al., 2010] Steder, B., Rusu, R. B., Konolige, K., and Burgard, W. (2010). Narf: 3d range image features for object recognition. In *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the International Conference on Intelligent Robots and Systems*, volume 44.

[Steder et al., 2011] Steder, B., Rusu, R. B., Konolige, K., and Burgard, W. (2011). Point feature extraction on 3D range scans taking into account object boundaries. In *IEEE International Conference on Robotics and Automation*.

[Tatarchenko et al., 2017] Tatarchenko, M., Dosovitskiy, A., and Brox, T. (2017). Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. *CoRR*, abs/1703.09438.

[Taylor et al., 1996] Taylor, C. J., Debevec, P. E., and Malik, J. (1996). Reconstructing polyhedral models of architectural scenes from photographs. In *European Conference on Computer Vision*, pages 659–668. Springer.

[Tchapmi et al., 2017] Tchapmi, L. P., Choy, C. B., Armeni, I., Gwak, J., and Savarese, S. (2017). Segcloud: Semantic segmentation of 3d point clouds. *arXiv preprint arXiv:1710.07563*.

[Tombari et al., 2010] Tombari, F., Salti, S., and Di Stefano, L. (2010). Unique signatures of histograms for local surface description. In *European Conference on Computer Vision*, pages 356–369. Springer.

[Torralba et al., 2008] Torralba, A., Fergus, R., and Freeman, W. T. (2008). 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970.

[Torralba et al., 2004] Torralba, A., Murphy, K., and Freeman, W. (2004). Sharing features: efficient boosting procedures for multiclass object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[Trimble, Inpho Geo-Modeling Module, 2016] Trimble, Inpho Geo-Modeling Module (2016). `http://www.trimble.com/imaging/inpho.aspx?tab=Geo-Modeling_Module`.

[Tupin et al., 1998] Tupin, F., Maitre, H., Mangin, J.-F., Nicolas, J.-M., and Pechersky, E. (1998). Detection of linear features in sar images: Application to road network extraction. *IEEE transactions on geoscience and remote sensing*, 36(2):434–453.

[Türetken et al., 2012] Türetken, E., Benmansour, F., and Fua, P. (2012). Automated reconstruction of tree structures using path classifiers and mixed integer programming. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 566–573. IEEE.

[Türetken et al., 2011] Türetken, E., González, G., Blum, C., and Fua, P. (2011). Automated reconstruction of dendritic and axonal trees by global optimization with geometric priors. *Neuroinformatics*, 9(2-3):279–302.

[Uhrig et al., 2017] Uhrig, J., Schneider, N., Schneider, L., Franke, U., Brox, T., and Geiger, A. (2017). Sparsity invariant cnns. *arXiv preprint arXiv:1708.06500*.

[Vallet et al., 2015] Vallet, B., Brédif, M., Serna, A., Marcotegui, B., and Paparoditis, N. (2015). Terramobilita/iqmulus urban point cloud analysis benchmark. *Computers & Graphics*, 49:126–133.

[Vandapel et al., 2004] Vandapel, N., Huber, D., Kapuria, A., and Hebert, M. (2004). Natural Terrain Classification using 3-D Ladar Data. In *IEEE International Conference on Robotics and Automation*.

[Vetrivel et al., 2017] Vetrivel, A., Gerke, M., Kerle, N., Nex, F., and Vosselman, G. (2017). Disaster damage detection through synergistic use of deep learning and 3d point cloud features derived from very high resolution oblique aerial images, and multiple-kernel-learning. *ISPRS journal of photogrammetry and remote sensing*.

[Weber et al., 2010] Weber, C., Hahmann, S., and Hagen, H. (2010). Sharp feature detection in point clouds. In *Shape Modeling International Conference (SMI), 2010*, pages 175–186. IEEE.

[Wegner et al., 2015] Wegner, J. D., Montoya-Zegarra, J. A., and Schindler, K. (2015). Road networks as collections of minimum cost paths. *ISPRS Journal of Photogrammetry and Remote Sensing*, 108:128–137.

[Weinmann et al., 2015a] Weinmann, M., Jutzi, B., Hinz, S., and Mallet, C. (2015a). Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers. *ISPRS Journal of Photogrammetry and Remote Sensing*, 105:286–304.

[Weinmann et al., 2013] Weinmann, M., Jutzi, B., and Mallet, C. (2013). Feature relevance assessment for the semantic interpretation of 3d point cloud data. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, II-5/W2.

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

[Weinmann et al., 2017] Weinmann, M., Weinmann, M., Mallet, C., and Brédif, M. (2017). A classification-segmentation framework for the detection of individual trees in dense mms point cloud data acquired in urban areas. *Remote sensing*, 9(3):277.

[Wen et al., 2016] Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. (2016). Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082.

[Wu et al., 2015] Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. (2015). 3d shapenets: A deep representation for volumetric shapes. In *Proceed-*

ings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1912–1920.

[Xia and Wang, 2017] Xia, S. and Wang, R. (2017). A fast edge extraction method for mobile lidar point clouds. *IEEE Geoscience and Remote Sensing Letters*, 14(8):1288–1292.

[Xiao and Furukawa, 2014] Xiao, J. and Furukawa, Y. (2014). Reconstructing the worlds museums. *International journal of computer vision*, 110(3):243–258.

[Xie and Tu, 2015] Xie, S. and Tu, Z. (2015). Holistically-nested edge detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1395–1403.

[Yan et al., 2015] Yan, W., Shaker, A., and El-Asjmawy, N. (2015). Urban land cover classification using airborne LiDAR data: A review. *Remote Sensing of Environment*, pages 295–310.

[Yao et al., 2011] Yao, W., Hinz, S., and Stilla, U. (2011). Extraction and motion estimation of vehicles in single-pass airborne lidar data towards urban traffic analysis. *ISPRS Journal of Photogrammetry and Remote Sensing*, 66(3):260–271.

[Zbontar and LeCun, 2016] Zbontar, J. and LeCun, Y. (2016). Stereo matching by training a convolutional neural network to compare image patches. *Journal of Machine Learning Research*, 17(1-32):2.

[Zeiler, 2012] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

[Zhuang et al., 2014] Zhuang, Y., He, G., Hu, H., and Wu, Z. (2014). A novel outdoor scene-understanding framework for unmanned ground vehicles with 3d laser scanners. *Transactions of the Institute of Measurement and Control*, page 0142331214541140.

[Zhuang et al., 2015] Zhuang, Y., Liu, Y., He, G., and Wang, W. (2015). Contextual classification of 3d laser points with conditional random fields in urban environments. In *International Conference on Intelligent Robots and Systems*, pages 3908–3913. IEEE.

[Zwicker et al., 2001] Zwicker, M., Pfister, H., Van Baar, J., and Gross, M. (2001). Surface splatting. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 371–378. ACM.